UNIVERSITÉ DE MONTRÉAL

EXTENSIONS À L'ALGORITHME DE RECHERCHE DIRECTE MADS POUR L'OPTIMISATION NON LISSE

SÉBASTIEN LE DIGABEL

DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION

DU DIPLÔME DE PHILOSOPHIÆ DOCTOR (Ph.D.)

(MATHÉMATIQUES DE L'INGÉNIEUR)

JUIN 2008

Your file Votre référence
ISBN: 978-0-494-46109-9
Our file Notre référence
ISBN: 978-0-494-46109-9

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

EXTENSIONS À L'ALGORITHME DE RECHERCHE DIRECTE MADS POUR L'OPTIMISATION NON LISSE

présentée par : LE DIGABEL Sébastien

en vue de l'obtention du diplôme de : Philosophiæ Doctor

a été dûment acceptée par le jury d'examen constitué de :

M. ORBAN Dominique, Doct. Sc., président

M. AUDET Charles, Ph.D., membre et directeur de recherche

M. PERRON Sylvain, Ph.D., membre

M. CONN Andrew Roger, Ph.D., membre externe

REMERCIEMENTS

Un grand merci à mon directeur Charles Audet, tout d'abord pour m'avoir encouragé à faire ce doctorat, puis ensuite et surtout pour sa disponibilité en tout temps et son aide précieuse. Ce fut réellement un plaisir d'évoluer sous son aile.

Merci ensuite à John Dennis, Mark Abramson et Vincent Béchard, mes coauteurs, Gilles Couture et Walid Zghal pour toute sorte d'aide et de commentaires, Christophe Tribes et Jean-Yves Trépanier et le CRIAQ qui a financé une grande partie de ma thèse.

Des remerciements aussi pour Francine Benoît pour le support LaTeX et ses blagues, Pierre Girard et Luc Rocheleau pour leurs dépannages et leurs réponses à tout.

Merci aussi à tous mes amis et en particulier Jean et Viviane pour la composante sorties, bières et discussions du doc, sans oublier Oka mon compagnon canin de cette dernière année.

Je voudrais enfin remercier mes parents Joël et Jacqueline et mon frère Yoann, pour leur soutien, leurs appels réguliers et leurs nombreux colis, et adresser mon dernier merci spécial à ma fiancée Karine pour ses encouragements, son attitude toujours positive, et sa faculté elle aussi à me faire évoluer.

RÉSUMÉ

L'optimisation non lisse s'applique aux problèmes ne possédant pas la structure dérivative usuelle requise par les méthodes d'optimisation classiques. Ces problèmes se rencontrent dans divers domaines, notamment l'ingénierie. En effet, la plupart des situations réelles se modélisent ou se simulent par des fonctions compliquées, typiquement des codes informatiques, dont on ne peut exploiter la structure.

Une récente famille d'algorithmes développée en 2006, la *recherche directe sur treillis adaptifs* (MADS) [20], est spécialement conçue pour ces problèmes, et assure des propriétés de convergence basées sur le calcul de Clarke des fonctions non lisses.

Cette thèse propose des améliorations à MADS, sous la forme de trois extensions correspondant à autant d'articles acceptés ou soumis pour publication : la première [12] permet de coupler MADS à la méta-heuristique de *recherche à voisinage variable* (VNS) [68, 100] utilisée habituellement en optimisation combinatoire. L'aspect complémentaire des deux méthodes accroît la stabilité des résultats.

La deuxième extension [22] décrit PSD-MADS, une parallélisation de MADS asynchrone permettant de résoudre des problèmes de plus grande taille, pour la première fois de l'ordre de plusieurs centaines de variables.

La dernière extension [6] donne une alternative déterministe, ORTHOMADS, à l'implémentation originale de MADS donnée en [20], LTMADS, qui comportait une composante aléatoire. Elle assure aussi une répartition également distribuée des directions de recherche à chaque itération de MADS.

Chacune de ces extensions est soutenue par une analyse de convergence rigoureuse basée sur le calcul non lisse de Clarke [35]. Elle est de plus testée numériquement sur des ensembles de problèmes tests comprenant des problèmes analytiques de la littérature ainsi que des problèmes réels tirés de diverses application du génie. Les résultats obtenus

permettent d'avancer que les extensions de cette thèse constituent bien des améliorations de MADS.

ABSTRACT

Nonsmooth optimization applies to problems that do not possess the derivative structure usually required by conventional optimization methods. These problems occur in a wide variety of fields, including engineering, where real situations are modelled or simulated using complicated functions, usually computer codes without exploitable structure.

A recent class of algorithms developed in 2006, the Mesh Adaptive Direct Search, or MADS [20], is specially designed for these problems, with a hierarchical convergence analysis based on the Clarke calculus for nonsmooth functions.

This thesis suggests improvements to MADS, through three extensions corresponding to three papers accepted or submitted for publication: The first extension [12] describes the introduction of the Variable Neighborhood Search (VNS) metaheuristic [68, 100], commonly used in combinatorial optimization, into MADS. The complementarity of the two methods increases the stability of the results.

The second [22] describes PSD-MADS, an asynchronous parallelization of MADS and targets problems with a large number of variables, for the first time in the order of several hundred variables.

The third extension [6] provides a deterministic new implementation of MADS, ORTHOMADS, the previous and original one (LTMADS) being defined with a random component. It also provides a better distribution of search directions in the space of variables at every iteration of MADS.

Each of these extensions is backed by a rigorous convergence analysis based on the Clarke calculus for nonsmooth functions and is tested on sets of problems including analytic problems from the literature as well as real problems originating from various applications of engineering . The results obtained support the conclusion that the proposed extensions are improvements of MADS.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES ABBRÉVATIONS ET SYMBOLES

APPS ..... Asynchronous Parallel Pattern Search

CUTEr ..... Constrained and Unconstrained Testing Environment, revisited

DE ..... Boeing Design Explorer

DFO ..... Derivative-Free Optimization

DIRECT ..... DIviding RECTangles

GADS ..... Genetic Algorithm and Direct Search

GPS ..... Generalized Pattern Search

GSS ..... Generating Set Search

KKT ..... Karush-Kuhn-Tucker

LH ..... Hypercubes Latins

LTMADS ..... Lower Triangular MADS

MADS ..... Mesh Adaptive Direct Search

MDO ..... Multidisciplinary Design Optimization

MPI ..... Message Passing Interface

NOMAD ..... Nonlinear Optimization for Mixed vAriables and Derivatives

ORTHOMADS ..... MADS déterministe avec directions orthogonales

pGPS ..... GPS parallèle

pMADS ..... MADS parallèle

PSD ..... Parallel Space Decomposition

PVD ..... Parallel Variable Distribution

VNS ..... Variable Neighborhood Search

$\mathbb{N}$ ..... Ensemble des entiers naturels positifs

$\mathbb{Q}$ ..... Ensemble des rationnels

$\mathbb{R}$ ..... Ensemble des réels

$\mathbb{R}^+$ ou $\mathbb{R}_+$ ..... Ensemble des réels positifs

$\mathbb{Z}$ ..... Ensemble des entiers relatifs

$\|x\|$ ..... Norme euclidienne de $x \in \mathbb{R}^n$

$\|x\|_\infty$ ..... Norme infinie de x

I_n ou I ..... Matrice identité en dimension n

e_i ..... $i$ème colonne de I_n

$B_\varepsilon(x)$ ..... Boule ouverte de rayon ε centrée en x

INTRODUCTION

Diverses disciplines, dont l'ingénierie, consistent à concevoir ou étudier des systèmes complexes (une aile d'avion, un procédé de fabrication, etc.). Pour mener ces études de la façon la plus efficace possible, on va construire un modèle mathématique, qui décrit le système, en identifiant des variables et des fonctions de ces variables qui mesurent la réalisabilité et l'efficacité du système. Une fois que cette étape de modélisation est achevée, et que le système est décrit mathématiquement, on obtient un problème d'optimisation : on va chercher les valeurs des variables (les solutions) qui font en sorte que l'efficacité (l'objectif) du système est la plus grande, tout en respectant la réalisabilité du système (les contraintes).

Les techniques usuelles d'optimisation supposent que les problèmes ont une certaine structure *lisse*, c'est-à-dire que les fonctions décrivant le problème possèdent des gradients indiquant leurs variations. Le gradient est exploité par ces méthodes, qui l'utilisent pour diriger une recherche vers les bonnes solutions.

La plupart des problèmes qui simulent des situations réelles n'ont pas de structure lisse. Ceci peut provenir par exemple du fait que certains comportements sont décrits par des systèmes d'équations différentielles dont la solution ne peut être déterminée que numériquement. Les fonctions qui décrivent ces problèmes non lisses peuvent ne pas être continues ni différentiables, ou même être non définies en certains points, constituant ainsi des contraintes cachées. Elles peuvent en outre contenir du bruit rendant difficile toute approximation de gradients, dont l'existence même peut être compromise. Une bonne façon de voir de telles fonctions est de les considérer comme des boîtes noires, des oracles auxquels on donnerait un point et qui éventuellement retourneraient des valeurs. En pratique, ces fonctions sont des simulations informatiques, ou peuvent même correspondre à des expériences manuelles. Ce dernier élément illustre le fait que l'évaluation de ces fonctions peut être très coûteuse en terme de temps. Le terme de

fonction est en lui-même abusif car pour certains problèmes, deux évaluations au même point donnent parfois des résultats différents (nous continuerons toutefois d'utiliser le terme *fonction*).

Pour donner une idée du large éventail des problèmes non lisses, on donne pêle-mêle les exemples suivants : la conception d'un bouclier thermique [3, 79], la géométrie de molécules [8], l'optimisation de paramètres algorithmiques [23], la conception d'un rotor d'hélicoptère [27, 28], le placement de puits [58], la combustion catalytique [71], le traitement des brasques dans la production d'aluminium [11], et la conception aéroacoustique [96]. Une grande famille de problèmes non lisses souvent évoquée concerne les problèmes d'optimisation multidisciplinaires (MDO [114]) qui décrivent des systèmes complexes composés de plusieurs disciplines possédant des interactions complexes.

Pour ces problèmes non lisses, les techniques usuelles ne sont pas adaptées. En effet, elles pourront rarement identifier des solutions optimales correspondant à des points où les dérivées ne sont pas définies. Nous nous intéressons ici à l'approche consistant à définir des méthodes spécialisées pour l'optimisation non lisse.

Tout algorithme ambitionnant de résoudre un problème non lisse ne peut qu'interroger les oracles, sans pouvoir exploiter aucune de leurs caractéristiques. Il doit en plus être robuste et capable de fournir une bonne solution après un nombre raisonnable d'évaluations. Afin de garantir son efficacité pratique, une condition nécessaire que doit remplir un tel algorithme est d'être muni d'une analyse de convergence, basée sur une hiérarchie de différentiabilité fournissant des garanties théoriques, même sur les problèmes lisses et faciles à résoudre.

La récente famille d'algorithmes de recherche directe sur treillis adaptifs, MADS [20], dont l'analyse de convergence repose sur le calcul de Clarke [35], est une méthode conçue pour répondre à ces exigences.

L'objectif de cette thèse rédigée par articles est de décrire certaines extensions amé-

liorant l'efficacité de MADS, tout en en conservant les propriétés de convergence. Ces extensions sont au nombre de trois, chacune correspondant à un article : le premier article propose de coupler MADS à la méta-heuristique VNS (MADS-VNS). Le deuxième décrit la parallélisation PSD-MADS de MADS permettant de traiter des problèmes de grande taille, et le dernier donne une nouvelle implémentation déterministe de l'algorithme, ORTHOMADS.

Dans un premier chapitre, nous présenterons une revue de la littérature sur les algorithmes de recherche directe dont fait partie MADS, qui sera lui-même détaillé. Ensuite, au chapitre 2, nous introduirons nos articles, en exposant notre démarche de recherche, l'organisation du travail, et les résultats escomptés. Les trois articles seront ensuite donnés à raison de un par chapitre, suivis d'une discussion générale et d'une conclusion.

CHAPITRE 1

REVUE DE LA LITTÉRATURE SUR LES MÉTHODES DE RECHERCHE DIRECTE POUR L'OPTIMISATION NON LISSE

Ce chapitre décrit les principales méthodes de recherche directe pour l'optimisation du problème

$$\min_{x \in \Omega} f(x) \qquad (\mathcal{P})$$

avec $f : X \to \mathbb{R} \cup \{\infty\}$ la fonction objectif à minimiser, $\Omega = \{x \in X : g(x) \leq 0\}$ l'ensemble réalisable, $g : X \to (\mathbb{R} \cup \{\infty\})^m$, et X un sous-ensemble de $\mathbb{R}^n$. Au delà de ces définitions, nous ne formulons pas d'autres hypothèses sur les fonctions f et g (on rappelle que le terme de fonction est un abus de langage dans le sens où un point x peut engendrer des valeurs de f ou g différentes). Les fonctions peuvent être telles que décrites dans l'introduction, bruitées, non continues, non différentiables, ou non définies en certains points de X. Pour de tels points, qui définissent des contraintes *cachées*, on considère que les fonctions valent l'infini. On doit en plus considérer que l'évaluation de f et g est coûteuse en terme de temps. L'ensemble réalisable Ω est défini à l'aide de l'ensemble X dans le but de distinguer deux types de contraintes : les contraintes de X sont appelées *fermées* et inviolables. Les autres contraintes de l'ensemble Ω sont qualifiées *d'ouvertes* dans le sens où f et g sont à priori définies sur $X \setminus \Omega$.

Les méthodes que nous évoquons sont uniquement celles pour lesquelles un cadre théorique et une analyse de convergence existent, prenant en compte la nature non lisse du problème (et donc, on ne considère pas les méthodes dont la première hypothèse sur f est qu'elle est différentiable).

Dans un premier temps, nous évoquerons la littérature générale des méthodes de recherche directe, puis nous détaillerons trois algorithmes en particulier : la recherche par

coordonnées, l'algorithme généralisé de recherche par motifs GPS, et MADS. Ce choix s'explique par le fait que MADS, qui constitue le sujet de cette thèse, est l'évolution et la généralisation des deux premières méthodes, et se décrit très clairement en exposant cette évolution.

1.1 Les méthodes de recherche directe

Nous entendons par méthodes de recherche directe des méthodes qui cherchent à minimiser $\mathcal{P}$ uniquement en évaluant f et implicitement g. Des revues détaillant certaines de ces méthodes sont données dans [37, 78, 81, 89].

Une première famille de méthodes que l'on identifie est composée de méthodes tentant de construire un modèle de la fonction à optimiser. Ce modèle est souvent un polynôme interpolant f, construit à partir d'un échantillonage de points où f a déjà été évaluée. Le modèle est ensuite lui même optimisé (par exemple avec un algorithme de régions de confiance). Ces méthodes, les *méthodes sans dérivées*, ou DFO *sampling methods*, sont décrites dans [36–39, 104, 105]. Dans cette catégorie, on trouve également des méthodes qui vont tenter d'approcher les gradients de f, et d'utiliser cette information afin de déterminer quels sont les points prometteurs où évaluer f (voir [31, 42–44, 78]).

Les autres méthodes de recherche directe ne tentent pas d'approcher de gradients ni de construire un modèle de f, bien qu'il soit possible de le faire avec MADS, comme dans [43] où des approximations de gradients sont utilisées pour ordonner les directions de recherche, ou encore à la section 1.6 sur les fonctions substitut. Parmi ces méthodes se trouvent l'algorithme de Nelder-Mead [101, 107], l'algorithme de Hooke et Jeeves [73], l'algorithme DIRECT [54, 62, 77], ainsi que la recherche par coordonnées [45], GPS [18, 29, 120], et MADS [20]. Une version parallèle de GPS, APPS, est décrite dans [64, 74, 80, 83, 84].

L'analyse de convergence de ces dernières méthodes repose sur l'hypothèse que les

points d'évaluation se situent sur une structure discrète de l'espace (le treillis) dont on contrôle la finesse. Une autre solution consiste à utiliser des conditions de décroissance suffisante, comme pour les algorithmes GSS (*Generating Set Search*) de [81, 82] ou les méthodes de cadres de [40, 106].

Concernant des exemples de problèmes du type de $\mathcal{P}$ résolus par des méthodes de recherche directe, on peut mentionner les exemples de [3, 8, 23, 27, 58, 71, 76, 79, 95–98].

Il n'y a pas d'article fournissant de comparaison exhaustive des différentes méthodes. Citons toutefois [59] qui en compare quelques-unes sur un problème particulier.

Nous pouvons maintenant exposer les méthodes de recherche par coordonnées et GPS, qui conduisent à MADS.

1.2 L'algorithme de recherche par coordonnées

Nous considérons ici le cas sans contraintes ($\Omega = X = \mathbb{R}^n$), et le problème $\mathcal{P}$ s'écrit $\min_{x \in \mathbb{R}^n} f(x)$. L'algorithme très simple de recherche par coordonnées (*Coordinate Search* ou *Compass Search*, en préface de [45]) est un cas particulier de l'algorithme GPS étudié à la prochaine section, lui-même étant un cas particulier des algorithmes MADS. Il n'est pas souvent efficace sur des problèmes réels, mais garantit tout de même des résultats de convergence du premier ordre lorsque f est strictement différentiable. L'analyse de convergence conduisant à ces résultats ne sera pas abordée ici car elle est un cas particulier de l'analyse effectuée pour MADS en 1.4.2.

1.2.1 Description de l'algorithme de recherche par coordonnées

L'algorithme est présenté à la figure 1.1. C'est un algorithme itératif, qui, à chaque itération k, va évaluer f autour de l'itéré courant x_k à une distance de Δ_k suivant les $2n$ directions de base de $\mathbb{R}^n$, $\{\pm e_i\}_{i=1}^n$, avec e_i la $i$ème colonne de la matrice identité I_n. Les

itérés se situent sur une discrétisation de l'espace appelée le treillis, et Δ_k sera appelé plus tard le *paramètre de treillis*, qui va tendre à être infiniment petit plus l'algorithme progresse. Cette recherche, uniquement locale, sur un espace discrétisé, est appelé une *sonde locale*, ou POLL. L'ensemble des points de sonde est défini par

$$P_k = \{x_k \pm \Delta_k e_i \ : \ i = 1, 2, \ldots, n\}$$

et appelé *cadre*. Cette terminologie est empruntée à Coope et Price [40] qui définissent des cadres plus généraux. L'itéré courant est également désigné *centre de sonde*. Si la sonde locale permet de déterminer un point $x \in P_k$ meilleur que x_k, alors l'itération k est qualifiée de *succès* et x sera le prochain itéré. Dans le cas contraire (*échec*), x_k va rester le centre de sonde pour l'itération suivante, mais Δ_k sera divisé par deux, ce qui permettra d'explorer l'espace plus près de x_k. Plusieurs conditions d'arrêt sont possibles, les deux plus courantes étant une taille de treillis minimale et un nombre maximum d'évaluations de f (notion de *budget d'évaluations*).

[0] Initialisations
- $x_0 \in \mathbb{R}^n$, $\Delta_0 \in \mathbb{R}$, $k \leftarrow 0$

[1] Itération k (SONDE LOCALE)
- $P_k \leftarrow \{x_k \pm \Delta_k e_i : i = 1, \ldots, n\}$
- évaluer f sur les points de P_k

[2] Mises à jour
- **si** il existe $x \in P_k$ tel que $f(x) < f(x_k)$ (succès)
  - $x_{k+1} \leftarrow x$
  - $\Delta_{k+1} \leftarrow \Delta_k$
- **sinon** (échec)
  - $x_{k+1} \leftarrow x_k$
  - $\Delta_{k+1} \leftarrow \Delta_k/2$
- $k \leftarrow k + 1$
- **aller en [1] si** aucune condition d'arrêt vérifiée

Figure 1.1 – L'algorithme de recherche par coordonnées.

1.2.2 Améliorations possibles

Plusieurs améliorations simples sont possibles, préfigurant la grande flexibilité des algorithmes présentés dans ce document.

Tout d'abord, l'utilisation d'une cache est souvent avantageuse pour mémoriser tous les points pour lesquels on a évalué f, pour éviter d'avoir à effectuer plusieurs fois la même évaluation (qui peut être coûteuse). Cette stratégie est rentable car pour les problèmes considérés, on n'aura jamais un grand nombre de points dans la cache. Cette cache peut être réutilisée d'un lancement de l'algorithme à l'autre.

Ensuite, à l'itération k, on peut ne pas évaluer f pour tous les points de P_k, mais cesser les évaluations de l'itération dès qu'un meilleur point x a été trouvé. C'est ce qu'on appelle la recherche incomplète, ou la stratégie opportuniste.

A priori, l'examen des points de P_k se fait selon l'ordre des directions selon lesquelles ces points ont été générés. Cet ordre peut être modifié de manière dynamique : si à l'itération k le point x est tel que $f(x) < f(x_k)$ et qu'on a $x_{k+1} \leftarrow x$, alors la direction utilisée pour générer x est placée en tête de la liste des directions pour la prochaine itération. On peut même faire mieux en ordonnant toutes les directions selon les valeurs croissantes de f.

1.2.3 Exemple pathologique de la recherche par coordonnées

Il est facile de présenter un problème simple pour lequel la recherche par coordonnées ne fournit pas un bon résultat : l'exemple de la figure 1.2, repris de [2], consiste à minimiser la fonction $f : x \rightarrow \|x\|_\infty$ avec $x \in \mathbb{R}^2$. La courbe de niveau $f = 1$ est représentée en gras sur la figure.

L'optimalité est atteinte au point $x^* = [0\ 0]^T$ avec $f(x^*) = 0$, mais si on lance l'algorithme avec un point initial x_0 tel que $x_0 \neq x^*$ et $|x_0^T e_1| = |x_0^T e_2|$, alors toutes

les itérations seront des échecs. En effet, pour tout $\Delta > 0$ et pour tout $j \in \{1, 2\}$, on a $f(x_0 \pm \Delta e_j) \geq f(x_0)$. Le problème est que f n'étant pas différentiable, on n'a pas de garantie que l'algorithme converge vers un point stationnaire. Ici, on converge en un point (x_0) où le gradient n'est pas défini, mais où il existe des directions de descente stricte.

Ceci a donc mis en lumière deux inconvénients de la recherche par coordonnées : tout d'abord, le nombre des directions possibles pour la sonde est limité, et ensuite, on n'utilise qu'une recherche locale, et pas de stratégie de recherche au niveau global (sur notre exemple, il aurait été facile de trouver des points générés au hasard meilleurs que $x_0 = [1\ 1]^T$).

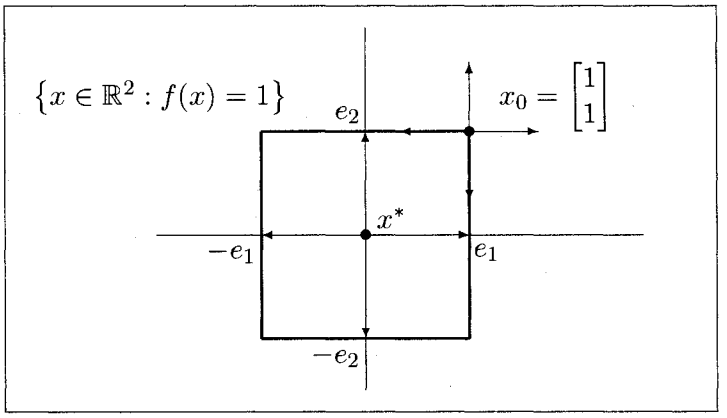

Figure 1.2 – Recherche par coordonnées avec $\|x\|_\infty$.

1.3 L'algorithme GPS

L'algorithme *Generalized Pattern Search*, ou *recherche généralisée par motifs*, a été introduit par Torczon [120], puis reformulé par Booker *et al.* [29] dans un format proche de l'algorithme GPS dont il est question ici. L'algorithme généralise la recherche par coordonnées et la méthode de Hooke et Jeeves [73]. Le concept de bases positives provenant de [46] fut intégré à l'algorithme dans [85] (une base positive est un ensemble

minimal de vecteurs dont des combinaisons linéaires positives peuvent engendrer tout l'espace). GPS constitue une évolution de l'algorithme de recherche par coordonnées auquel il ajoute un plus grand nombre de directions de recherche, une plus grande flexibilité dans leur choix, ainsi qu'une recherche à un niveau global, la *recherche globale*, ou SEARCH. Pour un historique détaillé des méthodes ayant mené aux algorithmes de type GPS, voir [81]. Dans un premier temps, nous présenterons l'algorithme en lui-même, puis, comme pour l'algorithme de recherche par coordonnées, nous donnerons un exemple pathologique pour lequel GPS ne donne pas de bons résultats. On revient ici dans le cadre du problème contraint $\mathcal{P}$, et la gestion des contraintes sera abordée en 1.5.

1.3.1 Description de l'algorithme GPS

L'algorithme est présenté à la figure 1.3. Deux types de recherches ont lieu à chaque itération k : la recherche globale, et la sonde locale. GPS apporte plus de liberté que la recherche par coordonnées dans le choix des directions de sonde. L'ensemble des directions possibles, D, obéit à certaines règles, dont la nécessité a été justifiée par Audet [10] : D doit être de la forme $D = GZ$ avec $G \in \mathbb{R}^{n\times n}$ non singulière et $Z \in \mathbb{Z}^{n\times p}$, tandis que l'ensemble des directions à l'itération k, $D_k = \{d_k^1, d_k^2, \ldots, d_k^{p_k}\} \subseteq D$ doit être un ensemble générateur positif (on a $n+1 \leq p_k \leq p = |D|$).

Une itération k est qualifiée de *succès* si on a trouvé un point x tel que $f(x) < f(x_k)$, sinon l'itération est un *échec*. A l'itération k, on définit le treillis M_k comme étant une discrétisation de $\mathbb{R}^n$ sur laquelle tous les itérés doivent se trouver. Plus précisément, le treillis est l'ensemble ainsi défini :

$$M_k = \{x + \Delta_k Dz \ : \ x \in V_k, z \in \mathbb{N}^p\} \ ,$$

où V_k est l'ensemble des points où la fonction f a déjà été évaluée. L'ensemble des

points de la recherche locale (le cadre) s'écrit

$$P_k = \{x_k + \Delta_k d \ : \ d \in D_k\} \subseteq M_k \ .$$

La condition sur le paramètre de treillis Δ_k est la suivante :

$$\Delta_{k+1} = \tau^{\omega_k} \Delta_k$$

avec $\tau \in \mathbb{Q}$ et ω_k un entier fini, positif ou nul si l'itération k est un succès et strictement négatif sinon (ceci est assuré par les choix d'un entier positif ω^+ et d'un entier négatif ω^-, voir figure 1.3).

Moins de règles régissent la recherche globale, qui doit uniquement générer un nombre fini de points du treillis. La stratégie de recherche globale est laissée à la discrétion de l'utilisateur, ce qui apporte une grande flexibilité à l'algorithme : en effet, il peut alors utiliser sa connaissance du problème pour tâcher de trouver des points prometteurs sur le treillis courant M_k. D'autres stratégies prédéterminées sont également possibles, comme une recherche aléatoire ou une recherche basée sur les hypercubes latins (LH) [117,118]. La recherche globale permettra d'autres choses par la suite, par exemple l'emploi de fonctions substitut (section 1.6), ou le couplage avec l'algorithme VNS (chapitre 3).

L'analyse de convergence est un cas particulier de celle effectuée pour l'algorithme MADS (résumée en 1.4.2), en se restreignant aux directions de D, mais elle peut être effectuée uniquement dans le contexte GPS comme dans [18].

On peut remarquer que la recherche par coordonnées est bien un cas particulier de GPS avec $D = [I_n \ -I_n]$, $\tau = 2$, $\omega^- = -1$ et $\omega^+ = 0$.

Les stratégies vues à la section 1.2.2 sont encore applicables ici. La méthode parallèle APPS [64, 74, 80, 83, 84] constitue également une amélioration de GPS. Abramson *et al.* [5] ont décrit des améliorations supplémentaires possibles lorsque l'on possède de

l'information (même incomplète) sur les dérivées partielles de f : dans l'étape de sonde, par exemple, si la direction du gradient au centre de sonde est connue, il sera possible de ne conserver qu'un seul point dans le cadre.

[0] Initialisations

x_0, Δ_0, V_0, $k \leftarrow 0$, ω^-, ω^+

[1] Itération k

RECHERCHE GLOBALE (optionnelle)

évaluer f sur $S_k \subset M_k$

SONDE LOCALE (optionnelle si la RECHERCHE GLOBALE a réussi)

choisir $D_k \subseteq D$ ensemble générateur positif

évaluer f sur $P_k \subset M_k$

[2] Mises à jour

si il existe $x \in S_k \cup P_k$ tel que $f(x) < f(x_k)$ (succès)

$x_{k+1} \leftarrow x$

choisir $\omega_k \in \{0, 1, \ldots, \omega^+\}$

sinon (échec)

$x_{k+1} \leftarrow x_k$

choisir $\omega_k \in \{\omega^-, \omega^- + 1, \ldots, -1\}$

$\Delta_{k+1} \leftarrow \tau^{\omega_k} \Delta_k$

$k \leftarrow k + 1$

aller en [1] si aucune condition d'arrêt vérifiée

Figure 1.3 – L'algorithme de recherche par motifs GPS.

1.3.2 Exemple pathologique pour GPS

Prenons l'exemple tiré de [81] avec la fonction

$$f(x) = \left(1 - e^{-\|x\|^2}\right) \max\left\{\|x - c\|^2, \|x - d\|^2\right\} ,$$

$x \in \mathbb{R}^2$ et $c = -d = [30\ 80]^T$ (courbes de niveau représentées à la figure 1.4 reprise de [20]). L'optimum est $x^* = [0\ 0]^T$ pour $f(x^*) = 0$. f est Lipschitz et strictement différentiable près de x^*. Trois essais sont effectués, avec $x_0 = [-3.3\ 1.2]^T$, et, $\forall k$,

- $D_k = D = \{e^1, e^2, -e^1, -e^2\}$ *ou*
- $D_k = D = \left\{[1\ 0]^T, [0\ 1]^T, \left[-\sqrt{2}/2\ \ -\sqrt{2}/2\right]^T\right\}$ *ou*
- $D_k = D = \left\{[1\ 0]^T, \left[-1/2\ \ \sqrt{3}/2\right]^T, \left[-1/2\ \ -\sqrt{3}/2\right]^T\right\}$

(directions représentées à la figure 1.5). Chaque essai converge vers $\hat{x} = [-3.2\ 1.2]^T$ où f n'est pas différentiable. On a bien que la dérivée généralisée de Clarke [35] est non négative pour toutes les directions de D, mais $\hat{x}$ n'est pas un optimum. On peut constater, sur la figure 1.4, qu'au point $\hat{x}$, aucune direction de descente ne peut être générée par l'algorithme GPS. Cet exemple met en évidence, de la même manière qu'en 1.2.3, que le fait d'utiliser un nombre fini de directions peut conduire à de mauvais résultats. D'autres exemples pathologiques sont décrits dans [10], dont même le cas de l'optimisation d'une fonction continûment différentiable sur $\mathbb{R}^2$ et où une sous-suite d'itérés de GPS converge vers un point où le gradient est non nul.

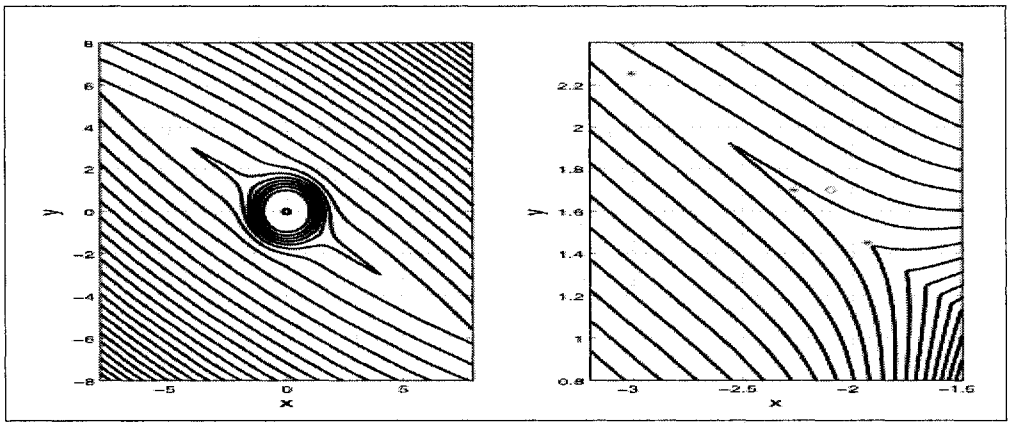

Figure 1.4 – Courbes de niveau de $\left(1 - e^{-\|x\|^2}\right) \max\{\|x - c\|^2, \|x - d\|^2\}$.

1.4 L'algorithme MADS

L'algorithme MADS est tiré de Audet et Dennis [20]. Il correspond en fait à une famille d'algorithmes dont plusieurs implémentations pratiques sont possibles. MADS est

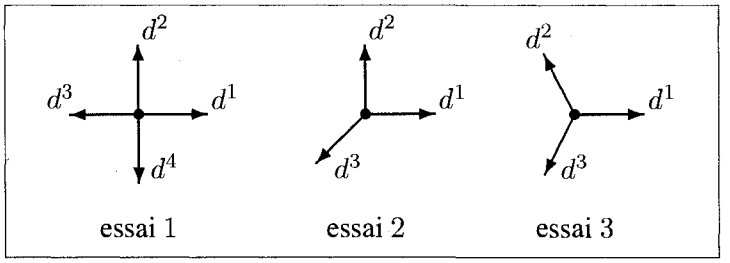

Figure 1.5 – Directions utilisées pour minimiser f avec GPS.

une généralisation de l'algorithme GPS, auquel il apporte un ensemble de directions de sonde dense. Trois codes informatiques de MADS sont disponibles : le premier avec le logiciel NOMAD [15], le deuxième dans la bibliothèque GADS de MATLAB [119], et le troisième dans [1]. Nous présenterons d'abord l'algorithme général, puis son analyse de convergence, et nous terminerons en décrivant l'implémentation LTMADS suggérée dans [20].

1.4.1 Description de l'algorithme MADS

Les contraintes sont gérées par un mécanisme dit de barrière extrême, qui sera détaillé en 1.5. L'algorithme possède la même structure que celui de la figure 1.3. La différence majeure avec GPS se situe au niveau des directions utilisées dans la sonde locale : l'ensemble des directions de sonde D_k n'est désormais plus un sous-ensemble de D, l'ensemble des directions qui sert à définir le treillis. Il en résulte que les ensembles de directions D_k peuvent être choisis de façon à définir un ensemble dense de directions normalisées, ce qui signifie que n'importe quelle direction sur le cercle unité est susceptible d'être approchée par une des directions de sonde normalisée, utilisée à une itération donnée de l'algorithme.

Le rôle du paramètre de taille de treillis de GPS est découplé : tandis que Δ_k est désormais noté Δ_k^m, un nouveau paramètre est introduit, Δ_k^p, le paramètre de taille de

cadre. A l'itération k, V_k est l'ensemble des points où f a été évaluée, et on redéfinit le treillis de la sorte :

$$M_k = \{x + \Delta_k^m Dz \ : \ x \in V_k, z \in \mathbb{N}^p\} \ .$$

L'itéré courant x_k est également appelé *centre de sonde*, qui devient :

$$P_k = \{x_k + \Delta_k^m d \ : \ d \in D_k\} \subseteq M_k \ .$$

Les conditions pour les directions sont :

- $D = GZ \in \mathbb{R}^{n \times p}$ doit être un ensemble générateur positif , avec $G \in \mathbb{R}^{n \times n}$ non singulière et $Z \in \mathbb{Z}^{n \times p}$.
- A l'itération k, D_k doit aussi être un ensemble générateur positif.
- Pour toute direction de sonde $d \in D_k$, il existe un vecteur d'entiers $u \in \mathbb{N}^p$ tel que $d = Du$.
- Les limites (telles que définies dans Coope et Price [40]) des ensembles D_k normalisés sont des ensembles générateurs positifs.
- La distance entre x_k et un point $x_k + \Delta_k^m d$ de P_k ($d \in D_k$) est bornée par un multiple de Δ_k^p :

$$dist(x_k, x_k + \Delta_k^m d) = \Delta_k^m \|d\|_\infty \leq \Delta_k^p \max\{\|d'\|_\infty : d' \in D\} \ .$$

L'observation primordiale à faire ici est qu'on n'a plus $D_k \subseteq D$. Les paramètres de treillis, quant à eux, doivent vérifier

$$\begin{cases} \Delta_k^m \leq \Delta_k^p \quad \text{pour tout } k \\ \lim\limits_{k \in K} \Delta_k^m = 0 \Leftrightarrow \lim\limits_{k \in K} \Delta_k^p = 0 \text{ , pour tout ensemble infini d'indices } K. \end{cases}$$

La figure 1.6 montre un exemple de cadres et de directions générées selon MADS, selon différentes valeurs de Δ_k^m et Δ_k^p. Les points du cadre, p^1, p^2 et p^3, peuvent être

choisis partout sur le treillis à l'intérieur des lignes en gras. Le nombre de points de sonde distincts est potentiellement de 24 dans la première situation de la figure, de 80 dans la deuxième et de 288 dans la dernière.

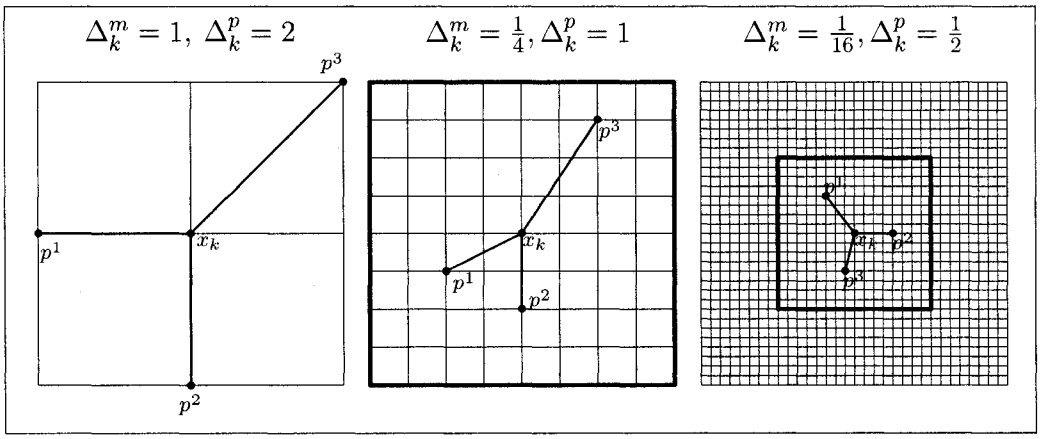

Figure 1.6 – Exemple de cadres de MADS $P_k = \{p^1, p^2, p^3\}$.

Notons que la stratégie de recherche incomplète vue en 1.2.2 peut s'appliquer, tandis que la stratégie d'ordre dynamique des directions doit être adaptée : elle consistait à promouvoir une direction de succès en tête de la liste des directions pour l'étape suivante de sonde. On ne peut plus le faire avec MADS parce qu'un point généré selon cette direction ne serait pas forcément sur le treillis. Il est possible toutefois de mimer ce comportement, en générant un point supplémentaire pour l'étape de recherche globale, plus loin selon la direction de succès.

1.4.2 Analyse de convergence de MADS

Nous résumons les résultats de convergence des algorithmes de type MADS, tel qu'ils apparaissent dans [20]. L'analyse de convergence est basée sur le calcul de Clarke [35] et sa définition de dérivée généralisée moins contraignante sur les hypothèses que doit vérifier f (typiquement il suffit que f soit Lipschitz pour que sa dérivée généralisée de

Clarke soit définie).

Cette analyse de convergence expose les propriétés de MADS de façon hiérarchisée, en supposant de plus en plus de propriétés pour f, pour montrer au final que si MADS est utilisé pour minimiser une fonction différentiable, il fournira un point ou le gradient est nul (même si MADS n'est pas conçu à la base dans le but d'optimiser des fonctions différentiables).

Tout d'abord, les paramètres Δ_k^p et Δ_k^m vérifient

$$\liminf_{k\to\infty} \Delta_k^p = \liminf_{k\to\infty} \Delta_k^m = 0 \, .$$

À l'itération k, si pour tout $x \in P_k$ on a $f(x) \geq f(x_k)$, l'itéré courant x_k est appelé un *optimum local du cadre*. Une sous-suite $\{x_k\}_{k\in K}$ d'optima locaux du cadre est dite *raffinante* si $\{\Delta_k^p\}_{k\in K}$ converge vers 0. Il est montré dans [18] qu'il existe au moins une sous-suite raffinante convergente (ceci est le premier résultat de MADS, qu'on appelle résultat de convergence d'ordre zéro).

On note $\hat{x} \in \Omega$ le point obtenu par l'algorithme MADS. Autrement dit, $\hat{x}$ est la limite d'une sous-suite raffinante $\{x_k\}_{k\in K}$. Si on suppose que f est Lipschitz près de $\hat{x}$ et que le cône hypertangent à Ω en $\hat{x}$ est non vide, alors $\hat{x}$ est un point stationnaire de Clarke, c'est-à-dire que la dérivée généralisée de Clarke de f en $\hat{x}$ selon toutes les directions du cône tangent de Clarke en $\hat{x}$ sont non négatives (les différentes définitions de cônes sont données dans [20]).

Il en découle que si f est strictement différentiable en $\hat{x}$, alors $\hat{x}$ est un point KKT de Bouligand de f sur Ω (la dérivée directionnelle de f dans les directions du cône de Bouligand sons non négatives), et si en plus $\Omega = \mathbb{R}^n$, $\hat{x}$ est un points stationnaire : $\nabla f(\hat{x}) = 0$.

Contre toute intuition, puisque les algorithmes de recherche directe n'utilisent pas

les premières dérivées, on montre que des résultats de convergence d'ordre deux sont disponibles dans [4] (c'était également le cas pour GPS [2]).

1.4.3 Une première implémentation de MADS : LTMADS

Les résultats majeurs de l'analyse de convergence que nous venons de résumer reposent sur l'hypothèse que l'ensemble des directions raffinantes (les directions des itérations qui ont échoué) doit être dense. Toute implémentation pratique de MADS doit donc faire en sorte que cette condition soit respectée. C'est le cas de l'implémentation LTMADS donnée dans [18] (LT pour *Lower Triangular*).

Le logiciel NOMAD [15] est codé selon cette implémentation. Les éléments suivants sont définis :

$$\left\{ \begin{array}{rcl} p & = & n+1 \text{ ou } 2n \text{ (nombre de directions)} \\ D & = & [I_n \ -I_n] \quad (G = I_n \text{ et } Z = [I_n \ -I_n]) \\ \tau & = & 4 \\ \omega^- & = & -1 \\ \omega^+ & = & 1 \\ \Delta_0^m & = & \Delta_0^p \ = 1 \, . \end{array} \right.$$

Lorsque x_k est un minimum local du cadre, on a donc la mise a jour $\Delta_{k+1}^m \leftarrow \Delta_k^m/4$, et dans le cas contraire, $\Delta_{k+1}^m \leftarrow 4^{\omega_k}\Delta_k^m$ avec $\omega_k \in \{0,1\}$. Si $\Delta_k^m < 1$, on prend $\omega_k = 1$, et sinon $\omega_k = 0$. Ainsi Δ_k^m est une puissance de 4 ne dépassant jamais 1. Un entier non négatif ℓ vérifiant $\Delta_k^m = 4^{-\ell}$ est introduit. Il sera plus grandement question de ℓ dans le chapitre 5 proposant une alternative à LTMADS.

À l'itération k, le choix des directions est basé sur une matrice B triangulaire inférieure (d'où le nom de l'implémentation), que l'on construit de cette façon : on définit

d'abord $B = (b_{ij}) \in \mathbb{R}^{n \times n}$ telle que

$$\text{pour tous } i, j \in \{1, 2, \ldots, n\}, \begin{cases} b_{ij} = \pm 2^{\ell} \text{ si } i = j \\ b_{ij} \in \{-2^{\ell} + 1, -2^{\ell} + 2, \ldots, 2^{\ell} - 1\} \text{ si } i > j \\ b_{ij} = 0 \text{ sinon.} \end{cases}$$

Tous les éléments de la partie inférieure de B, plus la diagonale, sont des entiers tirés au hasard avec la même probabilité. Les lignes et les colonnes de B sont ensuite permutées : les lignes pour faire en sorte que la direction selon un seul axe ne soit pas toujours sur ce même axe, et les colonnes pour qu'il n'y ait pas d'ordre implicite dans les directions. La matrice ainsi obtenue est notée B', qui, telle que construite, est une base de $\mathbb{R}^n$ qu'il faut compléter pour obtenir D_k, un ensemble générateur positif. On donne le choix entre des ensembles de $n + 1$ ou de $2n$ directions :

- Complétion à une base positive minimale : poser $D_k = [B' \; d_k^{n+1}]$ avec $d_k^{n+1} = -\sum_{j=1}^{n} d_k^j \in \mathbb{R}^n$ et où chaque vecteur d_k^j correspond à la $j$ème colonne de B', $j = 1, 2, \ldots, n$.
- Complétion à une base positive maximale : poser $D_k = [B' \; -B']$.

Le lien entre Δ_k^p et Δ_k^m dépend du choix du nombre de directions :

- $n + 1$ directions : $\Delta_k^p = n\sqrt{\Delta_k^m} = n2^{-\ell} \geq \Delta_k^m$.
- $2n$ directions : $\Delta_k^p = \sqrt{\Delta_k^m} = 2^{-\ell} \geq \Delta_k^m$.

Il est démontré dans [16,20] que LTMADS est bien une instance valide de MADS, et que l'ensemble des directions de sonde normalisées est dense dans le cercle unité avec une probabilité de un. On note aussi qu'un erratum corrigeant une des preuves de l'analyse de convergence pour LTMADS est donné dans [16].

1.5 Traitement des contraintes avec GPS et MADS

On rappelle que les contraintes auxquelles le problème $\mathcal{P}$ est sujet sont représentées par l'ensemble $\Omega = \{x \in X : g(x) \leq 0\} \subseteq X$. L'algorithme GPS avec contraintes de

bornes et contraintes linéaires (définissant l'ensemble X) a été décrit dans [86,87,92], où des directions conformes à la géométrie de X sont générées. Cette technique est étendue pour les contraintes linéaires dégénérées en [7]. Une méthode de pénalité pour traiter les contraintes générales, basée sur un lagrangien augmenté, est également proposée dans [88]. Enfin, les contraintes particulières que sont les variables de catégorie sont traitées dans [2,3,17,24,79].

Deux autres stratégies peuvent être employées pour traiter les contraintes générales : la barrière extrême et le filtre. La méthode de la barrière extrême consiste à rejeter systématiquement tout point en dehors du domaine réalisable Ω. Ceci revient à optimiser sans contraintes une nouvelle fonction $f_\Omega : \mathbb{R}^n \to \mathbb{R} \cup \{+\infty\}$ définie de la façon suivante :

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

La méthode de la barrière extrême est celle utilisée par MADS dans [20], tandis que la méthode du filtre est utilisée pour GPS. Les algorithmes de filtre ont été présentés par Fletcher et Leyffer [55–57]. Ils sont plus simples qu'une méthode de pénalité, car ils n'exigent pas la gestion d'un paramètre de pénalité. Un algorithme de filtre consiste à introduire une fonction h qui accumule les violations des contraintes de façon à traiter le problème comme étant biobjectif, un point n'étant accepté comme candidat au prochain itéré que s'il diminue l'objectif f ou la violation des contraintes h (avec une priorité donnée à la réalisabilité). La méthode du filtre a été adaptée pour les algorithmes GPS dans [19], qui comprend également une analyse de convergence.

Il est à noter que le traitement des contraintes dans MADS par la méthode du filtre a évolué en notion de barrière progressive dans [21], qui traite les contraintes de X avec la barrière extrême et les autres avec un nouveau mécanisme permettant des itérés non réalisables (ce pourquoi les contraintes de X sont nommées fermées et les autres ouvertes).

1.6 Utilisation de fonctions substitut

Nous avons vu que les algorithmes MADS traitent des problèmes pour lesquels les coûts d'évaluation des fonctions composant le problème $\mathcal{P}$ (f et g) sont élevés. L'idée ici est d'utiliser des fonctions substitut (*surrogate functions*) f_S et g_S approchant la fonction objectif f et les contraintes g, et qui sont peu coûteuses à évaluer.

Les fonctions substitut ne sont pas obligées d'être de bonnes approximations. Ce point est illustré dans [23] où les fonctions substitut utilisées sont d'ordres de grandeur totalement différents des fonctions originales. Elles constituent toutefois de bons substituts car elles miment le comportement des fonctions de départ (optima et allure sensiblement proches).

Dans le cas idéal, de bonnes fonctions substitut pourraient être déjà disponibles à l'avance : elles sont fournies par l'utilisateur, c'est-à-dire, la personne qui définit le problème $\mathcal{P}$, et ainsi il n'y a pas besoin de les construire. Sinon, lorsqu'aucun substitut n'est disponible, leur construction et calibrage doivent être effectués à partir de rien. C'est exactement ce que font les méthodes travaillant sur des modèles de f, évoquées au début de ce chapitre.

Des fonctions substitut sont utilisées, par exemple, dans [27, 28] pour optimiser la conception d'un rotor d'hélicoptère, et elles sont intégrées dans l'optimiseur DE (*Boeing Design Explorer*) [13, 29, 30]. L'emploi de fonctions substitut dans un contexte général d'optimisation est évoqué dans [9, 29, 48, 110].

Les fonctions substitut peuvent s'intégrer aisément dans MADS, grâce à la flexibilité de l'étape de recherche globale. Des stratégies sont exposées dans [23], comme utiliser f_S pour ordonner les points de la sonde (au lieu d'utiliser la stratégie d'ordre dynamique des directions), ou d'éliminer des points de la recherche globale jugés non prometteurs. On peut aussi, lors de la première recherche globale, exécuter une autre instance de MADS avec $f = f_S$. Nous proposons au chapitre 3 une nouvelle façon d'employer les

fonctions substitut.

1.7 Références additionnelles

Pour clore ce chapitre, nous indiquons au lecteur que des références additionnelles se trouvent disséminées dans le reste du texte : on trouvera les références sur la méta-heuristique de recherche à voisinage variable (VNS) dans le chapitre 3, ainsi qu'une revue complète sur les méthodes parallèles de recherche directe dans le chapitre 4.

CHAPITRE 2

DÉMARCHE ET ORGANISATION DE LA THÈSE

Nous présentons ici la démarche et l'organisation du travail de recherche accompli dans le cadre de cette thèse. On expose également les motivations et les résultats initialement espérés.

Au précédent chapitre, l'algorithme MADS a été présenté. Cette méthode se propose de résoudre $\mathcal{P}$ et plusieurs références comme [20,23,98] suggèrent son efficacité. Le caractère récent de la méthode (2006) indique clairement que des apports sont possibles. Ce que propose cette thèse est donc d'apporter des extensions à MADS afin d'en améliorer certains aspects.

La thèse est rédigée par articles, au nombre de trois, constituant les chapitres 3, 4, et 5. Cette présentation a été établie selon l'ordre chronologique, du travail le plus ancien au plus récent. Chaque article présente un apport nouveau à MADS. Le travail commun à chaque article est triple : tout d'abord, chaque nouvelle extension est expliquée et justifiée. Ensuite, on prouve que l'analyse de convergence de MADS est préservée. Enfin, des tests numériques doivent confirmer que l'extension remplit son office et permet d'améliorer certains aspects de MADS.

Nous exposons maintenant la démarche relative à chaque article. L'article [12] correspondant au chapitre 3 est publié dans la revue *Journal of Global Optimization*. Il propose d'intégrer une méthode heuristique dans MADS. Pour résoudre des problèmes du type de $\mathcal{P}$ possédant beaucoup d'optima locaux, on observe que MADS, et sa seule implémentation existante LTMADS, produisent des résultats très hétérogènes. Ceci s'explique par le fait que LTMADS comporte une composante aléatoire qui fait que deux exécutions successives ne donneront pas forcément le même résultat, et donc certaines exécutions

de l'algorithme conduisent à des optima locaux meilleurs ou pires que d'autres. Une idée pour remédier à la diversité des résultats obtenus, gagner en stabilité, et donc avoir plus souvent de bonnes solutions, est d'utiliser une stratégie de recherche globale avant chaque sonde locale (voir algorithme 1.3). Deux stratégies génériques et simples évoquées dans la littérature pour l'étape de recherche globale de MADS sont une recherche aléatoire et une recherche basée sur l'échantillonnage d'hypercubes latins [117, 118]. Ces deux stratégies ne fournissant pas la stabilité de résultats désirée, nous avons cherché une nouvelle méthode générique de recherche globale. C'est ici qu'est entrée en jeu la méta-heuristique VNS [68, 100]. Cette méthode, principalement utilisée en optimisation combinatoire, est détaillée au chapitre 3, mais ce que nous pouvons déjà en dire est qu'elle est très efficace justement pour les problèmes possédant beaucoup d'optima locaux. En effet, la méthode possède un mécanisme dit de perturbation permettant de s'échapper de ces bassins locaux. De plus, on a constaté que MADS et VNS possédaient un comportement complémentaire : alors que MADS explore l'espace des variables de plus en plus proche des itérés courants après des échecs, VNS fait le contraire et explore l'espace des variables de plus en plus loin des solutions courantes (justement pour échapper aux optima locaux sur lesquels on est bloqué). L'article du chapitre 3 consiste donc à utiliser VNS comme méthode de recherche globale de MADS. La réunion des deux méthodes est nommée MADS-VNS. On montre en outre une nouvelle façon d'utiliser les fonctions substitut dans le VNS adapté que nous utilisons. L'analyse de convergence de ce couplage utilise la flexibilité de l'étape de recherche globale de MADS, qui montre que tant que VNS génère un nombre fini de points d'évaluation du treillis, toutes les propriétés de convergence de MADS résumées en 1.4.2 sont conservées. Des tests numériques sont effectués sur trois problèmes dont un correspond à un problème réel d'ingénierie, et un autre à un problème d'optimisation multidisciplinaire (MDO). Les résultats espérés sont une plus grande stabilité dans la qualité des solutions et des meilleures performances que les méthodes de recherche globale usuelles.

Le chapitre 4 correspond à [22], qui a été accepté pour publication dans la revue SIAM *Journal on Optimization*. Deux autres aspects de MADS nécessitaient une amélioration : premièrement, il n'en existe pas de version parallèle, et deuxièmement la taille des problèmes traités de façon efficace n'excède pas l'ordre de quelques dizaines de variables. L'article sur la décomposition parallèle des variables (PVD) de Ferris et Mangasarian [53], bien qu'en dehors du contexte de l'optimisation non lisse, semblait apporter la réponse à ces deux problèmes à la fois. Les auteurs de [53] proposent de décomposer le problème initial en plusieurs sous-problèmes de plus petite dimension, et de résoudre ces sous-problèmes en parallèle. Ainsi, chaque processeur travaille uniquement sur un sous-ensemble des variables du problème initial. Une étape de synchronisation permet de diriger la recherche et de distribuer l'information aux processeurs. Ce principe a été entièrement adapté pour MADS, dans le but de traiter des problèmes de plus grande taille (de l'ordre de centaines de variables). L'algorithme résultant, PSD-MADS, est donc parallèle, mais également asynchrone, car l'étape de synchronisation qui constituait une barrière parallèle (c'est-à-dire qui imposait, à chaque itération de l'algorithme, d'attendre que tous les processeurs soient arrivés à la même étape), a été transformée de manière à ne plus bloquer aucun processeur. L'analyse de convergence de cette méthode est moins triviale que dans le cas du couplage MADS-VNS. Elle est néanmoins assurée par l'introduction d'un processeur spécial, qui, lorsque l'on se place de son point de vue, exécute un algorithme MADS valide (*i.e.* respectant les conditions de convergence de MADS), alors que les évaluations des autres processeurs constituent en fait l'étape de recherche globale de cet algorithme. L'objectif de l'article est de présenter PSD-MADS, son analyse de convergence, et des premiers tests numériques. Les stratégies pour décider quelles variables sont distribuées à quels processeurs n'est pas abordée, et les variables sont distribuées de façon aléatoire. Des tests numériques sont effectués sur des problèmes analytiques, et l'algorithme est comparé à une version parallèle synchrone et simple de MADS et à APPS, la version parallèle asynchrone de GPS déjà évoquée en 1.1.

Le chapitre 5 correspond au dernier et plus récent article [6], soumis à la revue SIAM *Journal on Optimization*. Il propose une nouvelle implémentation de MADS, ORTHO-MADS, en remplacement de LTMADS. ORTHOMADS a été conçu spécialement pour combler deux failles de LTMADS : la première, déjà constatée plus tôt, est que LT-MADS est une méthode non déterministe et que deux exécutions successives ne donneront pas la même solution. Ce non déterminisme est également un frein à la reproductibilité des expériences (même avec la même graîne aléatoire, deux machines différentes ont peu de chances d'obtenir le même résultat). Le deuxième défaut de LT-MADS est que les directions utilisées, bien que formant un ensemble dense (sous une hypothèse probabiliste), ne sont pas orthogonales. Ceci peut entraîner, en pratique, si l'on interrompt l'algorithme au bout d'un certain temps, de grands trous dans les directions utilisées, c'est-à-dire de larges régions de l'espace qui n'auront jamais pu être explorées. La nouvelle implémentation ORTHOMADS corrige ces deux défauts en utilisant une suite déterministe au comportement aléatoire, la suite de Halton [65], qui permet la génération d'un ensemble dense de directions, en se passant d'hypothèse probabiliste. La transformation de Householder [75] permet ensuite d'obtenir des directions orthogonales, tout en conservant la propriété de densité. Alors que les deux premiers articles de la thèse exploitaient la flexibilité de l'étape de recherche globale pour prouver leur convergence, ORTHOMADS procède autrement. En effet, comme c'est une nouvelle implémentation de MADS, il faut prouver que les nouvelles directions utilisées se conforment aux conditions introduites dans [20], principalement qu'elles constituent un ensemble dense. ORTHOMADS se détache également des deux premiers articles dans le sens où les résultats numériques sont effectués de façon moins détaillée mais sur un plus grand nombre de problèmes (45). Notre point était de comparer exhaustivement la nouvelle implémentation à l'ancienne, et, ayant constaté que rares sont les méthodes de recherche directe bonnes ou mauvaises sur tous les problèmes, un plus large éventail de problèmes était nécessaire. Ce que l'on aimerait voir ressortir de ces tests est que ORTHOMADS fasse au moins jeu égal avec LTMADS, tout en restant supérieur à GPS.

Nous exposons maintenant les trois articles composant cette thèse dans les trois prochains chapitres.

CHAPITRE 3

NONSMOOTH OPTIMIZATION THROUGH MESH ADAPTIVE DIRECT SEARCH AND VARIABLE NEIGHBORHOOD SEARCH[1]

Charles Audet[2] Vincent Béchard[3] Sébastien Le Digabel[4]

August 2007

Abstract

This paper proposes a way to combine the Mesh Adaptive Direct Search (MADS) algorithm, which extends the Generalized Pattern Search (GPS) algorithm, with the Variable Neighborhood Search (VNS) metaheuristic, for nonsmooth constrained optimization. The resulting algorithm retains the convergence properties of MADS, and allows the far reaching exploration features of VNS to move away from local solutions. The paper also proposes a generic way to use surrogate functions in the VNS search. Numerical results illustrate advantages and limitations of this method.

Keywords: Nonsmooth optimization, Mesh Adaptive Direct Search, Generalized Pattern Search, Variable Neighborhood Search.

---

[1]Work of the first author was supported by FCAR grant NC72792 and NSERC grant 239436-05, AFOSR F49620-01-1-0013, the Boeing Company and ExxonMobil Upstream Research Company. Work of the first and third author was supported by the Consortium for Research and Innovation in Aerospace in Quebec.

[2]GERAD and Département de mathématiques et de génie industriel, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montréal (Québec), H3C 3A7 Canada www.gerad.ca/Charles.Audet, Charles.Audet@gerad.ca.

[3]Département de mathématiques et de génie industriel, École Polytechnique de Montréal vincent.bechard@polymtl.ca.

[4]Département de mathématiques et de génie industriel, École Polytechnique de Montréal Sebastien.Le.Digabel@gerad.ca.

3.1 Introduction

The paper considers optimization problems of the form

$$\min_{x \in \Omega \subseteq \mathbb{R}^n} f(x) \tag{3.1}$$

where $f : \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$, $\Omega \doteq \{x \in X : c_j(x) \leq 0,\ j = 1, 2, \ldots, m\}$ and $X \subseteq \mathbb{R}^n$ represents closed constraints, i.e. constraints that necessarily need to be satisfied in order for the functions to evaluate. The closed constraints often include bound constraints $L \leq x \leq U$ with L and U in $(\mathbb{R} \cup \{\pm\infty\})^n$, and possibly boolean constraints that indicate if they are satisfied or not, and in the latter case, there is no quantification by which they are violated.

The functions $c_j : \mathbb{R}^n \to \mathbb{R}$ for $j = 1, 2, \ldots, m$ represent the other constraints and are referred to as the open constraints.

The objective function f and the different functions defining the set Ω are typically provided as black-boxes in the sense that the way to obtain a function value from a given $x \in \mathbb{R}^n$ is not provided in an analytical way, or may be time consuming or expensive to evaluate. The black-boxes may also fail to return a value at some points. One way to model this is by setting the function value to infinity. This is called the *barrier approach*. The case where some properties, differentiability for example, can not be exploited, are considered. Such black-box functions are widely used in different engineering disciplines [3, 8, 23, 27, 28, 58, 71, 79, 96]. Black-box functions are typically evaluated by running computer code. Approximations of the black-box functions can also be made through easier to evaluate surrogate functions (see [29]), and this paper proposes a different way to exploit such surrogates.

Different derivative-free direct search methods are designed for problem (3.1), such as GPS [29, 120], MADS [4, 20] and DIRECT [54, 77]. The reader may consult [81, 89] for

surveys of direct search methods. Under appropriate conditions, these methods ensure convergence to a point satisfying necessary optimality conditions based on the Clarke calculus [35].

In the present paper, we exploit the flexibility of the MADS algorithm so as to include the far reaching searches of the Variable Neighborhood Search (VNS [68, 100]). The fundamental structures of MADS and VNS are complementary: on the one hand, in case of failure to identify improved points, VNS explores increasingly larger regions, and on the other hand, MADS explores smaller and smaller neighborhoods. The purpose of this paper is to present a generic coupling of MADS and VNS that may be applied to the class of problems for which MADS was designed.

The main reason why we chose to combine VNS with the MADS algorithm instead of another optimization method is that the convergence analysis of the resulting method follows directly. Each MADS iteration is partitioned into a SEARCH and a POLL step. The SEARCH step is intended to be flexible (but still must satisfy some minimal requirements), and the POLL step must follow strict rules. The MADS algorithm was conceived in such a modular way precisely to allow the user to create and use his own SEARCH strategies. In the present paper we take advantage of the flexibility of the SEARCH step by proposing a generic VNS SEARCH step. The way in which VNS generates trial points makes it easy to verify that the SEARCH requirements are satisfied. The MADS POLL step and the update rules are the same as in [20], and thus the MADS convergence analysis holds.

The paper is divided as follows. Section 3.2 proposes an overview of the MADS and VNS methods. Section 3.3 presents a generic algorithm that couples MADS and VNS and allows the use of surrogate functions. Section 3.4 describes a practical implementation. Finally some numerical results, including the detailed description of an engineering problem, are presented in Section 3.5. The proposed algorithm is compared to the classic MADS algorithm and to MADS with a classic SEARCH strategy.

3.2 Descriptions of the MADS and VNS algorithms

3.2.1 MADS

The MADS algorithm [20] for problem (3.1) extends the Generalized Pattern Search (GPS) algorithm for linearly constrained optimization [29,120]. Both GPS and MADS are iterative algorithms where the black-box functions are evaluated at some trial points, which are either accepted as new iterates or rejected. At any iteration (denoted by the integer k), all trial points generated by these algorithms are constructed to lie on the mesh

$$M(k, \Delta_k) = \bigcup_{x \in V_k} \{x + \Delta_k D z : z \in \mathbb{N}^{n_D}\} \subset \mathbb{R}^n$$

where V_k is the set of points evaluated by the start of iteration k, $\Delta_k \in \mathbb{R}^+$ is the mesh size parameter, and D is a fixed matrix whose columns are in $\mathbb{R}^n$. In most cases, D is chosen to be the $n \times 2n$ matrix $[-I\ I]$ where I is the $n \times n$ identity matrix. We make the standard assumption that all the trial points are in a compact set C. This assumption, together with the fact that the mesh is constructed using integer combinations of $\Delta_k D$ ensures that $C \cap M(k, \Delta_k)$ contains a finite number of points.

In order to simplify the notation of the present paper, the parameter Δ_k is equivalent to Δ_k^m in [20].

Each iteration is divided into two main steps, the SEARCH and the POLL, followed by an update step that determines the success of the iteration. The new mesh size, the new current iterate and the stopping criteria are updated or verified at the end of the iteration.

The closed constraints defining the set X are handled by the barrier approach, as in MADS, consisting in rejecting trial points outside X. For the other constraints (c_j, $j = 1, 2, \ldots, m$), a filter approach is used [19,56]: if the points are in X, they are stored and classified, using their objective function value and a measure of the constraints violation, which permits the acceptance of promising points and the rejection of the others.

The POLL evaluates the functions f and c_j's at mesh points near the current iterate x_k. The convergence analysis of [20] relies on the rigid rules that the POLL step needs to follow. The POLL step remains unchanged in the present paper. The way of choosing the directions used to generate the POLL points makes the difference between GPS and MADS: in GPS, the normalized set of POLL directions is finite, whereas it may be asymptotically dense in the unit sphere with MADS, allowing a better exploration of the space of variables. At iteration k, the set containing the trial POLL points is called the frame P_k, given by $P_k = \{x_k + \Delta_k d : d \in D_k\}$, with D_k the set of directions used to construct P_k. D_k is a set formed by taking positive integer combinations of the columns of D. We will not say more about the POLL.

The SEARCH step is very flexible and gives the algorithm the opportunity to generate trial points anywhere on the mesh: the way of generating these points is free of any rules, as long as they remain on the current mesh $M(k, \Delta_k)$ and that the SEARCH terminates in finite time. This partition of an iteration into a SEARCH and a POLL steps is the key feature of MADS which is exploited in the present paper

Some SEARCH strategies are tailored for a specific application: for example if the problem is to optimize a wing shape, then some known wings models or configurations may be used. Other SEARCH strategies are generic, as the use of Latin Hypercube sampling [117, 118]. Furthermore, different types of SEARCHes may be combined. This paper introduces a generic SEARCH inspired by the VNS metaheuristic.

A high level description of the algorithm is summarized in Figure 3.1. The MADS parameters taken for the tests of Section 3.5 are the default parameters used in our NOMAD software [15]. The values of critical parameters (such as the initial mesh size parameter Δ_0) will be given in that section. We encourage the reader to consult [20] for a complete description of the algorithm.

A hierarchical convergence analysis is available for MADS, based on the black-boxes differentiability: the main convergence result is that under local Lipschitz assumptions,

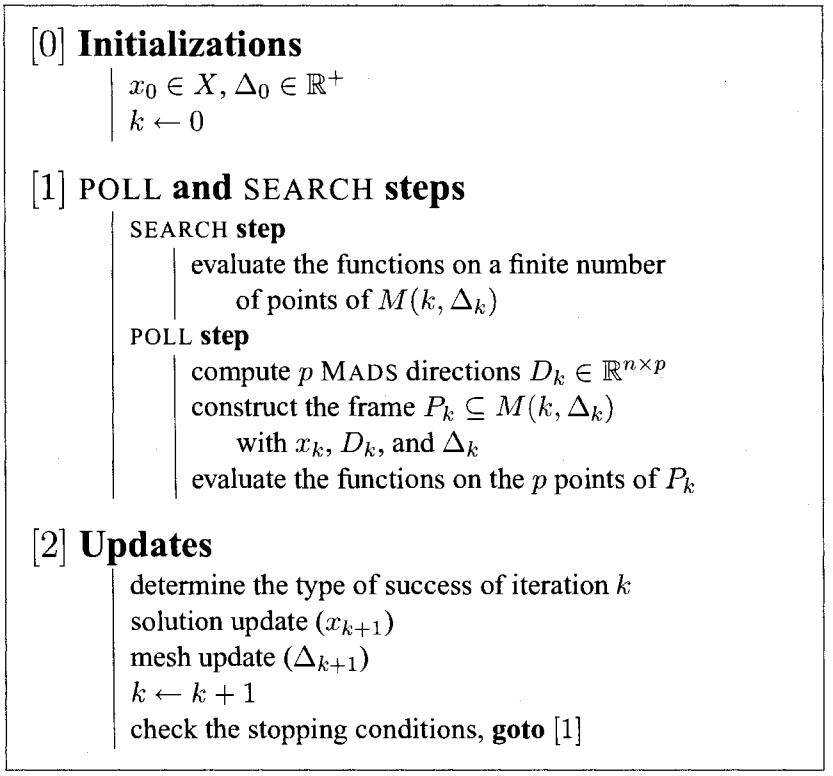

Figure 3.1: MADS Algorithm.

the algorithm produces a Clarke stationary point, i.e. a point $\hat{x} \in \Omega$ at which the generalized Clarke derivative of f is nonnegative for all directions in the Clarke tangent cone at $\hat{x}$ (see [35]). A corollary of this result is that, in the case without constraints, and if f is strictly differentiable, then $\nabla f(\hat{x}) = 0$.

3.2.2 VNS

The Variable Neighborhood Search (VNS) is a metaheuristic proposed by Hansen and Mladenović [68, 100]. It is applied to combinatorial problems [34, 67, 69, 70], but it is possible to use it with continuous variables as in [14, 33, 51] and in the present work.

Two fundamental elements are required to define a VNS method: a descent method and a neighborhood structure. The descent is a method executing moves with respect to the neighborhood structure, which defines all the different possible trial points reachable from the current solution. The objective of these moves is to improve the current solution, and they are repeated until no improvement is possible. The last point of the descent is a local optimum with respect to the neighborhood structure used.

Local searches often terminate in the vicinity of a nearby local optimum. VNS uses a random perturbation method to attempt to move away from a local optimal solution, far enough so that a new descent from the perturbed point leads to an improved local optimum, located in a new and hopefully deeper valley. The perturbation method relies on the neighborhood structure, and is parametrized by a nonnegative scalar ξ_k, the VNS amplitude at iteration k, which gives the order of the perturbation (it is not necessarily small, as the term "perturbation" might suggest, and "shaking" will be used for the routine executing it). The implementation details of the perturbation method has to be defined specifically for each type of problems, as long as the idea of amplitude is defined and is dependent on ξ_k. For example ξ_k could be a minimal desired distance between the two points before and after the perturbation, or the number of random elementary

moves leading to the perturbed point. The most efficient perturbation methods are often linked to the problem properties. In the present paper, a generic perturbation method is described.

A description of the VNS metaheuristic is given in Figure 3.2. The algorithm essentially consists of two loops. Each iteration of the inner loop is decomposed into two steps: first the current solution (typically a local optimum) is perturbed with an amplitude factor ξ_k, and then a descent is performed from the perturbed point. If a better solution is obtained, it becomes the new iterate, and the amplitude is reset to its initial value. Otherwise the amplitude is increased by a value $\delta > 0$ (called the VNS increment) so that the next perturbation will lead to a point further away than the previous one. Finally, the inner loop terminates after a maximum amplitude ξ_{max} is reached.

The outer loop consists in repeating this process it_{max} times. The it_{max} parameter of the first level loop is crucial for the efficiency of most VNS implementations. However, in our context, this loop will implicitly be made by the MADS algorithm, and therefore we fix $it_{max} = 1$.

3.3 Coupling the MADS and VNS algorithms

The VNS algorithm and the MADS POLL step have a complementary behavior: when no success has been made during an iteration, the next POLL step generates trial points closer to the POLL center, while the VNS algorithm explores a more distant region with a larger perturbation amplitude. This paper proposes to incorporate the VNS method in the MADS algorithm, as a SEARCH step (called the VNS SEARCH). The POLL step remains unchanged so that the convergence analysis of MADS still holds.

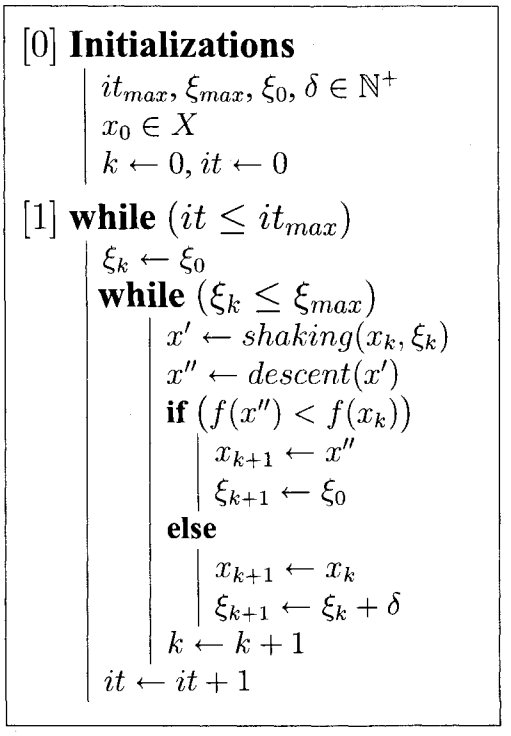

Figure 3.2: VNS metaheuristic for minimizing $f : \mathbb{R}^n \rightarrow \mathbb{R}$.

3.3.1 General description

The MADS mesh provides a natural neighborhood structure to be used by the two VNS components (descent and perturbation) and only the update of the perturbation amplitude ξ_k has to be made outside of the VNS SEARCH step.

The entire convergence analysis of MADS is preserved when the two following conditions are met: first, at iteration k, all the VNS SEARCH trial points must lie on the mesh $M(k, \Delta_k)$, and second, their number must be finite. The general way to define the perturbation and the descent is now given, and in the next section, a practical implementation will be described and proved to be a valid SEARCH, by verifying that the two conditions are satisfied.

Adding a VNS exploration in the SEARCH step of a MADS algorithm can be done by introducing two new parameters. One parameter is $\Delta_V > 0$ and relates to the VNS shaking method, as described in the next paragraph. The other parameter is $\rho > 0$ and defines a stopping criteria for the descent. It is introduced at the end of this subsection.

The shaking at iteration k generates a point x' belonging to the current mesh $M(k, \Delta_k)$. The amplitude of the perturbation is relative to a coarser mesh, whose coarseness is independent of Δ_k. We need to introduce the VNS mesh size parameter

$$\Delta_V > 0 \text{ and the VNS mesh } M(k, \Delta_V). \tag{3.2}$$

This parameter is constant and independent of the iteration number k. The reason why the VNS perturbation needs to be independent of the current mesh $M(k, \Delta_k)$ is that it should not be influenced by the specific MADS behavior (which is in fact the contrary of the VNS behavior, as was said in the introduction of this section). Only the fact that an iteration succeeds, or the number of successive failed iterations, can rule the VNS amplitude, as in the original VNS algorithm. Another way of viewing that fact is that for a given value of ξ_k, the perturbation has to be the same regardless of the mesh fineness

or coarseness.

The shaking may be viewed as the function

$$\begin{aligned} shaking: \ M(k,\Delta_k) \times \mathbb{N} &\rightarrow M(k,\Delta_V) \subseteq M(k,\Delta_k) \\ (x,\xi_k) &\mapsto x' = shaking(x,\xi_k) \end{aligned}$$

where $\xi_k \in \mathbb{N}$ is the perturbation amplitude.

As will be illustrated in the next section, the VNS mesh size parameter Δ_V can also be used as a criteria to decide if a VNS SEARCH should be performed at a given iteration.

VNS descent is viewed as a function

$$\begin{aligned} descent: \ M(k,\Delta_V) &\rightarrow M(k,\Delta_k) \\ x' &\mapsto x'' = descent(x') \end{aligned}$$

and has to generate a finite number of mesh points. While the shaking randomly changes a point in hopes of moving away from a local optimum, the idea of the descent is to obtain an improved point x'' from x' (in the filter sense of [19], i.e. in terms of objective value and constraints violation). Ideally, a descent step must lead toward a local optimum. The descent step in VNS is important because x', as a randomly perturbed point, has a weak probability of being an interesting point.

In the MADS context, local optimality is defined with respect to the mesh. The descent step ideally leads to a mesh local optimum, with respect to the current step size Δ_k and the directions used. The descent method described here is generic, but it is easy to see that a specialized descent method could be used for a given type of problem so as to exploit some inner properties. In order to reduce the overall number of functions evaluations, the descent step may terminate as soon as it generates a trial point, y, close

to another point, x, previously considered by the algorithm, i.e. when $\|x - y\|_\infty \leq \rho$,

$$\text{where} \quad \rho > 0 \quad \text{is called the Descent Stop (DS) parameter.} \tag{3.3}$$

The idea of this Descent Stop criteria is to avoid exploring a region previously visited. We believe that this strategy of terminating prematurely a descent could be exported to the general VNS metaheuristic. Figure 3.3 gives a description of the algorithm. In the update step, ξ_{k+1} is set to ξ_0 if no success has been achieved or if ξ_{max} has been reached. This last point allows us to mimic the outer loop of the original VNS algorithm, the loop on it in Figure 3.2.

3.3.2 Use of a static surrogate

Surrogate functions may be used in several ways in the context of GPS algorithms (see [29] for a generic framework using surrogates). They are useful when the functions defining the problem are costly to evaluate, because they are less complicated and give an approximation of the true functions. Surrogates do not need to be good approximations of the true functions: in [23], the surrogate function differed from f by a factor of roughly 200, but both f and the surrogate shared some similarities.

Two types of surrogates can be defined: static surrogates, tailored for a specific problem (see [28] for example), and dynamic surrogates, constructed dynamically during the execution of the algorithm through previous evaluations and possibly with some interpolation technique (kriging for example, see [91, 110]).

In [23], three strategies are proposed for the use of surrogates with the MADS algorithm: first an entire run of an optimization algorithm on the surrogate may lead to a good starting point for another run with the true function. A surrogate may be used to order the POLL and SEARCH trial points. The more promising points are evaluated first, and if an improvement is made, the others are not considered. Finally a surrogate may be

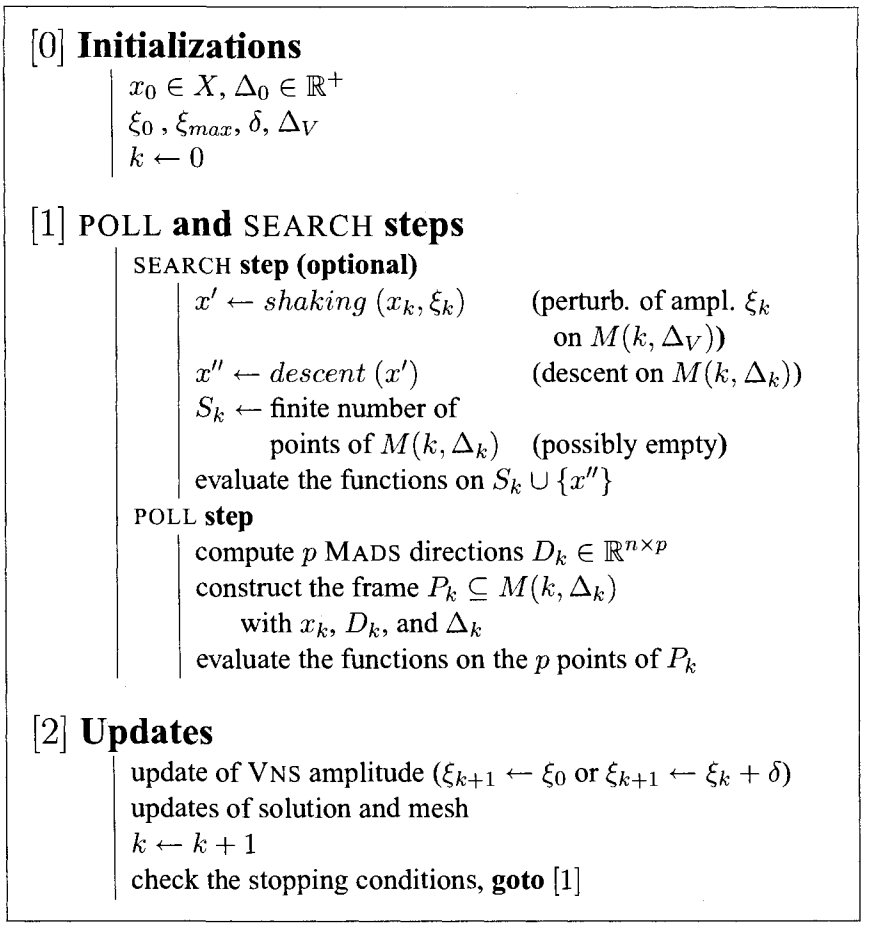

Figure 3.3: General algorithm of the coupling of MADS and VNS.

used to determine if a SEARCH point is valuable enough so that the true function is to be evaluated.

The use of surrogates in this paper can be seen as a fourth way to use them. The descent step can be entirely performed on the surrogate function. The true function can then be evaluated only once, at the final point produced by the descent. This strategy reduces the cost of a descent to a single expensive evaluation of the true function, and some less expensive surrogate functions.

3.4 Implementation

The algorithm presented in the previous section is generic and flexible. We now present a specific implementation, using the LTMADS implementation of [20], by specifying some parameter values and defining the perturbation and descent methods

$$shaking : M(k, \Delta_k) \times \mathbb{N} \rightarrow M(k, \Delta_V) \subseteq M(k, \Delta_k)$$

and

$$descent : M(k, \Delta_k) \rightarrow M(k, \Delta_k) \, .$$

For the perturbation $x' \leftarrow shaking(x_k, \xi_k)$, at iteration k, two conditions are imposed: first, the point $x' \in M(k, \Delta_V)$ is chosen so that the distance in ℓ_∞ norm between x_k and x' is $\xi_k \Delta_V$. This distance is not based on the current mesh size Δ_k because the perturbation amplitude should only be linked to the value of ξ_k, as in the original VNS. This is the case as Δ_V is a parameter fixed by the user at the beginning of the algorithm. Secondly, in order to ensure that x' belongs to the current mesh $M(k, \Delta_k)$, the *shaking* procedure is triggered at iteration k when the VNS mesh size parameter is an integer multiple of the mesh size parameter Δ_k, i.e., there exists a nonnegative integer ℓ such that $\Delta_V = \ell \Delta_k$. Under the LTMADS mesh size update rule and the basic $2n$ directions

$D = [-I\ I]$, this condition is met as soon as $\Delta_k \leq \Delta_V$, and x' necessarily belongs to $M(k, \Delta_k)$ since $M(k, \Delta_V) = M(k, \ell\Delta_k) \subseteq M(k, \Delta_k)$. The choice of the VNS mesh size parameter Δ_V directly influences at which iterations a VNS SEARCH is performed.

Figure 3.4 shows two examples of meshes of sizes Δ_V and Δ_k with possible choices for a perturbation, with $n = 2$. The perturbation algorithm used in Section 3.5 is given in Figure 3.5.

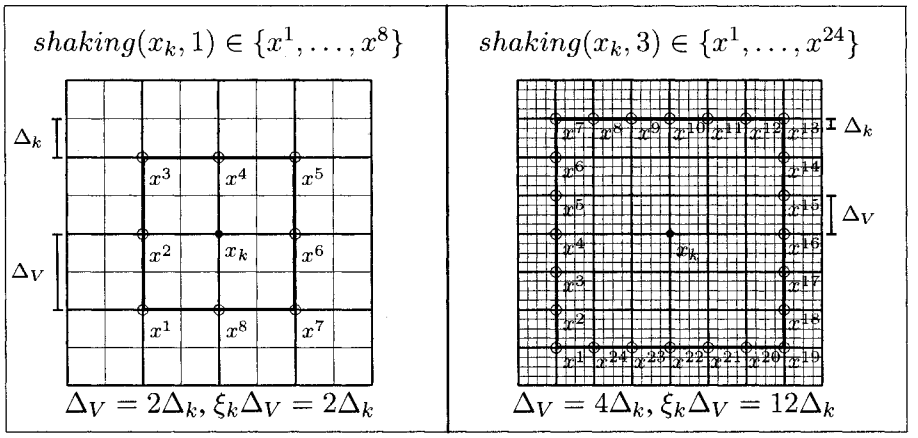

Figure 3.4: Two examples of meshes $M(k, \Delta_k)$ (gray), $M(k, \Delta_V)$ (black) and possible perturbation choices (points x^i on the bold frame at distance $\xi_k\Delta_V$ from x_k).

shaking (x_k, ξ_k)
  generate randomly $w \in \mathbb{Z}^n$ such that $\|w\|_\infty = \xi_k$
  for each variable index $i \in [1; n]$
    if lower bound L^i is finite and $(x_k^i + \Delta_V w^i < L^i)$
      $w^i \leftarrow \lceil (L^i - x_k^i)/\Delta_V \rceil$
    if upper bound U^i is finite and $(x_k^i + \Delta_V w^i > U^i)$
      $w^i \leftarrow \lfloor (U^i - x_k^i)/\Delta_V \rfloor$
  $x' \leftarrow x_k + \Delta_V w$

Figure 3.5: Practical implementation for the perturbation method; $L^i \in \mathbb{R} \cup \{-\infty\}$ and $U^i \in \mathbb{R} \cup \{+\infty\}$ respectively refer to the lower and upper bounds of variable i, $i \in [1; n]$.

A final remark concerning the choices of the amplitudes ξ_0 and ξ_{max} and of the VNS

increment δ is that they are chosen so that a perturbation of order 20 from a point at its lower bound generates a perturbed point on its upper bound. This value has been chosen as it is empirically good for VNS codes.

Since the POLL step is efficient in identifying mesh local optima, it is natural to use it for the VNS descent step. However the current mesh size cannot be reduced so that all the points evaluated during the descent step belong to the current mesh. The VNS descent terminates when the POLL fails on the current mesh size (this is similar to the extended POLL of [17]). It uses the MADS directions of the LTMADS implementation and its own mesh size parameter, called the descent mesh size. The initial value of the descent mesh size is taken to be the current mesh size at that point.

The descent works on its own filter for the constraints management (the descent filter), and is reset each time a new descent is performed. Each descent may be opportunistic (the evaluations are stopped at the first success) or complete (they are completed regardless of any earlier successes). The mesh update is made as in LTMADS, for the descent mesh size.

The LTMADS optimistic strategy is also used: if, during a POLL iteration of the descent, an improved point in the direction d is found, the next descent iteration will evaluate the functions at a point further along the direction d.

With this practical implementation for the perturbation and descent methods, the following pair of propositions ensures that the VNS SEARCH is valid as a SEARCH step of the MADS algorithm.

Proposition 3.4.1 *At iteration k, if the* VNS SEARCH *occurs, it generates trial points lying on the current mesh $M(k, \Delta_k)$.*

Proof. At iteration k, it has already been shown that the perturbed point x' lies on the current mesh, since $\exists \ell \in \mathbb{N}^+$ such that $\Delta_V = \ell \Delta_k$ (a pre-condition for the VNS SEARCH

to occur). By applying the rules of the MADS POLL for the descent, all the trial points will also belong to the mesh. Even if surrogates are used for the descent, the unique final evaluation will be made for a point on the mesh. ∎

Proposition 3.4.2 *At iteration k, if the* VNS SEARCH *occurs, it generates a finite number of trial points.*

Proof. Proposition 3.4.1 ensures that all the VNS SEARCH trial points are on the current mesh. Combining this with the assumption that all the trial points are in a compact set implies that their number is finite (see Proposition 3.4 in [18] for a more detailed proof). ∎

In practice, one may simply limit the number of different VNS trial points. For example, in the numerical tests of Section 3.5, a limit of 60 trial points is imposed.

3.5 Numerical Tests

This section describes numerical results for three different problems on which the algorithm was tested. The first one is a bound constrained analytic two variables problem, the second is a multidisciplinary optimization problem with ten variables and ten constraints, and the last one is an engineering problem from the chemical industry with eight variables and eleven black-box constraints. Surrogate functions are used in the last two problems. All source codes for these three problems are available on the web site www.gerad.ca/Charles.Audet.

3.5.1 Algorithm parameters and testing protocols

The MADS implementation used is the research version of the code NOMAD [15], with the following parameters: the POLL step uses the MADS $2n$ directions and is complete

(i.e., black-box functions are evaluated for all POLL trial points). The MADS dynamic ordering of the POLL directions is performed after each successful iteration (see [20] for details). Scaling of the variables is done in a way that the mesh size parameters Δ_0, Δ_{min}, and Δ_V are always presented as proportions of the variable ranges. For example, for a $n = 2$ problem with $L = [-1\ 0]$ and $U = [1\ 10]$, an indicated value of $\Delta_0 = 0.1$ corresponds in fact to the scaled vector $\Delta_0^{\text{scl}} = [0.2\ 1]$ (the same for Δ_{min} and Δ_V). Scaling can also be done directly into the black-box code.

The VNS mesh size parameter Δ_V is rounded so that the condition $\Delta_k = \ell\Delta_V$ is verified when $\Delta_k \leq \Delta_V$ so that the user does not need to compute the exact value of Δ_V compatible with some Δ_k). For example, without scaling, if the user chooses $\Delta_V = 0.001$ and $\Delta_0 = 1$, Δ_V will be rounded to $1/1024$, the closest integer power of 4.

The filter [19, 56] is used for open constraints with the squared ℓ_2 norm to define the constraint violation. The algorithm terminates when Δ_k drops below a parameter Δ_{min} or when a limit on the number of true evaluations is reached (n_{eval}^{max}). In the present work, we use n_{eval}^{max}=10000. These MADS parameters remain unchanged throughout the numerical tests, as it is not the point here to analyze the MADS-alone behavior.

The Latin Hypercube (LH [117,118]) SEARCH strategy is also used with two parameters $n_{init} = 100$ and $n_{iter} = 10$: n_{init} is the number of LH trial points generated at the first iteration of MADS, and n_{iter} the number of LH trial points generated at each subsequent iteration $k \geq 1$. The initial LH step is complete while the other steps are opportunistic (i.e., the iteration terminates immediately after a success).

An upper limit of 60 trial points is imposed for every VNS SEARCH. For the descent step of VNS, the evaluations are opportunistic and the directions used are the standard MADS $2n$ directions $\{\pm e_i : i = 1, 2, \ldots, n\}$ where e_i is the ith column of the identity matrix. When the premature descent stop technique (DS) is used, the parameter ρ of equation (3.3) is given. If available, surrogates are used in the descent step of VNS as described in Section 3.3.2, and not in the other MADS components.

Six algorithmic variants defined by different combinations of parameters are tested. The different algorithms configurations are detailed in Figure 3.6 and Table 3.1. For comparison purposes, the first two algorithms do not use the new features proposed in this paper: algorithm A uses only MADS without a SEARCH strategy and algorithm B combines MADS and Latin Hypercube (LH) SEARCH. All other algorithms use MADS and VNS, and differ only in their use of LH SEARCH, surrogates (Sgte = "yes" or "no") or DS strategy (ρ value or "no"). Latin Hypercube (LH) SEARCH always uses the parameters $n_{init} = 100$ and $n_{iter} = 10$.

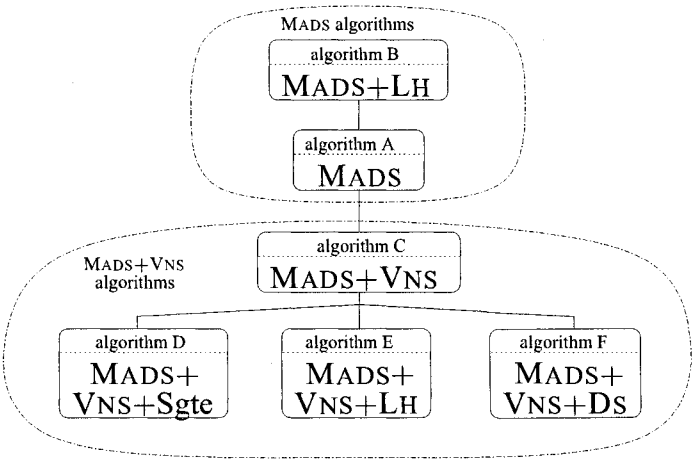

Figure 3.6: Schematic description of the 6 main types of algorithms. Algorithms C,D,E, and F are variations of the new algorithm proposed in this paper.

Because MADS directions are randomly chosen and two runs with the same parameters can give different results, 30 runs are made for each algorithm. Throughout the tests, algorithms D,E, and F, which use VNS and one additional feature (DS, LH or surrogates) can be mixed into other variants. These variants are denoted by D+E, D+F, and E+F (algorithm D+E uses MADS, VNS, surrogates, and LH, the same logic applies to D+F and E+F).

Results are summarized in Figures 3.8, 3.9, and 3.11 and are presented through 7 subgraphs. Each subgraph represents the objective function value (f) versus the number

Tableau 3.1: Detailed parameters description of the 6 main algorithms. The MADS parameters are default [15] parameters plus $\Delta_0 = 0.001$ or 0.05 and $\Delta_{min} = 10^{-12}$ or 10^{-7}.

algorithm	LH	VNS	Sgte	DS
A	no	no	no	no
B	$n_{init} = 100$ $n_{iter} = 10$	no	no	no
C	no	$\Delta_V =$ 0.01 or 0.05 or 0.1	no	no
D	no	$\Delta_V =$ 0.01 or 0.05 or 0.1	yes	no
E	$n_{init} = 100$ $n_{iter} = 10$	$\Delta_V =$ 0.01 or 0.05 or 0.1	no	no
F	no	$\Delta_V =$ 0.01 or 0.05 or 0.1	no	$\rho =$ 0.005 or 0.01 or 0.1

of black-box evaluations ($neval$) for the 30 runs of each series. One black-box evaluation is counted for the call of all the black-boxes defining the problem (objective and constraints). Note that when no surrogate is used, $neval$ denotes the number of true evaluations. With surrogates, another statistic (specific for each problem) must be used, taking also into account the surrogate cost. The last subgraph gives a summary of the 6 tests (one for each main algorithm) with average values. This larger subgraph allows one to compare directly the algorithms, in terms of quality of the solution and evaluations cost. Results for variants D+E, D+F, and E+F are not shown in the first 6 subgraphs but appear in the summary subgraph.

Finally, Tables 3.2, 3.4, and 3.7 present the numerical values of all the tests. Each row of these tables is dedicated to one algorithm, the first columns give the parameters used, and the other columns give average, best and worst values for f and $neval$ over the 30 runs. The runs above the horizontal line do not use the algorithmic features proposed in this document.

3.5.2 An analytic problem with many local optima

This problem is taken from [121],[problem 4]. It has 2 variables and the function to minimize is

$$\begin{aligned} f(a,b) = \quad & e^{\sin 50a} + \sin(60e^b) + \sin(70\sin a) + \sin\big(\sin(80b)\big) \\ & - \sin\big(10(a+b)\big) + \tfrac{1}{4}(a^2+b^2)\,. \end{aligned}$$

The closed bound constraints $-5 \leq a, b \leq 5$ are added since it can easily be shown that $f(a,b) \geq f(0,0)$ whenever (a,b) lies outside these bounds.

The graph of the objective function is shown in Figure 3.7. One may observe the numerous local optimal solutions. The plot on the top part of the figure shows the function on the entire domain, and the one on the bottom zooms in on the rectangle $-\frac{1}{4} \leq a, b \leq \frac{1}{4}$.

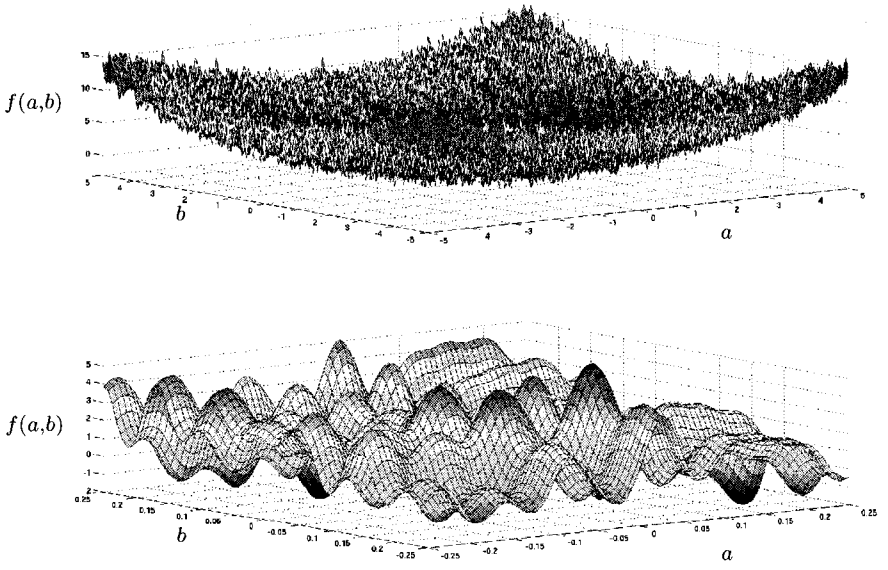

Figure 3.7: Graph of the analytic problem function with bounds $[-5;5]$ and $[-0.25;0.25]$.

Since the function is analytic and not costly, no surrogate function is used. Our first set of results used $x_0 = [0\ 0]^T$ as starting point. Unfortunately, this starting point is very close to the global optimal solution ($x^* \simeq [-0.024\ 0.211]^T$ with $f \simeq -3.307$), and all runs converged trivially to the global solution. Therefore, a new starting point far from the optimum was chosen: $x_0 = [3\ 3]^T$, for an objective function value of $f \simeq 4.721$. The initial mesh size parameter Δ_0 is set to be 0.05 times the variable ranges and Δ_{min} to 10^{-12} times the ranges. Two different values for Δ_V are used in the different runs: 0.01 and 0.05 times the variable range.

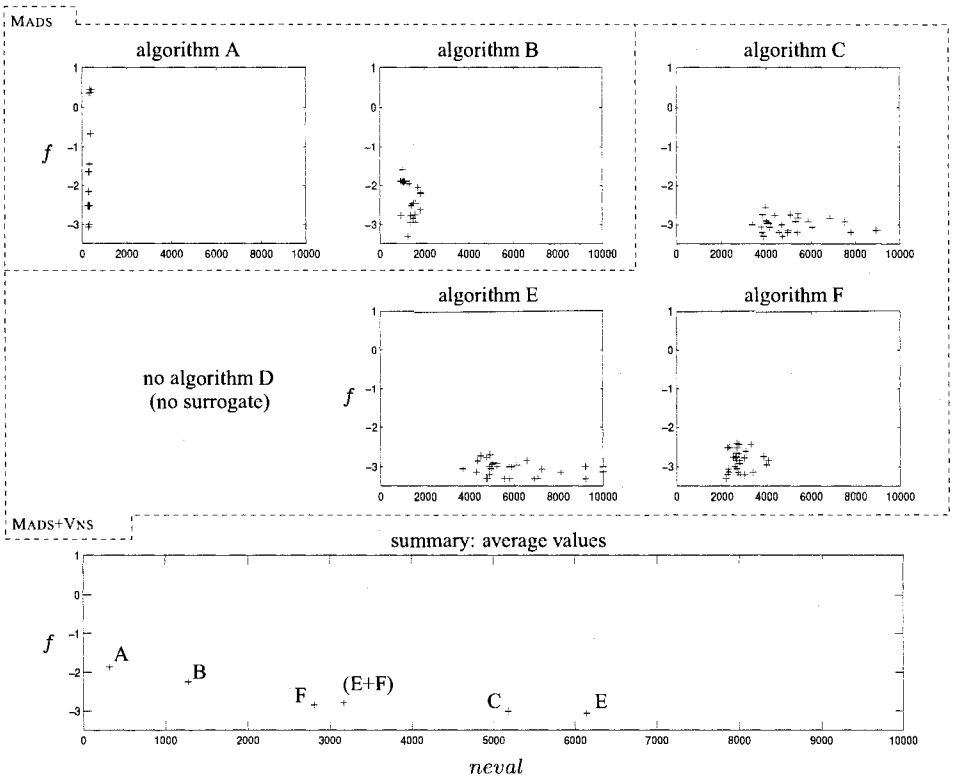

Figure 3.8: Analytic problem: f (objective function value) versus $neval$ (number of black-box evaluations).

Results are shown in Table 3.2 and Figure 3.8. The MADS alone and MADS with LH (algorithms A and B) performed well in terms of number of function evaluations, but did not often terminate near the global optimum. The VNS algorithms C and E (VNS and

VNS+LH) seem very appropriate in terms of global quality of the 30 runs (best average values for f). This quality came at the cost of additional evaluations. Tests using the DS strategy have fewer evaluations, but a lower quality of solution.

Tableau 3.2: Analytic problem: testing parameters and numerical results.

algorithm	parameters			average		objective (f)		*neval*	
	LH	VNS (Δ_V)	DS (ρ)	obj. (f)	*neval*	best	worst	best	worst
A	no	no	no	-1.865	321	-3.063	0.453	266	436
B	yes	no	no	-2.248	1283	-3.307	-1.596	916	1823
C	no	0.01	no	-3.009	5182	-3.307	-2.575	3399	9016
E	yes	0.01	no	-3.055	6146	-3.307	-2.705	3708	10000
F	no	0.01	0.01	-2.837	2809	-3.307	-2.413	2202	4088
E+F	yes	0.01	0.01	-2.778	3171	-3.307	-1.964	2401	4258

On a problem with a large number of local solutions, it is not surprising to see that the VNS SEARCH strategy is useful, since VNS is a metaheuristic designed to move away from local solutions.

We have also observed that the larger value reached by the ξ_k parameter is almost always below 10, for an allowed maximum of $\xi_{max} = 20$. This means that the outer loop on it in the original VNS algorithm (Figure 3.2), which is implicitly made in the update step of the new algorithm (Figure 3.3), is not useful for this problem. This remark holds also for the other two problems.

3.5.3 A MDO problem: aircraft range optimization

This problem is a Multidisciplinary Design Optimization (MDO) problem taken from [102, 113] from the mechanical engineering literature. Three coupled disciplines (structure, aerodynamics, and propulsion) are used to represent a simplified aircraft model, with 10 variables. The objective function is to maximize the aircraft range under bound constraints and 10 open constraints. The black-box performs an iterative fixed point method

through the different disciplines in order to compute the aircraft range. A natural surrogate consists in using a greater relative error as stopping criteria, and a smaller limit on the maximal number of fixed point iterations. The total number of fixed point iterations (from surrogates or true functions evaluations) is used instead of the number of black-box evaluations in the results ($fpit$).

The starting point x_0, the best known point x^* and the bounds are given in Table 3.3. The parameters Δ_0, Δ_{min}, and Δ_V are taken respectively as 0.001, 10^{-7}, and 0.1 times the variables ranges. Note that this problem can be solved more efficiently by other MDO-specialized optimization methods, as in [113], but the purpose of the results shown here is to make a comparison between the basic MADS algorithm and the coupling of MADS and VNS.

Results on Figure 3.9 and Table 3.4 show the same trend as the ones for the analytic problem: the use of the VNS improves the global quality of the solutions, at the cost of additional evaluations. Here, the runs that stand out as the best ones are for algorithms D and D+F (VNS with surrogates). The use of the DS strategy and surrogates in algorithm D+F gave excellent results by lowering the number of evaluations for almost the same quality of objective function value. For this problem the LH SEARCH seems not very appropriate, even when combined with the VNS SEARCH.

3.5.4 An engineering problem: styrene process optimization

The problem is to optimize is a styrene production process simulation. Optimization of chemical processes simulation using an outside optimizer has previously been studied in [11,32,52]. The styrene production process is divided into four steps: reactants preparation (pressure rise and evaporation), catalytic reactions (as in [112]), styrene recovery (first distillation), and benzene recovery (second distillation). There is also an important recycling of unreacted ethylbenzene. All these steps appear in Figure 3.10.

A chemical process simulator was developed based on the SMS (Sequential Modular

Tableau 3.3: MDO problem: starting point (x_0), best known point (x^*), and bounds ($L, U \in \mathbb{R}^n$).

	x_0	x^*	L	U
	0.25	0.40	0.10	0.40
	1	0.75	0.75	1.25
	1	0.75	0.75	1.25
	0.5	0.156244	0.1	1.0
	0.05	0.06	0.01	0.09
	45000	60000	30000	60000
	1.6	1.4	1.4	1.8
	5.5	2.5	2.5	8.5
	55	70	40	70
	1000	1500	500	1500
f	-535.82 (infeasible)	-3964.20		

Simulation) paradigm. SMS is a widely used approach [50, 103, 111], mostly because it relies on fast iterative methods. For some given operating parameters, each block can compute its output only when its input has been calculated. The main deficiency is recycling loops: the first blocks of the loop require the evaluation of the lasts ones.

The black-box simulator uses some common methods such as Runge-Kutta, Newton, Wegstein (fixed point), secant, bisection, and many other chemical engineering related solvers. The objective is to maximize the NPV (Net Present Value) of the styrene production process project, while satisfying industrial and environmental regulations. This is given by

$$NPV = \sum_{i=0}^{n} \frac{(S_i - C_i)(1 - T_a) - I_i + D_i}{(1 + T_r)^i}$$

where the index i denotes a year between 0 and n, S_i the sales, C_i the operating costs, T_a the income tax rate, I_i the investment, D_i is the depreciation, and T_r is the actualization rate. The source of difficulty in the optimization problem is that S_i, C_i, I_i, and D_i are functions of the simulator output. This output contains information such as equipment sizing, flow rates and compositions, units efficiencies, power needs, and so on. The sim-

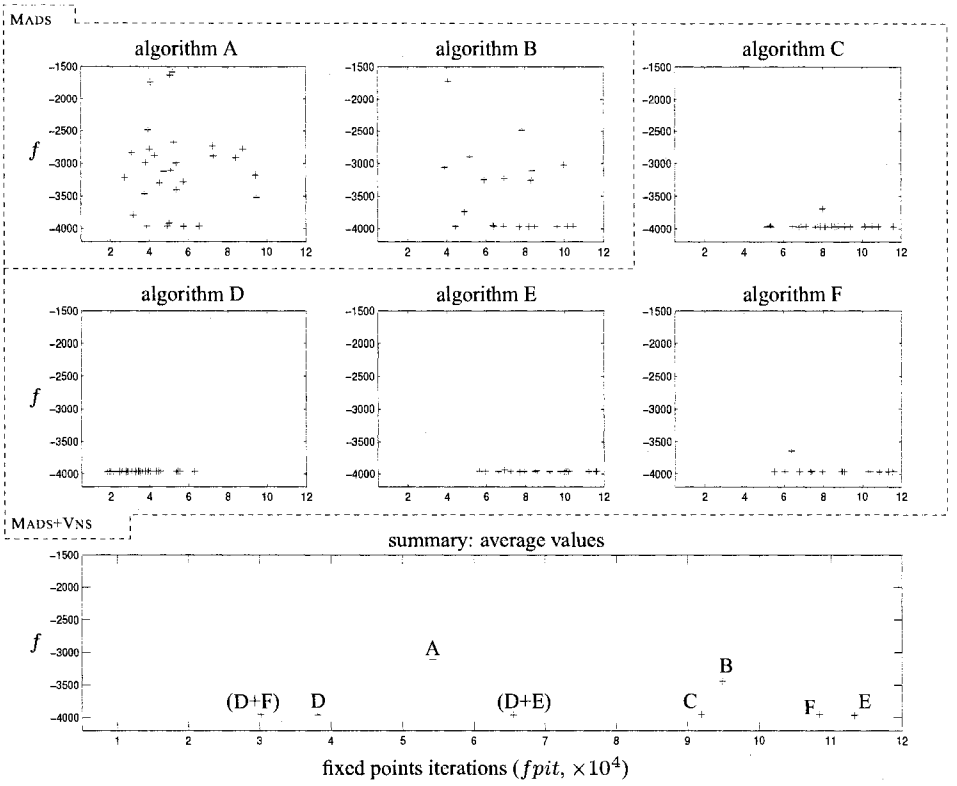

Figure 3.9: MDO problem: f (objective function value) versus $fpit$ (number of fixed point iterations).

ulation of the process must be successfully performed before the constraint and objective functions are evaluated. Tables 3.5 and 3.6 give additional information on the problem characteristics.

The surrogate is obtained by using greater tolerance values and smaller maximum numbers of iterations in the various numerical methods. As opposed to the MDO problem of the previous subsection, computational comparison between true and surrogate functions is not trivial. Intensive tests suggest that one true evaluation has approximatively the same cpu-time of three surrogate evaluations. For the results in Figure 3.11 and Table 3.7, the "$neval$" value is then an approximation of the evaluations cost, obtained by $neval$ defined to be the sum of number of true evaluations, plus a third of the number of surrogate evaluations. Because of this, $neval$ exceeds in some cases the

Tableau 3.4: MDO problem: testing parameters and numerical results.

algo.	parameters LH	VNS (Δ_V)	DS (ρ)	Sgte	average obj. (f)	$fpit$	objective (f) best	worst	$fpit$ best	worst
A	no	no	no	no	-3101.39	54253	-3964.20	-1588.35	27273	94815
B	yes	no	no	no	-3443.09	94881	-3964.20	-1355.66	18328	201797
C	no	0.1	no	no	-3954.30	91963	-3964.20	-3685.85	51513	152713
D	no	0.1	no	yes	-3964.16	38193	-3964.20	-3964.03	18339	125261
E	yes	0.1	no	no	-3962.66	113342	-3964.20	-3937.25	56589	165063
F	no	0.1	0.25	no	-3952.61	108393	-3964.20	-3640.72	55429	175015
D+E	yes	0.1	no	yes	-3961.38	65586	-3964.20	-3881.93	26220	148164
D+F	no	0.1	0.1	yes	-3950.60	30200	-3964.20	-3657.49	9984	69839

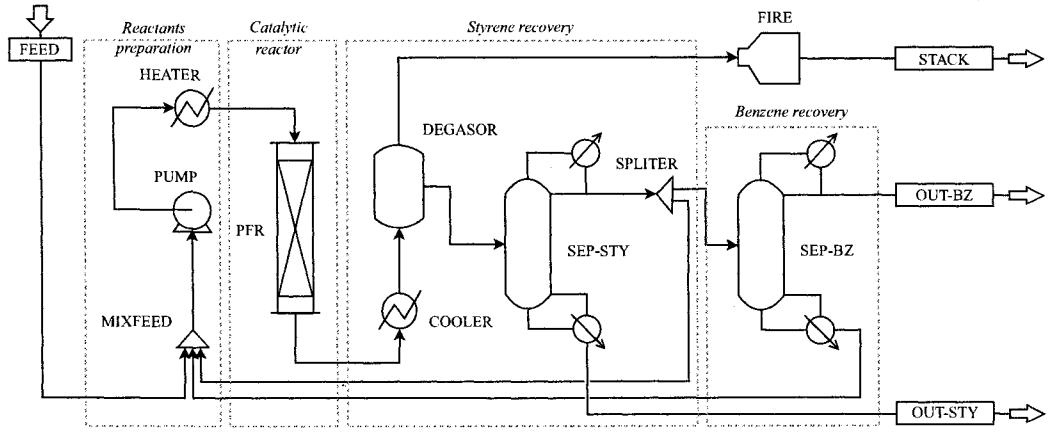

Figure 3.10: Flowsheet of the styrene production process.

upper bound of 10000. The scaling is this time done directly in the black-box code with new unified bounds of [0 1]. The fixed parameters used for these series of 30 tests are $[\Delta_0\ \Delta_{min}\ \Delta_V]^T = [0.001\ 10^{-7}\ 0.05]^T$.

Figure 3.11 and Table 3.7 do not suggest a clear domination of one VNS strategy over all others. However, one may observe that algorithms C and E using VNS are the more stable runs in the sense that they very often produce a good solution with only a few outliers (this is mostly apparent in the plots of Figure 3.11). Runs using MADS and MADS with LH SEARCH could not reach the best objective function values. VNS systematically generated the best solutions.

Tableau 3.5: Styrene problem: variables and objective description, starting point (x_0), best point found (x^*), and bounds ($L, U \in \mathbb{R}^n$).

description	units	x_0	x^*	L	U
outlet temperature in block HEATER	K	870	1100	600	1100
length of reactor (block PFR)	m	13.88	16.9836	2	20
light key fraction in block SEP-STY	–	0.086014	0.0968282	10^{-4}	0.1
light key fraction in block SEP-BZ	–	0.008092	0.0001	10^{-4}	0.1
outlet pressure of block PUMP	atm	7.22	2	2	20
split fraction in block SPLITER	–	0.2599	0.224742	0.01	0.5
air excess fraction in block FIRE	–	1.668	1.96261	0.1	5
cooling temperature of block COOLER	K	330	403.025	300	500
$f = -NPV$	-$	-10942600	-33539100		

3.6 Discussion

This paper proposes a generic way to incorporate the VNS metaheuristic into the SEARCH step of the MADS algorithm. Notice that a similar combination was recently detailed in [123] where a generic particle swarm GPS SEARCH strategy is defined.

Our proposed algorithm belongs to the general MADS framework, and thus preserves all of its convergence properties. The algorithm remains simple, with only two additional parameters: the VNS mesh size parameter Δ_V used in the shaking, and the optional descent stopping parameter ρ. In the numerical results presented we either used Δ_V to be one tenth, one twentieth or one hundredth of the range of the variables. In our software packages we will use one tenth as the default value. We observed that the algorithm was more sensitive to the descent parameter ρ, and will not attempt to suggest default values.

The algorithm was applied to three problems and compared to classic MADS with or without the classic Latin Hypercube (LH) SEARCH strategy. Good results were obtained in terms of quality: the random aspect of the MADS directions is attenuated, leading to more stable solutions. This improvement in terms of quality generally comes at the price of a higher number of black-box evaluations, but VNS seems to use these additional

Tableau 3.6: Styrene problem: constraints description, with partition into three groups.

group	j	description of constraint c_j
simulator	1	true if the simulation has succeed
process	2	true if column SEP-STY is structurally acceptable
	3	true if column SEP-BZ is structurally acceptable
	4	true if mixture in FIRE can burn and if environmental regulations on CO and NO_x are met
	5	minimal purity of produced styrene
	6	minimal purity of produced benzene
	7	minimal overall ethylbenzene conversion into styrene
economics	8	maximal payout time
	9	minimal discounted cashflow rate of return
	10	maximal total investment
	11	maximal annual equivalent cost

evaluations more efficiently, compared to other methods such as LH SEARCH.

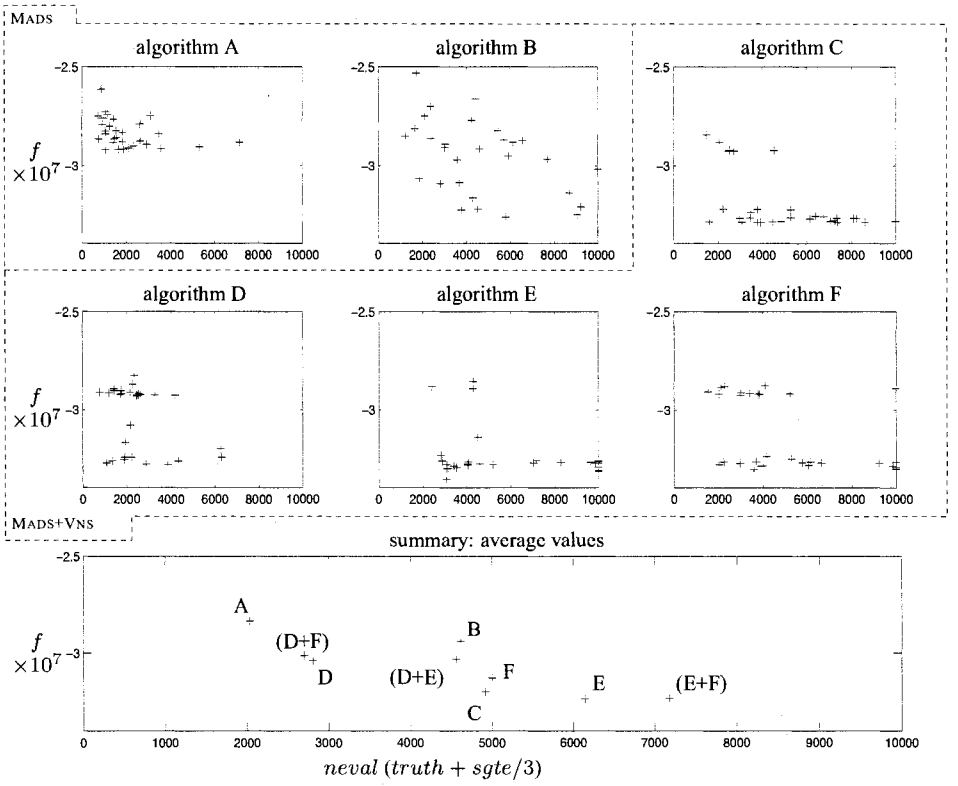

Figure 3.11: Styrene problem: f (objective function value) versus $neval$ = true evaluations + surrogate evaluations/3.

Tableau 3.7: Styrene problem: testing parameters and numerical results.

algo.	parameters LH	VNS (Δ_V)	DS (ρ)	Sgte	average obj. (f)	$neval$	objective (f) best	worst	$neval$ best	worst
A	no	no	no	no	-28334450	2036	-29189900	-26104800	722	7159
B	yes	no	no	no	-29397777	4612	-32650700	-24822000	1236	10000
C	no	0.05	no	no	-31974893	4919	-32932200	-28417900	1503	10000
D	no	0.05	no	yes	-30382607	2811	-32783500	-28266000	775	11193
E	yes	0.05	no	no	-32339227	6145	-33539100	-28566200	2410	10000
F	no	0.05	0.005	no	-31278540	5001	-33046500	-28765300	1538	10000
D+E	yes	0.05	no	yes	-30302470	4562	-32881500	-26903100	911	10681
D+F	no	0.05	0.005	yes	-30132383	2703	-32793100	-26222400	783	11280
E+F	yes	0.05	0.005	no	-32304503	7184	-33063200	-29173800	1898	10000

CHAPITRE 4

PARALLEL SPACE DECOMPOSITION OF THE MESH ADAPTIVE DIRECT SEARCH ALGORITHM[1]

Charles Audet[2] J.E. Dennis Jr.[3] Sébastien Le Digabel[4]

October 2007

Abstract

This paper describes a Parallel Space Decomposition (PSD) technique for the Mesh Adaptive Direct Search (MADS) algorithm. MADS extends Generalized Pattern Search for constrained nonsmooth optimization problems. The objective here is to solve larger problems more efficiently. The new method (PSD-MADS) is an asynchronous parallel algorithm in which the processes solve problems over subsets of variables. The convergence analysis based on the Clarke calculus is essentially the same as for the MADS algorithm. A practical implementation is described and some numerical results on problems with up to 500 variables illustrate advantages and limitations of PSD-MADS.

[1]Work of the first author was supported by FQRNT and NSERC. Work of the first and second authors was supported by AFOSR, the Boeing Company, and ExxonMobil Upstream Research Company. Work of the first and third authors was supported by the Consortium for Research and Innovation in Aerospace in Québec.

[2]GERAD and Département de mathématiques et de Génie Industriel, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montréal (Québec), H3C 3A7 Canada, www.gerad.ca/Charles.Audet, Charles.Audet@gerad.ca.

[3]Computational and Applied Mathematics Department, Rice University, 8419 42nd Ave SW, Seattle, WA 98136 http://www.caam.rice.edu/~dennis, dennis@rice.edu.

[4]GERAD and Département de mathématiques et de génie industriel, École Polytechnique de Montréal, Sebastien.Le.Digabel@gerad.ca.

Keywords: Parallel Space Decomposition, Mesh Adaptive Direct Search, Asynchronous parallel algorithm, Nonsmooth optimization, Convergence analysis.

4.1 Introduction

The paper considers optimization problems of the form

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

where the objective function $f : \Omega \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$. The feasible region Ω is assumed to satisfy a nonsmooth constraint qualification, and we only assume the presence of an oracle to tell whether or not a given $x \in \mathbb{R}^n$ is feasible. We are concerned primarily with cases where $f(x)$ or the oracle are given by black-box computer simulations, which are assumed to evaluate in finite time. This is common in engineering design. Indeed, the reason we allow $f(x)$ to take on the value ∞ is that for many such problems, no value of $f(x)$ is returned, even for some $x \in \Omega$, because of the internal workings of the simulation used to drive the design. See [3, 8, 23, 27, 58, 71, 79, 96].

There are other useful derivative-free direct search methods designed for problems similar to $\mathcal{P}$. These include the Nelder-Mead simplex [101], the DIRECT algorithm [54, 62, 77], frame based methods [40, 106], the Generalized Pattern Search (GPS) [18, 29, 120], the Asynchronous Parallel Pattern Search (APPS) approach [64, 74, 80, 83, 84], and the Mesh Adaptive Direct Search (MADS) [4, 20]. Related is the implicit filter method [78], though it does use a coarse difference gradient approximation. The reader may consult [78, 81, 89] for a survey of some of these direct search methods.

Using these methods to solve expensive problems with more than a few dozen variables may be impractical since they may need a large number of costly black-box evaluations. One possible approach is to use parallelization. Dennis and Wu [49] reviewed different parallel methods for continuous optimization and concluded that a combination of

GPS and the Parallel Variable Distribution (PVD) of Ferris and Mangasarian [53] should be considered.

PVD is an evolution of the block-Jacobi technique of [25] which optimizes in parallel a series of reduced subproblems on subspaces of the original variables of $\mathcal{P}$. The present paper is based on the remark of Dennis and Wu. Dennis and Torczon [47] described a first parallel version of GPS, which evaluates the black-box evaluations in parallel and synchronizes at each iteration to compare solutions and update the current iterates. The Asynchronous Parallel Pattern Search, APPS [64, 80], removes this synchronization barrier. In APPS, each process explores the space of variables using its own set of directions and does not wait for the other processes to terminate. APPS is expected to be more efficient than the synchronous version of [47], especially if the black-boxes have heterogeneous behavior that depends on the point where they are evaluated. A convergence analysis is presented in [84] for the smooth case.

Our work applies a decomposition of the variables of $\mathcal{P}$ based on the block-Jacobi technique of [25] that inspired the PVD method of [53]. This allows a natural parallel application of MADS to smaller subproblems, in an asynchronous way. The new algorithm, called PSD-MADS, can be interpreted as a particular instance of MADS, thus inheriting the main results of the MADS convergence analysis.

The paper is divided as follows: Section 4.2 gives an overview of the Parallel Space Decomposition and MADS methods. Section 4.3 presents the new asynchronous parallel algorithm, PSD-MADS, and Section 4.4 shows that the main convergence results of MADS are maintained by showing that the entire PSD-MADS algorithm may be interpreted as a specific MADS instance. An implementation of PSD-MADS is described in Section 4.5, with some numerical results on problems with a number of variables ranging from 20 to 500. Finally, Section 4.6 gives some conclusions and proposes possible extensions of PSD-MADS.

4.2 Relevant literature

This section presents an overview of parallel space decomposition methods, and the Mesh Adaptive Direct Search algorithm. MADS, its convergence analysis and its LT-MADS implementation are described in detail, since Sections 4.4 and 4.5 depend on them.

4.2.1 Parallel space decomposition methods

Parallel space decomposition methods decompose $\mathcal{P}$ into a finite number of smaller dimension subproblems, which can be solved in parallel with one process assigned to each subproblem.

Define $N = \{1, 2, ..., n\}$ where n is the number of variables of the optimization problem $\mathcal{P}$, and $Q = \{1, 2, ..., q\}$ where q is the number of available processes. Each process $p \in Q$ works on a nonempty subset $N_p \subseteq N$ of the variables. The other variables are fixed, based on the incumbent solution $x^* \in \Omega$, the current best known solution. More precisely, process $p \in Q$ works on the optimization subproblem

$$\min_{x \in \Omega_p(x^*)} f(x) \qquad (\mathcal{P}_p(x^*))$$

with $\Omega_p(x^*) = \left\{x \in \Omega : x_i = x_i^* \ \forall i \in \overline{N}_p\right\}$ and $\overline{N}_p = N \setminus N_p$. The subproblem $\mathcal{P}_p(x^*)$ contains $n_p = |N_p|$ free variables, indexed by N_p. In Section 4.5 we propose a strategy to build the subsets N_p.

The block-Jacobi method in [25] is an iterative two-steps algorithm and may be described in a very general way as follows. At each iteration, the first step, the *parallelization*, consists of solving the subproblems in parallel, and the second step, the *synchronization*, gathers the subproblems solutions and constructs the next iterate. Similar methods are described in [66, 94, 122].

A variant of the method was introduced by Ferris and Mangasarian [53], as the Parallel Variable Distribution (PVD) for a differentiable objective function f with continuous partial derivatives. In order to solve the subproblems more efficiently, the PVD method allows *a priori* fixed variables to change in a limited fashion, along directions typically based on ∇f. These variables are denoted as "forget-me-not" terms.

The convergence analysis in [53] requires that subproblems be solved to optimality. In the unconstrained case, if ∇f exists and is Lipschitz, then the accumulation points of the generated sequences are stationary points. In addition, if f is assumed to be convex, the convergence rate is shown to be linear. When Ω is nonempty, closed, convex, block-separable, and the functions defining it are also continuously differentiable, convergence results are still available. When there are general constraints, Ferris and Mangasarian recommend transforming the problem into unconstrained problems via penalty functions or exploiting possible block-separable constraints.

These are parallel synchronous algorithms because the synchronization step waits for all the processes to end. The conclusion of [53] is that an asynchronous version of the algorithm would increase efficiency. This is done in [90] for unconstrained problems, where the synchronization step is dropped at the expense of the convergence analysis.

Extensions of the PVD method are given in [109, 115, 116] with similar convergence results to those in [53] under less restrictive conditions. For example, subproblems do not need to be solved to complete optimality, as for example when one Newton-like iteration is used. A convergence analysis for the constrained case is given with either block separability or convexity assumptions on the structure of Ω. These are not reasonable assumptions for our target class of engineering design problems.

In the above references, no practical and generic strategy is given concerning the choice of the subproblems variables (sets N_p). However, the sets do need to form a partition of N, and they are fixed throughout the entire process. In the parallel space decomposition [60] the subspaces can be chosen differently at each iteration.

In Fukushima [61], the PVD method is extended to a more general framework for unconstrained problems. The sets of subproblems variables are not fixed through the iterations, are not required to form a partition of N, but they must span N. In particular, an overlapping of the subproblems variables is allowed. Some experiments with such methods are given in [124].

More recently, the MoVars algorithm [26] combines the GPS method with the synchronous PVD framework (including the "forget-me-not" terms from [53]) on fixed subsets N_p, but there is no convergence analysis.

In most of the references of this section, f is assumed to be at least differentiable, and constraints, if they are considered, are block-separable or convex. These assumptions are not reasonable for the problems of interest to us, and thus, our convergence analysis does not rely on the analysis of [53] or its extensions. Rather, by incorporating MADS with its weaker hypotheses, we will inherit the MADS convergence analysis. It will also give us greater flexibility concerning the choice of the subsets N_p, the way we handle constraints, the amount of work we must devote to the subproblems, and the lack of necessity for a synchronization step.

4.2.2 Mesh Adaptive Direct Search (MADS)

We now summarize the MADS algorithm [20] for problem $\mathcal{P}$, which extends the Generalized Pattern Search (GPS) algorithm for linearly constrained optimization [29, 120].

The constraints defining Ω are handled by the extreme barrier approach, as in [20, 86, 87]. This means that trial points outside Ω are simply rejected by setting their objective function value to ∞. Of course, this requires that the user provides a feasible initial point $x_0 \in \Omega$. We make the standard assumption that all the trial points generated by the algorithm lie in a compact set.

MADS is an iterative algorithm where the black-box functions are evaluated at some trial

points that are either accepted as new iterates because they are feasible and decrease the objective, or they are rejected.

All trial points generated by these algorithms are constructed to lie on a mesh

$$M(\Delta) = \bigcup_{x \in V} \{x + \Delta D z : z \in \mathbb{N}^{n_D}\} \subset \mathbb{R}^n \tag{4.1}$$

where the set V, called the *cache*, is a data structure memorizing all previously evaluated points so that no double evaluations occur, $\Delta \in \mathbb{R}^+$ represents a mesh size parameter, and D is a $n \times n_D$ matrix representing a fixed finite set of n_D directions in $\mathbb{R}^n$. More precisely, D is called the set of mesh directions and is chosen so that $D = GZ$, where G is a non-singular $n \times n$ matrix, and Z a $n \times n_D$ integer matrix. The definition given by (4.1) differs slightly from the one in [20]. There the mesh was indexed by the iteration number instead of being parameterized by Δ. The reason for this difference is that our parallel algorithm will be working simultaneously on different size meshes originally generated at different iterations. Note also that in order to simplify the notation, the mesh size parameter Δ used here is the equivalent of Δ^m in [20].

Each iteration is divided into three steps, the SEARCH, the POLL, and an update step determining the success of the iteration and producing the next iterate. The SEARCH and POLL are treated specially in that the POLL needs not be carried out at an iteration if the SEARCH finds a better point. At each iteration, the algorithm attempts to generate an improved incumbent solution on the current mesh $M(\Delta_k)$, where Δ_k is the mesh size parameter at iteration k. The SEARCH step is very flexible and allows for trial points anywhere on the mesh. The way of generating these points is free of any rules, as long as they remain on the current mesh $M(\Delta_k)$ and that the SEARCH terminates in finite time. Some SEARCH strategies can be tailored for a specific application, while others are generic, such as the use of Latin Hypercube sampling [118], or Variable Neighborhood Search [12]. In summary, if one wants to define a new MADS algorithm with its specific

SEARCH, all that needs to be done to ensure convergence is to show that the SEARCH requires finite time and generates a finite number of trial points lying on the mesh.

The POLL step explores the mesh $M(\Delta_k)$ near the current iterate x_k and its rules ensure theoretical convergence of the algorithm. The way of choosing the directions used to generate the POLL points is the difference between GPS and MADS. In GPS, the normalized set of potential POLL directions must be chosen from a finite set that is fixed across all iterations. In MADS, the directions may be chosen to be asymptotically dense in the unit sphere, which allows better coverage. We use the terminology of [40, 106] and say that at iteration k, the set of trial POLL points is called the frame P_k. The set of directions used to construct P_k is denoted D_k, and it is not a subset of D.

In the last step of the kth iteration, the mesh size parameter is updated according to $\Delta_{k+1} \leftarrow \tau^{\omega_k}\Delta_k$, where $\tau > 1$ is a fixed rational number and ω_k an integer that depends on the success of the iteration. When no improvement is made, the iteration is said to fail, and ω_k is taken to be an integer in the interval $[\omega^-; -1]$ with $\omega^- \leq -1$, forcing the next trial POLL points to be closer to the current iterate. When a new best iterate is found, the iteration is said to succeed, and Δ_k is possibly increased with ω_k in $[0; \omega^+]$, with the integer $\omega^+ \geq 0$. In the LTMADS implementation of [20], τ is fixed to 4, $\omega^- = -1$, and $\omega^+ = 1$.

A high level description of the algorithm is summarized in Figure 4.1. We encourage the reader to consult [20] for a complete description.

4.2.3 MADS convergence analysis

We will summarize the main convergence results for MADS given in [20]. These results assume that constraints are treated by the extreme barrier approach, and they constitute a hierarchical series of results relying on the Clarke calculus [35] for nonsmooth functions. Cone definitions are also given in [35].

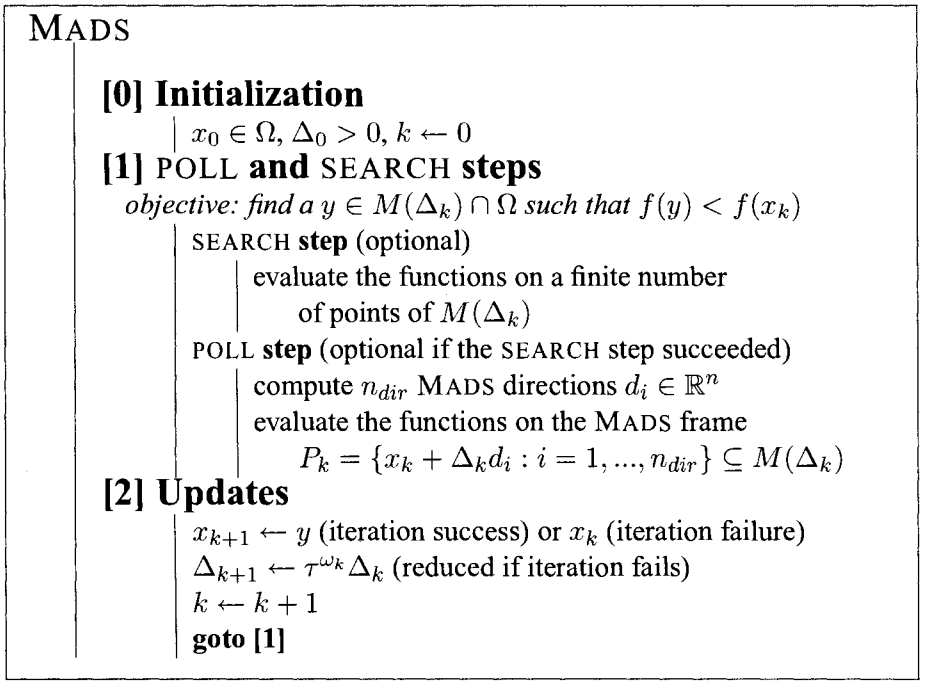

Figure 4.1: High level description of the MADS Algorithm. The directions d_i are positive integer combinations of the columns of D. The SEARCH or POLL steps can be stopped before all evaluations are terminated (opportunistic strategy).

The main theorem is that under a local Lipschitz assumption on f, and under the assumption that the set of all normalized POLL directions is dense in the unit sphere, the algorithm produces a Clarke stationary point. More precisely, MADS generates a point $\hat{x} \in \Omega$ at which the Clarke generalized directional derivatives of f in all the directions in the Clarke tangent cone at $\hat{x}$ are nonnegative. The only assumptions needed are that f is Lipschitz near $\hat{x}$ and the constraint qualification that the hypertangent cone of Ω at $\hat{x}$ is nonempty. A corollary to this result in the unconstrained case is that if f is strictly differentiable near $\hat{x}$, then $\nabla f(\hat{x}) = 0$.

The convergence result that requires the least assumptions on f and Ω, the zero'th order result, is that MADS generates a limit point $\hat{x}$, which is the limit of mesh local minimizers on meshes that get infinitely fine. The notion of local optimality is with respect to the current POLL set, defined using a positive spanning set of directions. More formally, MADS generates a convergent subsequence of iterates $\{x_k\}_{k \in K} \subset \Omega$ such that $x_k \rightarrow \hat{x}$, and $f(x_k) \leq f(x_k + \Delta_k d_k)$ for all directions d_k in a positive spanning set D_K, and $\|\Delta_k d_k\| \rightarrow 0$.

The price to pay for our new capability to handle a large number of variables is that this last convergence result will be lost. We will consider a MADS algorithm whose POLL set contains a single element instead of being built using a positive spanning set of directions. We will refer to this as a *single*-POLL MADS algorithm, and it still retains the property of generating asymptotically dense polling directions.

The next section discusses the LTMADS implementation of the MADS algorithm. LTMADS uses positive bases to construct the POLL sets. It is stated that the union of theses normalized directions forms a dense set because if one looks closely at the proof in [20], one sees that it is the subset of single-POLL normalized MADS directions that grows dense in the unit sphere. Thus, with the assumption of local Lipschitz continuity, the main convergence result guaranteeing a Clarke stationary point holds.

4.2.4 The LTMADS implementation of MADS

The SEARCH and POLL steps need to satisfy certain conditions for the convergence results to hold. In particular, one of these conditions is that the total set of normalized POLL directions used by the algorithm be dense in the unit sphere. In [20] there is a practical way of accomplishing this through the LTMADS implementation of MADS.

LTMADS fixes τ to 4 and the set of mesh directions $D = [-I_n \; I_n]$ where I_n represents the $n \times n$ identity matrix. The mesh is based on the nonnegative integer value $\ell = -\log_4(\Delta_k)$, $\Delta_k = 4^{-\ell}$, and directions are constructed randomly using a lower triangular matrix. One of these directions is a special case and fixed for each value of ℓ. This direction, called $b(\ell)$, has one coordinate (the largest in absolute value) set to $\pm 2^\ell$ so that POLL points are within $\sqrt{\Delta_k}$ of the POLL center x_k in the ℓ_∞ norm.

The result stated in [16, 20] is that with probability one, the series of normalized directions $b(\ell)$ grows dense in the unit sphere. In LTMADS, the direction $b(\ell)$ is augmented at each iteration with other directions to form a positive spanning set of polling directions. We will, as explained in the preceding section, construct a single-POLL MADS algorithm with dense polling directions using only the $b(\ell)$ directions, but the zero'th order convergence result of MADS is lost. Also, because we are not polling at each iteration in a positive spanning set of directions, the mesh size might drop too quickly with this single-POLL version of MADS, and so the SEARCH step is of extra importance. This is the key to the PSD-MADS algorithm described in the next section: one process executes a single-POLL MADS algorithm, while the work of the other processes may be interpreted as a SEARCH step.

4.3 Parallel Space Decomposition of MADS

This section describes the combination of MADS with a parallel space decomposition method. The resulting algorithm is called PSD-MADS. It is an asynchronous parallel algorithm where a master process decides on the subsets $N_p \subseteq N$ and assigns the resulting optimization subproblems $\mathcal{P}_p(x^*)$ to slaves. The slaves apply MADS to attempt to improve the incumbent solution x^*. No synchronization step is performed. When a slave completes its assigned task, the master assigns a new subproblem with a possibly new N_p and x^*.

4.3.1 General description of PSD-MADS

While PSD-MADS is an asynchronous parallel algorithm, the notion of iteration is kept, and corresponds to two successive calls by the master to one special slave, called the *pollster* slave, described more precisely in Section 4.3.2. The pollster slave executes a single-POLL MADS algorithm on the entire problem $\mathcal{P}$, while the other slaves, called the *regular* slaves, work on the subproblems $\mathcal{P}_p(x^*)$.

Each subproblem $\mathcal{P}_p(x^*)$ is a subproblem of $\mathcal{P}$ with a reduced number of variables indexed by the set N_p. When an optimization process terminates, the slave communicates its progress to the master. If it has found an improved solution, then that becomes the new incumbent solution. The slave immediately starts work on a new subproblem assigned by the master. There is no need to synchronize all the slaves.

With several MADS instances executing in parallel, it is necessary to define different mesh size parameters: first, Δ_j^p corresponds to the mesh $M(\Delta_j^p)$ used at iteration j of the MADS algorithm performed by a regular slave s_p. This mesh size parameter is denoted differently for the pollster slave, with Δ_k^1 (notice the same iteration counter k used both for the pollster slave and PSD-MADS). Δ_k^1 is called the *pollster mesh size parameter* at iteration k of PSD-MADS. Finally, an additional mesh size parameter, Δ_k^M, is called the

master mesh size parameter. The mesh $M(\Delta_k^M)$ is never used explicitly, but it is useful for comparing the two other meshes $M(\Delta_k^1)$ and $M(\Delta_j^p)$. At iteration k of PSD-MADS, and at iteration j of the MADS algorithm performed on a subproblem $\mathcal{P}_p(x^*)$ by a regular slave s_p for $p \in \{2, 3, \ldots, q-2\}$, the PSD-MADS construction ensures that

$$\Delta_k^1 \leq \Delta_k^M \leq \Delta_j^p \quad . \tag{4.2}$$

Inequalities (4.2) are formally proved in the convergence analysis of Section 4.4, where PSD-MADS is interpreted as a valid single-POLL MADS instance performed by the pollster slave. An additional hypothesis on the different meshes $M(\Delta_k^M)$, $M(\Delta_k^1)$, and $M(\Delta_j^p)$ is necessary:

Hypothesis 4.3.1 *If two mesh size parameters Δ and Δ' satisfy $\Delta = \tau^\omega \Delta'$ where $\omega \in \mathbb{N}$, then $M(\Delta) \subseteq M(\Delta')$.*

This assumption holds for the PSD-MADS implementation given in Section 4.5.

The q processes are partitioned into a master, $q-2$ slaves, and a cache server (process number $q-1$), which memorizes all points that have been evaluated. The $q-2$ slaves include the pollster slave (process number 1) and $q-3$ regular slaves. The notation s_p with $p \in Q \setminus \{q-1, q\}$ is used to identify the $q-2$ processes assigned as slaves, and $Q_{reg} = \{2, 3, ..., q-2\}$ is the set of indices of the $q-3$ regular slaves. The qth process is used as the master, which defines the lower dimensional subproblems $\mathcal{P}_p(x^*)$ and communicates them to the slaves.

An advantage of applying the parallel space decomposition method to MADS instead of another optimization method is that most of the conditions necessary for convergence in the other parallel space decomposition methods mentioned in Section 4.2.1 can be relaxed:

- f and the functions defining Ω need not be differentiable, and there are no con-

ditions on the constraints other than requiring a feasible initial point $x_0 \in \Omega$, and a nonempty hypertangent cone at a limit point. Constraints are easily handled by the extreme barrier approach;

- There is no synchronization step [53]. Each process takes the current best solution as its starting point without waiting for another process to terminate;
- The choice of sets N_p, $p \in Q_{reg} \cup \{1\}$, is completely flexible and dynamic. Moreover, sets N_p do not have to define a partition of the variables, and some variables can belong to more than one set. An example for a practical strategy deciding the sets N_p is given in Section 4.5.

This new algorithm is not a particular case of the method in [61], which generalizes many parallel variable decomposition methods, since general constraints are allowed, and f is not assumed to be smooth. PSD-MADS is also different from the recent MoVars algorithm [26], which does require N_p to partition the variables, because it provides a convergence analysis, dynamically changes the sets N_p, and it is an asynchronous parallel method. The next sections describe precisely the role of each process.

4.3.2 The pollster slave s_1, on $M(\Delta_k^1)$

The pollster slave s_1 has a special role; its set of variables is always fixed to $N_1 = N$, so that it works on the original problem $\mathcal{P}$. Due to its greater impact on the algorithm and to distinguish s_1 from the other slaves, we call it the pollster slave, or simply the pollster.

To reduce the expected high number of evaluations done by the successive pollster instances, a single-POLL MADS algorithm is used (the POLL directions are reduced to a single element), with the conditions that the union of all the normalized directions used throughout the algorithm are dense in the unit sphere, and that the norms of those directions is in the proper relation with the mesh size parameter.

Moreover, the pollster is limited to only one MADS iteration, with no SEARCH step and one POLL step. It follows that at most one function evaluation will be performed (zero function evaluation if the unique POLL trial point is in the cache), and the pollster mesh size parameter Δ_k^1 will not be updated (this is done by the master).

The notation MADS(pollster) or MADS(s_1) refer to the single-POLL MADS algorithm performed by the pollster. MADS(pollster) is defined so that its mesh size parameter Δ_k^1 cannot be larger than the master mesh size Δ_k^M at iteration k of PSD-MADS (see (4.2)).

The pollster pseudocode is shown in Figure 4.2. The pollster mesh size is updated by the master. The best obtained solution is described by x_p, which is sent to the master. The convergence analysis in Section 4.4 is based on the pollster, and on the fact that consecutive runs of MADS(s_1) form a valid single-POLL MADS instance on $\mathcal{P}$.

Pollster ($p = 1$)
- given the master data (pollster mesh size Δ_k^1, starting point x_0)
- solve problem $\mathcal{P}$: MADS(pollster)
- – terminate after a single evaluation
- send optimization data to master (pollster solution x_p)

Figure 4.2: Pseudocode for pollster slave. MADS(pollster) considers all n variables with the single-POLL direction $b(\ell)$, and terminates after one iteration.

4.3.3 The regular slaves s_2 to s_{q-2}, on $M(\Delta_j^p)$

The regular slaves s_p, $p \in Q_{reg}$, work on subsets N_p of N, and use positive spanning sets of directions. The MADS algorithm working on problem $\mathcal{P}_p(x^*)$ and performed by slave s_p is designated by MADS(s_p).

Subproblem $\mathcal{P}_p(x^*)$ is defined as a $|N_p|$ variable problem since all the variables in $N \setminus N_p$ are fixed. Trial points generated by MADS(s_p) are then in $\mathbb{R}^n$, with some coordinates fixed. The values of these fixed coordinates are directly taken from the starting point for

MADS(s_p), i.e., x^*, the incumbent solution. The user supplies a parameter, $bbe_{max} > 0$, that indicates the maximum allowed number of black-box calls for the application of MADS to the optimization of a subproblem.

The pseudocode for the regular slaves is shown in Figure 4.3. MADS(s_p) generates trial points on meshes of sizes Δ_j^p, where j is the iteration counter of the subproblem algorithm. The initial mesh size Δ_0^p for MADS(s_p) is set by the master. The value of the parameter Δ_{min}^p also is supplied by the master, and equals Δ_k^M, where k is the PSD-MADS iteration at which MADS(s_p) started. Finally, we impose that no mesh size for MADS(s_p), $p \in Q_{reg}$, exceeds the PSD-MADS initial mesh size, Δ_0^{user}, provided by the user. MADS(s_p) terminates if bbe_{max} evaluations are made, or if a minimal mesh size Δ_{min}^p is reached. The final mesh size (Δ_{stop}^p), and the best solution found (x_p), are sent to the master.

The union of all regular slaves MADS(s_p) instances is interpreted as a SEARCH step for the total problem single-POLL MADS algorithm. This is important to the convergence analysis in Section 4.4.

Slave s_p ($p \in Q_{reg}$)
given the master data (initial and minimum mesh sizes Δ_0^p, Δ_{min}^p ; starting solution x_0, set of variables N_p)
solve subproblem $\mathcal{P}_p(x^*)$: MADS(s_p)
– terminate when $\Delta_j^p < \Delta_{min}^p$ or after bbe_{max} function evaluations
send optimization data to master
(final mesh size Δ_{stop}^p, slave solution x_p)

Figure 4.3: Pseudocode for slaves processes. Does not include pollster slave, which is specifically described in Figure 4.2.

4.3.4 The cache server ($(q-1)^{\text{th}}$ process)

The cache server is a specialized process that simply memorizes all evaluated points. Each time a process generates a trial point, the cache server is interrogated in case this point has already been evaluated, to avoid unnecessary expensive functions evaluations. The cache server provides the global availability of any improvement made by any slave. This is interpreted in Section 4.5 as a SEARCH step (the *cache* SEARCH) by the regular slaves on their subproblems.

4.3.5 The master (q^{th} process)

The master process coordinates the work of the $q-2$ slaves. It waits for slave results, updates data, and assigns work to slaves. It only evaluates the black-box functions at the starting point x_0.

The master process provides the master mesh size Δ_k^M at iteration k of PSD-MADS, which is the link between the mesh sizes Δ_k^1 and Δ_j^p on which the different MADS(s_p), $p \in Q_{reg}$ work. The initial master mesh size $\Delta_0^M = \Delta_0^{user}$ is set by the user.

The master process updates the pollster mesh size Δ_k^1, after a pollster instance terminates. If no improvement is made by any slave s_1 to s_{q-1} during iteration k, the iteration is a failure and the pollster mesh size is reduced. If the iteration succeeds, then the pollster mesh size is increased. In all cases, the pollster mesh size is smaller than the master mesh size (4.2). The value of the pollster mesh size is also kept less than or equal to Δ_0^{user}.

For all regular slaves s_2 to s_{q-1}, the minimal mesh size Δ_{min}^p is set to the current value of Δ_k^M. This, as is explained in more detail in the convergence analysis, leads to the fact that at iteration k of PSD-MADS, no regular slave can generate trial points on meshes finer than $M(\Delta_k^M)$, and that all the slaves work in fact on the pollster mesh of size Δ_k^1.

The master process pseudocode is described in Figure 4.4, and the pollster mesh size update is detailed in Figure 4.5. The pseudocode for the master process implies that when the master mesh size is updated, it is always possible to find an integer $\alpha_k \in [0; w^+]$ such that $\tau^{\alpha_k}\Delta_k^1 \leq \min_{p \in Q_{reg}} \Delta_{min}^p$. The next proposition shows that $\alpha_k = 0$ is always a candidate.

Proposition 4.3.2 *At iteration k of the* PSD-MADS *algorithm, there exists a nonnegative integer α_k such that $\tau^{\alpha_k}\Delta_k^1 \leq \min_{p \in Q_{reg}} \Delta_{min}^p$.*

Proof. At iteration 0, we defined (Figure 4.4) $\Delta_0^1 = \Delta_0^M = \Delta_0^{user} (= \min_{p \in Q_{reg}} \Delta_{min}^p)$ so $\alpha_0 = 0$ satisfies the hypotheses. Then, following the algorithm in Figure 4.4, $\Delta_1^M = \Delta_0^{user}$ and $\min_{p \in Q_{reg}} \Delta_{min}^p$ at iteration 1 is equal to Δ_0^{user}. The algorithm in Figure 4.5 ensures that Δ_1^1 is bounded above by Δ_0^{user}, and therefore $\alpha_1 = 0$ is a possible value.

Suppose, by way of induction, that for some $k \geq 2$, the proposition is true at iteration $k-1$. It follows that $\Delta_k^M = \tau^{\alpha_{k-1}}\Delta_{k-1}^1 \leq \min_{p \in Q_{reg}} \Delta_{min}^p$, and as it corresponds possibly to a new value for Δ_{min}^p, $p \in Q_{reg}$, the new smaller possible value of $\min_{p \in Q_{reg}} \Delta_{min}^p$ at iteration k remains Δ_k^M. The largest value that Δ_k^1 may take is also Δ_k^M, which shows $\alpha_k = 0$ again applies, thus validating the result by induction. ∎

This proof allows all values of α_k to be zero, but in practice, non-zero values are likely. For example, if iteration 1 failed and $\Delta_1^1 = \Delta_0^{user}$, then the following mesh updates are preferred: $\Delta_2^M \leftarrow \Delta_0^{user}$ ($\alpha_1 = 0$) and $\Delta_2^1 \leftarrow \Delta_0^{user}/4$. Note that $\min_{p \in Q_{reg}} \Delta_{min}^p$ is still equal to Δ_0^{user} at iteration 2, and so α_2 can be either 0 or 1.

4.4 Convergence analysis of PSD-MADS

It is shown here that the entire algorithm may be interpreted as a single-POLL MADS algorithm applied to the original problem $\mathcal{P}$ and that conditions are met so that the main

Master

[0] initialization

$x^* \leftarrow x_0 \in \Omega$, $\Delta_0^1 \leftarrow \Delta_0^M \leftarrow \Delta_0^{user} > 0$, $k \leftarrow 0$, $\omega^- \leq -1$, $\omega^+ \geq 0$

start MADS(pollster) with (Δ_0^{user}, x_0)

for all $(p \in Q_{reg})$

construct the set of indices N_p and set $\Delta_{min}^p \leftarrow \Delta_0^M$

start MADS(s_p) with $(\Delta_0^{user}, \Delta_{min}^p, x_0, N_p)$

[1] iterations

given values from a slave s_p (Δ_{stop}^p, x_p)

if $\big(f(x_p) < f(x^*)\big)$ (success)

$x^* \leftarrow x_p$

if $(p = 1)$ (pollster, Δ_{stop}^p corresponds to Δ_k^1)

$\Delta_{k+1}^M \leftarrow \tau^{\alpha_k}\Delta_k^1 \leq \min_{p \in Q_{reg}} \Delta_{min}^p$ with $\alpha_k \in [0; \omega^+]$

$\Delta_{k+1}^1 \leftarrow \tau^{\omega_k}\Delta_k^1$ (detailed in Figure 4.5)

$k \leftarrow k + 1$

start MADS(pollster) with (Δ_k^1, x^*)

else (regular slave)

construct N_p

$\Delta_{min}^p \leftarrow \Delta_k^M$

$\Delta_0^p \leftarrow \tau^\gamma \Delta_{stop}^p$ with $\gamma \in \mathbb{Z}$ and so that $\Delta_k^M \leq \Delta_0^p \leq \Delta_0^{user}$

start MADS(s_p) with $(\Delta_0^p, \Delta_{min}^p, x^*, N_p)$

goto [1]

Figure 4.4: Pseudocode for master process. Δ_k^M and Δ_k^1 are the master and pollster mesh sizes at iteration k, and Δ_{stop}^p the last mesh size of a slave s_p. If $p = 1$, $\Delta_{stop}^p = \Delta_k^1 \leq \Delta_k^M$, and else $\Delta_{stop}^p \geq \Delta_k^M$. The master evaluates the black-boxes just once for x_0.

pollster mesh size update $\Delta^1_{k+1} \leftarrow \tau^{\omega_k}\Delta^1_k$

if (iteration success)

$\omega_k = \alpha_k \in [0;\omega^+]$ $\left(\Delta^1_{k+1} \leftarrow \Delta^M_{k+1}\right)$

(pollster mesh size increase, $\Delta^1_{k+1} \geq \Delta^1_k$)

else

$\omega_k \in [\omega^-;-1]$

(pollster mesh size decrease, $\Delta^1_{k+1} < \Delta^1_k$)

Figure 4.5: Update of the next pollster mesh size Δ^1_{k+1}. In any case, the pollster mesh size verifies $\Delta^1_k \leq \Delta^M_k$.

convergence results from [20] hold. These conditions are that the regular slaves generate a finite number of trial points lying on the pollster mesh, and that all these trial points can be interpreted as a SEARCH step with the pollster slave providing the POLL step. This is detailed in Figure 4.6, and we refer to it as the *apparent pollster algorithm*. This algorithm is another way of interpreting the PSD-MADS algorithm described by the pseudocodes in Figures 4.2, 4.3, 4.4, and 4.5. Iteration k of the apparent pollster algorithm corresponds to the iteration k of PSD-MADS (used by the master process), and the notions of iteration success and failure remain the same.

The convergence analysis in this section proves that the apparent pollster algorithm is a single-POLL MADS algorithm with the following components:

- A SEARCH step performed by regular slaves $s_2, s_3, ..., s_{q-2}$ on meshes of coarseness larger than or equal to Δ^M_k;
- A POLL step at iteration k (the same k used by the master process in Figure 4.4) performed by one call to the pollster slave s_1 on a mesh of size $\Delta^1_k \leq \Delta^M_k$;
- A mesh update performed by the master process with $\Delta^1_{k+1} \leftarrow \tau^{\omega_k}\Delta^1_k$ and the integer $\omega_k \in \begin{cases} [0;\omega^+] & \text{iteration success} \\ [\omega^-;-1] & \text{iteration failure.} \end{cases}$

Apparent pollster algorithm

[0] Initialization

$x_0 \in \Omega$, $\Delta_0^M \leftarrow \Delta_0^1 \leftarrow \Delta_0^{user} > 0$, $k \leftarrow 0$

[1] POLL **and** SEARCH **steps**

SEARCH **step** (by other slaves, opportunistic)

ask cache server for $x_s \in M(\Delta_k^M) \subseteq M(\Delta_k^1)$

Single-POLL **step**

construct and evaluate $P_k = \{x_{poll}\} \subseteq M(\Delta_k^1)$

[2] Updates

determine type of success of iteration k

$\Delta_{k+1}^1 \leftarrow \tau^{\omega_k} \Delta_k^1$ (cannot be larger than Δ_{k+1}^M)

$x_{k+1} \leftarrow (x_s$ or x_{poll} or $x_k)$

$k \leftarrow k + 1$

goto [1]

Figure 4.6: Detailed pseudocode of the apparent pollster algorithm, the algorithm from the point of view of the pollster slave. At every moment, a finite number of $M(\Delta_k^1)$ points are evaluated in parallel by other slaves. These evaluations are considered within the opportunistic SEARCH step. Δ_k^M is updated by the master after the POLL step.

The master mesh size parameter Δ_k^M at iteration k is the link described by inequalities (4.2) between the mesh size of MADS(pollster) and the different mesh sizes of MADS(s_p). It is updated by the master with the MADS(pollster) mesh (via $\Delta_{stop}^p = \Delta_k^1$), in such a way that, at every iteration k of the apparent pollster algorithm, Δ_k^1 satisfies $\Delta_k^1 \leq \Delta_k^M$. This Δ_k^M update by the master in the apparent pollster algorithm occurs when the mesh size Δ_k^1 is updated, and while its value does not change during the POLL step, it can possibly be updated during the SEARCH step, since that is performed in parallel. This possible change of the Δ_k^M value within the SEARCH step of the apparent pollster algorithm is governed by the fact that Δ_k^M cannot be exceeded by any regular slave mesh size ($\Delta_k^M \leq \min_{p \in Q_{reg}} \Delta_{min}^p$).

To show that the apparent pollster algorithm is a valid single-POLL MADS algorithm applied to the original problem $\mathcal{P}$, and that the convergence conditions of [20] hold, the

SEARCH trial points, whose evaluations are performed at any time in parallel by the other slaves, must remain finite in number and on the current pollster mesh at iteration k, Δ_k^1. This will be shown via the following propositions.

Proposition 4.4.1 *The mesh size parameter at iteration j of the* MADS *algorithm performed by a slave s_p, $p \in Q_{reg}$, on a subproblem $\mathcal{P}_p(x^*)$, satisfies $\Delta_j^p = \tau^{-\eta_j}\Delta_0^{user}$ for some integer $\eta_j \geq 0$. This can be extended to the pollster slave at iteration k with $\Delta_k^1 = \tau^{-\eta_k}\Delta_0^{user}$.*

Proof. We first show that the proposition is true for the first optimization subproblem solved by a regular slave s_p, $p \in Q_{reg}$. The initial mesh size parameter used for this MADS instance is Δ_0^{user}, and with the standard MADS mesh update rules, at iteration j, $\Delta_j^p = \tau^{\omega_{j-1}}\Delta_{j-1}^p = \ldots = \tau^{\sum_{i=0}^{j-1}\omega_i}\Delta_0^{user}$. Then $\eta_j = -\sum_{i=0}^{j-1}\omega_i \geq 0$ because no mesh size can be larger than Δ_0^{user}.

Suppose now that the proposition is true for the r^{th} MADS instance performed by s_p. In particular, the last mesh size parameter of this instance can be written $\Delta_{stop}^p = \tau^{-\eta_{stop}}\Delta_0^{user}$ where η_{stop} is a nonnegative integer. From the algorithm described in Figure 4.4, the first mesh size parameter of the $(r+1)^{\text{th}}$ MADS instance performed by s_p is $\Delta_0^p = \tau^{\gamma}\Delta_{stop}^p$ with $\gamma \in \mathbb{Z}$. Then at iteration j of the $(r+1)^{\text{th}}$ instance, $\Delta_j^p = \tau^{\sum_{i=0}^{j-1}\omega_i}\Delta_0^p$ and $\eta_j = -\sum_{i=0}^{j-1}\omega_i - \gamma + \eta_{stop} \geq 0$ because $\Delta_j^p \leq \Delta_0^{user}$. The proposition can be extended to the pollster slave with the same induction proof on k. ∎

Proposition 4.4.2 *At iteration k of* PSD-MADS*, and at iteration j of the* MADS *algorithm performed by s_p ($p \in Q_{reg}$) on a subproblem $\mathcal{P}_p(x^*)$, there exists a nonnegative integer β_j such that $\Delta_j^p = \tau^{\beta_j}\Delta_k^M$.*

Proof. From the algorithm in Figure 4.4, the master mesh size parameter, at iteration k of PSD-MADS, can be written $\Delta_k^M = \tau^{\alpha_k - 1}\Delta_{k-1}^1$ with $\alpha_{k-1} \in \mathbb{N}$, and $\Delta_{k-1}^1 =$

$\tau^{-\eta_{k-1}}\Delta_0^{user}$, with $\eta_{k-1} \in \mathbb{N}$, from Proposition 4.4.1. From the same proposition, the mesh size parameter at iteration j of MADS(s_p), $p \in Q_{reg}$, can be written $\Delta_j^p = \tau^{-\eta_j}\Delta_0^{user}$, $\eta_j \in \mathbb{N}$. Then $\Delta_j^p = \tau^{\beta_j}\Delta_k^M$ with $\beta_j = \eta_{k-1} - \eta_j - \alpha_{k-1}$. The minimal mesh size parameter Δ_{min}^p considered by MADS(s_p) corresponds to Δ_i^M where $i \leq k$ is an anterior iteration of PSD-MADS. The current value of Δ_k^M was chosen to be smaller than $\min_{p \in Q_{reg}} \Delta_{min}^p \leq \Delta_i^M$. Then, $\Delta_k^M \leq \Delta_i^M \leq \Delta_j^p$ and β_j is a nonnegative integer. ∎

An immediate corollary, with Hypothesis 4.3.1, is that at iterations k of PSD-MADS and j of MADS(s_p), $p \in Q_{reg}$, $M(\Delta_j^p) \subseteq M(\Delta_k^M)$.

Proposition 4.4.3 *At iteration k of* PSD-MADS, *every trial point generated by the* MADS *algorithm performed by s_p, $p \in Q_{reg}$, on any subproblem $\mathcal{P}_p(x^*)$, lies on the pollster mesh $M(\Delta_k^1)$.*

Proof. From the algorithm in Figure 4.4, the pollster and master mesh size parameters at iteration k of PSD-MADS are linked with $\Delta_k^M = \tau^{\alpha_k}\Delta_k^1$, $\alpha_k \in \mathbb{N}$. With Hypothesis 4.3.1 and Proposition 4.4.2, at iteration j of MADS(s_p), $M(\Delta_j^p) \subseteq M(\Delta_k^M) \subseteq M(\Delta_k^1)$. Since all MADS($s_p$) trial points are constructed on $M(\Delta_j^p)$, they also lie on $M(\Delta_k^1)$. ∎

This series of propositions ensures that all the trial points of the SEARCH step of the apparent pollster at iteration k, performed in parallel by regular slaves, lie on the current pollster mesh Δ_k^1. In addition, their number remains finite as the time between two iterations, corresponding to a single-point POLL, is finite (with the hypothesis that black-boxes evaluate, or are terminated to return ∞, in finite time). The PSD-MADS algorithm, viewed from the perspective of the pollster slave, thus executes a valid single-POLL MADS search, and the main convergence results of [20] remain valid. Let $\hat{x}$ be the limit of a subsequence of PSD-MADS incumbents at unsuccessful iterations, then:

- If f is Lipschitz near $\hat{x} \in \Omega$, then the Clarke derivative satisfies $f^\circ(\hat{x}; v) \geq 0$ for all $v \in T_\Omega^H(\hat{x})$, the hypertangent cone to Ω at $\hat{x}$;
- In the unconstrained case and if f is strictly differentiable at $\hat{x}$, $\nabla f(\hat{x}) = 0$.

As mentioned in Section 4.2.3, the fact that the single-POLL version of MADS is used sacrifices the zero'th order result of [20], i.e., $\hat{x}$ cannot be said to be the limit of local optima on meshes that get infinitely fine.

4.5 A practical implementation of PSD-MADS

This section proposes a practical implementation of the PSD-MADS algorithm described in Section 4.3 based on the LTMADS implementation proposed in [20] and summarized in Section 4.2.4. An illustrative example and some numerical tests complete the implementation description.

4.5.1 PSD-MADS implementation

Verification of Hypothesis 4.3.1

The above convergence analysis relies on Hypothesis 4.3.1. An easy way to satisfy this hypothesis is to simply choose τ to be an integer. Indeed, consider the mesh point $x \in M(\Delta)$, and mesh size $\Delta \in \mathbb{R}$. From the mesh definition (4.1), x can be written as $y + \Delta \sum_{i=1}^{n_D} z_i d_i$ where y belongs to V, the set of currently evaluated points, and the z_i are nonnegative integers. Now, if $\Delta' = \tau^\omega \Delta$ where $\omega \in \mathbb{N}$ and $1 \leq \tau \in \mathbb{N}$, then x can be rewritten as $x = y + \Delta' \sum_{i=1}^{n_D} \tau^\omega z_i d_i$. It follows that, $\tau^\omega z_i \in \mathbb{N}$, $i = 1, 2, ..., n_D$, and therefore $x \in M(\Delta')$. We have shown that $M(\Delta) \subseteq M(\Delta')$ and thus, Hypothesis 4.3.1 is satisfied. In the proposed PSD-MADS implementation, the LTMADS fixed value of $\tau = 4$ is used.

Directions used by the pollster

The LTMADS direction $b(\ell)$ is used in the single-POLL MADS algorithm executed by the pollster slave. The union of normalized directions $b(\ell)$, $\ell = 1, 2, ...$, is dense in the unit sphere with probability one, and MADS(pollster) with the $b(\ell)$ direction respects the conditions for a valid single-POLL MADS algorithm.

Sets N_p of subproblems variables

We take the sets N_p, $p \in Q_{reg}$, to be randomly generated by the master using an uniform distribution before each subproblem parameters are sent to a regular slave process. In order to keep an easy parametrization of this PSD-MADS implementation, the number of variables for each subproblem is fixed throughout the entire algorithm, $|N_2| = |N_3| = ... = |N_q| = ns$, where ns is a parameter chosen by the user (recall that for the pollster, $N_1 = N$). Furthermore, when MADS(s_p), $p \in Q_{reg}$, succeeds in improving the incumbent, the same set N_p is kept for the next run performed by the slave s_p.

Mesh update rules

The mesh directions of definition (4.1) are the standard LTMADS $2n$ directions, $D = [-I_n \; I_n]$. The following mesh size parameter updates are in accordance with the LTMADS mesh update rules:

- **Regular slaves mesh size Δ_j^p (at iteration j of MADS(s_p), $p \in Q_{reg}$):** after an iteration fails, the mesh size is updated with $\Delta_{j+1}^p \leftarrow \Delta_j^p/4$ ($\omega_j = -1$ in Figure 4.1). If a POLL step is successful, $\Delta_{j+1}^p \leftarrow 4\Delta_j^p$ ($\omega_j = 1$). In the next SEARCH step, if a successful point is found in the cache server, set $\Delta_{j+1}^p \leftarrow 4\Delta_{cache}$ where

Δ_{cache} is the mesh size used to generate this point. Equation (4.3) summarizes these updates:

$$\Delta_{j+1}^{p} \leftarrow \begin{cases} \min\{\Delta_0^{user}, 4\Delta_j^p\} & \text{POLL success} \\ \min\{\Delta_0^{user}, 4\Delta_{cache}\} & \text{cache SEARCH success} \\ \Delta_j^p/4 & \text{iteration failure.} \end{cases} \tag{4.3}$$

If $\Delta_{j+1}^{p} < \Delta_{min}^{p}$, or if the number of new function evaluations exceeds bbe_{max}, MADS(s_p) terminates and communicates $\Delta_{stop}^{p} = \Delta_j^p$ to the master. The next optimization performed by this slave will start with an initial mesh size parameter Δ_0^p equal to $4^{\gamma}\Delta_{stop}^{p}$, with $\gamma = 1$ if at least one success was achieved since the beginning of the current optimization (even by another slave), or else $\gamma = -1$. However, this may lead to a value smaller than $\Delta_{min}^{p} = \Delta_k^M$, and in this case, set $\Delta_0^p \leftarrow \Delta_k^M$.

The Δ_0^p choice for the next MADS(s_p) is summarized by:

$$\Delta_0^p \text{ (next MADS}(s_p)) \leftarrow \begin{cases} \min\{\Delta_0^{user}, 4\Delta_{stop}^p\} & \text{success} \\ \max\{\Delta_k^M, \Delta_{stop}^p/4\} & \text{else.} \end{cases} \tag{4.4}$$

- **Master mesh size Δ_k^M at iteration k of** PSD-MADS**:** the update of the master mesh size is performed by the master after a pollster instance terminates. Δ_{k+1}^M is bounded below by the mesh size parameter of the terminated pollster, Δ_k^1, and above by the minimum of all Δ_{min}^p values currently used by regular slaves. These Δ_{min}^p values correspond to previous master mesh sizes.

  It would be possible to choose the parameter α_k in Figure 4.4 at each update so that Δ_{k+1}^M is fixed to Δ_0^{user}, with α_k equal to the η_k from Proposition 4.4.1. However, such a strategy would not be efficient as regular slaves would always generate trial points on the same mesh $M(\Delta_0^{user})$. The master mesh size has then to be reduced

somehow through the PSD-MADS evolution. However, it should not be reduced too rapidly, or the algorithm would progress slowly, or even terminate prematurely in practice.

We propose the following strategy: from Figure 4.4, Δ_k^M is updated by $\Delta_{k+1}^M \leftarrow 4^{\alpha_k}\Delta_k^1$ with $\alpha_k \in \mathbb{N}$, and from Proposition 4.4.1, $\Delta_k^1 = 4^{-\eta_k}\Delta_0^{user}$ with some $\eta_k \in \mathbb{N}$. If iteration k succeeded, set $\alpha_k = \eta_k = \log_4(\Delta_0^{user}/\Delta_k^1)$ (maximal Δ_k^M increase), and else, $\alpha_k = \eta_k - \lfloor(\eta_k+1)/3\rfloor$ (attenuated Δ_k^M increase). In both cases, if Δ_{k+1}^M is greater than at least one of the regular slaves mesh size Δ_{min}^p, then Δ_{k+1}^M is set to the least Δ_{min}^p values. This can be summarized by the following:

$$\Delta_{k+1}^M \leftarrow \begin{cases} \min\left\{\Delta_0^{user}, \min\limits_{p\in Q_{reg}} \Delta_{min}^p\right\} & \text{iteration success} \\ \min\left\{4^{-\lfloor(\eta_k+1)/3\rfloor}\Delta_0^{user}, \min\limits_{p\in Q_{reg}} \Delta_{min}^p\right\} & \text{iteration failure.} \end{cases} \tag{4.5}$$

For example, if $\Delta_0^{user} = \Delta_{min}^p = 1$ for each $p \in Q_{reg}$ and if the pollster instance fails with a pollster mesh size of $\Delta_k^1 = 1/16$, then the master mesh size Δ_{k+1}^M is set to $1/4$ ($\eta_k = 2$, $\alpha_k = 1$; this is what happens at time $t = t_3$ in the example described in Section 4.5.2) .

- **Pollster mesh size Δ_k^1 at iteration k of** PSD-MADS**:** in the case of an iteration success, Δ_{k+1}^1 is set to Δ_{k+1}^M ($\omega_k = \alpha_k \in \mathbb{N}$), or else $\Delta_{k+1}^1 = \Delta_k^1/4$ ($\omega_k = -1$):

$$\Delta_{k+1}^1 \leftarrow \begin{cases} \Delta_{k+1}^M = \min\left\{\Delta_0^{user}, \min\limits_{p\in Q_{reg}} \Delta_{min}^p\right\} & \text{iteration success} \\ \Delta_k^1/4 & \text{iteration failure.} \end{cases} \tag{4.6}$$

MADS **parameters for** MADS(s_p), $p \in Q_{reg}$

The regular slaves $p \in Q_{reg}$ solve MADS(s_p) using the standard MADS $2|N_p|$ directions. All POLLs are opportunistic, meaning that a subproblem optimization terminates as soon as a better point is found. The one point dynamic SEARCH strategy of [20] is

also performed: it consists, after a successful POLL step, in evaluating, within a single-point SEARCH, the black-box functions at a mesh point located further along the same successful direction.

In addition to the POLL and the one-point dynamic SEARCH, MADS(s_p) performs a specialized SEARCH step, which simply consists in querying the cache server for the best available feasible point. This special SEARCH step generates no additional function evaluation and allows every regular slave to know the best points eventually obtained by other slaves. Note that this SEARCH step has no obligation to give a point lying on the current mesh of MADS(s_p), but this does not influence the convergence analysis as it is based on the pollster s_1, and as the point given by this SEARCH must come from another slave, thus lying on $M(\Delta_k^M)$.

Practical termination criteria

The regular slaves $p \in Q_{reg}$ terminate MADS(s_p) as soon as the mesh size parameter Δ_j^p drops below $\Delta_{min}^p = \Delta_k^M$ (where k is the PSD-MADS iteration at which MADS(s_p) was started), or after a finite number of bbe_{max} black-box function evaluations are made. The PSD-MADS algorithm is stopped after an overall limit of bbe_{max}^{global} black-box evaluations is reached.

The PSD-MADS implementation described above is illustrated in the next section, where an example of a few steps is presented (Figure 4.7).

4.5.2 A detailed PSD-MADS illustrative example

We consider a problem with $n = 4$ variables and $q = 5$ processes (one pollster, two regular slaves, one master, and one cache). The two regular slaves have a limit of $bbe_{max} = 2$

evaluations, and their sets of variables are of cardinality $ns = 2$. The initial (and maximal) mesh size value is $\Delta_0^{user} = 1$.

The progress of the four processes over time from iteration $k = 4$ to the end of $k = 6$ is illustrated in Figure 4.7. The cache server is not represented in the figure. The grayed rectangles illustrate the black-box evaluations and their widths represent their different running times to return a value. Arrows represent communications between processes.

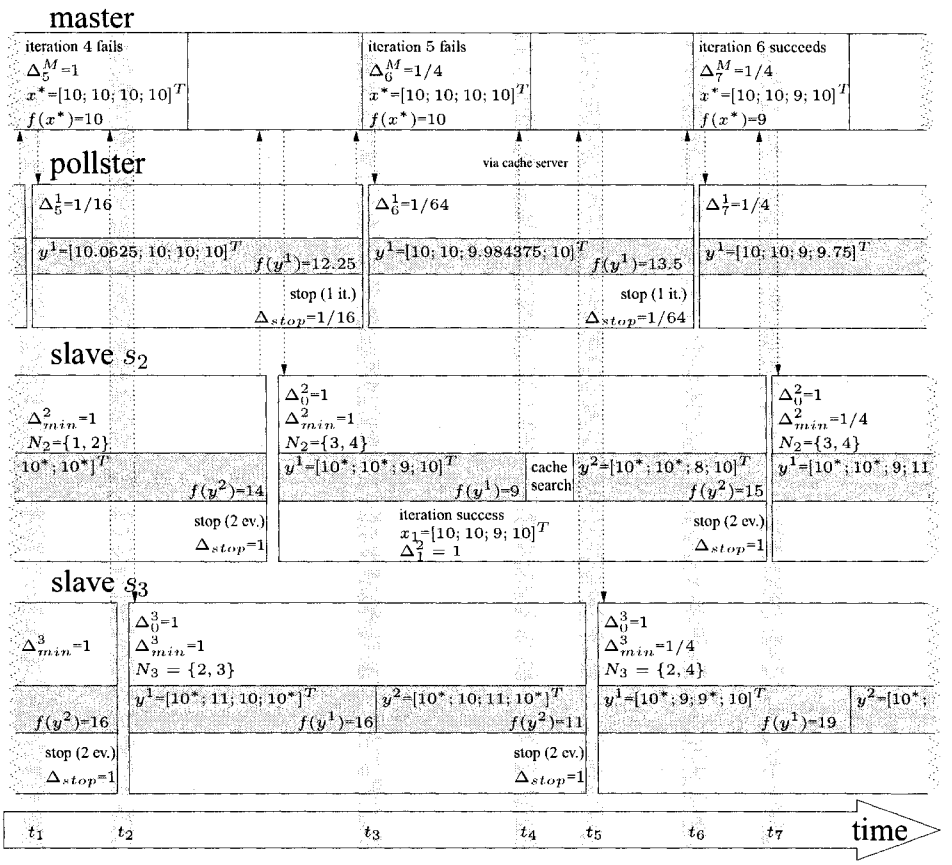

Figure 4.7: A PSD-MADS illustrative example.

The time t_1 corresponds to the beginning of iteration $k = 5$. The incumbent solution at the start of iteration $k = 5$ is $x^* = [10\ 10\ 10\ 10]^T$ with $f(x^*) = 10$ and the current master mesh size is $\Delta_5^M = 1$. This information appears in the figure in the master's section.

The notation y^i is used for the ith POLL trial point of one instance of MADS(s_p), $p \in \{1, 2, 3\}$ solved by a slave. The pollster evaluates the black-boxes at time t_1 at the POLL point $y^1 = [10.0625\ 10\ 10\ 10]^T$ with pollster mesh size $\Delta_5^1 = 1/16$.

At t_2, slave s_3 terminates and communicates to the master: the incumbent is not modified. The stopping criteria Δ_{min}^p value of slave s_3 is set to the current master mesh size value Δ_5^M. The coordinates of the regular slave trial points marked by stars indicate that these coordinates are fixed to the ones of the POLL center. For example, the two trial POLL points of slave s_3 at time t_2 and t_3 are $[10^*\ 11\ 10\ 10^*]^T$ and $[10^*\ 10\ 11\ 10^*]^T$ with POLL center $x_0 = x^* = [10\ 10\ 10\ 10]^T$, $N_2 = \{2, 3\}$, and with a mesh size parameter $\Delta_0^3 = 1$.

At time t_3, the pollster returns information to the master, and iteration 5 is declared to have failed. The master mesh size is set to $\Delta_6^M = 1/4$, with $\eta_5 = 2$ and $\alpha_5 = 1$, according to (4.5) (attenuated master mesh size increase). Also, the pollster mesh size is reduced to $\Delta_6^1 = \Delta_5^1/4 = 1/64$ (4.6).

At t_4, the slave s_2 improves the current incumbent solution, and the mesh size is increased: $\Delta_1^2 \leftarrow 1 = \Delta_0^{user}$ (4.3). Since MADS(s_2) is opportunistic, it begins a new iteration with a cache SEARCH.

At t_5, slave s_3 terminates and communicates to the master: the incumbent is not modified. MADS(s_3) starts with $N_3 = \{2, 4\}$, $\Delta_0^3 = 1$, and $\Delta_{min}^3 = 1/4$, because a new incumbent was produced by s_2 since s_3 last communicated with the master ("success" in equation (4.4)).

At time t_6, the pollster returns information to the master, and iteration 6 is declared to be successful. The maximal increase is performed for Δ_7^M, which is set to $\min_{Q_{reg}} \Delta_{min}^p = 1/4$ (4.5), and $\Delta_7^1 \leftarrow \Delta_7^M$.

Finally, at t_7, since MADS(s_2) was successful, the new instance of MADS(s_2) keeps the same free variables $N_2 = \{3, 4\}$.

4.5.3 Numerical experiments

The PSD-MADS implementation described in Section 4.5.1 is tested here, on two different problems. The implementation of MADS used to optimize subproblems is the research version of the NOMAD C++ code [15]. The parallel master/slaves paradigm is achieved with MPI with $q = 6$ or 14 processes.

PSD-MADS is compared to three other parallel algorithms, on the same number q of processes: first, the pGPS method described in [47], which corresponds to the unmodified GPS method where evaluations are made in parallel. Then pMADS, which is the trivial adaptation of pGPS that uses MADS instead of GPS. pGPS and pMADS are both synchronous parallel algorithms. The third method is APPS version 5.0 [64,80], the only available GPS asynchronous parallel algorithm.

The first problem (referred as Problem A) considered for the tests is the G2 example from [72]. It has been chosen for its difficulty and for its variable size: our tests involve $n = 20, 50, 250$ and 500 variables. Problem A is written as follows:

$$\min_{x \in \mathbb{R}^n} f(x) = -\left| \frac{\sum_{i=1}^{n} \cos^4 x_i - 2 \prod_{i=1}^{n} \cos^2 x_i}{\sqrt{\sum_{i=1}^{n} i x_i^2}} \right|$$

$$s.t. \begin{cases} -\prod_{i=1}^{n} x_i + 0.75 \leq 0 \\ \sum_{i=1}^{n} x_i - 7.5n \leq 0 \\ 0 \leq x_i \leq 10, \quad i = 1, 2, ..., n. \end{cases}$$

The problem is treated as a black-box, and an upper limit of $100n$ function evaluations is imposed. The feasible starting point for all methods is the center of the bound constrained domain $x_0 = [5\ 5\ ...\ 5]^T \in \Omega$. The best known value from [72], for $n = 20$, is $f(x) = -0.803619$. In [72], various genetic algorithms gave good solutions, after

several hundred thousand of evaluations. Here, after a maximum of 2000 evaluations, PSD-MADS achieved $f(x) \simeq -0.76$.

The second test problem (Problem B) was designed for the MoVars algorithm [26]. It has $n = 60$ variables and one constraint with two different versions: $G \geq 250$, or $G \geq 500$ (see [26] for a more complete description). An infeasible starting point is provided in [26], but cannot be used in the present work since constraints are treated with the extreme barrier approach. The feasible starting points considered here for the two versions of Problem B have been obtained by minimizing the constraint violation $\max\{0, (250 - G)^2\}$ or $\max\{0, (500 - G)^2\}$, from the starting point of [26], with the pMADS algorithm. These optimizations required 3 evaluations for $G \geq 250$, with the resulting feasible point x_0 giving $f(x_0) = 3678.35$ and 74 evaluations for $G \geq 500$, and $f(x_0) = 3014$. These evaluations costs are considered in Figure 4.9. The feasible starting points, our source code for Problem B, and our best points are available on the website www.gerad.ca/Charles.Audet. Our results for Problem B are not compared with the MoVars algorithm results because numerical values are not given in [26]. The best solutions found gave $f(x) = 13.565$ for $G \geq 250$, and $f(x) = 245.866$ for $G \geq 500$.

The various results of this section are measured considering two quantities: z represents the best value of the objective function of problem $\mathcal{P}$, and bbe, the total number of black-box evaluations. One evaluation is counted for the calls to both the objective f and constraints of Ω.

The most representative cost of a black-box algorithm is the number of black-box evaluations. For the same reason, no speedup curves are given and q is kept constant for each problem ($q = 14$ for Problem A and $q = 6$ for Problem B). The PSD-MADS method was not conceived in order to reduce the time to obtain a solution. Instead, we seek to obtain better solutions than a non-decomposing algorithm for problems with a large number of variables ($20 \leq n \leq 500$).

For all our tests, the termination criteria is the maximum total number of black-box evaluations, which is $bbe_{max}^{global} = 100n$ for Problem A and $bbe_{max}^{global} = 3000$ for Problem B (as in [26]).

The initial (and maximal) mesh size parameter is $\Delta_0^{user} = 2$ for Problem A. For Problem B, due to scaling reasons, the value of Δ_0^{user} differs for each variable and is set to be 0.2 times the range of the variables (i.e $\Delta_0^{user} = 0.3$ for the 15 first variables, 0.35 for the next 30 variables, and 0.44 for the last 15 variables). These values has been decided empirically to give good results with standard MADS and APPS runs. The linear nature of the second constraint of Problem A is exploited by APPS. Since PSD-MADS and pMADS involve randomness in the polling directions, 30 runs are made for each test Parallel execution of pGPS and APPS can affect their determinism. However, this effect was ignored and one run was performed for each test.

To measure the quality of the solutions found, the best (z_{best}), worst (z_{worst}), and average (z_{avg}) values of z after the $100n$ evaluations, are reported. Another measure is S_{avg}, representing the area between a curve z *vs* bbe and the line $z = -0.8$ for Problem A (no run gave $z < -0.8$), and $z = 0$ for Problem B. Best runs are obtained with small values for all these quantities.

PSD-MADS was tested on Problem A with $n = 20$ and 50 by varying bbe_{max}, the maximum number of black-box evaluations for each regular subproblem, and ns the number of variables in each subproblem. The number of processes has been set to $q = 14$, in order to fully exploit 12 processors. Good results were obtained by setting $bbe_{max} = 10$, and having the regular slaves working on small dimensional subspaces $ns = 2$. These values are kept for $n > 50$. For Problem B, bbe_{max} is kept to 10. The best results have been obtained by distributing the 60 variables amongst 3 regular slaves with $q = 6$ and $ns = 20$.

Table 4.1 and Figures 4.8 and 4.9 summarize the numerical results. For all instances of Problem A, APPS outperforms pGPS, but it does not do as well as PSD-MADS. In the

three larger instances of Problem A, the worst f value produced by PSD-MADS is always better than all the other methods' f values. For Problem B, pGPS outperforms APPS, and better results are obtained with pMADS and PSD-MADS, with a small advantage to PSD-MADS. In all the curves in Figures 4.8 and 4.9, one can notice that pMADS is always the fastest to descend, but PSD-MADS overtakes it and produces better solutions.

Tableau 4.1: Numerical results for problems A and B: z_{best}, z_{worst} and z_{avg} give information on the 30 runs performed for each PSD-MADS test series, and S_{avg} gives a measure of the area below the curves in Figures 4.8 and 4.9. Best values appear in bold.

Algorithm	Problem	z_{best}	z_{worst}	z_{avg}	S_{avg}
pGPS		-0.450	-0.450	-0.450	1,010
APPS	A	-0.517	**-0.517**	-0.517	806
pMADS	$n = 20$	**-0.775**	-0.434	-0.592	670
PSD-MADS		-0.761	-0.430	**-0.666**	**595**
pGPS		-0.279	-0.279	-0.279	3,400
APPS	A	-0.461	-0.461	-0.461	2,443
pMADS	$n = 50$	-0.498	-0.430	-0.457	1,939
PSD-MADS		**-0.727**	**-0.528**	**-0.663**	**1,553**
pGPS		-0.089	-0.089	-0.089	18,322
APPS	A	-0.193	-0.193	-0.193	16,980
pMADS	$n = 250$	-0.449	-0.438	-0.444	9,703
PSD-MADS		**-0.698**	**-0.464**	**-0.603**	**8,568**
pGPS		-0.073	-0.073	-0.073	37,395
APPS	A	-0.129	-0.129	-0.129	35,816
pMADS	$n = 500$	-0.447	-0.439	-0.443	19,380
PSD-MADS		**-0.688**	**-0.461**	**-0.576**	**17,660**
pGPS		764.741	764.741	764.741	2,731,920
APPS	B	813.237	813.237	813.237	3,906,060
pMADS	$G \geq 250$	32.700	317.167	112.522	1,071,870
PSD-MADS		**13.565**	**307.305**	**70.121**	**965,553**
pGPS		869.559	869.559	869.559	3,552,910
APPS	B	1,162.580	1,162.580	1,162.580	4,579,370
pMADS	$G \geq 500$	417.049	948.768	662.841	2,892,140
PSD-MADS		**245.866**	**731.023**	**463.969**	**2,603,480**

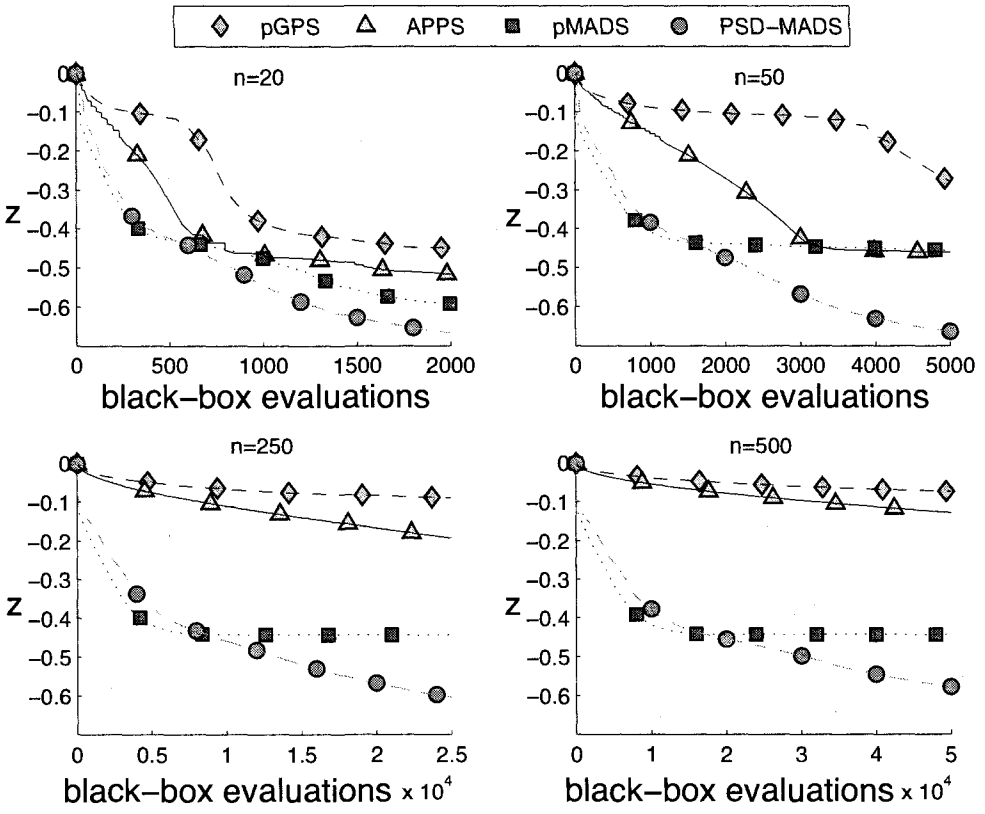

Figure 4.8: Problem A: graphs of the objective function value *vs* the number of evaluations for all test results. PSD-MADS and pMADS plots correspond to average values of 30 runs.

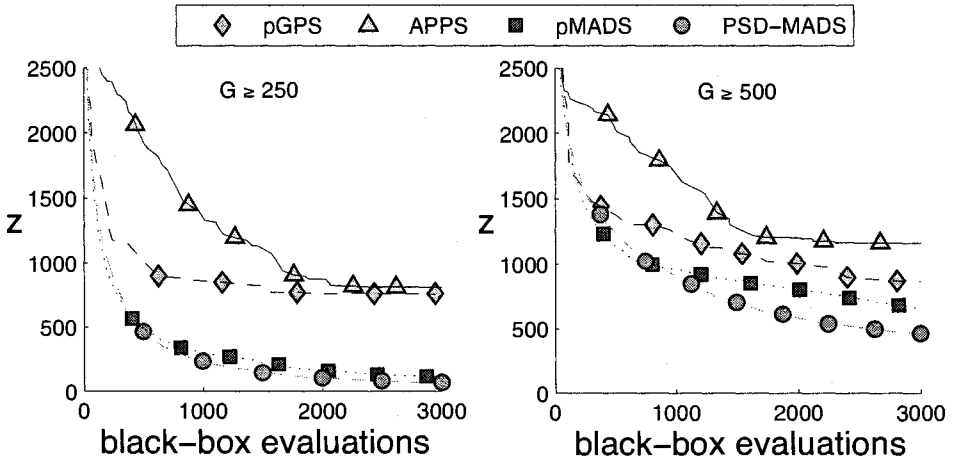

Figure 4.9: Problem B: graphs of the objective function value *vs* the number of evaluations for all test results. PSD-MADS and pMADS plots correspond to average values of 30 runs.

4.6 Discussion and possible extensions

This paper introduced PSD-MADS, a new parallel space decomposition technique with less restrictive conditions on the functions to be optimized than usual PSD methods. A convergence analysis is given based on the Clarke calculus and the MADS convergence analysis. A practical implementation is described, with a small number of parameters (bbe_{max} and ns), and very encouraging results have been obtained on a difficult problem from the literature, with up to 500 variables.

We presented a first basic implementation of PSD-MADS. An obvious extension is a strategy to decide on the sets of variables in the subproblems, which is done randomly for these tests. Of course, it is not clear how to do this in general or we would have done it here. However, for some application, the user may have special knowledge that would help in this task. For example, the user might put similarly scaled variables in the same subproblem.

It would also be interesting to incorporate the PVD idea of the "forget-me-not" terms,

and allow some basic changes in the subproblems for fixed variables. A third possibility would be to perform some additional SEARCH steps in the slave subspaces. Another possible extension would be to reintroduce the synchronization step of the original block-Jacobi method, but without the parallel barrier. This "recomposition" step could be performed in parallel by one of the regular slaves, from a pool of successful points, in order to create a problem similar to the one in [53]. Finally, constraints of Ω could be treated with the progressive barrier [21], instead of the extreme barrier approach. This would allow for infeasible iterates, including the starting point.

CHAPITRE 5

ORTHOMADS: A DETERMINISTIC MADS INSTANCE WITH ORTHOGONAL DIRECTIONS[1]

Mark A. Abramson[2] Charles Audet[3] J.E. Dennis Jr.[4] Sébastien Le Digabel[5]

February 2008

Abstract

The purpose of this paper is to introduce a new way of choosing directions for the Mesh Adaptive Direct Search (MADS) class of algorithms. The advantages of this new ORTHOMADS instantiation of MADS are that the polling directions are chosen deterministically, ensuring that the results of a given run are repeatable, and that they are orthogonal to each other, therefore the convex cones of missed directions at each iteration are minimal in size.

The convergence results for ORTHOMADS follow directly from those already published for MADS, and they hold deterministically, rather than with probability one, as for LT-

[1]Work of the first author was supported by FCAR grant NC72792 and NSERC grant 239436-05. The third author was supported by LANL 94895-001-04 34, and both were supported by AFOSR FA9550-07-1-0302, the Boeing Company, ExxonMobil Upstream Research Company.

[2]Air Force Institute of Technology, Department of Mathematics and Statistics, 2950 Hobson Way, Bldg 641, Wright Patterson AFB, Ohio 45433 USA, www.afit.edu/en/ENC/Faculty/MAbramson/abramson.html, Mark.Abramson@afit.edu.

[3]GERAD and Département de mathématiques et de génie industriel, École Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montréal, Québec H3C 3A7 Canada, www.gerad.ca/Charles.Audet, Charles.Audet@gerad.ca.

[4]Computational and Applied Mathematics Department, Rice University - MS 134, 6100 South Main Street, Houston, Texas 77005-1892 USA, www.caam.rice.edu/~dennis, dennis@rice.edu.

[5]GERAD and Département de mathématiques et de génie Industriel, Ecole Polytechnique de Montréal, C.P. 6079, Succ. Centre-ville, Montréal, Québec H3C 3A7 Canada, Sebastien.Le.Digabel@gerad.ca.

MADS, the first MADS instance. The initial numerical results are quite good for both smooth and nonsmooth, and constrained and unconstrained problems considered here.

Keywords: Mesh Adaptive Direct Search algorithms (MADS), deterministic, orthogonal directions, constrained optimization, nonlinear programming.

5.1 Introduction

This paper considers optimization problems of the form

$$\min_{x \in \Omega} f(x) \,,$$

where $f : \Omega \subset \mathbb{R}^n \to \mathbb{R} \cup \{\infty\}$ is typically evaluated through a black-box computer simulation with no derivatives available, and Ω is a set of feasible points also defined by black-box nonlinear, or even Boolean, constraint functions. Because no exploitable information on the nature of f or Ω exists, we consider direct search methods which only use functions evaluations to drive their search.

Mesh Adaptive Direct Search (MADS) is introduced in [20] as a SEARCH/POLL direct search class of methods with strong convergence properties. It extends the Generalized Pattern Search (GPS) method of [120]. The constraints are treated by the extreme barrier approach, which simply rejects points outside Ω by setting their objective function value to ∞. The first instance of this class of methods is called LTMADS.

LTMADS behaves well in practice, but it has drawbacks that we wish to correct in this paper. First, there is a probabilistic component to the choice of polling directions. For each new mesh size, a random direction is chosen. That direction is completed somewhat randomly to a positive spanning set of directions from the current iterate to other current mesh points. The resulting algorithm is shown to have Clarke stationary point convergence with probability one. However, it has been observed [44] that this way of

choosing polling directions can lead to undesirably large angles between some of the members of the LTMADS polling set at a given iteration.

The purpose of this paper is to introduce a new variant of MADS, which we call ORTHOMADS, that uses an orthogonal positive spanning set of polling directions and thus avoids large angles between polling directions. In Figure 5.3, we show some experiments in which the ORTHOMADS directions do seem better distributed than the LTMADS.

We show that ORTHOMADS shares the same theoretical convergence results as LTMADS, except that the convergence is not qualified by being of probability one. In the tests given here, ORTHOMADS performs generally better than LTMADS.

ORTHOMADS is detailed in Section 5.2, where we show a deterministic way to construct a polling set on the current mesh of orthogonal polling directions (the ORTHOMADS directions). Section 5.2 also gives the convergence results, based on those in [20]. Finally, we present numerical results in Section 5.3 and some concluding remarks in Section 5.4.

Notation: Throughout the text, $\|\cdot\|$ denotes the ℓ_2 norm, $e_i \in \mathbb{R}^n$ is the i^{th} coordinate vector, and $e \in \mathbb{R}^n$ is the vector whose components are all equal to 1. $B_\varepsilon(x)$ denotes the open ball of radius ε around x.

5.2 The ORTHOMADS algorithm

The ORTHOMADS algorithm is described in this section. We will not give details for the MADS class of algorithms and its LTMADS instantiation, since they are available in [20].

Each MADS iteration k is separated into two steps, the SEARCH and the POLL, where the objective function f and the test for feasibility are evaluated at finitely many trial points.

These trial points lie on the mesh M_k defined by

$$M_k = \{x + \Delta_k^m Dz : x \in V_k,\ z \in \mathbb{N}^{n_D}\} \subset \mathbb{R}^n ,$$

where $V_k \subset \mathbb{R}^n$ is the set of all evaluated points by the start of the iteration, $\Delta_k^m \in \mathbb{R}_+$ is the mesh size parameter at iteration k, and D is a matrix in $\mathbb{R}^{n \times n_D}$ composed of n_D directions in $\mathbb{R}^n$. This paper focuses on the POLL step which is characterized by the set of trial points

$$P_k = \{x_k + \Delta_k^m d : d \in D_k\} \subset M_k ,$$

where x_k is the POLL center at iteration k and D_k is the set of POLL directions, which have to form a positive spanning set and to be constructed so that POLL trial points lie on the mesh M_k. In GPS, a related method, the directions contained in D_k are always chosen among the columns of D. Therefore, in GPS, there is only the same finite number of possibilities for selecting the directions in every D_k.

The differences between LTMADS and ORTHOMADS reside in the way to generate the directions in D_k: With LTMADS, D_k is randomly generated and directions are not necessarily orthogonal, possibly leading to large angles between directions and large unexplored convex cones of directions at a given step. However, the union of all normalized LTMADS directions over all iterations k is dense in the unit sphere with probability one.

ORTHOMADS introduces a new way to generate the POLL directions D_k. This new method is deterministic and generates orthogonal directions, which together with their negatives form D_k, and such that the union of all normalized ORTHOMADS directions over all iterations is dense in the unit sphere. Furthermore, the components of these directions are integer, so that POLL points lie on the mesh defined with $D = [I_n \ -I_n]$, where I_n is the identity matrix in dimension n. The orthogonality of the ORTHOMADS directions offers a better distribution of the POLL trial points in the search space, and the advantage of determinism is that numerical results are now easily reproducible. Because

of the random component of LTMADS, we felt that numerical experiments had to be performed on a series of several runs to show the reader the variations in the results.

At each iteration of ORTHOMADS, the main steps for the construction of these directions are as follows. First, the quasi-random Halton sequence produces a vector in $[0, 1]^n$ (Subsection 5.2.1). Second, this vector is scaled and rounded to an appropriate length (Subsection 5.2.2). The resulting direction is called the *adjusted Halton* direction. Third, the Householder transformation is then applied to the adjusted Halton direction, producing n orthogonal and integer vectors, forming a basis for $\mathbb{R}^n$ (Subsection 5.2.3). Finally, the basis is completed to a positive basis formed by $2n$ ORTHOMADS POLL directions D_k, by including in D_k the basis and its negatives (Subsection 5.2.4). Figure 5.1 summarizes these steps, and will be referred to throughout the section.

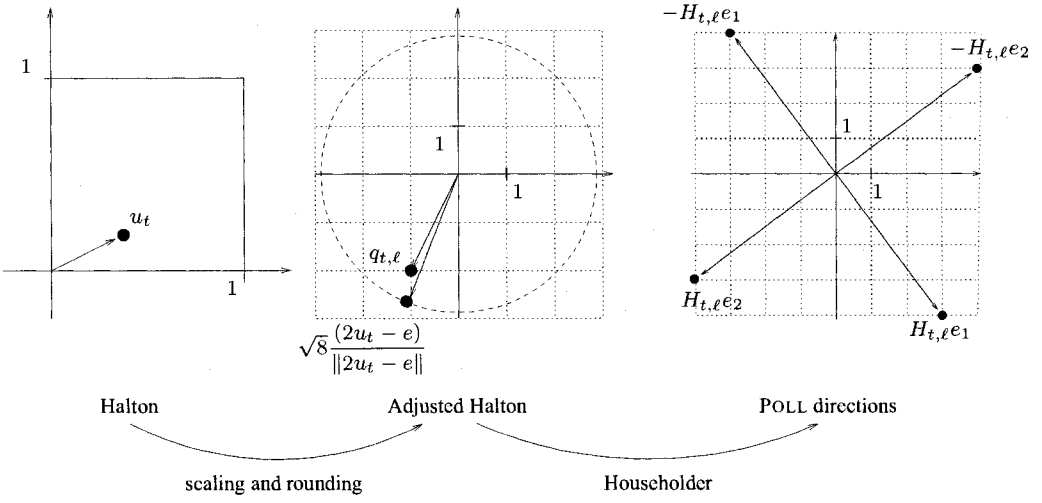

Figure 5.1: Example with $n = 2$ and $(t, \ell) = (6, 3)$. The Halton direction is $u_t = (3/8, 2/9)^T$, the adjusted Halton direction $q_{t,\ell} = (-1, -2)^T$ with $\alpha_{t,\ell} = 2$ and the set of POLL directions $D_k = [H_{t,\ell} \; -H_{t,\ell}]$ with $H_{t,\ell}e_1 = (3, -4)^T$ and $H_{t,\ell}e_2 = (-4, -3)^T$. Every POLL direction $d \in D_k$ satisfies $\Delta_k^m \|d\| = 5/64 < \Delta_k^p = 1/8$.

In this section we show that the ORTHOMADS directions meet all the conditions detailed in [16, 20], so that ORTHOMADS is a valid MADS instance and thus inherits all of its convergence properties.

5.2.1 The Halton sequence u_t

Halton [65] introduced a deterministic family of sequences that grow dense in the hypercube $[0,1]^n$. We consider the simplest sequence of this family, whose t^{th} element is

$$u_t = (u_{t,p_1}, u_{t,p_2}, \ldots, u_{t,p_n})^T \in [0,1]^n$$

where $p_1 = 2, p_2 = 3, p_3 = 5$ and p_j is the j^{th} prime number, and $u_{t,p}$ is the radical-inverse function in base p. More precisely,

$$u_{t,p} = \sum_{r=0}^{\infty} \frac{a_{t,r,p}}{p^{1+r}},$$

where the $a_{t,r,p} \in \mathbb{Z}_+$ are the unique coefficients of the base p expansion of t:

$$t = \sum_{r=0}^{\infty} a_{t,r,p} p^r \ .$$

Table 5.1 describes the first five elements of u_t for $n = 4$ (for example, $u_{5,3} = 1 \times 3^{-2} + 2 \times 3^{-1} = \frac{7}{9}$). Our specific sequence of u_t vectors is from this point addressed as the sequence of Halton directions.

Tableau 5.1: The sequence of Halton directions for $n = 4$ and $t = 0, 1, \ldots, 6$.

t	t in base				u_t			
	2	3	5	7	$u_{t,2}$	$u_{t,3}$	$u_{t,5}$	$u_{t,7}$
0	0	0	0	0	0	0	0	0
1	1	1	1	1	1/2	1/3	1/5	1/7
2	10	2	2	2	1/4	2/3	2/5	2/7
3	11	10	3	3	3/4	1/9	3/5	3/7
4	100	11	4	4	1/8	4/9	4/5	4/7
5	101	12	10	5	5/8	7/9	1/25	5/7
6	110	20	11	6	3/8	2/9	6/25	6/7

In order to remove the linear correlation of the last columns of u_t, it is proposed in [99] to

exclude initial points of the Halton sequence. In the present work, we start the sequence at $t = n + 1$.

The following properties will be used in Subsection 5.2.2:

$$2u_t - e = 0 \quad \Leftrightarrow \quad n = t = 1 \tag{5.1}$$

$$|2u_{t,p_i} - 1| = |2u_{t,p_j} - 1| \quad \Leftrightarrow \quad t = 0\,. \tag{5.2}$$

Property (5.2) follows from the fact that u_{t,p_i} and u_{t,p_j} can be written as reduced fractions with denominators that are powers of different prime numbers p_i and p_j.

The next result shows that the union of all the directions in the sequence of Halton is dense in $[0,1]^n$, i.e. any direction $v \in [0,1]^n$ is an accumulation point of the sequence $\{u_t\}_{t=1}^{\infty}$.

Proposition 5.2.1 *The Halton sequence $\{u_t\}_{t=1}^{\infty}$ is dense in $[0,1]^n$.*

Proof. It suffices to show that for any vector $v \in [0,1]^n$ and any $\varepsilon > 0$, there exists an integer t such that $\|u_t - v\| < \varepsilon$. A construction of such an integer t involves solving a system of n Diophantine equations, and existence of a solution is ensured by the Chinese Remainder Theorem [41], and by the fact that prime numbers are used in the definition of u_t. We refer the reader to [65] for a detailed proof. ∎

5.2.2 The adjusted Halton direction $q_{t,\ell}$

The directions in D_k used in the POLL step of MADS cannot be arbitrarily chosen, they must satisfy precise requirements. The Halton directions u_t do not satisfy these requirements and the first steps toward generating a satisfactory set D_k are to translate, scale and round u_t.

These operations depend on another integer parameter, ℓ, which is related to the mesh size parameter Δ_k^m (this relationship with Δ_k^m is unimportant at this point and will be detailed in Subsection 5.2.4). The parameter ℓ is used to transform the direction u_t into the *adjusted Halton direction* $q_{t,\ell} \in \mathbb{Z}^n$, a direction whose norm is close to $2^{|\ell|/2}$. Furthermore, the normalized direction $\frac{q_{t,\ell}}{\|q_{t,\ell}\|}$ will be constructed so that it is close to $\frac{2u_t - e}{\|2u_t - e\|}$. We already observed in (5.1) that $2u_t - e = 0$ is possible only if $n = 1$ and $t = 1$, and our algorithm never uses $t = 1$ (we begin our Halton sequence at $t = n + 1$, see Subsection 5.2.4).

In order to define $q_{t,\ell}$, we first introduce the following sequence of functions:

$$q_t(\alpha) \;=\; \text{round}\left(\alpha \frac{2u_t - e}{\|2u_t - e\|}\right) \;\in\; \mathbb{Z}^n \cap \left[-\alpha - \frac{1}{2}, \alpha + \frac{1}{2}\right]^n$$

where round refers to the rounding up operation, $\alpha \in \mathbb{R}_+$ is a scaling factor, and u_t is the t^{th} Halton direction. The function $q_t(\cdot)$ is a monotone non-decreasing step function on $\mathbb{R}_+$. Let $\alpha_{t,\ell}$ be a scalar such that $\|q_t(\alpha_{t,\ell})\|$ is as close as possible to $2^{|\ell|/2}$, without exceeding it:

$$\begin{aligned} \alpha_{t,\ell} \;\in\; & \underset{\alpha \in \mathbb{R}_+}{\text{argmax}} \;\; \|q_t(\alpha)\| \\ & \text{s.t.} \quad\;\; \|q_t(\alpha)\| \leq 2^{|\ell|/2} \,. \end{aligned} \tag{5.3}$$

Problem (5.3) can easily be solved using a bisection method. The adjusted Halton direction $q_{t,\ell}$ is defined to be equal to $q_t(\alpha_{t,\ell})$, and the following Lemma ensures that $q_{t,\ell}$ is a nonzero integer vector:

Lemma 5.2.2 *If $t \neq 0$, the adjusted Halton direction satisfies $\|q_{t,\ell}\| \geq 1$.*

Proof. From (5.2), if $t \neq 0$ and $\alpha = \frac{\|2u_t - e\|}{2\|2u_t - e\|_\infty}$, then $\|q_t(\alpha)\| = 1 \leq 2^{|\ell|/2}$ for all ℓ. ∎

The following lemma gives a lower bound on the value of $\alpha_{t,\ell}$. It will be used later to justify that $\alpha_{t,\ell}$ grows large with ℓ.

Lemma 5.2.3 *The optimal solution of Problem (5.3) satisfies* $\alpha_{t,\ell} \geq \dfrac{2^{|\ell|/2}}{\sqrt{n}} - \dfrac{1}{2}$.

Proof. Let $\alpha_{t,\ell}$ be an optimal solution of Problem (5.3) and set $q_{t,\ell} = q_t(\alpha_{t,\ell})$. Then every feasible solution α to Problem (5.3) satisfies

$$\begin{aligned} \|q_t(\alpha)\|^2 &= \left\| \text{round} \left(\frac{\alpha(2u_t - e)}{\|2u_t - e\|} \right) \right\|^2 \\ &= \sum_{i=1}^{n} \text{round} \left(\frac{\alpha(2u_t^i - 1)}{\|2u_t - e\|} \right)^2 \\ &\leq \sum_{i=1}^{n} \left(\alpha + \frac{1}{2} \right)^2 = n \left(\alpha + \frac{1}{2} \right)^2 . \end{aligned}$$

Define $\beta = \frac{2^{|\ell|/2}}{\sqrt{n}} - \frac{1}{2}$. Then β is feasible for Problem (5.3), since $\|q_t(\beta)\|^2 \leq n(\beta + \frac{1}{2})^2 = 2^{|\ell|}$; therefore, $\alpha_{t,\ell} \geq \beta$. ∎

Table 5.2 shows elements of the sequences u_t and $q_{t,\ell}$ for $n = 4$ and eight pairs (t, ℓ) whose values are compatible with the ORTHOMADS algorithm presented in Subsection 5.2.4. The values of $\alpha_{t,\ell}$ and the square norm $\|q_{t,\ell}\|^2$ are also reported. One can also notice that $\alpha_{t,\ell}$ often differs from $2^{|\ell|/2}$. In the example illustrated in Figure 5.1, $(t, \ell) = (6, 3)$ and $q_t(\alpha) = \text{round} \left(\frac{\alpha}{\sqrt{481}} (-9, -20)^T \right)$. An optimal solution of (5.3) is $\alpha_{t,\ell} = 2$ and satisfies $\|q_{t,\ell}\| = \sqrt{5} < \sqrt{8} = 2^{|\ell|/2} < \|q_t(2^{|\ell|/2})\| = \|(-1, -3)^T\| = \sqrt{10}$.

The following proposition gives a property of the scaling and rounding operations, which transform a vector v into $q = \text{round}\,(\alpha v/\|v\|)$. The property states that the directions $v/\|v\|$ and $q/\|q\|$ are arbitrarily close for sufficient large values of α:

Proposition 5.2.4 *Let* $v \neq 0$ *be a vector in* $\mathbb{R}^n$. *For any* $\varepsilon > 0$, *if* $\alpha > \dfrac{2\sqrt{n}}{\varepsilon} + \dfrac{\sqrt{n}}{2}$ *and* $q = \text{round} \left(\alpha \dfrac{v}{\|v\|} \right) \neq 0$, *then* $\left\| \dfrac{q}{\|q\|} - \dfrac{v}{\|v\|} \right\| < \dfrac{\varepsilon}{2}$.

Tableau 5.2: The sequence of Halton directions u_t and the adjusted Halton directions $q_{t,\ell}$ for $n = 4$ and eight pairs (t, ℓ).

(t,ℓ)	u_t				$\alpha_{t,\ell}$	$q_{t,\ell}$				$\|q_{t,\ell}\|^2$
	$u_{t,2}$	$u_{t,3}$	$u_{t,5}$	$u_{t,7}$						
$(5,0)$	5/8	7/9	1/25	5/7	1.0	0	0	−1	0	1
$(6,1)$	3/8	2/9	6/25	6/7	1.0	0	−1	0	1	2
$(7,2)$	7/8	5/9	11/25	1/49	1.0	1	0	0	−1	2
$(8,3)$	1/16	8/9	16/25	8/49	2.5	−2	1	1	−1	7
$(9,4)$	9/16	1/27	21/25	15/49	4.0	0	−3	2	−1	14
$(10,5)$	5/16	10/27	2/25	22/49	5.5	−2	−1	−5	−1	31
$(11,6)$	13/16	19/27	7/25	29/49	7.7	5	4	−4	2	61
$(12,7)$	3/16	4/27	12/25	36/49	11.0	−7	−7	0	5	123

Proof. Consider $\varepsilon > 0$ and $\alpha > 2\sqrt{n}/\varepsilon + \sqrt{n}/2$. The vector q may be expressed as $q = \alpha\dfrac{v}{\|v\|} + \delta$, where $\delta = (\delta_1, \delta_2, \ldots, \delta_n)^T$ and $|\delta_i| < 1/2$ for all $i = 1, 2, \ldots, n$. It follows that

$$\begin{aligned}\left\|\frac{q}{\|q\|} - \frac{v}{\|v\|}\right\| &= \left\|\left(\frac{\alpha}{\|q\|} - 1\right)\frac{v}{\|v\|} + \frac{\delta}{\|q\|}\right\| \\ &\leq \left\|\left(\frac{\alpha}{\|q\|} - 1\right)\frac{v}{\|v\|}\right\| + \left\|\frac{\delta}{\|q\|}\right\| \\ &= \frac{|\alpha - \|q\||}{\|q\|} + \frac{\|\delta\|}{\|q\|}.\end{aligned}$$

The norm of q can be bounded with $\alpha\dfrac{\|v\|}{\|v\|} - \|\delta\| \leq \|q\| \leq \alpha\dfrac{\|v\|}{\|v\|} + \|\delta\|$ and therefore $|\alpha - \|q\|| \leq \|\delta\|$. Furthermore, $\alpha > 2\sqrt{n}/\varepsilon + \sqrt{n}/2 > \sqrt{n}/2$ and $\|\delta\| < \sqrt{n}/2$ implies that α satisfies $0 < \alpha - \|\delta\|$. It follows that

$$\left\|\frac{q}{\|q\|} - \frac{v}{\|v\|}\right\| \leq \frac{2\|\delta\|}{\|q\|} \leq \frac{2\|\delta\|}{\alpha - \|\delta\|} < \frac{\sqrt{n}}{\alpha - \sqrt{n}/2} < \frac{\varepsilon}{2}.$$

■

5.2.3 Construction of an orthogonal integer basis

This subsection gives a way to transform a sequence of directions into a sequence of orthogonal bases. Given an integer nonzero vector $q \in \mathbb{Z}^n$, we apply the (symmetric) scaled Householder transformation [75] to construct an orthogonal basis for $\mathbb{R}^n$ composed of integer vectors:

$$H = \|q\|^2(I_n - 2vv^T), \text{ where } v = \frac{q}{\|q\|}. \tag{5.4}$$

Proposition 5.2.5 *The columns of H form an integer orthogonal basis for* $\mathbb{R}^n$.

Proof. First, the columns of H are mutually orthogonal, since $v^T v = 1$ and

$$\begin{aligned} H^T H &= \|q\|^4(I_n - 2vv^T)^T(I_n - 2vv^T) \\ &= \|q\|^4(I_n - 2vv^T - 2vv^T + 4vv^T vv^T) = \|q\|^4 I_n. \end{aligned}$$

Second, by dividing the previous equation by $\|q\|^4$ and applying symmetry, we reveal the inverse of H as $H^{-1} = \frac{1}{\|q\|^4}H$. Since H^{-1} exists, the columns of H form a basis in $\mathbb{R}^n$. Finally, the entries of

$$H = \|q\|^2 I_n - 2\|q\|^2 \frac{q}{\|q\|}\frac{q^T}{\|q\|} = \|q\|^2 I_n - 2qq^T$$

are integer, since q and $\|q\|^2$ are integer. ∎

The next proposition shows that the Householder transformation applied to a dense set of normalized directions produces a dense set of normalized directions:

Proposition 5.2.6 *For $t = 1, 2, \ldots$, let $v_t = \frac{q_t}{\|q_t\|}$ and $H_t = \|q_t\|^2(I_n - 2v_t v_t^T)$. If $\{v_t\}_{t=1}^{\infty}$ is dense on the unit sphere, then the normalized sequence composed of the i^{th} columns of H_t, $\left\{\frac{H_t e_i}{\|H_t e_i\|}\right\}_{t=1}^{\infty}$ is dense on the unit sphere, for any $i \in \{1, 2, \ldots, n\}$.*

Proof. Let $w \in \mathbb{R}^n$ with $\|w\| = 1$ be an arbitrarily unit vector, $\varepsilon > 0$ be some small positive number, and $i \in \{1, 2, \ldots, n\}$ be the index of a column. For $n > 1$ ($n = 1$ is trivial), we need to show that there exists an index $t \in \mathbb{N}$ such that the ith column of H_t, $H_t e_i$, satisfies

$$\left\| \frac{H_t e_i}{\|H_t e_i\|} - w \right\| < \varepsilon.$$

First, observe that $\|H_t e_i\| = \sqrt{e_i^T H_t^T H_t e_i} = \|q_t\|^2$, and therefore $\frac{H_t e_i}{\|H_t e_i\|} = e_i - 2v_t v_t^T e_i$. Now, define the vector $d \in \mathbb{R}^n$ where

$$d = \begin{cases} \frac{1}{\sqrt{2(1-w_i)}}(e_i - w) & \text{if } w_i < 1, \\ e_{i+1 \,(\mathrm{mod}\, n)} & \text{otherwise.} \end{cases}$$

Observe that if $w_i = 1$ then the vector d satisfies $\|d\| = 1$ and $2d_i d = 0 = e_i - w$, and if $w_i < 1$ then

$$\begin{aligned} \|d\| &= \sqrt{d^T d} = \sqrt{\frac{1}{2(1-w_i)}(e_i - w)^T(e_i - w)} = 1, \\ 2d_i d &= \frac{1}{(1-w_i)}(e_i - w)_i(e_i - w) = e_i - w. \end{aligned}$$

By assumption, $\{v_t\}_{t=1}^{\infty}$ is dense on the unit sphere, and therefore there exists some index t such that $v_t = d + \delta$, where $\delta \in \mathbb{R}^n$ is small enough to satisfy $\|\delta_i(d + \delta) + d_i\delta\| < \varepsilon/2$.

The proof may be completed as follows:

$$\begin{aligned}
\left\| \frac{H_t e_i}{\|H_t e_i\|} - w \right\| &= \|e_i - 2v_t v_t^T e_i - w\| \\
&= \|e_i - 2(d+\delta)(d+\delta)^T e_i - w\| \\
&= \|e_i - 2(d_i+\delta_i)(d+\delta) - w\| \\
&= \|e_i - 2d_i d - w - 2(\delta_i(d+\delta) + d_i\delta)\| \\
&= \|e_i - (e_i - w) - w - 2(\delta_i(d+\delta) + d_i\delta)\| \\
&= 2\,\|\delta_i(d+\delta) + d_i\delta\| \;<\; \varepsilon.
\end{aligned}$$

■

In Figure 5.1, the Householder transformation is applied to $q_{t,\ell} = (-1,-2)^T$ and produces the integer orthogonal basis $H_{t,\ell} = \begin{bmatrix} 3 & -4 \\ -4 & -3 \end{bmatrix}$.

5.2.4 The ORTHOMADS instance of MADS

The new ORTHOMADS instance of MADS can be now defined by combining the components introduced in Subsections 5.2.1–5.2.3. The POLL set P_k used by ORTHOMADS at iteration k is entirely determined by the values of the pair t_k and ℓ_k. The t_k^{th} element of the Halton sequence u_{t_k} is used to create the adjusted Halton direction q_{t_k,ℓ_k} whose norm is as close as possible to $2^{|\ell_k|/2}$. The Householder transformation on q_{t_k,ℓ_k} produces an orthogonal integer basis H_{t_k,ℓ_k}, and the norm of each column is close to $2^{|\ell_k|}$.

The LTMADS and ORTHOMADS algorithms are identical except for the construction of the set P_k and the POLL directions D_k. The set of directions $D = [I_n \; -I_n]$ defining the mesh M_k and the mesh update parameters $\tau = 4$, $w^- = -1$ and $w^+ = 1$ are the same for both algorithms. The mesh size parameter Δ_k^m and the POLL size parameter Δ_k^p are

still defined with the integer ℓ_k, except that it is allowed to be negative. This extension is not specific to ORTHOMADS and can be applied in LTMADS as well: at each iteration k, the POLL and mesh size parameters are entirely defined by the value of ℓ_k:

$$\Delta_k^p = 2^{-\ell_k} \text{ and } \Delta_k^m = \begin{cases} 4^{-\ell_k} & \text{if } \ell_k > 0 \\ 1 & \text{otherwise.} \end{cases} \tag{5.5}$$

At iteration $k = 0$, ℓ_k is set to 0 and $\Delta_0^m = \Delta_0^p = 1$. The mesh and POLL size parameters always satisfy $\Delta_k^m \leq \Delta_k^p$ and $\Delta_k^m 2^{|\ell_k|} = \Delta_k^p$.

In the update step of iteration k, if no new incumbent is found, the iteration is said to be unsuccessful and $\ell_{k+1} \leftarrow \ell_k + 1$. Otherwise, the iteration is a success and $\ell_{k+1} \leftarrow \ell_k - 1$. The MADS algorithm generates POLL trial points at a distance of order Δ_k^p from the POLL center, on a mesh M_k of size Δ_k^m. At an unsuccessful iteration, Δ_k^m is reduced faster than Δ_k^p and the number of possible POLL trial points increases, allowing more flexibility in the choice of the POLL directions D_k.

Figure 5.2 describes our algorithm. The POLL directions D_k depend entirely on the two integers t_k and ℓ_k. These integers are chosen to ensure that there will be a sequence of unsuccessful iterations for which the mesh size parameter goes to zero, and such that the directions used in that subsequence will be the tail of the entire Halton sequence. In order to accomplish that goal, we keep track of the value of the smallest POLL size parameter visited so far. At every iteration where Δ_k^p is equal to that value, we set $t_k = \ell_k + n + 1$. A consequence of this way of fixing t_k is that the set of ordered indices

$$U := \{k_1, k_2, \ldots\} = \{k \ : \ \text{iteration } k \text{ is unsuccessful, and } \Delta_k^p \leq \Delta_j^p \ \forall j = 0, 1, \ldots k\}$$

satisfies $(t_{k_1}, \ell_{k_1}) = (n+1, 0)$, $(t_{k_2}, \ell_{k_2}) = (n+2, 1), \ldots, (t_{k_i}, \ell_{k_i}) = (n+i, i-1)$, and the set of Halton directions $\{u_{t_k}\}_{k \in U}$ is precisely $\{u_t\}_{t=n+1}^{\infty}$.

At the other iterations, those for which smaller POLL sizes were previously considered,

[0] Initialization

$x_0 \in \Omega$, $\ell_0 \leftarrow 0$, $k \leftarrow 0$

[1] Iteration k

SEARCH (optional)

evaluate f on a set $S_k \subset M_k$

POLL (optional is the SEARCH was successful)

if the POLL size is the smallest one so far

(i.e., if $\Delta_k^p = \min\{\Delta_j^p : j = 0, 1, \ldots, k\}$)

$t_k \leftarrow \ell_k + n + 1$

else (i.e., smaller POLL sizes were considered)

$t_k \leftarrow \max\{t_j : j = 0, 1, \ldots, k-1\}$

compute u_{t_k}, q_{t_k,ℓ_k}, H_{t_k,ℓ_k}, and $D_k = [H_{t_k,\ell_k} \quad -H_{t_k,\ell_k}]$

evaluate f on $P_k \subset M_k$

[2] Updates

if the iteration is successful

$x_{k+1} \leftarrow x_s \in S_k$ or $x_p \in P_k$

$\ell_{k+1} \leftarrow \ell_k - 1$

else (iteration failure)

$x_{k+1} \leftarrow x_k$

$\ell_{k+1} \leftarrow \ell_k + 1$

$k \leftarrow k + 1$

goto [1] if no stopping condition is met

Figure 5.2: The ORTHOMADS algorithm.

we just keep increasing t_k so that a new Halton direction is used. Examples of pairs (t_k, ℓ_k) can be seen in Table 5.3. The boldface entries are those where the POLL size parameter is the smallest one so far. In this example, the first three indices of U would be $\{4, 5, 8\}$.

As in LTMADS, the basis H_{t_k,ℓ_k} is completed to a maximal positive basis composed of $2n$ directions,

$$D_k = [H_{t_k,\ell_k} \quad -H_{t_k,\ell_k}],$$

the set of POLL directions. A minimal positive basis with $n + 1$ directions is not considered in order to keep orthogonal directions. Table 5.4 illustrates ORTHOMADS bases

Tableau 5.3: Example of ORTHOMADS iterations for $n = 4$. Iterations $k \in \{4, 5, 8\}$ correspond to failed iterations with consecutive Halton elements $t_k = 5, 6$ and 7 satisfying $t_k = \ell_k + n + 1$.

k	Succ/Fail	$(t_k$	,	$\ell_k)$	Δ_k^m	Δ_k^p	$\|D_k e_i\|$
0	S	(**5**	,	**0**)	1	1	1
1	S	(6	,	-1)	1	2	2
2	F	(7	,	-2)	1	4	4
3	F	(8	,	-1)	1	2	2
4	F	(**5**	,	**0**)	1	1	1
5	F	(**6**	,	**1**)	1/4	1/2	2
6	S	(**7**	,	**2**)	1/16	1/4	4
7	F	(9	,	1)	1/4	1/2	2
8	F	(**7**	,	**2**)	1/16	1/4	5
9	S	(**8**	,	**3**)	1/64	1/8	8

H_{t_k,ℓ_k}, with possible pairs (t_k, ℓ_k).

Notice that any direction $D_k e_i$ $(1 \leq i \leq 2n)$ satisfies $\|D_k e_i\| = \|q_{t,\ell}\|^2 \leq (2^{|\ell|/2})^2 = 2^{|\ell|}$ and $\|D_k e_i\| \leq 2^{|\ell|}$. Therefore, the POLL trial point $x_k + \Delta_k^m D_k e_i$ is at an Euclidean distance of at most $\Delta_k^m 2^{|\ell|} = \Delta_k^p$ from the POLL center. This distance is comparable to that used in LTMADS, where the POLL trial points are exactly at a distance Δ_k^p (using the ℓ_∞ norm) from the POLL center.

We conclude this section with the following propositions that show that ORTHOMADS has the same convergence properties as in [20] with no need for a probabilistic argument.

Proposition 5.2.7 *The set of normalized directions* $\left\{\frac{q_{t,\ell}}{\|q_{t,\ell}\|}\right\}_{t=1}^{\infty}$ *with* $\ell = t - n - 1$ *is dense in the unit sphere.*

Proof. Let $\varepsilon > 0$ and $d \in \mathbb{R}^n$ with $\|d\| = 1$. Proposition 5.2.1 states that the Halton sequence $\{u_t\}_{t=1}^{\infty}$ is dense in the unit cube $[0, 1]^n$. Therefore, there exists an index t such that $\frac{2^{|t-n-1|/2}}{\sqrt{n}} - \frac{1}{2} > \frac{2\sqrt{n}}{\varepsilon} + \frac{\sqrt{n}}{2}$ and $\left\| \frac{2u_t - e}{\|2u_t - e\|} - d \right\| \leq \frac{\varepsilon}{2}$.

Tableau 5.4: A sequence of ORTHOMADS bases corresponding to seven consecutive failed iterations. Pairs (t_k, ℓ_k) correspond to consecutive Halton elements $t = 5, 6, \ldots, 12$ with $t_k = \ell_k + n + 1$.

(t_k, ℓ_k) $\|H_{t_k,\ell_k} e_i\|$	H_{t_k,ℓ_k}	(t_k, ℓ_k) $\|H_{t_k,\ell_k} e_i\|$	H_{t_k,ℓ_k}
$(5,0)$ 1	$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$(9,4)$ 14	$\begin{bmatrix} 14 & 0 & 0 & 0 \\ 0 & -4 & 12 & -6 \\ 0 & 12 & 6 & 4 \\ 0 & -6 & 4 & 12 \end{bmatrix}$
$(6,1)$ 2	$\begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 2 & 0 \\ 0 & 2 & 0 & 0 \end{bmatrix}$	$(10,5)$ 31	$\begin{bmatrix} 23 & -4 & -20 & -4 \\ -4 & 29 & -10 & -2 \\ -20 & -10 & -19 & -10 \\ -4 & -2 & -10 & 29 \end{bmatrix}$
$(7,2)$ 2	$\begin{bmatrix} 0 & 0 & 0 & 2 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 2 & 0 & 0 & 0 \end{bmatrix}$	$(11,6)$ 61	$\begin{bmatrix} 11 & -40 & 40 & -20 \\ -40 & 29 & 32 & -16 \\ 40 & 32 & 29 & 16 \\ -20 & -16 & 16 & 53 \end{bmatrix}$
$(8,3)$ 7	$\begin{bmatrix} -1 & 4 & 4 & -4 \\ 4 & 5 & -2 & 2 \\ 4 & -2 & 5 & 2 \\ -4 & 2 & 2 & 5 \end{bmatrix}$	$(12,7)$ 123	$\begin{bmatrix} 25 & -98 & 0 & 70 \\ -98 & 25 & 0 & 70 \\ 0 & 0 & 123 & 0 \\ 70 & 70 & 0 & 73 \end{bmatrix}$

Lemma 5.2.3 ensures that $\alpha_{t,\ell} \geq \frac{2^{|\ell|/2}}{\sqrt{n}} - \frac{1}{2} > \frac{2\sqrt{n}}{\varepsilon} + \frac{\sqrt{n}}{2}$. Combining this last inequality with Proposition 5.2.4 gives

$$\begin{aligned} \left\| \frac{q_{t,\ell}}{\|q_{t,\ell}\|} - d \right\| &\leq \left\| \frac{q_{t,\ell}}{\|q_{t,\ell}\|} - \frac{2u_t - e}{\|2u_t - e\|} \right\| + \left\| \frac{2u_t - e}{\|2u_t - e\|} - d \right\| \\ &< \frac{\varepsilon}{2} + \frac{\varepsilon}{2} = \varepsilon \,. \end{aligned}$$

■

This allows us to state our main result:

Theorem 5.2.8 ORTHOMADS *is a valid* MADS *instance.*

Proof. In order to show that ORTHOMADS is a valid MADS instance we need to show that the POLL directions satisfy the following four properties [16, 20]:

- *Any direction $D_k e_i$ ($1 \leq i \leq 2n$) can be written as a nonnegative integer combination of the directions of D*: This is the case by construction.

- *The distance from the* POLL *center x_k to a* POLL *trial point (in ℓ_∞ norm) has to be bounded above by Δ_k^p*: This is also the case by construction because we ensured that $\|D_k e_i\| \leq 2^{|\ell_k|}$ for all i in $\{1, 2, ..., 2n\}$ and $\|\Delta_k^m D_k e_i\|_\infty \leq \|\Delta_k^m D_k e_i\| \leq \Delta_k^m 2^{|\ell_k|} = \Delta_k^p$.

- *Limits (as defined in [40]) of convergent subsequences of the normalized sets $\overline{D_k} = \{d/\|d\| : d \in D_k\}$ are positive spanning sets.* This can be shown the same way as in [16] where the proof for LTMADS is detailed, since, for ORTHOMADS and with $\overline{H_{t_k,\ell_k}} = \{d/\|d\| : d \in H_{t_k,\ell_k}\}$, $\det\left(\overline{H_{t_k,\ell_k}}\right) = -1$.

- *The set of normalized directions used over all failed iterations is dense in the unit sphere*: The strategy chosen for the values of t_k and ℓ_k ensures that there

exists a sequence of failed iterations corresponding to consecutive values of t_k. These iterations $k \in U$ can be chosen to correspond to large values of ℓ_k because, from [20], $\lim_{\substack{k \in U \\ k \to \infty}} \Delta_k^m = 0$, and $\Delta_k^m = 4^{-\ell_k}$ for $\ell_k \geq 0$. For $k \in U$, the sets of directions $\{D_k\}_{k \in U}$ are constructed from consecutive directions q_{t_k,ℓ_k}, which are dense in the unit sphere after normalization (Proposition 5.2.7). Then, from Proposition 5.2.6 and since $D_k = [H_{t_k,\ell_k} - H_{t_k,\ell_k}]$, the set of normalized directions $\left\{ \frac{D_k e_i}{\|D_k e_i\|} \right\}_{k \in U}$ is also dense in the unit sphere for all $i = 1, 2, \ldots, 2n$.

■

5.3 Numerical Tests

In this section, ORTHOMADS is compared to its predecessor LTMADS [20] and to the GPS method [120], on 45 problems from the literature. In the MADS algorithms, the theory supports handling constraints by the extreme barrier approach: Points outside Ω are simply ignored and f is not evaluated. For GPS, the extreme barrier approach is supported by the theory only for a finite number of linear constraints [87]. Still, for comparison, we apply two different approaches: the extreme barrier (GPS-EB), and the filter method described in [19] (GPS-FILTER), which has stronger theoretical support.

Because of its random behavior, 30 instances of LTMADS are performed for each problem. GPS and ORTHOMADS are scored by comparing them against the 30 LTMADS instances. A score of s for GPS or ORTHOMADS means that this instance gave a value of f at least as good as s of the 30 LTMADS instances, with a relative precision of 1%. The worst score is 0 and the perfect score corresponds to 30. We consider that a bad instance has a score less than 10, an acceptable instance is between 10 and 19, and a good instance has a score greater than or equal to 20. This score criteria is not a perfect measure, in terms of objective comparison between deterministic and random methods.

In particular, if the 30 LTMADS instances gave the same result as the ORTHOMADS solution, then the ORTHOMADS score would be classified as perfect. More exhaustive tests will be conducted in future work.

The integer ℓ_k (see (5.5)), defining the mesh and POLL size parameters Δ_k^m and Δ_k^p at iteration k, is allowed to be negative for both LTMADS and ORTHOMADS. Maximal positive bases ($2n$ directions) are used in the three methods, as is the opportunistic strategy (the POLL is interrupted at the first success), and the optimistic strategy: after a successful point has been found, a SEARCH point is generated further along the same direction. No other SEARCH is performed. The stopping criteria is satisfied when the number of function evaluations reaches $1000n$ (or when the POLL size parameter Δ_k^p drops below 1E-12).

The methods are tested on 45 problems divided into 4 groups: our choice of smooth and nonsmooth unconstrained problems is the same as in [44] and [43], respectively, with 21 smooth problems from the CUTEr test set [63] and 13 nonsmooth problems from [93], which is a compilation of nonsmooth problems from the literature. We also tested on 9 constrained problems from [21,22,93], and in addition, we added two problems from [12] that correspond to real applications.

All results and problem descriptions are summarized in Tables 5.5–5.9, where $f(x^*)$ corresponds to the best known minimal value of f, $value$ to the final value of f for each method, and $evals$ to the number of function evaluations that each method performed. Tables 5.5 and 5.6 show results on the 21 unconstrained smooth problems from CUTEr. ORTHOMADS has a perfect score on 17 of these problems. Table 5.7 displays results on the 13 unconstrained nonsmooth problems, where ORTHOMADS achieves good scores on 7 problems. Table 5.8 shows results for the 9 constrained problems. The same number of problems (4) is considered good and bad for ORTHOMADS. Finally, Table 5.9 presents results for the two real applications, and ORTHOMADS has perfect scores on both of them.

Tableau 5.5: CUTEr unconstrained smooth problems (1 of 2). A score of s for a method indicates that the final f value is at least as good as s of the 30 LTMADS runs (with a relative error of 1%).

Problem	LTMADS×30		GPS		ORTHOMADS	
	worst	best				
n $f(x^*)$	*evals*	*evals*	*evals*	*score*	*evals*	*score*
	value	*value*	*value*		*value*	
ARWHEAD	6128	650	1039	30	660	30
10 0.00	0.00	0.00	0.00		0.00	
ARWHEAD	20000	1285	2079	30	1320	30
20 0.00	0.00	0.00	0.00		0.00	
BDQRTIC	6320	4497	3510	30	5763	30
10 11.9	18.3	18.3	18.3		18.3	
BDQRTIC	20000	17884	17074	30	20000	30
20 35.4	58.3	58.3	58.3		58.3	
BIGGS6	831	570	764	30	713	30
6 0.00	2.06	2.06	2.06		2.06	
BROWNAL10	10000	10000	10000	1	10000	3
10 0.00	0.07	0.00	0.06		0.04	
BROWNAL20	20000	20000	20000	3	20000	20
20 0.00	0.34	0.00	0.16		0.01	
PENALTY1	10000	10000	10000	0	10000	30
10 7.09E-5	7.09E-5	7.09E-5	8.82E-5		7.09E-5	
PENALTY1	20000	20000	20000	0	20000	30
20 1.58E-4	1.58E-4	1.58E-4	1.88E-4		1.58E-4	
PENALTY2	10000	10000	10000	30	10000	30
10 0.294E-3	1.280E-3	1.241E-3	1.243E-3		1.250E-3	
PENALTY2	20000	20000	20000	0	20000	30
20 0.829E-2	1.080E-2	1.078E-2	1.152E-2		1.079E-2	

... continued on Table 5.6

Tableau 5.6: CUTEr unconstrained smooth problems (2 of 2). A score of s for a method indicates that the final f value is at least as good as s of the 30 LTMADS runs (with a relative error of 1%).

Problem	LTMADS×30		GPS		ORTHOMADS	
	worst	best				
n $f(x^*)$	*evals*	*evals*	*evals*	*score*	*evals*	*score*
	value	*value*	*value*		*value*	
POWELLSG	12000	12000	10093	30	12000	30
12 0.00	0.00	0.00	0.00		0.00	
POWELLSG	20000	20000	20000	30	20000	30
20 0.00	0.00	0.00	0.00		0.00	
SROSENBR	10000	10000	10000	30	10000	25
10 0.00	6.31	0.00	0.00		0.06	
SROSENBR	20000	20000	20000	30	1958	30
20 0.00	16.52	0.48	0.00		0.00	
TRIDIA	10000	7591	9317	30	10000	30
10 0.00	0.00	0.00	0.00		0.00	
TRIDIA	20000	20000	20000	30	20000	30
20 0.00	0.00	0.00	0.00		0.00	
VARDIM	10000	8163	10000	0	10000	30
10 0.00	0.00	0.00	4.01		0.00	
VARDIM	20000	20000	20000	0	20000	0
20 0.00	0.00	0.00	110.59		110.84	
WOODS	10951	7327	8433	30	9479	30
12 0.00	104.91	104.91	104.91		104.91	
WOODS	20000	19181	20000	30	20000	30
20 0.00	174.84	174.84	174.84		174.84	
			average scores	20.2		26.6

Tableau 5.7: Results for unconstrained nonsmooth problems from [93]. A score of s for a method indicates that the final f value is at least as good as s of the 30 LTMADS runs (with a relative error of 1%).

Problem	LTMADS×30		GPS		ORTHOMADS	
	worst	best				
n $f(x^*)$	*evals*	*evals*	*evals*	*score*	*evals*	*score*
	value	*value*	*value*		*value*	
ELATTAR	456	1795	2392	16	3984	20
6 0.560	8.021	0.563	1.714		1.504	
EVD61	490	3280	920	5	4224	22
6 0.0349	1.6001	0.0417	0.5443		0.0709	
FILTER	1293	1761	1132	7	1332	9
9 0.00619	0.00971	0.00797	0.00950		0.00935	
GOFFIN	50000	50000	24097	30	16842	30
50 0.00	1.10	0.06	0.00		0.00	
HS78	403	1026	819	13	405	13
5 −2.92	10.00	−2.88	0.00		0.00	
L1HILB	17953	50000	8738	0	50000	14
50 0.00	1.84	0.04	3.95		0.22	
MXHILB	11523	20377	9384	0	20755	3
50 0.00	0.280	0.003	0.976		0.197	
OSBORNE2	2046	5414	1660	0	4555	20
11 0.0480	0.1703	0.0549	0.2799		0.1089	
PBC1	1211	1127	677	2	1291	17
5 0.0223	0.4146	0.0343	0.3845		0.1602	
POLAK2	1449	1742	1327	30	949	30
10 54.6	54.6	54.6	54.6		54.6	
SHOR	1345	882	787	0	1087	30
5 22.6	22.9	22.6	23.5		22.8	
WONG1	1161	2109	1100	30	1823	30
7 681	699	693	697		693	
WONG2	5403	5403	1871	0	4977	0
10 24.3	31.4	24.8	47.4		32.8	
			average scores	10.2		18.3

Tableau 5.8: Results for constrained problems. A score of s for a method indicates that the final f value is at least as good as s of the 30 LTMADS runs (with a relative error of 1%).

Problem	LTMADS×30		GPS-FILTER		GPS-EB		ORTHOMADS	
	worst	best						
n m $f(x^*)$	*evals*	*evals*	*evals*	*score*	*evals*	*score*	*evals*	*score*
	value	*value*	*value*		*value*		*value*	
CRESCENT10 [21]	1279	4473	2152	0	1172	0	5497	30
10 2 −9.00	−8.26	−8.95	−6.19		−2.32		−8.97	
DISK10 [21]	1909	2626	2322	0	1143	0	2359	30
10 1 −17.3	−17.2	−17.3	−13.0		−10.0		−17.2	
B250 [22]	60000	60000	15412	0	27773	0	60000	0
60 1 7.95	15.41	7.99	1142.01		1116.03		16.92	
B500 [22]	16705	15359	11189	0	18912	0	29858	6
60 1 104	557	104	1235		1254		277	
G2 [22]	3880	6461	2056	4	2689	29	5414	22
10 2 −0.728	−0.181	−0.728	−0.221		−0.706		−0.561	
G2 [22]	10877	15722	6376	3	6551	29	20000	29
20 2 −0.804	−0.203	−0.736	−0.241		−0.721		−0.711	
HS114 [93]	1506	2135	1756	0	1756	0	1661	4
9 6 −1769	−1012	−1312	−968		−968		−1016	
MAD6 [93]	1122	1542	1378	22	1378	22	1671	7
5 7 0.102	0.113	0.102	0.103		0.103		0.108	
PENTAGON [93]	859	2525	601	0	601	0	980	19
6 15 −1.86	−1.60	−1.86	0.00		0.00		−1.81	
			average scores	3.2		8.9		16.3

Tableau 5.9: Results for real applications. A score of s for a method indicates that the final f value is at least as good as s of the 30 LTMADS runs (with a relative error of 1%). Displayed z values for problem STY are divided by 10^7.

Problem	LTMADS×30 worst	LTMADS×30 best	GPS-FILTER		GPS-EB		ORTHOMADS	
n m $f(x^*)$	*evals* *value*	*evals* *value*	*evals* *value*	*score*	*evals* *value*	*score*	*evals* *value*	*score*
MDO [12]	1767	15	2719	0	2048	0	1212	30
10 10 −3964	−2530	−3964	−1386		−1386		−3964	
STY [12]	1590	1189	2073	22	2113	0	1214	30
8 11 −3.35	−2.88	−3.29	−3.11		−2.82		−3.27	
average scores				11.0		0.0		30.0

Table 5.10 summarizes the results. The first observation is that both MADS instances outperform GPS. For 25 problems out of 45 (20 without the relative error of 1%), ORTHOMADS found the same solution as the best of 30 LTMADS runs. For 20 of these 25 problems, the ORTHOMADS solution is the same as the 30 LTMADS instances. The new method solved 32 problems out of 45 problems efficiently enough that, for these problems, the single run of ORTHOMADS was better than two thirds of the 30 LTMADS runs. For 4 problems, the two methods performed equally well, and for 9 problems, at least two thirds of the LTMADS runs gave a better solution than the one produced by ORTHOMADS.

Figure 5.3 illustrates the spread of the directions for both LTMADS and ORTHOMADS. Rosenbrock's function [108] with $n = 2$ and $n = 3$ was used with 2000 and 3000 evaluations, respectively. In the two-dimensional case, all the normalized directions used to generate POLL trial points are directly represented on the top two subfigures. It is clear that ORTHOMADS directions are well distributed on the unit circle. This is not the case with LTMADS because half the directions correspond to either $\pm e_1$ or $\pm e_2$. For $n = 3$, the two plots on the bottom represent the standard angles of the normalized directions in spherical coordinates. There again it can be seen that ORTHOMADS directions have

Tableau 5.10: Summary for the GPS and ORTHOMADS performances. F, E and O correspond respectively to GPS-FILTER, GPS-EB, and ORTHOMADS. A bad instance has a score between 0 and 9, an acceptable (acc.) instance a score between 10 and 19, a good instance a score higher than 20 and a perfect (perf.) instance has a score of 30.

problems	average scores (on 30)			# of problems	# of bad instances			# of acc. instances			# of good instances			# of perf. instances		
	F	E	O		F	E	O	F	E	O	F	E	O	F	E	O
smooth	20.2	20.2	26.6	21	7	7	2	0	0	0	14	14	19	14	14	17
nonsmooth	10.2	10.2	18.3	13	8	8	3	2	2	3	3	3	7	3	3	4
constrained	3.2	8.9	16.3	9	8	6	4	0	0	1	1	3	4	0	0	2
real appli.	11.0	0.0	30.0	2	1	2	0	0	0	0	1	0	2	0	0	2
total or avg	13.5	14.2	22.3	45	24	23	9	2	2	4	19	20	32	17	17	25

a better distribution than those of LTMADS, since at least two thirds of the LTMADS directions possess some null coordinates. On the subfigure using LTMADS with $n = 3$, the horizontal bar at $\Phi = \pi/2$ corresponds to the set of directions where $z = 0$. The vertical bars at $\theta = \pm\pi/2$ correspond to directions with $x = 0$, and the one at $\theta = 0$ and $\theta = \pi$ correspond to directions with $y = 0$.

5.4 Discussion

This paper introduced ORTHOMADS, an alternative instantiation of the MADS class of algorithms. The advantages of ORTHOMADS over the original LTMADS are that the MADS directions are chosen deterministically, and that those directions are orthogonal to each other. Moreover, ORTHOMADS inherits all of the MADS convergence properties, without probabilistic arguments, and without additional parameters.

Intensive tests on 45 problems from the literature showed that both MADS instances outperform the GPS algorithm, and that ORTHOMADS is competitive with LTMADS, with a better distribution of the POLL directions.

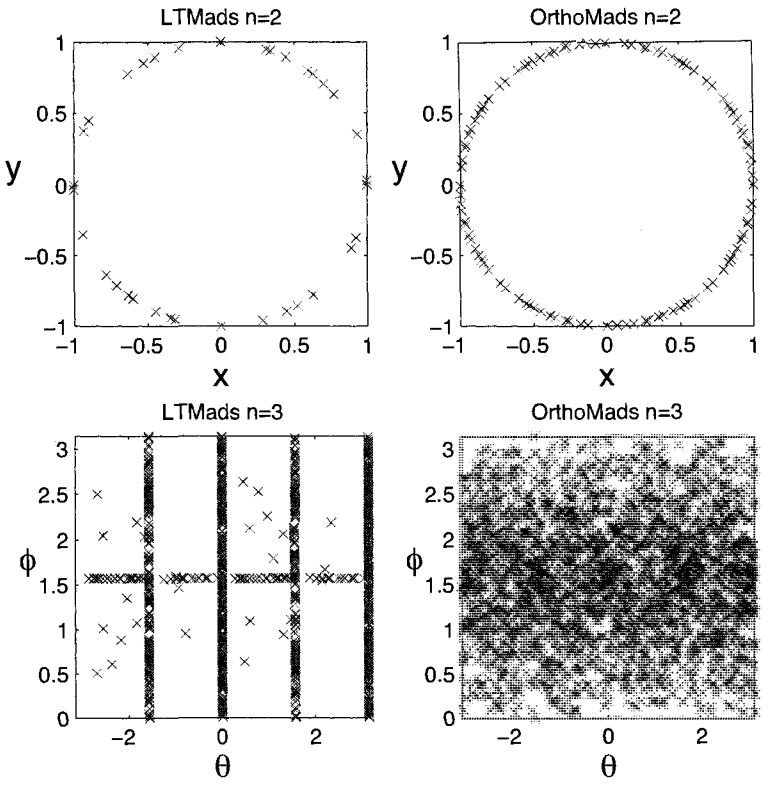

Figure 5.3: LTMADS and ORTHOMADS normalized POLL directions on the Rosenbrock function with $n = 2$ and $n = 3$.

Acknowledgements

We would like to thank Ana Custódio and Luis Vicente for helpful remarks about missed cones of directions in single iterations of LTMADS. These remarks and similar observations of our own helped motivate us to discover ORTHOMADS. We also wish to thank Andrew Booker for suggesting quasi-Monte Carlo methods which started us in directions that led to our use of Halton sequences.

DISCUSSION GÉNÉRALE ET CONCLUSION

Nous discutons dans cette dernière partie des résultats obtenus dans chaque article des chapitres 3 (MADS-VNS), 4 (PSD-MADS) et 5 (ORTHOMADS), et comparons ces résultats aux attentes que nous avions.

Concernant le couplage de MADS avec la méta-heuristique VNS, en plus d'avoir introduit une nouvelle méthode de recherche globale générique et d'avoir introduit VNS dans un contexte inhabituel, les résultats sont conformes aux attentes, c'est-à-dire qu'on a constaté une plus grande stabilité dans les solutions obtenues. En effet, tout en conservant un caractère non déterministe du fait de l'utilisation à la fois de LTMADS et de perturbations aléatoires, des séries de trente exécutions donnent en moyenne de meilleurs résultats. Ceci signifie également qu'une seule exécution de l'algorithme aura plus de chance de donner une bonne solution. Cette plus grand stabilité vient toutefois au prix d'un nombre plus important d'évaluations, ce surcoût d'évaluations étant néanmoins mieux utilisé par VNS que par des recherches globales plus classiques.

Pour le nouvel algorithme parallèle PSD-MADS décrit au chapitre 4, l'objectif était double : on voulait tout d'abord concevoir une parallélisation asynchrone de MADS, et ensuite que cette nouvelle méthode soit efficace pour traiter des problèmes de grande taille (au delà de cent variables). Ces deux objectifs ont été atteints : la méthode PSD-MADS est parallèle et asynchrone (les processeurs utilisés ne sont jamais mis en attente), et nous avons obtenu des résultats satisfaisants pour des problèmes allant jusqu'à 500 variables. L'algorithme a également été comparé à APPS, la version parallèle asynchrone de GPS déjà évoquée en 1.1, face auquel il s'est montré compétitif.

Enfin, la méthode ORTHOMADS du chapitre 5 propose une toute nouvelle implémentation déterministe de MADS et utilisant des directions orthogonales. L'objectif d'obtenir un algorithme favorisant la reproductibilité des expériences et utilisant de meilleures di-

rections a donc été encore une fois atteint. De plus, des tests intensifs sur 45 problèmes ont montré que ORTHOMADS fait au moins jeu égal avec LTMADS. Ces tests confirment aussi encore une fois que MADS et ses deux implémentations existantes, est plus performant que GPS.

L'objectif de cette thèse était d'apporter des améliorations à MADS pour l'optimisation de problèmes non lisses du type de $\mathcal{P}$. Les trois articles exposés dans ce document apportent trois extensions à MADS, renforçant son efficacité. De plus, une des forces de MADS est que l'algorithme possède une certaine simplicité, et que les paramètres sont peu nombreux et possèdent des valeurs par défaut. Nos trois extensions conservent cet esprit de simplicité et n'ajoutent qu'un nombre raisonnable de paramètres. Ajoutons également que les premier et troisième articles ont démontré l'efficacité des algorithmes de type MADS sur un problème caractéristique d'une situation réelle, typique de ce que les ingénieurs veulent étudier, et parfait représentant des problèmes cibles de cette thèse et du domaine de l'optimisation non lisse.

L'apport de MADS en 2006 au domaine de l'optimisation non lisse avait déjà été important, car il généralisait et étendait GPS, une méthode phare du domaine, tout en proposant une analyse de convergence basée sur le calcul non lisse de Clarke. Son succès avait été dû à la fois à un bon comportement pratique et théorique. Cette importante contribution au domaine a d'ailleurs été reconnue par le fait que la méthode fait partie du populaire logiciel MATLAB [119]. Toute amélioration avérée de MADS, comme cette thèse, constitue donc un apport important au domaine de l'optimisation non lisse à l'aide de méthodes de recherche directe. Les trois extensions présentées sont d'ailleurs en cours d'intégration dans le logiciel libre NOMAD [15] afin d'être rendues disponibles à la communauté.

Tandis que la recherche globale de type VNS et l'implémentation ORTHOMADS (chapitres 3 et 5) sont des travaux terminés, la parallélisation asynchrone de MADS suivant une distribution des variables, telle que celle présentée au chapitre 4, peut encore lar-

gement évoluer. Par exemple, le choix des groupes de variables distribuées à chaque processeur, actuellement décidé au hasard, peut être l'objet de nouvelles recherches. Également, les trois extensions présentées ont été étudiées de façon indépendante. Une prochaine étape pourrait être de les tester ensemble, et conjointement avec l'utilisation de la nouvelle méthode de gestion des contraintes, la barrière progressive de [21]. Enfin, d'autres extensions de MADS non étudiées dans cette thèse peuvent être envisagées. Par exemple l'utilisation des progrès récents effectués par les méthodes DFO dans la construction dynamique de fonctions substitut.

Pour finir, il est apparu au cours de notre travail qu'il n'y a pas encore dans la littérature d'étude complète et exhaustive permettant, pour un large éventail de problèmes, de comparer les méthodes de recherche directe, les méthodes d'optimisation sans dérivées, les méthodes traditionnelles de l'optimisation lisse et les méta-heuristiques.

RÉFÉRENCES

[1] ABRAMSON M.A., NOMADm Optimization Software, logiciel disponible à www.afit.edu/en/ENC/Faculty/MAbramson/NOMADm.html.

[2] ABRAMSON M.A. (2002), *Pattern Search Algorithms for Mixed Variable General Constrained Optimization Problems*. Thèse, Department of Computational and Applied Mathematics, Rice University.

[3] ABRAMSON M.A. (2004), Mixed Variable Optimization of a Load-Bearing Thermal Insulation System Using a Filter Pattern Search Algorithm, *Optimization and Engineering*, vol.5, no.2, p.157–177.

[4] ABRAMSON M.A. et AUDET C. (2006), Second-order convergence of mesh-adaptive direct search, SIAM *Journal on Optimization*, vol.17, no.2, p.606–619.

[5] ABRAMSON M.A., AUDET C. et DENNIS J.E. Jr. (2004), Generalized pattern searches with derivative information, *Mathematical Programming*, vol.100, p.3–25.

[6] ABRAMSON M.A., AUDET C., DENNIS J.E. Jr. et LE DIGABEL S. (2008), ORTHOMADS : A deterministic MADS instance with orthogonal directions. Rapport technique G-2008-15, Les Cahiers du GERAD.

[7] ABRAMSON M.A., BREZHNEVA O.A., DENNIS J.E. Jr. et PINGEL R.L. (2008), Pattern search in the presence of degenerate linear constraints. À paraître dans *Optimization Methods and Software*.

[8] ALBERTO P., NOGUEIRA F., ROCHA U. et VICENTE L.N (2004), Pattern search methods for user-provided points : application to molecular geometry problems, SIAM *Journal on Optimization*, vol.14, no.4, p.1216–1236.

[9] ALEXANDROV N., DENNIS J.E. Jr., LEWIS R. et TORCZON V. (1998), A trust region framework for managing the use of approximation models in optimization, *Structural Optimization*, vol.15, p.16–23.

[10] AUDET C. (2004), Convergence Results for Pattern Search Algorithms are Tight, *Optimization and Engineering*, vol.5, no.2, p.101–122.

[11] AUDET C., BÉCHARD V. et CHAOUKI J. (2008), Spent potliner treatment process optimization using a MADS algorithm. À paraître dans *Optimization and Engineering*, DOI : 10.1007/s11081-007-9030-2.

[12] AUDET C., BÉCHARD V. et LE DIGABEL S. (2008), Nonsmooth Optimization through Mesh Adaptive Direct Search and Variable Neighborhood Search, *Journal of Global Optimization*, vol.41, no.2, p.299–318.

[13] AUDET C., BOOKER A.J., DENNIS J.E.Jr., FRANK P.D. et MOORE D.W. (2000), A surrogate-model-based method for constrained optimization, AIAA Paper 2000–4891, AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, Long Beach, Californie.

[14] AUDET C., BRIMBERG J., HANSEN P., LE DIGABEL S. et MLADENOVIĆ N. (2004), Pooling Problem : Alternate Formulations and Solution Methods, *Management Science*, vol.50, no.6, p.761–776.

[15] AUDET C., COUTURE G. et DENNIS J.E. Jr., NOMAD project (LTMADS package), logiciel disponible à www.gerad.ca/nomad.

[16] AUDET C., CUSTÓDIO A.L. et DENNIS J.E. Jr. (2008), Erratum : Mesh adaptive direct search algorithms for constrained optimization, SIAM *Journal on Optimization*, vol.18, no.4, p.1501–1503.

[17] AUDET C. et DENNIS J.E. Jr. (2000), Pattern search algorithms for mixed variable programming, SIAM *Journal on Optimization*, vol.11, no.3, p.573–594.

[18] AUDET C. et DENNIS J.E. Jr. (2003), Analysis of generalized pattern searches, SIAM *Journal on Optimization*, vol.13, no.3, p.889–903.

[19] AUDET C. et DENNIS J.E. Jr. (2004), A pattern search filter method for nonlinear programming without derivatives, SIAM *Journal on Optimization*, vol.14, no.4, p.980–1010.

[20] AUDET C. et DENNIS J.E. Jr. (2006), Mesh Adaptive Direct Search Algorithms for constrained optimization, SIAM *Journal on Optimization*, vol.17, no.1, p.188–217.

[21] AUDET C. et DENNIS J.E. Jr. (2007), A MADS Algorithm with a Progressive Barrier for Derivative-Free Nonlinear Programming. Rapport technique G-2007-37, Les Cahiers du GERAD.

[22] AUDET C., DENNIS J.E. Jr. et LE DIGABEL S. (2008), Parallel Space Decomposition of the Mesh Adaptive Direct Search Algorithm. À paraître dans SIAM *Journal on Optimization*.

[23] AUDET C. et ORBAN D. (2006), Finding Optimal Algorithmic Parameters Using the Mesh Adaptive Direct Search Algorithm, SIAM *Journal on Optimization*, vol.17, no.3, p.642–664.

[24] BEAL J.M, SHUKLA A., BREZHNEVA O.A. et ABRAMSON M.A. (2008), Optimal sensor placement for enhancing sensitivity to change in stiffness for structural health monitoring. À paraître dans *Optimization and Engineering*.

[25] BERTSEKAS D.P. et TSITSIKLIS J.N. (1989), *«Parallel and distributed computation : numerical methods»*. Prentice-Hall, Inc., Upper Saddle River, NJ.

[26] BOOKER A.J., CRAMER E.J., FRANK P.D., GABLONSKY J.M. et DENNIS J.E. Jr. (2007), MoVars : Multidisciplinary Optimization Via Adaptive Response Surfaces, AIAA Paper 2007–1927, AIAA/ASME/ASCE/AHS/ ASC Structures, Structural Dynamics, and Materials, Honolulu.

[27] BOOKER A.J., DENNIS J.E. Jr., FRANK P.D., MOORE D.W. et SERAFINI D.B. (1998), Managing surrogate objectives to optimize a helicopter rotor design – further experiments, AIAA Paper 1998–4717, AIAA/ISSMO Symposium on Multidisciplinary Analysis and Optimization, St. Louis.

[28] BOOKER A.J., DENNIS J.E. Jr., FRANK P.D., SERAFINI D.B. et TORCZON V. (1998), Optimization using surrogate objectives on a helicopter test example, dans *«Optimal Design and Control»* (Cambridge, MA), J.Borggaard, J.Burns, E.Cliff, et S.Schreck, Eds., Progress in Systems and Control Theory, Birkhäuser, p.49–58.

[29] BOOKER A.J., DENNIS J.E. Jr., FRANK P.D., SERAFINI D.B., TORCZON V. et TROSSET M.W. (1999), A rigorous framework for optimization of expensive functions by surrogates, *Structural Optimization*, vol.17, no.1, p.1–13.

[30] BOOKER A.J., MECKESHEIMER M. et TORNG T. (2004), Reliability Based Design Optimization Using Design Explorer, *Optimization and Engineering*, vol.5, no.2, p.179–206.

[31] BORTZ D.M. et KELLEY C.T. (1998), The simplex gradient and noisy optimization problems, dans *«Optimal Design and Control»* (Cambridge, MA), J.Borggaard, J.Burns, E.Cliff, et S.Schreck, Eds., Progress in Systems and Control Theory, Birkhäuser, p.77–90.

[32] BOWDEN R.O. et HALL J.D. (1998), Simulation optimization research and development, Winter Simulation Conference, 1693–1698.

[33] BRIMBERG J. et MLADENOVIĆ N (1996), A Variable Neighbourhood Algorithm for Solving the Continuous Location-Allocation Problem, dans *«Studies in Location Analysis»*, D.Hamacher, Ed., 10, p.1–12, Athène, Grèce.

[34] CAPOROSSI G. et HANSEN P. (2000), Variable neighborhood search for extremal graphs 1 : The AutoGraphiX system, *Discrete Mathematics*, vol.212, p.29–44.

[35] CLARKE F.H. (1983), *«Optimization and Nonsmooth Analysis»*. Wiley, New York. Re-édité en 1990 pas SIAM Publications, Philadelphie, comme Volume 5 de la collection Classics in Applied Mathematics.

[36] CONN A.R., SCHEINBERG K. et TOINT Ph.L. (1997), On the convergence of derivative-free methods for unconstrained optimization, dans *«Approximation Theory and Optimization : Tributes to M.J.D. Powell»*, M.D. Buhmann et A.Iserles, Eds. Cambridge University Press, Cambridge, United Kingdom, p.83–108.

[37] CONN A.R., SCHEINBERG K. et TOINT Ph.L. (1997), Recent progress in unconstrained nonlinear optimization without derivatives, *Mathematical Programming*, vol.79, p.397–414.

[38] CONN A.R., SCHEINBERG K. et VICENTE L.N. (2008), Geometry of interpolation sets in derivative free optimization, *Mathematical Programming*, vol.111, p.141–172.

[39] CONN A.R., SCHEINBERG K. et VICENTE L.N. (2008), Geometry of sample sets in derivative free optimization : Polynomial regression and underdetermined interpolation. À paraître dans IMA *Journal of Numerical Analysis*.

[40] COOPE I.D. et PRICE C.J. (2000), Frame-based methods for unconstrained optimization, *Journal of Optimization Theory and Applications*, vol.107, no.2, p.261–274.

[41] CORMEN T.H., LEISERSON C.E., RIVEST R.L. et STEIN C. (2001), *«Introduction to Algorithms, Second Edition»*. MIT Press and McGraw-Hill, p.859–861.

[42] CUSTÓDIO A.L. (2007), *Aplicações de Derivadas Simplécticas em Métodos de Procura Directa*. Thèse, New University of Lisbon.

[43] CUSTÓDIO A.L., DENNIS J.E. Jr. et VICENTE L.N. (2008), Using Simplex Gradients of Nonsmooth Functions in Direct Search Methods. À paraître dans IMA *Journal of Numerical Analysis*.

[44] CUSTÓDIO A.L. et VICENTE L.N. (2007), Using Sampling and Simplex Derivatives in Pattern Search Methods, SIAM *Journal on Optimization*, vol.18, p.537–555.

[45] DAVIDON W.C. (1991), Variable metric method for minimization, SIAM *Journal on Optimization*, vol.1, no.1, p.1–17.

[46] DAVIS C. (1954), Theory of positive linear dependence, *American Journal of Mathematics*, vol.76, p.733–746.

[47] DENNIS J.E. Jr. et TORCZON V. (1991), Direct search methods on parallel machines, SIAM *Journal on Optimization*, vol.1, no.4, p.448–474.

[48] DENNIS J.E. Jr. et TORCZON V. (1997), Managing approximation models in optimization, dans *«Multidisciplinary Design Optimization : State of the Art»*, N.M. Alexandrov et M.Y. Hussaini, Eds. SIAM, Philadelphia, p.330–347.

[49] DENNIS J.E. Jr. et WU Z. (2003), Parallel continuous optimization, dans *«Sourcebook of parallel computing»*. Morgan Kaufmann Publishers Inc., San Francisco, CA, p.649–670.

[50] DOUGLAS J.M. (1988), *«Conceptual Design of Chemical Processes»*. McGraw-Hill, New York.

[51] DRAZIĆ M., LAVOR C., MACULAN N. et MLADENOVIĆ N. (2004), A Continuous VNS Heuristic for Finding the Tridimensional Structure of a Molecule. Rapport technique G–2004–22, Les Cahiers du GERAD, Montréal.

[52] EDGAR T.F., HIMMELBLAU D.M. et LASDON L.S. (2003), *«Optimization of chemical processes 2nd ed.»*. McGraw-Hill, New York.

[53] FERRIS M.C. et MANGASARIAN O.L. (1994), Parallel variable distribution, SIAM *Journal on Optimization*, vol.4, p.815–832.

[54] FINKEL D.E. et KELLEY C.T. (2004), Convergence analysis of the DIRECT algorithm. Rapport technique CRSC-TR04-28, Center for Research in Scientific Computation.

[55] FLETCHER R., GOULD N.I.M., LEYFFER S., TOINT Ph.L. et WÄCHTER A. (2002), On the global convergence of trust-region SQP-filter algorithms for

general nonlinear programming, SIAM *Journal on Optimization*, vol.13, no.3, p.635–659.

[56] FLETCHER R. et LEYFFER S. (2002), Nonlinear programming without a penalty function, *Mathematical Programming*, vol.91, p.239–269.

[57] FLETCHER R., LEYFFER S. et TOINT Ph.L. (2002), On the global convergence of a filter–SQP algorithm, SIAM *Journal on Optimization*, vol.13, no.1, p.44–59.

[58] FOWLER K.R., KELLEY C.T., MILLER C.T., KEES C.E., DARWIN R.W., REESE J.P., FARTHING M.W. et REED M.S.C. (2004), Solution of a Well-Field Design Problem with Implicit Filtering, *Optimization and Engineering*, vol.5, no.2, p.207–234.

[59] FOWLER K.R., REESE J.P., KEES C.E., DENNIS J.E. Jr., KELLEY C.T., MILLER C.T., AUDET C., BOOKER A.J., COUTURE G., DARWIN R.W., FARTHING M.W., FINKEL D.E., GABLONSKY J.M., GRAY G.A. et KOLDA T.G. (2008), Comparison of derivative-free optimization methods for groundwater supply and hydraulic capture community problems. À paraître dans *Advances in Water Resources*.

[60] FROMMER A. et RENAUT R.A. (1999), A unified approach to parallel space decomposition methods, *Journal of Computational and Applied Mathematics*, vol.110, p.205–233.

[61] FUKUSHIMA M. (1998), Parallel variable transformation in unconstrained optimization, SIAM *Journal on Optimization*, vol.8, no.3, p.658–672.

[62] GABLONSKY J. et KELLEY C.T. (2001), A locally-biased form of the DIRECT algorithm, *Journal of Global Optimization*, vol.21, p.27–37.

[63] GOULD N. I.M., ORBAN D. et TOINT Ph.L. (2003), CUTEr (and SifDec) : a Constrained and Unconstrained Testing Environment, revisited, ACM *Transactions on Mathematical Software*, vol.29, no.4, p.373–394.

[64] GRAY G.A. et KOLDA T.G. (2006), Algorithm 856 : APPSPACK 4.0 : Asynchronous parallel pattern search for derivative-free optimization., ACM *Transactions on Mathematical Software*, vol.32, no.3, p.485–507.

[65] HALTON J.H. (1960), On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals, *Numerische Mathematik*, vol.2, no.1, p.84–90.

[66] HAN S.-P. (1986), Optimization by updated conjugate subspaces, *Numerical Analysis*, vol.140, p.82–97.

[67] HANSEN P. et MLADENOVIĆ N. (2001), J-MEANS : a new local search heuristic for minimum sum of squares clustering, *Pattern Recognition*, vol.34, no.2, p.405–413.

[68] HANSEN P. et MLADENOVIĆ N. (2001), Variable Neighborhood Search : principles and applications, *European J. Oper. Res.*, vol.130, no.3, p.449–467.

[69] HANSEN P., MLADENOVIĆ N. et PEREZ-BRITOS D. (2001), Variable neighborhood decomposition search, *Journal of Heuristics*, vol.7, no.4, p.335–350.

[70] HANSEN P., MLADENOVIĆ N. et UROSEVIC D. (2004), Variable Neighborhood Search for the maximum clique, *Discrete Applied Mathematics*, vol.145, no.1, p.117–125.

[71] HAYES R.E., BERTRAND F.H., AUDET C. et KOLACZKOWSKI S.T. (2003), Catalytic combustion kinetics : Using a direct search algorithm to evaluate kinetic parameters from light-off curves, *The Canadian Journal of Chemical Engineering*, vol.81, no.6, p.1192–1199.

[72] HEDAR A. et FUKUSHIMA M. (2006), Derivative-free filter simulated annealing method for constrained continuous global optimization, *Journal of Global Optimization*, vol.35, no.4, p.521–549.

[73] HOOKE R. et JEEVES T.A. (1961), Direct search solution of numerical and statistical problems, *Journal of the Association for Computing Machinery*, vol.8, no.2, p.212–229.

[74] HOUGH P.D., KOLDA T.G. et TORCZON V. (2001), Asynchronous Parallel Pattern Search for nonlinear optimization, SIAM *Journal on Scientific Computing*, vol.23, no.1, p.134–156.

[75] HOUSEHOLDER A.S. (1958), Unitary Triangularization of a Nonsymmetric Matrix, *Journal of the Association for Computing Machinery (*ACM*)*, vol.5, no.4, p.339–342.

[76] IHME M., MARSDEN A.L. et PITSCH H. (2008), Generation of optimal artificial neural networks using a pattern search algorithm : application to approximation of chemical systems. À paraître dans *Neural Computation*.

[77] JONES D.R., PERTTUNEN C.D. et STUCKMAN B.E. (1993), Lipschitzian optimization without the Lipschitz constant, *Journal of Optimization Theory and Application*, vol.79, no.1, p.157–181.

[78] KELLEY C.T. (1999), *«Iterative Methods for Optimization»*. No.18 de Frontiers in Applied Mathematics. SIAM, Philadelphia.

[79] KOKKOLARAS M., AUDET C. et DENNIS J.E. Jr. (2001), Mixed variable optimization of the number and composition of heat intercepts in a thermal insulation system, *Optimization and Engineering*, vol.2, no.1, p.5–29.

[80] KOLDA T.G. (2005), Revisiting Asynchronous Parallel Pattern Search for Nonlinear Optimization, SIAM *Journal on Optimization*, vol.16, no.2, p.563–586.

[81] KOLDA T.G., LEWIS R.M. et TORCZON V. (2003), Optimization by direct search : new perspectives on some classical and modern methods, SIAM *Rev.*, vol.45, no.3, p.385–482.

[82] KOLDA T.G., LEWIS R.M. et TORCZON V. (2006), Stationarity results for generating set search for linearly constrained optimization, SIAM *Journal on Optimization*, vol.17, no.4, p.943–968.

[83] KOLDA T.G. et TORCZON V. (2003), Understanding Asynchronous Parallel Pattern Search, dans *«High Performance Algorithms and Software for Nonlinear Optimization»*, G.DiPillo et A.Murli, Eds. Kluwer Academic Publishers, B.V., p.316–335.

[84] KOLDA T.G. et TORCZON V. (2004), On the convergence of asynchronous parallel pattern search, SIAM *Journal on Optimization*, vol.14, no.4, p.939–964.

[85] LEWIS R.M. et TORCZON V. (1996), Rank ordering and positive bases in pattern search algorithms. Rapport technique 96–71, Institute for Computer Applications in Science and Engineering, Mail Stop 132C, NASA Langley Research Center, Hampton, Virginia 23681–2199.

[86] LEWIS R.M. et TORCZON V. (1999), Pattern search algorithms for bound constrained minimization, SIAM *Journal on Optimization*, vol.9, no.4, p.1082–1099.

[87] LEWIS R.M. et TORCZON V. (2000), Pattern search methods for linearly constrained minimization, SIAM *Journal on Optimization*, vol.10, no.3, p.917–941.

[88] LEWIS R.M. et TORCZON V. (2002), A globally convergent augmented Lagrangian pattern search algorithm for optimization with general constraints and simple bounds, SIAM *Journal on Optimization*, vol.12, no.4, p.1075–1089.

[89] LEWIS R.M., TORCZON V. et TROSSET M.W. (2000), Direct search methods : Then and now, *Journal of Computational and Applied Mathematics*, vol.124, no.1–2, p.191–207.

[90] LIU C.-S. et TSENG C.-H. (2000), Parallel synchronous and asynchronous space-decomposition algorithms for large-scale minimization problems, *Computational Optimization and Applications*, vol.17, no.1, p.85–107.

[91] LOPHAVEN S., NIELSEN H. et SØONDERGAARD J. (2002), DACE - A MATLAB Kriging Toolbox, Version 2.0. Rapport technique IMM-REP-2002-12, Informatics and Mathematical Modelling, Technical University of Denmark.

[92] LUCIDI S. et SCIANDRONE M. (2002), A derivative-free algorithm for bound constrained optimization, *Computational Optimization and Applications*, vol.21, no.2, p.119–142.

[93] LUKŠAN L. et VLČEK J. (2000), Test problems for nonsmooth unconstrained and linearly constrained optimization. Rapport technique V-798, ICS AS CR.

[94] MANGASARIAN O.L. (1995), Parallel gradient distribution in unconstrained optimization, SIAM *Journal on Control and Optimization*, vol.33, no.6, p.1916–1925.

[95] MARSDEN A.L., VASILYEV O.V. et MOIN P. (2002), Construction of commutative filters for LES on unstructured meshes, *Journal of Computational Physics*, vol.2, no.175, p.583–603.

[96] MARSDEN A.L., WANG M., DENNIS J.E. Jr. et MOIN P. (2004), Optimal aeroacoustic shape design using the surrogate management framework, *Optimization and Engineering*, vol.5, no.2, p.235–262.

[97] MARSDEN A.L., WANG M., DENNIS J.E. Jr. et MOIN P. (2004), Suppression of vortex-shedding noise via derivative-free shape optimization, *Physics of Fluids*, vol.10, no.16, p.L83–L86.

[98] MARSDEN A.L., WANG M., DENNIS J.E. Jr. et MOIN P. (2007), Trailing-edge noise reduction using derivative-free optimization and large-eddy simulation, *Journal of Fluid Mechanics*, no.572, p.13–36.

[99] MATOUŠEK J. (1998), On the L_2-Discrepancy for Anchored Boxes, *Journal of Complexity*, vol.14, no.4, p.527–556.

[100] MLADENOVIĆ N. et HANSEN P. (1997), Variable Neighborhood Search, *Computers & Operations Research*, vol.24, no.11, p.1097–1100.

[101] NELDER J.A. et MEAD R. (1965), A simplex method for function minimization, *The Computer Journal*, vol.7, no.4, p.308–313.

[102] PEREZ R., LIU H.H.T. et BEHDINAN K. (2004), Evaluation of multidisciplinary optimization approaches for aircraft conceptual design, dans «AIAA/ISSMO *Multidisciplinary Analysis and Optimization*», Albany, NY.

[103] PETERS M.S., TIMMERHAUS K.D. et WEST R.E. (2003), *«Plant Design and Economics for Chemical Engineers 5th ed.»*. McGraw-Hill, New York.

[104] POWELL M. J.D. (1998), Direct search algorithms for optimization calculations, *Acta Numerica*, vol.7, p.287–336.

[105] POWELL M. J.D. (2003), On trust region methods for unconstrained minimization without derivatives, *Mathematical Programming*, vol.97, p.605–623.

[106] PRICE C.J. et COOPE I.D. (2003), Frames and grids in unconstrained and linearly constrained optimization : A nonsmooth approach, SIAM *Journal on Optimization*, vol.14, no.2, p.415–438.

[107] PRICE C.J., COOPE I.D. et BYATT D. (2002), A convergent variant of the Nelder Mead algorithm, *Journal of Optimization Theory and Applications*, vol.113, no.1, p.5–19.

[108] ROSENBROCK H.H. (1960), An Automatic Method for Finding the Greatest or Least Value of a Function, *The Computer Journal*, vol.3, no.3, p.175–184.

[109] SAGASTIZÁBAL C.A. et SOLODOV M.V. (2002), Parallel variable distribution for constrained optimization, *Computational Optimization and Applications*, vol.22, no.1, p.111–131.

[110] SERAFINI D.B. (1998), *A Framework for Managing Models in Nonlinear Optimization of Computationally Expensive Functions*. Thèse, Department of Mathematical Sciences, Rice University, Houston, Texas.

[111] SIEDER W.D., SEADER J.D. et LEWIN D.R. (1999), *«Process design principles : Synthesis, analysis and evaluation»*. John Wiley and Sons Inc., New York.

[112] SNYDER J.D. et SUBRAMANIAM B. (1994), A novel reverse flow strategy for ethylbenzene dehydrogenation in a packed-bed reactor, *Chemical Engineering Science*, vol.49, p.5585–5601.

[113] SOBIESZCZANSKI-SOBIESKI J., AGTE J.S. et SANDUSKY R.R. Jr. (1998), Bi-level integrated system synthesis (BLISS). Rapport technique NASA/TM-1998-208715, NASA, Langley Research Center.

[114] SOBIESZCZANSKI-SOBIESKI J. et HAFTKA R.T. (1997), Multidisciplinary Aerospace Design Optimization : Survey of Recent Developments, *Structural Optimization*, vol.14, no.1, p.1–23.

[115] SOLODOV M.V. (1997), New inexact parallel variable distribution algorithms, *Computational Optimization and Applications*, vol.7, no.2, p.165–182.

[116] SOLODOV M.V. (1998), On the convergence of constrained parallel variable distribution algorithms, SIAM *Journal on Optimization*, vol.8, no.1, p.187–196.

[117] STEIN M. (1987), Large sample properties of simulations using latin hypercube sampling, *Technometrics*, vol.29, no.2, p.143–151.

[118] TANG B. (1993), Orthogonal array-based latin hypercubes, *Journal of the American Statistical Association*, vol.88, no.424, p.1392–1397.

[119] THE MATHWORKS, INC. (2005), MATLAB Genetic Algorithm and Direct Search (GADS) toolbox, www.mathworks.com/products/gads.

[120] TORCZON V. (1997), On the Convergence of Pattern Search Algorithms, SIAM *Journal on Optimization*, vol.7, no.1, p.1–25.

[121] TREFETHEN N. (2002), The hundred dollar hundred digit challenge, SIAM *News*, vol.35, no.1. www.siam.org/news/news.php ?id=388.

[122] TSENG P. (1993), Dual coordinate ascent methods for non-strictly convex minimization, *Mathematical Programming*, vol.59, no.2, p.231–247.

[123] VAZ A.I.F. et VICENTE L.N. (2007), A particle swarm pattern search method for bound constrained global optimization, *Journal of Global Optimization*, vol.39, no.2, p.197–219.

[124] YAMAKAWA E. et FUKUSHIMA M. (1999), Testing parallel variable transformation, *Computational Optimization and Applications*, vol.13, no.1-3, p.253–274.