Méthode multi-agent d'optimisation par partitionnement auto-organisé

G. Picard[†] picard@emse.fr
D. Villanueva[†‡] villanueva@emse.fr
R. Le Riche[†⋆] leriche@emse.fr
R. T. Haftka[‡] haftka@ufl.edu

[†]Institut Henri Fayol, ENSM.SE, Saint-Étienne, France
[‡]University of Florida, Gainesville, FL, 32611, USA
[⋆]CNRS UMR 5146, Saint-Étienne, France

Résumé

Dans le cadre de la conception de produits complexes, et en nous basant sur une méthode d'optimisation multi-agent par partitionnement de l'espace de conception, nous proposons une nouvelle méthode permettant au système multi-agent de découvrir tous les optima locaux par des mécanismes auto-organisés de création et de destruction d'agents, et ce avec un nombre limité d'appels aux fonctions coûteuses. Les agents sont en charge de la découverte d'optima locaux dans leur propre partition de l'espace de conception, qui évolue au cours de la résolution. Notre approche est mise en œuvre sur un problème illustratif en 2 dimensions et un problème difficile en 6 dimensions.

Mots-clés : *Auto-organisation, optimisation par métamodélisation, partionnement adaptatif, agents coopératifs*

Abstract

Based on an multi-agent optimization method for complex system design with adaptive partitioning of the design space, we propose a new multi-agent method to discover all the local optima using self-organizing mechanisms of creation and destruction of agents, with a limited number of expensive function calls. Each agent is responsible for the discovery of local optima in its own partition of the design space, which evolves during the solving process. Our approach is implemented on a sample illustrative 2D-problem and a more difficult 6D-problem.

Keywords: *Self-organization, Surrogate-based Optimization, Space Partitioning, Multi-agent Systems*

1 Introduction

Dans le cadre de la conception de systèmes complexes, comme des avions ou des véhicules terrestres, l'usage de simulateurs coûteux est souvent nécessaire. Ces simulateurs permettent de calculer les propriétés des objets simulés (par exemple d'un avion) en fonction de données d'entrée (par exemple, le nombre de passagers). En conception, le but est de proposer des configurations optimales, dans le sens où elles optimisent certaines fonctions de coût (par exemple, la masse de l'avion). De plus, certaines contraintes doivent être respectées pour que la configuration soit valide (par exemple, que l'avion soit capable de décoller d'une certaine longueur de piste). Ce cadre de conception consiste donc en la résolution de problèmes d'optimisation sous contraintes. Traditionnellement, en optimisation, il faut faire appel à la fonction à optimiser afin de calculer l'optimum. Or, dans le cadre présenté, cette fonction est éminemment coûteuse (plusieurs heures de calcul) puisque résultant de l'appel à un simulateur. Nous cherchons donc à proposer des méthodes qui permettent de trouver des configurations optimales et valides sans faire appel à la fonction à optimiser de trop nombreuses fois.

Dans des études antérieures [20], qui ont eut lieu dans le cadre du projet ANR ID4CS [1], nous avons décrit une méthodologie basée sur les systèmes multi-agents (SMA) pour la conception de systèmes complexes. Dans cette première approche, chaque agent construit un métamodèle différent (i.e. une approximation de la fonction réelle) sur lequel il réalise une optimisation globale sur tout l'espace de conception (c-à-d. sur tout l'espace défini par les variables dont on cherche les valeurs). Grâce à la coopération (échange de points de recherche choisis), le SMA définit non seulement une métamodélisation multiple mais aussi une méthode d'optimisation globale de haut niveau dont l'efficacité et la robustesse ont été évaluées. Cependant, les métamodèles étaient appliqués à tout l'espace de conception (comme dans la plupart des approches classiques). Par la suite,

1. http://www.irit.fr/ID4CS

nous avions donc proposé une méthode basée sur le partitionnement de l'espace de recherche [22]. L'idée consistait à attribuer un métamodèle à chaque sous-région de sorte que *(i)* chaque optimisation soit moins coûteuse en calculs car portant sur une partie seulement de l'espace de recherche, *(ii)* le processus d'optimisation se stabilise sur des optima locaux et globaux et *(iii)* le partitionnement final fournisse une meilleure compréhension du problème d'optimisation (par identification des optima et des meilleurs métamodèles locaux). L'avantage était de distribuer l'exploration et l'exploitation des solutions candidates, et ainsi de détecter différents optima locaux dans chacune des partitions. Par contre, notre méthode reposait sur un partitionnement a priori (par exemple en 4 sous-régions égales en volume) de l'espace de conception, ce qui en faisait un paramètre délicat de notre méthode.

Dans cet article, nous proposons une amélioration majeure à notre méthode multi-agent en permettant au SMA de découvrir tous les optima locaux par des mécanismes auto-organisés de création et de destruction d'agents. Les agents sont en charge de la découverte d'optima locaux dans leur propre sous-région de l'espace de conception, qui évolue au cours de la résolution. La nouveauté de cette approche réside dans l'utilisation conjointe de (*i*) techniques d'optimisation par métamodélisation et (*ii*) d'auto-organisation pour le partitionnement de l'espace de recherche afin de trouver tous les optima locaux. La coordination entre les agents, à travers l'échange de points et l'évolution auto-organisée des frontières des sous-régions permet aux agents de se stabiliser autour des optima locaux. Comme dans les techniques d'optimisation globale par essaim de particules [2, 11, 9], par algorithmes génétique [8], ou par clustering (chap. 5 de [17]), notre objectif est de localiser tous les optima locaux, mais contrairement à ces algorithmes, notre approche appelle la fonction objectif et/ou les contraintes coûteuses avec parcimonie. Pour des problèmes du monde réel, la capacité de localiser de nombreux optima dans un budget limité de calcul est souhaitable lorsque l'optimum global peut être trop coûteux à trouver, et parce qu'elle fournit à l'utilisateur un ensemble diversifié de solutions acceptables. Notre approche multi-agent en outre (*i*) utilise la création d'agents pour explorer l'espace de recherche et, (*ii*) la suppression d'agents pour accroître l'efficacité.

La section 2 présente le contexte théorique de notre travail, ainsi que les méthodes antérieures à notre travail. Dans la section 3, nous fournissons un aperçu de notre méthode d'optimisation par agents-métamodèles. Ensuite, nous décrivons dans la section 4 la méthode de partitionnement auto-organisé de l'espace de conception permettant de localiser les optima tout en maximisant la précision des métamodèles. Enfin, dans les sections 5 et 6, notre méthode est illustrée sur un problème d'optimisation à deux dimensions caractérisé par des régions admissibles déconnectées et un problème plus difficile en six dimensions, sans contraintes mais avec quatre optima locaux. Nous concluons et dressons quelques perpectives en section 7.

2 Optimisation multi-agent par partitionnement

Dans le cadre de l'optimisation par appel à simulateurs coûteux, la technique usuelle est de faire de l'optimisation par métamodélisation [6, 12, 14, 16]. Un métamodèle est une fonction mathématique *(i)* approximant les réponses du modèle étudié (par exemple, le modèle représentant la masse d'un avion en fonction de ses différentes dimensions), construite le plus souvent à partir d'un ensemble de taille restreinte d'entrées-sorties de ce modèle (plan d'expérience ou DOE) *(ii)* de coût de calcul négligeable et *(iii)* ayant pour objectif de prédire de nouvelles réponses dans une partie de l'espace des entrées (éventuellement tout cet espace) [7]. Un métamodèle est par exemple un simple polynôme, un réseau de neurones ou une fonction de krigeage.

De nombreux travaux ont proposé des stratégies d'utilisation de multiples métamodèles (*multi-surrogate*) pour l'optimisation [23, 15, 19, 4]. Dans cette vision multi-métamodèle, une méthode multi-agent avait été proposée [22]. Elle repose sur un partitionnement de l'espace de conception. Chaque agent est responsable de l'exploration d'une sous-région. Cette exploration consiste en une optimisation globale dans cette sous-région en utilisant le métamodèle le plus précis (en fonction des erreurs de métamodèle calculées tout au long du processus d'optimisation). Un agent possède une bibliothèque de métamodèles qu'il va pouvoir utiliser (e.g. surface de réponse polynomiale ou krigeage). Ainsi un agent va optimiser la fonction métamodèle notée $\hat{f}$ sous les contraintes $\hat{g}$ dans son sous-espace, au lieu d'optimiser la fonction réelle f sous les contraintes réelles g dans tout

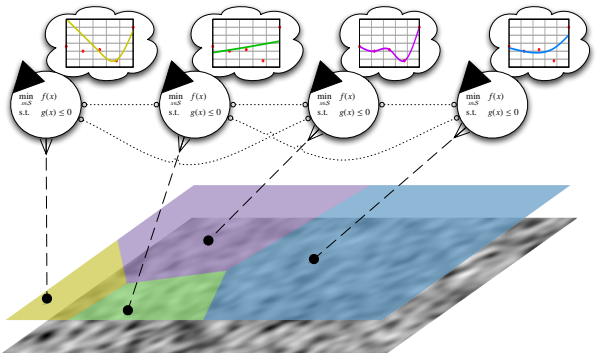


FIGURE 1 – Les agents-métamodèles optimisent une fonction commune sur une sous-région de l'espace de recherche (ici en 2D) suivant un métamodèle personnel, et communiquent des points à leurs voisins directs.

l'espace. La tâche d'optimisation d'un agent i au temps t de la fonction $\hat{f}$ sous les contraintes $\hat{g}$ dans sa sous-région $\mathcal{P}_i$ en fonction de sa base de croyances $\mathbb{X}^t$ est notée $\texttt{optim}(\hat{f}, \hat{g}, \mathbb{X}^t, \mathcal{P}_i)$, ce qui correspond au problème d'optimisation suivant :

$$\begin{array}{ll} \underset{x \in \mathcal{P}_i}{\text{minimiser}} & \hat{f}(x) \\ \text{avec} & \hat{g}(x) \leq 0 \end{array} \tag{1}$$

Les sous-régions au temps t sont simplement des cellules de Voronoi [1] dont les centres sont le meilleur point de chaque agent au temps t (l'optimum de la sous-région i, noté c_i). Une cellule contient plusieurs points déjà calculés et constitue la base de croyance de l'agent responsable de la sous-région. La base de croyance de l'agent i au temps t est notée $(\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)_i$. Deux caractéristiques vont changer pour chaque agent : le centre (donc les frontières des sous-régions) et le métamodèle utilisé pour approcher la fonction coûteuse. Les cellules et les métamodèles s'adaptent donc à l'espace de conception. Nous parlons alors d'optimisation par partionnement adaptatif.

La figure 1 illustre cette approche. Comme les frontières des cellules changent avec le temps, les agents se communiquent les points calculés afin de mettre à jour leurs métamodèles respectifs, mais uniquement sur leurs sous-régions respectives.

3 Agents-métamodèles

Le comportement suivi par chaque agent peut être résumé par l'algorithme 1. Notre méthode d'optimisation est une méthode pas-à-pas, dans laquelle les agents agissent en parallèle puis attendent avant de passer au pas suivant, ce qui permet, entre autres, de paralléliser les appels aux fonctions coûteuses. La terminaison est paramétrée par un nombre de pas maximum.

En supposant que les sous-régions sont déjà définies, chaque agent choisit le métamodèle le plus précis pour sa sous-région et sa base de croyance (lignes 5-9). Il est possible que l'agent demande des points à ses voisins s'il n'en a pas assez pour ajuster les métamodèles (lignes 5-7). La précision d'un métamodèle est déterminée par une mesure d'erreur par validation croisée : la somme des carrés des erreurs de prédiction [22]. Une fois le métamodèle le plus précis choisi (ligne 9), l'agent cherche l'optimum dans sa sous-région en utilisant ce métamodèle (ligne 10). En cas d'obtention d'un point proche d'un point déjà connu ou d'un point non admissible, l'agent va explorer la sous-région. Ceci se traduit par le fait que l'agent ajoute un point dans la base de croyance qui maximise la distance minimum aux points déjà présents dans la base, et dans ce cas ce point remplace $\hat{x}^*$ (lignes 11-12). L'appel au simulateur coûteux est alors effectué pour $\hat{x}^*$ (ligne 13) et un nouveau point $(\hat{x}^*, f(\hat{x}^*), g(\hat{x}^*))$ est ajouté à la base de croyances (ligne 14). Ensuite, l'agent va repositionner son centre (ligne 15) et éventuellement ré-organiser l'espace en fusionnant, en se séparant ou en créant un nouvel agent (ligne 16), avant de passer au pas suivant (ligne 17).

Cet algorithme est une reformulation de l'algorithme présenté dans [22] avec l'ajout de l'étape d'auto-organisation (ligne 16) qui est détaillée dans la section suivante et qui constitue le cœur de notre contribution.

4 Partionnement auto-organisé

Cette section a pour but de présenter les mécanismes qui vont permettre de créer et d'adapter le partitionnement en cours de résolution. En effet, le partitionnement dépend fortement de la topologie de l'espace de recherche. Nous proposons un mécanisme auto-organisé pour partitionner l'espace en sous-régions qui s'adaptent à la topologie de l'espace. Par auto-organisé nous entendons que les agents (et ainsi les sous-régions) sont créés et supprimés au travers d'un processus coopératif d'optimisation par les agents eux-mêmes. Les agents se divisent lorsque les points de leurs sous-régions forment des clusters distincts (*création*), et fusionnent lorsque des centres d'agents (des op-

Algorithme 1: Comportement d'un agent i

1 t = 1 (état initial)
2 **tant que** $t \leq t^{max}$ **faire**
3 Mettre à jour sa partition : $\mathcal{P}_i \leftarrow \{x \in \mathcal{S} \text{ s.t.} \|x - c_i\|^2 \leq \|x - c_j\|^2 \ , \ j \neq i\}$
4 Mettre à jour sa base de croyances
5 **si** *Pas suffisamment de points dans la base de croyance pour construire les métamodèles* **alors**
6 Demander des points aux agents voisins
7 Ajouter les points à la base de croyances
8 Construire les métamodèles $\hat{f}$ et $\hat{g}$ à partir de $(\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)_i$
9 Choisir le métamodèle le plus précis
10 Optimiser dans sa sous-région : $\hat{x}^* \leftarrow \texttt{optim}(\hat{f}, \hat{g}, \mathbb{X}^t, \mathcal{P}_i)$
11 **si** $\hat{x}^*$ *est proche d'un point existant ou* $\hat{x}^*$ *non admissible* **alors**
12 Explorer
13 Calculer $f(\hat{x}^*)$ et $g(\hat{x}^*)$
14 $(\mathbb{X}^{t+1}, \mathbb{F}^{t+1}, \mathbb{G}^{t+1})_i \leftarrow (\mathbb{X}^t, \mathbb{F}^t, \mathbb{G}^t)_i \cup (\hat{x}^*, f(\hat{x}^*), g(\hat{x}^*))$
15 Mettre à jour le centre c_i (voir Section 4.1)
16 Vérifier si fusion, séparation ou création (voir Section 4.2)
17 $t = t + 1$

tima locaux) convergent (*suppression*).

4.1 Déplacement des centres

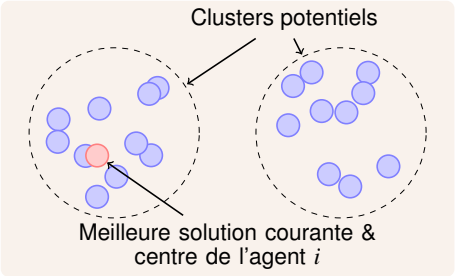
Les conditions de déplacement des centres à chaque pas sont les mêmes que dans [22]. Pour résumer, le centre d'une sous-région est le meilleur point de cette sous-région. Le centre est déplacé à l'optimum de l'itération précédente s'il améliore la valeur de la fonction objectif, et si à la fois x^{*t-1} et c^{t-1} satisfont la contrainte g. Le centre est déplacé lorsque le dernier optimum satisfait la contrainte mais pas le centre précédent. Le centre est également déplacé s'il s'approche au mieux des contraintes alors que celles-ci ne sont respectées ni par le nouveau point, ni par le centre actuel. Ceci correspond à la ligne 15 de l'algorithme 1. [22] introduit la relation $\succ$ pour comparer deux centres. $c_1 \succ c_2$ signifie que c_1 est meilleur (au sens précédemment décrit) que c_2.

4.2 Auto-organisation des sous-régions

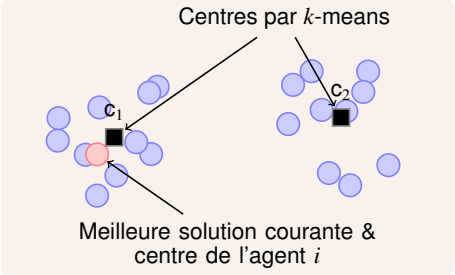
Une fois qu'un agent a ajouté un nouveau point dans sa base de croyances (ligne 14) et a déplacé son centre sur le meilleur point (ligne 15), il va vérifier s'il doit créer ou supprimer un agent ou fusionner avec un autre agent (ligne 16). Fusionner des agents (donc des sous-régions) évite que des agents ne se regroupent inutilement dans une même zone. Séparer un agent (donc créer un agent) est un moyen d'explorer plus efficacement l'espace par un affinement du partitionnement. La séparation et la fusion surviennent à la fin de chaque itération (ligne 16) : les agents sont d'abord fusionnés (si nécessaire), les points appartenant aux agents fusionnés sont partagés entre les agents restants en fonction du partitionnement, et chaque agent restant vérifie s'il se sépare ou non. Cette technique d'auto-organisation des agents et des sous-régions nous permet ainsi de ne pas fixer arbitrairemen un nombre initial de sous-régions, et de lancer la procédure d'optimisation avec un unique agent qui se séparera si nécessaire.

Fusion d'agents convergents. Des agents sont fusionnés (et donc un agent est supprimé) si les centres de leurs sous-régions respectives deviennent trop proches (en distance Euclidienne). Nous mesurons la distance minimum entre centres de sous-régions comme un pourcentage de la distance Euclidienne maximale possible entre points de l'espace de recherche. Lorsque l'on examine les agents, l'agent avec la plus faible performance (le plus mauvais optimum) est supprimé. Par exemple, pour les agent 1 et 2, si $c_1 \succ c_2$, l'agent 2 est supprimé. Avant la suppression, l'agent supprimé distribue ses points aux agents les plus proches de ces points.

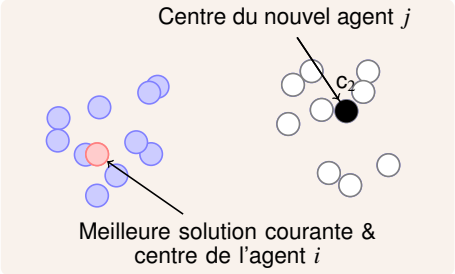
Séparation de sous-régions en cluster. Il est intéressant de créer un agent lorsqu'il se trouve que des points sont regroupés dans deux zones distinctes d'une même sous-région, comme illustré dans la Figure 2(a). Une telle situation



(a) Clusters potentiels dans une sous-région



(b) Clusters obtenus par k-means



(c) Clusters finals après déplacement des centres vers la meilleure solution courante et le plus proche point

FIGURE 2 – Illustration du processus de création d'un agent j étant donnés des points de la sous-région de l'agent i.

peut apparaître lorsqu'il y a plusieurs optima locaux dans une même sous-région.

Les agents sont créés en utilisant une méthode de k-means [5] pour deux clusters ($k = 2$) étant donnés les points dans une sous-région, où la position initiale des centres est la meilleure solution courante de la sous-région (le centre actuel) et la moyenne de l'ensemble des points. Comme le clustering par k-means fournit des centres qui ne sont pas forcément des points existants comme illustré en Figure 2(b), nous déplaçons le nouveau centre vers le point disponible le plus proche dans la sous-région pour éviter un appel supplémentaire à la fonction coûteuse. Ceci est effectué en mesurant tout d'abord la distance des centres fournis par k-means au meilleur point actuel, puis en déplaçant le centre le plus proche vers le meilleur point, afin de conserver ce bon point. Pour l'autre centre, nous mesurons la distance de ce centre vers les autres points, et déplaçons le centre vers le plus proche. Le clustering final est illustré dans la Figure 2(c). Le résultat est donc un nouvel agent avec un centre positionné sur un point existant, où l'agent créateur reste sur son centre, à savoir le meilleur point.

Le clustering final est ensuite validé en utilisant la *valeur de silhouette moyenne* des points de la sous-région. La silhouette, introduite par [13], est utilisée pour valider le nombre de clusters en fournissant une mesure de l'étanchéité à l'intérieur des clusters et la séparation avec les autres clusters pour chaque point de données i pour un ensemble de points. La valeur silhouette de chaque point est ainsi définie par :

$$s(i) = \frac{b(i) - a(i)}{max\{a(i), b(i)\}} \qquad (2)$$

où a_i est la distance moyenne entre le point i et tous les autres points étant dans le même cluster que i, et b_i est le minimum des distances moyennes entre le point i et tous les autres points dans les autres clusters. La valeur silhouette s_i varie de -1 à 1. Pour s_i proche de 0, le point peut être assigné à n'importe quel cluster. Si s_i est proche de -1, le point a été mal classé, et, si toutes les valeurs s_i sont proches de 1, l'ensemble de points est bien classé. La moyenne de silhouette d'un jeu de points est souvent utilisée pour caractériser un clustering. Dans cet article, nous acceptons (validons) un clustering si toutes les silhouettes s_i sont plus grandes que 0 et que la silhouette moyenne est plus grande qu'un certain seuil (donné en paramètre de notre méthode).

Création de nouveaux agents. Les agents peuvent atteindre un point pour lequel aucune amélioration n'est obtenue en plusieurs itérations. Par exemple, cela peut se produire lorsque chaque agent a localisé le meilleur point de sa sous-région. La zone autour du meilleur point est peuplée de points, et chaque agent est amené à explorer (en créant un point maximisant la distance minimum aux autres points, voir Section. 3) durant plusieurs itérations consécutives, et aucun autre optimum local potentiel n'est localisé. Ceci peut également se produire lors d'itérations en début de résolution lorsque les métamodèles sont mal ajustés dans la sous-région. Afin d'améliorer l'exploration, un nouvel agent est alors créé dans l'espace de recherche lorsque il n'y a pas d'amélioration depuis n itérations de résolution (i.e. les centres

des sous-régions ne se sont pas déplacés depuis n itérations). Nous appelons ce paramètre n le *seuil de stagnation*. Pour créer un nouvel agent, un nouveau centre est créé sur un point existant qui maximise la distance minimum aux centres existants. L'espace est alors repartitionné.

D'autres approches proposent de recourir au partitionnement de l'espace de conception lorsque l'évaluation des points de l'espace est coûteuse : GROPE [3]. Une différence simple et pratique entre notre approche et GROPE est que nous recherchons les optima locaux alors que GROPE recherche l'optimum global. Une autre différence connexe est que le nombre de partitions dans notre algorithme dépendra du nombre d'optima locaux alors le nombre de partitions dans GROPE dépend du nombre total de points où les vrais critères d'optimisation ont été calculées (parce que les partitions sont définies par une triangulation sur ces points).

5 Exemple illustratif

Afin d'illustrer notre approche, reprenons l'exemple de [22], qui est un problème 2D avec 3 optima locaux, dont un global (voir Figure 3) :

$$\begin{aligned} \underset{x\in\Re^2}{\text{minimiser}} \quad & f(x) = -(x_1-10)^2 - (x_2-15)^2 \\ \text{avec} \quad & g(x) = \left(x_2 - \frac{5.1}{4\pi^2}x_1^2 + \frac{5}{\pi}x_1 - 6\right) + \cdots \\ & 10\left(1-\frac{1}{8\pi}\right)cos(x_1) + 10 - 2 \le 0 \\ & -5 \le x_1 \le 10 \\ & 0 \le x_2 \le 15 \end{aligned} \tag{3}$$

Dans cet exemple, à la fois f et g sont considérées coûteuses et sont approchées par des métamodèles. Les six métamodèles possibles (parmi lesquels chaque agent choisit le meilleur pour sa sous-région) sont décrits dans la Table 1. Les différents paramètres de notre méthode sont également résumés dans la Table 2. Notons que compte tenu de notre contexte applicatif, nous allouons un budget d'appels à la fonction coûteuse au système (ici 132, soit par exemple 12 points initiaux plus 120 appels en cours de résolution).

Le nombre de succès et l'efficacité de notre approche sont également comparés à un solveur étalon, à savoir un agent unique optimisant par métamodèle sur tout l'espace de recherche. Ceci correspond à l'approche communément utilisée dans l'état de l'art. Cet agent a

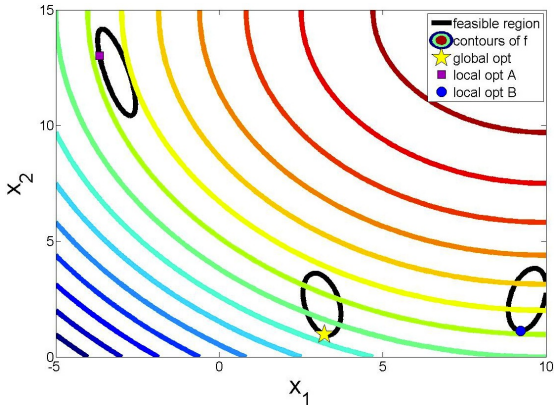


FIGURE 3 – Courbes de niveau de f pour le problème (3). Les frontières des régions admissibles sont représentées par des lignes noires pleines et l'optimum global est représenté par une étoile en $(3.214, 0.9633)$. Les optima locaux en $(-3.6685, 13.0299)$ et $(9.2153, 1.1240)$ sont respectivement représentés par un carré (A) et un cercle (B).

TABLE 1 – Métamodèles utilisables par les agents

ID	Description	# min. de points
1	Surface de réponse linéaire	
2	Surface de réponse quadratique	1.5 * # de cœfficients
3	Surface de réponse cubique	
4	Krigeage (tendance quadratique)	
5	Krigeage (tendance linaire)	# de param. +1
6	Krigeage (tendance constante)	

le même budget en appel que le système multi-agent (132 appels). Dans chaque cas (multi- ou mono-agent), afin d'évaluer la capacité des méthodes à explorer l'espace de recherche, nous avons conduit plusieurs expérimentations pour des jeux de points initiaux (*design of experiments* ou DOE) de tailles différentes (de 12 à 80) qui comptent cependant dans le budget d'appels. Les points initiaux sont placés par échantillonnage par hypercube latin (placement assez régulier dans l'espace). L'optimiseur utilisé localement par chaque agent est un algorithme de programmation quadratique récursive (le SQP de la fonction `fmincon` de Matlab [10]). Chaque expérimentation a été conduite 50 fois, et ce sont les valeurs médianes des résultats de ces expérimentations qui sont présentées.

Les Figures 4 et Fig. 5 montrent que le système multi-agent est bien plus efficace à trouver tous les optima (bien que l'agent seul soit capable de trouver l'optimum global avec succès) et ceci pour toutes les tailles de DOE (de 12 à 80).

La particularité de notre système est de créer et supprimer des agents en cours de résolution. La Figure 6 montre la dynamique des agents

TABLE 2 – Paramètres du système multi-agent

Paramètre	Valeur
Budget max. d'évaluations	132
Nb. max. d'agents	6
Nb. initial d'agents	1
Dist. min entre centres	10% distance max
Dist. min entre points	1e-3
Seuil de silhouette	0.4
Seuil de stagnation	3

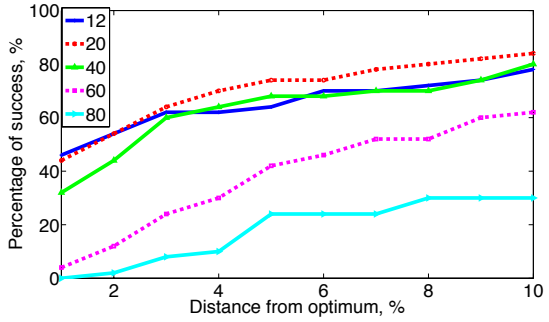


FIGURE 4 – Localisation de tous les optima pour le cas mono-agent

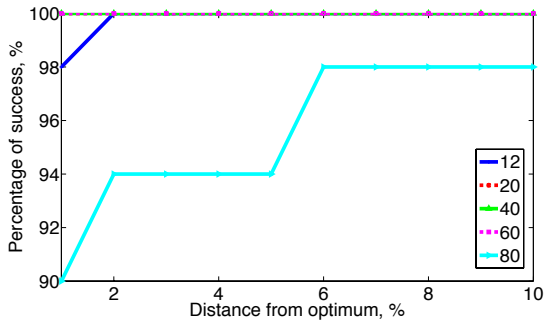


FIGURE 5 – Localisation de tous les optima pour le cas multi-agent

en fonction du temps. Nous pouvons noter que chaque courbe s'arrête à des itérations différentes puisque le budget d'appels est partagé par les différents agents. Nous pouvons observer que ce nombre se stabilise autour de 4 agents, ce qui explique la capacité pour le SMA à localiser tous les optima. C'est également un bon indicateur de la capacité de parallélisation de notre système et donc de diminution du temps de calcul. Dans cet exemple, nous pouvons attendre une diminution maximum de 75% (division par le nombre d'agents). Cette capacité de parallélisation est parfaitement réaliste dans le cadre de notre étude. En effet, le simulateur coûteux est souvent répliqué dans les infrastructures métiers.

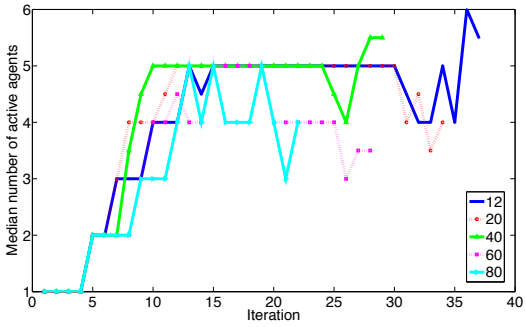


FIGURE 6 – Nombre médian d'agents dans le système multi-agent en fonction du temps

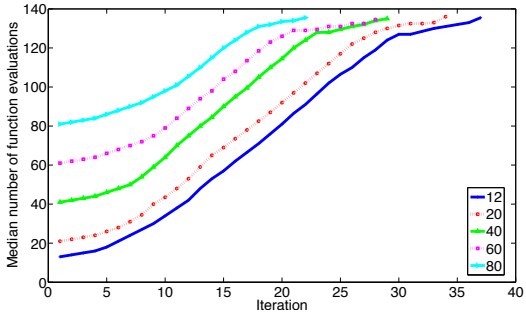


FIGURE 7 – Nombre médian d'appels au simulateur dans le cas multi-agent

De plus, le cas mono-agent semble avoir détecté tous les optima avec 60 appels et DOE de 12, alors que le SMA nécessite 70 appels, ce qui survient autour de l'itération 20 (voir Figure 7). Ceci équivaut à 18 appels en parallèle au simulateur. De cette manière, le SMA est théoriquement 3 fois plus efficace que le mono-agent. Pratiquement, les opérations qui sont ajoutées au multi-agent, par rapport au mono-agent, sont le partitionnement et la communication de points aux voisins (uniquement sur les tous premiers pas). Ces opérations sont de complexités négligeables par rapport aux opérations communes d'optimisation et d'appels à la fonction coûteuse.

Pour tous les résultats présentés, une tendance générale caractérise l'effet de la taille de DOE : le SMA est plus efficace avec de petits échantillons. Ceci souligne sa capacité à bien explorer l'espace de recherche.

En comparaison des travaux antérieurs [22], sans dynamique du nombre d'agents (avec 4 agents) et un DOE de taille 12, le SMA trouvait tous les optima à une distance de 2% dans seulement 66% des cas (contre 100% ici), avec une moyenne de 60 appels à la fonction coûteuse (contre 70 ici).

6 Problème d'optimisation 6D

Afin de démontrer le potentiel de passage à l'échelle de notre méthode, nous abordons ici un problème à 6 dimensions, de même ordre que les problèmes que nous rencontrons dans le projet ID4CS. La fonction à optimiser est la fonction Hartmann en 6 dimensions modifiée avec quatre optima, dont un global [18]. Pour des raisons de place, nous ne présentons pas la formulation de la fonction Hartmann modifiée, n'étant pas non plus représentable en courbes de niveaux. Pour plus de détails, nous re-dirigeons le lecteur vers [18].

Les valeurs des évaluations des optima sont présentées dans la Table 3. Pour obtenir une mesure approximative de la taille des bassins d'attraction qui contiennent les optima, nous avons mesuré le pourcentage d'exécutions d'optimisations locales qui converge vers chacun des optima. Pour ce faire, 20000 points ont été échantillonnés par hypercube latin et une optimisation locale a été effectuée en utilisant SQP. Ce pourcentage est également une mesure de la difficulté pour trouver ces optima.

TABLE 3 – Optima de la fonction Hartmann 6

Optimum	f	Pourcentage d'exécutions
Global	-3.33	50.4
Local 1	-3.21	21.1
Local 2	-3.00	8.7
Local 3	-2.90	19.8

Nous avons utilisé les mêmes conditions d'expérimentation que pour le cas 2D, exceptés les points suivants. Nous n'avons utilisé que du krigeage (métamodèles 4 et 6 de la Table 1) comme métamodèle. Seule la fonction objectif a été approchée par ces métamodèles (puisque les contraintes sont linéaires). Le budget est de 400 appels et les DOE sont de taille de 35, 56 et 100 points. La Table 4 résume ces paramètres.

TABLE 4 – Paramètres du système multi-agent pour Hartmann 6 modifiée

Paramètre	**Valeur**
Budget max. d'évaluations	400
Nb. max. d'agents	8
Nb. initial d'agents	1
Dist. min entre centres	10% distance max
Dist. min entre points	1e-3
Seuil de silhouette	0.25
Seuil de stagnation	3

Pour 50 répétitions, le nombre de succès à trouver tous les optima pour les cas mono- et multi-agent est présenté dans la Figure 8. Le SMA a bien plus de succès, même pour l'optimum local 2 qui est le plus difficile à trouver.

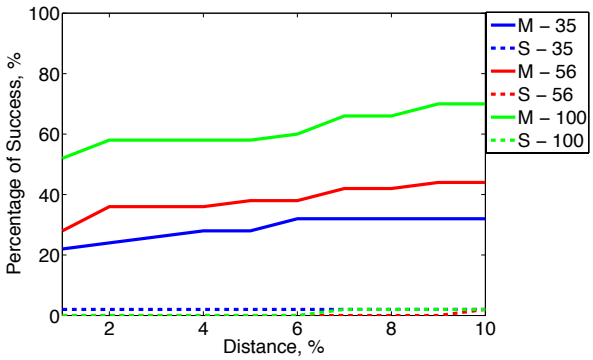


FIGURE 8 – Nombre de succès pour trouver tous les optima, dans le cas mono-agent (S) et multi-agent (M), et pour les différentes tailles de DOE (35,56 et 100).

La valeur médiane de la fonction objectif de la solution la plus proche de l'optimum est présentée dans la Figure 9. Pour l'optimum global et l'optimum local 1, l'efficacité est équivalente dans les cas mono- et multi-agent. On peut également observer que des petits DOE impliquent moins d'appels pour trouver ces optima.

Pour les optima locaux 2 et 3, le SMA a clairement l'avantage en trouvant des solutions proches de la vraie valeur optimale. Alors qu'il est clair pour l'optimum local 3 que des DOEs plus petits sont plus efficaces pour trouver l'optimum, il n'y a pas de relation claire entre taille de DOE et efficacité. Rappelons que l'optimum local 2 a été détecté comme étant le plus difficile à trouver (cf. Table 3). Ces résultats confirment que l'exploration est nécessaire pour localiser cet optimum local, et que le SMA, pour lequel l'exploration est une caractéristique inhérente, est plus capable de le localiser.

La Figure 10 présente le nombre médian d'agents. Alors que 8 agents au maximum peuvent être créés, on peut observer que le nombre médian se stabilise autour de 4. Ceci s'explique parce que lorsque les 4 optima sont trouvés, des nouveaux agents sont créés mais sont alors supprimés lorsqu'ils convergent vers les bassins d'attractions de ces optima.

7 Conclusions et perspectives

Cet article a introduit une technique multi-agent pour l'optimisation qui partitionne dynamique-

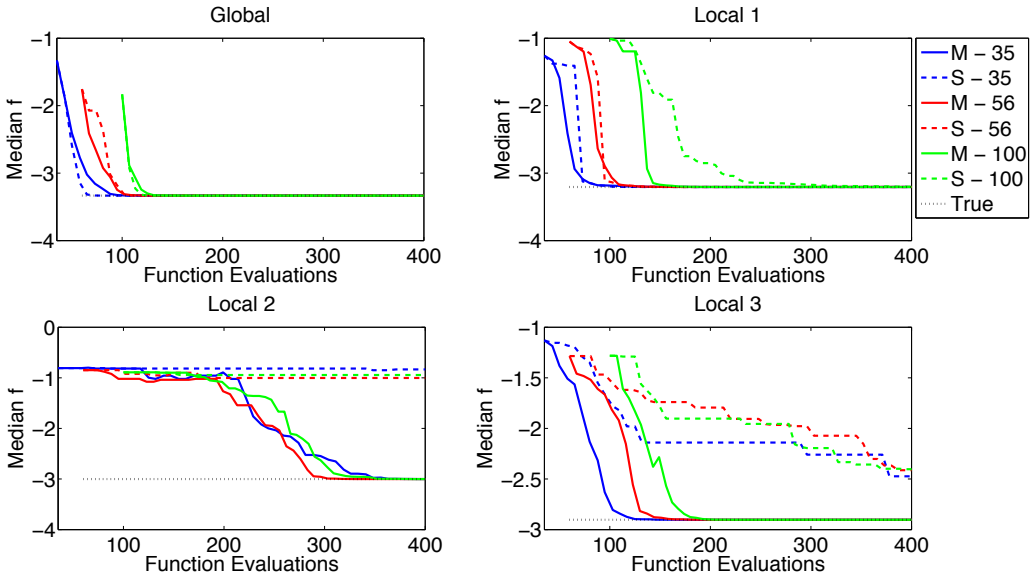


FIGURE 9 – Valeur médiane de la fonction objectif des solutions les plus proches de chaque optimum en fonction du nombre d'appels

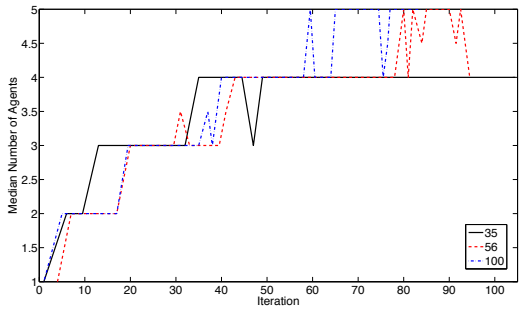


FIGURE 10 – Nombre médian d'agents dans le cas multi-agent

ment l'espace de recherche afin de trouver tous les optima locaux. Les centres des sous-régions sont déplacés pour se stabiliser autour d'optima locaux, et des agents sont créés ou supprimés en cours de fonctionnement, de manière auto-organisée, afin d'assurer exploration et efficacité, respectivement. Ces résultats font écho aux perspectives dressées et aux manques soulignés dans [22].

En appliquant notre méthode à deux exemples (2D et 6D), il a été montré empiriquement qu'un agent seul (e.g. un optimiseur par métamodélisation classique) a moins de succès pour trouver tous les optima que le système multi-agent, pour un nombre équivalent d'appels au simulateur coûteux, montrant ainsi que l'exploration au travers du partitionnement auto-organisé est une part importante de notre algorithme multi-agent. De plus, dans le cas 2D, il a été montré que notre approche multi-agent bénéficiait d'un budget de calcul réduit dédié à la recherche aléatoire : à nombre constant d'évaluations, les agents avec un nombre initial de points élevé sont moins efficaces pour trouver les solutions admissibles que les agents débutant leur optimisation avec un faible nombre de points initiaux. Dans l'exemple 6D, cette remarque est également vraie si nous observons l'évolution de la valeur médiane de la fonction de coût. Au-delà de ces problèmes académiques, notre méthode est actuellement appliquée à des cas réels comme la conception de boucliers thermiques [21].

Pour résumer, dans les deux cas que nous avons étudiés dans cet article, le SMA est bien plus efficace qu'un solveur standard pour *(i)* localiser l'optimum global et pour *(ii)* localiser tous les optima (à faible distance et faible valeurs g). Ceci démontre que notre comportement d'agent et de partitionnement auto-organisé entraîne que :

1. l'optimisation est moins coûteuse, car les agents s'exécutent en parallèle, pas-à-pas ;
2. le processus d'optimisation n'est pas seulement global mais peut également se stabiliser sur des optima locaux ;
3. le partitionnement final fournit une bonne compréhension du problème d'optimisation, comme chaque agent utilise le métamodèle le plus adapté à sa sous-région.

Ces propriétés restent encore à démontrer sur d'autres problèmes à plus grande dimension

ou avec de plus grands nombres d'optima, par exemple. Nous pensons que la méthode d'optimisation multi-agent proposée a un grand potentiel pour le calcul parallèle. En effet, avec l'augmentation du nombre de nœuds de calcul, le calcul des fonctions de coûts et contraintes coûteuses diminue de facteur d'ordre n en terme de temps de calcul réel (horloge). Mais le temps de résolution des problèmes devient alors limité par le temps pris par l'optimiseur. Dans l'algorithme que nous avons développé, la tâche d'optimisation en tant que telle peut être partagée parmi les n nœuds au travers d'agents. Nous envisageons d'étudier comment les agents peuvent fournir un paradigme utile pour l'optimisation dans des environnements de calcul parallèles, distribués et asynchrones.

Remerciements

Ces travaux ont été en partie financés par l'Agence Nationale de la Recherche (ANR) au travers du programme COSINUS (projet ID4CS ANR-09-COSI-005).

Références

[1] F. Aurenhammer. Voronoi diagrams : a survey of a fundamental geometric data structure. *ACM Computing Surveys (CSUR)*, 23(3) :345–405, 1991.

[2] R. Brits, A. P. Engelbrecht, and F. van den Bergh. Locating multiple optima using particle swarm optimization. *Applied Mathematics and Computation*, 189(2) :1859–1883, 2007.

[3] J. Elder. Global R^d optimization when probes are expensive : the GROPE algorithm. In *IEEE International Conference on Systems, Man and Cybernetics*, pages 577–582, 1992.

[4] B. Glaz, T. Goel, L. Liu, P. Friedmann, and R. T. Haftka. Multiple-surrogate approach to helicopter rotor blade vibration reduction. *AIAA Journal*, 47(1) :271–282, 2009.

[5] J. Hartigan and M. Wong. Algorithm as 136 : A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1) :100–108, 1979.

[6] R. Jin, X. Du, and W. Chen. The use of metamodeling techniques for optimization under uncertainty. *Structural and Multidisciplinary Optimization*, 25(2) :99–116, 2003.

[7] J. Kleijen. *Design and analysis of simulation experiments*. Springer, 2008.

[8] J. P. Li, M. E. Balazas, G. Parks, and P. J. Clarkson. A species conserving genetic algorithm for multimodal function optimization. *Evolutionary Computation*, 10(3) :207–234, 2002.

[9] X. Li. Adaptively choosing neighbourhood bests using species in a particle swarm optimizer for multimodal function optimization. In *Genetic and Evolutionary Computation (GECCO 2004)*, volume 3102 of *Lecture Notes in Computer Science*, pages 105–116, 2004.

[10] MATLAB. *version 7.9.0.529 (R2009b)*, chapter fmincon. The MathWorks Inc., Natick, Massachusetts, 2009.

[11] K. E. Parsopoulos and M. N. Vrahatis. *Artificial Neural Networks and Genetic Algorithms*, chapter Modification ofthe Particle Swarm Optimizer for Locating All the Global Minima, pages 324–327. Springer, 2001.

[12] N. V. Queipo, R. T. Haftka, W. Shyy, and T. Goel. Surrogate-based analysis and optimization. *Progress in Aerospace Sciences*, 41 :1–28, 2005.

[13] P. J. Rousseeuw. Silhouettes : a graphical aid to the interpretation and validation of cluster analysis. *Computational and Applied Mathematics*, 20 :53–65, 1987.

[14] J. Sacks, W. J. Welch, M. T. J., and H. P. Wynn. Design and analysis of computer experiments. *Statistical Science*, 4(4) :409–435, 1989.

[15] A. Samad, K. Kim, T. Goel, R. T. Haftka, and W. Shyy. Multiple surrogate modeling for axial compressor blade shape optimization. *Journal of Propulsion and Power*, 25(2) :302–310, 2008.

[16] T. W. Simpson, J. D. Peplinski, P. N. Koch, and J. K. Allen. Metamodels for computer based engineering design : survey and recommendations. *Engineering with Computers*, 17(2) :129–150, 2001.

[17] A. Torn and A. Zilinskas. Global optimization. In *Lecture Notes in Computer Science 350*. Springer Verlag, 1989.

[18] F. A. C. Viana. *Multiple Surrogates for Prediction and Optimization*. PhD thesis, University of Florida, 2011.

[19] F. A. C. Viana and R. T. Haftka. Using multiple surrogates for metamodeling. In *7th ASMO-UK/ISSMO International Conference on Engineering Design Optimization*, 2008.

[20] D. Villanueva, R. Le Riche, G. Picard, and R. Haftka. Surrogate-based agents for constrained optimization. In *14th AIAA Non-Deterministic Approaches Conference, Honolulu, HI*. AIAA, 2012.

[21] D. Villanueva, R. Le Riche, G. Picard, and R. Haftka. Dynamic design space partitioning for optimization of an integrated thermal protection system. In *54th AIAA/ASME/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. AIAA, 2013.

[22] D. Villanueva, G. Picard, R. Le Riche, and R. T. Haftka. Optimisation multi-agent par partitionnement adaptatif de l'espace de conception. In *20es Journées francophones des systèmes multi-agents (JFSMA'12)*, pages 149–158. Cépaduès, 2012.

[23] I. Voutchkov and A. J. Keane. Multiobjective optimization using surrogates. In *7th International Conference on Adaptive Computing in Design and Manufacture*, pages 167–175, Bristol, UK, 2006.