Institutional Repository - Research Portal
Dépôt Institutionnel - Portail de la Recherche
researchportal.unamur.be
University of Namur

THESIS / THÈSE

MASTER EN SCIENCES MATHÉMATIQUES À FINALITÉ SPÉCIALISÉE EN PERSPECTIVES PROFESSIONNELLES DES MATHÉMATIQUES APPLIQUÉES

Investigation sur les méta-modèles en présence de variables de nature mixte et en particulier de variables catégorielles pour la résolution de problèmes d'optimisation avec fonctions coûteuses

LAMBERT, Noémie

Award date:
2023

Awarding institution:
Universite de Namur

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal ?

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Download date: 29. Apr. 2024

UNIVERSITE DE NAMUR

Faculté des Sciences

INVESTIGATION SUR LES MÉTA-MODÈLES EN PRÉSENCE DE VARIABLES DE NATURE MIXTE ET EN PARTICULIER DE VARIABLES CATÉGORIELLES POUR LA RÉSOLUTION DE PROBLÈMES D'OPTIMISATION AVEC FONCTIONS COÛTEUSES

Mémoire présenté pour l'obtention

du grade académique de master en sciences mathématiques

à finalité spécialisée en Project Engineering

Noémie Lambert

Août 2023

UNIVERSITE DE NAMUR

Faculté des Sciences

INVESTIGATION SUR LES MÉTA-MODÈLES EN PRÉSENCE DE VARIABLES DE NATURE MIXTE ET EN PARTICULIER DE VARIABLES CATÉGORIELLES POUR LA RÉSOLUTION DE PROBLÈMES D'OPTIMISATION AVEC FONCTIONS COÛTEUSES

Promotrice : Annick Sartenaer

Co-promotrice : Charlotte Beauthier

Mémoire présenté pour l'obtention

du grade académique de master en sciences mathématiques

à finalité spécialisée en Project Engineering

Noémie Lambert

Août 2023

Remerciements

Je tiens à remercier tout particulièrement ma promotrice, Annick Sartenaer, ainsi que ma co-promotrice, Charlotte Beauthier, pour leur dévouement et leur disponibilité tout au long de ce mémoire. Merci à toutes les deux pour votre soutien sans faille car c'est grâce à vous que je suis parvenue à aller jusqu'au bout. Je vous remercie également beaucoup pour vos relectures et l'accompagnement dont j'ai pu bénéficier.

De plus, une autre personne que je tiens à remercier chaleureusement pour toute son aide et les réponses aux nombreuses questions que je lui ai posées est Sylvério Pool Marquez. Merci pour toutes les contributions que tu as apportées à la réalisation de ce mémoire.

Enfin, je souhaite remercier mes parents, Emilie, ma soeur jumelle, ainsi que Margaux, Chloé et Stéphanie, mes amies. Merci de m'avoir toujours soutenue. Mes derniers remerciements vont à mon compagnon, Sébastien, qui a toujours été là également pour m'encourager dans ce travail.

Résumé

Réalisé en collaboration avec le centre de recherche Cenaero, ce mémoire cherche à implémenter, tester et comparer l'utilisation de modèles, appelés méta-modèles, efficaces pour la résolution de problèmes d'optimisation faisant intervenir des fonctions coûteuses et des variables de nature différente. Afin d'atteindre ces objectifs, nous avons effectué des investigations sur deux méthodes issues de la littérature que sont les techniques d'arbre de régression et de forêt aléatoire d'arbres de régression. Les tests que nous avons effectués renvoient la performance, en termes de validation externe et d'optimisation, de différentes versions des modèles d'arbre de régression et de forêt aléatoire d'arbres de régression que nous avons créées en fixant les valeurs de certains hyperparamètres de ces modèles à des valeurs autres que leur valeur par défaut. En manipulant les valeurs des hyperparamètres de ces modèles, le but que nous avons poursuivi est de parvenir à trouver des versions de ces modèles qui seraient plus performantes que la version par défaut de Minamo, nommée trbf. Nous définissons Minamo comme étant le logiciel d'optimisation de Cenaero. Les tests effectués ont été réalisés grâce au supercalculateur Zenobe, mis à notre disposition par Cenaero.

Les résultats obtenus en termes de validation externe montrent que les meilleures valeurs testées pour les hyperparamètres étudiés de l'arbre de régression sont celles qui se rapprochent des valeurs par défaut de ces hyperparamètres. Par ailleurs, les résultats relatifs au modèle de forêt aléatoire montrent que la qualité des versions testées augmente avec le nombre d'arbres qui constituent la forêt aléatoire. Les résultats pour l'optimisation, quant à eux, ont permis de sélectionner trois meilleures versions parmi celles testées. Cependant, lorsque nous comparons ces trois versions à la version par défaut de Minamo, celle-ci s'avère plus performante. Nous mentionnons également dans ce mémoire quelques problèmes tests particuliers pour lesquels certaines des versions testées se sont montrées plus performantes que la version trbf. En validation externe et en optimisation, les résultats montrent une meilleure performance du modèle de forêt aléatoire par rapport à l'arbre de régression.

Abstract

Carried out in collaboration with the Cenaero research center, this thesis seeks to implement, test and compare the use of models, known as meta-models, that are effective in solving optimization problems involving black-box functions and mixed variables. In order to achieve these objectives, we have investigated two methods from the literature, namely the regression tree and random forest regression techniques. The tests we have conducted report on the performance, in terms of external validation and optimization, of different versions of the regression tree and random forest regression models that we have created by setting the values of certain hyperparameters of these models to values other than their default values. By manipulating the values of the hyperparameters of these models, our aim was to find versions of these models that would perform better than the default version of Minamo, named trbf. We define Minamo as Cenaero's optimization software. The tests were run using the Zenobe supercomputer provided by Cenaero.

The results obtained in terms of external validation show that the best values tested for the studied hyperparameters of the regression tree are those that are close to the default values of these hyperparameters. Furthermore, the results for the random forest model reveal that the quality of the versions tested increases with the number of trees that compose the random forest. The results for the optimization, meanwhile, allowed us to select three of the best versions tested. However, when we compare these three versions with the default Minamo version, the latter performs better. We also mention in this thesis a few specific test problems for which some of the versions tested outperformed the trbf version. In external validation and optimization, the results highlight the better performance of the random forest model compared with the regression tree.

Table des matières

Introduction

Ce mémoire vise à étudier des modèles qui servent à approximer des fonctions. Ceux-ci sont appelés des "méta-modèles". Nous les étudions dans le cadre de la résolution de problèmes d'optimisation avec ou sans contraintes et faisant intervenir des fonctions coûteuses. De plus, nous nous intéressons particulièrement aux méta-modèles efficaces en présence de variables mixtes, c'est-à-dire de nature différente. Dans la littérature, les méthodes prometteuses que nous allons développer sont les techniques de machine learning relatives au modèle d'arbre de régression et de forêt aléatoire d'arbres de régression. Afin d'atteindre les divers objectifs du mémoire qui sont d'implémenter, de tester mais aussi de comparer ces différents méta-modèles, nous avons collaboré avec le centre de recherche appliqué, Cenaero. Ce dernier a mis à notre disposition le supercalculateur Zenobe sur lequel l'ensemble des tests de la phase d'expérimentation ont été effectués.

Nous introduisons tout d'abord ce mémoire par une brève mise en contexte qui rappelle le thème et les divers objectifs que nous poursuivons. De plus, le problème d'optimisation que nous considérons tout au long du mémoire y est détaillé. Dans ce chapitre, l'algorithme utilisé par le logiciel d'optimisation de Cenaero, nommé Minamo, est expliqué. Une présentation de Cenaero, Minamo et Zenobe constitue le second chapitre du mémoire. Le troisième chapitre est consacré au modèle d'arbre de régression. Nous développons cette technique de machine learning au travers de sa terminologie, sa description et son utilisation dans un premier temps. Nous expliquons et illustrons a posteriori dans ce chapitre toutes les étapes de la construction d'un tel méta-modèle à l'aide d'un exemple simple que nous avons nous-même élaboré. Nous terminons ce chapitre avec l'explication de la construction d'un arbre de régression plus complexe, qui représente davantage la situation des modèles étudiés lors de la phase d'expérimentation. Le quatrième chapitre du mémoire concerne la forêt aléatoire d'arbres de régression. Il s'agit du second méta-modèle sur lequel nous nous penchons dans le cadre du mémoire. Nous nous intéressons au concept qui se cache derrière ce sujet et à la création d'une telle forêt. Nous terminons ce quatrième chapitre en énonçant quelques avantages et inconvénients de la forêt aléatoire d'arbres de régression. Le chapitre suivant, quant à lui, développe les hyperparamètres associés à ces deux méta-modèles que sont l'arbre de régression et la forêt aléatoire d'arbres de régression. Ce chapitre est scindé en deux parties : l'une concerne l'explication des hyperparamètres que nous avons étudiés lors de nos tests, et l'autre détaille les caractéristiques des hyperparamètres non étudiés et explique les raisons qui ont motivé notre choix de ne pas les sélectionner pour la phase d'expérimentation.

Les trois derniers chapitres du mémoire concernent la phase d'expérimentation. Nous précisons que nos expériences ont été dirigées selon deux aspects. En effet, nous avons évalué la perfomance des modèles en étudiant la qualité de ceux-ci en termes de validation externe d'une part et d'optimisation d'autre part. Le sixième chapitre expose les détails des problèmes que nous avons sélectionnés pour la phase de tests sur Zenobe. Ce mémoire se poursuit ensuite avec un chapitre relatif aux outils d'analyse des résultats. Nous y expliquons les éléments graphiques et quantitatifs dont nous disposions grâce à des rapports automatiques de Cenaero générés à partir des résultats obtenus sur Zenobe. Un ultime chapitre présente les résultats des tests effectués sur les modèles d'arbre de régression et de forêt aléatoire d'arbres de régression. Ce chapitre rapporte les résultats obtenus en validation externe et en optimisation. Enfin, les conclusions que nous avons pu tirer de la phase d'expérimentation clôturent ce mémoire. Nous accompagnons ces conclusions de perspectives.

Chapitre 1

Mise en contexte

Ce premier chapitre a pour but de fixer le cadre et de définir le contexte d'étude du mémoire. Tout d'abord, nous allons introduire le thème et expliciter les différents objectifs que nous poursuivons. Ensuite, nous détaillerons le problème d'optimisation considéré ainsi que l'algorithme de résolution utilisé. La rédaction de ce premier chapitre a été réalisée sur base des documents [1], [3] et [4].

1.1 Thème et objectifs du mémoire

Le sujet du mémoire est d'explorer des méta-modèles capables de traiter des variables de nature mixte dans le cadre de la résolution de problèmes d'optimisation avec ou sans contraintes et faisant intervenir des fonctions coûteuses. Dans cette section, nous allons définir brièvement les différents mots-clés de ce sujet. Une explication plus détaillée de ces concepts sera donnée dans le chapitre 2. De plus, nous pouvons souligner ici également le fait que ce mémoire offre une expérience en collaboration directe avec Cenaero [1]. Cette entreprise, qui constitue un centre de recherche, développe principalement des méthodes et outils de simulation pour ses clients dans les domaines de l'aéronautique, l'énergie, le bâtiment et la santé, entre autres.

Les méta-modèles, qui sont aussi appelés modèles de substitution ou surfaces de réponses, vont être utilisés pour répondre au défi lié au coût calcul de la simulation numérique. Il s'agit de modèles de prédiction. Grâce à ceux-ci, la fonction objectif ainsi que les contraintes d'un problème d'optimisation pourront être évaluées de façon approchée en un temps réduit. Les méta-modèles sont très importants dans le cadre de la stratégie d'optimisation de Minamo, qui est le logiciel interne d'optimisation multi-disciplinaire de Cenaero pouvant résoudre des problèmes d'optimisation pour des applications complexes de l'industrie.

En outre, afin d'aborder le problème sous tous ses angles, une deuxième difficulté s'ajoutera dans ce mémoire à celle du temps de calcul ; il s'agit de la présence potentielle de variables mixtes, c'est-à-dire de nature différente. Parmi ces variables, nous comptons

les variables continues, entières, discrètes ou encore catégorielles. Ces dernières sont des variables qui disposent d'attributs tels que le bois, l'acier ou le verre, par exemple. Ce mémoire s'intéresse à l'étude de méta-modèles efficaces en présence de variables mixtes et en particulier de variables catégorielles.

Les fonctions coûteuses sont des fonctions dont le coût de calcul de leur simulation numérique est élévé. En particulier, ces fonctions ne disposent pas d'une expression analytique connue. Lors de la résolution d'un problème d'optimisation faisant intervenir de telles fonctions, les méthodes requérant des informations sur le gradient ou le hessien de ces fonctions ne sont donc pas applicables. Un choix judicieux de l'algorithme utilisé pour résoudre ce type de problème sera dès lors nécessaire, comme nous le verrons dans la suite de ce mémoire. Ces fonctions sont considérées comme des boîtes noires ("black box", en anglais) et sont issues pour la plupart de l'industrie.

Les divers objectifs du mémoire sont donc d'examiner, mais aussi d'implémenter, de tester et de comparer, plusieurs types de méta-modèles aptes à traiter des variables de nature mixte dans le cadre de la résolution de problèmes d'optimisation faisant intervenir des fonctions coûteuses. Les méta-modèles qui ont été étudiés et qui seront développés dans le cadre de ce mémoire sont les modèles d'arbre de régression ("regression tree", en anglais) et de forêt aléatoire ("random forest", en anglais) d'arbres de régression.

1.2 Problème d'optimisation

Le problème d'optimisation que nous considérons a pour formulation mathématique l'expression suivante :

$$\begin{cases} \min\limits_{x \in D^n} & f(x) \\ \text{s.c.} & c_i(x) \geq 0 \quad \forall i \in I \\ & c_j(x) = 0 \quad \forall j \in E \\ & x_{inf} \leq x \leq x_{sup}, \end{cases} \tag{1.1}$$

où f représente la fonction objectif à minimiser. Nous précisons que $f(x) \in \mathbb{R}$ pour tout $x = (x_1, x_2, \ldots, x_n) \in D^n$. Les contraintes d'inégalité et d'égalité sont modélisées par les fonctions c_i indicées par l'ensemble I et c_j indicées par l'ensemble E, respectivement. Notons que dans le cas de problèmes d'optimisation sans contraintes, les ensembles I et E sont vides. Les vecteurs de D^n notés x_{inf} et x_{sup} correspondent aux frontières imposées sur l'ensemble des solutions admissibles. Notons que le problème d'optimisation (1.1) ne dispose pas de ces frontières pour des variables catégorielles. Nous entendons par solution admissible une solution qui respecte toutes les contraintes. Nous définissons l'espace de recherche de f, noté D^n, comme étant l'union de l'espace des solutions admissibles avec le domaine de f représenté par la dernière ligne de (1.1).

Dans le cadre de ce mémoire, nous nous intéressons aux algorithmes capables de déterminer une solution globale et non locale du problème (1.1), ou du moins d'en trouver

une approximation. Rappelons qu'une solution globale du problème (1.1), notée $x_g \in D^n$, vérifie la définition suivante :

$$\forall x \in D^n : f(x_g) \leq f(x).$$

Par ailleurs, une solution locale du problème (1.1), notée $x_l \in D^n$, minimise la fonction objectif du problème dans un voisinage de cette solution tout en respectant les contraintes imposées par les fonctions c_i et c_j. Mathématiquement, la définition de x_l s'énonce comme suit :

$$\exists\, V \text{voisinage de } x_l \in D^n \text{ tel que } \forall x \in V \cap D^n : f(x_l) \leq f(x).$$

Dans la section suivante de ce chapitre, nous allons décrire l'algorithme d'optimisation utilisé pour résoudre le problème (1.1).

1.3 Algorithme génétique

L'algorithme génétique est l'algorithme utilisé dans le cadre de la stratégie d'optimisation de Minamo. Ce type d'algorithme fait partie de la classe des méthodes heuristiques. Dans les paragraphes suivants, nous allons tout d'abord décrire brièvement les méthodes heuristiques puis nous expliquerons ce qu'est un algorithme génétique.

Les méthodes heuristiques sont des méthodes itératives qui permettent d'obtenir une approximation d'une solution du problème (1.1) de bonne qualité et dans un temps raisonnable. Cependant, ces méthodes ne nous permettent pas de nous assurer que la solution approchée obtenue est la meilleure. De telles méthodes sont dites stochastiques car deux exécutions successives de ces algorithmes à partir de conditions initiales identiques sont susceptibles de fournir des résultats différents. En exécutant plusieurs fois une méthode heuristique, nous pouvons obtenir une liste des solutions approximées possibles et ainsi déterminer laquelle répond le mieux au problème en terme d'optimisation globale. Une classe d'algorithmes parmi les méthodes heuristiques connues est représentée par l'ensemble des algorithmes évolutionnaires.

Les algorithmes génétiques, quant à eux, forment une sous-classe des algorithmes évolutionnaires. Ces derniers s'inspirent de la théorie de l'évolution darwinienne. Les algorithmes génétiques sont des méthodes itératives qui définissent une solution approchée x^* d'un problème d'optimisation comme un individu d'une population. Le nombre initial d'individus considérés au départ de l'algorithme correspond à la taille de la population. De plus, chaque individu dispose de gènes. Le nombre de gènes est identique à la dimension du problème d'optimisation étudié, c'est-à-dire au nombre de variables de ce problème. Dans le cas de l'étude du problème d'optimisation (1.1), le nombre de variables est n. Cette méthode est coûteuse car il faut évaluer la fonction à minimiser et les contraintes en chaque nouvel individu de la population à chaque itération. Cependant, le fait d'évaluer ces fonctions en chaque individu va permettre l'obtention d'un meilleur itéré. Dans le contexte de Minamo, une fonction appelée "Global Objective" ou "GO" est similaire à

la fonction habituellement nommée "fitness" dans le cadre des algorithmes génétiques. La GO permet de représenter au mieux la performance d'un individu en lui attribuant une valeur qui combine son évaluation par la fonction objectif et une mesure du non-respect des contraintes de cet individu. L'idée d'un algorithme génétique est d'itérer le processus de génération d'une population en la faisant évoluer. Ce processus contient dès lors trois étapes. Il y a d'abord la sélection de deux parents, puis leur croisement afin de créer un ou plusieurs enfants, et enfin la mutation potentielle des enfants. Tout ce cycle nous permet d'obtenir une nouvelle population qui sera la base de la génération suivante. L'algorithme génétique fonctionne de manière à ce que les meilleurs individus donnent des enfants et les moins bons soient retirés de la population.

Chapitre 2

Présentation de Cenaero, Minamo et Zenobe

La première section de ce chapitre a été développée sur base de la présentation faite dans le mémoire [4] et également sur base des informations recueillies dans [5]. Ainsi, nous pouvons introduire le centre de recherche appliquée Cenaero. Dans un second temps, une présentation du logiciel Minamo est établie grâce à un document interne de Cenaero [6], aux diapositives [7] et au mémoire [34]. Enfin, les informations sur Zenobe reprises dans ce chapitre sont tirées de [5], [15] et [16].

2.1 Cenaero

Cenaero est un centre de recherche appliquée créé en 2002 et situé à Gosselies, près de l'aéroport de Charleroi. Il a été fondé grâce au soutien des fonds structurels européens et par plusieurs partenaires publics (région wallonne, universités en Fédération Wallonie-Bruxelles) et privés (membres de l'EWA, IGRETEC). Il dispose d'une infrastructure de calcul de pointe via un supercalculateur nommé Zenobe. De plus, nous rappelons ici que l'objectif de cette entreprise est de construire des outils de simulation performants pour ses partenaires industriels.

Cenaero fournit ses services principalement dans les domaines de l'aéronautique mais aussi dans le transport (terrestre, automobile et ferroviaire), le bâtiment, l'énergie, la santé et le développement durable. Le centre dispose également d'une filiale située à Moissy-Cramayel à Paris. Ce centre de recherche possède plusieurs départements dont notamment un destiné à la conception et l'optimisation multi-disciplinaire, un autre pour la modélisation, la simulation et l'optimisation de procédés de fabrication et un département de calcul haute performance. Parmi les employés de cette société, 50% sont titulaires d'une thèse de doctorat.

Cenaero a développé les logiciels avancés Argo, Morfeo et Minamo dans le but de pouvoir répondre à certaines demandes de ses clients. Le premier a été conçu afin de fournir

des simulations numériques en mécanique des fluides. Morfeo est un logiciel permettant de simuler des structures pour la modélisation de composants en grande dimension ou encore pour la modélisation de procédés tels que l'usinage ou le soudage. Minamo, quant à lui, a pour mission de résoudre les problèmes de conception faisant intervenir un processus d'optimisation. Ce dernier logiciel fait d'ailleurs l'objet du point suivant de ce mémoire.

2.2 Minamo

Dans cette section, nous reviendrons tout d'abord sur la méthode d'optimisation utilisée par Minamo pour résoudre le problème (1.1). Nous apporterons quelques informations complémentaires à cette méthode et nous justifierons les choix opérés par Cenaero au sein de son logiciel d'optimisation. Ensuite, nous expliquerons ce que représente le "plan d'expériences" utilisé par Minamo lors de la phase de construction des méta-modèles. Enfin, nous décrirons le processus de construction des méta-modèles au sein de Minamo et nous évoquerons les méta-modèles utilisés actuellement par Minamo. Nous ajouterons également à cela une explication concernant la définition des variables mixtes dans Minamo.

2.2.1 Optimisation assistée par méta-modèles

Nous remarquons que Minamo est à la fois le nom d'un logiciel avancé développé par Cenaero mais aussi le nom qui a été attribué à l'équipe au sein de l'entreprise qui travaille avec ce logiciel. Cette équipe est d'ailleurs "dédiée au développement de la plate-forme d'optimisation et d'exploration de l'espace de conception de Cenaero", comme mentionné dans [3]. C'est dans le cadre d'applications industrielles complexes que le logiciel Minamo exerce ses capacités à résoudre des problèmes d'optimisation. Dans les paragraphes suivants, nous allons développer les éléments qui motivent l'intervention de méta-modèles dans le cadre de la stratégie d'optimisation de Minamo et nous détaillerons les raisons qui justifient le choix de Minamo d'utiliser l'algorithme génétique comme technique d'optimisation.

Il est important de rappeler ici que Minamo implémente un algorithme génétique afin de résoudre le problème décrit par (1.1). Ce type d'algorithme heuristique nécessite d'évaluer un grand nombre de fois la fonction objectif et les contraintes du problème étudié. Cependant, le problème d'optimisation que nous considérons fait intervenir des fonctions coûteuses qui ne peuvent donc pas être évaluées autant de fois que nous le souhaitons. Afin de contrer ce problème, Minamo se sert de la méthode nommée "optimisation assistée par méta-modèles" (ou "Surrogate-Based Optimization" (SBO), en anglais). Cette technique consiste en une boucle d'optimisation, représentée par un algorithme génétique, qui exploite les informations fournies par les méta-modèles. En d'autres mots, les méta-modèles vont servir, au sein de la boucle d'optimisation, de substituts aux fonctions du problème (1.1). Les méta-modèles forment donc, dans ce sens, une solution qui a été élaborée afin de répondre au défi lié au coût calcul de la simulation numérique des fonctions que nous considérons dans le cadre de ce mémoire. Ils répondent à ce défi en remplaçant

la simulation numérique par une approximation des fonctions coûteuses. Nous noterons ces approximations $\tilde{f}$, $\tilde{c_i}$, $\tilde{c_j}$ pour la fonction objectif, les contraintes d'inégalité et les contraintes d'égalité, respectivement.

Plusieurs raisons motivent Minamo à utiliser les algorithmes génétiques pour résoudre des problèmes d'optimisation dans le cadre d'applications industrielles. D'une part, le choix d'une méthode heuristique s'explique par le fait que nous nous plaçons dans le contexte d'une optimisation globale. D'autre part, les fonctions objectifs et les contraintes venant de l'industrie font souvent référence à des fonctions de type boite noire. Etant donné que seuls les entrées et l'évaluation des entrées par ces fonctions sont connues, nous ne pouvons désormais pas faire usage de toute méthode nécessitant des informations sur le gradient ou le hessien de la fonction objectif et des contraintes pour résoudre ce problème, comme nous l'avons déjà explicité précédemment. Par conséquent, le choix d'une méthode heuristique telle que l'algorithme génétique semble adéquat.

2.2.2 Plan d'expériences

Nous allons maintenant consacrer cette partie du mémoire à l'explication du "plan d'expériences" ("Design Of Experiments" (DOE), en anglais) sur base duquel se créent les méta-modèles au sein de la boucle d'optimisation de Minamo. Le DOE contient un certain nombre d'individus issus de l'espace de recherche ainsi que les évaluations de ceux-ci par la fonction f et les fonctions associées aux contraintes d'égalité et d'inégalité du problème d'optimisation (1.1). Ces informations contenues dans le DOE forment une base de données qui sera mise à jour au cours des itérations de l'algorithme de Minamo pour construire des modèles de substitution. Afin de créer un DOE, nous allons échantillonner l'espace de recherche considéré. Pour cela, nous disposons de trois méthodes :

- l'échantillonnage par hypercubes latins ("Latin Hypercube Sampling" (LHS), en anglais) ;
- la tesselation de centroïdes de Voronoï ("Centroidal Voronoi Tessellation" (CVT), en anglais) ;
- la latinisation de la tesselation centroïde de Voronoï (LCVT).

La FIGURE 2.1 représente ces trois méthodes pour un échantillonnage de cent individus (représentés par des points sur le schéma) en 2D. La méthode LHS échantillonne l'espace de conception en le découpant en différentes zones aléatoires. Le nombre de zones correspond au nombre d'individus introduits. L'inconvénient de cette méthode est que certaines zones peuvent être non couvertes lorsque peu d'individus sont générés. Cela est dû au caractère aléatoire de ces zones. La deuxième méthode, quant à elle, disperse les individus de façon uniforme dans l'espace. L'inconvénient de cette deuxième méthode est que, comme le processus ne s'intéresse qu'aux centroïdes des régions, aucun individu ne sera répertorié au bord. Enfin, la méthode LCVT est une méthode qui combine les deux méthodes précédentes. Elle permet ainsi d'obtenir une meilleure répartition des individus.

Il est donc naturel que ce dernier type d'échantillonnage soit préféré par Minamo en dépit des deux autres.

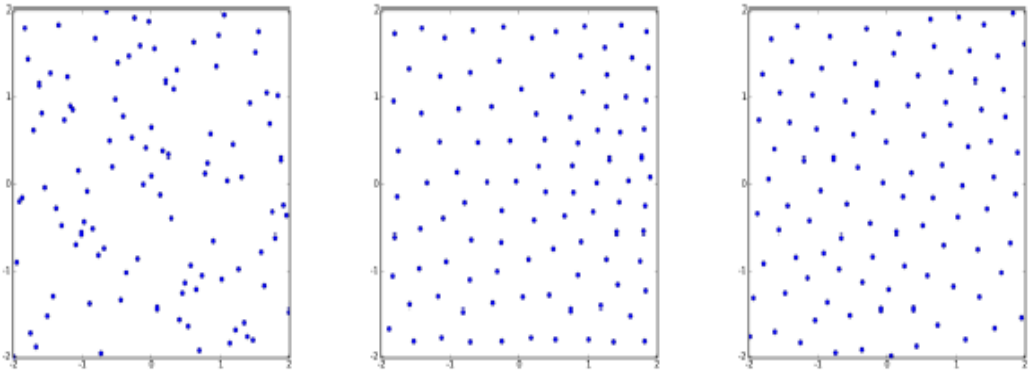

Figure 2.1 – Réprésentation de gauche à droite des méthodes LHS, CVT et LCVT pour construire le plan d'expériences [6].

2.2.3 Méta-modèles

Nous décrivons ici le fonctionnement en cinq étapes de la boucle d'optimisation assistée par méta-modèles de Minamo. Cela nous permettra en même temps de comprendre comment les méta-modèles, c'est-à-dire les approximations $\tilde{f}$, $\tilde{c_i}$ et $\tilde{c_j}$, sont créés et améliorés de façon continue au cours de ce processus itératif. La première étape de la boucle SBO de Minamo consiste à concevoir le DOE initial contenant un certain nombre d'individus issus de l'espace de recherche et en lesquels nous évaluerons les fonctions f, c_i et c_j. Ce DOE sera donc créé sur base de la méthode d'échantillonnage LCVT. Lors de la seconde étape, la première version des méta-modèles $\tilde{f}$, $\tilde{c_i}$ et $\tilde{c_j}$ est générée sur base du DOE et à l'aide de méthodes telles que l'arbre de régression ou la forêt aléatoire d'arbres de régression que nous développerons dans les chapitres 3 et 4, respectivement. Le problème d'optimisation faisant intervenir les fonctions approchées $\tilde{f}$, $\tilde{c_i}$ et $\tilde{c_j}$ est ensuite résolu via un algorithme génétique. Ceci constitue la troisième étape de la boucle d'optimisation. La solution approchée obtenue, notée $\tilde{x}$, est ensuite récupérée et les fonctions f, c_i et c_j sont évaluées en cette solution. Cette simulation coûteuse représente la quatrième étape de la méthode que nous développons. Enfin, dans la dernière étape de la boucle SBO, le respect de critères d'enrichissement de la base de données est vérifié afin de pouvoir mettre à jour le DOE en y intégrant le nouvel individu $\tilde{x}$ puis, une version améliorée des approximations $\tilde{f}$, $\tilde{c_i}$ et $\tilde{c_j}$ est générée sur base de ce nouveau DOE. Ces cinq étapes de la construction d'un méta-modèle sont répétées jusqu'à ce qu'un critère d'arrêt soit rencontré. Ce critère d'arrêt correspond dans Minamo à un nombre maximal d'itérations fixé. L'enrichissement de la base de données au cours de ce processus permet donc à terme de rendre les approximations $\tilde{f}$, $\tilde{c_i}$ et $\tilde{c_j}$ de plus en plus performantes au cours des itérations. Il est important de noter que les méta-modèles désignent non seulement les approximations

$\tilde{f}$, $\tilde{c_i}$ et $\tilde{c_j}$ mais ils désignent aussi, par abus de langage, la méthode utilisée pour construire ces approximations, à savoir l'arbre de régression ou la forêt aléatoire.

Afin d'expliciter dans ce paragraphe les méta-modèles qui sont utilisés actuellement au sein de Minamo, nous nous servirons du travail déjà réalisé par Y. Derlet dans le cadre de son mémoire [4] à ce sujet. Les méta-modèles utilisés actuellement au sein de Minamo sont construits sur base de "fonctions à base radiale" ("radial basis function" (rbf), en anglais). Ces modèles de substitution rbf, dénotés par $\tilde{y}$, peuvent être définis comme une combinaison linéaire de p fonctions à base radiale h, où p représente le nombre d'individus qui doivent être introduits dans le DOE. Autrement dit, les méta-modèles de type rbf s'expriment comme :

$$\tilde{y}(x) = \sum_{k=1}^{p} \omega_k h(\|x - x_k\|_2, \sigma_k),$$

où ω_k représente le poids attribué à la k-ième fonction à base radiale, elle-même associée à l'individu x_k du DOE. De plus, nous pouvons noter que x est un vecteur de D^n. Les fonctions à base radiale sont désignées par $h(r, \sigma)$ où r est la base radiale et σ un paramètre.

En outre, nous allons nous intéresser dans ce mémoire aux méta-modèles tels que la forêt aléatoire d'arbres de régression. Cependant, pour comprendre comment fonctionne une forêt d'arbres de régression, nous avons d'abord besoin de comprendre ce que sont les arbres de régression qui la constituent et qui pourront également être utilisés comme méta-modèles au sein de Minamo. Ces deux modèles, l'arbre de régression et la forêt aléatoire d'arbres de régression, feront l'objet de notre étude dans les chapitres suivants. Ils seront implémentés puis testés sur Zenobe, le supercalculateur Tiers-1 mis à disposition par Cenaero pour lancer les expérimentations dans le cadre de ce mémoire. Lors de la phase d'expérimentation, nous avons également comparé nos résultats au modèle trbf ("tuned-rbf", en anglais) de Minamo. Il s'agit d'une version similaire au modèle rbf que nous avons décrit mais pour laquelle un choix des valeurs optimales de h et de σ a été effectué au préalable. Afin de réaliser ce choix, plusieurs modèles avec différentes valeurs de h et σ sont tout d'abord générés. Ensuite, le modèle trbf est défini comme étant le modèle disposant de la meilleure qualité parmi tous les modèles créés. Cette qualité est mesurée à l'aide de la technique de validation croisée "Leave-one-out" (LOO).

2.2.4 Variables mixtes

Grâce à [7], nous expliquons les différentes types de variables considérées dans Minamo. Les lettres R, I, D, C sont associées à chacun de ces types de variables dont voici la description :

- **"R"** : les variables de type "R" (pour "Real", en anglais) prennent leur valeur dans des intervalles de nombres réels. Par exemple, une variable x de type R est telle que $x \in [-1, 80]$, où $[-1, 80]$ est un intervalle contenant une infinité de nombres réels.

- **"I"** : les variables de type "I" (pour "Integer", en anglais) prennent des valeurs entières. Par exemple, une variable x de type I est telle que $x \in [-1, 80]$, où $[-1, 80]$ contient tous les nombres entiers allant de -1 à 80 inclus.
- **"D"** : les variables de type "D" (pour "Discrete", en anglais) prennent leur valeur dans des ensembles finis de nombres réels ou bien correspondent à des variables catégorielles ordonnées. Par exemple, une variable x de type D est telle que $x \in \{0.2, 3, 4.6, 2.55\}$. Un autre exemple serait de considérer que $x \in$ {"petit", "moyen", "grand"}.
- **"C"** : les variables de type "C" (pour "Categorical", en anglais) représentent les variables catégorielles non ordonnées. Par exemple, une variable x de type C est telle que $x \in$ {"acier", "bois", "verre"}.

Il est important dès lors de bien comprendre que les variables que Minamo définit comme étant "catégorielles" font référence à des variables non ordonnées. Les noms des problèmes tests que nous présenterons dans le chapitre 6 seront suivis de ces lettres afin de préciser le type de variables intervenant dans chacun de ces problèmes.

2.3 Zenobe

La mise en place de Zenobe a duré quatre ans et s'est terminée en 2015. Il est ensuite devenu le supercalculateur Tiers-1 de Wallonie. L'élaboration de ce calculateur s'est effectuée en plusieurs étapes. Tout d'abord, le matériel originel a été acquis en 2011 suite à un co-financement par le programme FEDER 2007-2013. Zenobe se constituait alors d'environ 3300 coeurs de calcul. Plus tard en 2013, un investissement fourni par la subvention PRACE [1] a permis d'étendre la machine en y ajoutant 8200 coeurs. Enfin, vers la fin de l'année 2015, un renouvellement des noeuds de calcul a pu être réalisé et 5760 coeurs ont été fournis.

Le document [15] nous informe que Zenobe tient son nom de l'électricien belge Zenobe Gramme qui n'est autre que le créateur de la dynamo. Ce supercalculateur est tenu et utilisé par Cenaero. Son usage a été pensé pour convenir à des travaux de calculs massifs devant s'effectuer en parallèle. Sa capacité de calcul est de l'ordre du milliards d'opérations à la seconde. En 2014, Zenobe a été classé à la 300ème position dans le classement des supercalculateurs les plus puissants au monde.

C'est après plus de 11 ans de service que Zenobe sera désaffecté le 8 août 2023, tandis qu'au moment de la rédaction de ce mémoire, l'arrêt du lancement de calculs sur celui-ci était programmé au 20 mai 2023. Zenobe était donc en fin de vie lors de la finalisation de ce mémoire et cela a eu notamment comme impact un très fort ralentissement de la vitesse de calcul de la machine.

[1] PRACE est l'acronyme de "Partnership for Advanced Computing in Europe". Ce projet a pour objectif de fournir des infrastructures pour le calcul de Haute Performance. La Belgique en est membre depuis octobre 2012 selon [15].

Chapitre 3

L'arbre de régression

L'arbre de régression faisant partie de la famille des modèles d'arbres de décision, nous introduirons ce chapitre en développant tout d'abord la terminologie relative aux arbres de décision. Ensuite, le modèle d'arbre de régression sera décrit et nous expliquerons comment l'utiliser à l'aide d'un exemple simple. Puis, nous expliquerons les différentes étapes de la construction de ce modèle en nous basant toujours sur le même exemple simple. Nous analyserons par après le cas de la construction d'un arbre de régression plus complexe ; un objectif de ce chapitre étant d'expliquer plus en détails comment sont générées les approximations $\tilde{f}$, $\tilde{c_i}$ et $\tilde{c_j}$ lors de la seconde étape de la boucle d'optimisation assistée par méta-modèles de Minamo.

3.1 Terminologie

Nous allons ici expliquer ce que sont les arbres de décision et expliciter leur terminologie. Les arbres de décision font partie des méthodes d'apprentissage automatique ("Machine Learning", en anglais). En particulier, il s'agit de techniques d'apprentissage supervisé ("Supervised Learning", en anglais). Dans le cadre de la science des données, ces arbres vont permettre de résoudre des problèmes à la fois de classification mais aussi de régression. Ils peuvent être définis comme des modèles prédictifs qui utilisent un ensemble de règles pour déterminer une valeur cible.

Avant toute chose, il est important de comprendre qu'un arbre de décision se construit à partir d'une base de données qui seront appellées "données d'entrainement". Ce sont ces données qui vont permettre à l'arbre d'être créé. Cette base de données correspond, dans le cadre de ce mémoire, au DOE sur base duquel l'approximation de $\tilde{f}$, par exemple, va être générée lors de la seconde étape de la boucle d'optimisation SBO de Minamo. Pour rappel, ce DOE contient un certain nombre d'individus issus de l'espace de recherche considéré ainsi que leur évaluation par la fonction f. Nous prenons ici l'exemple du méta-modèle associé à f mais nous aurions pu également prendre celui associé à c_i ou c_j, cela ne change rien au raisonnement qui va suivre dans ce chapitre. La terminologie relative aux arbres de décision veut qu'un élément d'un tel DOE, c'est-à-dire un individu muni de son évaluation

par f, soit appélé "échantillon" de la base de données d'entrainement. Un "noeud" dans un arbre de décision correspond, quant à lui, à un ensemble d'échantillons issus de la base de données d'entrainement. Un arbre de décision est une structure descendante composée de noeuds, comme nous pouvons le voir sur le schéma de la FIGURE 3.1. Il se lit de haut en bas. Il faut imaginer que chaque noeud possède une condition menant vers plusieurs réponses et permet ainsi de nous diriger vers un prochain noeud.

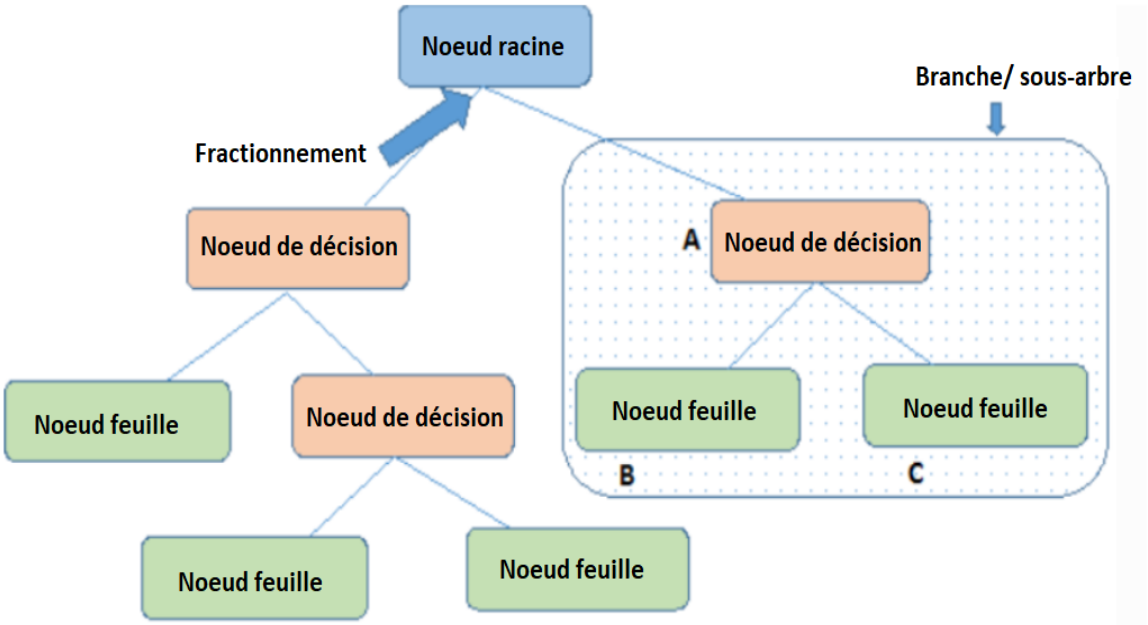

FIGURE 3.1 – Illustration d'un exemple d'arbre de décision [9].

Nous décrivons ci-dessous chacun des mots-clés se trouvant sur le schéma de la FIGURE 3.1 :

- **"nœud racine"** : le noeud racine représente l'ensemble des données d'entrainement. Il se trouve à la base de l'arbre de décision, tout en haut du schéma.
- **"nœud de décision"** : lorsqu'un nœud se divise en d'autres nœuds, il s'agit d'un nœud de décision. En particulier, le noeud racine est également un noeud de décision.
- **"noeud feuille"** : il s'agit d'un noeud qui ne se divise pas en d'autres noeuds. Il peut aussi être appelé "noeud terminal".
- **"fractionnement"** : le fractionnement représente le processus de division d'un nœud en plusieurs autres noeuds.

- **"branche/sous-arbre"** : il s'agit d'une section de l'arbre entier.

Nous précisons qu'un arbre de décision comme celui illustré à la FIGURE 3.1 est dit "binaire" car dans cet arbre, chaque noeud non terminal donne naissance à deux nouveaux noeuds exactement (un noeud à gauche et un noeud à droite). Dans le cadre de ce mémoire, nous travaillerons avec ce type d'arbre de décision et nous étudierons le cas de la régression.

3.2 Description et utilisation

Considérons la base de données d'entrainement reprise à la TABLE 3.1 sous forme d'un tableau qui contient différentes valeurs du dosage en mg d'un complément alimentaire vitaminé qu'une personne peut prendre et son efficacité correspondante sur la personne. Cette base de données s'inspire de celle présente dans [2] et nous la définissons par la lettre Q. Chaque ligne de ce tableau correspond à un échantillon de la base de données d'entrainement. Ces échantillons contenus dans la base de données Q peuvent être modélisés par les couples $(x_i, y_i) \in D^n \times \mathbb{R}$ tels que $i = 1, \ldots, l$ avec $l = 8$ et $n = 1$ dans notre exemple. Dans le cadre de la création du modèle de substitution $\tilde{f}$ à partir du plan d'expériences (DOE) dont nous disposons lors de la seconde étape de la boucle d'optimisation SBO de Minamo, les différentes valeurs du dosage seraient représentées par les individus du DOE et les valeurs correspondantes de l'efficacité du complément alimentaire seraient équivalentes aux évaluations de ces individus par f. Chaque individu ne disposerait donc ici que d'un seul gène car $n = 1$. La base de données d'entrainement présentée ici est simple et contient peu de données. Ce choix a été effectué dans le but de faciliter la compréhension de la description et de l'utilisation du modèle d'arbre de régression. Le dosage ici est ce que l'on appelle un "prédicteur" car c'est en fonction de ce critère que nous allons prédire une valeur pour l'efficacité correspondante des gelules vitaminées. Notre exemple ne dispose que d'un seul prédicteur car $n = 1$ mais nous analyserons le cas d'un arbre de régression plus complexe avec des prédicteurs multiples dans la section 3.4 de ce chapitre.

Dosage (en mg)	Efficacité (en %)
5	10
6	9
13	100
18	100
23	55
27	52
31	23
36	24

TABLE 3.1 – Base de données inspirée de [2].

La FIGURE 3.2 illustre l'arbre de régression créé à l'aide du package *tree* de la librairie ScikitLearn [17] en Python3 sur base de la base de données que nous venons de décrire.

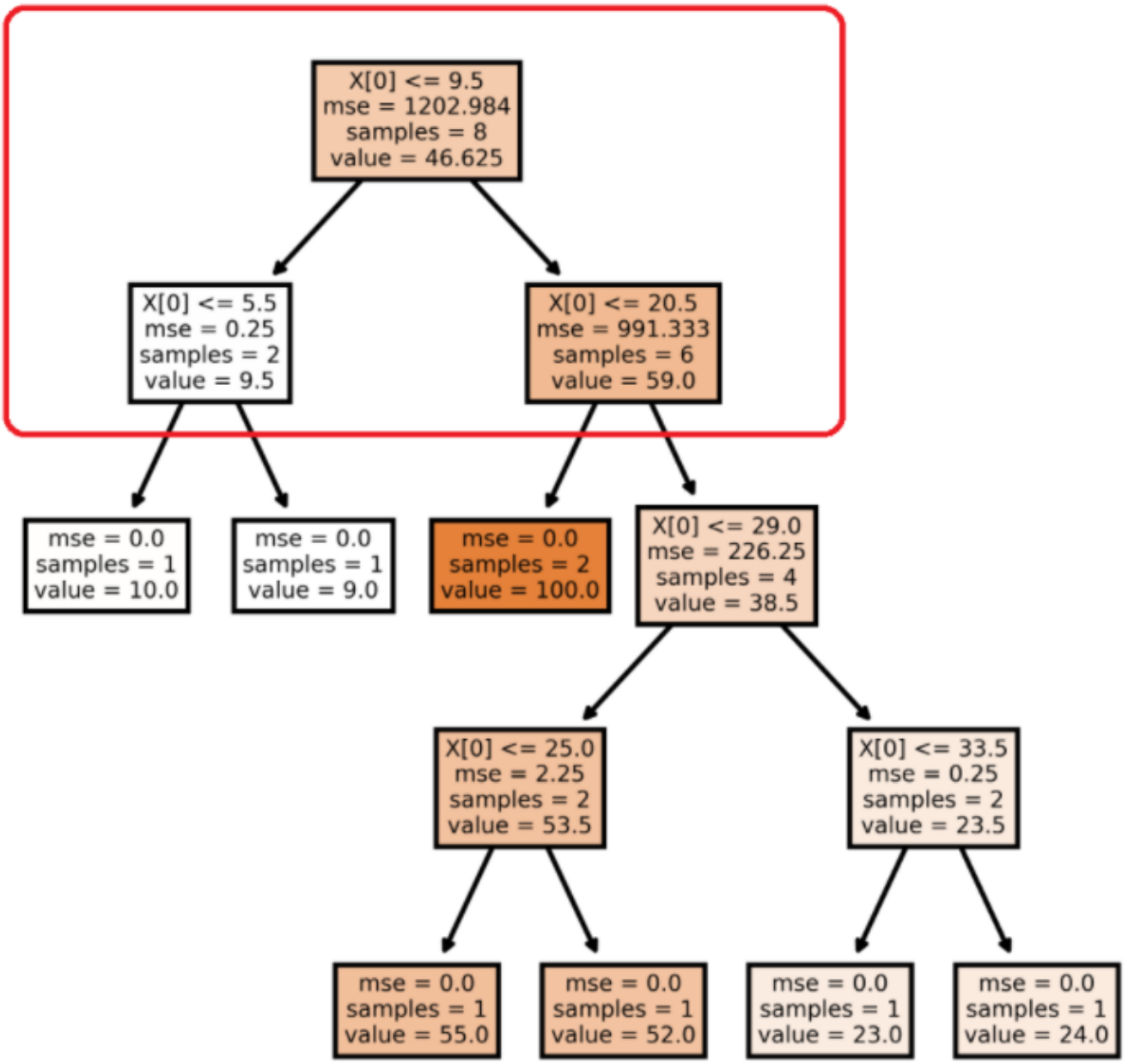

FIGURE 3.2 – Illustration de l'arbre de régression obtenu à l'aide du package *tree* de la librairie ScikitLearn en Python3 [17].

Avant de nous pencher sur la construction elle-même d'un arbre de régression, nous allons expliquer, sur base de l'illustration de la FIGURE 3.2, les différents éléments que nous retrouvons dans les noeuds de cet exemple d'arbre de régression :

- "$X[0]$" : cet élément correspond à une valeur du dosage pour laquelle nous allons chercher à prédire l'efficacité des gelules vitaminées. Il s'agit d'une valeur arbitraire que nous choisissons pour le prédicteur étudié.

- **"mse"** : ce critère correspond à l'erreur quadratique moyenne ("Mean Squared Error", en anglais) de prédiction des échantillons présents dans ce noeud.
- **"samples"** : ce terme se traduit par "échantillons" en français. Le nombre associé à cette variable, dans chaque noeud, correspond au nombre d'échantillons de ce noeud.
- **"value"** : cet élément détermine la valeur prédite de l'efficacité des compléments alimentaires pour un certain dosage $X[0]$. Il s'agit de la moyenne des valeurs de l'efficacité des échantillons de ce noeud.

Nous observons sur le graphe de la FIGURE 3.2 que chaque noeud de décision contient quatre de ces variables tandis que les noeuds feuilles en contiennent trois.

Nous allons maintenant repartir de l'exemple d'arbre de régression représenté à la FIGURE 3.2 afin d'expliquer comment ces modèles sont utilisés. Fixons la valeur de $X[0]$ à 7 mg par exemple et tentons de prédire l'efficacité des gelules en pourcents si la dose administrée au patient est de 7 mg. Pour ce faire, nous devons démarrer en haut de l'arbre et analyser le noeud racine. Ce noeud nous demande si la valeur du prédicteur $X[0]$ que nous avons posée égale à 7 mg est bien inférieure ou égale à 9.5 mg. Il s'agit de la première règle de décision de cet exemple d'arbre de régression. Etant donné que nous pouvons répondre "oui" à cette première règle de décision de l'arbre, nous allons parcourir celui-ci de haut en bas en nous dirigeant d'abord vers la gauche. Nous atteignons désormais le noeud dont la règle de décision est de se demander si la valeur que nous avons attribuée à $X[0]$ est inférieure ou égale à 5.5 mg. Comme ce n'est pas le cas, la prochaine direction que nous allons prendre à partir de ce noeud en descendant l'arbre est la direction de droite cette fois. Nous avons maintenant atteint un noeud feuille car il ne contient plus qu'un seul échantillon (samples = 1) et ne peut donc pas être divisé. Dès lors, aucune règle de décision ne peut être associée à ce noeud car ces règles servent à diviser l'ensemble des échantillons de ce noeud en deux parties. La valeur que nous cherchions à prédire sur l'efficacité des gelules pour une dose de 7 mg administrée à un patient est donc de 9 pourcents car il s'agit de la valeur de la variable "value" inscrite dans le noeud feuille dans lequel nous sommes arrivés.

3.3 Construction

Maintenant que l'utilisation de l'arbre de régression a été explicitée, voyons comment celui-ci se construit. Plus précisément, nous allons expliquer, sur base de notre exemple, comment les seuils dans les règles de décision sont fixés et comment les valeurs de la variable "value" sont calculées. Afin de rendre la compréhension de la construction d'un arbre de régression plus aisée, nous allons d'abord répondre à ces questions en nous focalisant sur la construction du noeud racine contenu dans le sous-arbre encadré en rouge à la FIGURE 3.2. Nous verrons par la suite que la construction de l'arbre en entier n'est qu'une application récursive des étapes ci-dessous à chacun des noeuds de décision qui se crée au fur et à mesure du développement de l'arbre.

Notons que l'explication que nous allons donner à propos de l'algorithme de construction du modèle d'arbre de régression se base sur la documentation [17] de la version 0.20.4 de ScikitLearn. Il s'agit de la version de ScikitLearn utilisée par Zenobe. Cette documentation nous informe que le package *tree* de la librairie ScikitLearn développe dans le langage Python3 un arbre de régression selon une version améliorée de l'algorithme CART ("Classification and Regression Trees", en anglais). Des informations supplémentaires à celles expliquées dans ce mémoire par rapport à cet algorithme peuvent être trouvées dans [13] et [14].

3.3.1 Étape 1 - Partitionnement de la base de données d'entrainement en deux régions pour chacun des seuils possibles

La FIGURE 3.3 représente graphiquement la base de données d'entrainement de la TABLE 3.1 dans un espace à deux dimensions. Nous remarquons qu'un cercle bleu sur la représentation graphique ci-dessous correspond à un échantillon de cette base de données.

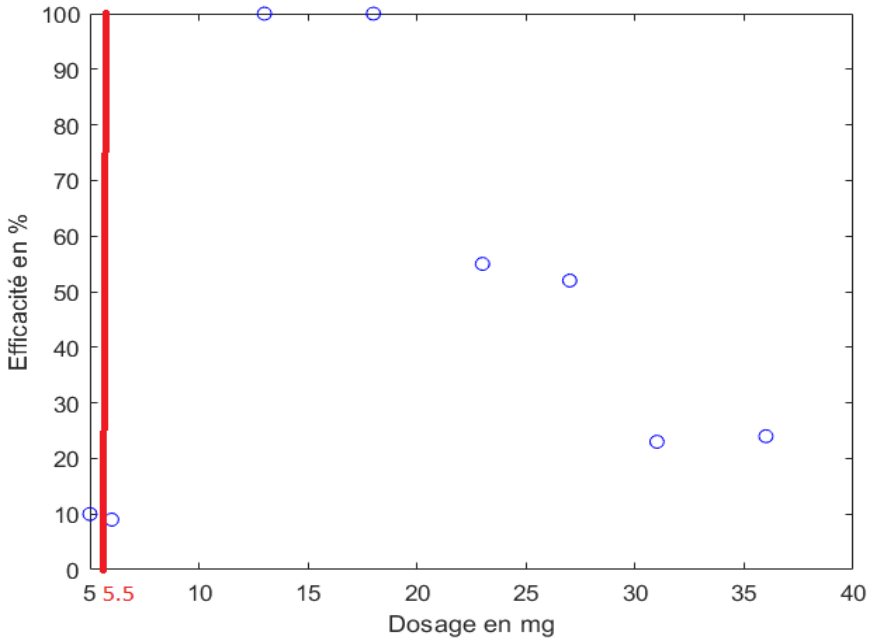

FIGURE 3.3 – Représentation de la base de données de la TABLE 3.1 dans un espace à deux dimensions.

La première étape de la construction de l'arbre de régression illustré à la FIGURE 3.2 consiste à calculer la moyenne des valeurs du dosage des échantillons pris successivement 2 à 2. Remarquons que la succession des échantillons se fait selon la lecture des points de données du graphe de gauche à droite. Chaque moyenne obtenue constituera un seuil candidat à la règle de décision à définir dans le noeud racine du sous-arbre encadré en rouge

sur la FIGURE 3.2. La base de données d'entrainement sera triée automatiquement par l'algorithme de construction de l'arbre de régression selon un ordre croissant des valeurs du dosage de chaque échantillon si elle n'était pas déjà triée au préalable. Sur base de chacune des moyennes obtenues, la base de données d'entrainement sera divisée en 2 sous-groupes d'échantillons ; ceux pour lesquels la valeur du dosage est inférieure à cette moyenne et ceux pour lesquels cette valeur est supérieure à cette moyenne. C'est ainsi que la base de données d'entrainement est partitionnée en deux régions pour chacun des seuils possibles.

Par exemple, le premier seuil candidat pour définir la règle de décision à la racine sera, dans le cas de notre exemple, la moyenne $\frac{5+6}{2} = 5.5$ mg. Il est représenté à la FIGURE 3.3 par la droite verticale rouge qui passe par l'abscisse ayant pour valeur 5.5 mg. Le deuxième seuil candidat sera la moyenne $\frac{6+13}{2} = 9.5$, etc. Les seuils vont donc séparer la base de données en 2 régions, c'est-à-dire en deux ensembles distincts de données qui auront pour caractéristique d'avoir une valeur du dosage correspondante soit inférieure soit supérieure au seuil considéré. Les valeurs des 7 seuils candidats possibles sont dès lors les suivantes : $\frac{5+6}{2} = 5.5$ mg, $\frac{6+13}{2} = 9.5$ mg, $\frac{13+18}{2} = 15.5$ mg, $\frac{18+23}{2} = 20.5$ mg, $\frac{23+27}{2} = 25$ mg, $\frac{27+31}{2} = 29$ mg et $\frac{31+36}{2} = 33.5$ mg.

3.3.2 Étape 2 - Représentation du graphe du total des erreurs quadratiques moyennes pour chacun des seuils possibles

Cette deuxième étape va permettre l'identification du meilleur seuil candidat à la règle de décision du noeud racine. Afin d'expliquer la procédure à suivre lors de cette deuxième étape, reprenons le premier seuil que nous avons calculé d'une valeur de 5.5 mg. Imaginons que ce seuil soit, à l'issue de l'étape 2, le meilleur seuil candidat pour intégrer la règle de décision du noeud racine. Dès lors, la règle de décision au noeud racine sera de savoir si $X[0]$ est inférieur ou égal à 5.5 mg ou non. Si la réponse à cette régle de décision, pour une valeur de $X[0]$ que nous aurons choisie et fixée au préalable, est "oui" alors une branche sera créée dans l'arbre vers la gauche et vers le bas à partir du noeud racine. Le nombre d'échantillons au sein du nouveau noeud de décision ainsi atteint correspondra au nombre d'échantillons du noeud racine dont le dosage correspondant est inférieur ou égal à 5.5 mg. Dans notre exemple, ce nombre vaudra 1 et le noeud de décision atteint sera donc en réalité un noeud feuille. La prédiction de l'efficacité du complément alimentaire sera dès lors de 10 % (il s'agit de la moyenne des efficacités des échantillons qui ont un dosage inférieur ou égal à 5.5 mg). Si maintenant nous souhaitons prédire une valeur de l'efficacité du médicament pour des dosages supérieurs à 5.5 mg, il suffit de calculer la moyenne des valeurs des efficacités associées aux échantillons ayant un dosage supérieur à 5.5 mg. Dans ce cas, cette moyenne vaut

$$\frac{9 + 100 + 100 + 55 + 52 + 23 + 24}{7} = 51.85714286\%$$

et constitue la valeur de prédiction pour des dosages supérieurs à 5.5 mg.

Calculons maintenant la somme des erreurs de prédiction au carré associée au seuil valant 5.5 mg. Il s'agit de la somme des carrés des erreurs de prédiction relatives à l'efficacité des gelules pour chacun des échantillons de la base de données d'entrainement lorsque l'on considère que le seuil du dosage est fixé à 5.5 mg. L'erreur de prédiction est considérée comme la différence entre la valeur de la prédiction et la valeur effective de l'éfficacité des gelules vitaminées pour l'échantillon considéré. Cette erreur au carré vaut $(10-10)^2 = 0$ pour les échantillons dont le dosage est inférieur ou égal au seuil de 5.5 mg. Au carré de cette erreur, il faut lui ajouter les carrés des erreurs de prédiction des échantillons dont le dosage est supérieur au seuil de 5.5 mg :

$$(9-51.857)^2 + (100-51.857)^2 + ... + (52-51.857)^2 + (23-51.857)^2 + (24-51.857)^2,$$

ce qui nous donne une somme totale des erreurs de prédiction au carré associée au seuil de 5.5 mg de 8090.86. Nous allons ensuite calculer ces erreurs pour chacun des seuils possibles et reprendre les valeurs obtenues sur le graphe de la FIGURE 3.4. Sur ce graphe, les valeurs reprises en abscisse correspondent aux 7 seuils candidats possibles calculés lors de l'étape 3.3.1 de la construction d'un arbre de régression, à savoir 5.5 mg, 9.5 mg, 15.5 mg, 20.5 mg, 25 mg, 29 mg et 33.5 mg.

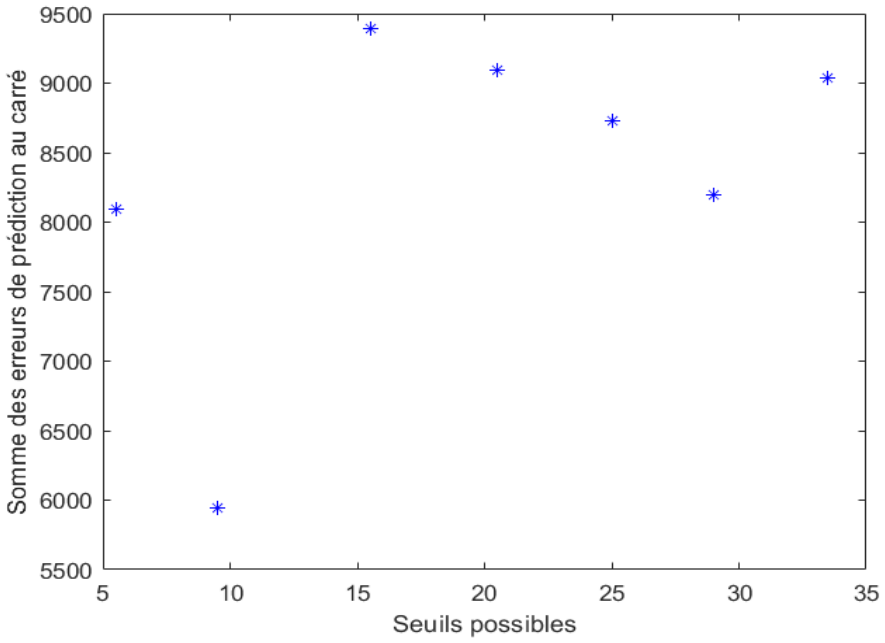

FIGURE 3.4 – Graphique de la somme des erreurs de prédiction au carré pour chaque seuil possible.

En sélectionnant le seuil pour lequel la somme des erreurs de prédiction au carré est la plus petite, nous obtenons finalement le meilleur candidat pour être le seuil de la règle

de décision du noeud racine de l'arbre. Dans notre cas, il s'agit du seuil de 9.5 mg pour lequel la somme des erreurs de prédiction au carré vaut 5948.5. Le seuil que nous avons obtenu correspond bien à celui qui se trouve dans la règle de décision du noeud racine de l'arbre de régression illustré à la FIGURE 3.2.

Seuls deux échantillons de la base de données vérifient le fait que la caractéristique $X[0]$ soit inférieure ou égale à 9.5 mg. Les six autres échantillons de la base de données ont un dosage correspondant strictement supérieur au seuil déterminé de 9.5 mg. La valeur prédite dans le noeud racine correspond à la moyenne des valeurs de l'efficacité de tous les échantillons de la base de données d'entrainement. Celle-ci vaut

$$\frac{10+9+100+100+55+52+23+24}{8} = 46.625\%$$

et signifie que, de prime à bord, sans avoir répondu à une première règle de décision, la prédiction de l'efficacité des gelules vitaminées pour n'importe quelle valeur du dosage sera fixée au départ à 46.625%. Le "mse" d'un noeud correspond au rapport entre la somme des carrés des erreurs de prédiction des échantillons de ce noeud (avec comme prédiction la valeur prédite inscrite dans ce noeud et avec comme valeurs effectives, celles de l'efficacité des gelules pour les échantillons de ce noeud) et le nombre d'échantillons dans ce noeud. Par exemple, le "mse" du noeud racine est calculé de la manière suivante :

$$\frac{(10-46.625)^2+(9-46.625)^2+\cdots+(23-46.625)^2+(24-46.625)^2}{8} = 1202.984.$$

3.3.3 Étape 3 - Construction de l'arbre en entier

Maintenant que nous avons construit le noeud racine qui se trouve dans le sous-arbre encadré en rouge à la FIGURE 3.2, l'idée pour poursuivre le développement de l'arbre en entier sera d'appeler récursivement les étapes 1 et 2 au prochain noeud de décision que nous souhaitons créer afin de calculer le seuil de sa règle de décision. Il faut cependant faire attention à redéfinir la nouvelle base de données sur laquelle travailler comme étant celle qui sera constituée uniquement des échantillons du noeud de décision étudié.

3.4 Cas de prédicteurs multiples

Si nous considérons un problème à deux ou à plus de deux dimensions, c'est-à-dire avec deux ou bien plus de deux prédicteurs, il devient plus difficile de représenter la base de données sur un graphe comme nous l'avons fait précédemment. Cependant, les arbres de régression s'adaptent facilement aux prédicteurs supplémentaires. Nous allons maintenant décrire le processus avec lequel ces arbres de régression vont traiter les bases de données d'entrainement composées de plusieurs prédicteurs.

Imaginons, à titre d'exemple, que nous tentons de prédire l'efficacité de gelules vitaminées sur un patient dont nous connaissons l'âge, le sexe et la dose de vitamines qui

lui a été administrée. Le dosage, l'âge et le sexe sont trois facteurs qui vont influencer la valeur de l'efficacité du médicament sur le patient et vont être utilisés pour prédire cette valeur. Ces trois facteurs peuvent donc être considérés comme des prédicteurs.

Afin de résoudre un tel problème, nous pouvons commencer comme dans le cas d'un seul prédicteur et utiliser d'abord uniquement le dosage pour prédire l'efficacité du complément alimentaire. L'idée est donc de construire le noeud racine contenu dans le sous-arbre comme celui encadré en rouge sur la FIGURE 3.2. Une fois ce premier noeud racine établi, il suffit de faire de même avec les données concernant l'âge, puis le sexe. Nous disposons dès lors de trois noeuds racines. Nous ne devons néanmoins pas oublier de garder en tête la valeur de la somme des erreurs de prédiction au carré associée au seuil qui a été sélectionné pour intégrer la règle de décision de chacun des noeuds racines que nous avons engendrés. Finalement, parmi les trois seuils dont nous disposons, le seuil associé à la plus petite somme des erreurs de prédiction au carré formera le seuil du noeud racine de l'arbre final. Le développement de ce dernier se poursuit ensuite de la même manière que la construction d'un arbre avec un seul prédicteur, à l'exception que désormais, nous comparons la plus petite somme des erreurs de prédiction au carré pour chacun des prédicteurs au moment de la sélection du seuil de la règle de décision de chaque nouveau noeud de décision qui apparait dans l'arbre au cours de son développement.

Chapitre 4

La forêt aléatoire d'arbres de régression

Dans ce chapitre, nous allons expliquer ce qu'est une forêt aléatoire d'arbres de régression. De plus, nous préciserons quelles sont les étapes à suivre afin de créer une telle forêt. Enfin, nous citerons divers avantages et inconvénients liés à ce modèle. Ce chapitre s'inspire principalement de [8] et [11] ainsi que de [12].

4.1 Concept de forêt aléatoire d'arbres de régression

La notion de forêt aléatoire a été introduite pour la première fois dans la littérature par L.Breiman [8] en 2001. L'idée d'une forêt aléatoire d'arbres de régression est de produire une multitude d'arbres de régression différents dont les prédictions individuelles seront combinées pour obtenir une seule prédiction finale. La FIGURE 4.1 illustre une forêt aléatoire d'arbres de régression, où Q représente la base de données d'entrainement initiale à partir de laquelle les différents arbres de régression qui constituent la forêt sont construits. Comme pour le modèle d'arbre de régression, la base de données Q est consituée d'un ensemble d'échantillons qui peuvent être modélisés par les couples $(x_i, y_i) \in D^n \times \mathbb{R}$, où la variable n définit le nombre de prédicteurs, c'est-à-dire le nombre de variables du problème d'optimisation étudié. L'indice i prend des valeurs allant de 1 à l où l est le nombre d'échantillons, c'est-à-dire le nombre d'individus contenus dans le DOE sur base duquel nous allons construire le méta-modèle $\tilde{f}$, par exemple, lors de la seconde phase de la boucle SBO de Minamo. La valeur de y_i correspond à l'évaluation par f du $i$ème individu de la base de données Q. Il est important de préciser qu'au cours du processus de construction d'une forêt aléatoire d'arbres de régression, la base de données d'entrainement initiale Q sera modifiée à l'aide d'un procédé que nous décrirons dans la section suivante de ce chapitre, afin de construire chacun des arbres de la forêt aléatoire. Ce procédé permet la diversification des arbres de la forêt aléatoire. Les arbres de décision d'une forêt aléatoire sont aussi parfois appelés "estimateurs". La FIGURE 4.1 nous indique que la prédiction finale de la forêt aléatoire, notée $\hat{y}$, est définie par

$$\hat{y} = \frac{1}{K} \sum_{k=1}^{K} \hat{y}_k,$$

où K désigne le nombre d'arbres de régression au sein de la forêt aléatoire et $\hat{y}_k$ est la prédiction obtenue pour l'arbre k. Il s'agit donc de la moyenne des prédictions de tous les arbres de régression qui constituent la forêt.

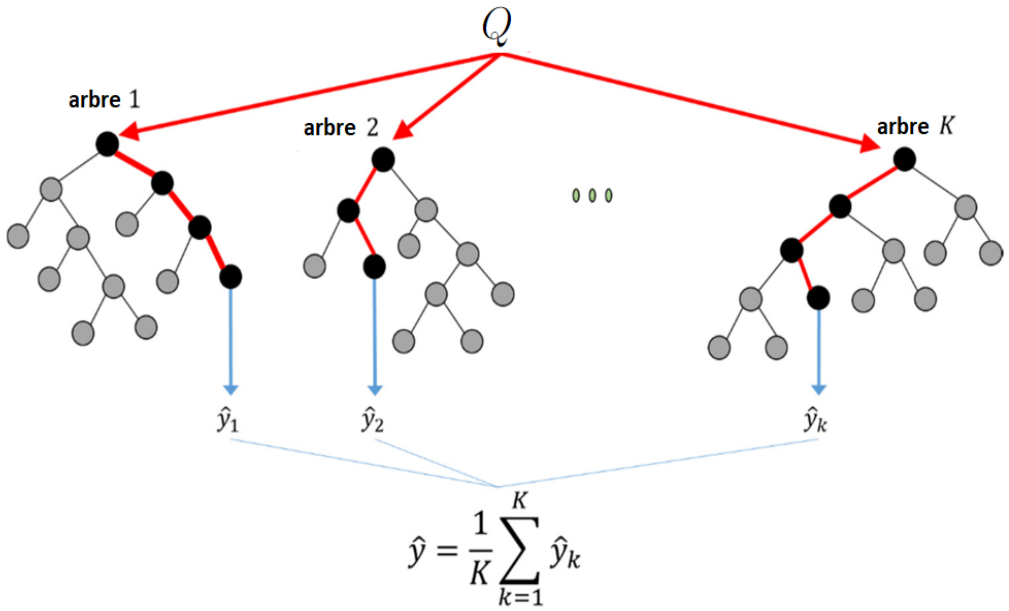

FIGURE 4.1 – Illustration d'une forêt aléatoire d'arbres de régression [10].

4.2 Création d'une forêt aléatoire d'arbres de régression

Afin de créer une forêt aléatoire d'arbres de régression, la technique dite d'"ensachage" ("Bootstrapped aggregating" ou son acronyme "Bagging", en anglais) sera utilisée et permettra de diversifier les arbres de régression de celle-ci. Cette technique consiste à créer une base de données dite "bootstrapped", c'est-à-dire amorcée sur base de la base de données d'entrainement initiale.

4.2.1 Étape 1 - Création d'une base de données amorcée

Une base de données amorcée est utilisée afin de faire intervenir une part d'aléatoire dans l'algorithme de la forêt aléatoire d'arbres de régression. Cette base de données amorcée est conçue à partir de la base de données d'entrainement initiale. Des échantillons issus de la base de données d'entrainement initiale vont être sélectionnés aléatoirement dans le but de créer la base de données amorcée. La technique d'ensachage autorise le fait qu'un même échantillon de la base de données d'entrainement initiale puisse être sélectionné plusieurs fois. Notons cependant qu'une des règles respectée par la technique d'ensachage est de créer une base de données amorcée qui conserve le même nombre d'échantillons que celui de la base de données d'entrainement initiale.

4.2.2 Étape 2 - Création d'un arbre de la forêt aléatoire

Une fois que la base de données amorcée est créée, la deuxième étape de l'algorithme de construction d'une forêt aléatoire d'arbres de régression consiste à concevoir un arbre de régression de la forêt à partir de cette base de données amorcée. Cette étape nécessite l'usage de l'algorithme de construction d'un arbre de régression que nous avons détaillé dans le chapitre précédent du mémoire.

4.2.3 Étape 3 - Répétition des étapes 1 et 2

La dernière étape consiste simplement à répéter les étapes 1 et 2 autant de fois que nous souhaitons ajouter d'arbres de régression au sein de la forêt aléatoire que nous créons.

4.3 Avantages et inconvénients

Les arbres de régression sont particulièrement sensibles au "surajustement". Ce concept définit un manque de flexibilité de ces modèles pour donner de bonnes performances sur des données de test en dépit du fait qu'elles soient performantes sur les données d'entrainement. Des données de test peuvent être qualifiées comme étant une base de données différente de celle d'entrainement, c'est-à-dire de celle à partir de laquelle le modèle a été créé. Une des solutions possibles pour palier à ce problème est de faire appel au modèle de forêt aléatoire d'arbres de régression plutôt qu'au simple modèle d'arbre de régression. En effet, le regroupement des arbres en une forêt permet de diminuer la sensibilité globale du problème au surajustement et ainsi d'améliorer la prédiction finale. Ceci forme un premier avantage de l'utilisation de forêts aléatoires d'arbres de régression. Un second avantage de ce modèle réside dans le fait qu'il soit très efficace pour des bases de données d'entrainement initiales de grande dimension.

Parmi les inconvénients du modèle de forêt aléatoire d'arbres de régression, nous comptons le fait que même si la forêt aléatoire d'arbres de régression est moins sujette au surajustement que l'arbre de régression, elle peut tout de même être touchée par ce phénomène surtout si le nombre d'arbres de régression qu'elle contient est élevé. Un second inconvénient de la forêt aléatoire d'arbres de régression est l'utilisation de la mémoire. En effet, cet algorithme requiert plus de mémoire que les autres car il stocke plusieurs estimateurs. Enfin, un dernier inconvénient est son temps d'entrainement qui est plus long que celui d'un arbre de régression.

Chapitre 5

Les hyperparamètres

Dans ce chapitre, nous décrirons, dans un premier temps, les différents hyperparamètres associés aux modèles d'arbre de régression et de forêt aléatoire d'arbres de régression qui peuvent être réglés et que nous avons étudiés lors de la phase d'expérimentation. Toutes les informations concernant les hyperparamètres sont issues principalement de la documentation sur la version 0.20.4 de ScikitLearn [17] installée sur Zenobe. Dans un second temps, nous décrirons les hyperparamètres que nous n'avons pas testés lors de la phase d'expérimentation et nous expliquerons les raisons qui ont motivé notre sélection.

5.1 Hyperparamètres étudiés lors de la phase d'expérimentation

Les modèles d'arbre de régression et de forêt aléatoire d'arbres de régression que nous avons implémentés dans le langage Python3 font appel aux modules `sklearn.tree` et `sklearn.ensemble` de la librairie ScikitLearn, repectivement. Ces modules disposent de classes nommées DecisionTreeRegressor et RandomForestRegressor, respectivement. Nous allons, dans cette section, décrire tout d'abord les hyperparamètres communs au modèle d'arbre de régression et au modèle de forêt aléatoire d'arbres de régression. Puis, nous expliquerons les hyperparamètres propres à chacun de ces deux modèles.

5.1.1 Hyperparamètres communs au modèle d'arbre de régression et au modèle de forêt aléatoire d'arbres de régression

- **"criterion"** : cet hyperparamètre prend comme valeur une chaine de caractères. Ce critère permet de sélectionner la mesure utilisée pour déterminer la qualité de la division d'un noeud. Il s'agit pour la valeur de cette mesure de déterminer la qualité d'une division, autrement dit, les performances d'un seuil d'une règle de décision dans un noeud donné d'un arbre de régression, en utilisant une métrique calculant l'erreur moyenne de prédiction des échantillons de ce noeud. Parmi les valeurs qui peuvent lui être assignées, il y a "mse" ("mean squared error", en anglais). Une autre valeur possible est "friedman_mse" qui fait référence à l'erreur quadratique moyenne

de Friedman, mais nous ne nous y attarderons pas dans le cadre de ce mémoire. Le calcul effectué par le critère "mse" est identique à celui décrit dans la section 3.3.2 de ce mémoire. Ce critère permet de calculer l'erreur quadratique moyenne de prédiction des échantillons d'un noeud donné dans un arbre de régression. Une autre possibilité est d'utiliser "mae" ("mean absolute error", en anglais) pour calculer l'erreur absolue moyenne de prédiction des échantillons d'un noeud donné dans un arbre de régression. Celle-ci minimise la perte en norme L1 en utilisant la médiane des échantillons dans chaque noeud courant. La valeur par défaut de cet hyperparamètre est "mse".

- **"max_depth"** : il s'agit de la profondeur de l'arbre, autrement dit du nombre de niveaux ou d'étages dans l'arbre. Lorsque la valeur par défaut "None" est utilisée, les noeuds de l'arbre sont étendus jusqu'à ce que toutes les feuilles soient pures ou bien jusqu'à ce qu'elles contiennent toutes moins d'échantillons que la valeur assignée à l'hyperparamètre "min_samples_split". Une feuille est dite "pure" lorsqu'elle ne contient qu'un seul échantillon ou bien plusieurs échantillons ayant la même évalutation.

- **"min_samples_split"** : la valeur assignée à cet hyperparamètre correspond au nombre minimum d'échantillons nécessaires pour qu'un noeud d'un arbre de régression puisse être divisé. Le type de la valeur choisie pour l'hyperparamètre "min_samples_split" peut être soit un entier soit un nombre décimal. Dans ce dernier cas, la valeur effective qui sera utilisée par "min_samples_split" sera retournée par la fonction "ceil(x)" du langage Python3 qui calcule le plus petit entier strictement plus grand que "x". Attention, il est important de noter que ce x sera remplacé ici par la valeur de "min_samples_split" multipliée par le nombre d'échantillons de la base de données d'entrainement. Par exemple, si nous travaillons avec une base de données d'entrainement contenant 8 échantillons et que nous choisissons d'attribuer la valeur 0.2 à l'hyperparamètre "min_samples_split", la valeur effective qui sera utilisée par cet hyperparamètre sera le plus petit entier strictement supérieur à 1.6, c'est-à-dire ici 2. En effet, 2 est bien le plus petit entier strictement plus grand que 1.6. Nous avons maintenant déterminé que $ceil(1.6) = 2$. De plus, notons que la valeur par défaut de cet hyperparamètre est 2.

- **"min_samples_leaf"** : cet hyperparamètre constitue le nombre minimum d'échantillons nécessaires dans un noeud feuille. À n'importe quelle profondeur de l'arbre, une division d'un noeud ne se fera que dans le cas où, suite à cette division, le nombre d'échantillons restants dans les noeuds gauche et droit créés est au moins égal à "min_samples_leaf" échantillons. Cet hyperparamètre peut prendre des valeurs entières ou décimales. Tout comme pour l'hyperparamètre "min_samples_split", la valeur utilisée par "min_samples_leaf" dans le cas d'un nombre décimal sera retournée par la fonction "ceil(x)" où x sera remplacé ici par la valeur de "min_samples_leaf" multipliée par le nombre d'échantillons de la base de données d'entrainement. La valeur par défaut de cet hyperparamètre est 1.

- **"max_features"** : cet hyperparamètre correspond au nombre de prédicteurs qui seront pris en compte lors de la recherche de la meilleure division pour chaque noeud. L'hyperparamètre "max_features" admet des valeurs de tous les types (entier, nombre décimal ou chaine de caractère). La valeur par défaut de cet hyperparamètre est "None" et cette valeur correspond au nombre de variables du problème étudié, autrement dit n.

- **"max_leaf_nodes"** : cet hyperparamètre permet de fixer le nombre maximal de noeuds feuilles dans l'arbre de régression. Lorsque sa valeur est définie à sa valeur par défaut "None", le nombre de noeuds feuilles que l'arbre peut développer est illimité.

5.1.2 Hyperparamètre propre à l'arbre de régression

- **"splitter"** : cet hyperparamètre représente la stratégie de division qui va être utilisée à chaque noeud. Les deux stratégies possibles sont "best" ou "random". La première stratégie choisit la meilleure division et la seconde choisit la meilleure division aléatoire. Si nous imaginons la représentation de l'arbre lorsque la valeur de "splitter" est fixée à "random", nous obtiendrons un arbre qui possède autant de noeuds feuilles que d'échantillons tandis qu'avec la stratégie "best", les doublons de la base de données d'échantillons seront regroupés dans le même noeud feuille. La valeur par défaut de cet hyperparamètre est la stratégie "best".

5.1.3 Hyperparamètre propre à la forêt aléatoire d'arbres de régression

- **"n_estimators"** : il s'agit simplement du nombre d'arbres de régression qui composent la forêt aléatoire. Pour rappel, un arbre peut aussi être défini comme un estimateur. Seuls des valeurs entières peuvent être assignées à cet hyperparamètre. Sa valeur par défaut dans la version 0.20.4 de ScikitLearn est 10 mais celle-ci a été mise à jour et vaut 100 dans les versions ultérieures.

5.2 Hyperparamètres non étudiés lors de la phase d'expérimentation

Avant de commencer nos expérimentations, nous avons sélectionné les hyperparamètres que nous voulions tester sur Zenobe pour l'arbre de régression et pour la forêt aléatoire d'arbres de régression. Dans cette section, nous allons donc brièvement parler des hyperparamètres que nous n'avons pas mentionnés précédemment.

5.2.1 Hyperparamètres communs au modèle d'arbre de régression et au modèle de forêt aléatoire d'arbres de régression

- **"min_weight_fraction_leaf"** : pour travailler avec cet hyperparamètre, nous devons considérer que les échantillons de la base de données d'entrainement ont chacun un poids qui leur est associé. En d'autres termes, cet hyperparamètre considère le cas où les échantillons d'entrainement n'ont pas tous la même importance et certains ont plus d'impact lors de la construction de l'arbre. Dans ce cadre, min_weight_fraction_leaf correspond à la part de la somme totale de tous les poids des échantillons de la base de données d'entrainement requise pour être à un noeud feuille. Cet hyperparamètre prend comme valeur un nombre décimal et sa valeur par défaut est 0.0. Dans notre contexte d'étude, cet hyperparamètre ne nous intéresse pas car les DOE que nous considérons comme base de données d'entrainement disposent d'un poids égal pour chacun des individus les constituant. Dès lors, modifier la valeur de cet hyperparamètre n'avait aucun sens.

- **"random_state"** : il s'agit de la graine utilisée pour générer un nombre aléatoire au sein de l'algorithme de construction d'un arbre de régression. Par exemple, si, lors du développement d'un arbre, deux seuils candidats à la règle de décision du noeud considéré admettent la même somme totale des erreurs de prédiction au carré, alors ce nombre aléatoire permettra de choisir le seuil qui sera sélectionné. Afin d'obtenir un comportement déterministe lors de l'élaboration de l'arbre, la valeur de cet hyperparamètre doit être fixée à une valeur entière. Assigner la valeur par défaut "None" à cet hyperparamètre permet d'empêcher un comportement déterministe. Nous n'avons pas modifié la valeur par défaut de l'hyperparamètre "random_state" car nous souhaitions conserver le caractère aléatoire du modèle d'arbre de régression.

- **"min_impurity_split"** : il s'agit d'un seuil utilisé comme référence pour décider si un noeud peut être divisé ou non, au cours de l'algorithme de construction de l'arbre. L'hyperparamètre "min_impurity_split" est obsolète depuis la version 0.19 de ScikitLearn. Il a été déprécié en faveur de l'hyperparamètre "min_impurity_decrease" et c'est la raison pour laquelle nous avons fait le choix de ne pas le tester durant nos expérimentations.

- **"min_impurity_decrease"** : les valeurs pouvant être assignées à cet hyperparamètre sont des valeurs décimales. Un noeud de l'arbre de régression sera divisé dans le cas où la diminution de l'impureté ("mse" ou "mae") du noeud engendrée par cette division est plus grande ou égale à la valeur attribuée à cet hyperparamètre. Nous n'avons pas testé différents réglages de cet hyperparamètre par manque de temps.

5.2.2 Hyperparamètre propre à l'arbre de régression

- **"presort"** : cet hyperparamètre correspond à une variable booléenne qui permet de trier ou non la base de données d'entrainement au préalable afin de rendre la

recherche des meilleures divisions plus rapide lors de l'élaboration de l'arbre. Pour des bases de données d'entrainement contenant beaucoup d'échantillons, utiliser cet hyperparamètre peut potentiellement augmenter le temps d'entrainement de l'algorithme de construction de l'arbre. L'hyperparamètre "presort" a été retiré des versions ultérieures de ScikitLearn. Cette information a davantage motivé notre choix d'enlever cet hyperparamètre de notre sélection pour la phase de tests.

5.2.3 Hyperparamètres propres à la forêt aléatoire d'arbres de régression

- **"bootstrap"** : l'hyperparamètre "bootstrap" joue sur l'aspect aléatoire et la diversité des arbres de la forêt aléatoire. Nous n'avions dès lors trouvé aucun intérêt à changer sa valeur. Sa valeur par défaut est "True" et elle permet à la technique d'ensachage décrite dans le chapitre précédent d'être utilisée.

- **"oob_score"** : cet hyperparamètre désigne une variable booléenne et sa valeur par défaut est "False". La valeur assignée à cet hyperparamètre permet de savoir si les échantillons qui n'ont pas été pris en compte par la technique d'ensachage lors de l'élaboration de la base de données d'entrainement de chaque arbre de la forêt doivent être utilisés pour calculer le score global de l'arbre ou non. Une fonction proposée par ScikitLearn permet de calculer le score global de l'arbre et se nomme `score()`. Cette fonction calcule la moyenne de la précision des prédictions de l'arbre pour chacun des échantillons de la base de données d'entrainement de cet arbre. Tester cet hyperparamètre nous semblait avoir moins d'intérêt dans le cadre de ce mémoire.

- **"n_jobs"** : ce nombre correspond au nombre de tâches à effectuer en parallèle. Nous ne souhaitions pas travailler sur cet aspect dans le cadre de ce mémoire.

- **"verbose"** : cet hyperparamètre nous permet de contrôler ce qu'il se passe lors de la phase d'entrainement du modèle de prédiction. Il ne nous a pas semblé pertinent de jouer sur les valeurs de cet hyperparamètres afin d'atteindre nos objectifs.

- **"warm_start"** : lorsqu'il est défini à "True", cet hyperparamètre permet de réutiliser la solution de l'appel précédent pour ajuster et ajouter plus d'estimateurs à la forêt aléatoire. Sinon, lorsque "warm_start" est assigné à sa valeur par défaut "False", une toute nouvelle forêt est entrainée. Dans le cadre de nos expérimentations, nous souhaitions créer une toute nouvelle forêt aléatoire à chacune des utilisations de ce modèle, c'est pourquoi nous n'avons pas manipulé les valeurs de cet hyperparamètre.

Chapitre 6

Présentation des problèmes tests étudiés

Ce chapitre présente les différents problèmes tests étudiés lors de la phase d'expérimentation. Nous avons subdivisé ce chapitre en deux parties. La première est consacrée à la description des dix-sept problèmes tests qui représentent des problèmes d'optimisation sans contraintes. La seconde partie regroupe les sept problèmes tests qui constituent des problèmes d'optimisation avec contraintes. L'ensemble de ces problèmes tests est constitué de fonctions de référence issues de la littérature. Ces problèmes de référence peuvent être trouvés dans [20], [21], [22], [23], [24], [25], [26] pour les problèmes sans contraintes et dans [27], [28], [29], [30], pour les problèmes avec contraintes. Nous les décrirons sur base de leur définition ainsi que de la valeur de leur solution globale. De plus, des détails seront apportés quant aux caractéristiques de leur fonction objectif. Nous précisons ici que ces descriptions ont été reprises de la littérature. La nature des variables considérées par Minamo, pour chaque problème test, est indiquée grâce aux initiales R, I, D, C qui sont accolées aux noms de la plupart des problèmes test. Les problèmes tests dont le nom n'est pas suivi d'une de ces lettres est un problème qui ne dispose que de variables continues. Pour rappel, la signification de chacune des lettres R, I, D, C a été expliquée dans la section 2.2.4 du mémoire.

6.1 Problèmes tests sans contraintes

Ackley_10

L'expression analytique de la fonction Ackley de dimension n est définie par :

$$f(x) = -20\ exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=1}^{n} x_i^2}\right) - exp\left(\frac{1}{n}\sum_{i=1}^{n} cos(2\pi x_i)\right) + 20 + e.$$

Le domaine de cette fonction est $[-3, 2]^n$ où n est le nombre de variables. L'optimum global de cette fonction correspond à $x^* = (0, \ldots, 0)$ et $f(x^*) = 0$. Le problème test Ack-

ley_10 considère la fonction objectif comme étant la fonction Ackley de dimension dix. Cette fonction non convexe.

Rosenbrock_10

Le domaine de la fonction Rosenbrock est $x_i \in [-2, 2],\ \forall i = 1, \ldots, n$, où n correspond au nombre de composantes de x. La définition de cette fonction est donnée par :

$$f(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2].$$

Pour le problème test Rosenbrock_10, la fonction objectif à minimiser sera la fonction de Rosenbrock de dimension $n = 10$. Pour cette dimension, la fonction est multimodale et admet un optimum global en $x^* = (1, \ldots, 1)$. L'évaluation de f en ce x^* vaut 0. De plus, il s'agit d'une fonction non convexe.

Borehole-RC1/2/3

L'expression de la fonction objectif à sept variables associée au problème test Borehole est donnée par :

$$f(x) = \frac{2\pi x_1 x_5}{\ln(x_6 x_2)[1 + \frac{2x_3 x_1}{\ln(x_6/x_2)x_2^2 x_7} + \frac{x_1}{x_4}]}.$$

Cette fonction à minimiser modélise le débit d'eau qui s'écoule au travers d'un trou de forage (en m^3/s), où :

- x_1 représente la valeur de la perméabilité de la nappe aquifère supérieure (en m^2/s),
- x_2 mesure le rayon du trou de forage (en m),
- x_3 représente la longueur du trou de forage (en m),
- x_4 représente la valeur de la perméabilité de la nappe aquifère inférieure (m^2/s),
- x_5 donne une valeur de la différence entre le niveau de pression de la nappe aquifère supérieure et celui de la nappe aquifère inférieure (en m),
- x_6 représente la valeur du rayon d'action (en m),
- x_7 représente la conductivité hydraulique du trou de forage (en m/s).

En ce qui concerne le domaine de la fonction Borehole, nous avons $x_1 \in [0.00199, 0.00366]$, $x_2 \in [1.58 \times 10^{-9}, 4.75 \times 10^{-9}]$, $x_3 \in [0.000035, 0.000053]$ et $x_4 \in [0.0000019, 0.0000036]$. Les variables x_5, x_6 et x_7, quant à elles, ont pour valeurs des points équidistants pris dans les intervalles $[0.0000092, 0.000012]$, $[0.0000031, 0.0015]$ et $[0.00031, 0.00038]$, respectivement.

Afin de connaitre le nombre de points équidistants à sélectionner dans chacun de ces intervalles, trois situations différentes sont considérées dans la littérature pour chacune des variables x_5, x_6 et x_7. La TABLE 6.1 reprend, pour chacune de ces trois variables et en fonction de chacune des trois situations, le nombre de points équidistants à sélectionner dans l'intervalle $[0.0000092, 0.000012]$ pour la variable x_5, dans l'intervalle $[0.0000031, 0.0015]$ pour la variable x_6 et dans l'intervalle $[0.00031, 0.00038]$ pour la variable x_7.

Situation	x_5	x_6	x_7
1	3	3	3
2	2	2	10
3	2	2	15

TABLE 6.1 – Table reprenant, pour chacune des trois situations considérées, le nombre de points équidistants à sélectionner dans les intervalles définis pour x_5, x_6 et x_7, respectivement.

Par exemple, pour la première situation, nous devons sélectionner trois points équidistants au sein de l'intervalle $[0.0000092, 0.000012]$ pour la variable x_5. Ceci revient à dire que les valeurs que peut prendre la variable x_5 sont les points équidistants 0.0000092, 0.000106 et 0.000012 contenus dans l'intervalle $[0.0000092, 0.000012]$. Pour ce problème test, nous ne connaissons ni la valeur d'un optimum global ni la valeur de la meilleur solution approchée qu'il est possible d'obtenir. Lors de la phase d'expérimentations du mémoire, nous avons considéré les trois situations possibles pour le problème test Borehole. Nous les avons distinguées en les nommant Borehole-1, Borehole-2 et Borehole-3.

Branin-RC1

La fonction Branin-1 possède quatre variables. L'expression analytique de la fonction objectif de ce problème de minimisation est la suivante :

$$f(x) = \begin{cases} h(x_1, x_2) & \text{si } x_3 = 0 \text{ et } x_4 = 0 \\ 0.4h(x_1, x_2) + 1.1 & \text{si } x_3 = 0 \text{ et } x_4 = 1 \\ -0.75h(x_1, x_2) + 5.2 & \text{si } x_3 = 1 \text{ et } x_4 = 0 \\ -0.5h(x_1, x_2) - 2.1 & \text{si } x_3 = 1 \text{ et } x_4 = 1 \end{cases},$$

où :

$$h(x_1, x_2) = \frac{(15x_2 - \frac{5}{4\pi^2}(15x_1 - 5)^2 + \frac{5}{\pi}(15x_1 - 5) - 6)^2 + 10(1 - \frac{1}{8\pi})\cos(15x_1 - 5) + 10 - 54.8104}{51.9496}.$$

Les variables x_1 et x_2 sont continues et prennent leurs valeurs dans l'intervalle $[0, 1]$, tandis que les valeurs des variables discrètes x_3 et x_4 appartiennent à l'ensemble {0,1}. Pour ce problème test, nous ne disposons ni de la valeur d'un optimum global ni de la valeur de la meilleure solution approchée qu'il soit possible d'obtenir.

DSMM-RC

Le problème test DSMM est un problème bidimensionnel avec une variable continue x_1 et une variable catégorielle x_2 disposant de 10 attributs. La fonction objectif du problème de minimisation DSMM est définie par :

$$f(x) = f(x_1, x_2) = \begin{cases} \cos(3.6\pi(x_1 - 2)) + x_1 - 1 & \text{si} \quad x_2 = \text{"1"}, \\ 2\cos(1.1\pi \exp(x_1)) - \frac{x_1}{2} + 2 & \text{si} \quad x_2 = \text{"2"}, \\ \cos(2\pi x_1) + \frac{1}{2}x_1 & \text{si} \quad x_2 = \text{"3"}, \\ x(\cos(3.4\pi(x_1 - 1)) - \frac{x_1 - 1}{2}) & \text{si} \quad x_2 = \text{"4"}, \\ -\frac{x_1^2}{2} & \text{si} \quad x_2 = \text{"5"}, \\ 2\cos(\frac{\pi}{4}\exp(-x_1^4))^2 - \frac{x_1}{2} + 1 & \text{si} \quad x_2 = \text{"6"}, \\ x_1\cos(3.4\pi x_1) - \frac{x_1}{2} + 1 & \text{si} \quad x_2 = \text{"7"}, \\ x_1(-\cos(7\frac{\pi}{2}x_1) - \frac{x_1}{2}) + 2 & \text{si} \quad x_2 = \text{"8"}, \\ -\frac{x_1^5}{2} + 1 & \text{si} \quad x_2 = \text{"9"}, \\ -\cos(5\frac{\pi}{2}x_1)^2\sqrt{x_1} - \frac{\ln(x_1+0.5)}{2} - 1.3 & \text{si} \quad x_2 = \text{"10"} \end{cases}.$$

La fonction DSMM admet un minimum global en $x^* = (0.808, \text{"10"})$ et $f(x^*) = -2.329$.

FCATMIX-RC1

Le problème test FCATMIX-1 dispose de quatre variables. Les variables x_1 et x_2 sont des variables continues tandis que les variables x_3 et x_4 sont catégorielles. Afin de définir ce problème test, nous introduisons au préalable une fonction auxiliaire notée f_{aux1}. Cette fonction prend en argument les deux variables continues x_1 et x_2. Cette fonction se définit comme :

$$f_{aux1}(x_1, x_2) = 0.5x_1^2x_2^2 - 0.5x_1^3 + 0.2x_2^3 + 0.17x_1^2 - x_2^2 - 0.5x_1x_2 + 0.5x_1 + 0.3x_2 - 0.45,$$

pour $(x_1, x_2) \in [0, 1]^2$ et $(x_3, x_4) \in \{a, b\} \times \{d, e, f\}$. Notons que dans l'expression de la fonction objectif du problème test FCATMIX-1, nous allons retrouver l'expression de la fonction Branin échelonnée f_{b2} que nous avons définie précédemment. La fonction $f_{catmix1}$ associée au problème test FCATMIX-1 peut être formulée de la manière suivante :

$$f_{catmix1}(x) = \begin{cases} f_{aux1}(x_1, x_2), & \text{pour } (x_3, x_4) = (a, d) \\ 0.9f_{aux1}(x_1, x_2), & \text{pour } (x_3, x_4) = (a, e) \\ 1.05f_{aux1}(x_1, x_2), & \text{pour } (x_3, x_4) = (a, f) \\ 0.5x_1^3 - 0.25x_1^2 - 0.025x_2 - 0.86 + 0.04, & \text{pour } (x_3, x_4) = (b, d) \\ f_{b2}(x_1, x_2), & \text{pour } (x_3, x_4) = (b, e) \\ 0.5x_1^2x_2^2 - 0.5x_1^3 + 0.2x_2^3 + 0.17x_1^2 - x_2^2 - 0.5x_1x_2 & \\ +0.5x_1 + 0.3x_2 - 0.42, & \text{pour } (x_3, x_4) = (b, f) \end{cases}.$$

Le problème test FCATMIX-1 admet trois optima globaux : un premier en $(0.9616520, 0.15, b, e)$,

un second en $(0.1238946, 0.8166644, b, e)$ et un dernier optimum en $x^* = (0.5427730, 0.15, b, e)$. L'évaluation de ces points par $f_{catmix1}$ vaut -1.04741.

FCATMIX-RC2

Le problème test FCATMIX-2 dispose de huit variables au total dont cinq variables continues et trois variables catégorielles. Les cinq variables continues $x_1, \ldots, x_5$ prennent chacune leurs valeurs dans l'intervalle $[0, 1]$. Les variables catégorielles x_6, x_7 et x_8 possèdent 3, 2 et 2 attributs, respectivement. Afin de définir ce problème test, nous introduisons au préalable une fonction auxiliaire notée f_{aux2}. Cette fonction prend en argument les cinq variables continues x_1 à x_5. Cette fonction se définit comme :

$$f_{aux2}(x_1, x_2, x_3, x_4, x_5) = \sum_{i=1}^{5} (5x_i + (1 - x_i))^2 2^{i-1} - 2.75,$$

pour $(x_1, \ldots, x_5) \in [0, 1]^5$. Les valeurs de x_6, x_7 et x_8 sont telles $(x_6, x_7, x_8) \in \{a, b, c\} \times \{d, e\} \times \{f, g\}$. Définissons maintenant la fonction $f_{catmix2}$ associée au problème test FCATMIX-2 par

$$f_{catmix2}(x) = \begin{cases} f_{aux2}(x_1, x_2, x_3, x_4, x_5), & \text{pour } (x_6, x_7, x_8) = (a, d, f) \\ f_{aux2}(x_1, x_2, x_3, x_4, x_5) + 0.2, & \text{pour } (x_6, x_7, x_8) = (a, d, g) \\ 0.9 f_{aux2}(x_1, x_2, x_3, x_4, x_5), & \text{pour } (x_6, x_7, x_8) = (a, e, f) \\ f_{aux2}(x_1, x_2, x_3, x_4, x_5) + 0.25, & \text{pour } (x_6, x_7, x_8) = (a, e, g) \\ f_{aux2}(x_1, x_2, x_3, x_4, x_5) + 0.5, & \text{pour } (x_6, x_7, x_8) = (b, d, f) \\ 0.8 f_{aux2}(x_1, x_2, x_3, x_4, x_5), & \text{pour } (x_6, x_7, x_8) = (b, d, g) \\ 0.5 f_{aux2}(x_1, x_2, x_3, x_4, x_5), & \text{pour } (x_6, x_7, x_8) = (b, e, f) \\ f_{aux2}(x_1, x_2, x_3, x_4, x_5) + 0.8, & \text{pour } (x_6, x_7, x_8) = (b, e, g) \\ f_{aux2}(x_1, x_2, x_3, x_4, x_5) + 0.9, & \text{pour } (x_6, x_7, x_8) = (c, d, f) \\ 0.5 f_{aux2}(x_1, x_2, x_3, x_4, x_5), & \text{pour } (x_6, x_7, x_8) = (c, d, g) \\ f_{aux2}(x_1, x_2, x_3, x_4, x_5) + 1, & \text{pour } (x_6, x_7, x_8) = (c, e, f) \\ f_{aux2}(x_1, x_2, x_3, x_4, x_5) + 1.25, & \text{pour } (x_6, x_7, x_8) = (c, e, g) \end{cases}.$$

Le problème test FCATMIX-2 atteint son optimum global en $x^* = (0.5, 0.5, 0.5, 0.5, 0.5, a, d, f)$, avec $f_{catmix2}(x^*) = -2.75$.

FMIX-RC

Le problème test FMIX prend en compte cinq variables. Les variables x_i pour $i = 1, \ldots, 4$ sont des variables continues et prennent chacune leurs valeurs dans l'intervalle $[0, 1]$. La variable x_5 est une variable catégorielle et ses attributs sont des éléments de l'ensemble $A = \{a, b, c\}$. Notons que dans l'expression de la fonction objectif du problème test FMIX, nous allons retrouver l'expression de la fonction Branin échelonnée f_{b2} que nous avons définie précédemment. La fonction f_{mix1} associée au problème test FMIX a pour formulation l'expression suivante :

$$f_{mix}(x) = \begin{cases} 20x_3x_4^2 + f_{b2}(x_1, x_2), & \text{pour } x_5 = a \\ f_{b2}(x_1, x_2), & \text{pour } x_5 = b \\ f_{b2}(x_1, x_2), & \text{pour } x_5 = c \end{cases}.$$

Pour ce problème test, nous ne connaissons ni la valeur d'un optimum global ni la valeur de la meilleur solution approchée qu'il soit possible d'atteindre.

FPOLY-RC

Le problème test FPOLY ne compte que deux variables. La variable x_1 est une variable continue et elle est à valeurs dans $[0, 1]$. La variable x_2 est une variable catégorielle qui ne dispose que de deux attributs. Nous pouvons dès lors définir le domaine de définition de la fonction objectif f_{poly} comme étant l'ensemble des couples (x_1, x_2) tels que $x_1 \in [0, 1]$ et $x_2 \in \{a, b\}$. Nous définissons la fonction objectif f_{poly} comme :

$$f_{poly}(x) = f_{poly}(x_1, x_2) = \begin{cases} (x_1 - 0.3)^2(x_1 + 2)(x_1 + 4)(x_1 + 0.1), & \text{pour } x_2 = a \\ (x_1 + 0.2)^2(x_1 - 1.1)^2, & \text{pour } x_2 = b \end{cases}.$$

Le problème test FPOLY atteint son optimum global en $x^* = (0.3, a)$ auquel $f_{poly}(x^*) = 0$.

FQUAD-RC

Le problème test FQUAD ne dispose également que de deux variables. La variable x_1 est une variable continue et elle est à valeurs dans $[0, 1]$. La variable x_2 est une variable catégorielle qui ne dispose que de deux attributs. Nous pouvons dès lors définir le domaine de définition de la fonction objectif f_{quad} comme étant identique à celui du problème test f_{poly} décrit précédemment. La définition de la fonction objectif f_{quad} est la suivante :

$$f_{quad}(x) = \begin{cases} 6(x_1 - 0.5)^2 - 0.5, & \text{pour } x_2 = a \\ -6(x_1 - 0.5)^2 + 0.5, & \text{pour } x_2 = b \end{cases}.$$

La fonction objectif du problème test FQUAD atteint son minimum global en $x^* = (1, b)$ auquel $f_{quad}(x^*) = -1$.

FTRIG-RC

Pour le problème test FTRIG, nous travaillerons aussi avec deux variables : une variable continue x_1 et une variable x_2 catégorielle. La variable x_1 est à valeurs dans $[0, 1]$ et la variable x_2 ne possède que deux attributs : a et b. Le domaine de définition de la fonction objectif f_{trig} est donc exactement le même que celui du problèmes tests f_{poly} et f_{quad} décrit précédemment. La définition de la fonction objectif f_{trig} est la suivante :

$$f_{trig}(x) = f_{trig}(x_1, x_2) = \begin{cases} \sin(6(x_1^2 - \frac{1}{4})) + 1, & \text{pour } x_2 = a \\ \sin(x_1)\tan(x_1) + 0.1, & \text{pour } x_2 = b \end{cases}.$$

La fonction objectif du problème test FTRIG atteint son minimum global en $x^* = (0.9321, a)$ auquel $f_{trig}(x^*) = 0$.

Goldstein-RC

Le problème test Goldstein traite quatre variables : deux variables continues x_1 et x_2 et deux variables discrètes x_3 et x_4. Comme pour le problème test Borehole, les valeurs que peuvent prendre les variables discrètes x_3 et x_4 dépendent de la situation 1, 2 ou 3 dans laquelle nous nous trouvons. Afin de savoir dans laquelle de ces trois situations nous nous trouvons pour chacune des variables x_3 et x_4, nous allons définir deux nouvelles variables z_1 et z_2. Ces variables z_1 et z_2 ont des valeurs qui appartiennent à l'ensemble $\{1, 2, 3\}$. La TABLE 6.2 caractérise les valeurs prises par les variables discrètes x_3 et x_4 selon les valeurs attribuées aux variables z_1 et z_2 de la fonction Goldstein, notée $f(x_1, x_2, z_1, z_2)$. Les valeurs de x_3 et x_4 vont donc dépendre des valeurs qui auront été attribuées au préalable aux variables z_1 et z_2.

	$z_1 = 1$	$z_1 = 2$	$z_1 = 3$
$z_2 = 1$	$x_3 = 20$, $x_4 = 20$	$x_3 = 50$, $x_4 = 20$	$x_3 = 80$, $x_4 = 20$
$z_2 = 2$	$x_3 = 20$, $x_4 = 50$	$x_3 = 50$, $x_4 = 50$	$x_3 = 80$, $x_4 = 50$
$z_2 = 3$	$x_3 = 20$, $x_4 = 80$	$x_3 = 50$, $x_4 = 80$	$x_3 = 80$, $x_4 = 80$

TABLE 6.2 – Caractérisation des valeurs des variables discrètes x_3 et x_4 en fonction des valeurs attribuées à z_1 et z_2.

La fonction Goldstein $f(x_1, x_2, z_1, z_2)$ se définit de la manière suivante :

$$f(x_1, x_2, z_1, z_2) = h(x_1, x_2, x_3, x_4),$$

où $x_1 \in [0, 100]$, $x_2 \in [0, 100]$, $z_1 \in \{1, 2, 3\}$, $z_2 \in \{1, 2, 3\}$ et où l'expression analytique de la fonction $h(x_1, x_2, x_3, x_4)$ est donnée par :

$$\begin{aligned} h(x_1, x_2, x_3, x_4) =& 53.3108 + 0.184901x_1 - 5.02914x_1^3 10^{-6} + 7.72522x_1^4 10^{-8} - \\ & 0.0870775x_2 - 0106959x_3 + 7.98772x_3^3 10^{-6} + \\ & 0.00242482x_4 + 1.32851x_4^3 10^{-6} - 0.00146393x_1x_2 - \\ & 0.00301588x_1x_3 - 0.00272291x_1x_4 + 0.0017004x_2x_3 + \\ & 0.0038428x_2x_4 - 0.000198969x_3x_4 + 1.86025x_1x_2x_3 10^{-5} - \\ & 1.88719x_1x_2x_4 10^{-6} + 2.50923x_1x_3x_4 10^{-5} - \\ & 5.62199x_2x_3x_4 10^{-5} \end{aligned}$$

Dans la littérature, aucune information n'est disponible quant à la valeur d'un optimum global pour ce problème test ou encore par rapport à une éventuelle meilleur solution atteinte.

OTL-RC

Le problème test OTL est un problème à six variables. Ce problème traduit la tension moyenne d'un circuit électrique sans transformateur. Les variables qui interviennent dans ce problème sont ici décrites :

- R_{b1} représente la résistance b1. Cette variable se mesure en kOhms et ses valeurs appartiennent à l'intervalle $[50, 150]$.
- R_{b2} représente la résistance b2. Cette variable se mesure en kOhms et ses valeurs appartiennent à l'intervalle $[25, 70]$.
- R_f est la résistance f. Cette variable se mesure en kOhms et ses valeurs appartiennent à l'intervalle $[0.5, 3]$.
- R_{c1} est la résistance c1. Cette variable se mesure en kOhms et ses valeurs appartiennent à l'intervalle $[1.2, 2.5]$.
- R_{c2} est la résistance c2. Cette variable se mesure en kOhms et ses valeurs appartiennent à l'intervalle $[0.25, 1.2]$.
- β représente le gain de courant et est mesuré en ampère. Son intervalle de valeurs correspond à $[50, 300]$.

La tension moyenne du circuit électrique, notée V_m, que nous cherchons à minimiser se traduit par l'expression ci-dessous :

$$V_m = V_m(R_{b1}, R_{b2}, R_f, R_{c1}, R_{c2}, \beta) = \frac{(V_{b1} + 0.74)\beta(R_{c2} + 9)}{\beta(R_{c2} + 9) + R_f} + \frac{11.35R_f}{\beta(R_{c2} + 9) + R_f} + \frac{0.74R_f\beta(R_{c2} + 9)}{(\beta(R_{c2} + 9) + R_f)R_{c1}},$$

où $V_{b1} = \frac{12R_{b2}}{R_{b1}+R_{b2}}$. Dans la littérature, aucune information n'est disponible quant à la valeur d'un optimum global pour ce problème test ou encore par rapport à une éventuelle meilleure solution atteinte.

Piston-RC

Le problème Piston-RC représente la modélisation du temps de cycle d'un piston qui se déplace dans un cylindre. L'expression de la fonction objectif associée à ce problème est la suivante :

$$f(M, S, V_0, k, P_0, T_a, T_0) = 2\pi\sqrt{\frac{M}{k + S^2\frac{P_0V_0T_a}{V^2T_0}}},$$

où $V = \frac{S}{2k}\sqrt{A^2 + 4k\frac{P_0}{T_0}T}$ et $A = P_0S + 19.62M - \frac{kV_0}{S}$. Les variables M, S, V_0, T_a, T_0 sont continues et sont telles que $M \in [30, 60]$, $S \in [0.005, 0.020]$, $V_0 \in [0.002, 0.010]$,

$T_a \in [290, 296]$ et $T_0 \in [340, 360]$. Les variables P_0 et k sont des variables catégorielles. Plus d'informations sur les variables de ce problème test peuvent être trouvées dans [32]. Dans la littérature, aucune information n'est disponible quant à la valeur d'un optimum global pour ce problème test ou encore par rapport à une éventuelle meilleure solution atteinte.

Roustant-RC2

Le problème test Roustant compte deux variables et dispose de la fonction objectif définie par :

$$f(x_1, x_2) = \begin{cases} (x_1 + 0.01(x_1 - 1/2)^2) \times x_2/10, & \text{si } x_2 = 1, 2, 3, 4, \\ 0.9\cos(2\pi(x_1 + (x_2 - 4)/20)) \times \exp(-x), & \text{si } x_2 = 5, 6, 7, \\ -0.7\cos(2\pi(x_1 + (x_2 - 7)/20)) \times \exp(-x), & \text{si } x_2 = 8, 9, 10, \end{cases}$$

où $x_1 \in [0, 1$ et $x_2 \in \{1, ..., 10\}$. Dans la littérature, aucune information n'est disponible quant à la valeur d'un optimum global pour ce problème test ou encore par rapport à une éventuelle meilleure solution atteinte.

6.2 Problèmes tests avec contraintes

G7

Le cas-test G7 utilise comme fonction objectif un fonction à dix variables continues. Elle se définit comme suit :

$$\begin{aligned} f(x) =& x_1^2 + x_2^2 + x_1x_2 - 14x_1 - 16x_2 + (x_3 - 10)^2 + 4(x_4 - 5)^2 + (x_5 - 3)^2 \\ &+ 2(x_6 - 1)^2 + 5x_7^2 + 7(x_8 - 11)^2 + 2(x_9 - 10)^2 + (x_{10} - 7)^2 + 45, \end{aligned}$$

avec $x_i \in [-10, 10]$, $\forall i = 1, \ldots, 10$. De plus, cette fonction est soumise aux huit contraintes suivantes :

$$\begin{aligned} c_1(x) &= -105 + 4x_1 + 5x_2 - 3x_7 + 9x_8 && \leq 0 \\ c_2(x) &= 10x_1 - 8x_2 - 17x_7 + 2x_8 && \leq 0 \\ c_3(x) &= -8x_1 + 2x_2 + 5x_9 - 2x_{10} - 12 && \leq 0 \\ c_4(x) &= 3(x_1 - 2)^2 + 4(x_2 - 3)^2 + 2x_3^2 - 7x_4 - 120 && \leq 0 \\ c_5(x) &= 5x_1^2 + 8x_2 + (x_3 - 6)^2 - 2x_4 - 40 && \leq 0 \\ c_6(x) &= x_1^2 + 2(x_2 - 2)^2 - 2x_1x_2 + 14x_5 - 6x_6 && \leq 0 \\ c_7(x) &= 0.5(x_1 - 8)^2 + 2(x_2 - 4)^2 + 3x_5^2 - x_6 - 30 && \leq 0 \\ c_8(x) &= -3x_1 + 6x_2 + 12(x_9 - 8)^2 - 7x_{10} && \leq 0 \end{aligned} .$$

Pour ce problème test, l'optimum global est défini par la meilleure solution connue. Nous la représenterons par x_{best} et son évaluation par f vaut $f(x_{best}) = 24.30620$.

G10

Le second problème avec contraintes est G10. Bien qu'il soit de dimension huit, l'expression de la fonction objectif à minimiser pour ce problème test ne reprend que trois variables. En effet, cette fonction s'exprime de la manière suivante :

$$f(x) = f(x_1, \ldots, x_8) = x_1 + x_2 + x_3.$$

Le domaine de cette fonction est tel que $x_1 \in [100, 10000]$, x_2 et $x_3 \in [1000, 10000]$. La fonction f est sujette à six contraintes qui sont les suivantes :

$$\begin{array}{ll} c_1(x) = -1 + 0.0025(x_4 + x_6) & \leq 0 \\ c_2(x) = -1 + 0.0025(x_5 + x_7 - x_4) & \leq 0 \\ c_3(x) = -1 + 0.01(x_8 - x_5) & \leq 0 \\ c_4(x) = -x_1x_6 + 833.33252x_4 + 100x_1 - 83333.333 & \leq 0 \\ c_5(x) = -x_2x_7 + 1250x_5 + x_2x_4 - 1250x_4 & \leq 0 \\ c_6(x) = -x_3x_8 + 1250000 + x_3x_5 - 2500x_5 & \leq 0 \end{array} .$$

Ces contraintes sont définies pour $x_i \in [10, 1000], \forall i = 4, \ldots, 8$. La meilleure solution connue est donnée par :$x_{best} = (579.30668, 1359.97067, 5109.97065, 182.01769, 295.60117, 217.98230, 286.41652, 395.60117)$. Nous définirons celle-ci comme notre optimum global et sa valeur est donnée par $f(x_{best}) = 7049.24802$.

PressureVessel-RIDC

Ce problème test est issu d'un problème industriel et dispose de quatre variables. Il représente un contenant destiné à retenir un gaz ou un liquide possédant une pression interne non équivalente à celle de l'atmosphère externe. Parmi les paramètres de ce problème, nous retrouvons par exemple la température et la pression. La fonction objectif associée à ce problème est définie par :

$$f(x) = 0.6224x_1x_3x_4 + 1.7781x_2x_3^2 + 3.1661x_1^2x_4 + 19.84x_1^2x_3,$$

où $x_1, x_2 \in [0, 1]$, $x_3 \in [0, 50]$ et $x_4 \in [0, 240]$. Les contraintes auxquelles cette fonction est soumise sont :

$$\begin{array}{ll} c_1(x) = -x_1 + 0.0193x_3 & \leq 0 \\ c_2(x) = -x_2 + 0.00954x_3 & \leq 0 \\ c_3(x) = \text{plog}(-\pi x_3^2 x_4 - \frac{4}{3}\pi x_3^3 + 1296000) & \leq 0 \end{array},$$

où la fonction plog a été introduite par Regis et Shoemaker [31] et est définie par :

$$\text{plog}(x) = f_{trig}(x_1, x_2) = \begin{cases} \log(1 + x) & \text{si } x \geq 0 \\ -\log(1 - x), & \text{si } x < 0 \end{cases}.$$

A notre connaissance, la meilleure solution est donnée par $x_{best} = (0.8125, 0.4375, 42.098446, 176.636596)$ et $f(x_{best}) = 6059.71435$.

G9-RIDC

L'expression analytique de la fonction objectif pour ce problème test à sept variables est la suivante :

$$f(x) = (x_1 - 10)^2 + 5(x_2 - 12)^2 + x_3^4 + 3(x_4 - 11)^2 + 10x_5^6 + 7x_6^2 + x_7^4 - 4x_6x_7 - 10x_6 - 8x_7,$$

avec $x_i \in [-10, 10],\ \forall i = 1, \ldots, 7$. Les contraintes qui lui sont associées sont :

$$\begin{array}{ll} c_1(x) = 2x_1^2 + 3x_2^4 + x_3 + 4x_4^2 + 5x_5 - 127 & \leq 0 \\ c_2(x) = 7x_1 + 3x_2 + 10x_3^2 + x_4 - x_5 - 282 & \leq 0 \\ c_3(x) = 23x_1 + x_2^2 + 6x_6^2 - 8x_7 - 196 & \leq 0 \\ c_4(x) = 4x_1^2 + x_2^2 - 3x_1x_2 + 2x_3^2 + 5x_6 - 11x_7 & \leq 0 \end{array} .$$

L'optimum global pour ce problème test se trouve en $x^* = (2.330499, 1.951372, -0.4775414, 4.365726, -0.6244870, 1.038131, 1.594227)$ et auquel $f(x^*) = 680.6300573$.

CarSideImpact-RC

Ce problème test modélise l'impact latéral entre une voiture et une barrière. L'enjeu de ce problème est de minimiser le poids du véhicule tout en maintenant ses performances lors d'un tel accident. Les contraintes de ce problème test sont modélisées par des critères liés à la sécurité des passagers. La formulation mathématique de ce problème test est la suivante :

$$\begin{cases} \min\limits_{x \in D^{11}} f(x) = W \text{ (poids du véhicule)} \\ s.c. \quad c_1(x) = L_a \text{ (charge au niveau de l'abdomen)} \leq 1 \text{ kN} \\ \qquad c_2(x) = C_u \text{ (thorax supérieur du mannequin)} \leq 0.32 \text{ m/s} \\ \qquad c_3(x) = C_m \text{ (thorax moyen du mannequin)} \leq 0.32 \text{ m/s} \\ \qquad c_4(x) = C_l \text{ (thorax inférieur du mannequin)} \leq 0.32 \text{ m/s} \\ \qquad c_5(x) = R_u \text{ (déviation de la côte supérieure)} \leq 32 \text{ mm} \\ \qquad c_6(x) = R_m \text{ (déviation de la côte moyenne)} \leq 32 \text{ mm} \\ \qquad c_7(x) = R_l \text{ (déviation de la côte inférieure)} \leq 32 \text{ mm} \\ \qquad c_8(x) = P_f \text{ (force du pubis)} \leq 4 \text{ kN} \\ \qquad c_9(x) = V_M \text{ (Vitesse du pilier V au point central)} \leq 9.9 \text{ mm/ms} \\ \qquad c_{10}(x) = V_F \text{ (Vitesse de la porte avant au niveau du pilier V)} \leq 15.7 \text{ mm/ms}, \end{cases}$$

où x est un point issu de l'espace de recherche disposant de 11 variables. La TABLE 6.3 reprend les valeurs de chacune des variables de la meilleure solution connue définie par $x_{best} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11})$.

x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}	x_{11}
0.5	1.36	0.5	1.202	0.5	1.12	0.5	0.345	0.192	8.87307	-18.99808

TABLE 6.3 – Valeurs des variables de la meilleure solution connue x_{best} pour le problème test CarSideImpact.

La fonction objectif et les contraintes du problème test CarSideImpact sont définies de manière exhaustive par les expressions suivantes :

$$
\begin{aligned}
W &= 1.98 + 4.9x_1 + 6.67x_2 + 6.98x_3 + 4.01x_4 + 1.78x_5 + 2.73x_7,\\
L_a &= 1.16 - 0.3717x_2x_4 - 0.00931x_2x_{10} - 0.484x_3x_9 + 0.01343x_6x_{10},\\
C_u &= 0.261 - 0.0159x_1x_2 - 0.188x_1x_8 - 0.019x_2x_7 + 0.0144x_3x_5\\
&\quad + 0.00087575x_5x_{10} + 0.08045x_6x_9 + 0.00139x_8x_{11} + 0.00001575x_{10}x_{11},\\
C_m &= 0.214 + 0.00817x_5 - 0.131x_1x_8 - 0.0704x_1x_9 + 0.03099x_2x_6 - 0.018x_2x_7\\
&\quad + 0.0208x_3x_8 + 0.121x_3x_9 - 0.00364x_5x_6 + 0.0007715x_5x_10 - 0.0005354x_6x_{10} + 0.00121x_8x_{11},\\
C_l &= 0.74 - 0.061x_2 - 0.163x_3x_8 + 0.001232x_3x_{10} - 0.166x_7x_9 + 0.227x_2^2,\\
R_u &= 28.98 + 3.818x_3 - 4.2x_1x_2 + 0.0207x_5x_{10} + 6.63x_6x_9 - 7.7x_7x_8 + 0.32x_9x_{10},\\
R_m &= 33.86 + 2.95x_3 + 0.1792x_{10} - 5.057x_1x_2 - 11x_2x_8 - 0.0215x_5x_{10} - 9.98x_7x_8 + 22x_8x_9,\\
R_l &= 4.36 - 9.9x_2 - 12.9x_1x_8 + 0.1107x_3x_{10},\\
P_f &= 4.72 - 0.5x_4 - 0.19x_2x_3 - 0.0122x_4x_{10} + 0.009325x_6x_{10} + 0.000191x_{11}^2,\\
V_M &= 10.58 - 0.674x_1x_2 - 1.95x_2x_8 + 0.02054x_3x_{10} - 0.0198x_4x_{10} + 0.028x_6x_{10},\\
V_F &= 16.45 - 0.489x_3x_7 - 0.843x_5x_6 + 0.432x_9x_{10} - 0.0556x_9x_{11} - 0.000786x_{11}^2.
\end{aligned}
$$

La valeur de la fonction objectif évaluée en x_{best} est égale à 22.84298. Les bornes utilisées pour chaque variable dans ce problème test sont $x_i \in [0.5, 1.5]$, pour $i = 1, 2, \ldots, 7$, $x_8, x_9 \in \{0.192, 0.345\}$, et $x_{10}, x_{11} \in [-30, 30]$.

Spring-RC

Ce problème cherche à minimiser le poids d'une corde de tension. Il s'agit d'un problème à trois dimensions, où chaque variable représente les éléments suivants :

- le diamètre du câble pour x_1,
- le diamètre moyen de la bobine pour x_2,
- le nombre de bobines pour x_3.

Ce problème est donné par la formulation ci-après :

$$\begin{cases} \min\limits_{x\in D^3} f(x) = (x_3+2)x_1^2x_2 \\ \text{s.c. } c_1(x) = 1 - \dfrac{x_2^3x_3}{7.178x_1^4} \leq 0 \\ \qquad c_2(x) = \dfrac{4x_2^2 - x1x_2}{12.566(x_1^3x_2) - x_1^4} + \dfrac{1}{5108x_1^2} - 1 \leq 0 \\ \qquad c_3(x) = 1 - \dfrac{140.45x_1}{x_2^2x_3} - 1 \leq 0 \\ \qquad c_4(x) = \dfrac{x_1+x_2}{1.5} - 1 \leq 0 \end{cases},$$

où les bornes associées aux variables sont $0.05 \leq x_1 \leq 2.0$, $0.25 \leq x_2 \leq 1.3$ et $2 \leq x_3 \leq 15$. La meilleure solution, à notre connaissance, de ce problème est donnée par $x_{best} = (0.05169, 0.35675, 11.287126)$. L'évaluation de la fonction objectif en ce point vaut 0.012665.

WB4-RC

La formulation du problème test WB4-RC à six variables est donnée par l'expression suivante :

$$\begin{cases} \min\limits_{\mathbf{x}\in D^6} f(x) = 1.10471x_3^2(x_6 + x_1x_4) + 0.04811x_4x_5(14+x_6) \\ \text{s.c. } c_1(x) = \tau(x) - 13600 \leq 0 \\ \qquad c_2(x) = \sigma(x) - 30000 \leq 0 \\ \qquad c_3(x) = x_1 - x_4 \leq 0 \\ \qquad c_4(x) = 0.10471(x_1^2) + 0.04811x_3x_4(14+x_2) - 5 \leq 0 \\ \qquad c_5(x) = 0.125 - x_1 \leq 0 \\ \qquad c_6(x) = \delta(x) - 0.25 \leq 0 \\ \qquad c_7(x) = 6000 - Pg(x) \leq 0 \end{cases},$$

où

$$Pg(x) = \frac{4.013(30\times10^6)\sqrt{\frac{x_3^2x_4^6}{36}}}{196}\left(1 - \frac{x_3\sqrt{\frac{30\times10^6}{4(12\times10^6)}}}{28}\right)$$

$$\tau(x) = \sqrt{(t_1)^2 + (2t_1t_2)\frac{x_2}{2R} + (t_2)^2}$$

$$t_1 = \frac{6000}{\sqrt{2}x_1x_2}$$

$$t_2 = \frac{MR}{J}$$

$$
\begin{aligned}
M &= 6000(14 + \frac{x_2}{2}) \\
R &= \sqrt{\frac{x_2^2}{4}\left(\frac{x_1 + x_3}{2}\right)^2} \\
J &= 2\left\{x_1 x_2 \sqrt{2}\left[\frac{x_2^2}{12} + \left(\frac{x_1 + x_3}{2}\right)^2\right]\right\} \\
\delta(x) &= \frac{65856000}{(30 \times 10^6) x_4 x_3^3} \\
\sigma(x) &= \frac{504000}{x_4 x_3^2}
\end{aligned}
$$

La meilleure solution connue est donnée par $x_{best} = (1, \text{Steel}, 0.1875, 8.25, 0.25, 1.75)$ et $f(x_{best}) = 1.9509$. Ce problème dispose de deux variables continues qui sont x_5 et x_6. Les variables x_1, x_2, x_3 et x_4 sont continues.

Chapitre 7

Outils d'analyse des résultats

Au coeur de ce chapitre, nous allons décrire les outils graphiques et quantitatifs qui ont été utilisés pour analyser les résultats obtenus sur Zenobe. Lors de nos expérimentations, nous avons testé les modèles d'arbre de régression et de forêt aléatoire d'arbres de régression en manipulant diverses valeurs des hyperparamètres qui leur sont associés. Nous avons dès lors construit différentes "versions" de ces modèles et nous avons évalué les performances de celles-ci en terme de validation externe et en terme d'optimisation. Les informations nécessaires à la compréhension de la procédure de validation externe sont développées dans la première section de ce chapitre. Les outils que nous allons expliquer dans ce chapitre ne sont pas les mêmes selon le cadre d'étude dans lequel nous nous plaçons, à savoir la validation externe ou l'optimisation. Dès lors, nous avons consacré la dernière section aux outils propres à l'optimisation. Dans chacune des deux dernières sections se distinguent les outils permettant de mesurer la performance des différentes versions des méta-modèles d'un point de vue global d'une part, et de manière spécifique pour chaque problème test, d'autre part.

7.1 Principe de la validation externe

Nous allons maintenant expliquer le principe de la technique de validation externe. Il est important de préciser qu'en pratique, un modèle qui donne de bons résultats en validation externe ne donne pas forcément des résultats aussi bons une fois appliqué au sein de la boucle SBO d'optimisation de Minamo. Ceci constitue la raison pour laquelle nous avons évalué la qualité des différentes versions des méta-modèles étudiés selon ces deux aspects, la validation externe et l'optimisation, lors de nos expérimentations. La validation externe consiste à évaluer la qualité d'un méta-modèle sur un ensemble de problèmes tests en faisant appel à un DOE dit "de validation" qui contient $1000 * n$ individus, où n est la dimension du problème (1.1). Ce DOE de validation est différent du DOE d'entrainement sur base duquel le méta-modèle est construit et qui contient, quant à lui, $3 * n$ individus. La technique de validation externe évalue la performance d'un méta-modèle, pour un problème test donné, en mesurant l'erreur de prédiction réalisée par ce méta-modèle sur les données du DOE de validation. Autrement dit, des métriques

d'erreur vont servir à calculer l'erreur effectuée par le méta-modèle pour prédire la valeur des évaluations des individus du DOE de validation par la fonction objectif f, par exemple, issue du problème (1.1). Notons que Cenaero se sert de métriques d'erreur basées sur le coefficient de Pearson. Le coefficient de corrélation de Pearson, noté R, est défini dans [6] par :

$$R = \frac{l\sum_{i=1}^{l} y_i\tilde{y}_i - \sum_{i=1}^{l} y_i \sum_{i=1}^{l} \tilde{y}_i}{\sqrt{l\sum_{i=1}^{l} y_i^2 - (\sum_{i=1}^{l} y_i)^2}\sqrt{l\sum_{i=1}^{l} \tilde{y}_i{}^2 - (\sum_{i=1}^{l} \tilde{y}_i)^2}},$$

où y_i dénote l'évaluation par f de l'individu $x_i \in D^n$ et $\tilde{y}_i$ représente la prédiction que le méta-modèle étudié associe à x_i. Nous précisons également qu'à partir de ce chapitre, toutes les fonctions intervenant dans (1.1) seront appelées des "réponses" pour chacun des problèmes tests étudiés. Plus précisément, les réponses correspondant à la fonction objectif du problème d'optimisation considéré seront dénotées par la lettre "F", tandis que la ième contrainte du problème étudié sera nommée "G0i".

7.2 Validation externe

7.2.1 Outils de comparaison par problème test

Chacune des versions des méta-modèles testés a été exécutée cent fois, c'est-à-dire que l'ensemble des tests que nous avons effectués ont été réalisés sur cent runs différents. Cela signifie également que cent DOE d'entrainement différents ont été créés pour chacune des versions des méta-modèles testés et pour chaque problème test étudié. Etant donné le caractère aléatoire de la génération du DOE lors de la construction d'un méta-modèle, il est nécessaire d'effectuer ces exécutions successives afin d'obtenir des données statistiques fiables pour comparer les différentes versions. Nous disposons de scripts fournis par Cenaero pour générer des rapports automatiquement sur base des résultats récupérés sur le supercalculateur Zenobe. Dans ces rapports, les résultats obtenus sont détaillés à l'aide des graphiques explicités ci-dessous pour chaque problème test.

Boîte à moustaches

Nous abordons les boîtes à moustaches décrivant la qualité de chacune des versions testées pour un problème test donné. Nous pouvons visualiser un exemple d'un tel graphe à la FIGURE 7.1. La première version dont nous pouvons observer les résultats sur le graphe assigne la valeur "mae" à l'hyperparamètre "criterion" pour le modèle de l'arbre de régression. Celle-ci se nomme "DT-crit-mae". La seconde version, nommée "DT-default" correspond à celle de l'arbre de régression avec ses valeurs par défaut. Ce type de graphe est disponible dans les rapports selon plusieurs coefficients de qualité : corpea, pea_rp et iv. Le premier coefficient de qualité correspond à une version corrigée du coefficient de

corrélation de Pearson. Le second coefficient, nommé pea_rp, combine le coefficient de Pearson avec un coefficient qui tient compte de la "préservation de l'ordre" du modèle ("ranking preservation", en anglais). Cette préservation de l'ordre du modèle peut être assimilée à la préservation de la monotonie du modèle. Des informations supplémentaires concernant la préservation de l'ordre des modèles sont disponibles dans [33]. Le troisième coefficient permet de déterminer si le modèle est interpolant ou non. Les FIGURES 7.1, 7.2 et 7.3 illustrent ces graphiques pour la réponse F du problème test Ackley_10 et pour les coefficients corpea, pea_rp et iv, respectivement. Pour chaque boite à moustache, le triangle vert représente la moyenne des valeurs du coefficient de qualité considéré sur les cent exécutions effectuées. La barre horizontale orange, quant à elle, indique la médiane des valeurs du coefficient de qualité considéré sur l'ensemble des exécutions pour ce problème test et pour la version observée.

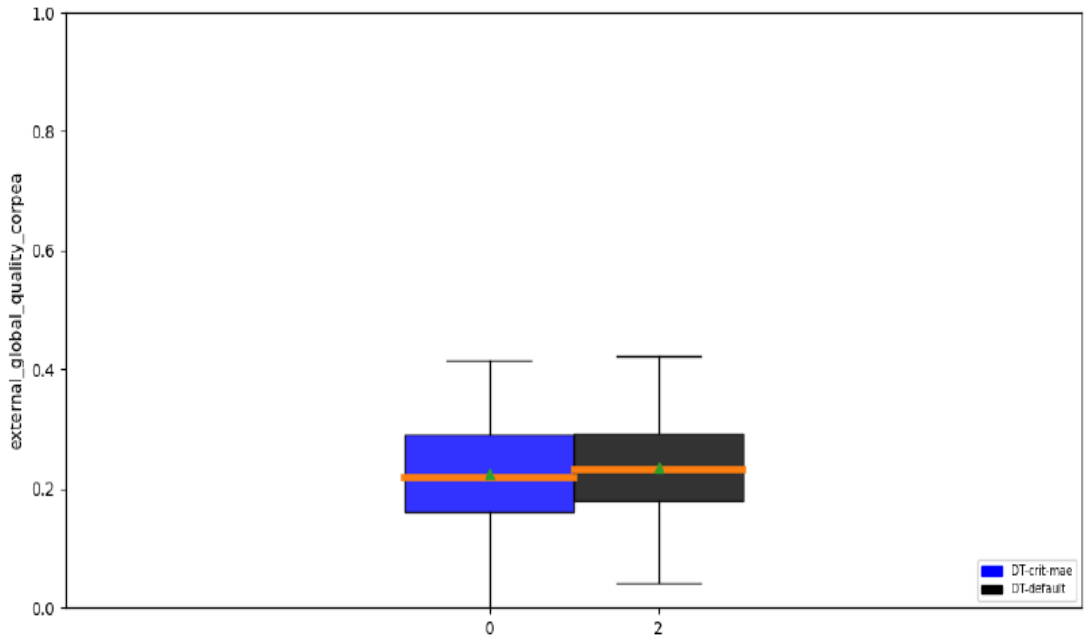

FIGURE 7.1 – Illustration par les boîtes à moustaches de la qualité selon le coefficient corpea des versions DT-crit-mae et DT-default pour la réponse F du problème test Ackley_10.

Les coefficients de qualité corpea, pea_rp et iv prennent des valeurs dans l'intervalle $[0, 1]$. Si la qualité d'une version testée pour une réponse du problème étudié est proche de 1 pour les coefficients corpea et pea_rp, cela sigifie que la version testée est un bon modèle car elle fournit une bonne approximation de la réponse que nous cherchons à approximer. L'interprétation des valeurs du coefficient iv est différente : si la valeur de ce coefficient pour une version testée est proche de 1, cela signifie que cette version est interpolante. Lors

des expérimentations en validation externe, nous nous sommes particulièrement intéressés aux valeurs des coefficients corpea et pea_rp pour évaluer la qualité des versions des modèles d'arbre de décision ou de forêt aléatoire que nous avons testées.

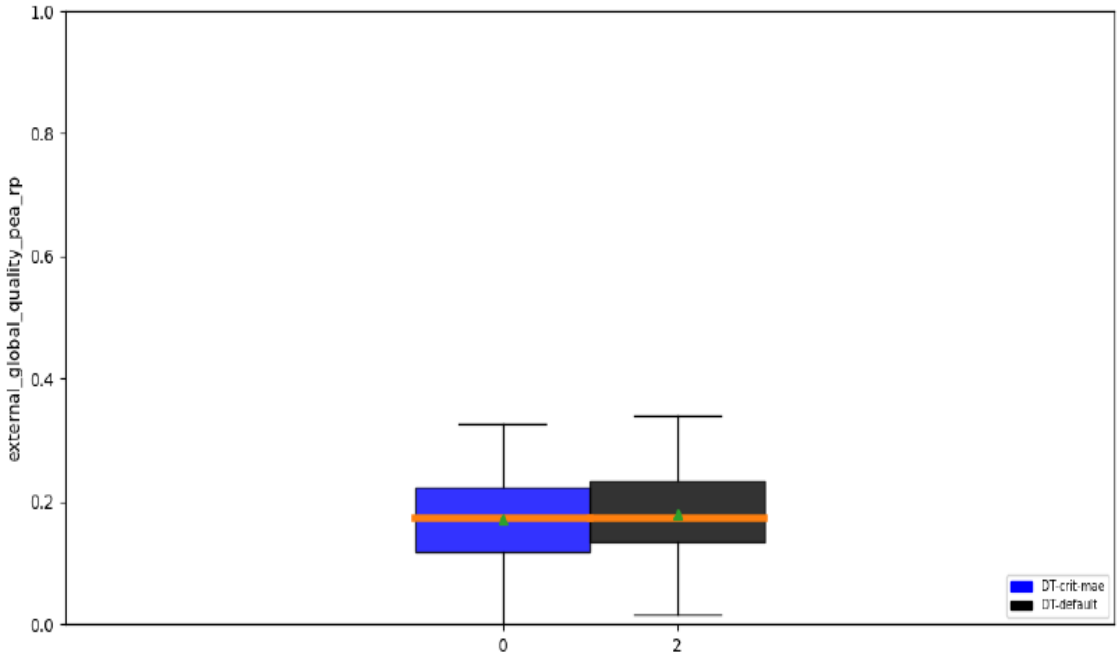

FIGURE 7.2 – Illustration par les boîtes à moustaches de la qualité selon le coefficient pea_rp des versions DT-crit-mae et DT-default pour la réponse F du problème test Ackley_10.

Les boîtes à moustaches des FIGURES 7.1 et 7.2 ne nous permettent pas de conclure qu'une version est meilleure qu'une autre car leurs boites à moustaches sont sensiblement identiques. Grâce à la FIGURE 7.3 nous savons que les deux versions testées, à savoir DT-crit-mae et DT-default sont toutes les deux des modèles interpolants car la boite à moustache associée à chacune de ces versions est réduite à la valeur 1.

Tableau statistique

Un tableau statistique reprenant les informations concernant la moyenne et la médiane pour chacun des trois coefficients de qualité étudié et pour toutes les versions testées fait également partie des outils dont nous disposons dans les scripts automatiques fournis par Cenaero. Les informations de ce tableau correspondent aux données de la médiane et de la moyenne des boites à moustaches décrites précédemment. Cependant, le tableau statistique donne des valeurs plus précises des résultats de nos tests concernant la moyenne et la médiane. Un exemple de ce tableau statistique est fourni à la FIGURE 7.4 pour la

réponse F du problème test Ackley_10. Nous nous sommes intéressés en particulier aux valeurs de la médiane dont nous disposons dans ces tableaux statistiques lors de nos tests afin d'évaluer les différentes versions créées.

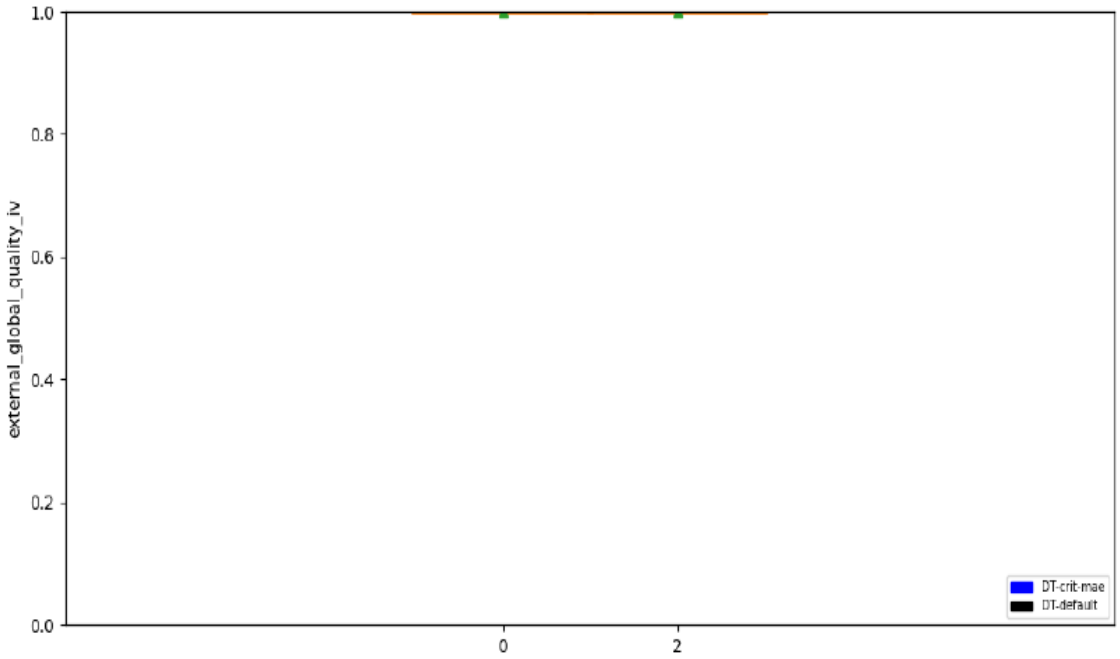

FIGURE 7.3 – Illustration par les boîtes à moustaches de la qualité selon le coefficient iv des versions DT-crit-mae et DT-default pour la réponse F du problème test Ackley_10.

Quality	Minamo	1-1	
		mean	median
corpea	DT-crit-mae	[0.222,0.225]	0.219
	DT-default	**[0.233,0.236]**	**0.23**
iv	DT-crit-mae	**[1.0,1.0]**	**1.0**
	DT-default	[0.999,0.999]	**1.0**
pea_rp	DT-crit-mae	[0.169,0.172]	**0.173**
	DT-default	**[0.177,0.18]**	**0.173**

FIGURE 7.4 – Illustration du tableau statistique reprenant les valeurs moyennes et médianes des coefficients corpea, pea_rp et iv pour chacune des versions testées pour la réponse F du problème test Ackley_10.

7.2.2 Performances globales

Afin d'obtenir une vue globale des performances d'une version sur l'ensemble des réponses de l'ensemble des problèmes tests étudiés, nous disposons de diagrammes en bâtonnets. Ceux-ci affichent le score global normalisé de chaque version pour les coefficients de qualité corpea, pea_rp et iv cités précédemment. Ceux-ci ont été compilés grâce aux implémentations de S. Pool Marquez dans les scripts de Cenaero. Prenons, à titre d'exemple, les résultats présentés par ce type de graphe lorsque le coefficient de qualité utilisé est corpea et pour les versions DT-crit-mae et DT-default que nous évoquons depuis le début de cette section. Ce diagramme en bâtonnets est illustré à la FIGURE 7.5.

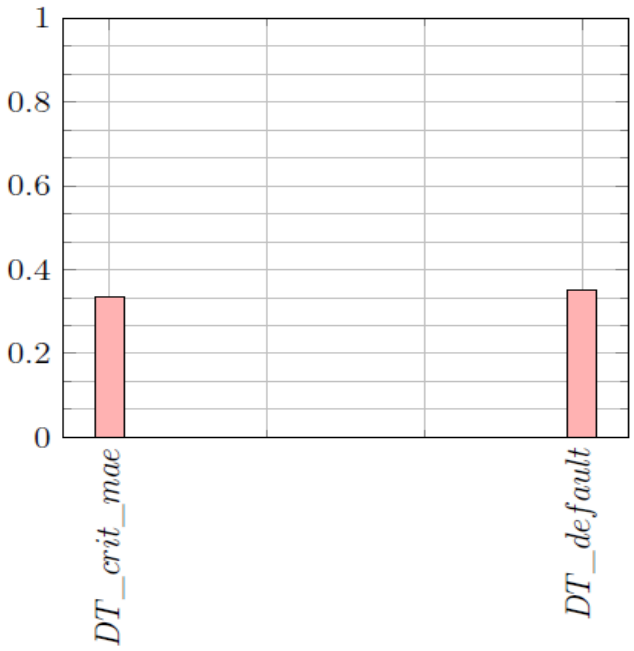

FIGURE 7.5 – Score global normalisé des versions DT-crit-mae et DT-default pour le coefficient de qualité corpea.

Afin de construire le graphique de la FIGURE 7.5, des points sont tout d'abord attribués à chacune des versions sur base de sa performance. Ces points sont accordés pour chaque réponse de chaque problème test, à chaque run. Un nombre de points plus important est attribué aux versions ayant des performances proches de 1 pour le coefficient de qualité étudié. Ensuite, les points de chaque version sont divisés par le nombre maximal de points attribués parmi ces deux versions afin de normaliser les résultats. Dès lors, nous pouvons noter qu'au plus proche de 1 est la valeur du score global normalisé d'une version, au plus cette version est performante. Dans le cas de la FIGURE 7.5, aucune des deux versions ne se démarque par rapport à l'autre de manière franche même si la performance de la version DT-default semble très légèrement supérieure à celle de la version DT-crit-mae.

7.3 Optimisation

7.3.1 Outils de comparaison par problème test

Nous avons testé différentes versions des modèles d'arbre de régression et de forêt aléatoire d'arbres de régression en évaluant leur performance, une fois que ceux-ci sont utilisés au sein de l'algorithme génétique de la boucle SBO de Minamo. Pour ce faire, nous avons considéré un DOE de départ contenant $n + 1$ individus, où n est le nombre de variables du problème (1.1). Le nombre d'itérations au sein de la boucle SBO est fixé, pour chaque problème test étudié, à $100 - (n + 1)$. En ce qui concerne les caractéristiques de l'algorithme génétique, la taille de sa population initiale est 100 et le nombre de générations vaut 100 également. De plus, nous précisons que cent exécutions de la boucle SBO ont été réalisées pour chaque problème test. Nous développons ici les différents graphiques générés automatiquement sur base des résultats obtenus sur Zenobe pour chaque problème test. Dans cette section sur les outils d'analyse propres à l'optimisation, nous avons choisi de montrer des graphiques faisant intervenir les versions DT-default, DT-omss3 et RF-default évaluées lors de la phase d'expérimentation. Les versions DT-default et RF-default correspondent aux versions par défaut des modèles d'arbre de régression et de forêt aléatoire d'arbres de régression, respectivement. La version DT-omss3, quant à elle, représente le modèle d'arbre de régression pour lequel la valeur de l'hyperparamètre "min_samples_split" a été fixée à 3, tout en laissant les valeurs des autres hyperparamètres de ce modèle à leur valeur par défaut.

Graphique de convergence

Les graphiques de convergence, dont nous pouvons voir un exemple à la FIGURE 7.6 pour le problème test PressureVessel-RIDC, permettent d'observer la rapidité de la version testée à converger vers la valeur optimale de l'objectif. La FIGURE 7.6 nous indique ici clairement que la version RF-default est la plus rapide car elle est la première à atteindre ce que nous appelons le seuil "upper_quartile". Ce seuil est représenté par la ligne en pointillés bleus sur le graphe. Ce seuil est considéré comme le plus large, c'est-à-dire qu'il s'agit du seuil le moins contraignant sur la valeur de l'objectif. Le seuil nommé "lower_quartile" en pointillés noirs, quant à lui, définit le seuil le plus fin, c'est-à-dire le seuil le plus contraignant sur la valeur de l'objectif. Nous devons également préciser que sur les graphes de convergence, le tracé des courbes des différentes versions n'apparait que lorsqu'un point admissible est trouvé. La courbe qui représente la médiane des valeurs de l'objectif pour la version RF-default sur les cent exécutions effectuées est celle qui converge le plus rapidement. En effet, après un peu moins de 50 itérations, cette courbe rouge atteint déjà le seuil "upper_quartile". Sur le graphe de la FIGURE 7.6, nous observons également un axe en bas du graphique, nommé "iterations axis", et un autre en haut du graphe, nommé "Evaluations axis". Nous remarquons que la graduation de ces axes est décalée de 5 unités. Cela est dû au fait qu'au départ de la boucle SBO, les 5 individus qui constituent le DOE initial pour le problème test PressureVessel-RIDC doivent être évalués.

Nous allons maintenant expliquer comment ces seuils "fin" et "large" sur la valeur de l'objectif sont déterminés. Pour cela, nous allons faire appel aux boites à moustaches illustrées à la FIGURE 7.7. Il s'agit du graphe de la distribution finale de la valeur de l'objectif pour chacune des trois versions testées dans le cadre du problème d'optimisation PressureVessel-RIDC. Nous rappelons que pour chaque problème test, cent runs de la boucle SBO ont été effectués. Sur base de ces cent runs, nous allons pouvoir établir les boîtes à moustaches de la FIGURE 7.7. La valeur du troisième quartile de la boîte à moustache de chacune des versions est ensuite stockée dans un ensemble, noté T. Parmi les trois valeurs de cet ensemble, la plus petite est sélectionnée pour définir la valeur du seuil fin. Quant au seuil large, c'est la plus grande valeur contenue dans l'ensemble T qui sera reprise pour représenter sa valeur.

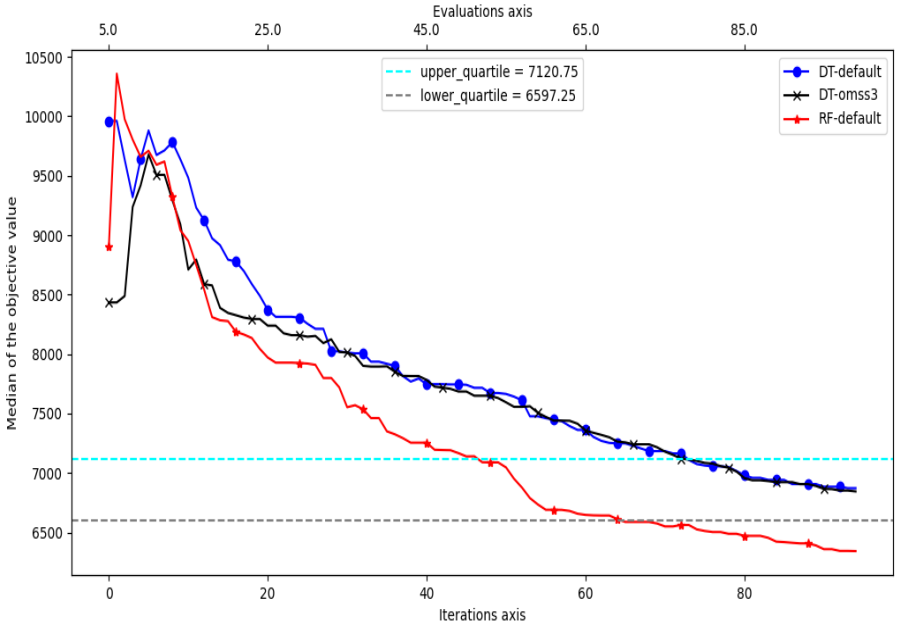

FIGURE 7.6 – Convergence de la médiane pour la fonction objectif du problème test PressureVessel-RIDC.

Respect des contraintes

Le second type de graphique que nous allons analyser dans le cadre de l'optimisation est le graphe de l'évolution de la moyenne du nombre de contraintes non respectées en fonction des itérations pour un problème test avec contraintes donné sur les cent runs effectués de la boucle SBO. Un exemple de ce type de graphique pour le problème test PressureVessel-RIDC est donné à la FIGURE 7.8. Sur cette figure, nous pouvons observer que les courbes des trois versions testées DT-default, DT-omss3 et RF-default ont un comportement similaire et ont tendance à tendre vers 0 au-delà des 20 premières itérations. Aucune des trois versions ne se démarque dès lors par rapport à une autre.

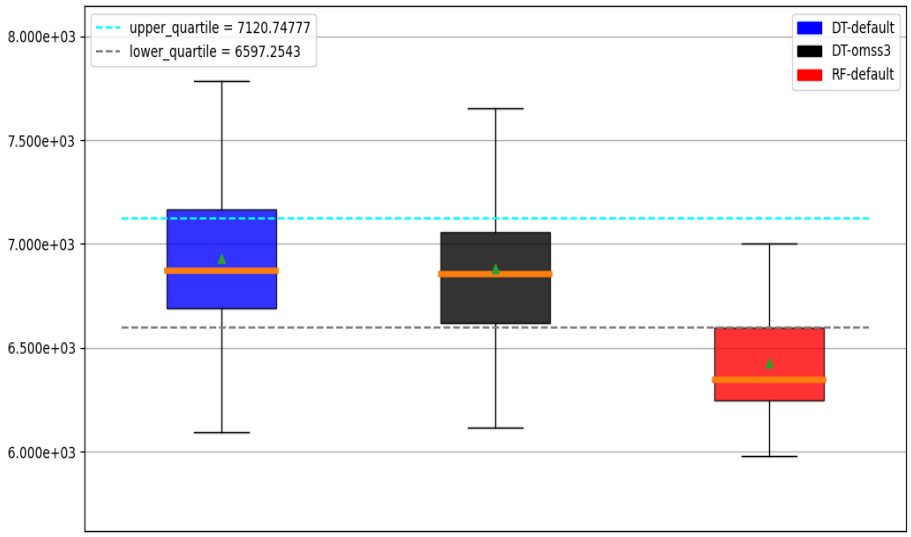

FIGURE 7.7 – Illustration de la distribution finale de la valeur de la fonction objectif pour chaque version pour le problème test PressureVessel-RIDC.

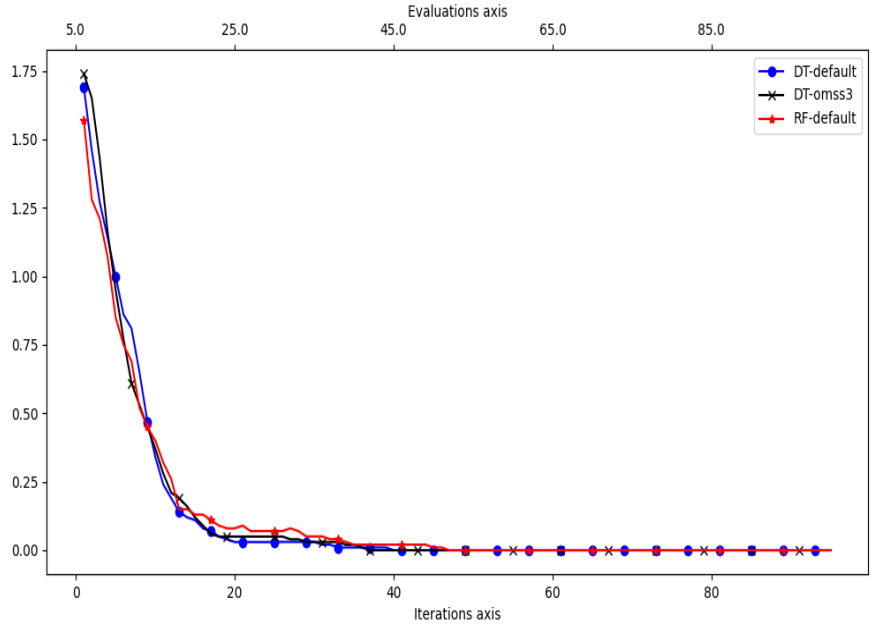

FIGURE 7.8 – Moyenne du nombre de contraintes non respectées pour le problème test PressureVessel-RIDC au cours des itérations.

7.3.2 Performances globales

Afin d'évaluer la performance globale des versions testées en optimisation, nous avons recours aux graphiques du profil de performance introduit par Dolan et Moré [18]. Ces graphiques nous permettent de connaitre le pourcentage de problèmes résolus pour chaque version testée en fonction de la proportion, notée α, d'itérations utilisées pour résoudre le problème test. Afin d'établir ces graphiques, les seuils fins et larges que nous avons définis précédemment sont utilisés comme seuil pour déterminer la valeur de l'objectif à atteindre pour que le problème test soit considéré comme résolu. Ce seuil peut-être qualifié de "seuil de réussite". Nous précisons qu'un problème est considéré "résolu" lorsqu'une solution admissible a été trouvée et que ce seuil de réussite est atteint. Les graphiques des FIGURES 7.9 et 7.10 illustrent ces profils de performance lorsque les seuils de réussite de la résolution du problème d'optimisation sont donnés par le seuil fin et le seuil large, respectivement. Une version possédant un pourcentage élevé de problèmes résolus pour $\alpha = 1$ est une version efficace car elle est capable de résoudre un grand nombre de problèmes en peu d'itérations. Les versions capables de résoudre un grand nombre de problèmes pour une valeur de α grande sont dites "robustes".

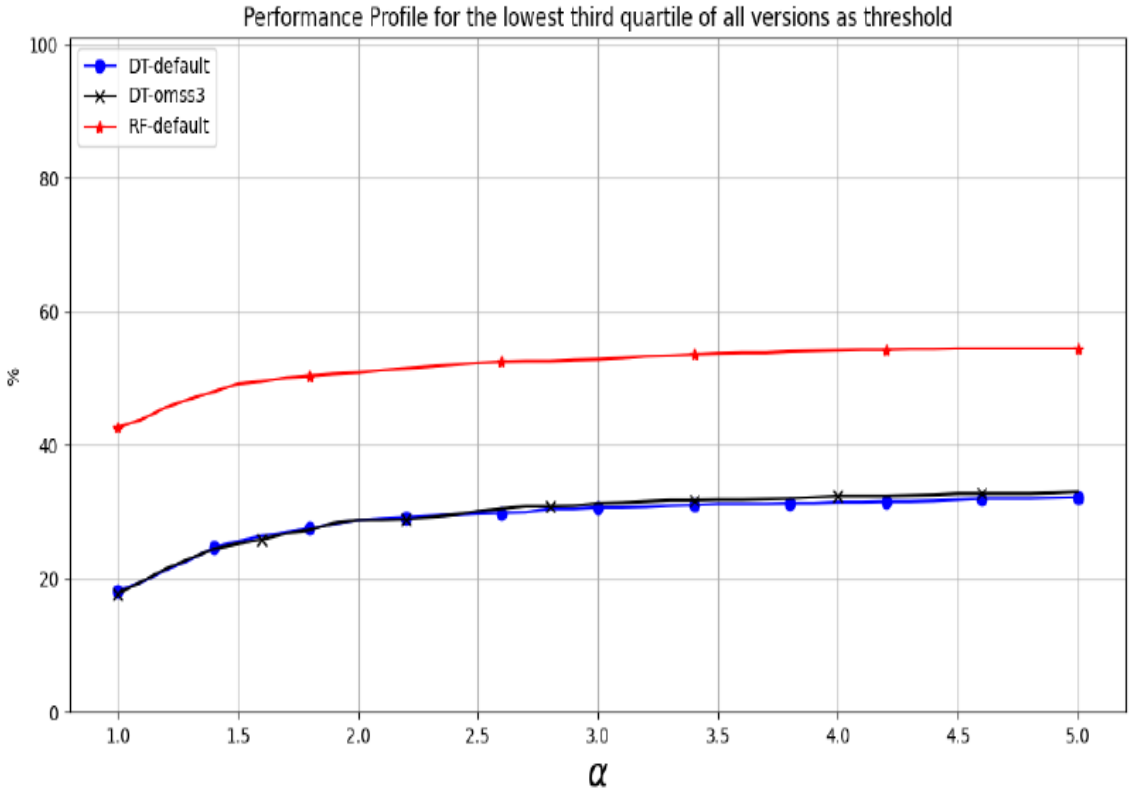

FIGURE 7.9 – Profil de performance pour une proportion α d'itérations et avec pour seuil de réussite le 3ème quartile le plus petit de toutes les versions.

La FIGURE 7.9 indique un profil de performance similaire pour les versions DT-default et DT-omss3. Cependant, le profil de performance de la version RF-default est bien meilleur par rapport aux deux autres versions. Nous déduisons de ce graphe le fait que la versions RF-default est plus efficace et plus robuste que les deux autres versions testées car le pourcentage de problèmes résolus est d'environ 20% supérieur à celui des deux autres versions pour des valeurs de $\alpha = 1$ et $\alpha = 5$.

Sur base du graphe du profil de performance illustré à la FIGURE 7.10, nous observons que les profils des versions DT-default et DT-omss3 ont de nouveau un comportement similaire. La version RF-default est de nouveau la plus robuste et la plus efficace parmi les trois versions testées. Nous pouvons néanmoins noter, si nous comparons les graphiques des FIGURES 7.9 et 7.10, que la robustesse des versions DT-default et DT-omss3 est meilleure dans le cas où le seuil utilisé est le troisième quartile le plus grand de toutes les versions plutôt que lorsqu'il s'agit du troisième plus petit.

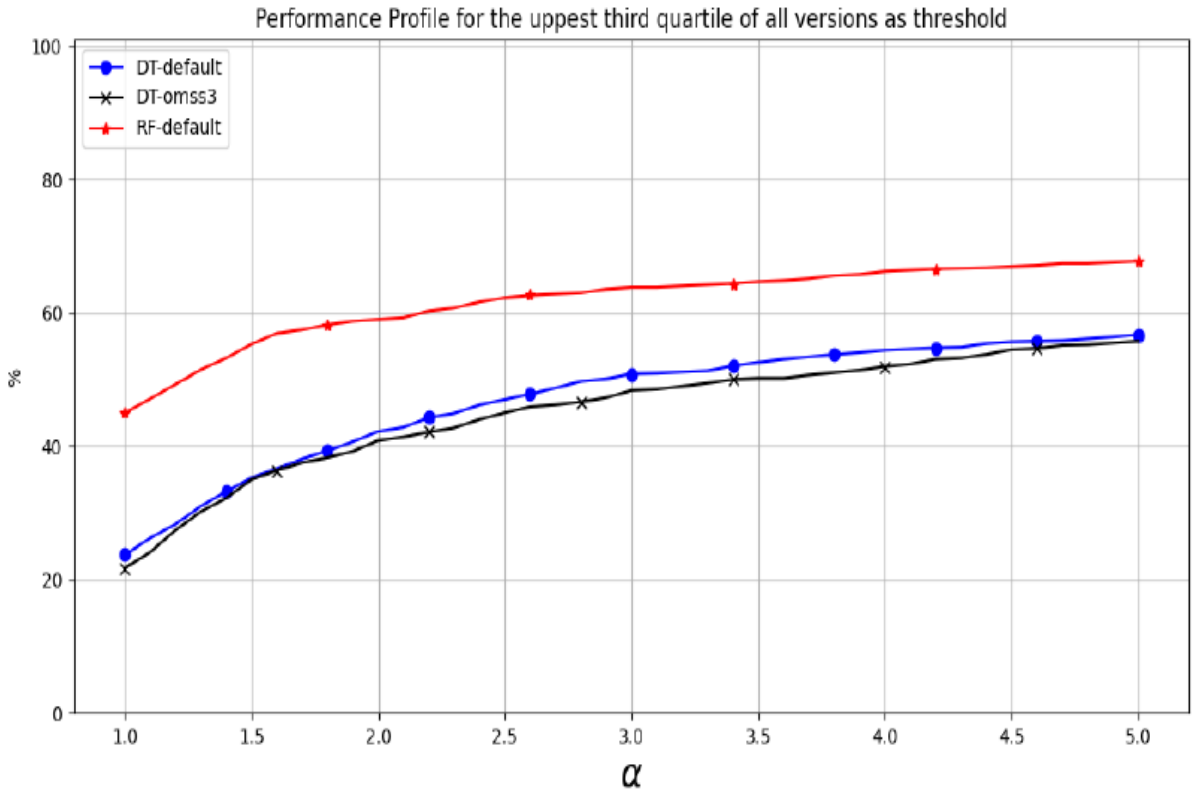

FIGURE 7.10 – Profil de performance pour une proportion α d'itérations et avec pour seuil de réussite le 3ème quartile le plus grand de toutes les versions.

Chapitre 8

Résultats de la phase d'expérimentation

Ce dernier chapitre reprend les résultats obtenus lors de la phase d'expérimentation. Etant donné que nous avons étudié différentes versions des méta-modèles d'arbre de régression et de forêt aléatoire d'arbres de régression selon les deux aspects que sont la validation externe et l'optimisation, nous avons choisi de scinder également ce chapitre en deux parties selon ces deux aspects. Dès lors, nous décrirons tout d'abord les résultats obtenus quant à la qualité des modèles testés en validation externe. Nous distinguerons les expérimentations effectuées sur le modèle d'arbre de décision de celles effectuées sur le modèle de forêt aléatoire. Ensuite, nous nous pencherons sur les résultats des expérimentations en optimisation. Dans la seconde partie de ce chapitre, nous décrirons en premier lieu les différentes versions testées et nous listerons les problèmes tests étudiés dans ce cadre. Ensuite, nous analyserons les meilleures versions parmi celles que nous avons testées. Après, ces meilleurs versions seront comparées au modèle trbf de Minamo. Enfin, nous évoquerons le cas de plusieurs problèmes tests particuliers étudiés dans le cadre de l'optimisation.

8.1 Validation externe

8.1.1 Tests effectués sur l'arbre de régression

Dans le cadre de la validation externe, nous avons effectué nos tests sur les 22 problèmes tests suivants : Borehole-RC1/2/3, Branin-RC1, CarSideImpact-RC, DSMM-RC, FCATMIX-RC1/2, FMIX-RC, FPOLY-RC, FQUAD-RC, FTRIG-RC, Goldstein-RC, OTL-RC, Piston-RC, Roustant-RC, Spring-RC, WB4-RC, Ackley_10, G7, G10 et Rosenbrock_10. Les versions que nous avons testées sont des versions pour lesquelles nous avons fixé la valeur d'un hyperparamètre du modèle d'arbre de régression aux valeurs extrêmes que peuvent prendre celui-ci tout en prennant soin de ne pas choisir la valeur par défaut de ces hyperparamètres. De plus, nous avons également testé une valeur intermédiaire entre ces valeurs extrêmes pour les hyperparamètres pour lesquels cela était possible. Pour rappel, les hyperparamètres qui ont été étudiés lors de la phase d'expérimentation sont explicités à la section 5.1. Le diagramme en bâtonnets de la FIGURE 8.1 reprend les différentes valeurs testées pour chaque hyperparamètre étudié pour le modèle d'arbre

ainsi que les résultats obtenus en validation externe pour ces différentes versions. La notation "crit" fait référence à l'hyperparamètre "criterion", "md" à "max_depth", "mf" à "max_features", "mln" à "max_leaf_nodes", "omsl" à "min_samples_leaf" et "omss" à "min_samples_split". Les résultats affichés sur ce diagramme représentent en pourcentage, la proportion des réponses des problèmes testés pour lesquelles la valeur médiane des coefficients corpea ou pea_rp est à la fois supérieure ou égale à 0.4 et strictement supérieure à celle de la version par défaut du modèle d'arbre de régression. Toutes les hauteurs des batonnets s'interprètent donc de cette manière, excepté en ce qui concerne le premier bâtonnet à gauche sur la FIGURE 8.1 puisqu'il s'agit de la version par défaut de l'arbre de régression. Cette version est nommée DT-Default et attribue à la hauteur de son bâtonnet le pourcentage de réponses pour lesquelles cette version admet une valeur médiane de corpea ou pea_rp supérieure ou égale à 0.4. Nous avons fixé ce critère pour pea_rp à 0.4 car nous estimons qu'une version atteignant ou dépassant cette valeur donne de bons résultats en validation externe. Pour rappel, la valeur médiane des coefficients corpea et pea_rp est reprise dans les tableaux statistiques du type de la FIGURE 7.4 des rapports automatiques que nous générons sur base des résultats des tests effectués sur Zenobe. Sur le graphique de la FIGURE 8.1, la variable n désigne la dimension du problème (1.1).

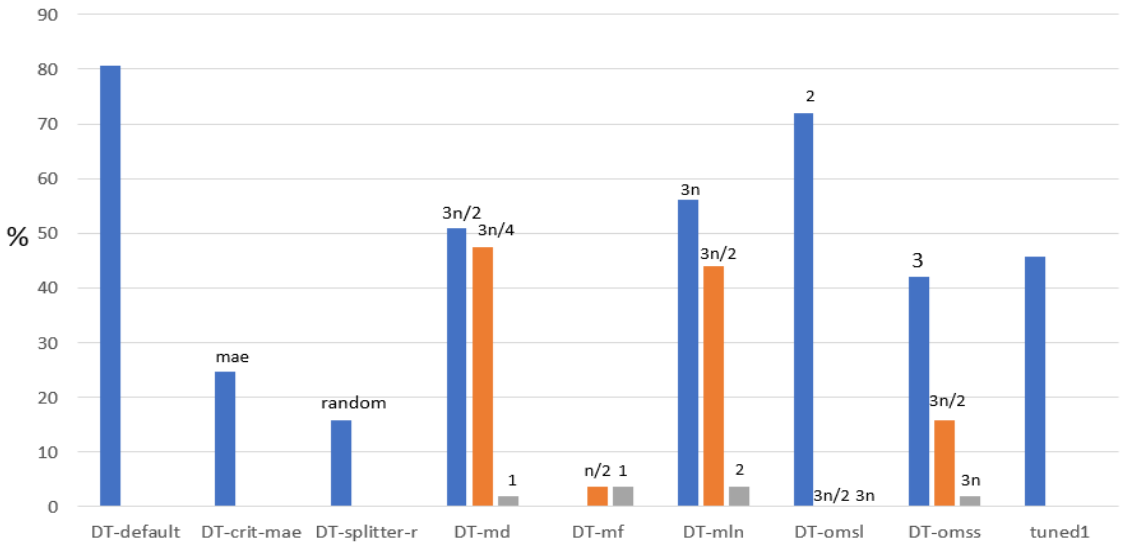

FIGURE 8.1 – Diagramme en batonnets reprenant le pourcentage de réponses des problèmes testés en validation externe pour lesquelles la valeur médiane de corpea ou pea_rp est à la fois supérieure ou égale à 0.4 et strictement supérieure à celle de la version DT-default, pour chacune des versions testées du modèle d'arbre de régression.

Nous remarquons sur ce graphique une version nommée "tuned1". Il s'agit d'une version dont nous dirons qu'elle est "tunée", c'est-à-dire qu'elle fait appel à une fonction "best_params" dans son implémentation qui calcule la meilleure combinaison de valeurs d'hyperparamètres possibles parmi plusieurs valeurs d'hyperparamètres qui lui sont

proposées. La version tuned1 a pour fonction best_params une fonction qui balaye les combinaisons possibles des valeurs des hyperparamètres suivants : $\{None, \frac{3n}{4}, \frac{3n}{2}\}$ pour max_depth, $\{'mse','mae'\}$ pour criterion, $\{'best','random'\}$ pour splitter, $\{None, \frac{3n}{4}\}$ pour max_features, $\{None, \frac{3n}{2}, 3*n\}$ pour max_leaf_nodes, $\{1, 2, \frac{3n}{4})\}$ pour min_samples _leaf et $\{2, 3, \frac{3n}{4}\}$ pour min_samples_split. Nous notons également que la fonction best_ params de la version tuned1 teste toutes ces combinaisons de valeurs d'hyperparamètres tout en ayant 1 pour valeur fixée à l'hyperparamètre *random_state*. C'est au cours des expérimentations que nous avons compris que le modèle d'arbre n'était pas toujours déterministe et nous avons donc décidé de fixé la valeur de l'hyperparamètre random_state à 1 pour cette version tuned1. Pour rappel, lorsque la valeur de random_state est fixée à une valeur numérique, cela implique que le modèle d'arbre créé est déterministe. Afin de sélectionner la meilleure combinaison de valeurs d'hyperparamètres parmi celles qui lui sont proposées, la fonction best_params de la version tuned1 se sert de la technique de validation croisée de type "leave-one-out". Grâce à cette technique de validation croisée, la fonction best_params de la version tuned1 évalue la qualité de l'arbre créé pour chacune des combinaisons de valeurs d'hyperparamètres testées et sélectionne la combinaison qui attribue la meilleure qualité au modèle d'arbre. Nous observons sur le graphe de la FIGURE 8.1 que les valeurs, pour chaque hyperparamètre, qui donnent les meilleures résultats sont celles qui se rapprochent de la valeur par défaut de l'hyperparamètre en question. Nous notons que la valeur $n - 1$ de l'hyperparamètre max_features n'a pas été testée par manque de temps.

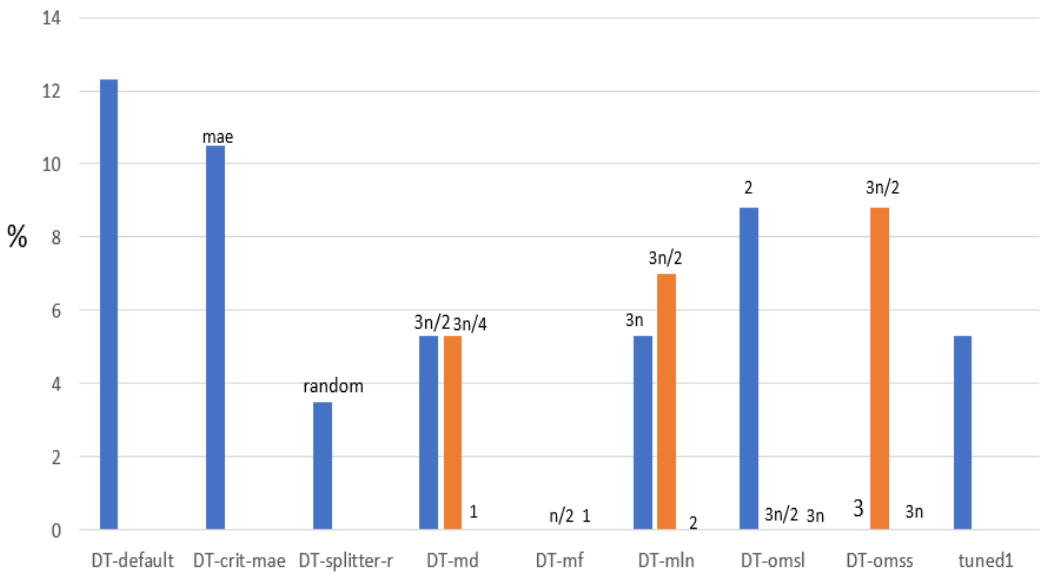

FIGURE 8.2 – Diagramme en batonnets reprenant le pourcentage de réponses des problèmes testés en validation externe pour lesquelles la valeur médiane de corpea ou pea_rp est à la fois supérieure ou égale à 0.4, strictement supérieure à celle de la version DT-default et supérieure ou égale à celle de la version trbf de Minamo, pour chacune des versions testées du modèle d'arbre de régression.

Le second diagramme en bâtonnets que nous avons réalisé sur base des résultats obtenus en validation externe pour l'arbre de régression est représenté à la FIGURE 8.2. Ce graphe affiche, pour chaque version de l'arbre de régression testée, la proportion en pourcentage des réponses des problèmes tests pour lesquelles la valeur médiane des coefficients corpea ou pea_rp est à la fois supérieure à 0.4, strictement supérieure à celle de la version par défaut du modèle d'arbre de régression et supérieure ou égale à celle de la version trbf de Minamo. Nous constatons que, pour l'ensemble des versions testées, ce pourcentage est relativement bas et ne dépasse jamais la valeur associée à la version DT-crit-mae qui est de 10.5%. Certaines versions se démarquent tout de même telles que celle qui concerne l'hyperparamètre max_leaf_nodes pour lequel la valeur a été fixée à $3n/2$ mais aussi le modèle qui fixe la valeur de min_samples_leaf à 2 et celui qui fixe la valeur de min_samples_split à $3n/2$.

8.1.2 Tests effectués sur la forêt aléatoire d'arbres de régression

Le diagramme en bâtonnets illustré à la FIGURE 8.3 reprend les résultats en validation externe pour différentes versions du modèle de forêt aléatoire que nous avons testées. Pour rappel, en ce qui concerne la forêt aléatoire, le seul hyperparamètre étudié et propre à ce modèle est l'hyperparamètre "n_estimators". La valeur associée à cet hyperparamètre correspond au nombre d'estimateurs, c'est-à-dire au nombre d'arbres de régression que contient la forêt. Les différentes valeurs que nous avons testées pour cet hyperparamètre sont : 10 (valeur par défaut), 50, 80 et 1000. Nous avons comparé ces versions de la forêt aléatoire à la version qui contient 100 arbres et non à la version par défaut de la forêt aléatoire qui ne contient que 10 arbres. Cela nous semblait être un choix plus pertinent.

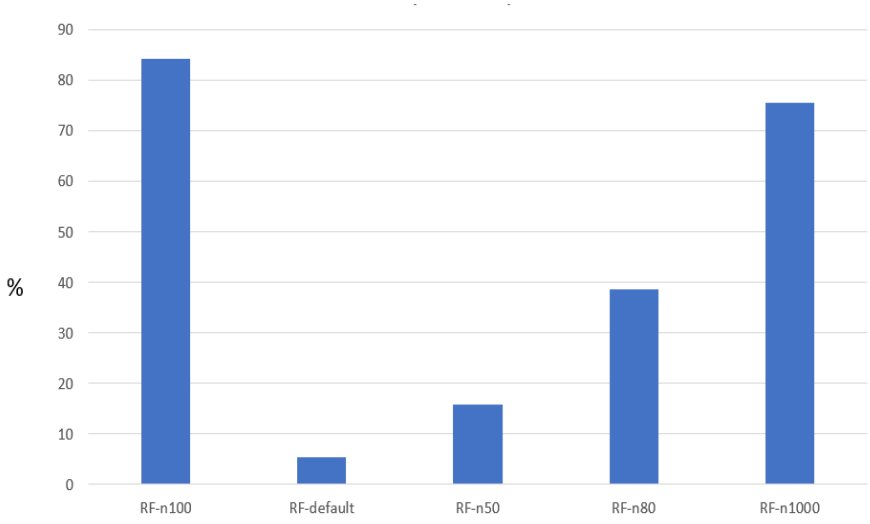

FIGURE 8.3 – Diagramme en batonnets reprenant le pourcentage de réponses des problèmes testés en validation externe pour lesquelles la valeur médiane de corpea ou pea_rp est à la fois supérieure ou égale à 0.4 et supérieure ou égale à celle de la version RF-n100, pour chacune des versions testées du modèle de forêt aléatoire.

Le graphe de la FIGURE 8.3 indique la proportion en pourcentage, pour chaque version, des réponses des problèmes testés pour lesquelles la valeur médiane des coefficients corpea ou pea_rp est à la fois supérieure ou égale à 0.4 et à celle de la version du modèle de forêt aléatoire qui contient 100 arbres. Grâce à ce graphique, nous observons que la version de la forêt aléatoire qui contient 1000 arbres est la version qui donne les meilleurs résultats en terme de validation externe. Notons cependant que cette version n'est pas celle que nous sélectionnerons pour les tests d'optimisation dans la suite du mémoire car le coût d'entrainement d'un modèle contenant autant d'arbres n'est pas raisonnable : en effet, il aura fallu plus de trois jours pour entrainer cette version sur Zenobe.

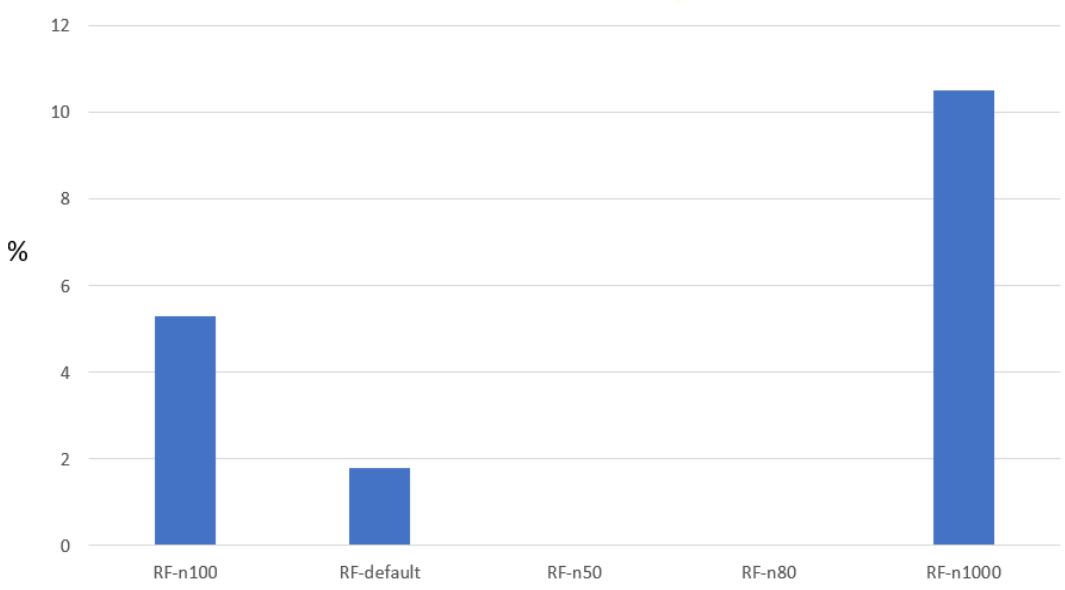

FIGURE 8.4 – Diagramme en batonnets reprenant le pourcentage de réponses des problèmes testés en validation externe pour lesquelles la valeur médiane de corpea ou pea_rp est à la fois supérieure ou égale à 0.4 et à celle de la version RF-n100 et à celle de la version trbf, pour chacune des versions testées du modèle de forêt aléatoire.

Le diagramme de la FIGURE 8.4 reprend pour chaque version que nous avons testée en validation externe pour le modèle de forêt aléatoire d'arbres de régression le pourcentage de réponses sur l'ensemble des problèmes testés pour lesquelles la valeur médiane de corpea ou pea_rp est à la fois supérieure ou égale à 0.4, à celle de la version RF-n100 et à celle de la version trbf. Nous pouvons tirer les mêmes conclusions que pour le graphe de la FIGURE 8.3 en ce qui concerne les résultats de la versions RF-n1000. En effet, c'est cette version qui parvient à égaler le plus de fois la valeur de la médiane du coefficient corpea ou pea_rp sur les cent runs effectués de la version par défaut de Minamo mais la version RF-n1000 n'est pas avantagée étant donné son coût d'entrainement important.

8.1.3 Performance globale

Les outils dont nous disposons grâce aux graphes des FIGURES 8.5 et 8.6 vont nous permettre d'évaluer la performance globale des modèles DT-default et RF-n100 que nous avons utilisés comme versions de référence afin d'évaluer la qualité des versions de modèle d'arbre de régression et de forêt aléatoire testés en validation externe. Pour rappel, ce type de graphe a été décrit dans la section 7.2.2 du mémoire. La FIGURE 8.5 représente le score global normalisé pour le coefficient de qualité corpea des versions DT-default et RF-n100 que nous allons comparer à celui de la version trbf. La FIGURE 8.6 représente le score global normalisé pour le coefficient de qualité pea_rp des versions DT-default et RF-n100 que nous allons également comparer à celui de la version trbf. Grâce à ces graphes, nous observons clairement que le score normalisé de la version trbf devance largement celui des deux autres versions pour les coefficients corpea et pea_rp.

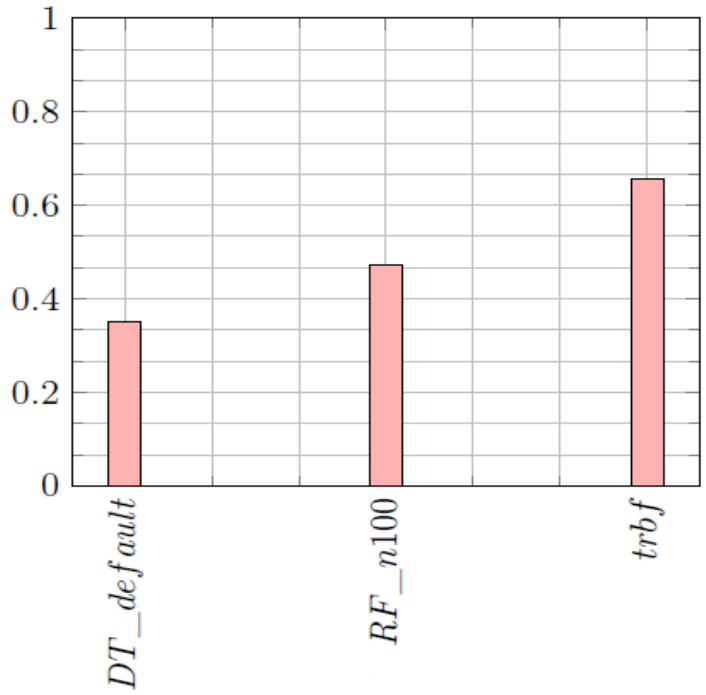

FIGURE 8.5 – Score global normalisé des versions DT-default, RF-n100 et trbf pour le coefficient de qualité corpea.

L'information la plus importante que nous apporte ces graphes réside dans le fait que nous observons clairement ici que le modèle RF-n100 lié à la forêt aléatoire est plus performant que le modèle DT-default associé à l'arbre de régression muni de ses hyperparamètres fixés à leur valeur par défaut.

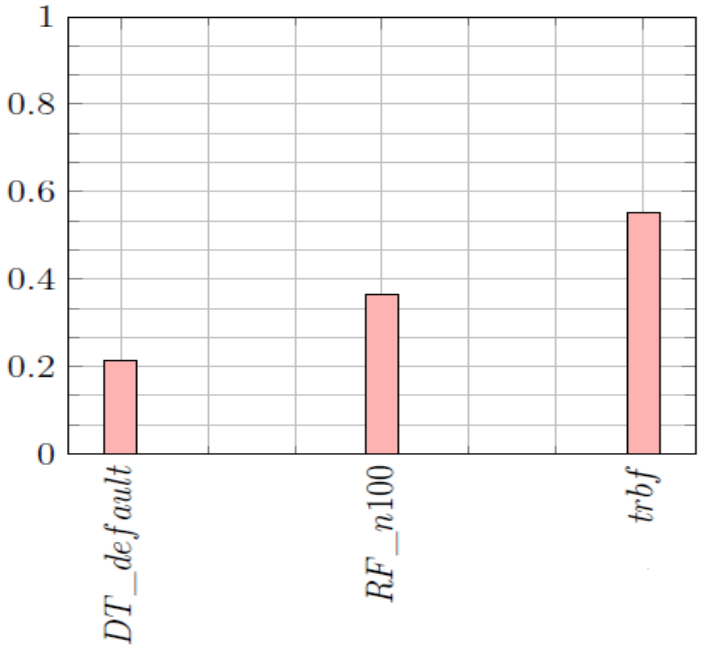

FIGURE 8.6 – Score global normalisé des versions DT-default, RF-n100 et trbf pour le coefficient de qualité pea_rp.

8.2 Optimisation

8.2.1 Versions testées

Dans cette section, nous allons montrer deux profils de performance regroupant l'ensemble des versions des modèles d'arbre de régression et de forêt aléatoire d'arbres de régression testées en optimisation. L'ensemble des problèmes tests sur lequels nous avons basé nos expérimentations en optimisation contient les onze problèmes suivants : Borehole-RC3, CarSideImpact-RC, DSMM-RC, FCATMIX-RC1, FCATMIX-RC2, G7, G9, G10, Piston-RC, PressureVessel-RIDC et WB4-RC. Sur base des résultats que nous avons obtenus en validation externe et étant donné la fin de vie de Zenobe, nous avons sélectionné les versions qui nous semblaient être les plus pertinentes à tester en optimisation et nous avons été contraints de limiter le nombre de problèmes tests considérés par rapport à ceux que nous avons étudiés en validation externe. Nous détaillons ici chacune de ces versions :

- **"DT-bct1"** : il s'agit de la version qui utilise la meilleure combinaison de valeurs des hyperparamètres que nous avons récupérée à partir de la version tuned1 testée en validation externe. Cette combinaison d'hyperparamètres est donnée par max_depth=None, criterion="mse", splitter="random", max_features=None, max_leaf_nodes=None, min_samples_leaf=1, min_samples_split=2 et random_state=1.

- **"DT-default"** : il s'agit de la version par défaut du modèle d'arbre de régression.
- **"DT-omsl2"** : cette version fait référence au modèle d'arbre de régression pour lequel la valeur de l'hyperparamètre min_samples_leaf a été fixée à 2.
- **"DT-omss3"** : cette version fait référence au modèle d'arbre de régression pour lequel la valeur de l'hyperparamètre min_samples_split a été fixée à 3.
- **"DT-tuned1"** : cette version est identique à la version tuned1 testée en validation externe.
- **"DT-tuned2"** : cette version est une version "tunée" tout comme la version "DT-tuned1". Les valeurs des hyperparamètres qu'elle teste et parmi lesquelles elle va rechercher la meilleure combinaison possible sont : $\{None, \frac{3n}{2}\}$ pour max_depth, $\{'best','random'\}$ pour splitter, $\{None, \frac{3n}{4}\}$ pour max_features, $\{None, 3*n\}$ pour max_leaf_nodes, $\{1, 2\}$ pour min_samples_leaf et $\{2, 3\}$ pour min_samples_split. La valeur de random_state pour cette version est fixée à 1 pour garantir le caractère déterministe de l'arbre ainsi créé.
- **"RF-default"** : il s'agit du modèle par défaut de la forêt aléatoire de la version 0.20.4 de ScikitLearn. Pour rappel, celui-ci ne contient que 10 arbres.
- **"RF-n100"** : il s'agit du modèle de la forêt aléatoire qui contient 100 arbres de régression.

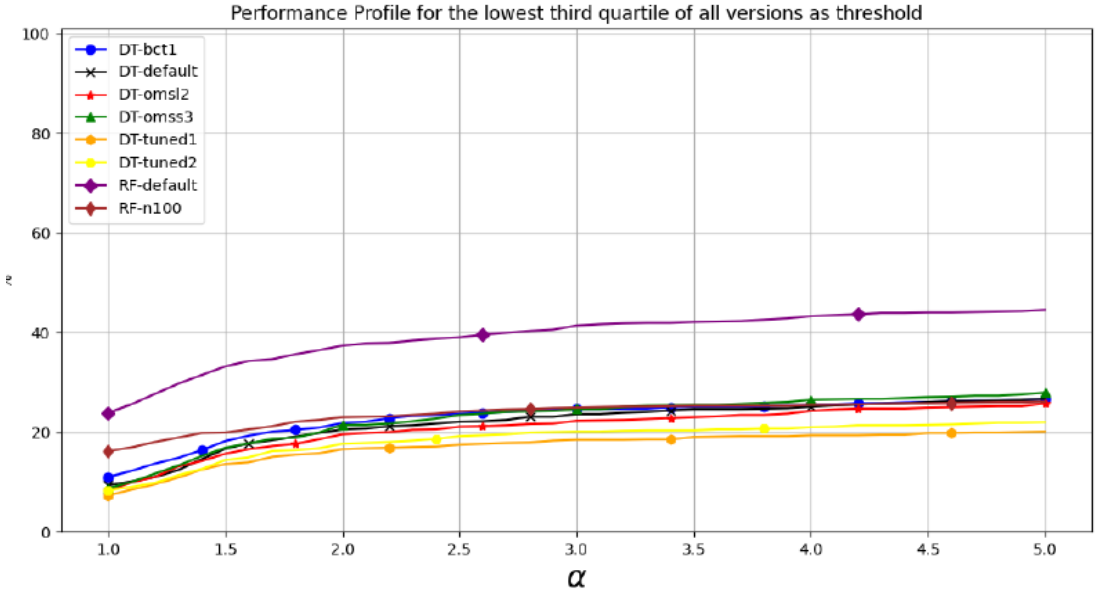

FIGURE 8.7 – Profil de performance de toutes nos versions testées en fonction de la proportion α d'itérations pour le 3ème quartile le plus petit de toutes les versions comme seuil.

Passons dès à présent à l'analyse du graphique de la FIGURE 8.7 qui reprend les profils de performance de ces huit versions que nous avons testées en optimisation. Une des versions semble déjà se démarquer par rapport aux sept autres. Il s'agit de la version RF-default. En effet, cette version est plus efficace et plus robuste que les autres car le pourcentage de problèmes résolus pour cette version est clairement supérieur aux autres et ce pour n'importe quelle valeur de α. Les autres versions ont tendance à avoir un comportement sensiblement similaire entre elles. Le graphique de la FIGURE 8.7 tient compte du troisième plus petit quartile de toutes les versions comme seuil alors que le graphique de la FIGURE 8.8 représente le profil de performance pour le troisième plus grand quartile utilisé comme seuil de "réussite" de résolution des problèmes tests, mais ces deux graphiques disposent de courbes assez similaires et peuvent s'interpréter de la même manière.

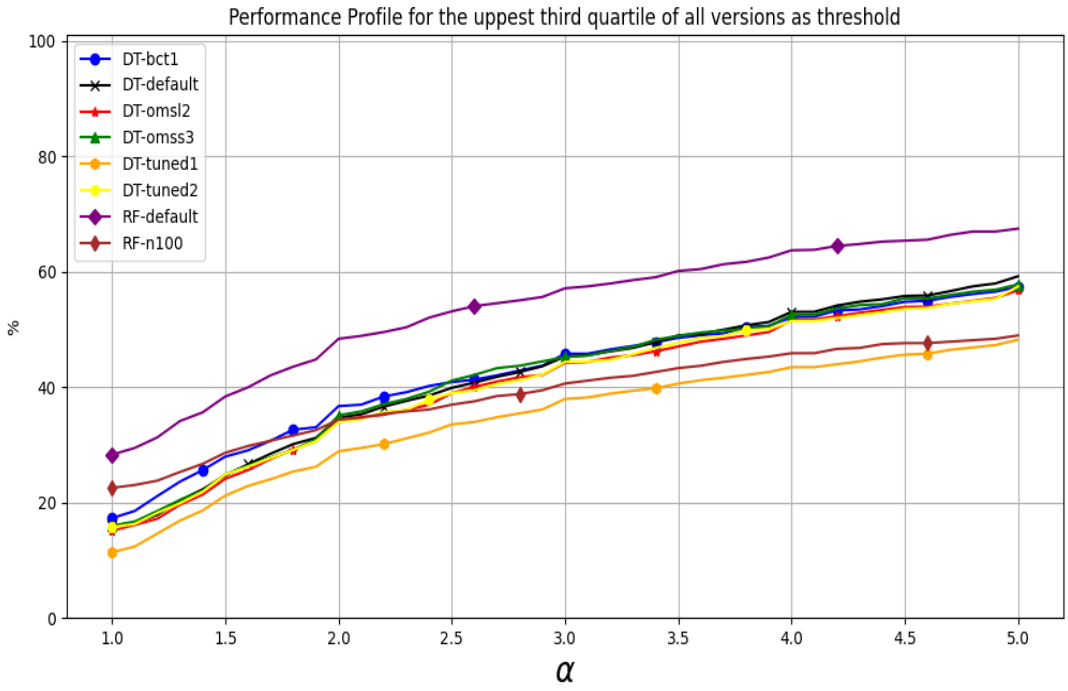

FIGURE 8.8 – Profil de performance de toutes nos versions testées en fonction de la proportion α d'itérations pour le 3ème quartile le plus grand de toutes les versions comme seuil de réussite.

8.2.2 Meilleures versions parmi les versions testées

Parmi tous les graphes illustrant la convergence de la médiane pour la fonction objectif dont nous disposons, nous allons montrer ici ceux qui nous ont permis de sélectionner les trois meilleures versions des modèles d'arbre de régression et de forêt aléatoire d'arbres de régression parmi les versions testées en optimisation. Les autres graphiques de convergence

de la médiane pour la fonction objectif qui ne sont pas montrés ici se trouvent en annexe A. Sur base des illustrations des FIGURES 8.9, 8.10 et 8.11 nous pouvons noter que les trois meilleures versions que nous allons retenir pour la suite de ce mémoire parmi celles que nous avons testées en optimisation sont les modèles RF-default, RF-n100, DT-bct1. Sur ces graphes de convergence de la médiane pour la fonction objectif, nous observons que ce sont ces trois versions qui se démarquent par rapport aux autres. En particulier, sur les graphes des FIGURES 8.9 et 8.11, ce sont les courbes associées aux modèles RF-default et RF-n100 qui atteignent le plus rapidement le seuil large pour les problèmes test PressureVessel-RIDC et CarSideImpact-RC. Sur le graphe de la FIGURE 8.10, ce sont les versions RF-default et DT-bct1 qui se démarquent pour les mêmes raisons dans le cadre du problème test Piston-RC.

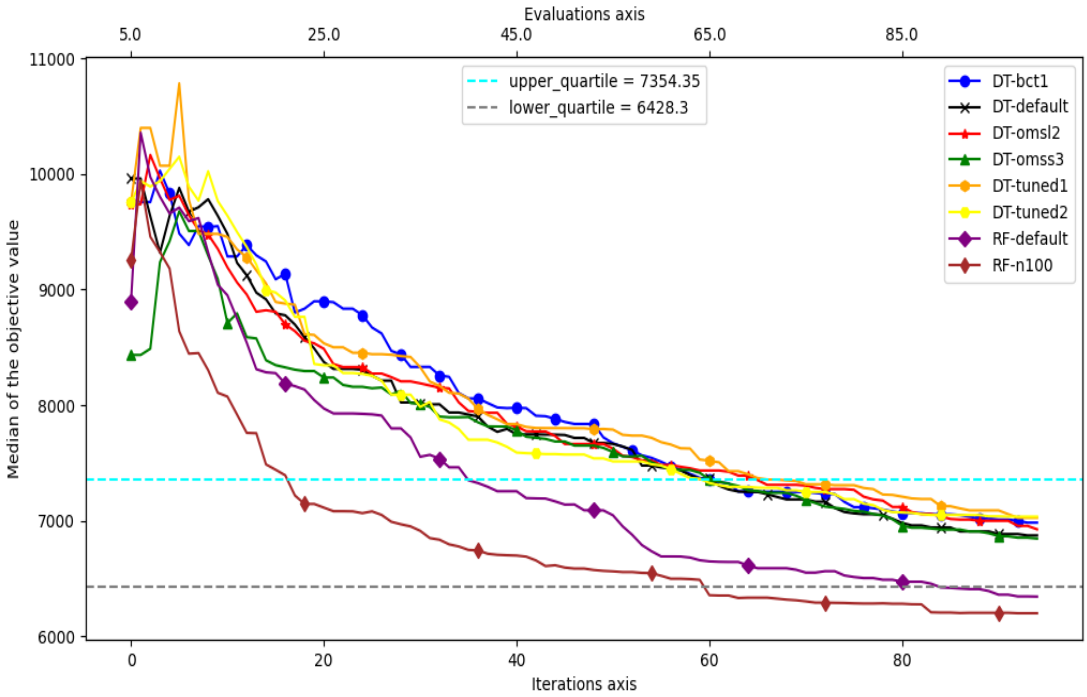

FIGURE 8.9 – Convergence de la médiane pour la fonction objectif pour le problème test PressureVessel-RIDC.

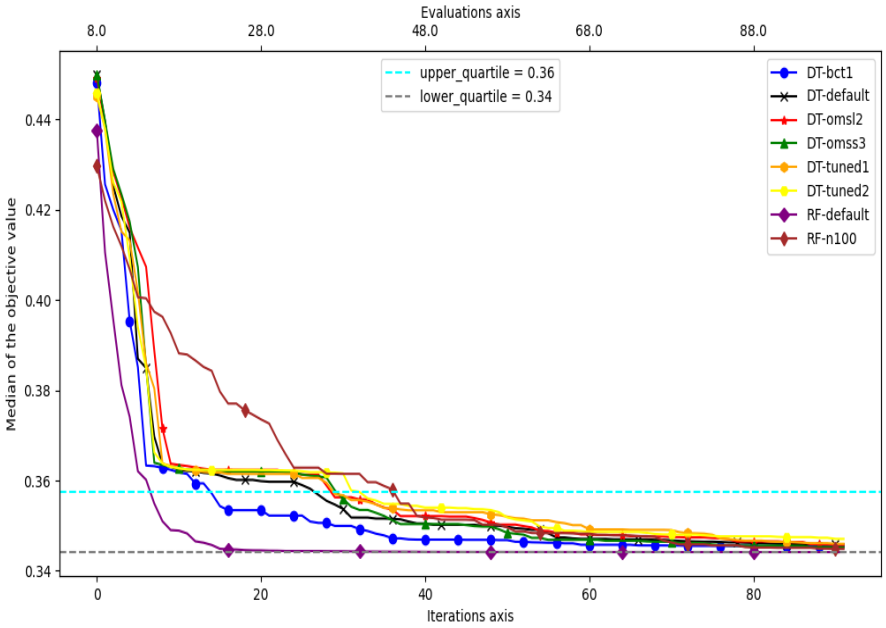

Figure 8.10 – Convergence de la médiane pour la fonction objectif pour le problème test Piston-RC.

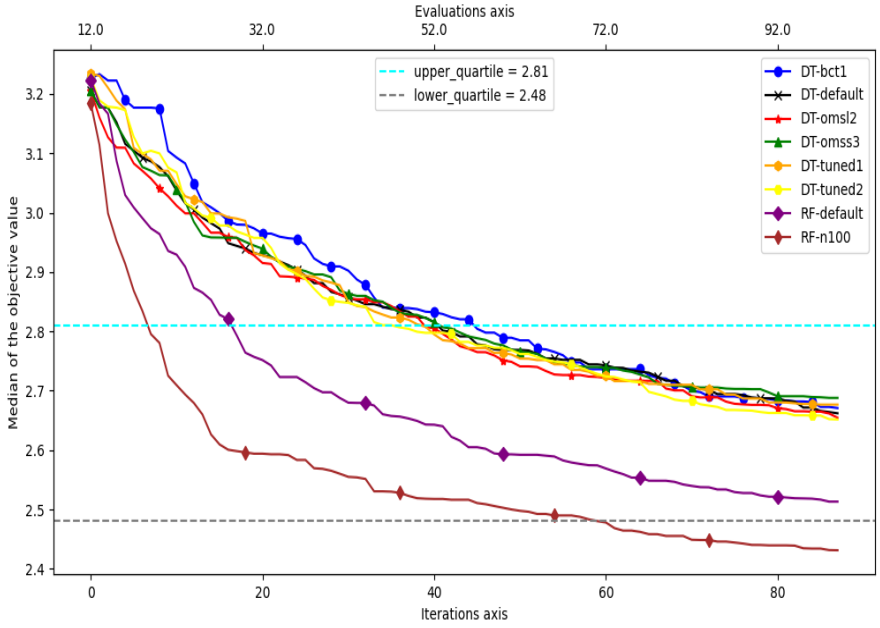

Figure 8.11 – Convergence de la médiane pour la fonction objectif pour le problème test CarSideImpact-RC.

8.2.3 Comparaison des meilleures versions testées avec la version par défaut de Minamo

Les graphiques de cette section montrent clairement que le modèle trbf est plus performant que les trois meilleures versions des modèles d'arbre de régression et de forêt aléatoire que nous avons déterminées. Nous avons sélectionné aux FIGURES 8.12 et 8.13 les profils de performance des ces trois versions accompagnées du modèle trbf afin d'établir cette comparaison. Que le seuil de réussite considéré pour la résolution des problèmes tests soit égal au troisième plus petit quartile de toutes les versions ou bien au plus grand, le pourcentage de problèmes résolus pour la version trbf de Minamo est nettement supérieur pour $\alpha = 1$ ou encore pour $\alpha = 5$. Le modèle trbf est donc dans tous les cas le modèle le plus efficace et le plus robuste.

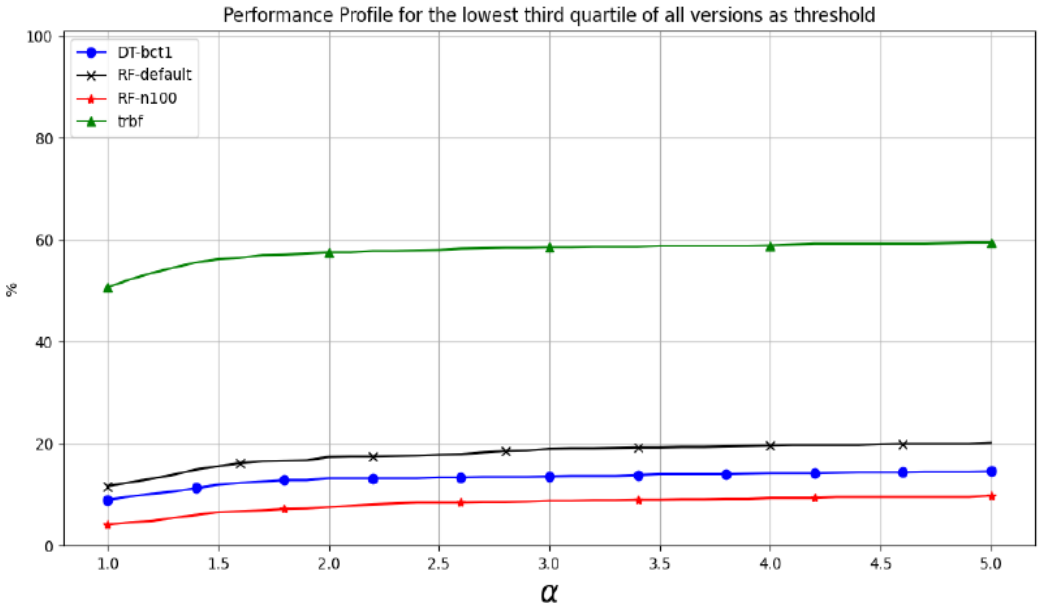

FIGURE 8.12 – Profil de performance en fonction de la proportion α d'itérations pour le 3ème quartile le plus petit de toutes les versions comme seuil et pour les 3 meilleurs versions comparées à trfb.

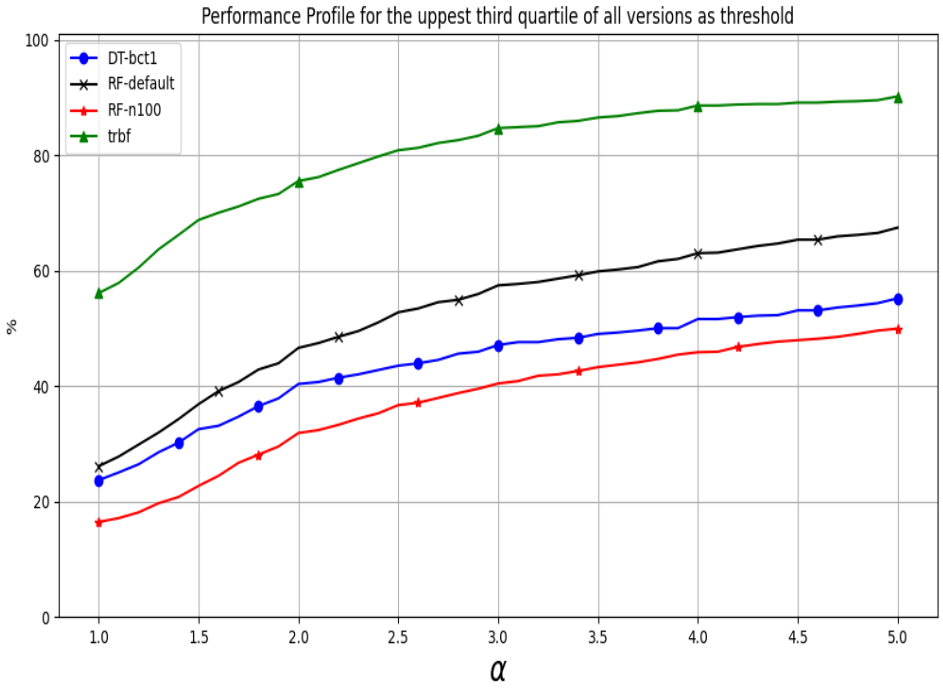

FIGURE 8.13 – Profil de performance en fonction de la proportion α d'itérations pour le 3ème quartile le plus grand de toutes les versions comme seuil et pour les 3 meilleurs versions comparées à trfb.

8.2.4 Problèmes tests particuliers

Dans cette section, nous allons présenter des problèmes tests pour lesquels nous avons pu déterminer des versions qui donnaient de meilleurs résultats que le modèle par défaut de Minamo. Les graphes des FIGURES 8.14 et 8.15 expriment la convergence de la médiane pour la fonction objectif pour les problèmes tests DSMM-RC et Piston-RC, tandis que les FIGURES 8.16 et 8.17 illustrent la moyenne du nombre de contraintes non respectées pour les problèmes test G9-RIDC et CarSideImpact-RC. Grâce à la FIGURE 8.14, nous pouvons observer que les versions DT-tuned1, DT-tuned2 et DT-omss3 sont particulièrement performantes et devancent le modèle trbf. Sur le graphe de la FIGURES 8.15, il s'agit de la version RF-default qui est sensiblement plus performante que la version trbf. Nous remarquons en particulier que le graphe de la FIGURE 8.15 est identique à celui de la FIGURE 8.10 mais nous y avons ajouté la courbe associée au modèle trbf de Minamo. Enfin, en ce qui concerne les graphes reprenant la moyenne par itération du nombre de contraintes non respectées, la majorité des versions testées performent mieux que le trbf excepté la version RF-n100 pour le graphe de la FIGURE 8.16.

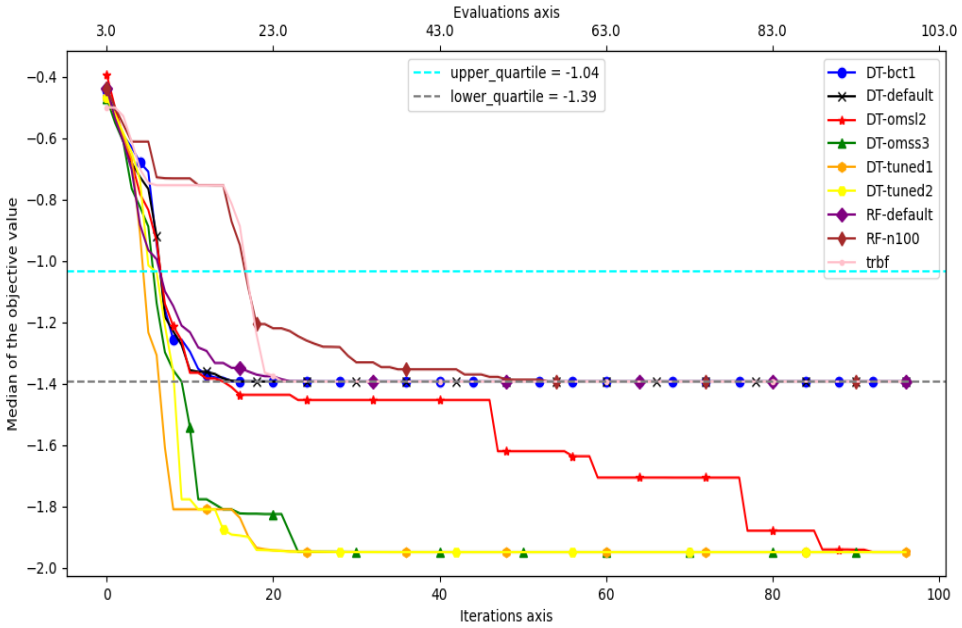

FIGURE 8.14 – Convergence de la médiane pour la fonction objectif pour le problème test DSMM-RC.

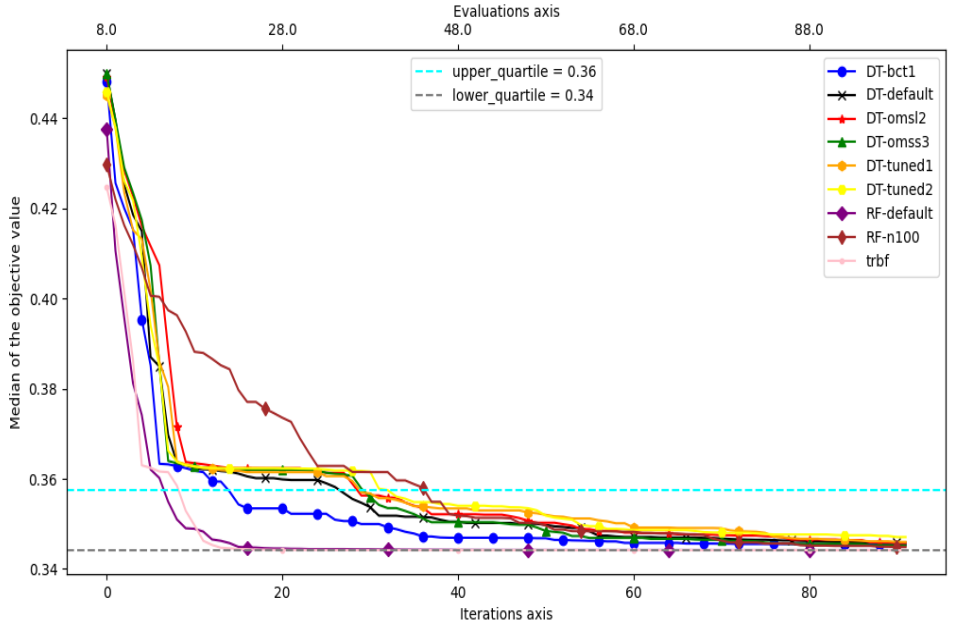

FIGURE 8.15 – Convergence de la médiane pour la fonction objectif pour le problème test Piston-RC.

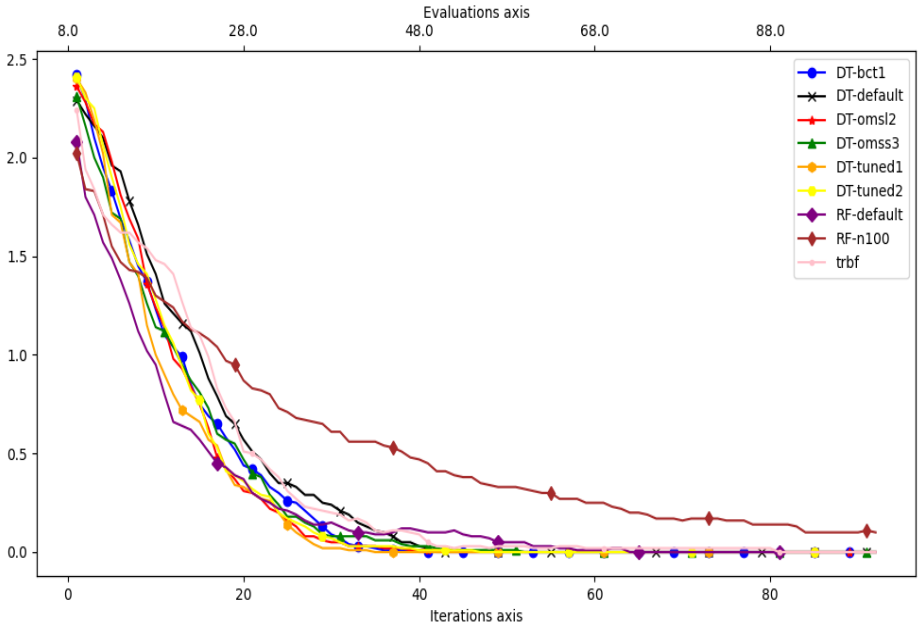

FIGURE 8.16 – Moyenne du nombre de contraintes non respectées pour le problème test G9.

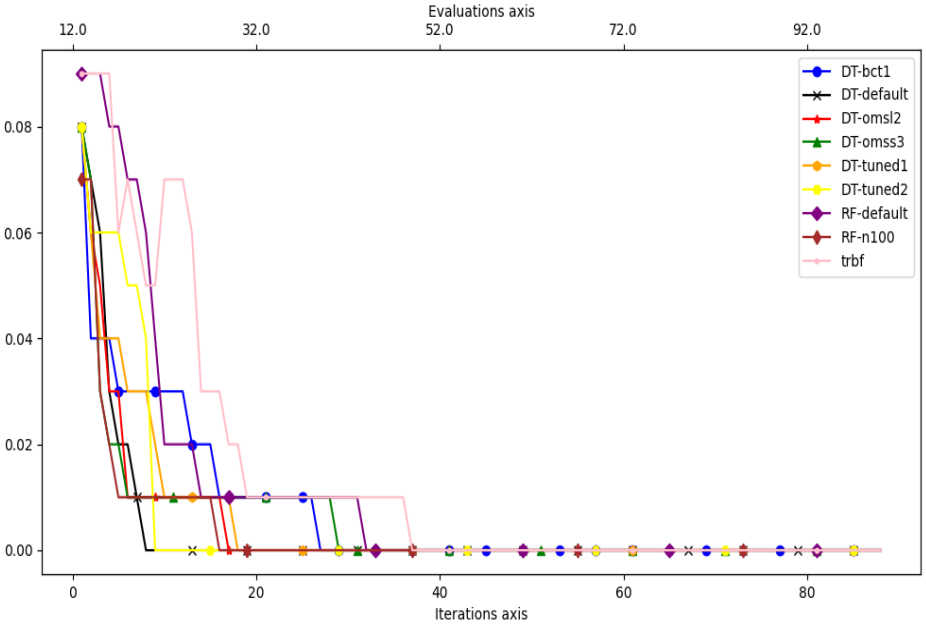

FIGURE 8.17 – Moyenne du nombre de contraintes non respectées pour le problème test CarSideImpact-RC.

Conclusions et perspectives

Après une mise en contexte du mémoire rappelant le thème et les objectifs de celui-ci, nous avons développé le problème d'optimisation avec ou sans contraintes et faisant intervenir des fonctions coûteuses que nous considérons, ainsi que l'algorithme de résolution utilisé par Minamo. La technique d'optimisation assistée par méta-modèles utilisée par Minamo a ensuite été détaillée, ainsi que le concept de plan d'expériences qui lui est associée. Nous avons également évoqué le méta-modèle par défaut de Minamo, nommé trbf. Pour rappel, ce modèle est le fruit de la combinaison linéaire de plusieurs fonctions à base radiale et c'est en regard de ce modèle que nous avons comparé les techniques d'arbre de régression et de forêt aléatoire d'arbres de régression. Ensuite, nous avons développé ces deux techniques issues de la littérature en expliquant le concept qui se cache derrière celles-ci et en illustrant, sur base d'un exemple simple, les différentes étapes de la construction de ces modèles. En particulier, en ce qui concerne l'arbre de régression, nous avons également expliqué le cas plus complexe de la construction d'un arbre disposant de prédicteurs multiples. Nous avons introduit ce cas car il représente celui utilisé lors de nos tests. En ce qui concerne la forêt aléatoire d'arbres de régression, nous avons évoqué quelques-uns de ses avantages et inconvénients. Les caractéristiques propres à chacun des hyperparamètres associés aux modèles étudiés ont été expliquées et nous avons motivé les raisons des choix que nous avons effectués quant à la sélection ou non de certains hyperparamètres dans notre étude. Pour rappel, les hyperparamètres que nous avons sélectionnés pour la phase d'expérimentation sont "criterion", "max_depth", "min_samples_split", "min_samples_leaf","max_features","max_leaf_nodes", ainsi que "splitter", propre au modèle d'arbre de régression et "n_estimators", propre au modèle de forêt aléatoire d'arbres de régression.

Les expérimentations que nous avons menées l'ont été sur un ensemble de problèmes tests avec et sans contraintes que nous avons détaillés. Pour rappel, les problèmes tests sans contraintes que nous avons considérés regroupent les problèmes Ackley_10, Rosenbrock_10, Borehole-RC1/2/3, Branin-RC1, DSMM-RC, FCATMIX-RC1,FCATMIX-RC2, FMIX-RC, FPOLY-RC, FQUAD-RC, FTRIG-RC, Goldstein-RC, OTL-RC, Piston-RC et Roustant-RC. Les problèmes tests disposant de contraintes sont donnés, quant à eux, par : G7, G10, PressureVessel-RIDC, G9-RIDC, CarSideImpact-RC, Spring-RC et WB4-RC. Nous avons également décrit les outils fournis dans les documents générés de manière automatique via des scripts de Cenaero à partir des tests récupérés sur Zenobe et qui nous ont permis d'analyser les résultats de nos expériences, en séparant les outils dé-

veloppés par Cenaero dans le cadre de la validation externe de ceux qui ont été développés pour analyser les résultats de tests en optimisation. En particulier, pour chacun des deux aspects étudiés, nous distinguons les outils de comparaison par problème test de ceux qui représentent la perfomance globale de nos modèles. Notons que les graphiques de profils de performance générés lors des tests d'optimisation ont été introduits par Dolan et Moré [18]. Enfin, nous présentons l'ensemble des résultats que nous avons obtenus lors de nos expériences. Pour rappel, les tests en validation externe ont été réalisés en considérant un DOE d'entrainement contenant $3 * n$ individus, tandis que le DOE de validation en contenait $1000 * n$. La liste des problèmes tests étudiés dans ce cadre est donnée par : Borehole-RC1/2/3, Branin-RC1, CarSideImpact-RC, DSMM-RC, FCATMIX- RC1/2, FMIX-RC, FPOLY-RC, FQUAD-RC, FTRIG-RC, Goldstein-RC, OTL-RC, Piston-RC, Roustant-RC, Spring-RC, WB4-RC, Ackley_10, G7, G10 et Rosenbrock_10. Nous avons testé en validation externe et pour chacun des hyperparamètres étudiés les valeurs extrêmes qui peuvent être attribuées à chacun d'eux. De manière globale, les résultats montrent que ce sont les valeurs proches des valeurs par défaut de chaque hyperparamètre qui rendent les meilleurs résultats pour le modèle d'arbre de régression. Les résultats concernant le modèle de forêt aléatroire d'arbres de régression montrent qu'au plus la forêt aléatoire contient d'arbres de régression au plus la qualité du modèle en termes de validation externe est grande. Cependant, il ne faut pas négliger le temps d'entrainement d'une forêt aléatoire contenant 1000 arbres qui est de trois jours, par exemple. Nous retenons également de nos tests en validation externe que le modèle de forêt aléatoire RF-n100 montre de meilleurs performances que le modèle DT-default.

Nous avons ensuite décidé de présenter les différents résultats que nous avons obtenus en optimisation en explicitant tout d'abord les versions testées. Pour rappel, le DOE de départ utilisé en optimisation a pour caractéristique de contenir $n + 1$ individus, où n est la dimension du problème test étudié. Le nombre d'itérations de la boucle SBO était fixé, pour chaque problème test étudié, à $100 - (n + 1)$. La population initiale de l'algorithme génétique est 100, tout comme son nombre de générations. Les problèmes d'optimisation que nous avons testés sont : Borehole-RC3, CarSideImpact-RC, DSMM-RC, FCATMIX-RC1, FCATMIX-RC2, G7, G9, G10, Piston-RC, PressureVessel-RC et WB4-RC. Ils sont moins nombreux que ceux testés en validation externe car la fin de vie de Zenobe ne nous a pas permis d'en tester plus. Les modèles que nous avons testés en optimisation sont : DT-bct1, DT-default, DT-omsl2, DT-omss3, DT-tuned1, DT-tuned2, RF-default et RF-n100. Sur base des graphes de profil de performance affichés pour ces huit versions ainsi que des graphes de convergence de la médiane de la valeur de l'objectif, nous avons sélectionné les trois meilleures versions. Celles-ci sont données par RF-default, RF-n100 et DT-bct1. Nous avons ensuite comparé nos meilleures versions avec la version par défaut de Minamo, le trbf. Cette version par défaut de Minamo dispose de performances bien meilleures que celles des versions RF-default, RF-n100 et DT-bct1. La version trbf s'avère plus efficace et plus robuste que nos trois meilleures versions. Enfin, nous avons clôturé notre travail en présentant plusieurs graphes de convergence de la médiane de la valeur de l'objectif et de moyenne du nombre de contraintes non respectées pour lesquels certaines de nos versions testées donnaient de meilleurs résultats que la version par défaut de Minamo.

Nous observons également lors de nos tests en optimisation que les versions liées au modèle de forêt aléatoire d'arbres de régression sont de manière générale plus performantes que les versions liées au modèle d'arbre de régression.

Les perspectives que nous pouvons envisager pour ce mémoire seraient, d'une part, de réaliser une étude qualité/coût d'entrainement des modèles testés et, d'autre part, d'investiguer d'autres méthodes prometteuses de la littérature, telles que les forêts extrêmement aléatoires détaillées dans [19] par Pierre Geurts. Cependant, il est à noter que pour la première perspective, les résultats en termes de temps d'entrainement de nos modèles ne doivent pas être comparés à ceux de la version trbf de Minamo car le temps d'entrainement de celui-ci sera difficile à égaler, étant donné que ce modèle est implémenté dans Minamo et que son temps de compilation est dès lors fortement réduit.

Bibliographie

[1] *Cenaero, Simulation technologies for Aeronautics*, `http://www.cenaero.be`, consulté le 12 février 2022.

[2] J. Starmer, *Regression Trees, clearly explained*, `https://www.youtube.com/watch?v=g9c66TUylZ4`, consulté le 01 avril 2022.

[3] A. Sartenaer, C. Beauthier, C. Sainvitu, *Proposition de mémoire en Master Mathématiques*, 2020.

[4] Y. Derlet, *Réglage de la taille de la population de l'algorithme génétique au sein de l'optimisation assistée par modèles de substitution présent dans le logiciel Minamo*, mémoire, Namur, UNamur, 2019.

[5] *Cenaero, Supercalculateur zenobe*, `https://tier1.cenaero.be/fr/zenobe`, consulté le 20 février 2021.

[6] *Cenaero, Minamo 3.2.0 Theoretical Manual*, 2020.

[7] Equipe Minamo, *Workshop optimization - Session 1 Theoretical Introduction to Optimization*, présentation, *Cenaero*, 2021.

[8] L. Breiman, *Random Forests. Machine Learning*, vol. 45, no. 1, pages 5–32, 2001.

[9] *GDC Coder, Decision Tree Regressor explained in depth*, `https://gdcoder.com/decision-tree-regressor-explained-in-depth/`, consulté le 12 avril 2021.

[10] Wikipedia, *Forêt d'arbres décisionnels*, `https://en.wikipedia.org/wiki/Random_forest`, consulté le 12 avril 2021.

[11] R. Couronné, P. Probst, A. L. Boulesteix, *Random forest versus logistic regression : a large-scale benchmark experiment*, Technical Report 380, 2020.

[12] *GDC Coder, Random Forest Regressor explained in depth*, `https://gdcoder.com/random-forest-regressor-explained-in-depth/`, consulté le 16 avril 2021.

[13] L. Breiman, J. Friedman, R. Olshen, & C. Stone. *Classification And Regression Trees.* Chapman et Hall, 1984.

[14] C. Malot-Tuleau, *Méthodes CART Introduction à la sélection de variables*, `https://math.unice.fr/~malot/CART.pdf`, consulté le 15 juillet 2021.

[15] *Cenaero*, *Preparing to run on Zenobe*, rapport, page 4/29, 2015, `https://tier1.cenaero.be/en/system/files/filedepot/3/PRACE_T1FWB-NR-005-03_0.pdf`, consulté le 10 mars 2023.

[16] CECI, *Clusters at CECI*, `https://www.ceci-hpc.be/clusters.html`, consulté le 18 avril 2023.

[17] Scikit-learn, *Scikit-learn user guide Release 0.20.4*, 2019, `https://scikit-learn.org/0.20//_downloads/scikit-learn-docs.pdf`, consulté le 15 novembre 2022.

[18] E.D. Dolan & J.J. Moré, *Benchmarking Optimization Software with Performance Profiles*, Mathematical Programming, Vol. 91, No. 2, pp. 201-213, 2002.

[19] P. Geurts, D. Ernst & L. Wehenkel, *Extremely randomized trees*. Mach Learn 63, 3–42 (2006).

[20] X. Yao, Y. Liu and G. Lin, "Evolutionary programming made faster," in *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 2, pp. 82-102, July 1999, doi : 10.1109/4235.771163.

[21] Q. Zhang, P. Chien, Q. Liu, Li Xu & Yili Hong (2020) : *Mixed-input Gaussian process emulators for computer experiments with a large number of categorical levels*, Journal of Quality Technology, DOI : 10.1080/00224065.2020.1778431.

[22] J.Pelamatti, *Mixed-variable Bayesian optimization - Application to aerospace system design*, thèse, Lille, Université de Lille, 2020.

[23] M. Munoz Zuniga & D. Sinoquet, *Global optimization for mixed categorical-continuous variables based on Gaussian process models with a randomized categorical space exploration step*, 2020, INFOR : Information Systems and Operational Research, DOI : 10.1080/03155986.2020.1730677.

[24] M. Halstrup, *Black-box optimization of mixed discrete-continuous optimization problems*, thèse, Dortmund, Université de Dortmund, 2016.

[25] Y. Zhang, S. Tao, W. Chen & D. W. Apley, *A Latent Variable Approach to Gaussian Process Modeling with Qualitative and Quantitative Factors*, Technometrics, 62 :3, 2020, 291-302, DOI : 10.1080/00401706.2019.1638834.

[26] O. Roustant, E. Padonou, Y. Deville, A. Clément, G. Perrin, et al., *Group kernels for Gaussian process metamodels with categorical inputs*. 2019.

[27] R. G. Regis, *Constrained Optimization by Radial Basis Function Interpolation for High-Dimensional Expensive Black-Box Problems with Infeasible Initial Points*, Engineering Optimization, Vol. 46, No. 2, pp. 218-243, 2014.

[28] Z. Michalewicz, M. Schoenauer, *Evolutionary algorithms for constrained parameter optimization problems*, Evolutionary Computation, 4(1) :1–32, 1996.

[29] A. H. Gandomi, X.-S. Yang, A. H. Alavi, *Mixed variable structural optimization using Firefly Algorithm*, Computers and Structures 89 (2001) 2325-2336

[30] L. C. Cagnina, S. C. Esquivel, *Solving Engineering Optimization Problems with the Simple Constrained Particle Swarm Optimizer*, Informatica 32 (2008) 319-326.

[31] R.G. Regis, C. A. Shoemaker, *A quasi-multistart framework for global optimization of expensive functions using response surface models*. Journal of Global Optimization, 2012, DOI : 10.1007/s10898-012-9940- 1, in press.

[32] J. Sacks, W. J. Welch, T. J. Mitchell, & H. P. Wynn, *Design and Analysis of Computer Experiments*, Statistical Science, 4,1989, 409–423.

[33] A. Díaz-Manríquez, G. Toscano-Pulido & W. Gómez-Flores, *On the selection of surrogate models in evolutionary optimization algorithms*, IEEE Congress of Evolutionary Computation (CEC), New Orleans, LA, USA, 2011, pp. 2155-2162, doi : 10.1109/CEC.2011.5949881.

[34] S. Pool Marquez, *Adaptation de la « One Fifth Success Rule » pour le réglage de la taille de la population dans l'algorithme génétique du logiciel MINAMO*, mémoire, Namur, UNamur, 2021.

Annexes

Annexe A

Meilleures versions testées en optimisation

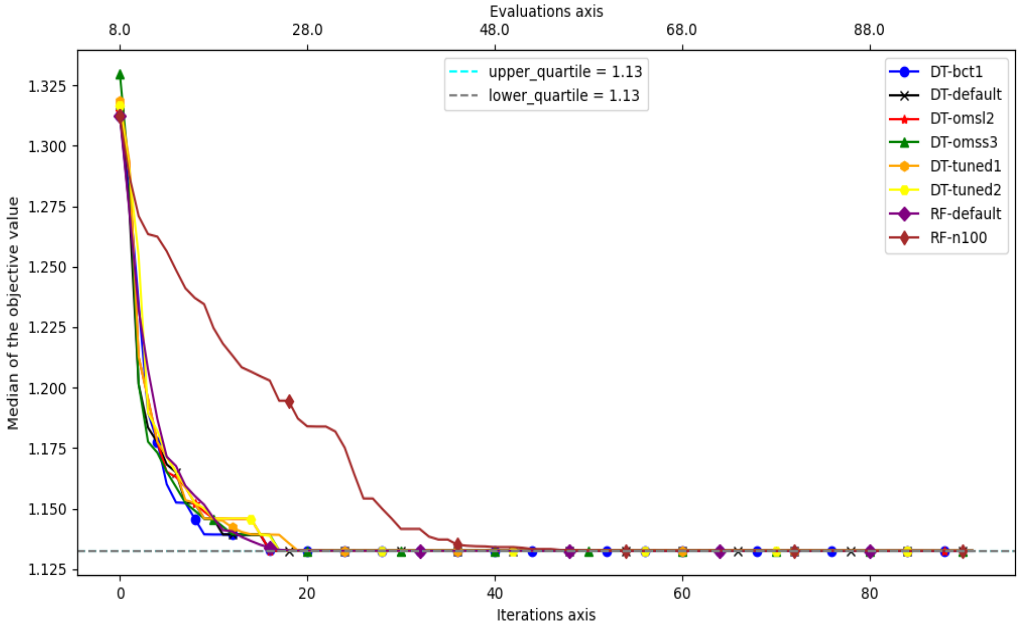

Figure A.1 – Convergence de la médiane pour la fonction objectif pour le problème test Borehole-RC3.

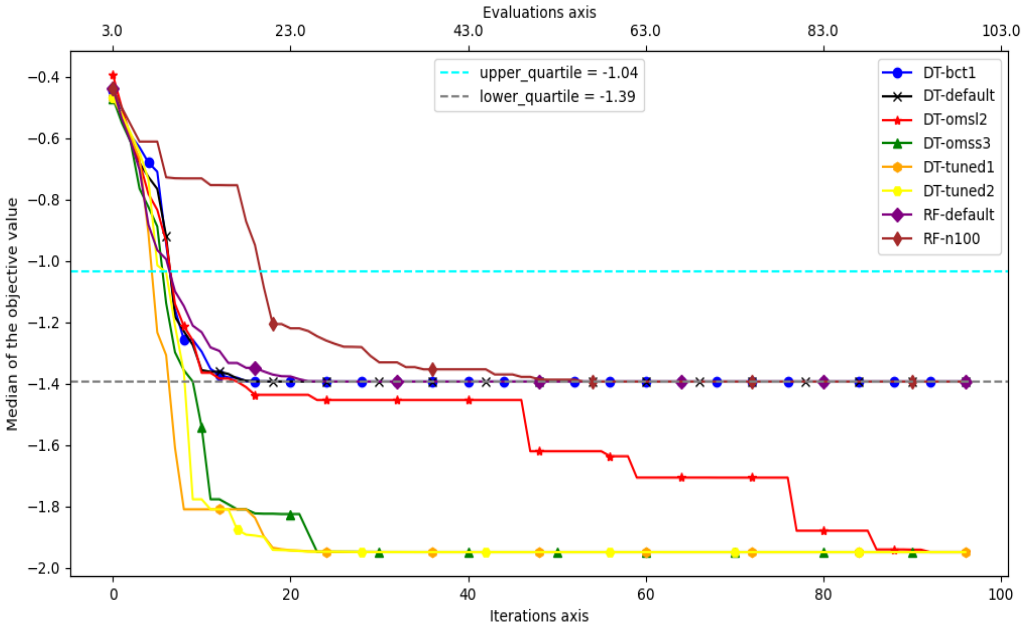

Figure A.2 – Convergence de la médiane pour la fonction objectif pour le problème test DSMM-RC.

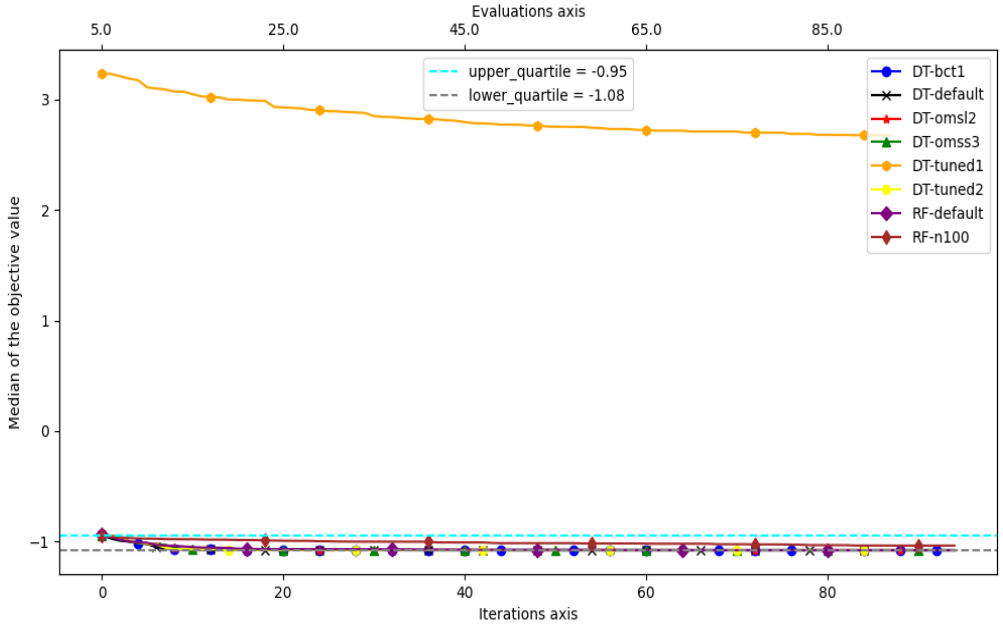

Figure A.3 – Convergence de la médiane pour la fonction objectif pour le problème test FCATMIX-RC1.

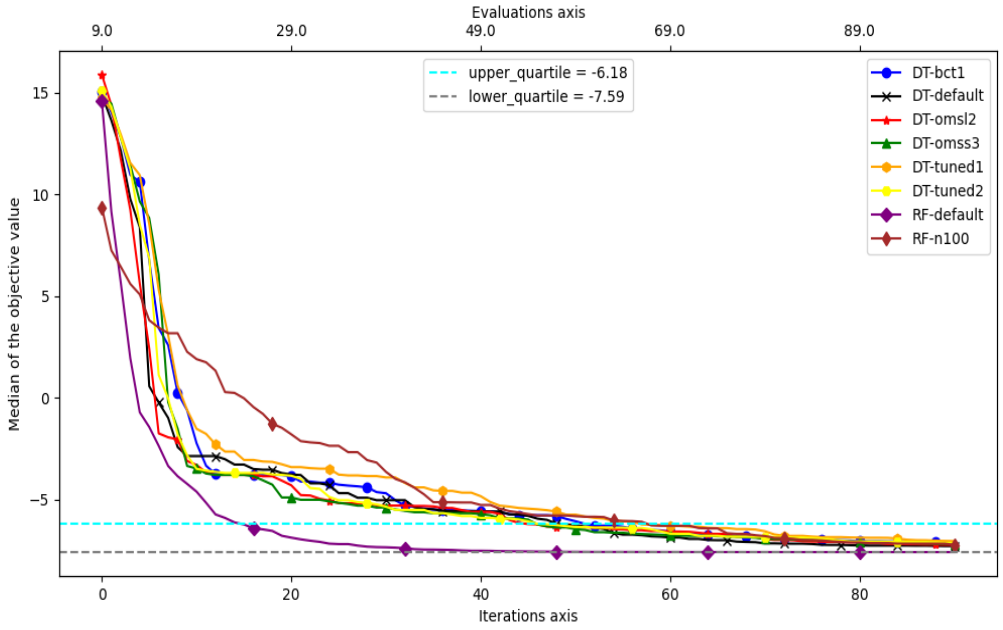

Figure A.4 – Convergence de la médiane pour la fonction objectif pour le problème test FCATMIC-RC2.

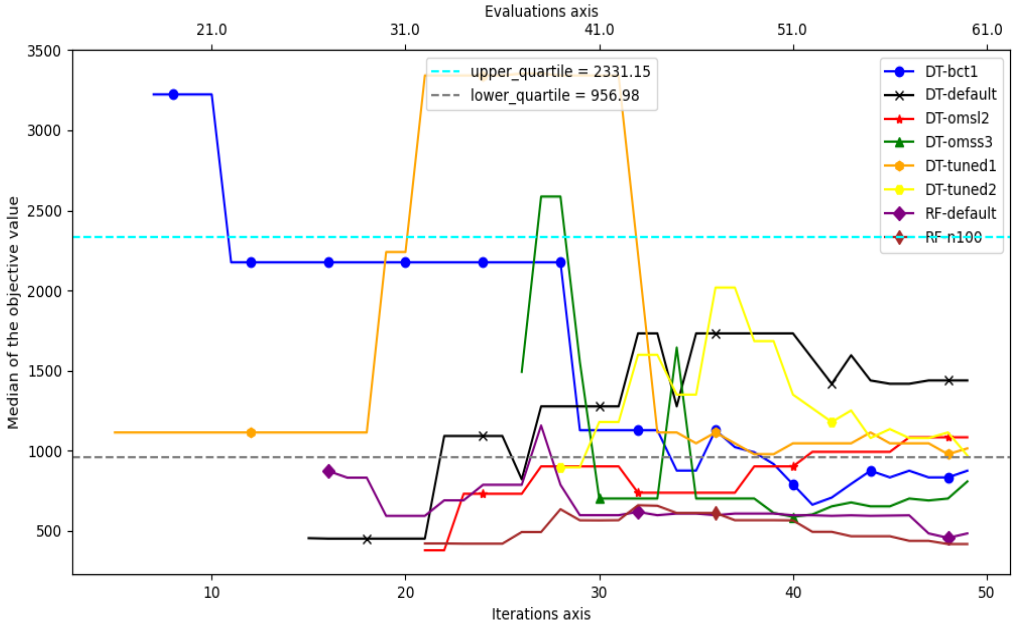

Figure A.5 – Convergence de la médiane pour la fonction objectif pour le problème test G7.

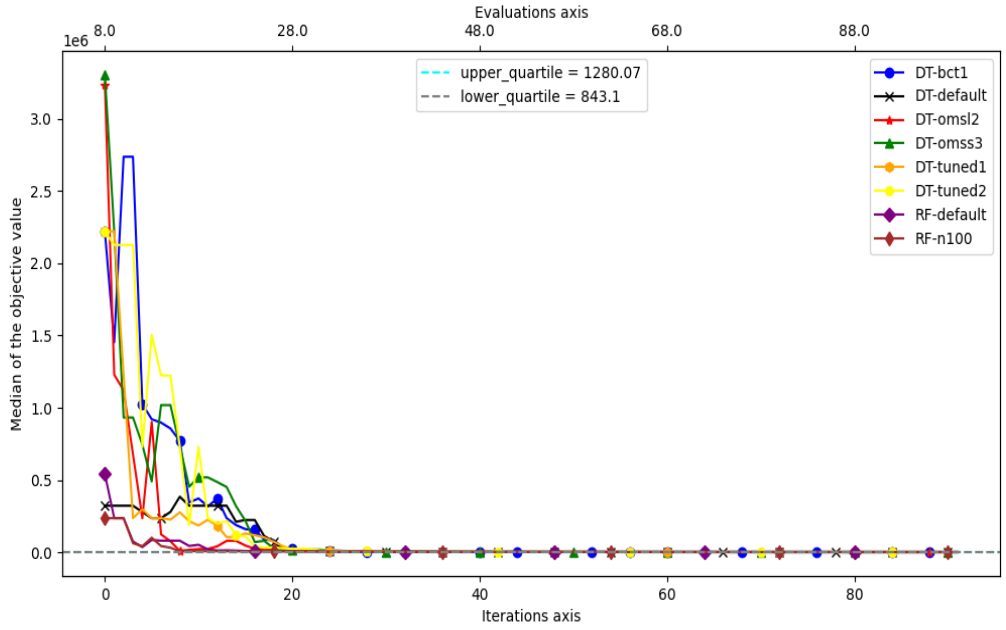

Figure A.6 – Convergence de la médiane pour la fonction objectif pour le problème test G9.

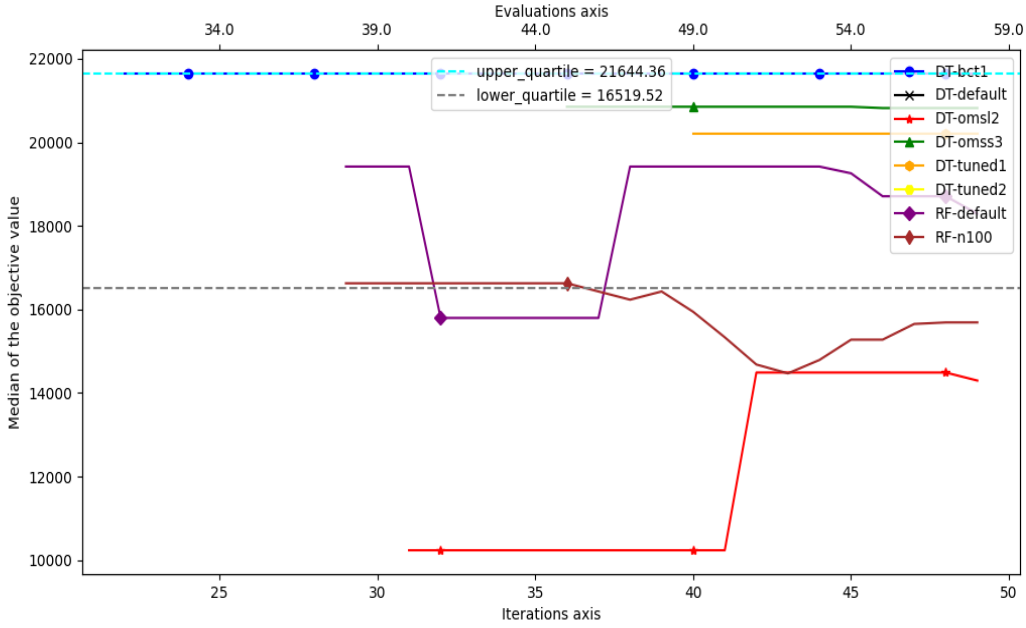

Figure A.7 – Convergence de la médiane pour la fonction objectif pour le problème test G10.

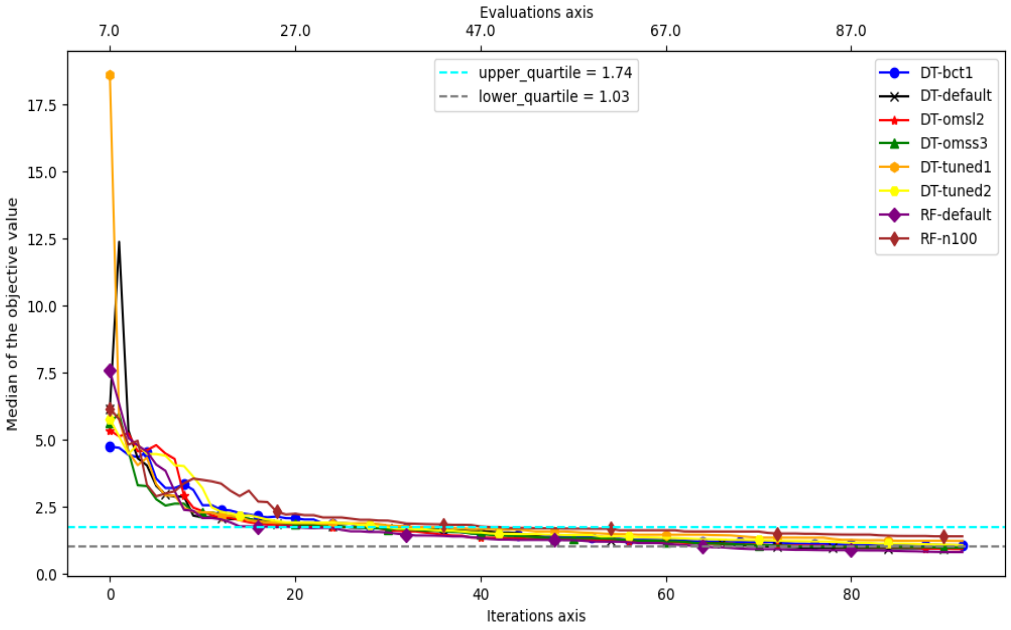

Figure A.8 – Convergence de la médiane pour la fonction objectif pour le problème test WB4-RC.