REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE MOULOUD MAMMERI, TIZI-OUZOU

FACULTÉ DES SCIENCES

DÉPARTEMENT DE MATHÉMATIQUES

MEMOIRE DE MASTER

en

RECHERCHE OPÉRATIONNELLE

OPTION : MÉTHODES ET MODÈLES DE DÉCISION

Sujet :

Méthode Hybride en Optimisation Globale

Présenté par :

M^r **CHIKHI Abderzak** et M^r **BOUAFIA Nazim**

Soutenu le 03/07/2013 Devant le jury d'examen composé de :

M^r	MORSLI M.	Professeur	U.M.M.T.O	Président
M^r	OUANES M.	Maître de conférence A	U.M.M.T.O	Rapporteur
M^{me}	LOUADJ K.	Maître de conférence B	U.M.M.T.O	Examinatrice
M^r	KASDI K.	Maître assistant A	U.M.M.T.O	Examinateur

⁂ *Remerciements* ⁂

D'abord nous remercions
le bon dieu de nous avoir donné santé, courage et foie
pour réaliser ce travail avec volonté.

Tous nos vifs remerciements ; nos profondes reconnaissances
s'adressent à notre promoteur Mr **OUANES Mohand** à qui nous tenons
témoigner notre sincère gratitude de nous avoir confié ce sujet intéressant pour
les conseils et son aide pour accomplir notre travail.

Que Monsieur le président et Messieurs les membres de jury
trouvent ici l'expression de notre gratitude et respect de nous avoir
fait l'honneur d'examiner et juger notre travail.

Nous remercions nos très chers parents pour leurs
soutiens et leurs encouragements durant
notre cycle d'étude.

Enfin, merci à tous ceux qui ont
contribué de près ou de loin à ce mémoire, du point de vue
scientifique ou administratif.

⁂Nazim, Abderzak ⁂

⁕ *Dédicaces* ⁕

Je remercie le Dieu de m'avoir donné
la force d'accomplir ce travail pour aller plus loin.

Je dédie ce travail :

A mes parents, mes grand parents,
mes tantes pour leurs encouragements tout au long
de mes études.

A mes frères : " Lounes, Djamel, Henni."
A mes soeurs : " Zohra et mon adorée Katia."
A l'unique femme avec laquelle je partagerai ma vie : Sara

A mes très chers amis :
Sofiane, Djamel, Djamel (Medecin), Salim, Joseph, Rachid, tonton Rahim, Boussad,
Rabah, Kaka,Hacene..."

A toute la famille CHIKHI de tagarssift
(Kamel,Athmane,...)

A tous les gens que j'aime dont je n'ai pas
cité les noms.

A mon binôme Nazim, ainsi que toute sa famille..

⁕ ***Abderzak*** ⁕

⁂ *Dédicaces* ⁂

Je dédie ce modeste travail :

A mes très chers parents,
qui m'ont éduqué, soutenue le long de mon cursus, ils m'ont
donné tout pour arriver à ce point et je les souhaite une belle et longue vie.

Merci mon père pour tout ce que t'as fait pour moi
et mes frères. Merci ma mère pour votre soutien et encouragement moral surtout dans les moments difficiles que j'ai passé à UMMTO. Vous êtes ma source de poursuivre mes études et ce modeste travail.

A mes très chers frères Ouaffi et Oualid pour
leur soutien et orientation.

A mon binôme Abderzak, ainsi que toute
sa famille.

A tous mes amis et camarades de la promotion
et même des autres promotions.

A tous les profs à qui j'ai assisté durant mes années à l'université.

A tous ceux qu'ont participé
de près ou de loin.

⁂ *Nazim* ⁂

Table des matières

Table des figures

Liste des tableaux

Introduction générale

Nous faisons tous de l'optimisation. Dans notre vie quotidienne, nous cherchons à optimiser notre temps de travail, nos espaces de rangement, ou encore le trajet que nous aurons à parcourir pour nous rendre quelque part,...etc. Nous recherchons tous une meilleure solution aux problèmes qui jalonnent notre existence. De manière générale, l'optimisation va donc consister à trouver cette meilleure solution.

Comme nous le rappelle l'adage populaire selon lequel : "les mathématiques permettent de mettre le monde en équation", il peut être tracé un parallèle entre l'optimisation quotidienne et celle, plus technique que l'on retrouve en science. En mathématique, la meilleure solution se recherche au sein d'un domaine initial. Cette solution est souvent soumise à des contraintes qui correspondent à des obligations ou des souhaits à respecter. Le critère permettant de distinguer qu'une solution est la meilleure s'appelle la fonction objectif. L'optimisation mathématique va consister à rechercher dans le domaine initial une solution qui maximise ou minimise une fonction objectif tout en respectant des contraintes.
Pour un domaine continu, on distingue classiquement deux types d'optimisation :

- L'optimisation locale recherche une solution qui est la meilleure localement, c'est-à-dire que dans son voisinage aucune solution n'est meilleure qu'elle. Cette solution est appelée un optimum local.
- L'optimisation globale recherche quant à elle la meilleure solution du domaine en entier, c'est-à-dire que dans tout le domaine il n'existe aucune solution qui lui soit meilleure tout en respectant les contraintes. Cette solution est appelée l'optimum global.

L'intérêt de l'optimisation globale par rapport à l'optimisation locale est patent. Elle garantit en effet que personne ne peut avoir une solution meilleure que celle trouvée. Or, pour une entreprise, cette information a son importance, car la différence entre la solution globale et une solution locale est bien souvent significative. Mais l'intérêt n'est pas que compétitif. Dans de nombreux problèmes, l'optimum global est la seule solution mathématique correspondant à une réalité physique. C'est ce qu'illustre par exemple la recherche de la quantité de chaque élément présent dans un mélange chimique à l'équilibre.

De nos jours, afin de résoudre des problèmes d'optimisation globale avec contraintes, de nombreuses stratégies algorithmiques s'avèrent disponibles. Pour guider le choix de la meilleure stratégie à utiliser, il est nécessaire de regarder : (i) la taille du problème, (ii) les propriétés de la fonction objectif et des contraintes (continuité, différentiabilité, linéarité, convexité,...), (iii) ainsi que le temps disponible pour résoudre le problème.

Dans ce travail nous nous somme intéressés à deux types d'algorithmes de résolution (génétiques et branch-and-bound par intervalle) et aux techniques hybridation que l'on peut développer pour améliorer leurs performances.

Le travail accompli est présenté à travers les différents chapitres. Dans le premier chapitre, nous allons donner un aperçu général sur le problème de l'optimisation non linéaire, ainsi que des rappels sur quelques notions théoriques de base. Ainsi, quelques méthodes d'optimisation globale.

Dans le deuxième chapitre, nous présenterons, les concepts qui sont à la base du calcul par intervalles. Les structures fondamentales de compact sur $\mathbb{R}^n$ seront représentées par des intervalles, des pavés et des sous-pavages. Nous verrons, à travers l'arithmétique par intervalles et les fonctions d'inclusion, les outils algébriques permettant d'étendre, les opérations mathématiques sur les réels, aux calculs sur des intervalles de valeurs réelles.

Dans le troisième chapitre, nous commençons par donner le principe des algorithmes génétiques en détaillant les différents paramètres utiles pour l'implémentation de l'approche.

Dans le quatrième chapitre nous présentons une approche de recherche en deux phases. Elle utilise une hybridation de l'approche proposée dans le deuxième avec une approche proposé dans le troisième chapitre, pour touver une solution plus éfficace.

Le dernier chapitre est consacré à une application informatique des méthodes cités dans les chapitres précédent, ainsi une modèlisation d'un problème du gestion de trafic aérien a été faite.

Chapitre 1

Optimisation Globale

1.1 Introduction

Résoudre un problème d'optimisation revient à localiser l'optimum global du critère en présence ou en absence de contraintes. Ces dernières années, avec l'évolution de l'informatique, une attention particulière a été accordée à l'optimisation globale dont l'objectif est de développer des algorithmes sophistiqués et rapides permettant de localiser l'optimum global d'une fonction mathématique et d'échapper ainsi au problème rencontré avec les méthodes d'optimisation classique qui sont souvent piégées dans un minimum local.

Les méthodes de gradient classiques et de recherche aléatoire se comportent bien, sur des fonctions unimodales simples mais sont inefficaces pour être appliquées aux problèmes difficiles, comme des fonctions non différentiables, multimodales ou bruyantes. Dans ces cas-ci, où les algorithmes d'optimisation classiques ne fournissent pas des résultats fiables. l'algorithme reproduit le comportement d'une goutte de pluie qui tombe sur une surface imperméable : elle tombe en un point, puis suit la ligne de plus grande pente en s'arrêtant sur le premier creux qu'elle rencontre.

Ce premier creux correspond à un minimum de la fonction mais n'a aucune raison de correspondre au minimum global de celle-ci. Pour poursuivre la métaphore météorologique, ceci correspond aux lacs qui se remplissent jusqu'à déborder ; l'eau poursuit ainsi son chemin de minimum local de l'altitude en minimum local jusqu'à la mer qui réalise le minimum global. L'algorithme de gradient s'arrête au premier lac. En d'autres termes, le résultat final de l'application de l'algorithme de gradient dépend très fortement du point initial choisi.

Dans le cadre des applications pratique, les minima locaux ne présentent que peu d'intérêt, seul le ou les minima globaux comptent. Il convient donc d'essayer de se débarrasser de ce blocage dans un minimum local. La façon de faire est d'utiliser une méthode d'optimisation globale.

Dans cette partie nous allons donner un aperçu général sur le problème de l'optimisation non linéaire, ainsi des rappels de notions théoriques de base. Ainsi, quelques méthodes d'optimisation globale. Dans l'étude présentée dans ce mémoire, on considère les différents résultats et théories relatifs à la formulation " minimiser ".

1.2 Formulation mathématique d'un problème d'optimisation non linéaire

Considérons une fonction scalaire de plusieurs variables $x_1, x_2, ..., x_n$, appelées variables de décision, notée $f(x)$, et appelée fonction objectif ou critère. Le vecteur de variables de décision, noté $x = (x_1, x_2, ..., x_n)^T$. Nous présentons un problème d'optimisation mathématiquement sous la forme suivante :

$$\begin{cases} \min(f(x)) & & \\ s.c: & & \\ g_i(x) \leq 0 & i = 1, ..., n & (1) \\ h_j(x) = 0 & j = 1, ..., m & (2) \\ x \in \mathcal{S} \subseteq \mathbb{R}^n & & (3) \end{cases} \tag{1.1}$$

- f : représente la fonction objectif à optimiser (minimiser ou maximiser)
- (1),(2),(3) : représentent les contraintes du problème.
- $\mathcal{S}$: représente le domaine ou l'ensemble des solutions réalisables, appelé aussi l'espace de recherche.

L'objectif de l'optimisation globale est de rechercher parmi les $x \in \mathcal{S}$, un x^* particulier qui est appelé minimum absolu ou global (n'est pas obligatoirement unique), et vérifiant les contraintes (1), (2), noté x^*, tel que :

$$f(x^*) \leq f(x), \quad \forall x \in \mathcal{S} \tag{1.2}$$

On peut passer d'une formulation minimiser à une formulation maximiser et vice versa : minimiser $f(x)$ revient à maximiser $-f(x)$, c'est-à-dire :

$$\min_x f(x) = -\max_x(-f(x)) \tag{1.3}$$

Lorsque (1) et (2) sont absentes, le problème (1.1) se ramèner à un problème d'optimisation sans contraintes. Même avec la présence de (1) et (2) ou l'une des deux contraintes le problème (1.1) peut se ramener à un problème sans contraintes et ceci en faisant appel aux méthodes de pénalités.

Définition 1.1. On appelle solution réalisable du problème (1.1) tout vecteur x vérifiant les contraintes (1),(2),(3), C'est-à- dire tel que :

$$\begin{cases} g_i(x) \leq 0 & i = 1, ..., n \\ h_j(x) = 0 & j = 1, ..., m \\ x \in \mathcal{S} \subseteq \mathbb{R}^n \end{cases} \tag{1.4}$$

1.2.1 Position du problème

Dans la suite du travail présenté, on considère seulement un cas particulier de contraintes du type inégalités, il s'agit des contraintes " de box " données comme suit :

$$x_i^{min} \leq x_i \leq x_i^{max} \tag{1.5}$$

Où x_i^{min} et x_i^{max} sont des composantes de deux vecteurs x^{min} et x^{max}, de dimension n, données. Ces deux vecteurs définissent un domaine admissible $\mathcal{S}$ hyper-rectangulaire.

Nous abordons les problèmes d'optimisation globale définie comme suit :

$$\begin{cases} \min f(x) \\ x \in \mathcal{S} \end{cases} \tag{1.6}$$

Où $f : \mathbb{R}^n \longrightarrow \mathbb{R}$ est continue différentiable, non linéaire et non convexe, et l'ensemble compact $\mathcal{S} \subseteq \mathbb{R}^n$ est un box de dimension n. Le but est de rechercher parmi les $x \in \mathcal{S}$, un (ou plusieurs) x^* particulier qui est appelé minimum absolu ou global.

1.3 Rappel de notions théoriques de base

1.3.1 Signe d'une matrice

On définit une matrice carrée $A(n \times n)$ d'éléments réels par un tableau :

$$A = \begin{pmatrix} a_{11} & ... & a_{1n} \\ . & . & . \\ . & . & . \\ . & . & . \\ a_{n1} & ... & a_{nn} \end{pmatrix}$$

Définition 1.2. Soit A une matrice carrée symétrique $(n \times n)$.

1. On dit que A est définie positive si : $x^T.A.x > 0$, $x \neq 0_{\mathbb{R}^n}$, $x \in \mathbb{R}^n$
2. On dit que A est semi-définie positive si : $x^T.A.x \geq 0$, $x \neq 0_{\mathbb{R}^n}$, $x \in \mathbb{R}^n$
3. On dit que A est définie négative si : $x^T.A.x < 0$, $x \neq 0_{\mathbb{R}^n}$, $x \in \mathbb{R}^n$
4. On dit que A est semi-définie négative si : $x^T.A.x \leq 0$, $x \neq 0_{\mathbb{R}^n}$, $x \in \mathbb{R}^n$

Définition 1.3. Soit A une matrice carrée symétrique $(n \times n)$

1. A est définie positive $\Leftrightarrow \quad det(A_k) > 0\ , \qquad \forall k = 1, ..., n$
2. A est définie négative $\Leftrightarrow \quad (-1)^k.det(A_k) > 0\ , \qquad \forall k = 1, ..., n$

où

$$A_k = \begin{pmatrix} a_{11} & ... & a_{1k} \\ . & . & . \\ . & . & . \\ . & . & . \\ a_{k1} & ... & a_{kk} \end{pmatrix}$$

Définition 1.4. (Critère qui utilise les valeurs propre)
Soit A une matrice carrée symétrique $(n \times n)$, β_i les valeurs propres de A.

1. Si $\beta_i > 0, \quad \forall i \Rightarrow A$ est définie positive.
2. Si $\beta_i \geq 0, \quad \forall i \Rightarrow A$ est semi-définie positive.
3. Si $\beta_i < 0, \quad \forall i \Rightarrow A$ est définie négative.
4. Si $\beta_i \leq 0, \quad \forall i \Rightarrow A$ est semi-définie négative.

1.3.2 Continuité

Soit $A \subseteq \mathbb{R}^n$, une fonction $f : A \longrightarrow \mathbb{R}$ est dite continue en $x^0 \in A$ si :

$$\forall \varepsilon > 0, \quad \exists \delta > 0, \quad tq : \forall x \in A, \| x - x^0 \| < \delta \Rightarrow \| f(x) - f(x^0) | < \varepsilon \tag{1.7}$$

on écrit : $\lim_{x \longrightarrow x^0} f(x) = f(x^0)$

Remarque 1.1. on dit que f est continue sur A si elle est continue en tout point de A.

1.3.3 Différentiabilité

– **Dérivées partielles :**
Soit $f : A \longrightarrow \mathbb{R}$ une fonction définie par un voisinage de $x^0 \in A$.
Si la $lim_{h \longrightarrow 0} \frac{f(x_1, x_2, ..., x_i + h, x_{i+1}, ..., x_n) - f(x^0)}{h}$ existe et est finie alors elle est appelée la i^{me} dérivée partielle, elle est notée $\frac{\partial f(x^0)}{\partial x_i}$ ou $\nabla_i f(x)$.

– **Différentiabilité :**
Soit $f : A \longrightarrow \mathbb{R}$ une fonction définie par un voisinage de $x \in A$, on dit que est différentiable en x^0, si :

$$lim_{x \longrightarrow x^0} \frac{f(x) - f(x^0) - \nabla f(x^0)(x - x^0)}{\| x - x^0 \|} = 0 \tag{1.8}$$

Remarque 1.2. f est différentiable sur A si elle est différentiable en chaque point de A.

Remarque 1.3. f est dite continûment différentiable sur A si les dérivées partielles de f sont continuent sur A, et on dit que f est de classe $\mathcal{C}^1$.

1.3.4 Gradient

Si $f \in \mathcal{C}^1(A, \mathbb{R})$, on appelle gradient de f en x le vecteur de $\mathbb{R}^n$:

$$\nabla f(x) = (\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, ..., \frac{\partial f(x)}{\partial x_n}) \tag{1.9}$$

avec $x = (x_1, ..., x_n)$

1.3.5 Hessienne

Soit f de classe $\mathcal{C}^2$, on définit la hessienne de f par :

$$Hf(x) = J(\nabla f) = \begin{pmatrix} \frac{\partial^2 f}{\partial x_1^2} & \frac{\partial^2 f}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f}{\partial x_2 \partial x_1} & \frac{\partial^2 f}{\partial x_2^2} & \cdots & \frac{\partial^2 f}{\partial x_2 \partial x_n} \\ . & . & \cdots & . \\ . & . & \cdots & . \\ . & . & \cdots & . \\ \frac{\partial^2 f}{\partial x_n \partial x_1} & \frac{\partial^2 f}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n^2} \end{pmatrix}$$

Remarque 1.4. $Hf(x)$ est une matrice carrée $(n \times n)$ symétrique.

1.3.6 Eléments d'analyse convexe

Pour étudier les problèmes d'optimisation, il est nécessaire de recourir à des outils spécifiques dont l'étude est basée sur l'analyse convexe. Nous allons ici rappeler brièvement les définitions de quelques notions importantes.

1.3.7 Ensemble convexe

Un ensemble $\mathcal{S} \subseteq \mathbb{R}^n$ est dit convexe ssi :

$$\lambda x + (1 + \lambda)y \in \mathcal{S}, \quad \forall x, y \in \mathcal{S}, \quad \forall \lambda \in [0, 1]$$

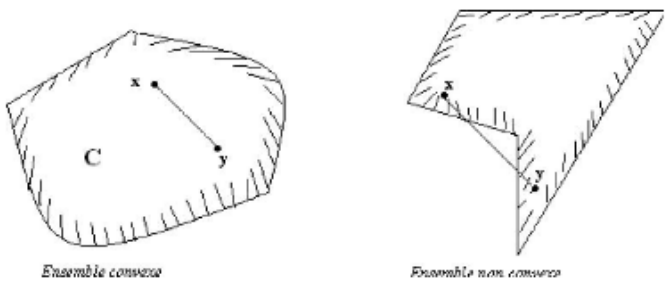


FIG. 1.1 – Ensemble convexe et non convexe

Remarque 1.5.

- Les ensembles convexes de $\mathbb{R}$ sont les intervalles.
- L'intersection d'ensembles convexes est un ensemble convexe.
- $\mathbb{R}^n$ est convexe

1.3.8 Fonction convexe

Etant donné une fonction : $f : \mathcal{S} \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$, nous avons trois type de définitions essentiels :

Définition 1.5. On dit qu'une fonction, $f : \mathcal{S} \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$, définie sur un ensemble convexe $S \subseteq \mathbb{R}^n$ est convexe ssi :

$$\forall x, y \in \mathcal{S}, \quad \forall \lambda \in [0,1], \quad f(\lambda x + (1+\lambda)y \leq \lambda f(x) + (1+\lambda)f(y)$$

Lorsque $n = 1$ cette définition s'interprète bien géométriquement : le graphe de la fonction et toujours en dessous du segment reliant les points $(x, f(x))$ et $(y, f(y))$ (Fig.1.2).

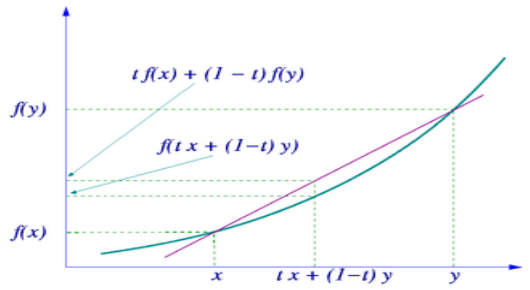


FIG. 1.2 – Fonction convexe

Exemple 1.1. $n = 2$

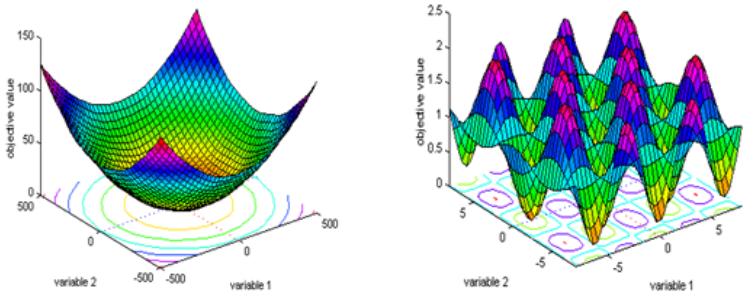


FIG. 1.3 – fonction (sphère) convexe et fonction Griewangk non convexe

Lorsqu'une fonction convexe est dérivable, la caractérisation suivante sera utile.

Proposition 1.1. *On dit qu'une fonction, $f \in \mathcal{C}^1(\mathcal{S} \subseteq \mathbb{R}^n, \mathbb{R})$, définie sur un ensemble convexe $\mathcal{S}$ est convexe ssi :*

$$f(y) \geq f(x) + \nabla f^T(x)(y - x), \quad \forall x, y \in \mathcal{S}$$

Proposition 1.2. *On dit qu'une fonction, $f \in \mathcal{C}^2(S \subseteq \mathbb{R}^n, \mathbb{R})$, définie sur un ensemble convexe $\mathcal{S}$ est convexe ssi :*

$$Hf(x) \text{ est semi-définie positive.}$$

Remarque 1.6. – Une fonction f est concave si $(-f)$ est convexe.

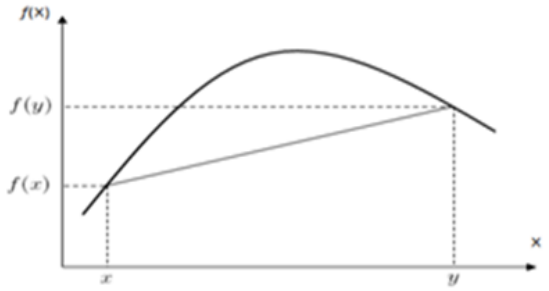


FIG. 1.4 – Fonction concave.

- Une fonction f est strictement convexe si les inégalités ci-dessus est stricte et $(x \neq y)$.
- $Hf(x)$ est définie positive $\forall x \in \mathbb{R}^n \Rightarrow f$ strictement convexe (la réciproquement n'est pas nécessairement vraie).

1.3.9 Fonction Unimodale et Multimodale

Soit $f : [a, b] \longrightarrow \mathbb{R}$ on dit que f est unimodale sur $[a, b]$ si f possède un minimum $x^* \in [a, b]$ tel que :

$$\forall x, y, \quad x < y : \begin{cases} si \;\; y \leq x^* \;\; alors \;\; f(x) > f(y) \\ si \;\; x^* \leq x \;\; alors \;\; f(x) < f(y) \end{cases}$$

Une fonction unimodale sur [a,b] possède un minimum unique sur [a,b]. En générale les fonctions unimodales sont des fonctions qui n'ont qu'un seul optimum (minimum). Comme exemples de fonctions unimodales nous citons la fonction (sphère) (voir Fig.1.3).

Les fonctions admettant plusieurs minima (locaux ou globaux) sont appelées multimodales, on peut citer les fonctions sinusoïdales,fonction Griewangk, et les fonctions non convexes (voir Fig.1.3).

1.3.10 Minimum d'une fonction

Avant de définir la notion de minimum local nous rappelons d'abord la notion de voisinage. On définit un voisinage de x^*, $\mathcal{V}(x^*)$ de taille par :

$$\mathcal{V}(x^*) = \{x \in \mathcal{S} \ \mid \ \| x - x^* \| < \varepsilon\}$$

1.3.11 Conditions suffisantes d'existence d'un point minimum

La première question est celle de l'existence du point de minimum global de la fonction f. Il existe principalement deux théorèmes qui permettent de répondre à cette question. Le premier affirme l'existence d'un point de minimum lorsque l'ensemble des contraintes est fermé borné. Le second est son équivalent pour les ensembles de contraintes fermés mais non bornés.

Théorème 1.1. *(Weierstrass)*
Soit $\mathcal{S}$ un compact (i.e. un fermé borné) non vide de $\mathbb{R}^n$ et $f : \mathcal{S} \subseteq \mathbb{R}^n \longrightarrow \mathbb{R}$ une application continue sur $\mathcal{S}$. Alors f est bornée et atteint ses bornes. Autrement dit, il existe $x^* \in \mathcal{S}$ point de minimum global de f sur $\mathcal{S}$ i.e. :

$$\forall y \in \mathcal{S}, \quad f(x^*) \leq f(y)$$

Théorème 1.2. $\lim_{\|x\| \longrightarrow +\infty} f(x) = +\infty$ *et f est continue sur $\mathbb{R}^n$, alors le problème (1.6) admet au moins une solution. Il y a unicité si on ajoute une hypothèse de la convexité stricte.*

1.3.12 Minimum local

On dit qu'une solution $x^* \in \mathcal{S}$ est un minimum local du problème (1.6) si est seulement s'il existe un voisinage $\mathcal{V}(x^*)$ de x^* tel que :

$$\forall x \in \mathcal{V}(x^*) : f(x^*) \leq f(x)$$

Le minimum local $x \in \mathcal{S}$ est strict si l'inégalité est strict et $x \neq x^*$.

1.3.13 Minimum global

On dit qu'une solution $x^* \in \mathcal{S}$ est un minimum globale du problème (1.6) si est seulement si :

$$\forall x \in \mathcal{S} \ : f(x^*) \geq f(x)$$

Le minimum global $x \in \mathcal{S}$ est strict si l'inégalité est strict et $x \neq x^*$.

La figure 1.5- illustre sur une fonction à une seule variable, les notions d'optimum global et d'optimum local.

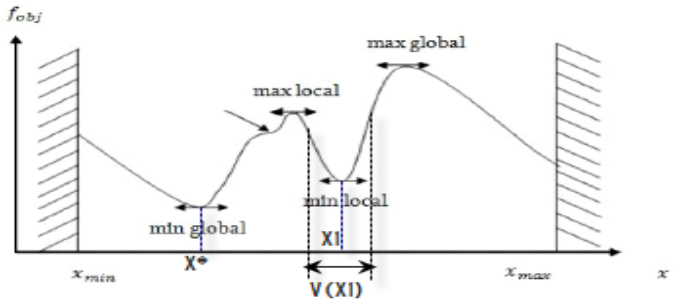


FIG. 1.5 – Optimum local et optimum global

Remarque 1.7. Les maxima locaux et globaux sont définis de manière similaire. Notons que x^* est un maximum local (respectivement global) de la fonction f sur l'ensemble $\mathcal{S}$ si x^* est un minimum local (respectivement global) de $-f$ sur $\mathcal{S}$.

1.4 Conditions d'optimalités

Afin d'analyser ou de résoudre de manière efficace un problème d'optimisation, il est fondamental de pouvoir disposer des conditions d'optimalités.

1.4.1 Conditions nécessaires d'optimalité locale

Etant donné un vecteur x^*, nous souhaiterions être capables de déterminer si ce vecteur est un minimum local ou global de la fonction f. La propriété de différentiabilité continue de f fournit une première manière de caractériser une solution optimale.

Définition 1.6. Une solution x de l'équation $\nabla f(x) = 0_{\mathbb{R}^n}$ est appelé point critique (stationnaire) de la fonction f.

Théorème 1.3. *(Condition nécessaire d'ordre 1)*
Si $f \in \mathcal{C}^1(\mathcal{S} \subseteq \mathbb{R}^n, \mathbb{R})$. Si f admet un minimum local en $x^* \in \mathcal{S}$, alors

$$\nabla f(x^*) = 0_{\mathbb{R}^n}$$

Remarque 1.8.

- Cette condition est nécessaire mais non suffisante.
- En un point critique, une fonction peut atteindre un extremum (soit minimum soit maximum) ou non.

Dans le cas où f est deux fois continuellement différentiable, une autre condition nécessaire est donnée par le théorème suivant. Elle est appelée condition nécessaire du second ordre.

Théorème 1.4. *(Condition nécessaire du 2^{me} ordre)*
Si $f \in \mathcal{C}^1(\mathcal{S} \subseteq \mathbb{R}^n, \mathbb{R})$. Si f admet un minimum local en $x^* \in \mathcal{S}$, alors

1. $\nabla f(x^*) = 0_{\mathbb{R}^n}$;
2. $Hf(x^*)$ *est semi-définie positive.*

1.4.2 Conditions suffisantes d'optimalité locale

Les conditions données précédemment sont nécessaires (si f n'est pas convexe), c'est-à-dire qu'elles doivent être satisfaites pour tout minimum local ; cependant, tout vecteur vérifiant ces conditions n'est pas nécessairement un minimum local. Le théorème suivant établit une condition suffisante pour qu'un vecteur soit un minimum local, si f est deux fois continuellement différentiable.

Théorème 1.5. *(Condition suffisante du 2^{me} ordre)*
Soient $f \in \mathcal{C}^2(\mathcal{S} \subseteq \mathbb{R}^n, \mathbb{R})$, et $x^* \in \mathcal{S}$ vérifiant :

1. $\nabla f(x^*) = 0_{\mathbb{R}^n}$;
2. $Hf(x^*)$ *est définie positive.*

Alors $x^* \in \mathcal{S}$ est un minimum local de f au sens strict.

1.4.3 Conditions suffisantes d'optimalité globale

Théorème 1.6. *Soit $f \in \mathcal{C}^2(\mathcal{S} \subseteq \mathbb{R}^n, \mathbb{R})$ convexe, et $\mathcal{S}$ convexe alors :*

1. *$x^* \in \mathcal{S}$ est un minimum global $\Leftrightarrow \nabla f(x^*) = 0_{\mathbb{R}^n}$;*
2. *Si de plus f est strictement convexe, $x^* \in \mathcal{S}$ est l'unique minimum global de f.*

1.5 Notion de Complexité

1.5.1 Définition d'un algorithme

Un algorithme est un ensemble de règles opératoires dont l'application permet de résoudre un problème énoncé au moyen d'un nombre fini d'opérations.

1.5.2 Propriétés d'un algorithme

- Les entrées : un algorithme prend des valeurs d'entrées à partir d'ensembles définis.
- La sortie : constitue la solution du problème de départ.
- La finitude : l'algorithme doit produire la sortie souhaitée en un nombre fini (mais peut-être très grand) d'étapes, quel que soit l'entrée.
- L'efficacité : chaque étape de l'algorithme doit pouvoir s'exécuter dans un temps fini.
- La généralité : l'algorithme s'applique à tous les problèmes d'une forme désirée.

1.5.3 Complexité des algorithmes

L'analyse de la complexité d'un algorithme consiste à déterminer une fonction $F : N \longrightarrow \mathbb{R}_+$ qui à un paramètre n, dépendant de la donnée soumise à l'algorithme (en général, n est la taille de cette donnée), associe le coût $F(n)$ (exprimé en unités arbitraires de temps ou d'espace) de l'exécution de l'algorithme pour la donnée. Ceci permet en particulier de comparer deux algorithmes traitant le même problème.

1.5.4 Complexité des problèmes

La complexité des problèmes a aboutie à une classification des problèmes en fonction des performances des meilleurs algorithmes connus qui les résolvent. Techniquement, les ordinateurs progressent de jour en jour. Cela ne change rien à la classification précédente, Elle a été conçue indépendamment des caractéristiques techniques des ordinateurs.

1.6 Optimisation convexe

Dans les problèmes d'optimisation, avec ou sans contraintes, la convexité joue un rôle très important. En effet pour la plupart des algorithmes que nous décrirons, la convergence vers un optimum global ne pourra être démontrée qu'avec des hypothèses de convexité.

Le problème (1.6) est convexe si les conditions suivantes sont vérifiées :

1. Le problème est une minimisation ;
2. $f(x)$ est une fonction convexe ;
3. Le domaine $\mathcal{S}$ est un ensemble convexe.

Le problème (1.6) est non convexe si une des conditions est violée.

Théorème 1.7. *$x^* \in \mathcal{S}$ solution optimale locale $\Leftrightarrow$ x^* solution optimale globale.*

Remarque 1.9. Dans les situations convexes, le problème d'optimisation globale peut être abordé par un ensemble de méthodes classiques, telles, par exemple, celles basées sur le gradient, qui ont montré leur efficacité en ce domaine.

1.7 Méthodes d'optimisation globale

Après avoir définie les notions théoriques de base relatives à un problème d'optimisation non linéaire, nous passons aux méthodes d'optimisations globales (voir Fig.1.6).

On peut définir une méthode exacte comme une méthode qui garantit l'obtention de la solution optimale pour un problème d'optimisation. L'utilisation de ces méthodes s'avèrent

particulièrement intéressante, mais elles sont souvent limitées au cas des problème de petite taille. Il existe différents méthodes exactes pour la résolution de tels problèmes.

Tous les problèmes d'optimisation n'ont pas le même degré de difficulté, celui-ci étant surtout lié à la dimension de l'espace de recherche et au paysage de la fonction à optimiser (nombre de minima, dérivabilité, continuité,...etc.), en conséquence, plusieurs algorithmes d'optimisation ont été développés (Fig.1.6), certains étant plus adaptés à des problèmes présentant certaines caractéristiques tandis qu'ils échouent sur des problèmes où d'autre méthodes sont bonnes, et inversement.

Ces algorithmes d'optimisation peuvent être classés en algorithmes d'optimisation locale et algorithmes d'optimisation globale. Alors que les algorithmes de la première classe sont piégés par le premier minimum qu'ils rencontrent ou sont handicapés par la taille de l'espace de recherche, les algorithmes de la seconde classe ne présentent pas ces inconvénients et permettent de trouver une solution proche de l'optimum global.

Dans la littérature, on distingue trois classes pour les méthodes d'optimisation globales : les méthodes exactes (appelées déterministes), les méthodes métaheuristique (appelées encore méthodes stochastiques), et les méthodes hybrides.

La Figure (1.6)- schématise une classification possible des différents types de méthodes d'optimisation.

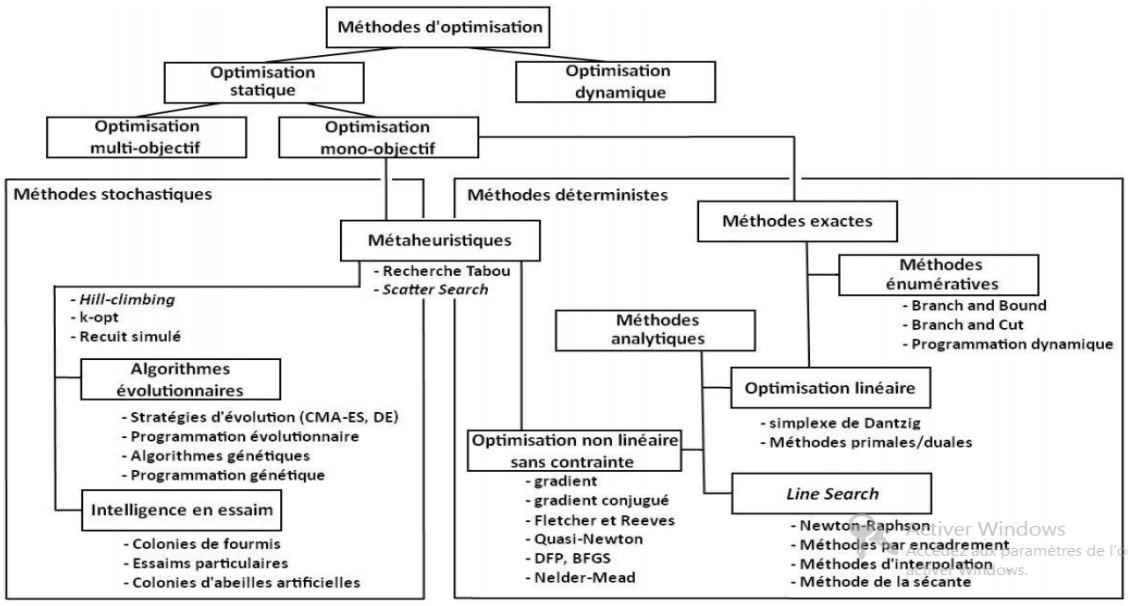


FIG. 1.6 – Une classification des différents types de méthodes d'optimisation.

Pour notre travail, nous considérons les méthodes d'optimisation globale décrites par :

1.7.1 Méthode énumérative

Bien entendu, dans un espace de recherche fini, dont la taille est d'autant plus petite que le calcul de f est numériquement long, la fonction coût peut être évaluée en chaque point de l'espace pour localiser les optima.

Parmi les méthodes énumératives, nous présentons dans le deuxième chapitre, l'algorithme branch-and-bound d'intervalle, qui est une méthode permettant de détecter des régions qui ne contiennent pas l'optimum (Fig.1.7), et ainsi les exclure pour le reste de la recherche.

FIG. 1.7 – Détection de région qui ne contient pas le minimum global.

1.7.2 Méthodes approchées

1.7.3 Les heuristiques

Une heuristique d'optimisation est une méthode approchée se voulant simple, rapide et adapté à un problème donné. Sa capacité à optimiser un problème avec un minimum d'informations est contrebalancée par le fait qu'elle n'offre aucune garantie quant à l'optimalité de la meilleure solution trouvée.

1.7.4 Les métaheuristiques

Apparues dans les années 80, forment un ensemble d'algorithmes, c'est-à-dire, une suite d'instructions effectuant une tâche donnée visant à résoudre une large gamme de problèmes d'optimisation difficile, pour lesquels on ne connaît pas de méthode classique plus efficace.

La figure (1.8) résume le principe des métaheuristiques.

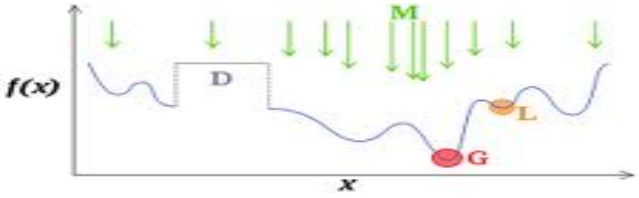


FIG. 1.8 – Principe des Métaheuristiques.

En utilisant certains mécanismes, la métaheuristique (M) tente de trouver l'optimum global (G) d'un problème d'optimisation difficile (D), sans être piégé par les optimums locaux (L).

D'une manière générale, les métaheuristiques s'articulent autour de trois notions : "Exploration ", " Exploitation "et " Mémoire/Apprentissage". L'exploration désigne les processus visant à récolter de l'information sur le problème optimisé. L'exploitation vise à utiliser l'information déjà récoltée pour définir et parcourir les solutions les plus prometteuses. La mémoire est le support de l'apprentissage, qui permet à l'algorithme de ne tenir compte que des zones où l'optimum global est susceptible de se trouver, évitant ainsi les optima locaux (de bonnes solutions, mais qui ne sont pas les meilleures des solutions possibles).

FIG. 1.9 – Les trois phases d'une métaheuristique.

Les métaheuristiques progressent de façon itérative, en alternant des phases d'exploration, d'exploitation et d'apprentissage. L'état de départ est souvent choisi aléatoirement, l'algorithme se déroulant ensuite jusqu'à ce qu'un critère d'arrêt soit atteint.

Parmi les métaheuristiques les plus utilisées, nous distinguons le Recuit simulé, la Recherche Tabou, les méthodes évolutionnistes et les algorithmes de colonies de fourmis.

La figure 1.10- schématise les principales métaheuristiques. Ce qui est en noire représente l'approche utilisé dans notre travail, que nous avons présenté dans le troisième chapitre.

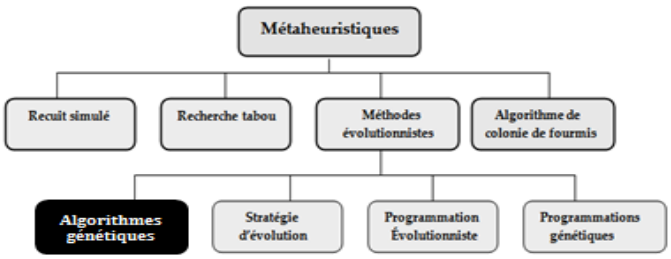


FIG. 1.10 – Les principales métaheuristiques.

1.7.5 Méthodes hybrides

L'utilisation d'une hybridation entre une approche métaheuristique et une approche exacte est souvent souhaitée pour construire un algorithme plus efficace, qui permet de contourner plus ou moins la difficulté pour résoudre un problème d'optimisation. Pour cela un algorithme génétique d'intervalle hybride (HIG) est présenté dans le chapitre quatre afin d'obtenir de meilleurs résultats pour notre problème posé. Ce dernier est capable de déterminer le bassin d'attraction de l'optimum global ainsi que la détermination de ce dernier.

1.8 Conclusion

Ce chapitre a été consacré à des généralités sur l'optimisation, en particulier l'optimisation globale. L'objectif consiste à présenter les éléments essentiels utilisés et de montrer l'intérêt de l'optimisation globale, dont une méthode d'optimisation globale très intéressante a attiré notre attention, elle s'agit de l'algorithme génétique d'intervalle hybride (HIG) consistant au couplage des deux méthodes :

1. La méthode d'analyse d'intervalle qui fait partie des méthodes exactes ;
2. Et l'algorithme génétique qui est une méthode métaheuristique.

Dans le deuxième chapitre, nous présentons la méthode d'analyse d'intervalle (Méthode brunch-and-bound d'intervalle).

Chapitre 2

Méthode d'analyse d'intervalles

2.1 Introduction

L'analyse d'intervalle est un outil développé par Moore en 1966 pour contrôler les erreurs occasionnées par les calculs numériques. Au lieu d'approcher x par un nombre représentable en machine, nous le remplacerons par un intervalle qui le contient. Les bornes de cet intervalle seront des nombre représentables en machine.

L'idée de l'analyse d'intervalle est de représenter tous les nombres réels par deux nombres flottants qui l'encadrent. L'analyse d'intervalle a rapidement été utilisée en optimisation, pour la résolution de systèmes linéaires et non linéaires. Grace à celle-ci, de nombreux algorithmes d'optimisation globale ont été développés, l'analyse d'intervalle utilisée ici permet de déterminer l'optimum global ainsi que tous les optimums d'un problème d'optimisation avec ou sans contraintes.

2.2 Opérations ensemblistes pures

2.2.1 Inclusion ensembliste

Définition 2.1. Soit A et B deux sous-ensembles de $\mathbb{R}^n$. On dit que A est inclus dans B si et seulement si tout élément de A appartient à B. Il vient de ce fait

$$A \subset B \Leftrightarrow (\forall x \in A,\ x \in B)$$

Les opérations suivantes s'appliquent sur des sous-ensembles de $\mathbb{R}^n$ en général. Elles regroupent l'union, l'intersection, le produit scalaire et la projection.L'union et l'intersection d'ensemble sont deux notions très utilisées. Etant donnés A et B deux sous-ensembles de $\mathbb{R}^n$ et $\mathbb{R}^m$.Nous avons :

- $A \cup B = \{x \in \mathbb{R}^n \ \mid \ x \in A \ ou \ x \in B\}$, avec $n = m$
- $A \cap B = \{x \in \mathbb{R}^n \ \mid \ x \in A \ et \ x \in B\}$, avec $n = m$

- $A \cdot B = \{(x,y)^T \in \mathbb{R}^{n+m} \mid x \in A \ \ ou \ \ y \in B\}$
- $proj_i(A) = \{x_i \in \mathbb{R} \mid \exists (x_1, ..., x_i, ..., x_n)^T \in A\}$

D'une manière générale, il va de soit que des principes valables pour les sous ensembles, le seront en particulier pour les pavés ainsi que les intervalles.

2.3 Intervalles, pavés et sous-pavages

Cette section a pour objet un rappel des notions sur les intervalles et les pavés. Ces notions constituent des éléments fondamentaux de l'analyse par intervalles. La plupart des définitions et opérations ensemblistes présentées sont tirées de [9] et [12].

2.3.1 Définitions et notations

Intervalle : Un intervalle $[x]$ est un sous-ensemble convexe, borné et fermé de réels.Nous avons

$$[x] = [\underline{x}, \overline{x}] = \{x \in \mathbb{R}, \ \underline{x} \leq x \leq \overline{x}\}$$

où $\underline{x}$ et $\overline{x}$ sont respectivement les bornes inférieures et supérieures de [x]. L'ensemble des intervalles est noté $\mathbb{I} = \{x = [\underline{x}, \overline{x}] \mid \underline{x}, \overline{x} \in \mathbb{R} \ et \ \underline{x} \leq \overline{x}\}$.

- La largeur d'un intervalle est donnée par $w([x]) = \overline{x} - \underline{x}$
- Point milieu d'un intervalle est donnée par $m([x]) = \frac{\overline{x}+\underline{x}}{2}$
- Un intervalle est dégénéré quand sa largeur est nulle c.à.d un intervalle dont les deux borne sont égales $\overline{x} = x = \underline{x}$

Remarque 2.1. l'union de deux intervalles ne donne pas toujours un intervalle, par exemple l'union des intervalles [1 ; 2] et [3 ;4] donne pour résultat un ensemble disjoint de deux intervalles. Bien que ce résultat soit intéressant, il rend difficile la manipulation de l'union, le nombre d'intervalles disjoints lors de la résolution des problèmes pouvant croitre rapidement .il faut donc introduire une notion d'union différente qui renvoie le plus petit intervalle contenant l'ensemble des solutions de l'union. L'intervalle solution est donc l'encadrement extérieur de l'union défini par la formule $[x] \cup [y] = [[x] \cup [y]]$.

Ou l'opérateur [..] renvoie le plus petit intervalle encadrant l'union. Ces opérations sont codées par les formules suivantes pour des intervalles non vides :

$$[x] \cup [y] = [min(\underline{x}; \underline{y}); max(\overline{x}; \overline{y})]$$

$$[x] \cap [y] = \begin{cases} \phi \quad si \ \ (\overline{x} < \underline{y}) \vee (\overline{y} < \underline{x}) \\ [max(\underline{x}; \underline{y}); min(\overline{x}; \overline{y})] \quad sinon \end{cases}$$

Pavé (Boite) : De la même façon que nous représentons un réel par un intervalle, il est possible de concevoir une structure de compact simple afin de représenter un vecteur à composantes réelles. Un pavé $[X]$ (ou vecteur n-dimensionnel d'intervalle qui définit l'espace de recherche dans lequel se trouvent les vecteur des inconnues) est un compact de $\mathbb{R}^n$ défini par le produit cartésien de n intervalles. On note

$$[X] = [\overline{x}_1; \underline{x}_1] \times [\overline{x}_2; \underline{x}_2] \times ... \times [\overline{x}_n; \underline{x}_n] = [x_1] \times [x_2] \times ... \times [x_n]$$
$$[X] = ([\overline{x}_1; \underline{x}_1] \times [\overline{x}_2; \underline{x}_2] \times ... \times [\overline{x}_n; \underline{x}_n])^T$$

Pour tout $i \in \{1, 2, ..., n\}$, l'intervalle $[x_i]$ correspond à la i^{eme} composante de $[X]$. Les caractéristiques du pavé sont décrites comme suit :

- La longueur du pavé $[X]$ est donnée par $w([X]) = \max_{i=1}^{n}(w[x_i])$
- Le centre de $[X]$ est défini par $m([X]) = (\frac{\overline{x}_1+\underline{x}_1}{2}; \cdots; \frac{\overline{x}_i+\underline{x}_i}{2}; \cdots; \frac{\overline{x}_n+\underline{x}_n}{2})^T$
- Le volume de $[X]$ est donné par :

$$Vol([X]) = \prod_{i=1}^{n} w(\{x_i\}) = (\overline{x}_1 + \underline{x}_1) \cdot (\overline{x}_2 + \underline{x}_2) \cdots (\overline{x}_n + \underline{x}_n)$$

Sous-pavage : Un sous pavage est défini par une union de pavés. En particulier, un sous-pavage régulier est constitué d'un ensemble de pavés disjoints ou des pavés qui ne partagent que leurs frontières.

Remarque 2.2. C'est cette structure de pavé qui permet d'approximer des vecteurs incertains. Chaque composante du vecteur appartient à un intervalle.

Cependant la représentation par pavé implique une perte de précision. Un ensemble S de forme quelconque peut être encadré par un pavé $[X]$. L'encadrement réalisé contient tous les points de S mais aussi tous les points du pavé n'appartenant pas à S. Sur la figure 2-1 le pavé qui encadre $\mathcal{S}$ contient aussi des points n'appartenant pas à $\mathcal{S}$ comme par exemple le couple (a_1, a_2). Toute la zone en gris clair est donc une incertitude sur la connaissance de $\mathcal{S}$ qui est due à la représentation des variables sous forme de pavés. Ce pessimisme engendré par les pavés est aussi appelé wrapping effect.

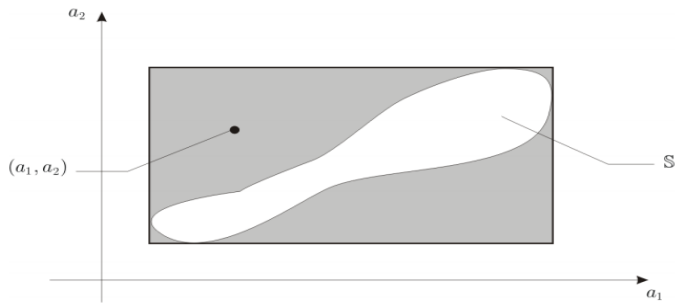


FIG. 2.1 – Illustration de l'encadrement d'un ensemble par une boîte, tous les points gris n'appartiennent pas à l'ensemble.

2.4 Arithmétique d'Intervalles

La première de ces arithmétiques présentées dans ce rapport est l'arithmétique d'intervalle développé par R. E. Moore .Sa théorie fit une énorme avancée en mathématique appliquée car ce fut la première fois que l'on put écrire des algorithmes exacts entièrement déterministes ; c'est à dire que d'une exécution à l'autre on est sûr d'obtenir le même résultat et que ce résultat est prouvé exact mathématiquement à la précision près.

2.4.1 Principe de l'arithmétique d'intervalles

Le principe consiste à représenter tout réel (ou calcul intermédiaire) par un intervalle de deux nombres flottants représentables en machine et contenant ce réel (exemple 2.1). Ainsi toutes les erreurs d'arrondi machine, peuvent être automatiquement prises en compte.

Exemple 2.1.

$$\frac{1}{3} = [0.33333333; 0.33333334]$$
$$log(2) = [0.69314718055; 0.69314718056]$$

Tout comme une arithmétique classique, le calcul par intervalles manipule des opérations arithmétiques.

2.5 Opération Arithmétiques

Une opération mathématique quelconque $\circ$ entre deux sous-ensembles A et B de $\mathbb{R}^n$ est définie comme suit :

$$A \circ B = \{x \circ y \mid x \in A \wedge y \in B\}$$

Les opérations arithmétiques considérées ici sont stables. Autrement dit, les résultats de ces opérations sont de même type que leurs arguments. Par exemple, le résultat d'une addition ou du produit de deux intervalles est aussi un intervalle.

Pour $\circ \in \{+, -, \cdot, /\}$, le récapitulatif des opérations sur les intervalles [x] et [y] est donné par :

1. $[x] + [y] = [\underline{x} + \underline{y}, \overline{x} + \overline{y}]$
2. $[x] - [y] = [\underline{x} - \underline{y}, \overline{x} - \overline{y}]$
3. $-[y] = [-\overline{y}, -\underline{y}]$
4. $[x] \cdot [y] = [min(\underline{x} \cdot \underline{y}, \overline{x} \cdot \overline{y}, \underline{x} \cdot \overline{y}; \overline{x} \cdot \underline{y}); max(\underline{x} \cdot \underline{y}; \overline{x} \cdot \overline{y}; \underline{x} \cdot \overline{y}; \overline{x} \cdot \underline{y})]$

   Dans le cas de la division, Nous Avons :

5. $\frac{1}{[y]} = [\{\frac{1}{y} \mid y \in [y]\}]$

   Qui représente le plus petit intervalle qui contient l'ensemble des réels $\frac{1}{y}$ pour tout y appartenant à $[y]$.Plus concrètement, il vient :

$$\frac{1}{[y]} = \begin{cases} \mathbb{R} \quad si \quad [y] = [0;0] \\ [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}] \quad si \quad 0 \notin [y] \\ [\frac{1}{\overline{y}}, +\infty] \quad si \quad \underline{y} < 0 \quad et \quad \overline{y} > 0 \\ [-\infty, \frac{1}{\underline{y}}] \quad si \quad \underline{y} < 0 \quad et \quad \overline{y} = 0 \\ \mathbb{R} \quad si \quad \underline{y} < 0 \quad et \quad \overline{y} > 0 \end{cases} \tag{2.1}$$

   Remarque 2.3. L'intérêt de l'ajout de bornes $-\infty$ et $+\infty$ est l'obtention d'un résultat, quelque soient les singularités et les points où les fonctions ne sont pas définies et la possibilité d'explorer $\mathbb{R}$ dans son intégralité pour les recherches de preuves. Dans les faits, si nous obtenons le résultat $]-\infty, +\infty[$, une étude reste nécessaire et ce résultat n'a que peu de valeur.

6. $\frac{[x]}{[y]} = \frac{[\underline{x}, \overline{x}]}{[\underline{y}, \overline{y}]} = [\underline{x}, \overline{x}] \cdot [\frac{1}{\overline{y}}, \frac{1}{\underline{y}}] \quad si \quad 0 \notin [\underline{y}, \overline{y}]$

2.6 Le problème de dépendance

Pour illustrer une dernière fois le problème de dépendance, prenons l'exemple de l'évaluation par intervalles de deux expressions quand $[x] = [-1, 1]$:

$$[x] + [x] - [x] = [-1, 1] + [-1, 1] - [-1, 1] = [-3, 3]$$

$$[x] = [-1, 1]$$

L'intervalle solution est dépendant du nombre d'occurrences de la variable x.

$$[x] + [x] - [x] = \{a + b - c | a \in [x], b \in [x], c \in [x]\} \tag{2.2}$$

$$[x] = \{a \in [x]\}$$

L'expression (2.2) correspond donc à l'ensemble des réels $a + b - c$ pour tout a, b et c pris chacun séparément dans l'intervalle $[-1; 1]$, et non pas à l'ensemble des réels avec $a = b = c$.

Du fait de ce problème de dépendance, il faut veiller à limiter le nombre d'occurrences de chaque variable afin d'obtenir des intervalles plus petits.

2.7 Propriétés algébriques

On peut constater que les opérations définies ci-dessus ne présentent pas les propriétés algébriques de leurs contreparties ponctuelles. Tout d'abord, la soustraction n'est pas la réciproque de l'addition.

Exemple 2.2.

Si $[x] = [2, 3]$, $[x] - [x] = [2, 3] - [2, 3] = [2, 3] + [-3, -2] = [-1, 1] \neq 0$
Même s'il le contient. En effet,

$$[x] - [x] = \{a - b \ \mid \ a \in [x] \quad b \in [x]\} \supset \{[x] - [x] \ \mid \ a \in [x]\} = \{0\}$$

et l'inclusion est stricte. De la même façon, la division n'est pas la réciproque de la multiplication :

si $[x] = [2, 3]$, l'intervalle $\frac{[x]}{[x]} = \frac{[2,3]}{[2,3]} = [\frac{2}{3}, \frac{3}{2}]$ n'est pas égal à 1a même s'il le contient. De plus, la multiplication d'un intervalle par lui-même n'est pas égal à l'élévation au carré. Enfin, la multiplication n'est pas distributive par rapport à l'addition :

Si $[x] = [-2, 3]$, $[y] = [1, 4]$ et $[z] = [-2, 1]$,

$$\begin{aligned}[x] \cdot ([y] + [z]) &= [-2, 3] \cdot ([1, 4] + [-2, 1]) \\ &= [-2, 3] \cdot [-1, 5] \\ &= [-10, 15]\end{aligned}$$

$$[x] \cdot [y] + [x] \cdot [z] = [-2, 3] \cdot [1, 4] + [-2, 3] \cdot [-2, 1]$$

$$= [-8, 12] + [-6, 4]$$

$$= [-14, 16]$$

Comme l'illustre cet exemple, la multiplication est sous-distributive par rapport à l'addition, c'est-à-dire

$$[x] \cdot ([y] + [z]) \subset [x] \cdot [y] + [x] \cdot [z]$$

2.8 Fonction Usuel

On peut également définir des fonctions usuelles prenant x un intervalle de $\mathbb{R}$, nous avons :

2.8.1 Puissance

$$\forall p \in \mathbb{N}^+, x^{2p} = \begin{cases} [0;0] & \text{si} \quad \text{p=0} \\ [0; max(-\underline{x}, \overline{x})^{2p}] & si \quad 0 \in [x] \\ [\underline{x}^{2p}; \overline{x}^{2p}] & si \quad x > 0 \\ [\overline{x}^{2p}; \underline{x}^{2p}] & \text{sinon} \end{cases}$$

$x^{2p+1} = [\underline{x}^{2p+1}; \overline{x}^{2p+1}]$

2.8.2 Logarithme

Le logarithme est une fonction strictement croissante sur $]0; +\infty[$ nous obtenons :

$$log[x] = [log(\underline{x}); log(\overline{x})] \qquad [x] \subset [0; +\infty[$$

2.8.3 Racine carrée

Est aussi une fonction strictement croissante sur $]0; +\infty[$ nous obtenons :

$$\sqrt{[x]} = \sqrt{[\underline{x}; \overline{x}]} = [\sqrt{\underline{x}; \overline{x}}] \qquad [x] \subset [0; +\infty[$$

2.8.4 Exponentiel

Est une fonction strictement croissante sur $\mathbb{R}$

$$\exp[x] = [\exp[\underline{x}]; \exp[\overline{x}]]$$

2.8.5 Valeur absolue

$$|[x]| = \begin{cases} [0, max\{|\underline{x}|; |\overline{x}|\}] & si \;\; 0 \in [x] \\ [\underline{x}; \overline{x}] & si \;\; \underline{x} \geq 0 \\ [\underline{x}\overline{x}] & si \;\; \overline{x} \leq 0 \end{cases}$$

2.9 Fonctions d'inclusion

Nous avons vu dans les paragraphes précédents quelques bases du calcul par intervalles. Ici, nous allons nous intéresser à l'évaluation de fonctions vectorielles dont les variables sont des intervalles. Il sera présenté à cet effet plusieurs techniques d'évaluation plus ou moins efficaces.

Définition 2.2. Soit f une fonction vectorielle définie de $\mathbb{R}^n$ dans $\mathbb{R}^m$. Une fonction d'inclusion est une fonction de $\mathbb{R}^n \rightarrow \mathbb{R}^m$ qui satisfait,

$$[f]([x]) \supset \{f(x) \;\; | \;\; x \in [x]\}$$

Ce qui revient à :

$$\forall [x] \in \mathbb{R}^n \;\; , \;\; f([x]) \subset [f]([x])$$

où $[x]$ est un pavé de $\mathbb{R}^n$.

Etant données les fonctions d'inclusion pour chaque composante f_i de f telles que :

$$[f_i]([x]) \supset \{f_i(x) \;\; | \;\; x \in [x]\} \;\; , \;\; i = 1, 2, ..., m$$

Il vient

$$[f]([x]) = [f_1]([x]) \times [f_2]([x]) \times ... \times [f_m]([x])$$

Dans le cadre d'application des méthodes intervalles, il est clair que nous avons intérêt à trouver le plus petit pavé contenant $f([x])$ pour tout $[x] \in \mathbb{R}^n$, c'est la fonction d'inclusion minimale pour f .

Nous avons : $[f]^*([x]) = [f([x])]$ où $[A]$ est le plus petit pavé qui contient l'ensemble A

2.9.1 Propriétés

la fonction d'inclusion f est :

– Une fonction monotone si et seulement si

$$[x] \subset [y] \Rightarrow [f]([x]) \subset [f]([y])$$

avec [x] et [y] sont deux pavés de $\mathbb{R}^n$.

– Une fonction minimale

$$\forall x \in \mathbb{R}^n \quad , \quad [f]([x]) = [f([x])]$$

La fonction d'inclusion minimale est unique pour chaque fonction vectorielle.

– Une fonction convergente si et seulement si pour toute suite de pavés $[x]_k$,nous avons

$$\lim_{k\to\infty} w([x_k]) = 0 \Rightarrow \lim_{k\to\infty} ([f]([x]_k)) = 0$$

ce qui entraîne, $\forall x \in \mathbb{R}^n, [f](x) = f(x)$

Dans (Neumaier 1990) , il est démontré qu'il est toujours possible de trouver une fonction d'inclusion convergente $[f]$ quand la fonction f est continue et définie par une expression arithmétique. Notons qu'en, général $[f([x])]$ est la boite enveloppée de $f([x])$ (la plus petite boite qui contient $f([x])$).les inclusions suivantes sont vérifiées :

$$f([x]) \subset [f([x])] \subset [f]([x])$$

une illustration de ces inclusions est donnée sur la Fig.2.2.

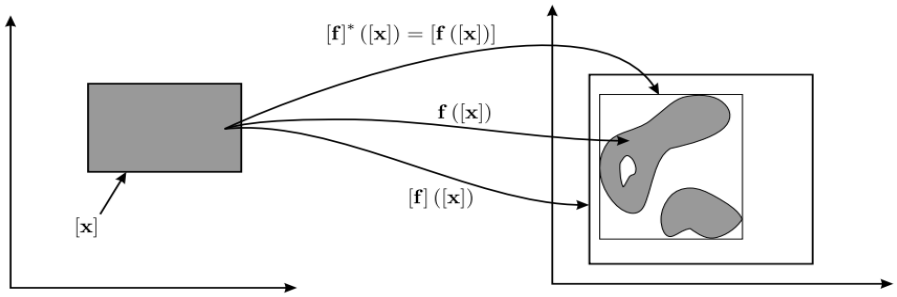


FIG. 2.2 – Image d'un pavé par les fonctions d'inclusion $[f]([x]), [f]^*([x])$ et $f([x])$.

2.9.2 Fonctions élémentaires

Considérons E et F deux sous-ensembles de $\mathbb{R}$. On appelle fonctions élémentaires toute fonction f définie de E dans F pour laquelle, nous disposons d'une fonction d'inclusion minimale $[f]^* : \mathbb{R} \to \mathbb{R}$,telle que :

$$[f]^*([x]) = [\{f(x) \mid x \in [x]\}]$$

Toute fonction élémentaire f, continue et monotone, admet comme fonction d'inclusion minimale,

$$[f]^*([x]) = [min(f(\underline{x}), f(\overline{x})), max(f(\underline{x}), f(\overline{x}))]$$

Si f n'est pas monotone, la réalisation de la fonction d'inclusion minimale dépend alors d'une étude globale des variations de la fonction. Dans ce cas, la méthode employée consiste à exploiter la "monotonie par morceaux" de f. Notons Z l'ensemble des entiers relatifs. Le domaine E est décomposé en sous domaines $E_k(E = \bigcup_{k\in Z} E_k)$ dans lesquels la fonction est continue et garde un comportement monotone (strictement croissante ou décroissante). Dans la suite, nous appellerons les domaines $E_k(k \in Z)$. Les domaines de monotonie de f . Notons que le nombre de ces domaines n'est pas nécessairement fini.

par exemple dans le cas de fonctions périodiques comme $f(x) = sin(x)$.

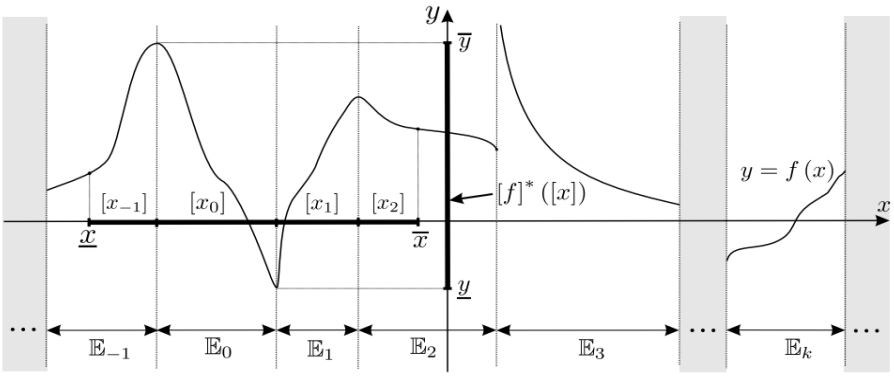


FIG. 2.3 – Fonction d'inclusion minimale dans le cas d'une fonction f non monotone. Décomposition en domaines de monotonie $E_k(k \in Z)$.

Comme illustré sur la figure 2.3, nous considérons un intervalle $[x] \in \mathbb{R}$ qui s'étend sur un nombre fini de domaines E_k. Notons $[x_k] = [x] \cap E_k$, la fonction d'inclusion minimale pour f devient :

$$[f]^*([x]) = [f]^*(\bigcup_{k=r}^{s}[x_k]) = [\bigcup_{k=r}^{s}[f]^*([x_k])]$$

Avec s et r deux entiers relatifs telle que $s \geq r$ la fonction f étant continue et monotone sur $[x_k]$ il vient :

$$[f]^*([x]) = [\bigcup_{k=r}^{s}(min(f(\underline{x}_k); f(\overline{x}_k)); max(f(\underline{x}_k); f(\overline{x}_k)))]$$

Après développement, nous obtenons

$$[f]^*([x]) = [\underline{y}, \overline{y}]$$

avec :

$$\underline{y} = (min(f(\underline{x}_k); f(\underline{x}_{k+1}); \cdots; f(\underline{x}_s); f(\overline{x}_r); f(\overline{x}_{r+1}); \cdots; f(\overline{x}_s))$$

et

$$\overline{y} = (max(f(\underline{x}_k); f(\underline{x}_{k+1}); \cdots; f(\underline{x}_s); f(\overline{x}_r); f(\overline{x}_{r+1}); \cdots; f(\overline{x}_s))$$

Il est ainsi possible de définir des fonctions d'inclusion minimales pour de nombreuses fonctions usuelles $(cos, sin, tan, exp, log, (.)^n, ...etc.)$.

Exemple 2.3.

soit la fonction $cos(x)$ non monotone, nous procédons de la façon suivante :
$[cos]([x]) = [a, b]$,avec

$$a = \begin{cases} -1 & si \ \ \exists k \in \mathbb{Z} \ \ | \ \ (2k+1)\pi \in [x] \\ min(cos(\underline{x}), cos(\overline{x})) & \text{sinon} \end{cases}$$

$$b = \begin{cases} 1 & si \ \ \exists k \in \mathbb{Z} \ \ | \ \ 2k\pi \in [x] \\ min(cos(\underline{x}), cos(\overline{x})) & \text{sinon} \end{cases}$$

Exemple 2.4.

Considérons la fonction $f(x) = x^2$, la fonction d'inclusion minimale pour f conformément à la figure 2.4 est donnée par,

$$[f]^*([x]) = \begin{cases} [0, max(\underline{x}^2, \overline{x}^2)] & si \ \ \underline{x}, \overline{x} \leq 0 \\ [\underline{x}^2, \overline{x}^2] & \text{sinon} \end{cases}$$

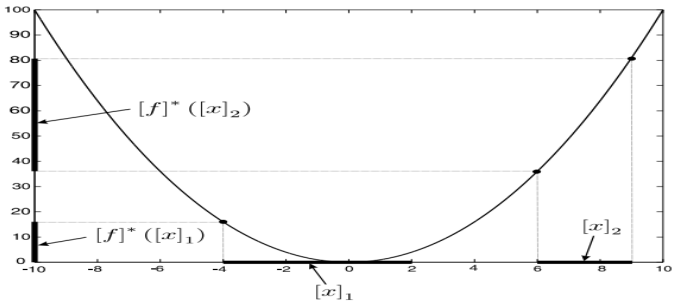


FIG. 2.4 – Fonction d'inclusion minimale pour $f(x) = x^2$

2.9.3 Fonction d'inclusion naturelle

Le théorème suivant permet de déterminer une fonction d'inclusion naturelle pour des fonctions de plusieurs variables. Ces fonctions sont construites à partir de compositions d'opérateurs arithmétiques $(+, -, \cdot, /)$ ainsi que de fonctions élémentaires usuelles $(tan, cos, sin, exp, log, \sqrt{.}, ...etc.)$.

Théorème 2.1. *Considérons une fonction définie par*

$$f : \mathbb{R}^n \to \mathbb{R}$$

$$(x_1, x_2, \cdots, x_n) \mapsto f(x_1, x_2, \cdots, x_n)$$

Une fonction d'inclusion convergente $[f] : \mathbb{R}^n \to \mathbb{R}$ *pour* f *est obtenue par la substitution, de chaque variable réelle* x_i *par son intervalle d'appartenance* $[x_i](i = 1, 2, ..., n)$*, ainsi que chaque opérateur ou fonction élémentaire par la fonction d'inclusion minimale équivalente. La fonction d'inclusion ainsi définie est appelée fonction d'inclusion naturelle pour* f*.*

Toutefois, la fonction d'inclusion naturelle n'est minimale qu'à certaines conditions. Nous reviendrons sur ces conditions à travers l'étude de l'exemple suivant.

Exemple 2.5.

la fonction $f : \mathbb{R}^2 \to \mathbb{R}$ dont l'expression est donnée par :

$$f(x_1, x_2) = \frac{x_2}{_1 + x_2} + sin(x_1)cos(x_1) \tag{2.3}$$

avec $x_1 \in [-1, 2]$ *et* $x_2 \in [3, 5]$. En appliquant directement le théorème 2.1 sur f, nous obtenons la fonction d'inclusion naturelle suivante,

$$[f]_1([x_1], [x_2]) = \frac{[x_2]}{[x_1] + [x_2]} + [sin]([x_1])[cos]([x_1])$$

Il vient après évaluation : $[f]_1([-1, 2], [3, 5]) = [-0.42, 3.5]$. Il est possible de trouver une autre fonction d'inclusion naturelle $[f]_2$ pour f, en manipulant l'expression (2.3) puis en utilisant le théorème 2.1. Nous avons :

$$[f]_2([x_1], [x_2]) = \frac{1}{1 + \frac{[x_1]}{[x_2]}} + [sin]([x_1])[cos]([x_1])$$

Nous obtenons par la suite $[f]_2([-1, 2], [3, 5]) = [-0.2415, 2.5]$. En fin , par le même procédé que précédemment, il vient une troisième fonction d'inclusion $[f]_3$ pour f donnée par

$$[f]_3([x_1], [x_2]) = \frac{1}{1 + \frac{[x_1]}{[x_2]}} + \frac{[sin](2.[x_1])}{2}$$

Nous calculons $[f]_3([-1,2],[3,5]) = [0.1,2]$.

Dans l'exemple 2.5, $[f]_1$, $[f]_2$ et $[f]_3$ représentent les extensions intervalles de la même fonction f. Pourtant, nous constatons lors de l'évaluation avec les mêmes arguments intervalles, des résultats nettement différents. En effet, $[f]_3$ est ici, la fonction d'inclusion la plus précise, même si celle-ci n'est pas minimale. Le pessimisme (ou la surévaluation des bornes d'incertitudes) observé sur les trois fonctions d'inclusion dépend du nombre d'occurrences des variables intervalles. Pour $[f]_1$, il y a 3 occurrences de x_1 et 2 occurrences de x_2. Dans le cas de $[f]_2$, nous avons 3 occurrences de x_1 et une occurrence de x_2. Dans le cas de $[f]_3$, nous ne comptons plus que 2 occurrences de x_1 et une occurrence de x_2. Ainsi, plus le nombre d'occurrences des variables est important dans une fonction, plus le pessimisme sur les évaluations des fonctions d'inclusion naturelles est important. Nous expliquons cette situation par le point de vue suivant. Dans l'expression d'une fonction d'inclusion, les intervalles spécifient des variables qui n'entretiennent aucune relation de dépendance entre-elles, cette situation est connue sous le nom de problème de dépendance.

Considérons la fonction : $g(x) = (x.x) + x + 1 = (x + \frac{1}{2})^2 + \frac{3}{4}$
Nous avons par définition la fonction d'inclusion naturelle

$$[g]_1([x]) = [x].[x] + [x] + 1$$

$$= \{(x.y) + z + 1 \mid x \in [x], y \in [x] \; et \; z \in [x]\}$$

Qui est différente de

$$[g]_2([x]) = ([x] + \frac{1}{2})^2 + \frac{3}{4}$$

$$= \{(x + \frac{1}{2})^2 + \frac{3}{4} \mid x \in [x]\}$$

$$= \{x^2 + x + 1 \mid x \in [x]\}$$

$$= g([x])$$

D'où

$$\forall [x] \in \mathbb{R}, [g]_2([x]) \subset [g]_1([x])$$

Ainsi, la fonction d'inclusion naturelle est minimale si chacune des variables n'intervient qu'une seule fois dans l'expression de la fonction.

Parmi les types d'extensions intervalles de fonctions présentés, la fonction d'inclusion naturelle reste la plus facile à mettre en œuvre. Cependant, afin de réduire le problème de

pessimisme des fonctions d'inclusion naturelles, il est utile de procéder à des manipulations algébriques sur les expressions des fonctions. Nous pouvons ainsi tenter de réduire les occurrences des différentes variables.

2.9.4 Fonction d'inclusion centrée

Ce type de fonction d'inclusion permet dans certaines situations, d'atténuer le problème de multi occurrences des variables (voir [11] et [10]). Considérons une fonction $f : \mathbb{R}^n \to \mathbb{R}$ différentiable sur un pavé $[x]$, notons la variable $x = (x_1, x_2, ..., x_n)^T$ et $x_c = m([x])$ le centre du pavé $[x]$. Le théorème de la valeur moyenne s'écrit :

$$\forall\ x \in [x],\quad \exists\ x_0 \in [x]\ \mid\ f(x) = f(x_c) + \nabla f(x_0)(x - x_c)$$

où $\nabla f(x) = (\frac{\partial f(x)}{\partial x_1}, \frac{\partial f(x)}{\partial x_2}, ..., \frac{\partial f(x)}{\partial x_n})$ représente le gradient de f. Par la suite, nous avons pour tout $x \in [x]$,

$$f(x) \in f(x_c) + \nabla f([x])(x - x_c)$$

Le théorème 2.1 sur la fonction d'inclusion naturelle permet d'établir la relation

$$f([x]) \in f(x_c) + \nabla f([x])([x] - x_c)$$

A partir de ceci, nous définissons la fonction d'inclusion centrée ou forme centrée pour f par :

$$[f]_c([x]) = f(x_c) + [\mathrm{g}]([x])([x] - x_c)$$

avec $[g]([x])$ qui correspond à une fonction d'inclusion pour le gradient de f. Cette fonction est construite à partir de fonctions d'inclusion naturelle ou centrée. Dans le cas où la forme centrée est utilisée, $[f]_c([x])$ devient alors une fonction d'inclusion dite de Taylor notée $[f]_T([x])$. La fonction d'inclusion centrée est convergente car elle est essentiellement composée de fonctions d'inclusion naturelles qui sont convergentes. Afin de comprendre le principe de la forme centrée, considérons une fonction scalaire à une variable $f(x)$ dérivable sur l'intervalle $[x]$. La fonction d'inclusion centrée pour f est donnée par

$$[f]_c([x]) = f(x_c) + [g]([x])([x] - x_c)$$

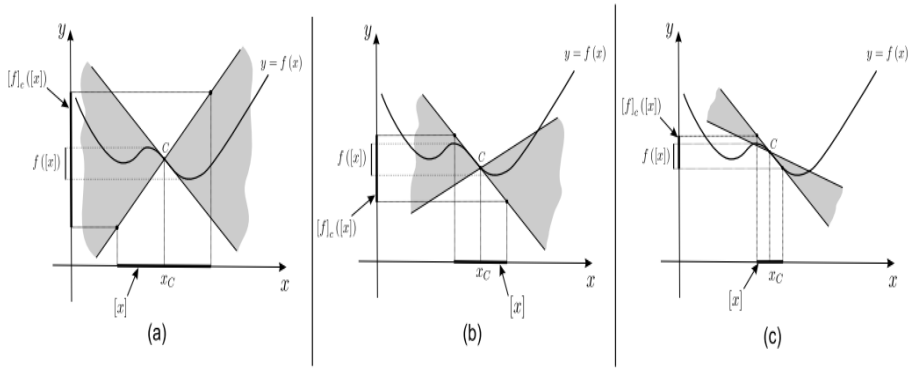


FIG. 2.5 – Principe de la forme centrée dans le cas à une dimension

avec $[g]([x])$ une fonction d'inclusion pour $f'(x)$. Les figures 2.5a, 2.5b et 2.5c représentent le principe de la forme centrée pour trois intervalles de largeurs différentes $[x]$.

Sur ces figures, les secteurs ou cônes en gris sont formés par les intersections au point C $(x_c = m([x]), y_c = f(m([x])))$ des plus grandes et plus petites pentes atteintes par $f(x)$ sur $[x]$. Les différents cônes sont représentés dans un cas où nous supposons disposer de la fonction d'inclusion minimale pour la dérivée de f. Nous remarquons sur les figures 2.5a, 2.5b et 2.5c, selon la taille des intervalles $[x]$, un pessimisme nettement différent sur l'approximation de $f([x])$ par $[f]_c([x])$. En effet, le pessimisme relevé suite à l'évaluation de $[f]_c([x])$. est relativement important sur la figure 2.5(a) , tandis qu'il diminue sur la figure 2.5(b) et encore plus sur la figure 2.5(c). Ces évolutions laissent présumer d'une efficacité de la méthode sur des pavés de faibles longueurs. La forme centrée sera aussi utilisée dans le cadre de fonctions à variables multi-occurrentes et dont il est difficile de réduire les occurrences par des simples manipulations algébriques.

2.9.5 Comparaison

Dans ce paragraphe, nous allons apporter quelques éléments de comparaison pour différentes fonctions d'inclusion. La plupart de ces éléments sont issus de [9]. Le taux de convergence d'une fonction d'inclusion $[f] : \mathbb{R}^n \longrightarrow \mathbb{R}^m$ est défini par le plus grand coefficient α tel que

$$\exists \beta, \forall [x] \in \mathbb{R}^n | w([f]([x])) - w([f([x])]) \leq \beta w([x])^\alpha \tag{2.4}$$

quand $w([x])$ tend vers 0.
Pour une fonction d'inclusion minimale le taux de convergence est infini ($\alpha \rightarrow +\infty$) car $w([x]) \rightarrow 0$. Pour une fonction d'inclusion naturelle, nous avons au moins une convergence linéaire ($\alpha \geq 1$), tandis que la convergence d'une fonction d'inclusion centrée ou de Taylor est au moins d'ordre quadratique ($\alpha \geq 2$). Par conséquent, pour des pavés infinitésimaux ,

l'évaluation d'une forme centrée sera moins pessimiste que celle d'une fonction d'inclusion naturelle. Par contre pour des pavés relativement volumineux et des fonctions avec peu de dépendances multi-occurrentes, nous privilégierons l'utilisation de fonctions d'inclusion naturelles.

Reprenons l'exemple 2.5 où $f(x_1, x_2) = \frac{1}{1+\frac{x_1}{x_2}} + \frac{sin(2.x_1)}{2}$. Nous cherchons à comparer pour f, l'efficacité de la fonction d'inclusion naturelle et de la forme centrée. Pour cela, nous considérons 3 pavés de longueurs et de volumes différents :
$[x]_1 = [-1, 2] \times [3, 5]$, $[x]_2 = [-0.1, 0.2] \times [3.9, 4.1]$ et $[x]_3 = [-0.01, 0.02] \times [3.98, 4.02]$.

Nous constatons effectivement que le pavé $[x]_1$ est de longueur et de volume supérieurs à ceux des deux autres pavés $[x]_2$ et $[x]_3$. La forme centrée pour f s'écrit :

$$[f]_c([x]) = f(m_1, m_2) + [g_1]([x]) \cdot ([x_1] - m_1) + [g_2]([x]) \cdot ([x_2] - m_2)$$

avec $[g_1]([x])et[g_2]([x])$ qui sont respectivement les fonctions d'inclusion de $\frac{\partial f(x_1,x_2)}{\partial x_1}$ et $\frac{\partial f(x_1,x_2)}{\partial x_1}$,m_1 et m_2 sont les centres respectifs des intervalles de$[x_1]$ et $[x_2]$. Nous avons plus précisément

$$[f]_c([x]) = f(m_1, m_2) + (cos(2 \cdot [x_1]) - \frac{[x_1]}{[x_2] \cdot (1 + \frac{[x_1]}{[x_2]})^2}) \cdot ([x_1] - m_1) + (\frac{[x_2] - m_2}{[x_2]^2 \cdot (1 + \frac{[x_1]}{[x_2]})^2})$$

Les résultats obtenus à l'issu de l'évaluation des fonctions d'inclusion naturelle $[f]_n$ et centrée $[f]_c$ sont consignés dans le tableau ci-dessous.

	[x] = [-1,2]×[3,5]	[x] = [-0.1,0.2]×[3.9,4.1]	[x] = [-0.01,0.02]×[3.98,4.02]
w ([x])	3	0.3	0.03
$[f]_n([x])$	[0.1, 2]	[0.8518, 1.221025]	[0.985, 1.022514]
$[f]_c([x])$	[-1.8154, 4.435]	[0.91928, 1.155852]	[0.9924, 1.0151]
$\triangle([f]([x]))$	4.3504	-0.132653	-0.014814

Dans ce tableau, nous avons calculé les erreurs entre les largeurs des intervalles évalués :

$$\triangle([f]([x])) = w([f]_c([x]) - w([f]_n([x]))$$

Ces résultats viennent confirmer les tendances et les comportements attendus selon l'expression (2.4). Dans le premier cas ($[x] = [-1, 2] \times [3, 5]$), la fonction d'inclusion naturelle se révèle nettement moins pessimiste que la forme centrée ($\triangle([f]([x])) < 0$). Dans les deux autres cas ($[x] = [-0.1, 0.2] \times [3.9, 4.1]$ et $[x] = [-0.01, 0.02] \times [3.98, 4.02]$), les pavés sont beaucoup plus petits au sens du volume et de la longueur, il se trouve que la fonction d'inclusion naturelle devient moins efficace donc plus pessimiste que la forme centrée, en effet nous remarquons pour ces cas que l'erreur $\triangle([f]([x]))$ est négative.Finalement, nous pouvons envisager dans

la majorité des cas, l'utilisation d'une fonction d'inclusion hybride entre fonction d'inclusion naturelle et forme centrée, nous aurons alors :

$$[f]_{nc}([x]) = [f]_n([x]) \cap [f]_c([x])$$

2.10 Application à l'optimisation globale

L'optimisation globale consiste à trouver le minimum global d'une fonction avec ou sans contraintes, contrairement aux méthodes de type descente telle que Newton, qui convergent vers des minima locaux. Il existe beaucoup d'algorithmes heuristiques ou métaheuristiques très efficaces tel que la méthode Tabou ou le VNS qui permettent de converger vers un minimum s'approchant du minimum global, mais hélas ces méthodes n'apportent aucune preuve de l'optimalité de la solution ainsi trouvé. L'Algorithme de Branch and Bound par Intervalle (IBBA : pour Interval Branch and Bound Algorithm) est l'une des rares méthodes dans ce domaine permettant d'avoir une preuve mathématique que le résultat retouvé est bien le minimum global recherché à la précision près. [5,12]

2.10.1 Branch and Bound par Intervalle

Dans ce paragraphe, on rappelle le principe général d'un Algorithme de type Branch and Bound basé sur des méthodes d'intervalles Les premiers algorithmes de Branch and Bound par intervalles sont nés dans les années 70. Les premières études étaient surtout très théoriques et assez avares en résultats numériques. Ceci devait être certainement dû à la capacité des ordinateurs qui à l'époque ne permettait de disposer que de quelques méga-octets de mémoire. Le premier algorithme de Branch and Bound par intervalles fut celui de Moore-Skelboe en 1974 [4], puis Ichida et Fujii y rajoutèrent le test au point milieu en 1979 [3], et Hansen [2], Kearfott [7], Ratz [1] et beaucoup d'autres continuèrent à développer l'algorithme en y incluant des tests de monotonies, de concavité, etc., tout en considérant des problèmes de plus en plus généraux. Décrivons tout d'abord le principe de base d'un algorithme de type Branch and Bound utilisant l'analyse d'intervalles pour la résolution d'un problème d'optimisation.

2.10.2 Principe du Branch and Bound par Intervalle

L'algorithme IBBA que nous avons utilisé a été développé par F. Messine dans le cadre de sa thèse [6]. Depuis il a beaucoup évolué mais le principe reste toujours celui d'un Branch and Bound classique. Pour simplifier l'explication, supposons que l'on cherche le minimum d'une fonction f sans contrainte sur un intervalle $[x]$. Le but est de trouver un encadrement du minimum de f de largeur inférieure à la précision demandée. Pour cela on va découper l'intervalle d'étude $[x]$ en deux intervalles. Puis on calcule une borne inférieure et une borne

supérieure de f sur chacun des deux intervalles . Si la borne supérieure d'un des deux est inférieure à la borne inférieure du deuxième alors on est sûr que le minimum de f sur $[x]$ est situé dans le premier intervalle, on peut donc éliminer le deuxième (cf figure6). Si ce n'est pas le cas, on doit garder les deux intervalles que l'on stocke dans une liste. Ainsi on réitère le procédé sur les éléments présents encore dans la liste jusqu'à ce qu'on est un encadrement du minimum assez précis (Algorithme 1).

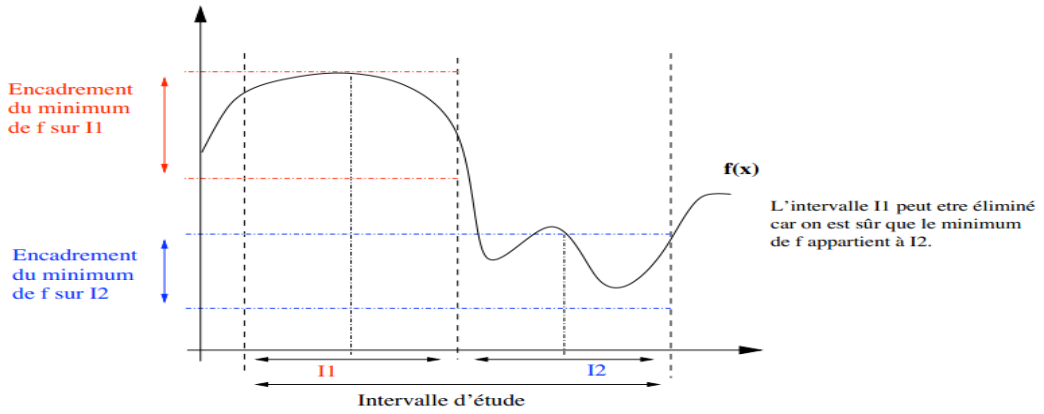


FIG. 2.6 – Principe de l'algorithme IBBA

Algorithme 1 (IBBA)

1. Initialiser $[x]$:= domaine initial contenant le minimum global.

2. Initialiser $\widetilde{f} := +\infty$: valeur courante de la borne supérieure du minimum.
3. Initialiser $\widetilde{y} := [x]$: valeur courante du minimum de f.

4. Initialiser la liste $L := (-\infty, [x])$:Structure qui va contenir tous les éléments qui restent encore à traiter.
$\forall (z, Z) \in L$,z représente une borne inférieure de fsur Z.

5. Extraire de L l'élément ayant la plus petite borne inférieure $min_{(z,Z)\in L} Z$

6. Découper l'intervalle Z associé en deux intervalles, donnant V_1 et V_2

7. Pour j :=1 à 2 faire
(a) Calculer v_j := lowbound (f, V_j) (calcul d'une borne inférieure de f sur V_j)
(b) Si $\widetilde{f} \geq v_j$ et si aucune contrainte n'est insatisfaite sur V_j alors :

- Insérer (v_j, V_j) dans L.

– Soit m le milieu de V_j, si m satisfait toutes les contraintes alors
faire $\widetilde{f} := min(\widetilde{f}, f(m))$
– Si $\widetilde{f}$ a changé alors
Supprimer de L tous les éléments (z, Z) tel que $z > \widetilde{f}$
et faire $\widetilde{y} := m$

8. Si $\widetilde{f} - min_{(z,Z)\in L} Z < \epsilon$ alors stop.
Sinon retourner à l'étape 5.

Comme on peut le constater l'algorithme 1 est très modulaire. En effet, il est facile d'ajouter des méthodes de découpe, de calcul de borne ou d'élimination. En théorie l'algorithme IBBA doit converger et donner un résultat dans un temps fini si l'on borne la découpe des intervalles à la précision sur les variables, c'est-à-dire si l'on ne découpe pas les intervalles plus petits que la précision demandée. Hélas, la complexité de l'algorithme étant exponentielle, le temps nécessaire est certe fini mais peut, vite devenir astronomique et la capacité mémoire des machines étant limitée , le nombre d'éléments devient tellement important qu'une partie des données est swapé sur le disque dur, ce qui aboutit à une perte totale des performances de calcul. C'est pourquoi il est nécessaire d'intégrer des méthodes d'accélération à IBBA sans pour autant perdre la propriété déterministe qui représente tout l'intérêt de ce type de méthode.

Remarque 2.4. Considérons un problème à 10 variables.

Pour stocker un intervalle, on a besoin de 16 octets en Fortran double précision. Un élément stocké est représenté par 1 intervalle par variable et 1 double précision pour la valeur de la borne inférieure de f.Ainsi on a besoin au minimum de 168 octets pour stocker 1 élément sans compter la place nécessaire à la structure. Donc si on considère une machine avec 2 Go de mémoire vive, on peut contenir au maximum environ 10 millions d'éléments, sans swaper sur le disque dur, ce qui est généralement facilement atteint au bout de 5 à 6 heures de calcul.

2.11 Algorithme de l'encadrement de l'optimum global

2.11.1 Construction des hyperplans

Nous allons utiliser le développement de Taylor du premier ordre et le théorème de la valeur moyenne pour construire des hyperplans d'appui de la fonction sur le pavé. Dans une première partie nous construirons des hyperplans minorant de la fonction f sur différents pavés de $\mathbb{R}^n$. Nous en déduirons une minoration de la fonction f sur le pavé considéré. Nous allons étudier dans cette première partie des méthodes permettant d'améliorer l'encadrement du minimum global. D'après la forme de Taylor d'une fonction, nous obtenons quelque fonctions de l'inclusion de f dans la boite $[X]$, cette fonction d'inclusion est appelée la forme Taylor :

$$\forall (x,y) \in [X]^2, f(y) \in f(x) + ([X]-x)^T g([X])$$

Ou

$$g([X]) = (\frac{\partial f}{\partial x_1}, \cdots, \frac{\partial f}{\partial x_n})$$

est un vecteur de gradient de f, g sera calculée par une méthode de différentiation automatique.

Soit $[X]_i = [\underline{x}_i, \overline{x}_i]$ et $G_i(x) = [\underline{g}_i, \overline{g}i]$ un encadrement du gradient de f sur $[X]$, pour $i = 1, 2, \cdots, n$ donc :

$$\forall x \in [X], \frac{\partial f}{\partial x_1}(x) \in G_i(x)$$

Soit $S = (s_1, \cdots, s_n)$ les coordonnées d'un sommet S de la boite $[X]$ et $g^s([X]) = (g_1^s([X]), \cdots, g_n^s([X]))^T$ pour tout k on définit le vecteur $g_k^s([X])$ par :

$$g_k^S([X]) = \frac{s_k - \overline{x}_k}{\underline{x}_k - \overline{x}_k}\underline{g}_k([X]) + \frac{s_k - \underline{x}_k}{\overline{x}_k} - \underline{x}_k\overline{g}_k([X]) \ \forall \ k \in \{1, 2, \cdots\}$$

On obtient les inégalités suivantes :

$$(y_i - \underline{x}_i)\underline{g}_i([X]) \leq (y_i - \underline{x}_i)\alpha \leq (y_i - \underline{x}_i)\overline{g}_i([X])$$

et

$$(y_i - \overline{x}_i)\overline{g}_i([X]) \leq (y_i - \overline{x}_i)\alpha \leq (y_i - \overline{x}_i)\underline{g}_i([X])$$

pour tout $i = 1, 2, \cdots, n$ et tout $\alpha \in G_i([X])$, nous pouvons déduire pour chaque sommet S de la boite X, un hyperplan d'appui d'une fonction f sur $[X]$ qui est définit comme suit :

$$\forall \ y \in [X], f(y) \geq f(s) + (y-s)^T g^s([X])$$

Donc, nous pouvons obtenir 2^n hyperplans de la fonction f pour boite $[X]$. Chacun de ces hyperplans est un hyperplan d'appui maximal au sommet S pour la boite $[X]$. On suppose par la suite :

$$\underline{g_i}([X]) \cdot \overline{g_i}([X]) < 0$$

Pour tout i ; cela est obtenu par l'application d'une épreuve de monotonie à f sur $[X]$, ceci nous conduit à la construction de l'hyperplan $\mathbb{R}_k$ sous la forme :

$$u_k(x) = f(s_k) + (x - s_k)^T g^{S_k}([X])$$

Proposition 2.1. *Pour tout $y_i \in [X]_i$, les deux minorations sont obtenues aux deux points $\underline{x}_i$ et $\overline{x}_i$:*

$$\bullet\ \forall x \in (y_i - \underline{x}_i)g_i([X]), x \geq (y_i - \underline{x}_i)\underline{g}_i([X])$$

$$\bullet\ \forall x \in (y_i - \overline{x}_i)g_i([X]), x \geq (y_i - \overline{x}_i)\overline{g}_i([X])$$

ceci pour tout $i \in (1.\cdots.n)$

Démonstration 2.1. *Pour tout $i = (1.\cdots.n)$, et pour tout $y_i \in [X]_i$, nous avons par définition de l'opérateur $\times$ de l'arithmétique d'intervalle*

$$(y_i - \underline{x}_i)g_i([X]) = [min\{((y_i - \underline{x}_i)\underline{g}_i([X]), (y_i - \underline{x}_i)\overline{g}_i([X]))\},$$

$$max\{((y_i - \underline{x}_i)\underline{g}_i([X]), (y_i - \underline{x}_i)\overline{g}_i([X]))\}]$$

Or, $(y_i - \underline{x}_i) \geq 0$ donc :

$$(y_i - \underline{x}_i)g_i([X]) = (y_i - \underline{x}_i)[min\{\underline{g}_i([X]), \overline{g}_i([X])\}, \{\underline{g}_i([X]), \overline{g}_i([X])\}]$$

$$= [(y_i - \underline{x}_i)\underline{g}_i([X]), (y_i - \underline{x}_i)\overline{g}_i([X])]$$

Nous démontrons de la même façon que

$$(y_i - \underline{x}_i)g_i([X]) = [(y_i - \underline{x}_i)\overline{g}_i([X]), (y_i - \underline{x}_i)\underline{g}_i([X])]$$

Parce que $(y_i - \overline{x}_i) \leq 0$.

2.11.2 Fonctions minorantes et hyperplan d'appui

Fonction minorants sur un pavé :
Soit f une fonction définie de $[X] \subset \mathbb{R}^n$ dans $\mathbb{R}$,une fonction g définie de $[X]$ dans $\mathbb{R}$ minore f sur le pavé $[X]$, si pour tout $x \in [X]$,$f(x) \geq g(x)$

Hyperplan d'appui en un point du pavé :
Un hyperplan $\mathbb{R}$ défini sur $\mathbb{R}^n$, de fonction affine associée u, est un hyperplan d'appui en un point x du pavé $[X]$,si et seulement si :

$$\forall\ y \in [X], f(y) \geq u(y) et f(x) = u(x)$$

$\mathbb{E}^+$ sera le demi-espace associé à l'hyperplan d'appui $\mathbb{R}$, il contient par construction le graphe de f sur le pavé $[X]$.

Théorème 2.2. *En chacun des* 2^n *sommets de* $[X]$*, un hyperplan d'appui peut être construit :*

$$\forall\ x \in X, f(x) \geq f(x^S) + f(x - x^S)^T g^S([X])$$

Ou x^S *est un sommet de l'hyperplan* $[X]$ *et suivant que* $x_i^S = \underline{x}_i$*,ou* $x_i^S = \overline{x}_i$ *, on prendra* $g_i^S([X]) = \underline{g}_i(x)$ *ou* $g_i^S([X]) = \overline{g}_i(x)$*, ceci pour tout* $i \in \{1, ..., n\}$*.*

Démonstration 2.2. *Par contraction, la fonction* $f(x^S)+f(x-x^S)^T g([X])$ *est une fonction d'inclusion de* f *sur le pavé* $[X]$*.*

$$\forall x \in X, f(x) \in f(x^S) + f(x - x^S)^T g(X)$$

Or d'après la proposition 1, comme x^S *est un sommet du pavé,*

$$(x_i - x_i^S) g_i([X]) \geq (x_i - x_i^S) g_i^S([X]), \forall i \in 1, ..., n$$

Suivant que $x_i^S = \underline{x}_i$*,ou* $x_i^S = \overline{x}_i$*.*

Nous obtenons donc , $(x - x^S)^T g([X]) \geq (x - x^S)^T g^S([X])$*.*

Proposition 2.2. *Si 0 appartient à* $g_i([X])$ *, ceci pour tout* $i \in 1, ..., n$*, alors tous les hyperplans construits seront des hyperplans d'appui en chacun des différents sommets.*

Propriété 2.1. *: Chaque fonction de l'affine* u_k *satisfait les relations :*

$$\begin{cases} u_k(x^S) = & f(x^S) \\ u_k(x) \leq & f(x) \\ u_k(x^S) \geq & u_k(x) \end{cases}$$

2.12 Conclusion

Dans ce chapitre, nous avons présenté les différents éléments du calcul et de l'analyse par intervalles. En d'autres termes, nous avons décrit comment et dans quel cadre, il était possible d'effectuer des calculs non pas avec des réels mais avec des données de type intervalle. En cela, la notion de fonction d'inclusion constitue la base des algorithmes ensemblistes. Ici, nous avons choisi de présenter deux types de réalisation de fonctions d'inclusion : la fonction d'inclusion naturelle et la forme centrée. Nous avons ensuite montré que la fonction d'inclusion naturelle était efficace pour des pavés larges et volumineux, alors que la forme centrée retrouvait sa précision sur des petits pavés.

Nous avons vu aussi dans ce chapitre, l'utilisation de l'analyse d'intervalles au sein des algorithmes de Branch-and-Baund. Si l'efficacité de tels algorithmes, en point de vue obtention de l'optimum global et des optimiseurs ; n'est plus à démontrer, les difficultés engendrées par ces méthodes gênent la convergence de tels algorithmes. Nous verrons dans le chapitre suivant des algorithmes qui convergent plus rapidement, c'est les algorithmes génétiques.

Chapitre 3

Méthodes des algorithmes génétiques

3.1 introduction

Les algorithmes génétiques sont issus des travaux de John Hollond [1975] appartenant à la famille des algorithmes évolutionnistes (un sous-ensemble des métaheuristiques). Leur but est d'obtenir une solution approchée, en un temps correct, à un problème d'optimisation, lorsqu'il n'existe pas (ou qu'on ne connaît pas) une méthode exacte pour le résoudre en un temps raisonnable. Les algorithmes génétiques (AG) utilisent la notion de sélection naturelle développée au XIX^{eme} siècle par le scientifique Charles Darwin et les méthodes de combinaison de gènes introduites par Gregor Mendel au XX^{eme} siècle qui s'appliquent à une population de solutions potentielles au problème donné.[25]

Les algorithmes génétiques ont la particularité de s'inspirer de l'évolution des espèces dans leur cadre naturel. Les espèces s'adaptent à leur cadre de vie qui peut évoluer, les individus de chaque espèce se reproduisent, créant ainsi de nouveaux individus, certains subissent des modifications de leur ADN, certains disparaissent.[25]

Un algorithme génétique va reproduire ce modèle d'évolution dans le but de trouver des solutions pour un problème donné. On fera usage, alors, de termes empruntés au monde des biologistes et des généticiens et ceci afin de mieux représenter chacun des concepts abordés :

1. Dans notre cas, une population sera un ensemble d'individus.
2. Un individu sera une réponse à un problème donné, qu'elle soit ou non une solution valide du problème.
3. Un gène sera une partie d'une réponse au problème, donc d'un individu.
4. Un Génotype (ou Chromosome) : représentation sous forme de code (suite de gènes d'un individu).
5. Phénotype : représentation réelle d'un individu (instance du problème d'optimisation)
6. Locus : c'est la position du gène dans le chromosome.

7. Allèle : c'est un symbole attaché à un gène. Pour un codage binaire, un allèle renvoie à 1 ou 0.
8. Une génération est une itération de notre algorithme.

Un algorithme génétique va faire évoluer une population dans le but d'en améliorer les individus. Et c'est donc, à chaque génération, un ensemble d'individus qui sera mis en avant et non un individu particulier. Nous obtiendrons donc un ensemble de solutions ou une solution unique pour un problème donné. Les solutions trouvées seront généralement différentes, mais seront d'une qualité équivalente.[25,28]

Dans ce chapitre, nous commençons par donner le principe des algorithmes génétiques en détaillant les différents paramètres utiles pour l'implémentation de l'approche et nous terminons par citer quelques avantages et inconvénients de ces algorithmes.

3.2 Principes généraux

Dans cette partie, nous allons considérer comme problème d'optimisation, la maximisation (ou minimisation) d'une fonction objectif f. le but des AG est de déterminer l'extrémum d'une fonction $f : \mathcal{S} \longrightarrow \mathbb{R}$, où $\mathcal{S} \subseteq \mathbb{R}^n$ est un ensemble quelconque appelé espace de recherche et f appelée fonction d'adaptation ou fonction d'évaluation ou encore Fitness. Chaque élément de $\mathcal{S}$ est noté $I_i = (x_1^i, ..., x_n^i)$. Une population T_{pop} est donc un ensemble de T_{pop} individus de l'espace $\mathcal{S}$ telle que $T_{pop} = (I_1, I_2, ..., I_{N_{pop}})$.

Pour utiliser l'algorithme génétique, on doit disposer des cinq éléments suivants : [28,31]

1. Un principe de codage de l'élément de population. Cette étape associée à chacun des points de l'espace d'états une structure de données. Elle se place généralement après une phase de modélisation mathématique du problème traité. La qualité du codage des données conditionne les succès des algorithmes génétiques. Les codages binaires ont été très utilisés à l'origine. Les codages réels sont désormais largement utilisés, notamment dans les domaines applicatifs pour l'optimisation des problèmes à variables réelles.
2. Un mécanisme de génération de la population initiale doit être capable de produire une population d'individus non homogène qui servira de base pour les générations futures. Le choix de la population initiale est important car il peut rendre plus ou moins rapide la convergence vers l'optimum global. Dans le cas où l'on ne connait rien du problème à résoudre, il est essentiel que la population initiale soit répartie sur le domaine de recherche.
3. Une fonction à optimiser. Celle-ci retourne une valeur appelée fitness ou fonction d'évaluation de l'individu.

4. Des opérations permettant de diversifier la population au cours des générations et d'explorer l'espace d'état. L'opérateur de croisement recompose des gènes d'individus existant dans la population. L'operateur de mutation a pour but de garantir l'exploration de l'espace d'états.
5. Des paramètres de dimensionnement : taille de la population, nombre total de générations ou critère d'arrêt, probabilités d'application des opérations de croisement et de mutation.

La fonction d'adaptation est propre à chaque type de problème. Elle devrait tenir compte de la représentation choisie et de la nature des opérateurs afin de pouvoir donner des indications non trompeuses sur la progression vers l'optimum. Cependant, dans le cas des problèmes industriels, l'évaluation de la fonction d'adaptation consomme de loin la plus grande part de temps de calcul durant une optimisation évolutionnaire. Il faudrait donc veiller à la simplifier autant que possible afin de réduire le temps de calcul nécessaire.

Il y a beaucoup de choses qui peuvent être implémentées différemment dans divers problèmes. Les points qui différent un algorithme à un autre sont les suivants :

Que représente le chromosome ? Quel type de codage utilise-t-on ? Quel type de croisement et de mutation à utiliser ? Comment sélectionner des parents pour croisement ? Une chose qui peut être faite de différentes manières, mais l'idée principale est de choisir les meilleurs parents (dans l'espérance que les meilleurs parents vont produire les meilleurs enfants), toutes ces questions vont être discutés dans ce qui va suivre.

Le schéma de la Figure illustre la structure générale d'un algorithme génétique.

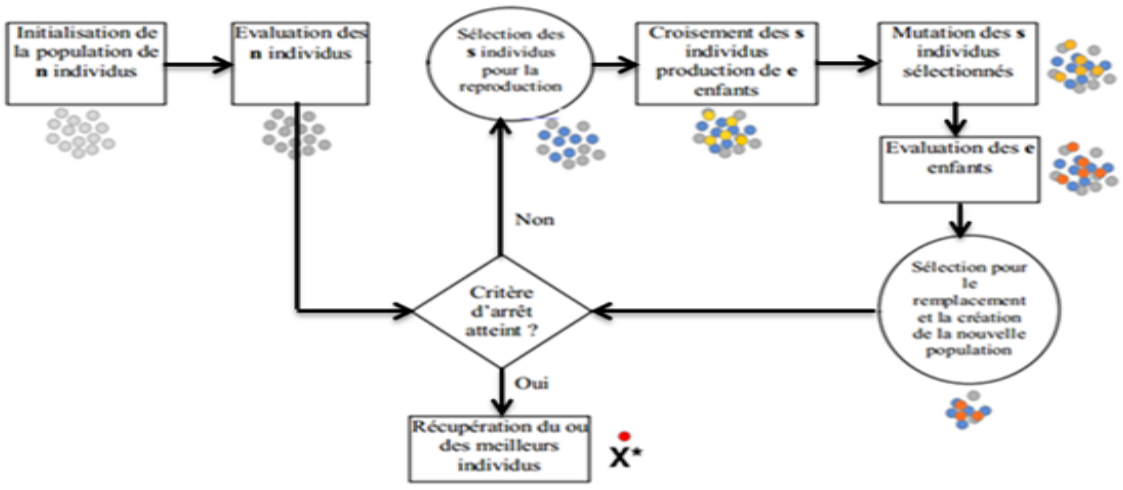


FIG. 3.1 – Principe du fonctionnement d'un algorithme génétique.

3.3 Codage des individus

Le premier travail donc à être réalisé afin de pouvoir appliquer les opérateurs, chaque solution doit être complétement définie par un vecteur numérique X.

$$Individu\ X = \boxed{x_1}\boxed{x_2}\boxed{x_3}\boxed{...}\boxed{x_{n-1}}\boxed{x_n}$$

n : Nombre de paramètres décrivant le problème.
Evaluation d'un individu :$F(X) = F(x_1, x_2, ..., x_n)$

Chaque variable de contrôle réelle x_i, on définit une borne inférieure x_i^{min} et une borne supérieure x_i^{max} au domaine de variation.

3.3.1 Codage binaire

Historiquement, le codage choisi était le codage binaire représenté sous forme de chaînes de bits (0 ou 1) contenant toute l'information nécessaire à la description d'un point dans l'espace d'état. Ce type de codage a pour intérêt de permettre de créer des opérateurs de croisement et de mutation simples, et on peut facilement coder toutes sortes d'objets : des réels, des entiers, des valeurs booléennes, des chaînes de caractères...etc.[30]

pour passer d'une représentation à l'autre, le codage/décodage de chaque paramètre $x_i \in [x_i^{min}, x_i^{max}]$ s'effectue en trois étapes :[28,30]

1. Recherche du nombre de bits l_{x_i} nécessaire au codage de la variable x_i suivant la précision souhaitée $Prec$ (nombre de chiffre après la virgule) tel que l_{x_i} est le plus petit entier vérifiant :

$$(x_i^{max} - x_i^{min}) * 10^{Prec} \leq 2^{l_{x_i}} - 1$$

Le nombre de bits l_{x_i} se calcule suivant la fonction que nous avons programmé sous Matlab :

Function $[l_{x_i}] = Nbit(x_i^{min}, x_i^{max}, Prec)$

1. $l_{bit} = (x_i^{max} - x_i^{min}) * prec_{x_i} + 1;$
2. $j = 1; prod = 2;$
3. $while(prod < l_{bit})$
4. $\quad prod = prod * 2;$
5. $\quad j = j + 1;$
6. end
7. $l_{x_i} = j$

2. Le nombre total de bits d'un chromosome, caractérisant un individu et un point dans l'espace de recherche est alors :

$$l_x = \sum_{i=1}^{n} l_{x_i}$$

n : Le nombre de paramètres (nombre de variables)

Pour chaque paramètre x_i situé dans l'intervalle $[x_i^{min}, x_i^{max}]$, on associe une chaîne binaire $\langle b_1^i, b_2^i, ..., b_{l_{x_i}}^i \rangle$ définie sur l_{x_i} bits. A cette chaîne correspond une valeur entière naturelle :

$$\langle b_1^i, b_2^i, ..., b_{l_{x_i}}^i \rangle = \sum_{j=1}^{l_{x_i}} b_j^i . 2^{l_{x_i}-1} = E_{x_i}$$

3. Pour retrouver la valeur réelle des variables x_i codées en binaire, une fonction de décodage γ^i de chaque gène i est définie comme suit :[28,30]

$$\gamma^i : B^{l_{x_i}} \longrightarrow [x_i^{min}, x_i^{max}]$$

$$x_i = \gamma^i(b_1^i, b_2^i, ..., b_{l_{x_i}}^i) = x_i^{min} + \frac{x_i^{max} - x_i^{min}}{2^{l_{x_i}} - 1} * E_{x_i}$$

avec :

- x_i : la valeur réelle du gène i ;
- $b_j^i \in 0, 1$: le bit du gène i à la position j ;
- l_{x_i} : le nombre de bits du gène i ;
- x_i^{min}, x_i^{max} : les bornes de l'intervalle de la variable i ;
- $B^{l_{x_i}} = (b_1^i, b_2^i, ..., b_{l_{x_i}}^i)$, base binaire dans laquelle le gène i est codé sur l_{x_i} bits.

Exemple 3.1.

On considère deux variable de contrôle, x_1 et x_2 :

x_1^{min}	x_1^{max}	Prec	l_{x_1}	E_{x_1}	x_1	x_2^{min}	x_2^{max}	Prec	l_{x_2}	E_{x_2}	x_2
0	7	0	3	1	**1**	-2	5	2	10	336	0.30

$$l_x = l_{x_1} + l_{x_2} = 13$$

Donc l'individu x (chromosome) est codé comme suit :

1	**0**	**0**	0	1	0	1	0	1	0	0	0	0

Remarque 3.1. Pour un chromosome de bits, cela se traduit par l_x tirages au sort de la valeur χ. Si $\chi \geq 0.5$, alors le bit $b_j = 1$, sinon le bit $b_j = 0$, avec $j = 1, ..., l_x$.

Cependant, le codage binaire n'est pas toujours bon pour les problèmes d'optimisation de grandes dimensions à haute précision numérique. Par exemple, avec 100 variables appartenant au domaine $[-500, 500]$ et dont une précision de 6 chiffres après la virgule est requise, la taille du chromosome est 3000 bits, occupe un grand espace mémoire. Et pour cela, les algorithmes génétiques utilisant des vecteurs réels évitent ce problème en conservant les variables du problème dans le codage de l'élément de population sans passer par le codage binaire intermédiaire.

3.3.2 Codage réel

Avec ce type de codage, chaque variable x_i (ou gène) est représentée directement par sa valeur réelle. Ce codage procure de meilleures performances pour les algorithmes génétiques, car il se traduit par une diminution des temps de calcul. En effet, les étapes de décodages des variables d'une valeur binaire en valeur décimale sont supprimées.[30]

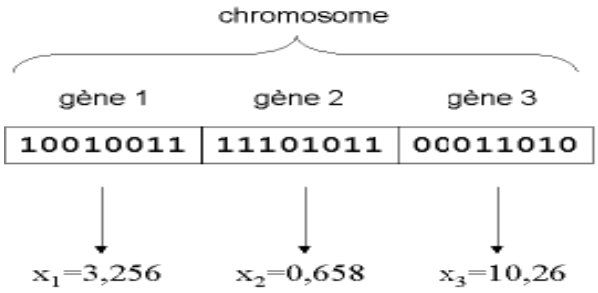


FIG. 3.2 – Codage réel

Pour générer la valeur de la variable (ou gène) x_i dans l'intervalle de recherche $[x_i^{min}, x_i^{max}]$, on utilise la formule suivante :[28]

$$x_i = x_i^{min} + (x_i^{max} - x_i^{min}) * rand \ , \quad avec \ rand \in [0, 1]$$

3.4 Initialisation de la population

L'objectif de l'étape d'initialisation est de choisir un ensemble de solutions potentielles au problème d'optimisation. En effet, chaque solution potentielle va représenter un individu (dit aussi chromosome dans la terminologie de Holland (1975)). Tous ces individus vont être rassembler dans ce que l'on nommera la population initiale.

Deux questions principales se posent lors de la construction de cette population initiale :

1. Quel doit être le nombre N_{pop} d'individus dans la population initiale ?
2. Comment doit-on générer les solutions initiales ?

Pour ce qui est du nombre d'individus (ou taille de la population), on peut à nouveau faire l'analogie avec l'évolution des espèces, il a été observé qu'une petite population peut évoluer beaucoup plus vite qu'une population plus importante. En effet, si un caractère favorable est présent chez un ou plusieurs individus de la population, ce caractère pourra rapidement se propager (par la reproduction) dans une population de taille réduite alors qu'il faudra beaucoup de temps pour qu'il se réponde dans une grande population. Cependant, une population de taille importante est un réservoir de plus grande diversité génétique qui permet une possibilité d'adaptation à une plus grande diversité de situations environnementales. Dans le cas des AG, nous devons également réaliser un compromis entre deux objectifs contradictoires : minimiser le temps de calcul et limiter le risque d'obtenir un optimum local. Concernant le temps de calcul, il dépend évidemment du nombre d'individus dans la population. Quant au problème des optima locaux, plus la population est grande, plus on a de chances de bien explorer l'espace des solutions et moins on a de chances d'avoir de grandes zones de cet espace inexplorées. On doit donc réaliser un équilibre entre ces deux objectifs.

Certains se sont intéressés à une détermination de la dépendance entre la taille optimale de la population et la longueur des individus (Alander [1992] propose une valeur comprise entre l_x et $2 \times l_x$) ou entre 50 et 100 individus (Mitchell, 1996).[27]

3.5 Mécanismes de sélection

3.5.1 Sélection par roulette

C'est une méthode stochastique introduite par Holland, qui exploite la métaphore d'une roulette de casino (voir Fig.3.3). Chaque individu occupe un secteur sur la roulette de casino (roulette-whell) dont l'angle est proportionnel à son indice de qualité P_s. La roue étant lancée, l'individu sélectionné est celui sur lequel la roue s'est arrêtée. Ainsi, un large secteur sur la roulette le conduira à être sélectionné avec une grande chance.[24,28,31]

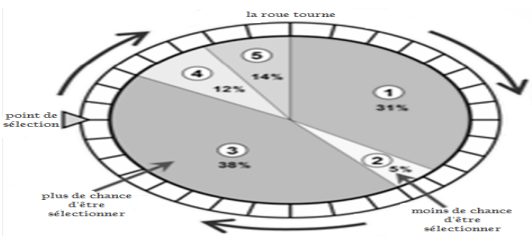


FIG. 3.3 – La roulette de casino

Soit $f(i)$ la valeur de la fitness de l'individu i ($i = 1, 2, ..., N_{pop}$). On peut alors définir $P_s(i)$, la probabilité de sélection de l'individu i, comme suit :

$$P_s(i) = \frac{f(i)}{\sum_{k=1}^{N_{pop}} f(k)} > 0$$

Une fois que les probabilités de sélection sont calculées pour l'ensemble des individus de la population, il reste à s'en servir pour construire la population suivante. La façon la plus simple de faire la sélection est d'appliquer un opérateur de sélection stochastique pur. Pour cela, on construit un segment de longueur 1. Ensuite, on calcule la position de chaque individu de la population sur ce segment en calculant les probabilités de sélection cumulées :

$$position(i) = \sum_{k=1}^{i} P_s(k)$$

Enfin, on génère aléatoirement un nombre $r \in [0, 1]$ et on reporte le nombre obtenu sur le segment pour choisir l'individu à sélectionner. Par exemple, si le nombre r choisi appartient à l'intervalle $[position(i-1), position(i)]$, on sélectionne l'individu i. Ce procédé est illustré dans l'exemple 3.2.

Exemple 3.2.

N° Individu	Chromosome	Valeurs décodées x_i	Performance $F(x) = 4.x.(1-x)$	$P_s(i)$	Intervalle prob. cumulées
1	10111010	0.7294	0.7895	0.29	[0 , 0.29]
2	11011110	0.8706	0.4507	0.18	[0.29 , 0.47]
3	00011010	0.1020	0.3665	0.15	[0.47 , 0.62]
4	01101100	0.4235	0.9766	0.38	[0.62 , 1]

TAB. 3.1 – Résultats de l'évaluation des individus dans la population initiale.

Ensuite, nous associons à chaque intervalle de probabilité un secteur équivalent de la roulette.

N° Tirage	Valeur de r de	Individus sélectionnés	Chromosome obtenue
1	0.43	X_2	11011110
2	0.89	X_4	01101100
3	0.18	X_1	10111010
4	0.75	X_4	01101100

TAB. 3.2 – Résultat de sélection.

Nous trouvons que l'individu X_3 est éliminé de la population tandis que l'individu X_4 est reproduit deux fois.

3.5.2 Sélection aléatoire

C'est la sélection la plus évidente et la plus simple à mettre en oeuvre, indépendamment de la fonction d'évaluation des individus. Si N_{pop} est la taille de la population, chaque individu aura alors une probabilité de sélection uniforme égale à $(\frac{1}{N_{pop}})$.[31]

3.5.3 Sélection par tournoi

L'avantage de cette sélection, par rapport à la sélection aléatoire, est quelle augmente les chances des individus de mauvaise qualité de participer à l'évolution de la population. D'une manière générale, M individus sont pris au hasard parmi les N_{pop} individus de la population. Ces M individus sont comparés entre eux et seul le meilleur individu est considéré comme vainqueur du tournoi.

Cette étape est répétée jusqu'à ce que la génération intermédiaire soit complétée. Il est tout à fait possible que certains individus participent à plusieurs tournois. S'ils gagnent plusieurs fois, ils auront donc le droit d'être copiés plusieurs fois dans la génération intermédiaire.

Le paramètre M est fixé a priori par l'utilisateur et joue un rôle important dans l'algorithme. En effet, si $M = N_{pop}$, l'algorithme génétique est réduit à un algorithme de recherche locale travaillant sur une seul solution. L'inconvénient est de converger parfois, rapidement vers un optimum local. Si au contraire, $M = 1$, la sélection devient alors une sélection aléatoire.

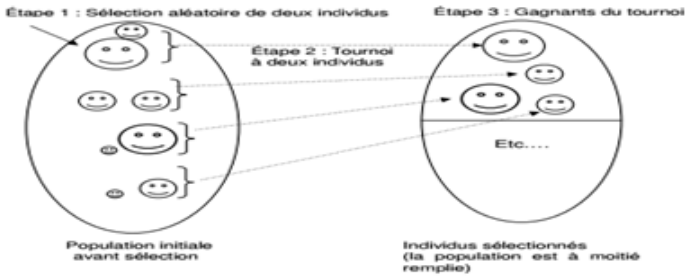


FIG. 3.4 – Représentation d'une sélection par tournoi d'individus pour un critère de maximisation.chaque individu représente une solution possible

Exemple 3.3.

Indices population	1	2	3	4	5	6	7	8	9	10
Individus	I_1	I_2	I_3	I_4	I_5	I_6	I_7	I_8	I_9	I_{10}
Coûts	3,2	0,5	0,2	1,5	2,5	0,3	0,2	0,4	1,5	0,3

TAB. 3.3 – Distribution des individus selon leur coût

Tournoi	I_1-I_6	I_2-I_5	I_3-I_7	I_1-I_5	I_8-I_{10}	I_4-I_9	I_5-I_8	I_2-I_6	I_3-I_9	I_1-I_2
Résultat	I_1	I_5	I_3	I_1	I_8	I_4	I_5	I_6	I_9	I_1

TAB. 3.4 – Exemple de sélection par tournoi avec $M = 2$ et $N_{pop} = 10$

3.5.4 $N/2$-Elitisme

Cette méthode de sélection permet de mettre en avant les meilleurs individus de la population. Ce sont donc les individus les plus prometteurs qui participent à l'amélioration de notre population. Cette méthode a l'avantage de permettre une convergence plus rapide des solutions, mais au détriment de la diversité des individus. Dans ce type de sélection, les individus sont tirés selon leur fonction de fitness. Seule la moitié supérieure de la population, correspondant aux meilleurs chromosomes, est sélectionnée.

3.5.5 Sélection par rang

La sélection par rang est une variante du système de roulette. Il s'agit également d'implémenter une roulette, mais cette fois ci les secteurs de la roue ne sont plus proportionnels à la qualité des individus, mais à leur rang dans la population triée en fonction de la qualité des individus.

D'une autre manière , il faut trier la population en fonction de la qualité des individus puis leur attribuer à chacun un rang. Les individus de moins bonne qualité obtiennent un rang faible (à partir de 1). Et ainsi, en itérant sur chaque individu, on finit par attribuer le rang N_{pop} au meilleur individu (où N_{pop} est la taille de la population) ; la suite de la méthode consiste uniquement en l'implémentation d'une roulette basée sur les rangs des individus. L'angle de chaque secteur de la roue sera proportionnel au rang de l'individu qu'il représente. Des probabilités de sélection $P_s(i)$ sont alors calculées pour chaque individu i en fonction du rang selon la formule suivante :[31]

$$P_s(i) = \frac{rang_i}{\sum_{i=1}^{N_{pop}} rang_i}$$

3.6 Opérateur de croisement

Le croisement permet, par la manipulation de la structure des chromosomes, l'enrichissement de la population. Il consiste en un échange de gènes entre deux ou plusieurs chromosomes afin d'en former de nouveaux. Classiquement, les croisements impliquent deux parents qui génèrent un ou deux enfants. C'est le croisement qui fait la force des algorithmes génétiques. Il s'agit de sélectionner deux individus parmi les parents potentiels, aléatoirement ou à l'aide d'une des méthodes de sélection pour la reproduction. Ces derniers sont croisés suivant une

probabilité de croisement p_c , afin d'obtenir de nouveaux individus appelés "enfants". Parfois les "bons" gènes d'un parent se substituent aux "mauvais" gènes de l'autre pour former un meilleur descendant. La probabilité de croisement est plus ou moins élevée dans les AG. Elle varie souvent entre $0,7$ et $0,95$.

Il existe plusieurs méthodes de croisement en fonction de la méthode de représentation des solutions.

3.6.1 Croisement binaire

3.6.2 Croisement simple à un point

Le cas le plus élémentaire est le croisement à un point. Cela consiste simplement à choisir un point m aléatoirement entre 1 et $(l_x - 1)$. Le changement va se faire entre le point sélectionné et la position finale l_x des deux chaînes comme le montre la figure 3.5.

Soient deux chromosomes parents $P^1 = (p_1^1, p_2^1, ..., p_i^1, ..., p_{l_x}^1)$ et $P^2 = (p_1^2, p_2^2, ..., p_i^2, ..., p_{l_x}^2)$ et deux chromosomes enfants $E^1 = (e_1^1, e_2^1, ..., e_i^1, ..., e_{l_x}^1)$ et $E^2 = (e_1^2, e_2^2, ..., p_i^2, ..., e_{l_x}^2)$, avec l_x, longueur du chromosome et p_i^1, $p_i^1 \in \{0, 1\}$. L'opérateur de croisement est défini par $C_{\{P_c\}}$:

$$(E^1, E^2) = C_{\{P_c\}}(P^1, P^2)$$

Telque :

$$si\ \chi \leq P_c =: \begin{cases} e_i^1 = p_i^1 \ \ et \ \ e_i^2 = p_i^2 & | \quad i \in [1, m] \\ e_i^1 = p_i^2 \ \ et \ \ e_i^2 = p_i^1 & | \quad i \in [m+1, l_x] \end{cases}$$

$$si\ \chi > P_c =: \begin{cases} e_i^1 = p_i^1 \ \ pour & | \quad i \in [1, l_x] \\ e_i^2 = p_i^2 \ \ pour & | \quad i \in [1, l_x] \end{cases}$$

Avec $\chi \in [0, 1]$, nombre aléatoire généré avec une probabilité uniforme.[24,28]

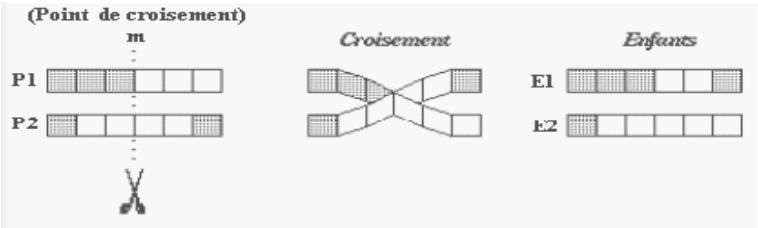


FIG. 3.5 – Croisement à un point

3.6.3 Croisement en deux points

Afin d'améliorer les performances des AG, il est recommandé d'utiliser plusieurs points de croisement, mais en pratique, le nombre le plus courant est 2 (voir Fig. 3.6).

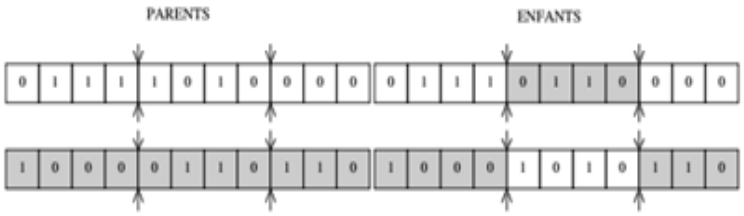


FIG. 3.6 – Croisement en deux point

3.6.4 Croisement réel

3.6.5 Croisement standard

Le croisement réel standard est très proche de celui décrit pour le codage binaire. Il consiste à sélectionner 1 ou plusieurs points de croisement compris entre 1 et $(N-1)$ et d'échanger les paramètres compris entre deux points pour la formation des enfants. Il ne se différencie du croisement binaire que par la nature des éléments qu'il altère.

3.6.6 Croisement arithmétique

Le croisement arithmétique est propre à la représentation réelle. Il s'pplique à une paire de chromosomes et se résume à une moyen pondéré des gènes des deux chromosomes.
Soient $X^1 = (x_1^1, ..., x_i^1, ..., x_n^1)$ et $X^2 = (x_1^2, ..., x_i^2, ..., x_n^2)$ deux parent et $\lambda \in [0, 1]$, leurs enfants $Y^1 = (x_1^1, ..., y_i^1, ..., y_n^1)$ et $Y^2 = (y_1^2, ..., y_i^2, ..., y_n^2)$ sont obtenus de la suivante :

$$\left.\begin{array}{l}(x_1^1, ..., x_i^1, ..., x_n^1)\\(x_1^2, ..., x_i^2, ..., x_n^2)\end{array}\right\} \Rightarrow \begin{array}{ll}(y_1^1, ..., y_i^1, ..., y_n^1) & avec \;\; y_i^1 = \lambda.x_i^1 + (1-\lambda).x_i^2\\(y_1^2, ..., y_i^2, ..., y_n^2) & avec \;\; y_i^2 = \lambda.x_i^2 + (1-\lambda).x_i^1\end{array} \tag{3.1}$$

- Si $\lambda = 1$ on recopie intégralement le paramètre du parent1 pour l'enfant1 et du parent2 pour l'enfant2.
- Si $\lambda = 0.5$ on prend la moyenne des deux valeurs de paramètres des parents.

Cette modification des valeurs des paramètres peut s'effectuer sur tous les gènes ou seulement sur une partie. Par exemple, si on introduit aléatoirement un point de croisement, le changement est effectué seulement pour tous paramètres situés à gauche ou à droite de ce point.[24,28]

3.7 Opérateur de mutation

Dans la nature, la mutation introduit une certaine variabilité dans le processus d'évolution qui peut donner à certains individus une chance, au départ négligeable, d'évoluer vers la solution optimale. L'opérateur de mutation apporte aux algorithmes génétiques l'aléa nécessaire à une exploration efficace de l'espace. Cet opérateur nous garantit que l'algorithme génétique sera susceptible d'atteindre tous les points de l'espace d'état, sans pour autant les parcourir tous dans le processus de résolution. La mutation a pour but de permettre aux AG de ne pas rester sur des optimums locaux au cours de l'évolution.

De ce fait, le taux de mutation est fortement dépendant de la nature et de la forme des fonctions objectives et donc de la fonction d'adaptation. Plus elle comporte de minimums locaux, plus l'AG a besoin d'une probabilité de mutation importante au début de l'évolution.[24,28]

3.7.1 Mutation binaire

Elle consiste à échanger un seul bit d'un gène (voir Fig.3.7). Pour cela, on définit une probabilité de mutation P_m qu'un bit subisse une mutation. Ainsi, on génère aléatoirement $[P_m \times N_{pop} \times l_x]$ couples de points appelés points de mutation définissant respectivement l'individu et le bit à muter. L'opération de mutation consiste à appliquer aux chromosomes de chaque individu la fonction suivante :

Soient deux chromosomes $A = (a_1, ..., a_i, ..., a_{l_x})$ et $D = (d_1, ..., d_i, ..., d_{l_x})$, l_x étant la longeur du chromosome et $a_i, d_i \in \{0, 1\}$, $i = 1, ..., l_x$. L'opérateur de mutation est $M_{\{P_m\}} : B^{l_x} \longrightarrow B^{l_x}$. Alors $D = M_{\{P_m\}}(A)$ tel que :

$$d_s = \begin{cases} a_s & si\ \chi > P_m \\ 1 - a_s & si\ \chi \leq P_m \end{cases}$$

s étant l'indice du point de mutation (choisi de façon aléatoire).

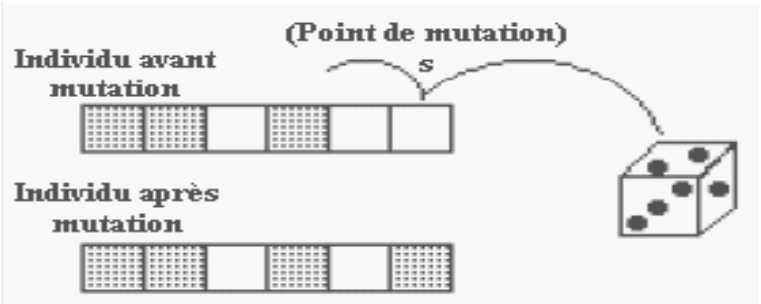


FIG. 3.7 – mutation dans un chromosome

3.7.2 Mutation réelle

Dans le cas d'une mutation réelle, ce n'est plus un bit qui est inversé, mais une variable réelle qui est de nouveau tirée au hasard sur son intervalle de définition.

$$gene_{(aprsmutation)} = gene_{(avantmutation)} + \delta$$

alors :

$$\delta = r \times \sigma$$

Où r est un nombre aléatoire compris entre $[-1, 1]$, et σ l'écart-type du même gène dans la population.[28]

Exemple 3.4.

Soit $11,656$, le nouveau nombre tirée au hasard sur l'intervalle de définition, et $r = -1$, et $\sigma = 2.13 = 11,656 - 9,526$ Alors :

Enfant avant mutation	5.390	8.143	**9.526**	3.991	8.620	8.167
Enfant après mutation	5.390	8.143	**7.396**	3.991	8.620	8.167

TAB. 3.5 – Exemple de mutation réelle uniforme

3.8 L'insertion des nouveaux individus dans la population

Une fois que nous avons créé de nouveaux individus que ce soit par croisement ou par mutation, il nous faut sélectionner ceux qui vont continuer à participer à l'amélioration de notre population. Une fois encore, libre au programmeur de choisir ceux qu'il souhaite conserver. Il est possible de refaire une étape d'évaluation des individus nouvellement créés. De même qu'il est possible de conserver tous les nouveaux individus en plus de notre population. Il n'est toutefois pas recommandé de ne conserver que les nouveaux individus meilleurs que les individus de départ.

Une méthode relativement efficace consiste à insérer les nouveaux individus dans la population, à trier cette population selon l'évaluation de ses membres, et à ne conserver que les N_{pop} meilleurs individus.

Une méthode efficace est de toujours garder la même taille de la population d'une génération à l'autre. Ainsi, il est possible de dérouler l'algorithme sur un grand nombre de générations.

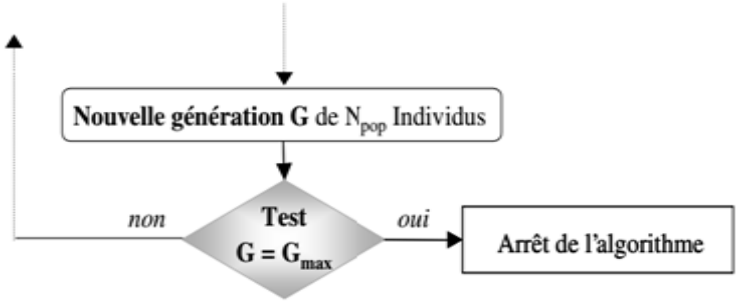
3.9 Critères d'arrêt

Le critère d'arrêt indique que la solution est suffisamment approchée de l'optimum. Trois grands types de critères d'arrêt sont généralement employés :[24]

- **On fixe un nombre de génération total :** G_{max}

L'A.G. s'arrête lorsque le nombre de génération maximal G_{max} est atteint.
G_{max} grand =: convergence atteinte mais trop long.
G_{max} petit =: convergence non atteinte.

Ce critère peut s'avérer coûteux en temps de calcul si le nombre d'individus à traiter dans chaque population est important.

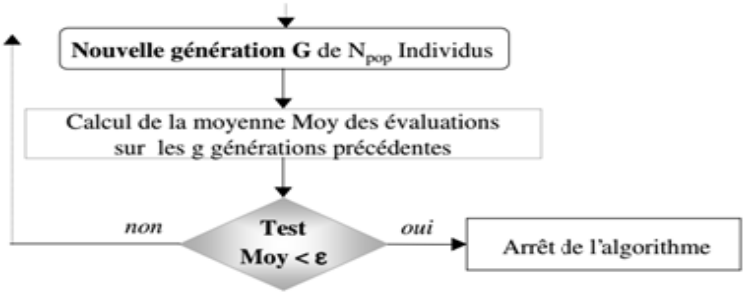


- **On utilise le phénotype :**

Cette méthodes conservent des traces statistiques des différentes générations (moyenne des évaluations d'une population, performance du meilleur individu de chaque génération, écart type,...) et s'en servent pour le test d'arrêt.

Exemple 3.5.

On mesure les progrès réalisés par les individus sur un nombre prédéfini g de générations que l'on compare à une valeur ε fixe à l'avance.

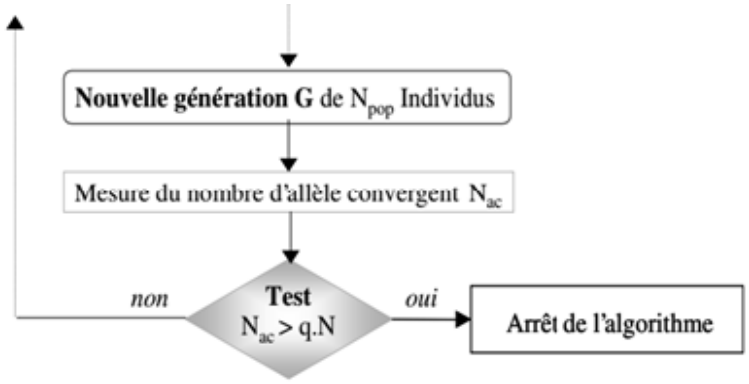


- **On utilise le génotype (structure de chromosome)**

Une des méthodes repose sur la mesure d'allèles convergents.

(Allèle convergent : un allèle est considéré convergent lorsqu'un pourcentage prédéterminé de la population possède la même valeur d'allèle).

L'A.G. s'arrête lorsque le nombre d'allèles convergents noté N_{ac} atteint un certain pourcentage q du nombre total N d'allèles.



Remarque 3.2. d'autres critères peuvent être appliquées pour déterminer l'arrêt de l'AG tels que :

- l'amélioration de la solution ne dépasse plus un certain seuil.
- la fonction objectif du problème atteint une valeur donnée.
- le temps de calcul atteint une valeur prédéterminer.

3.10 Avantages et inconvénients

Les algorithmes génétiques sont des outils efficaces pour une classe de problèmes très large. Leurs avantages sont :

- il n'y a pas de contraintes sur les fonctions à optimiser (dérivabilité, continuité,...etc.).
- ils peuvent donner plusieurs solutions.
- leurs performances par rapport au algorithmes classiques sont bien remarquées lorsque par exemple les espaces de recherches sont importants.
- Outre leur facilité de programmation et de manipulation, ils sont facilement adaptables à tout type de problème d'optimisation.
- ils peuvent être utilisés avec profit pour traiter des problèmes n'étant pas optimisables efficacement par des approches purement mathématiquement.
- Ils peuvent parcourir rapidement un grand ensemble de solutions.

- La nature inductive des AG signifie qu'il ne doit connaître aucune règle du problème.
- Ils travaillent avec leurs propres règles internes.

Néanmoins, ils présentent aussi, un certain nombre de limitations :

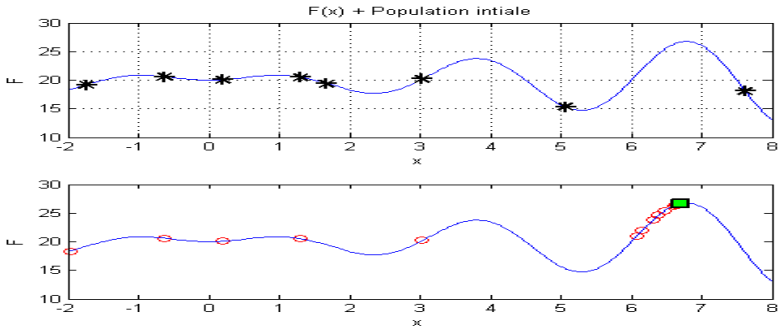
- les paramètres de réglage (telles la taille de la population, la probabilité de croisement,...) sont parfois difficiles à déterminer or le succès de l'évolution en dépend. Plusieurs essais sont donc nécessaires.
- Ils ne garantissent pas toujours la découverte de l'optimum global en un temps fini. En effet, lorsqu'une population évolue, il se peut que certains individus occupant à un instant une place importante au sein de cette population deviennent majoritaires. A ce moment, il se peut que la population converge vers cet individu en s'écartant ainsi d'individus plus intéressants et en s'éloignant de l'individu vers lequel on devrait converger.
- Ils sont moins efficaces qu'un algorithme déterministe spécifique (lorsqu'il en existe un)
- Les nombreux paramètres qui les contrôlent sont délicats à régler (probabilités de croisement et de mutation notamment, ainsi que le codage des chromosomes qui peut faire varier radicalement la vitesse de convergence).

3.11 Exemple d'application

On se propose de déterminer le maximum de la fonction f suivante (on prendra $f = F$ comme fonction d'adaptation) dépendante d'un seul paramètre x.

$f(x) = 20 + x.sin(\frac{2\pi}{3}.x)$ pour $-2 \leq x \leq 8$

$N = 1$, Précision : 10^{-2}
$N_{pop} = 8$, $P_c = 0.75$
$P_m = 0.05$, Stratégie : élitisme
Sélection : tournoi , Critère d'arrêt : $G_{max} = 20$



Codage des paramètres :

- $l_x = 10$
- $P_m \times N_{pop} \times l_x = 4$ mutation par génération

Population initiale :

N°	Individus	Génotype	Phénotype (F(x))
1	0 1 0 0 0 1 0 1 0 1	0.7135	20.7114
2	0 0 1 0 1 1 0 1 0 0	-02340	20.1101
3	1 1 0 0 0 0 1 0 1 1	5.6186	15.9744
4	1 0 1 1 0 1 1 1 0 1	5.1662	14.9131
5	0 1 1 0 0 0 0 1 0 1	1.8118	18.8993
6	1 0 1 1 1 0 0 1 0 0	5.2403	14.7608
7	1 1 1 1 1 0 0 1 0 0	7.7451	16.1967
8	0 1 1 0 1 0 0 0 1 0	2.0886	18.0295

Tri des individus selon leurs évaluations :

N°	Individus	Phénotype (F(x))
1	0 1 0 0 0 1 0 1 0 1	20.7114
2	0 0 1 0 1 1 0 1 0 0	20.1101
3	0 1 1 0 0 0 0 1 0 1	18.8993
4	0 1 1 0 1 0 0 0 1 0	18.0295
5	1 1 1 1 1 0 0 1 0 0	16.1967
6	1 1 0 0 0 0 1 0 1 1	15.9744
7	1 0 1 1 0 1 1 1 0 1	14.9131
8	1 0 1 1 1 0 0 1 0 0	14.7608

Construction du groupe de reproducteurs :

N°	Individus	Phénotype (F(x))
1	0 1 0 0 0 1 0 1 0 1	20.7114
2	0 0 1 0 1 1 0 1 0 0	20.1101
3	0 1 1 0 0 0 0 1 0 1	18.8993
4	0 1 1 0 1 0 0 0 1 0	18.0295

Sélection par tournoi :

On effectue 4 tournois pour sélectionner 2 couples

1er tournoi : 1 1 $\longrightarrow$ 1	0	1	0	0	0	1	0	1	0	1
2ème tournoi : 1 3 $\longrightarrow$ 1	0	1	0	0	0	1	0	1	0	1
3ème tournoi : 3 3 $\longrightarrow$ 3	1	1	0	0	0	0	1	0	1	1
4ème tournoi : 4 2 $\longrightarrow$ 2	0	0	1	0	1	1	0	1	0	0

Croisement :

$S = 3$: Point de croisement tiré pour la 1^{er} couple

$S = 9$: Point de croisement tiré pour le 2^{me} couple

N°	4 nouveaux enfants sont crées	Génotype (x)	Phénotype (F(x))
1	0 1 0 0 0 1 0 1 0 1	0.7077	20.7049
2	0 1 0 0 0 1 0 1 0 1	0.7077	20.7049
3	0 1 1 0 0 0 0 1 0 0	1.7928	18.9682
4	0 0 1 0 1 1 0 1 0 1	-0.2307	20.1072

Construction de la population intermédiaire :

On ajoute au groupe de reproducteurs les quatre enfants crées auxquels on associe leurs efficacités :

N°	Individus	Génotype (x)	Phénotype (F(x))
1	0 1 0 0 0 1 0 1 0 1	0.7135	20.7114
2	0 0 1 0 1 1 0 1 0 0	-0.2340	20.1101
3	0 1 1 0 0 0 0 1 0 1	1.8118	18.8993
4	0 1 1 0 1 0 0 0 1 0	2.0886	18.0295
1	0 1 0 0 0 1 0 1 0 1	0.7077	20.7049
2	0 1 0 0 0 1 0 1 0 1	0.7077	20.7049
3	0 1 1 0 0 0 0 1 0 0	1.7928	18.9682
4	0 0 1 0 1 1 0 1 0 1	-0.2307	20.1072

Mutation :

On réalise 4 mutations par tirage au sort de 4 couples de points :

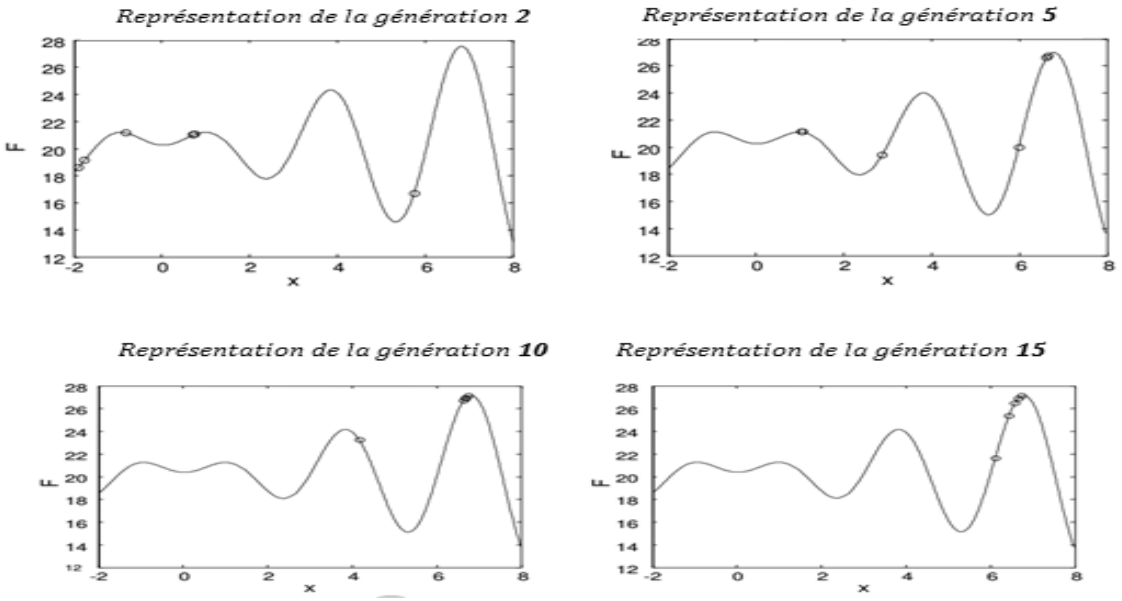
N°	Pt mut	Individus avant mutation	Génotype (x)	Phénotype (F(x))
1	1	0 1 0 0 0 1 0 1 0 1	0.7077	20.7049
2	2	0 1 1 0 0 0 0 1 0 1	1.8118	18.8993
3	4	0 0 1 0 1 1 0 1 0 0	-0.2340	20.1101
4	3	0 0 1 0 1 1 0 1 0 0	-0.2340	20.1101

N°	Individus après mutation	Génotype (x)	Phénotype (F(x))
1	1 1 0 0 0 1 0 1 0 1	5.7126	16.7654
2	0 0 1 0 0 0 0 1 0 1	-0.6999	20.5951
3	0 0 1 1 1 1 0 1 0 0	-0.3851	20.2781
4	0 0 0 0 1 1 0 1 0 0	-1.4917	20.0250

Tri :

N°	Présentation de la génération 1	Génotype (x)	Phénotype (F(x))
1	0 0 0 1 1 1 0 1 0 0	-0.8661	20.8406
2	0 1 0 0 0 1 0 1 0 1	0.7077	20.7049
3	0 1 0 0 0 1 0 1 0 1	0.7077	20.7049
4	0 0 1 0 0 0 0 1 0 1	-0.6999	20.6961
5	0 0 1 0 1 1 0 1 0 1	-0.2307	20.1072
6	0 1 1 0 0 0 0 1 0 0	1.7928	18.9683
7	0 1 1 0 1 0 0 0 1 0	2.0860	180358
8	1 1 0 0 0 1 0 1 0 1	5.7126	16.7654

Evolution suivant les générations :



Composition et représentation de la dernière génération :

N°	Présentation de dernière génération	Génotype (x)	Phénotyp(F(x))e
1	1101111111	6.7488	26.7488
2	1101111111	6.7488	26.7488
3	1101111111	6.7488	26.7488
4	1111111111	8.0000	13.0718
5	1101111111	6.7488	26.7488
6	1101111011	6.7097	26.6858
7	1101011100	6.4066	24.8203
8	1001111101	4.2268	22.2888

$X^* = 6.7488$ et $F(X^*) = 26.7488$

Mémorisation au cours de l'évolution de l'AG :

- Il est utile de mémoriser le meilleur individu au cours des générations plutôt que de se contenter de la population finale.
- Moyenne des évaluations augmentent bien au cours des générations.

3.12 Conclusion

Malheureusement, les algorithmes génétiques seuls ne sont pas très efficaces dans la résolution d'un problème. Ils apportent cependant assez rapidement une solution acceptable. Néanmoins, il est possible de les améliorer assez efficacement en les combinant avec un algorithme de Branch-and-Bound d'Intervalles (analyse d'intervalles) présentée dans le deuxième chapitre.

Chapitre 4

Hybridation

4.1 introduction

La résolution d'un problème d'optimisation consiste à explorer un espace de recherche afin de maximiser (ou minimiser) une fonction donnée. Pour être performante, une méthode de résolution doit associer judicieusement exploration et exploitation de l'espace de recherche. Or cette méthode est rarement aussi efficace pour exploiter que pour explorer l'espace de recherche. Cette difficulté due aux complexités (en taille ou en structure) relatives de l'espace de recherche et de la fonction à minimiser (ou maximiser) conduit à des méthodes de résolutions radicalement différentes. Une première approximation consiste à dire qu'une méthode déterministe est adaptée à un espace de recherche petit et complexe et qu'un espace de recherche grand nécessite plutôt une méthode de recherche stochastique (algorithme génétique,...). Une solution consiste à ajouter des mécanismes complémentaires dans une méthode de résolution donnée. Il peut être extrêmement bénéfique d'associer une méthode de recherche dont les caractéristiques d'exploration sont très élevées à une méthode de recherche dont le point fort est l'exploitation. D'où l'idée de méthodes hybrides.

Les méthodes hybrides permettent non seulement d'élargir le spectre d'application de certaines méthodes de résolution mais aussi d'augmenter leur performances. Pour appliquer efficacement ces techniques, nous devons avoir une bonne visibilité vis-à-vis de points forts de chaque méthodes à part. par exemple, les algorithmes génétiques sont très performants lorsqu'il s'agit d'explorer l'espace de recherche, mais ils s'avèrent ensuite incapables d'exploiter efficacement la zone vers laquelle la population converge. Il est alors bien plus intéressant (en terme de durée d'exécution et de qualité de solutions) de stopper l'algorithme génétique pour utiliser une autre méthode. Hybridation peut, aussi, être utilisée pour résoudre simultanément différents aspects d'un même problème : cette méthode est souvent utilisée dans le domaine de la gestion de production [[36], [37]] où se posent simultanément diffents problèmes tels que les affectations de machines, des personnels et des opérations aussi bien que la gestion de stocks...

4.2 Choix des méthodes à hybrider

Il est possible d'hybrider toutes les technique, y compris méthodes exacte et heuristique. Pratiquement, le souci de performances et les ressources informatiques limitent les possibilités d'hybridation. De ce fait, on doit être prudent vis-à-vis des techniques utilisées pour obtenir une bonne coopération entre les constituants et les méthodes hybrides.

4.3 Les techniques hybridations

L'hybridation peut prendre place dans un ou plusieurs composants d'une méthode de recherche. Elle peut également consister à assembler plusieurs méthodes d'hybridation pour former une seule méthode hybride. Les différentes techniques d'hybridation peuvent être réparties en trois catégories principales [35] :

- Hybridation séquentielle ;
- Hybridation parallèle synchrone ;
- Hybridation parallèle asynchrone.

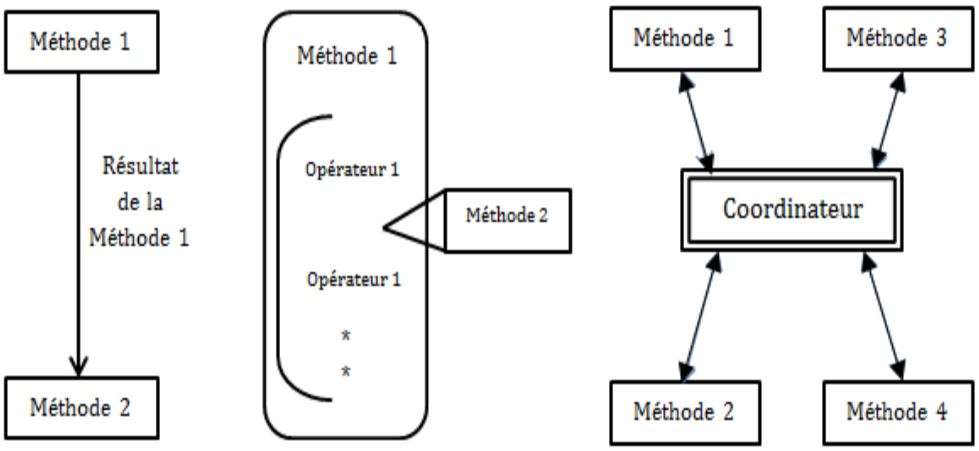


FIG. 4.1 – Techniques d'hybridation.

L'hybridation séquentielle consiste à exécuter séquentiellement différentes méthodes de recherche de telle façon que le (ou les) résultat(s) d'une méthode serve(nt) de solution(s) initiale(s) à la suivante. Cette technique d'hybridation est la plus simple, elle ne nécessite pas de modification des méthodes de résolution utilisées : il suffit d'initialiser chaque méthode à partir de solutions pré-calculées.

L'hybridation parallèle synchrone est obtenue en incorporant une méthode de résolution dans une autre. C'est une technique plus fine que la précédente. En effet, il faut tenir compte fortes interactions entre les méthodes dans ce type d'hybridation.

L'hybridation parallèle asynchrone consiste à faire évoluer en parallèle différentes méthodes de résolution. Cette co-évolution permet une bonne coopération des méthodes de résolution au travers d'un coordinateur chargé d'assurer le transfert d'informations entre les méthodes de résolution ; cette technique exige la modification de méthodes de résolution pour assurer la communication avec le coordinateur.

4.4 Le schéma d'hybridation proposée (HIG)

Dans le cadre de nos travaux, nous avons applique la première technique d'hybridation (algorithme génétique d'intervalle hybride), nous recourons à l'arithmétique d'intervalle et particulièrement un algorithme branch-and-bound d'intervalle est employé pour obtenir de petites régions où la (ou les) solution(s) optimale(s) se trouve(nt). De cette façon, une population des solutions possibles est initialisée et des bornes initiales pour le minimum global f^* sont obtenues.

Dans l'ordre, un algorithme génétique est appliqué de telle manière que toutes les informations ci-dessus soient exploitées. La construction d'un mécanisme qui met à jour les bornes dans chaque génération, donne la capacité de définir un critère efficace d'arrêt. Quand le critère est rempli, l'algorithme converge vers le minimum global f^*.

Le schéma suivant décrit l'algorithme.

ALGORITHME 3 : Algorithme génétique d'intervalle hybride, HIG.

1. Appliquer un algorithme de subdivision ;
2. Initialiser la population ;
3. Evaluer chacun individu de la population
4. Tantque (le critère d'arrêt non atteint) faire{
5. Mettre à jour des bornes
6. Choisir les individus pour la prochaine population ;
7. Appliquer les opérations génétiques pour produire des nouveaux individus ;
8. Evaluer les nouveaux individus ;}
9. Afficher f^* et x^*.

les bornes $\underline{F^*}$ et $\overline{F^*}$ qui sont obtenus par l'étape 1 de l'algorithme ci-dessus satisfisant la relation suivante :

$$\underline{F^*} \leq f^* \leq \overline{F^*}$$

4.5 Conclusion

Dans ce chapitre nous avons présenté une approche hybride qui est HIG. Cela a permis de construire un algorithme plus efficace pour l'optimisation globale des fonctions non convexes. Les résultats obtenus montrent comment cette technique converge vers l'optimum global dans un temps d'exécution minimum. La comparaison avec les résultats obtenus avec un algorithme génétique, permet de conclure que la technique développée dans ce chapitre est satisfaisante en termes de temps nécessaire pour atteindre l'optimum global.

Chapitre 5

Implémentations et exemple d'application

Dans ce chapitre, nous avons testé l'algorithme HIG de manière réelle grace au logiciel de programmation MATLAB. Un programme informatique écrit à l'aide du langage de programmation MATLAB mettant en oeuvre cet algorithme, a été testé avec succés sur une application numérique.

Avant d'entamer l'application numérique, nous allons donner un petit aperçu sur le logiciel MATLAB, qui est une abréviation de MATrix LABoratory. MATLAB est avant tout un programme de calcul matriciel et il peut être un logiciel de programmation en utilisant les m.files (fichiers avec extension .m), où on peut écrire notre algorithme en suivant les mots clés proposés par la console, ensuite il suffit de l'exécuter et là le résultat sera affiché sur la fenêtre de travail.

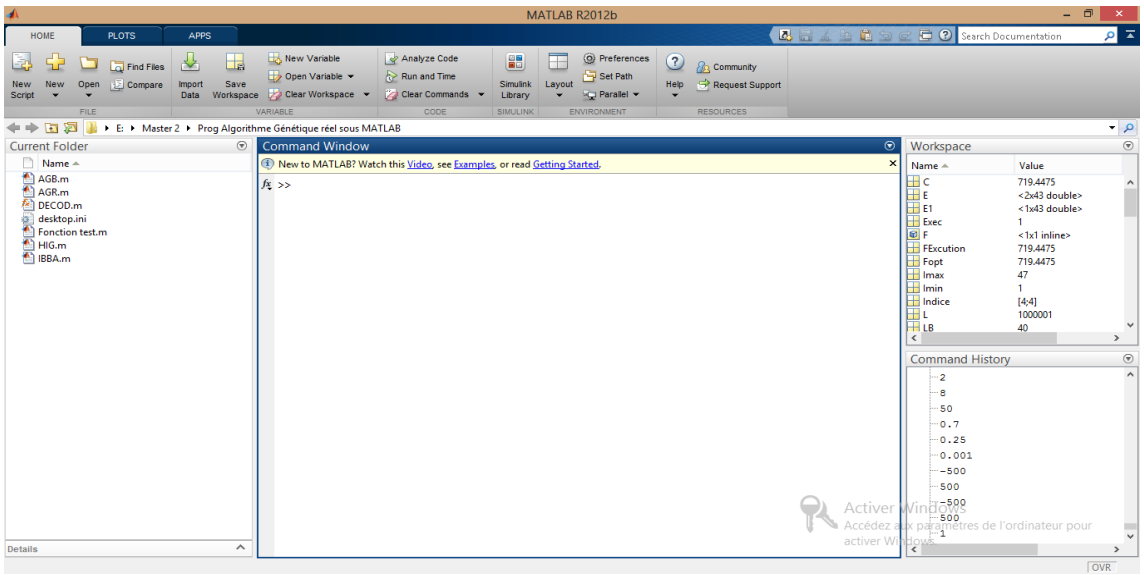


FIG. 5.1 – Fenêtre principale de MATLAB 2012.

5.1 Implémentations sous MATLAB

5.1.1 Branch-and-Bound par intervalle

```
clear all
clc
format long
disp('@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@')
disp('@                                                                     @')
disp('@       BRANCH-AND-BOUND PAR INTERVALLE DE RESOLUTION                 @')
disp('@         D"UN PROBLEME D"OPTIMISATION SOUS FORME:                    @')
disp('@                   MIN(ou MAX) F(X)                                  @')
disp('@                   S.C:                                              @')
disp('@                       Xmin(i) <= x(i) <= Xmax(i)                    @')
disp('@                                                                     @')
disp('@                   1. Minimisation                                   @')
disp('@                   2. Maximisation                                   @')
disp('@                                                                     @')
disp('@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@')
fobj=input('Donnez la fonction F :','s');
f=inline([fobj])
Nbvar=input('Donnez le nombre de variable du problème:');
for i=1:Nbvar
    disp(['Domaine de la variable X(',num2str(i),'):'])
    Xmin(i)=input(['Xmin(',num2str(i),')=']);
    Xmax(i)=input(['Xmax(',num2str(i),')=']);
end
epsXsup=input('Donnez la valeur de espX:');
Fsup=realmax;
%Intervalle Q <----- X0
k=0;
for j=1:Nbvar
    H(1,j+k)=Xmin(j);
    H(1,j+k+1)=Xmax(j)
    k=k+1;
end
L=[];
s=j+k+1;
H(1,s)=-inf;
a=1;
 while(H~=[])
     Y=H(1,:);
     X=H(1,1:s-1);
     for k=1:Nbvar
         j=1;
         for i=1:Nbvar
             midpoint(i)=(X(k,j)+X(k,j+1))./2;
             j=j+2;
         end
         midpoint
         switch(Nbvar)
             case 1
                 E=f(midpoint(Nbvar))
             case 2
                 E=f(midpoint(1),midpoint(Nbvar))
             case 3
                 E=f(midpoint(1),midpoint(2),midpoint(Nbvar))
```

```
        case 4
            E=f(midpoint(1),midpoint(2),midpoint(3),midpoint(Nbvar))
        case 5

            E=f(midpoint(1),midpoint(2),midpoint(3),midpoint(4),midpoint(
            Nbvar))
    end
end
Fsup=min([Fsup,E])
ligne=size(H);
if(ligne(1))==1)
    if(H(1,s)>Fsup)
        H=[]
    end
else
    % Supprimer de H tous les couples pour lesquels le minorant garanti
    est supérieur à Fsup
    % Trier la matrice H par ordre croissant
    for i=1:ligne(1)-1
        for j=i+1:ligne(1)
            if(H(i,s)>H(j,s))
                inter=H(j,s);
                H(j,s)=H(i,s);
                H(i,s)=inter;
            end
        end
    end
    % recherche l'élément supérieur à Fsup
    position=0;
    for i=1:ligne(1)
        if(H(i,s)>Fsup)
            position=i;
        end
    end
    T=H;
    % SUPRESSION
    if(position~=0)
        clear 'H'
        H=T(1:position-1,:);
    end
end
% Calculer la largeur de de l'intervalle X
posi=1;
maxi=abs(X(2)-X(1))
for i=2:Nbvar
    j=2*i-1;
    M=abs(X(j+1)-X(j))
    if(maxi<M)
        maxi=M;
        posi=j;
    end
end
Xsup=[X(posi),X(posi+1)];
if(abs(Xsup(2)-Xsup(1))<=epsXsup)
    % Ajouter [X] à la liste L des intervalles résultats
    L(a,:)=Y;
    a=a+1;
```

```
    else
        ligne=size(H);
        X12(1,:)=X;
        X12(2,:)=X;
        X12(1,posi+1)=(Xsup(2)+Xsup(1))/2;
        X12(2,posi)=(Xsup(2)+Xsup(1))/2;
        X12
        % Calculer les minorants garantis LB1 et LB2 pour [X1] et [X2]
        LB1=min([f(X12(1,1)),f(X12(1,2))])
        LB2=min([f(X12(2,1)),f(X12(2,1))])
        if(LB1<=Fsup)
           % Ajouter [X1] à la liste Q
           if(ligne(1)==0)
               Q=X12(1,:);
           else
               Q(ligne(1)+1,:)=X12(1,:);
           end
        end
        if(LB2<=Fsup)
            % Ajouter [X1] à la liste Q
            if(ligne(1)==0)
                Q=X12(2,:);
            else
                Q(ligne(1)+1,:)=X12(2,:);
            end
        end
    end
end
% Supprimer de H tous les couples pour lesquels le minorant garanti est
supérieur à Fsup
% Trier la matrice H par ordre croissant
for i=1:ligne(1)-1
    for j=i+1:ligne(1)
        if(H(i,s)>H(j,s))
            inter=H(j,s);
            H(j,s)=H(i,s);
            H(i,s)=inter;
        end
    end
end
% Recherche l'élément supérieur à Fsup
position=0;
for i=1:ligne(1)
    if(H(i,s)>Fsup)
        position=i;
    end
end
T=H;
% SUPRESSION
if(position~=0)
    clear 'H'
    H=T(1:position-1,:);
end
disp('Liste qui contient tout les minimums :')
L
```

```
function X=DECOD(LX,U,BI,BS)
% LX : NOMBRE DE BIT NECESSAIRE POUR CODE LA VARIABLE X
% U : CHROMOSOME (VECTEUR BINAIRE)
% BI : BORNE INF DE LA VARIABLE X
% BS : BORNE SUP DE LA VARIABLE X
% X : LA VALEUR REELLE DE CHROMOSOME
X=0;
for i=1:LX
    X=X+U(i)*2^(LX-i);
end
X=BI+((BS-BI)/(2^(LX)-1))*X;
End
```

5.1.2 Algorithme génétique

5.1.2.1.Algorithme génétique binaire

```
clear all
clc
format long
disp('@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@')
disp('@                                                                  @')
disp('@          ALGORITHME GENETIQUE BINAIRE DE RESOLUTION              @')
disp('@           D"UN PROBLEME D"OPTIMISATION SOUS FORME :              @')
disp('@                    MIN(ou MAX) F(X)                              @')
disp('@                    S.C:                                          @')
disp('@                        Xmin(i) <= x(i) <= Xmax(i)                @')
disp('@                                                                  @')
disp('@                    1. Minimisation                               @')
disp('@                    2. Maximisation                               @')
disp('@                                                                  @')
disp('@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@')
choix=input('VOTRE CHOIX (1/2):');
fobj=input('ENTEZ LA FONCTION OBJECTIF F(x):','s');
F=inline([fobj])
NV=input('NOMBRE DE VARIABLES DU PROBLEME :');
NPop=input('NOMBRE D"INDIVIDUS DE LA POPULATION INTIALE :');
NGen=input('NOMBRE DE GENERATIONS :');
Pc=input('PROBABILITE DE CROISEMENT :');
Pm=input('PROBABILITE DE MUTATION :');
Prec=input('PRECISION SOUHAITE DES VARIABLES :');
Moitpop=NPop/2;
Quart_pop=NPop/4;
for i=1:NV
   disp(['DOMAINE ADMISSIBLE DE LA VARIABLE X(',num2str(i),'):'])
    Xmin(i)=input(['Xmin(',num2str(i),')=']);
    Xmax(i)=input(['Xmax(',num2str(i),')=']);
end
NEP=input('NOMBRE D"EXECUTION DE PROGRAMME :');
% TROUVER LE NOMBRE DE BIT POUR CODER CHAQUE VARIABLE
for i=1:NV
    L=1+(Xmax(i)-Xmin(i))/Prec;
    j=1;
    P=2;
    while(P<L)
```

```
        P=P*2;
        j=j+1;
    end
    LXB(i)=j;
end
LXB
% TAILLE DE CHROMOSOME
LB=sum(LXB)
for Exec=1:NEP   % NOMBRE D'EXECUTION DE PROGRAMME
    % INITIALISATION DE LA POPULATION
    %pop=round(rand(NPop,LB+NV+1));
    for i=1:NPop
        for j=1:LB
            if(rand<0.5)
                pop(i,j)=0;
            else
                pop(i,j)=1;
            end
        end
        k=1;
        t=0;
        for h=1:NV
            t=t+LXB(h);
            pop(i,LB+h)=DECOD(LXB(h),pop(i,k:t),Xmin(h),Xmax(h));
            k=t+1;
        end
        switch(NV)
            case 1
                pop(i,LB+NV+1)=F(pop(i,LB+NV));
            case 2
                pop(i,LB+NV+1)=F(pop(i,LB+1),pop(i,LB+NV));
            case 3
                pop(i,LB+NV+1)=F(pop(i,LB+1),pop(i,LB+2),pop(i,LB+NV));
            case 4

               pop(i,LB+NV+1)=F(pop(i,LB+1),pop(i,LB+2),pop(i,LB+3),pop(i,LB+N
               V));
            case 5
               pop(i,LB+NV+1)=F(pop(i,LB+1),pop(i,LB+2),pop(i,LB+3),pop(i,LB+4
               ),pop(i,LB+NV));
        end
    end
    % VISUALISATION DE LA POPULATION INTIALE DANS LE GRAPHE (2D)
    if(NV==1)
        figure(1)
        fplot(F,[Xmin Xmax]);
        grid
        hold on

        plot(pop(:,LB+1),pop(:,LB+2),'ko','LineWidth',2,'MarkerEdgeColor','k',
        'MarkerFaceColor','k','MarkerSize',5)
        hold off
    else
        if(NV==2)
            x=linspace(Xmin(1), Xmax(1),150);
                y=linspace(Xmin(2), Xmax(2),150);
                [X Y]=meshgrid(x,y);
```

```
            Z=F(X,Y);
            %surfl(X,Y,Z)
            shading interp
            colormap jet
            view(-37.5+90, 30)
            mesh(X,Y,Z)
            meshc(X,Y,Z)
            grid
            hold on

             plot3(pop(:,LB+1),pop(:,LB+2),pop(:,LB+3),'ko','LineWidth',2,
             'MarkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',10)
            hold off
    end
end
% EVOLUTION
for i=1:NGen
    if(choix==1)
        % CALCULE DU MINIMUM DE FITNESSMIN A LA GENERATION i
        [fitnessmin(i),Imin]=min(pop(:,LB+NV+1))
        % SOLUTION A LA GENERATION i
        Sol(i,:)=pop(Imin,LB+1:LB+NV)
        % TRIER LA POPULATION PAR ORDRE DECROISSANT (PROBLEME DE MIN)
        for i=1:NPop-1
            for j=i+1:NPop
                if(pop(i,LB+NV+1)>pop(j,LB+NV+1))
                    inter=pop(j,:);
                    pop(j,:)=pop(i,:);
                    pop(i,:)=inter;
                end
            end
        end
        % CONSTRUCTION DU GROUPE DE REPRODUCTEURS
        group_sel(1:Moitpop,1:LB+NV+1)=pop(1:Moitpop,1:LB+NV+1)
        % SELECTION PAR TOURNOI
        for i=1:Moitpop
            % CHOISIR ALEATOIREMENT 2 ENTIER ENTRE 1 ET Moitpop
            Indice=random('Discrete Uniform',Moitpop,2,1);
            if(group_sel(Indice(1),LB+NV+1)<=group_sel(Indice(2),LB+NV+1))
                Selection(i)=Indice(1);
            else
                Selection(i)=Indice(2);
            end
        end
    else
        % CALCULE DE MAXIMUM DE FITNESSMIN A LA GENERATION i
        [fitnessmax(i),Imax]=max(pop(:,LB+NV+1));
        % SOLUTION A LA GENERATION i
        Sol(i,:)=pop(Imax,LB+1:LB+NV);
        % TRIER LA POPULATION PAR ORDRE CROISSANT (PROBLEME DE MAX)
        for i=1:NPop-1
            for j=i+1:NPop
                if(pop(i,LB+NV+1)<pop(j,LB+NV+1))
                    inter=pop(j,:);
                    pop(j,:)=pop(i,:);
                    pop(i,:)=inter;
                end
```

```
        end
    end
    % CONSTRUCTION DU GROUPE DE REPRODUCTEURS
    group_sel(1:Moitpop,1:LB+NV+1)=pop(1:Moitpop,1:LB+NV+1)
    % SELECTION PAR TOURNOI
    for i=1:Moitpop
        % CHOISIR ALEATOIREMENT 2 ENTIER ENTRE 1 ET Moitpop
        Indice=random('Discrete Uniform',Moitpop,2,1);
        if(group_sel(Indice(1),LB+NV+1)<=group_sel(Indice(2),LB+NV+1))
            Selection(i)=Indice(2);
        else
            Selection(i)=Indice(1);
        end
    end
end
s=0;
k=1+Moitpop;
for i=1:Quart_pop
    % DEBUT DE CROISEMENT
    P1=group_sel(Selection(s+1),1:LB+NV+1);
    P2=group_sel(Selection(s+2),1:LB+NV+1);
    if (rand<Pc)
        % GENERER ALEATOIRE UN POINT DE CROISEMENT ENTRE 1 ET LB-1
        PTC=random('Discrete Uniform',LB-1,1,1)
        E=[P1(1:PTC) P2(PTC+1:LB);P2(1:PTC),P1(PTC+1:LB)];
        for i=1:2
            m=1;
            t=0;
            for j=1:NV
                t=t+LXB(j);
                E(i,LB+j)=DECOD(LXB(j),E(i,m:t),Xmin(j),Xmax(j));
                m=t+1
            end
            switch(NV)
                case 1
                    E(i,LB+NV+1)=F(E(i,LB+1));
                case 2
                    E(i,LB+NV+1)=F(E(i,LB+1),E(i,LB+NV));
                case 3
                    E(i,LB+NV+1)=F(E(i,LB+1),E(i,LB+2),E(i,LB+NV));
                case 4

                    E(i,LB+Nbvar+1)=F(E(i,LB+1),E(i,LB+2),E(i,LB+3),E(
                    i,LB+NV));
                case 5

                    E(i,LB+NV+1)=F(E(i,LB+1),E(i,LB+2),E(i,LB+3),E(i,L
                    B+4),E(i,LB+NV));
            end
        end
    else
        E=[P1;P2];
    end
    % FIN DE CROISEMENT
    s=s+2;
    group_sel(k:k+1,1:LB+NV+1)=E;
    k=k+2;
```

```
end
for a=1:round(NPop*Pm*LB)
    % GENERER ALEATOIREMENT UN ENTIER ENTRE 1 ET NPop
    Num_Indiv=random('Discrete Uniform',NPop,1,1)
    % P EST L'INDIVIDU SELECTIONNER
    P=group_sel(Num_Indiv,1:LB+NV+1);
    if (rand<Pm)
        %GENERER ALEATOIREMENT UN POINT DE MUTATION ENTRE 1 ET LB
        PTM=random('Discrete Uniform',LB,1,1)
        E1=P;
        E1(PTM)=1-P(PTM);
        k=1;
        t=0;
        for j=1:NV
            t=t+LXB(j);
            E1(LB+j)=DECOD(LXB(j),E1(k:t),Xmin(j),Xmax(j));
            k=t+1;
        end

        switch(NV)
            case 1
                E1(LB+NV+1)=F(E1(LB+NV));
            case 2
                E1(LB+NV+1)=F(E1(LB+1),E1(LB+NV));
            case 3
                E1(LB+NV+1)=F(E1(LB+1),E1(LB+2),E1(LB+NV));
            case 4
                E1(LB+NV+1)=F(E1(LB+1),E1(LB+2),E1(LB+3),E1(LB+NV));
            case 5

                E1(LB+NV+1)=F(E1(LB+1),E1(LB+2),E1(LB+3),E1(LB+4),E1(L
                B+NV));
        end
        % REMPLACER LE PARENT PAR SON ENFANT
        group_sel(Num_Indiv,:)=E1;
    end
end
% REMPLACER L'ANCIENNE POPULATION PAR LA NOUVELLE
pop=group_sel;
if(NV==1)
    figure(i+1)
    fplot(F,[Xmin Xmax]);
    grid
    hold on

   plot(pop(:,LB+1),pop(:,LB+2),'ro','LineWidth',2,'MarkerEdgeColor',
    'r','MarkerFaceColor','r','MarkerSize',5)
    hold off
else
    % DESSINER LE GRAPHE DE F (3D)
    if(NV==2)
        figure(i+1)
        x=linspace(Xmin(1), Xmax(1),150);
        y=linspace(Xmin(2), Xmax(2),150);
        [X Y]=meshgrid(x,y);
        Z=F(X,Y);
        %surfl(X,Y,Z)
```

```
                shading interp
                colormap jet
                view(-37.5+90, 30)
                mesh(X,Y,Z)
                meshc(X,Y,Z)
                grid
                hold on

               plot3(pop(:,LB+1),pop(:,LB+2),pop(:,LB+3),'ko','LineWidth',2,'M
               arkerEdgeColor','k','MarkerFaceColor','k','MarkerSize',10)
                hold off
            end
        end
    end
    % FIN DE LA BOUCLE GENERATION
    if(choix==1)
        [FExcution(Exec),Imin]=min(fitnessmin);
        Optimum(Exec,:)=Sol(Imin,:);
    else
        [FExcution(Exec),Imax]=max(fitnessmax);
        Optimum(Exec,:)= Sol(Imax,:);
    end
end
if(choix==1)
    disp('LA SOLUTION DU PROBLEME EST :')
    [C,Imin]=min(FExcution);
    Xopt=Optimum(Imin,:)
    Fopt=C
    if(NV==1)
        hold on

        plot(Xopt,Fopt,'go','LineWidth',1,'MarkerEdgeColor','g','MarkerFaceCol
        or','g','MarkerSize',7)
    else
        if(NV==2)
            hold on

            plot3(Xopt(1),Xopt(2),Fopt,'ro','LineWidth',2,'MarkerEdgeColor','r
            ','MarkerFaceColor','r','MarkerSize',15)
        end
    end
else
    disp('LA SOLUTION DU PROBLEME EST :')
    [C,Imin]=max(FExcution);
    Xopt=Optimum(Imin,:)
    Fopt=C
    if(NV==1)
        hold on

        plot(Xopt,Fopt,'go','LineWidth',2,'MarkerEdgeColor','g','MarkerFaceCol
        or','g','MarkerSize',7)
    else
        if(NV==2)
            hold on

            plot3(Xopt(1),Xopt(2),Fopt,'ro','LineWidth',2,'MarkerEdgeColor','r
            ','MarkerFaceColor','r','MarkerSize',15)
```

```
        end
    end
end
```

5.1.2.2.Algorithme génétique réel

```
clear all
clc
format long
disp('@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@')
disp('@                                                                   @')
disp('@          ALGORITHME GENETIQUE REEL DE RESOLUTION                  @')
disp('@          D"UN PROBLEME D"OPTIMISATION SOUS FORME :                @')
disp('@                    MIN(ou MAX) F(X)                               @')
disp('@                    S.C:                                           @')
disp('@                        Xmin(i) <= x(i) <= Xmax(i)                 @')
disp('@                                                                   @')
disp('@                    1. Minimisation                                @')
disp('@                    2. Maximisation                                @')
disp('@                                                                   @')
disp('@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@')
choix=input('VOTRE CHOIX (1/2):');
% PARAMETRES D'ENTRE
fobj=input('ENTEZ LA FONCTION OBJECTIF F(x):','s');
F=inline([fobj])
NV=input('NOMBRE DE VARIABLES DU PROBLEME :');
NPop=input('NOMBRE D"INDIVIDUS DE LA POPULATION INTIALE :');
NGen=input('NOMBRE DE GENERATIONS :');
Pc=input('PROBABILITE DE CROISEMENT :');
Pm=input('PROBABILITE DE MUTATION :');
Moitpop=NPop/2;
Quart_pop=NPop/4;
for i=1:NV
    disp(['DOMAINE ADMISSIBLE DE LA VARIABLE X(',num2str(i),'):'])
    Xmin(i)=input(['Xmin(',num2str(i),')=']);
    Xmax(i)=input(['Xmax(',num2str(i),')=']);

end
NEP=input('NOMBRE D"EXECUTION DE PROGRAMME :');
for Exec=1:NEP   % NOMBRE D'EXECUTION DE PROGRAMME
    % INITIALISATION DE LA POPULATION
    for i=1:NPop
        for j=1:NV
            pop(i,j)=(Xmax(j)-Xmin(j))*rand+Xmin(j);
        end
        switch(NV)
            case 1
                pop(i,NV+1)=F(pop(i,NV));
            case 2
                pop(i,NV+1)=F(pop(i,1),pop(i,NV));
            case 3
                pop(i,NV+1)=F(pop(i,1),pop(i,2),pop(i,NV));
            case 4
                pop(i,NV+1)=F(pop(i,1),pop(i,2),pop(i,3),pop(i,NV));
            case 5
                pop(i,NV+1)=F(pop(i,1),pop(i,2),pop(i,3),pop(i,4),pop(i,NV));
```

```
    end
end
% VISUALISATION DE LA POPULATION INTIALE DANS LE GRAPHE (2D)
if(NV==1)
    figure(1)
    fplot(F,[Xmin Xmax]);
    grid
    hold on

    plot(pop(:,NV),pop(:,NV+1),'ko','LineWidth',2,'MarkerEdgeColor','k','M
    arkerFaceColor','k','MarkerSize',5)
    hold off
else
    if(NV==2)
          x=linspace(Xmin(1), Xmax(1),150);
            y=linspace(Xmin(2), Xmax(2),150);
            [X Y]=meshgrid(x,y);
            Z=F(X,Y);
            %surfl(X,Y,Z)
            shading interp
            colormap jet
            view(-37.5+90, 30)
            mesh(X,Y,Z)
            meshc(X,Y,Z)
            grid
            hold on

           plot3(pop(:,1),pop(:,2),pop(:,NV+1),'ko','LineWidth',2,'MarkerE
           dgeColor','k','MarkerFaceColor','k','MarkerSize',10)
            hold off
    end
end
% EVOLUTION
for i=1:NGen
    if(choix==1)
        % CALCULE DE MINIMUM DE FITNESSMIN A LA GENERATION i
        [fitnessmin(i),Imin]=min(pop(:,NV+1));
        % SOLUTION A LA GENERATION i
        Sol(i,:)=pop(Imin,1:NV);
        % TRIER LA POPULATION PAR ORDRE DECROISSANT (PROBLEME DE MIN)
        for i=1:NPop-1
            for j=i+1:NPop
                if(pop(i,NV+1)>pop(j,NV+1))
                    inter=pop(j,:);
                    pop(j,:)=pop(i,:);
                    pop(i,:)=inter;
                end
            end
        end
        % CONSTRUCTION DU GROUPE DE REPRODUCTEURS
        group_sel(1:Moitpop,1:NV+1)=pop(1:Moitpop,1:NV+1);
        % SELECTION PAR TOURNOI
        for i=1:Moitpop
            % CHOISIR ALEATOIREMENT 2 ENTIER ENTRE 1 ET Moitpop
            Indice=random('Discrete Uniform',Moitpop,2,1);
            if(group_sel(Indice(1),NV+1)<=group_sel(Indice(2),NV+1))
                Selection(i)=Indice(1);
```

```
        else
            Selection(i)=Indice(2);
        end
    end
else
    % CALCULE DE MAXIMUM DE FITNESSMIN A LA GENERATION i
    [fitnessmax(i),Imax]=max(pop(:,NV+1));
    % SOLUTION A LA GENERATION i
    Sol(i,:)=pop(Imax,1:NV);
    % TRIER LA POPULATION PAR ORDRE CROISSANT (PROBLEME DE MAX)
    for i=1:NPop-1
        for j=i+1:NPop
            if(pop(i,NV+1)<pop(j,NV+1))
                inter=pop(j,:);
                pop(j,:)=pop(i,:);
                pop(i,:)=inter;
            end
        end
    end
    % CONSTRUCTION DU GROUPE DE REPRODUCTEURS
    group_sel(1:Moitpop,1:NV+1)=pop(1:Moitpop,1:NV+1);
    % SELECTION PAR TOURNOI
    for i=1:Moitpop
       % CHOISIR ALEATOIREMENT 2 ENTIER ENTRE 1 ET Moitpop
        Indice=random('Discrete Uniform',Moitpop,2,1);
        if(group_sel(Indice(1),NV+1)<=group_sel(Indice(2),NV+1))
            Selection(i)=Indice(2);
        else
            Selection(i)=Indice(1);
        end
    end
end
s=0;
k=1+Moitpop;
for i=1:Quart_pop
    % DEBUT DE CROISEMENT
    P1=group_sel(Selection(s+1),1:NV+1);
    P2=group_sel(Selection(s+2),1:NV+1);
    if (rand<Pc)
        lamdaa=rand;
        % GENERER ALEATOIRE UN POINT DE CROISEMENT ENTRE 1 ET NV-1
        if(NV==1)
            E1=lamdaa*P1+(1-lamdaa)*P2;
            E2=lamdaa*P2+(1-lamdaa)*P1;
        else
            PTC=random('Discrete Uniform',NV-1,1,1);
            E1(1:PTC)=P1(1:PTC);
            E2(1:PTC)=P2(1:PTC);
            for j=PTC+1:NV
                E1(j)=lamdaa*P1(j)+(1-lamdaa)*P2(j);
                E2(j)=lamdaa*P2(j)+(1-lamdaa)*P1(j);
            end
            switch(NV)
                case 1
                    E1(NV+1)=F(E1(NV));
                    E2(NV+1)=F(E2(NV));
                case 2
```

```
                    E1(NV+1)=F(E1(1),E1(NV));
                    E2(NV+1)=F(E2(1),E2(NV));
                case 3
                    E1(NV+1)=F(E1(1),E1(2),E1(NV));
                    E2(NV+1)=F(E2(1),E2(2),E2(NV));
                case 4
                    E1(NV+1)=F(E1(1),E1(2),E1(3),E1(NV));
                    E2(NV+1)=F(E2(1),E2(2),E2(3),E2(NV));
                case 5
                    E1(NV+1)=F(E1(1),E1(2),E1(3),E1(4),E1(NV));
                    E2(NV+1)=F(E2(1),E2(2),E2(3),E2(4),E2(NV));
            end
        end
        E=[E1;E2];
    else
        E=[P1;P2];
    end
    % FIN DE CROISEMENT
    s=s+2;
    group_sel(k:k+1,1:NV+1)=E;
    k=k+2;
end
for a=1:round(Pm*NPop*NV)
    % GENERER ALEATOIREMENT UN ENTIER ENTRE 1 ET NPop
    Num_Indiv=random('Discrete Uniform',NPop,1,1)
    % P EST L'INDIVIDU SELECTIONNER
    P=group_sel(Num_Indiv,:);
    if (rand<Pm)
        %GENERER ALEATOIREMENT UN POINT DE MUTATION ENTRE 1 ET NV
        PTM=random('Discrete Uniform',NV,1,1)
        ENF=P;
        ENF(PTM)=Xmin(PTM)+(Xmax(PTM)-Xmin(PTM))*rand
        switch(NV)
            case 1
                ENF(NV+1)=F(ENF(NV));
            case 2
                ENF(NV+1)=F(ENF(1),ENF(NV));
            case 3
                ENF(NV+1)=F(ENF(1),ENF(2),ENF(NV));
            case 4
                ENF(NV+1)=F(ENF(1),ENF(2),ENF(3),ENF(NV));
            case 5
                ENF(NV+1)=F(ENF(1),ENF(2),ENF(3),ENF(4),ENF(NV));
        end
        % REMPLACER LE PARENT PAR SON ENFANT
        group_sel(Num_Indiv,:)=ENF;
    end
end
% REMPLACER L'ANCIENNE POPULATION PAR LA NOUVELLE
pop=group_sel;
if(NV==1)
    figure(i+1)
    fplot(F,[Xmin Xmax]);
    grid
    hold on
```

```
            plot(pop(:,1),pop(:,2),'ro','LineWidth',2,'MarkerEdgeColor','r','M
            arkerFaceColor','r','MarkerSize',5)
            hold off
        else
            % DESSINER LE GRAPHE DE F (3D)
            if(NV==2)
                figure(i+1)
                x=linspace(Xmin(1), Xmax(1),150);
                y=linspace(Xmin(2), Xmax(2),150);
                [X Y]=meshgrid(x,y);
                Z=F(X,Y);
                %surfl(X,Y,Z)
                shading interp
                colormap jet
                view(-37.5+90, 30)
                mesh(X,Y,Z)
                meshc(X,Y,Z)
                grid
                hold on

               plot3(pop(:,1),pop(:,2),pop(:,NV+1),'ko','LineWidth',2,'MarkerE
               dgeColor','k','MarkerFaceColor','k','MarkerSize',10)
                hold off
            end
        end
    end
    % FIN DE LA BOUCLE GENERATION
    if(choix==1)
        [FExcution(Exec),Imin]=min(fitnessmin);
        Optimum(Exec,:)=Sol(Imin,:);
    else
        [FExcution(Exec),Imax]=max(fitnessmax);
        Optimum(Exec,:)= Sol(Imax,:);
    end
end
if(choix==1)
    disp('LA SOLUTION DU PROBLEME EST :')
    [C,Imin]=min(FExcution);
    Xopt=Optimum(Imin,:)
    Fopt=C
    if(NV==1)
        hold on

        plot(Xopt,Fopt,'go','LineWidth',1,'MarkerEdgeColor','g','MarkerFaceCol
        or','g','MarkerSize',7)
    else
        if(NV==2)
            hold on

            plot3(Xopt(1),Xopt(2),Fopt,'ro','LineWidth',2,'MarkerEdgeColor','r
            ','MarkerFaceColor','r','MarkerSize',10)
        end
    end
else
    disp('LA SOLUTION DU PROBLEME EST :')
    [C,Imin]=max(FExcution);
```

```
    Xopt=Optimum(Imin,:)
    Fopt=C
    if(NV==1)
        hold on

        plot(Xopt,Fopt,'go','LineWidth',1,'MarkerEdgeColor','g','MarkerFaceCol
        or','g','MarkerSize',7)
    else

        if(NV==2)
            hold on

            plot3(Xopt(1),Xopt(2),Fopt,'ro','LineWidth',2,'MarkerEdgeColor','r
            ','MarkerFaceColor','r','MarkerSize',10)
        end
    end
end
```

5.1.5 HIG

Pour l'algorithme hybride, on applique l'algorithme Branch-and-Bound par intervalles tel qu'en subdivisent les intervalles, puis il intervient l'algorithme génétique pour améliorer la borne supérieure. On aura à la fin une borne inférieure donnée par l'algorithme Branch-and-Bound par intervalles et une borne supérieur donné par l'algorithme génétique.

5.2 Exemple d'application

5.2.1 Problème à résoudre

On se limite à la résolution dans le plan horizontal, avec des avions volant à une vitesse constante sans incertitude. La première étape est de calculer les positions susceptibles d'être occupées par un avion à un instant donné quand ses manœuvres prennent place à l'intérieur d'un intervalle de temps.

5.2.2 Modélisation des trajectoires d'évitement

La trajectoire d'évitement utilisée est illustrée par la figure 5.1 et comporte quatre étapes :

1. Jusqu'à un temps t_0 , l'avion reste sur sa trajectoire d'origine.

2. Au temps t_0 , l'avion quitte sa trajectoire, vers la droite ou vers la gauche,mais avec un angle α fixé. Il poursuit ensuite sa trajectoire déviée avec un cap constant jusqu'à un temps t_1.

3. Au temps t_1, l'avion prend un cap de retour tel que sa trajectoire de retour fasse avec sa trajectoire d'origine un angle égal à l'angle de déviation α pris par l'avion au temps t_0. L'utilisation que nous présentons ici de cette modélisation de l'évitement reste valable si cet angle de retour a une valeur différente de celle de l'angle selon lequel l'avion a quitté sa trajectoire. Il est nécessaire cependant que ces deux angles aient des valeurs fixées : cette méthode repose sur le fait que l'ensemble des positions possibles d'un avion, quand ces temps de manœuvres prennent leurs valeurs dans des intervalles, ont des formes simples . Cela n'est plus le cas si l'angle de retour vers la trajectoire d'origine n'est plus constant. Dans toute la suite, ces deux angles seront égaux, ce qui permettra d'obtenir des expressions mathématiques plus simples pour les retards et les positions des avions.

4. L'avion rejoint sa trajectoire d'origine à un temps t_2 , reprend son cap d'origine et poursuit ensuite sa route sur sa trajectoire d'origine.

Ainsi définie, la trajectoire d'évitement d'un avion, l'angle α étant fixé, est donc entièrement déterminée par, le sens de la déviation (à droite ou à gauche) et les instants t_0 et t_1 ($t_2 = 2t_1 - t_0$).
On vérifiera que $t_1 \geq t_0$. Le cas limite $t_1 = t_0$ correspond à une trajectoire non déviée. t_0 et t_1 sont remplacés par des intervalles, l'angle α prend des valeurs discrètes, selon le sens de la déviation. Le choix du sens de déviation est intégré à l'algorithme d'optimisation.

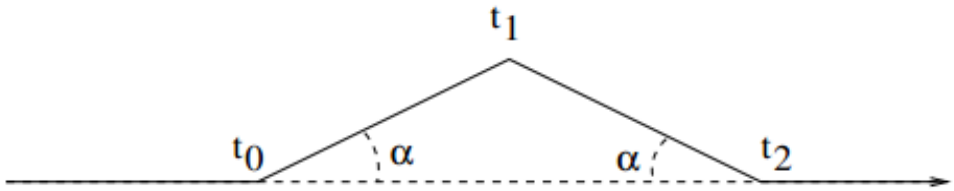


FIG. 5.2 – Modélisation d'une trajectoire d'évitement par point tournant.

5.2.3 Position de l'avion à un temps t

On considère un repère orthonormé dans le plan, tel que l'avion soit sur l'origine O de ce repère au début de l'évitement. Pour t_0 et t_1 fixés, la position $P(t)$ de l'avion à l'instant t dépend de la position de t relativement à t_0, t_1 et t_2. On distingue quatre cas, correspondant aux quatre étapes de la trajectoire d'évitement :

1. $t \leq t_0$. On a alors :

$$\begin{cases} x(t) = & v.t \\ y(t) = & 0 \end{cases} \tag{5.1}$$

   L'avion n'a pas encore été dévié.

2. $t_0 \leq t \leq t_1$. On a alors :

$$\begin{cases} x(t) = & v(t_0(1 - cos(\alpha)) + tcos\alpha) \\ y(t) = & (t - t_0)sin\alpha \end{cases} \tag{5.2}$$

   L'avion a été dévié, et s'écarte de sa trajectoire d'origine, avec l'angle α.

3. $t_1 \leq t \leq t_2$. On a alors :

$$\begin{cases} x(t) = & v(t_0(1 - cos(\alpha)) + tcos\alpha) \\ y(t) = & (2t_1 - t_0 - t)sin\alpha \end{cases} \tag{5.3}$$

   L'avion revient sur sa trajectoire d'origine, avec un angle de retour égal à α.

4. $t \geq t_2$. On a alors :

$$\begin{cases} x(t) = & v(t - 2(t_1 - t_0)(1 - cos\alpha)) \\ y(t) = & 0 \end{cases} \tag{5.4}$$

L'avion est revenu sur sa trajectoire d'origine.

Une fois l'avion revenu sur sa trajectoire d'origine, on peut calculer le retard entraîné par la trajectoire d'évitement définie par les temps de manœuvre est 0 et t_1 :

$$r(t_0, t_1) = 2(t_1 - t_0)(1 - cos\alpha) \tag{5.5}$$

5.2.4 Conséquences de l'introduction des intervalles

On considère maintenant que l'on a $t_0 \in T_0$ et $t_1 \in T_1$, où T_0 et T_1 sont deux intervalles de $\mathbb{R}$ ($T_0 = [t_0^{min}, t_0^{max}]$ et $T_1 = [t_1^{min}, t_1^{max}]$). On peut calculer l'ensemble des positions possibles d'un avion à un instant t quand ses instants de manœuvres sont donnés par de tels intervalles. L'ensemble des positions possibles à un instant t pour l'avion est noté $P(t)$.

Comme on doit pouvoir avoir $t_0 \leq t_1$, nous considérerons, qu'on a toujours $t_0^{min} \leq t_1^{min}$ et $t_0^{max} \leq t_1^{max}$.

Pour savoir quelle équation utiliser, parmi les équations 5.1 à 5.4, pour calculer l'ensemble des positions possibles de l'avion au temps t, il faut déterminer à quelle étape de la trajectoire d'évitement peut se trouver l'avion au temps t, en fonction des intervalles T_0 et T_1.

Première étape : l'avion peut ne pas avoir quitté sa trajectoire d'origine si et seulement si :

$$t \leq t_0^{max} \tag{5.6}$$

Deuxième étape :l'avion peut être en train de s'éloigner de sa trajectoire d'origine si et seulement si :

$$t_0^{min} \leq t \leq t_1^{max} \tag{5.7}$$

Troisième étape :l'avion peut être en train de revenir vers sa trajectoire d'origine si et seulement si

$$t_1^{min} \leq t \leq 2t_1^{max} - 2t_0^{min} \tag{5.8}$$

Quatrième étape :l'avion peut être revenu sur sa trajectoire d'origine si et seulement si :

$$t \geq max(t_1^{min}, 2.t_1^{min} - t_0^{max}) \tag{5.9}$$

Selon la position de t par rapport aux intervalles T_1 et T_0, l'ensemble $P(t)$des positions possibles de l'avion à l'instant t peut correspondre à une ou plusieurs des étapes décrites ci-dessus. On suppose que l'instant t est fixé, ainsi que les deux intervalles $T_0 = [t_0^{min}, t_0^{max}]$, et $T_1 = [t_1^{min}, t_1^{max}]$.

L'ensemble $P(t)$ est construit de la manière suivante :

1. Si $t \leq t_0^{min}$, on obtient un point, correspondant à la position de l'avion non dévié. C'est le cas (a) de la figure 5.2.

2. Si $t_0^{min} < t \leq t_1^{min}$, on obtient un segment de droite, correspondant aux positions possibles de l'avion en deuxième étape de l'évitement. C'est le cas (c) de la figure 5.2.

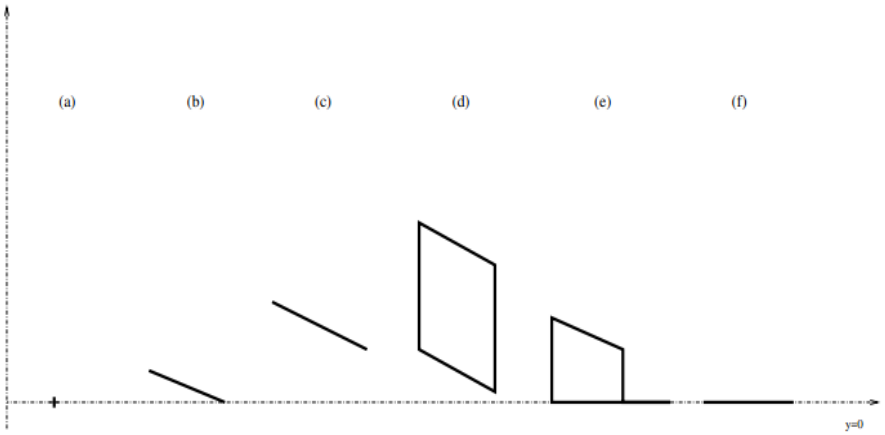


FIG. 5.3 – Différentes formes possibles pour l'ensemble $P(t)$.

3. Si $t \geq t_1^{min}$, il faut tenir compte de la possibilité qu'a l'avion d'être en troisième étape (l'avion revient vers sa trajectoire d'origine). Le cas (d) de la figure 5.2 montre la situation dans laquelle l'avion ne peut être qu'en troisième étape, tandis que dans le cas (e) l'avion peut être en troisième ou en quatrième étape.

4. si $t \geq 2t_1^{max} - t_0^{min}$, l'avion ne peut-être qu'à la quatrième étape de l'évitement, $P(t)$ est un segment de droite. C'est le cas (f) de la figure 5.2.

Pour mesurer la distance entre les avions, on discrétise l'intervalle de temps sur lequel est menée l'optimisation, et on minimise, sur l'ensemble des pas de temps,les séparations minimales et maximales entre les positions possibles des avions correspondant aux intervalles de manœuvres à chaque pas de temps. On peut alors déterminer si la violation de la contrainte est certaine (dans ce cas, l'intervalle n'est pas admissible), si elle est impossible, (dans ce cas,

l'intervalle est dit admissible,et la vérification ne sera plus à faire) ou si elle est indéterminée (il faudra alors de nouveau vérifier la contrainte après le prochain découpage de l'intervalle de manœuvres en deux).

Pour des intervalles de manœuvres T_0 et T_1, l'ensemble des retards possibles est un intervalle. On note

$$r_{min}(T_0, T_1) = max(0, (t_1^{min} - t_0^{max})(1 - cos\alpha))$$

et

$$r_{max}(T_0, T_1) = (t_1^{max} - t_0^{min})(1 - cos\alpha)$$

Pour un avion, l'ensemble des valeurs possibles de $r(t_0, t_1)$ est donné par l'intervalle $R(T_0, T_1)$:

$$R(T_0, T_1) = [r_{min}(T_0, T_1), r_{max}(T_0, T_1)] \tag{5.10}$$

Le problème d'optimisation s'ecrit :

$$\begin{cases} minx(t) = & v(t_o \ - \triangle t \ cos\alpha) \\ t_0 \in T_0 = & [t_0^{min}, t_0^{max}] \\ t_1 \in T_1 = & [t_1^{min}, t_1^{max}] \end{cases}$$

5.3 Modèle d'optimisation

5.3.1 Variables de l'optimisation

Pour un problème à n avions, traité dans le cadre de l'approche globale, l'espace de recherche est :

$$E_{opt}^n = ([0, t_f] \times [0, t_f] \times \{\alpha, -\alpha\})_n$$

On cherche donc à minimiser une fonction f_{opt}^n définie sur un sous ensemble D_{opt}^n de l'ensemble E_{opt}^n . L'ensemble D étant l'ensemble des points de E_{opt}^n permettant de définir une trajectoire d'évitement, c'est-à-dire tels que l'on a $t_0 \leq t_1$:

$$D_{opt}^n = \{(t_0^1, t_1^1, \alpha^1), .., (t_0^n, t_1^n, \alpha^n) \ \mid \ t_0^i \leq t_1^i \ \ , \ \ i = 1..n\}$$

A chaque étape de partage de l'intervalle en deux, on vérifie que la contrainte $t_{0min}^i \leq t_{1max}^i$ est respectée pour tout i, faute de quoi l'intervalle est supprimé. De même, on doit vérifier que $t_{0min}^i \leq t_{1min}^i$ et $t_{0max}^i \leq t_{1max}^i$. Si tel n'est pas le cas, on réduit la taille des intervalles. Un point de D_{opt}^n définit une trajectoire d'évitement pour chacun des n avions impliqués dans le conflit. On dira qu'un point de D_{opt}^n respecte les contraintes de séparations si, en suivant

les trajectoires définies par ce point, les n avions restent séparés deux à deux sur toute la durée de l'évitement.

Le fait qu'un point de D respecte ou non les contraintes de séparation se traduit par la valeur de la fonction f^n_{opt} en ce point.

Pour cela on définit un réel suffisamment grand, noté G_r (nous allons préciser tout de suite ce que nous entendons par suffisamment grand).
Considérons un point $t_r = ((t^1_0, t^1_1, \alpha^1), .., (t^n_0, t^n_1, \alpha^n)) \in D$.

- Si t_r respecte les contraintes de séparation, la fonction f^n_{opt} traduit le retard entraîné par les trajectoires :

$$f^n_{opt}(t_r) = \sum_{i=1}^{n} t^i_1 - t^i_0 \tag{5.11}$$

- Si t_r ne respecte pas les contraintes de séparation ,

$$f^n_{opt}(t_r) = G_r - \partial_{min}(t_r) \tag{5.12}$$

Où ∂_{min} est la distance minimale possible entre deux avions. En fonction des données du problème, on choisit G_r suffisamment grand pour que les valeurs De f^n_{opt} sur les points ne respectant pas les contraintes de séparation soient plus grandes que ses valeurs sur les points les respectant. le problème d'optimisation est :

$$\begin{cases} \min & f^n_{opt} \\ t_r \in D & \subset E^n_{opt} \end{cases}$$

5.4 Conclusion

A travers ces exemples d'application, on a pu montrer que la combinaison d'un algorithme génétique et d'un algorithme de type Branch and Bound pouvait s'avérer plus efficace que l'une des deux approches prises séparément. Les algorithmes génétiques s'avèrent efficaces lorsque l'on est confronté à des problèmes combinatoires de grande taille, mais moins performants pour des optimisations locales. Les méthodes déterministes de type BB ne sont applicables que sur des problèmes de taille réduite sur lesquels elles garantissent l'optimalité des solutions trouvées. Ces exemples nous a également permis d'utiliser la séparabilité partielle du problème pour améliorer les performances de l'algorithme génétique.

Conclusion générale

Pour résoudre de manière efficace des problèmes d'optimisation globale, l'utilisateur devait toujours choisir entre des algorithmes de recherche locale, telle que la méthode du lagrangien augmenté, des algorithmes métaheuristiques basés sur une méthode de recherche locale, tels que les méthodes tabou ou VNS , ou des techniques d'optimisation stochastique, telles que les algorithmes génétiques que nous avons étudié dans le troisième chapitre.

Les algorithmes d'optimisation globale déterministe ont fait d'énormes avancées ces dernières années en se révélant toujours plus efficaces pour résoudre de manière exacte un grand nombre de problèmes. Tel que l'algorithme de Branch and Bound par intervalles. Cependant, IBBA (Interval Branch-and-Bound Algorithm) a aussi montré son efficacité pour résoudre des problèmes de grande dimension.

Les combinaisons entre les algorithmes génétiques et Branch-and-Bound par intervalles développées dans le quatrième chapitre sont prometteuse, car les encadrements obtenus sont toujours meilleurs que les autres . Grâce aux techniques développées dans cette étude, les limites de ce type d'algorithme ont pu encore être repoussées.

Finalement, à travers l'exemple d'application, nous avons pu montrer que la combinaison d'un algorithme génétique et d'un algorithme de type Branch and Bound pouvait s'avérer plus efficace que l'une des deux approches prises séparément. Les algorithmes génétiques s'avèrent efficaces lorsque l'on est confronté à des problèmes combinatoires de grande taille, mais moins performants pour des optimisations locales. Les méthodes déterministes de type BB ne sont applicables que sur des problèmes de taille réduite sur lesquels elles garantissent l'optimalité des solutions trouvées. Cet exemple nous a également permis d'utiliser la séparabilité partielle du problème pour améliorer les performances de l'algorithme génétique.

Perspective : c'est par exemple généraliser au cas avec contraintes, c'est-à-dire sur un domaine quelconque au lieu d'un box. Le problème sera difficille.

Bibliographie

[1] R. Hammer, M. Hocks, U. Kulishet D.Ratz : Numerical Toolbox for Verified Computing I. Springer-Verlag, Berlin, 1993.

[2] E.R.Hansen et W.G. William : Global Optimization Using Interval Analysis.Marcel Dekker Inc., New York, 2^{eme} édition, 2004.

[3] K. Ichida et Y. Fujii : An interval arithmetic method for global optimization. Computing, 23(1) : 85-97, 1979.

[4] S.Skelboe : Computation of rational interval functions. BIT Numerical Mathematics, 14(1) : 87-95, 1974.

[5] H. Ratschek and J. Rokne.New Computer Methods for Global Optimization. Ellis Horwood Ltd., Chichester, 1988.

[6] Frédéric Messine. Méthodes d'Optimisation Globale basées sur l'Analyse d'intervalle pour la Résolution des Problèmes avec Contraintes. Thèse de doctorat, Institut National Polytechnique de Toulouse, Toulouse, France, septembre 1997.

[7] R.B. Kearfott : An interval branch and bound algorithm for bound constrained optimization problems. Journal of Global Optimization,2(3) :259-280,1992.

[8] Neumaier, A. (1990). Interval Methods for Systems of Equations. Cambridge University Press, Cambridge, UK.

[9] Moore, R. E. (1966). Interval Analysis. Prentice-Hall, Englewood Cliffs, NJ.

[10] Krawczyk, R. and Neumaier, A. (1985). Interval slopes for rational functions and associated centered forms. SIAM Journal of Numerical Analysis, 22 :604-616.

[11] Hansen, E. R. and Greenberg, R.I. (1983). An interval Newton method. Applied Mathematical Computing, 12 :89-98.

[12] Hansen, E. R. (1992b). Global Optimization using Interval Analysis. Marcel Dekker, New York, NY.

[13] Nicolas Durand 5 octobre 2004. Algorithmes génétiques et autres outils d'optimisation appliqués à la gestion de trafic aérien.

[14] Massa DAO le 7 Septembre 2006 à Cholet - Université d'Angers.Caractérisation d'ensembles par des méthodes intervalles. Applications en automatique.

[15] A.D.BETHEKE. Genetic algorithms as function optimizers. PhDthesis, University of Michigan, 1981.

[16] J. E.DENNIS, R.B. SCHNABEL. Numerical Methods for Unconstrained Optimization and Non-linear Equations. Prentice-Hall, 1983. Pages 99-132.

[17] R.LERICHE. Optimization of composite structures by genetic algorithms. PhDthesis, Virginia Institute and State University, 1994. 188 pages.

[18] J-M. ALLIOT. Techniques d'optimisation stochastique appliquées à certains problèmes du trafic aérien. Thèse d'habilitation Institut National Polytechnique de Toulouse INPT 1998. 187 pages.

[19] P. HAJELA, C-Y. LIN.Real versus binary coding in genetic algorithms : a comparative study. Computational engineering using metaphors from nature, CIVIL-COMPPress, 2000. Pages 77-83

[20] V. MANGIN. Optimisation - Cours sur les Algorithmes Génétiques.http ://www.eudil.fr/7Evmagnin/coursag/optimisation.html École Universitaire D'Ingénieurs de Lille EUDIL, 1999.

[21] Z.MICHALEWICZ. Genetic Algorithms + Data Structures = EvolutionPrograms. Berlin Springer-Verlag, 1992.387 pages.

[22] K.DEJONG. An analysis of the behavior of a class of genetic adaptive systems. PhDthesis, University of Michigan, 1975.

[23] D.E.GOLDBERG, KDEB, B.KORB. Don't worry, bemessy. In Belew, R.K.et Booker, L.B., editors, Proceedings of the Fourth International Conference on Genetic Algorithms.

[24] S.ROBERT, "cours sur les algorithmes génétiques", Institut Supérieur des techniques avancées de Saint-Etienne, Année 2003-2004.

[25] J.H.HOLLAND. Adaptationin natural and artificial systems. Brad ford Book, 1992. 211 pages.

[26] T. BACK. Evolutionary Algorithmsin Theory and Practice. Oxford University Press 1996. 314 pages.

[27] Alander,J.T.(1992). " On optimal population size of genetic algorithms. "CompEuro'92. 'Computer Systems and Software Engineering'. Proceedings.

[28] PAPA ALDEMBA FAYE, "Couplage Algorithme Génétique - code éléments finis pour le dimensionnement de structures en matériaux composites, Thèse doctorat, l'université CLERMONT-FERRAND **II**, N° d'ordre 1497, Année 2004

[29] N. Benahmed, "Optimisation de réseaux de neurones pour la reconnaissance de chiffres manuscrits isolés : sélection et pondération des primitives par algorithme génétique". Université du Québec, 2002.

[30] T.Vallé et M.Yildizoglu "Présentation des algorithmes génétiques et de leurs applications en économie", Université de Nantes et Université Montesquieu Bordeaux IV 2001.

[31] Bourazza S. : Variantes d'algorithmes génétiques appliquées aux problèmes d'ordonnancement. Thèse de Doctorat de l'Université du Havre, soutenue le 30/11/2006.

[32] J.-M. ARNAUDIES, H. FRAYSSE. Cours de mathématiques 3. Dunod, Paris 1989. Compléments d'analyse. [Complements of analysis].

[33] X. GOURDON. Les Maths en tête, mathématiques pour M' : Analyse. Ellipses, Paris 1994.

[34] J.-B. HIRIART-URRUTY. L'optimisation. Que sais-je ? PUF, Paris 1996.

[35] M.Talbi P. Preux Assessing the evolutionary algorithm paradigm to solve hard problems.constraint processing, workshop on really hard problem solving, September 1995.

[36] M.Hoeck K .w.hansmann.Production control of a flexible manufact uring system in Job shop environment. pages 341-351. Internationl Transactions in operational Reasearch, September-November 1997.

[37] M.Widmer A.Hertz. An improved tabou search approach for solving the job shop scheduling problem with tooling containts. Discret Applied Mathematics, pages 319-345, 1996.