Chapitre 8. OPTIMISATION DE PARAMETRES

Anthony Collin

LEMTA – UMR 7563 CNRS

Université de Lorraine

Benjamin Batiot

Institut Pprime – UPR 3346 CNRS

Université de Poitiers

ESIA – Ecole des Sciences de l'Incendie et Applications – Obernai, 27 mai au 1er juin 2018

1. Introduction

Dans de nombreuses applications dédiées à l'incendie, il n'est pas rare de vouloir développer un modèle physique à partir de données d'expérience. De manière classique, ce modèle physique va dépendre de variables et de paramètres, qui seront ses données d'entrées. Ce modèle permettra alors d'estimer une ou plusieurs sorties. La Figure 1 donne la représentation « mathématique » d'un modèle dit physique.

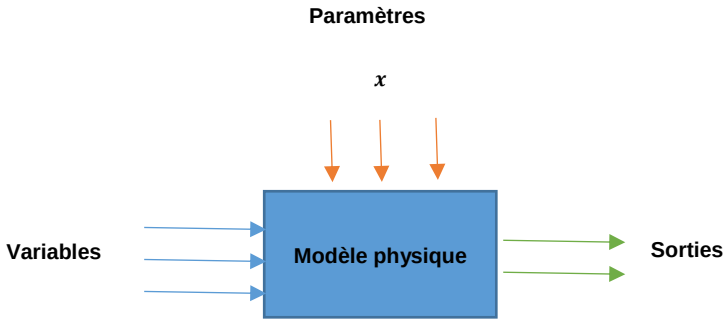


Figure 1 : Représentation d'un modèle physique

La ou les sorties du modèle physique vont bien évidemment dépendre des données d'entrée (variables ou paramètres d'entrée). Néanmoins, au cours du processus physique que représente le modèle, les paramètres n'évoluent pas contrairement aux variables d'entrée.
Un exemple pourrait être la modélisation d'une pièce en feu, où le temps serait une variable d'entrée et la géométrie de la pièce constituerait un paramètre d'entrée.
Les paramètres d'un modèle physique sont, soient connus ou mesurables, comme les dimensions d'une pièce en feu par exemple, soient inconnues et inaccessibles, par des moyens classiques de mesure, comme la conductivité thermique d'un milieu homogène équivalent par exemple. Le recours à des données expérimentales associées à un algorithme d'identification est bien souvent alors le seul moyen de définir ou de calibrer correctement le modèle.
L'objectif de ce module est donc de présenter aux lecteurs, les grands principes des méthodes d'optimisation en s'attardant sur quelques approches metaheuristiques très pertinentes pour les problèmes complexes.

1.1 Qu'est-ce qu'un problème d'optimisation ?

Les paramètres d'entrée, notés pour la suite $\boldsymbol{x}$, sont recherchés sur un espace de phase donné : à chaque paramètre $\boldsymbol{x_i}$ est associé un domaine de recherche défini par une borne inférieure et une borne supérieure, ce qui constituera les contraintes de notre problème d'optimisation.

La première étape d'un processus d'identification de paramètre consiste à construire une fonction « coût » ou fonction « objectif », notée $\boldsymbol{f}$, qui sera l'image de la « qualité » des paramètres d'entrée, appelés ici $\boldsymbol{x}$, qui auront été utilisés pour générer la ou les sorties du modèle physique. Bien souvent, cette fonction « coût » est construite sur la norme des écarts entre la sortie du modèle et les données expérimentales.

Le problème qui se pose alors est de rechercher le meilleur couple de paramètres d'entrée, $\boldsymbol{x}$, qui minimisera de manière globale la fonction « coût » : c'est le problème d'optimisation. La notion de minimum global est très importante, car il n'est pas rare d'obtenir des couples de paramètres d'entrée qui minimisent de manière locale la fonction « coût ». Ainsi, l'efficacité d'un algorithme d'optimisation est mesurée par sa capacité à s'affranchir des différents minimums locaux qu'il pourrait trouver pour

se concentrer sur la recherche de la solution optimale qui minimise la fonction « coût » sur l'ensemble de recherche.

Les problèmes d'optimisation peuvent être classés en plusieurs grandes familles :

- Optimisation numérique où les paramètres recherchés, $\boldsymbol{x}$, appartiennent à $\mathbb{R}^{\boldsymbol{n}}$, où $\boldsymbol{n}$ est le nombre de paramètres recherchés ;
- Optimisation discrète (ou combinatoire) où $\boldsymbol{x}$ est fini ou dénombrable ;
- Commande optimale, où $\boldsymbol{x}$ est un ensemble de fonctions ;
- Optimisation stochastique, où $\boldsymbol{x}$ constituent des paramètres aléatoires ;
- Optimisation multicritère où $\boldsymbol{x}$ doit minimiser plusieurs fonctions « coûts » à la fois.

Une étape importante en pratique consiste à identifier le type de problème que l'on traite afin de choisir une famille d'algorithme d'optimisation qui sera pertinente. Voici une liste, non exhaustive, de différents type d'algorithmes les plus utilisés de nos jours :

- Programmation linéaire : méthode du simplex, points intérieurs, …
- Optimisation non linéaire locale :
  - Avec dérivées et sans contraintes : méthodes de gradient, de Newton, de quasi-Newton, approche de Levenberg-Marquardt, …
  - Avec dérivées et avec contraintes : méthode SQP, Wilson, méthode de pénalisation, méthode lagrangienne, …
  - Sans dérivée : DFO, NEWUOA, MADS, …
- Optimisation non linéaire globale :
  - Méthodes déterministes : simplexe non linéaire, réseaux de neurones, krigeage, …
  - Méthodes stochastiques : méthodes évolutionnaires, recuit-simulé, recherche de tabou

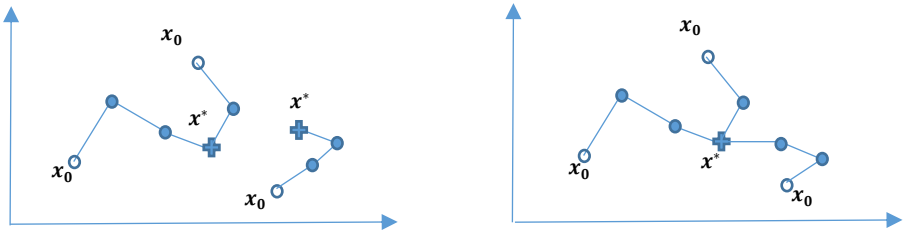
Dans le cadre de ce module, c'est cette dernière classe de modèles (méthodes stochastiques) qui sera présentée par la suite.

1.2 Convergence et vitesse de convergence

La convergence d'un algorithme est donnée par la suite des itérés $\boldsymbol{x}$ qui sont fournis par l'algorithme d'optimisation. Un algorithme de minimisation est dit convergent, si la suite de ses itérés, $(\boldsymbol{x_k})_{\boldsymbol{k\epsilon\mathbb{N}}}$ appartenant à $\boldsymbol{I}$**,** converge vers un point limite $\boldsymbol{x}^*$**, solution du problème,**

$$\min_{\boldsymbol{x\epsilon I}}\big(\boldsymbol{f}(\boldsymbol{x})\big)$$

La convergence est dite locale, si elle n'a lieu que pour des points initiaux $\boldsymbol{x_0}$ dans un voisinage de $\boldsymbol{x}^*$. Sinon, elle est dite globale. En pratique, le but d'un algorithme d'optimisation est généralement d'estimer un point critique, vérifiant les conditions ou contraintes nécessaires.



(a) Deux minima locaux (b) Un minimum global

Figure 2 : Notion de convergence locale et globale

La définition de la notion de convergence globale peut dont être énoncée de la manière suivante : soit un algorithme itératif qui génère une suite $(\boldsymbol{x_k})_{k\epsilon\mathbb{N}}$ dans $\boldsymbol{I}$ afin de résoudre le problème,

$$\min_{x\epsilon I}\big(f(x)\big)$$

où $\boldsymbol{f}$ est une application de classe $\boldsymbol{C^1}$ de $\boldsymbol{I} \rightarrow \mathbb{R}$. L'algorithme est dit globalement convergent si quel que soit le point initial $\boldsymbol{x_0 \epsilon I}$,

$$\lim_{k\rightarrow+\infty} \|\nabla f(x_k)\| = 0$$

Cette propriété garantit que le critère d'arrêt « $\|\nabla f(x_k)\| < \varepsilon$ » sera satisfait à partir d'un certain rang quelle que soit la précision $\boldsymbol{\varepsilon}$ demandée.

Il est bien entendu très important de garantir la convergence d'un algorithme de minimisation sous certaines hypothèses, mais la vitesse de convergence et la complexité du modèle utilisé sont également deux facteurs à prendre en compte lors de la conception ou de l'utilisation d'une approche donnée. En effet, on a tout intérêt à ce que la méthode choisir soit à la fois rapide, précises et stage. Pour cela, on introduit une notion supplémentaire, la vitesse ou le taux de convergence qui mesure l'évolution de l'erreur commise par $\|\boldsymbol{x_k} - \boldsymbol{x^*}\|$.

Soit $(\boldsymbol{x_k})_{k\epsilon\mathbb{N}}$ une suite d'itérés générés par un algorithme convergent donné. On note $\boldsymbol{x^*}$ la limite de la suite $(\boldsymbol{x_k})_{k\epsilon\mathbb{N}}$ et on suppose que $\forall \boldsymbol{k\epsilon}\mathbb{N}, \boldsymbol{x_k} \neq \boldsymbol{x^*}$ (sinon l'algorithme convergerait en un nombre fini d'itérations). La convergence de l'algorithme est dite,

- Linéaire, si l'erreur $\boldsymbol{e_k} = \|\boldsymbol{x_k} - \boldsymbol{x^*}\|$ décroit linéairement, tel que,

$$\lim_{k\rightarrow+\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = \tau$$

Où $\boldsymbol{\tau\epsilon}]\boldsymbol{0}, \boldsymbol{1}[$.

- Super-linéaire, si,

$$\lim_{k\rightarrow+\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 0$$

- D'ordre $\boldsymbol{p}$, s'il existe $\boldsymbol{\tau} \geq \boldsymbol{0}$ tel que,

$$\lim_{k\rightarrow+\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^p} = \tau$$

En particulier, si $\boldsymbol{p} = \boldsymbol{2}$, la convergence est dite quadratique (à partir d'un certain rang, le nombre de chiffres significatifs exacts double à chaque itération).

Bien entendu, on a intérêt à ce que la convergence d'un algorithme soit la plus élevée possible afin de converger vers la solution en un minimum d'itérations pour une précision donnée.

Enfin, la convergence est dite sous-linéaire si,

$$\lim_{k\rightarrow+\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = 1$$

Un algorithme à convergence sous-linéaire converge tellement lentement qu'il est considéré en pratique comme inacceptable.

1.3 Critères d'arrêts

Soit $\boldsymbol{x^*}$ un minimum local de la fonction coût $\boldsymbol{f}$ à minimiser. Supposons que l'on choisisse comme test d'arrêt dans l'algorithme de descente, le critère idéal « $\boldsymbol{x_k} = \boldsymbol{x^*}$ ». Dans un monde parfait, en

supposant tous les calculs exacts et la capacité de calcul illimitée, soit l'algorithme s'arrête après un nombre fini d'itérations, soit il construit théoriquement une suite infinie de points $\boldsymbol{x_1, x_2, x_3}$, … de $\boldsymbol{I}$ qui converge vers $\boldsymbol{x^*}$.
En pratique, un test d'arrêt devra être choisi pour garantir que l'algorithme s'arrête toujours après un nombre fini d'itérations et que le dernier point estimé soit suffisamment proche de $\boldsymbol{x^*}$ quand l'algorithme converge.
Soit $\boldsymbol{\varepsilon > 0}$ la précision demandée au cours de la recherche de paramètres. Plusieurs critères sont à notre disposition. Tout d'abord, un critère d'optimalité basé sur les conditions nécessaires de minimisation du premier ordre peut être, dans le cas d'une optimisation différentiable sans contrainte,

$$\|\boldsymbol{\nabla f(x_k)}\| < \boldsymbol{\varepsilon}$$

Auquel cas l'algorithme s'arrête et fournit l'itéré courant, $\boldsymbol{x_k}$, comme solution.
En pratique, le test d'optimalité n'est pas toujours satisfait, ou nous n'avons pas toujours accès la dérivée première de la fonction « coût » ; et on devra faire appel à d'autres critères, fondés sur l'expérience du numérique,

- Stagnation de la solution : $\|\boldsymbol{x_{k+1} - x_k}\| < \boldsymbol{\varepsilon(1 + \|x_k\|)}$
- Stagnation de la valeur de la fonction « coût » : $\|\boldsymbol{f(x_{k+1}) - f(x_k)}\| < \boldsymbol{\varepsilon(1 + |f(x_k)|)}$
- Nombre d'itérations dépassant un seuil fixé à l'avance : $\boldsymbol{k < \mathbf{IterMax}}$

De manière complémentaire, il peut être également envisagé une combinaison de ces critères.
En pratique, on préférera travailler avec les erreurs relatives plutôt qu'avec les erreurs absolues, trop dépendantes de l'échelle des grandeurs à identifier. Une autre solution consiste à normaliser les paramètres recherchés, sur un intervalle soit $[\mathbf{0}, \mathbf{1}]$ou soit $[-\mathbf{1}, \mathbf{1}]$ par,

$$\tilde{\boldsymbol{x}} = \frac{\boldsymbol{x - x_{min}}}{\boldsymbol{x_{max} - x_{min}}} \text{ ou } \tilde{\boldsymbol{x}} = \mathbf{2}\frac{\boldsymbol{x - x_{min}}}{\boldsymbol{x_{max} - x_{min}}} - \mathbf{1}$$

Dans lesquelles $\boldsymbol{x}$ est le vecteur de paramètres recherchés, $\boldsymbol{x_{min}}$ le vecteur contenant les bornes inférieures et $\boldsymbol{x_{max}}$ les bornes supérieures de l'espace de recherche.

2. L'optimisation multicritère

Dans de nombreuses applications liées à l'incendie, la recherche de paramètres peut être appliquée au regard de plusieurs sorties d'un modèle physique (et non une seule) : par exemple pour la simulation d'une expérience en ATG (pour la dégradation thermique des matériaux), l'évolution de la masse de l'échantillon et de sa vitesse de perte de masse peuvent être utilisées pour procéder à une identification de paramètres.

L'optimisation multicritère vise à minimiser un problème à plusieurs objectifs différents :

$$\boldsymbol{f(x) = \big(f_1(x), f_2(x), \dots, f_n(x)\big)}$$

Où $\boldsymbol{f(x)}$ est le vecteur objectif et $\boldsymbol{f_i(x)}$ représente une des $\boldsymbol{n}$ fonctions coût à minimiser (ici $\boldsymbol{i = (1, 2, \dots, n)}$). Afin de minimiser $\boldsymbol{f(x)}$ nous pouvons utiliser la technique du front de Pareto décrite dans cette partie.
Le principe de fonctionnement est de minimiser une fonction « coût » mais, en multicritère nous en avons plusieurs alors que nous recherchons une seule solution au problème. Ainsi ce pose le problème de la comparaison des critères entre eux pour sélectionner finalement cette solution unique.

La méthode fait le choix réduire le problème multicritère en un problème uni-critère, dès le départ. Pour cela, cette méthode s'appuie sur des critères de comparaison, contrairement à la méthode du front de Pareto qui utilise tous les critères sans comparaison. Pour que cette dernière méthode fonctionne, elle utilise la notion de dominance des solutions les unes par rapport aux autres. Cette dominance est expliquée dans la formulation suivante :

$$\boldsymbol{x} \preceq \boldsymbol{y} \leftrightarrow \forall \boldsymbol{i} \in \{\mathbf{1}, \dots, \boldsymbol{n}\} | \boldsymbol{f_i}(\boldsymbol{x}) \leq \boldsymbol{f_i}(\boldsymbol{y}) \,\&\, \exists \boldsymbol{i} \in \{\mathbf{1}, \dots, \boldsymbol{n}\} | \boldsymbol{f_i}(\boldsymbol{x}) < \boldsymbol{f_i}(\boldsymbol{y})$$

Dans cette définition, $\boldsymbol{x}$ et $\boldsymbol{y}$ représente des ensembles de solutions tels que $\boldsymbol{x} = (\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_r})$ et $\boldsymbol{y} = (\boldsymbol{y_1}, \boldsymbol{y_2}, \dots, \boldsymbol{y_r})$ où $\boldsymbol{r}$ représente le nombre de paramètres à optimiser. Prenons $\boldsymbol{K}$ comme l'ensemble des solutions réalisables ainsi, l'ensemble $\boldsymbol{L} = \boldsymbol{F}(\boldsymbol{K})$ correspond à l'ensemble des images des solutions réalisables dans l'espace des objectifs (cf. Figure 3) avec $\boldsymbol{j_i} = \boldsymbol{f_i}(\boldsymbol{x})$ le point de l'espace des objectifs correspondant à la solution x.

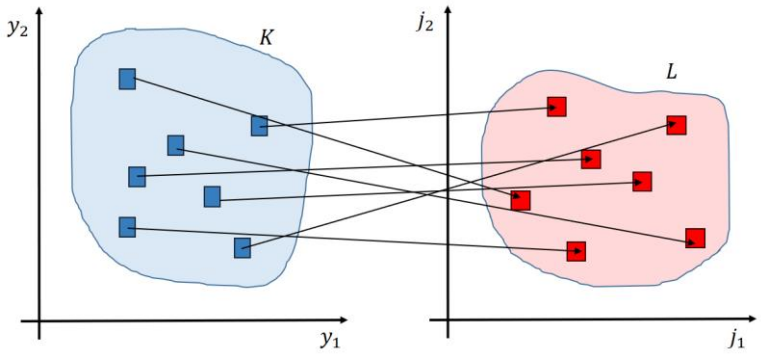


Figure 3 : Partie de gauche, représentation des solutions y dans l'espace des solutions réalisables $\boldsymbol{K}$ et partie de droite, images j des solutions y dans l'espace des critères $\boldsymbol{L}$

Ainsi, cette technique vise à travailler à partir de l'espace des critères afin de sélectionner les solutions réalisables les plus performantes (minimisent la fonction coût associée). Sur la partie droite nous voyons que les solutions sont désordonnées, l'enjeu est de les ordonner en ramenant progressivement les points sur un front appelé front de Pareto, comme le montre la Figure 4.

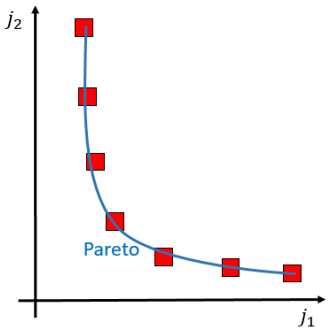


Figure 4 : un exemple de front de Pareto, pour deux fonctions « coûts »

Le front de Pareto représente la limite pour laquelle aucune solution n'est meilleure qu'une autre. Ainsi, si nous souhaitons minimiser une fonction, ça se fera nécessairement au détriment d'une autre. Ce front de Pareto est déterminé en fonction des critères de dominance exprimés dans la formulation précédente et représentés dans le graphique suivant :

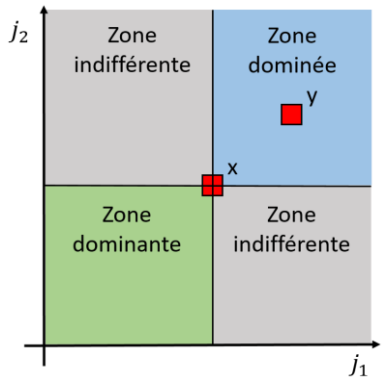


Figure 5 : critère de dominance pour définir le front de Pareto

Dans la Figure 5, nous voyons par exemple que la solution $\boldsymbol{x}$ domine la solution $\boldsymbol{y}$ dans l'espace des critères. Une fois l'optimisation terminée et le front de Pareto défini, il reste le problème du choix d'une solution unique sur l'ensemble des points situé sur ce front. Diverses solutions existent pour ce choix, le plus simple étant une sélection aléatoire pour l'ensemble des solutions.

3. Les approches métaheuristiques

Par définition, une métaheuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficile pour lesquels les techniques abordées dans les parties précédentes ne s'appliquent pas.

Les problèmes d'optimisation difficile se caractérisent par une structure complexe de leur espace des configurations. C'est-à-dire que lorsque nous cherchons à minimiser la fonction « coût », plusieurs minimums locaux apparaissent ce qui rend difficile la recherche de l'optimum (minimum global). Ainsi, les méthodes citées dans la partie précédente convergent difficilement vers ce minimum global. Ce problème a inspiré des analogies avec des problèmes physiques ou biologiques afin de trouver une solution à l'optimisation de problèmes complexes.

Ces algorithmes ont été développés tout d'abord à partir de phénomènes physiques à partir de 1982, c'est la première méthode que nous allons aborder, celle du recuit simulé. Ensuite, de nombreux algorithmes ont vu le jour. Nous pouvons classer ces algorithmes en deux grandes familles, les algorithmes évolutionnaires et les algorithmes comportementaux. Nous verrons un algorithme de chaque catégorie, la première approche présentée sera les algorithmes génétiques et la seconde, les algorithmes par essaims de particules.

L'ensemble des algorithmes présentés ici ne garantissent pas de trouver la solution exacte, mais ils essaient de s'en approcher au maximum.

3.1 Le recuit simulé

L'algorithme du recuit simulé fait partie des premiers métaheuristiques et s'appuie sur une analogie avec des phénomènes physiques.

Imaginons une pièce de métal en fusion. Nous voulons que le métal final, une fois refroidi, soit le plus solide et résistant possible. Pour ce résultat, il faut que les atomes soient parfaitement organisés en structure cristalline. Un refroidissement trop rapide ne laisserait pas le temps aux atomes de s'organiser et créera des imperfections, un refroidissement trop lent ne servirait à rien. Par conséquent, il faut trouver l'optimum de décroissance pour parvenir au meilleur résultat en terme de résistance du solide. Ces phénomènes font intervenir le comportement des atomes soumis à une certaine température donc, la physique statistique.

Ainsi, l'objectif consiste à trouver la bonne rampe de décroissance de la température. Pour ça, des paliers de diminution sont utilisés. Dans le recuit réel, ces paliers sont calculés en fonction du comportement des atomes dans le solide. Dans le recuit simuler, il faut trouver des critères qui permettent de valider et de transiter d'un palier à un autre.

Ainsi, les analogies suivantes avec la physique sont utilisées par l'algorithme d'optimisation pour calculer les paliers :

1) D'une part, lorsqu'un système a atteint son état d'équilibre thermodynamique a une température T, la probabilité pour un système physique (par exemple atome ou molécule) de posséder un niveau d'énergie E est donnée par l'équation suivante [Boltzmann, 83] :

$$\frac{N_i}{N} = \frac{g_i e^{\frac{E_i}{k_b T}}}{Z(T)}$$

Dans cet équation, $Z(T)$ est la fonction de partition, k_b est la constante de Boltzmann, g_i est la dégénérescence, N le nombre total de systèmes physiques et N_i le nombre de systèmes physiques dans l'état d'énergie i. Dans cette équation, le terme $e^{\frac{E}{k_b T}}$ donne la probabilité relative (non normalisée) d'un état, c'est le facteur de Boltzmann. C'est ce facteur qui va nous intéresser dans le recuit simulé.

2) D'autre part, l'algorithme de Métropolis permet de simuler l'évolution d'un système physique pour qu'il atteigne son état d'équilibre thermodynamique à une température T fixée [Métropolis et al. 53]. Le principe de cet algorithme est de partir d'une configuration particulière et de lui imposer une variation élémentaire. Ensuite, si cette transformation permet de diminuer le paramètre de contrôle (ou fonction coût, $\Delta E = \Delta f$) alors les modifications sont conservées par contre, si la transformation augmente la fonction de contrôle, les modifications sont conservées qu'avec une probabilité de $e^{-\frac{\Delta E}{T}}$. Il faut appliquer un grand nombre de fois cette règle d'acceptation de Métropolis afin que statistiquement une configuration ne dépende que de celle qui la précède (chaine de Markov). Nous obtenons ainsi une distribution de Boltzmann de l'énergie autour d'une température fixée.

En pratique, l'algorithme du recuit simulé commence avec une configuration choisie aléatoirement, x_0. Ensuite, des variations élémentaires sont prises sur les paramètres d'entrée de manière aléatoire,

$$x_{k+1} = x_k + \Delta x$$

Ces variations sont immédiatement conservées si elles diminuent la fonction « coût » $\Delta f < 0$ avec $\Delta f = f(x_{k+1}) - f(x_k)$ ou si elles sont acceptées avec une probabilité,

$$p = e^{-\frac{\Delta f}{T}}$$

Dans ce fonctionnement, quand la température est élevée, l'acceptation des variations est proche de 1 car $\lim_{T \to +\infty} e^{\frac{-\Delta E}{T}} = 1$. Dans ce cas de figure, presque toutes les solutions sont acceptées et le code se rapproche d'un fonctionnement aléatoire simple ce qui permet de bien scruter l'espace des solutions. Par contre, au fur et à mesure que la température diminue, seules les solutions qui permettent de diminuer la fonction coup sont conservées ce qui permet à l'algorithme de se concentrer sur une zone ou le minimum global semble être présent. Cependant, dans les derniers temps une faible probabilité d'acceptation existe ce qui laisse à l'algorithme la possibilité de sortir d'un minimum local pour aller chercher un possible minimum global ailleurs.
Lorsque la chaine de Markov est satisfaite, la température est diminuée en suivant une loi de décroissance.
Ainsi, l'algorithme va converger progressivement vers un minimum de la fonction coût.

Cet algorithme est très robuste et efficace dans beaucoup de cas, mais il nécessite de fixer l'ensemble des conditions ci-dessous pour converger vers une solution acceptable rapidement :

- Température initiale
- Configuration initiale (commun à tous les algorithmes)
- Règle pour valider un palier de température (durée de relaxation)
- Règle permettant de modifier les solutions
- Loi de décroissance de la température
- Critère d'arrêt

Algorithme du recuit simulé

$\boldsymbol{x^* = x_0}$ // Solution initiale
$\boldsymbol{k = 0}$
Choisir une suite décroissante de nombre $\boldsymbol{T_k}$
Tant que (Condition d'arrêt non vérifiée)
    Générer une nouvelle solution $\boldsymbol{y}$ voisine de $\boldsymbol{x_k}$
    $\boldsymbol{\Delta f = f(y) - f(x_k)}$
    $\boldsymbol{p = \min\left(1, e^{-\frac{\Delta f}{T_k}}\right)}$
Si Variable aléatoire $< \boldsymbol{p}$
    $\boldsymbol{x_{k+1} = y}$
Fin
Si $\boldsymbol{f(x_{k+1}) < f(x^*)}$
        $\boldsymbol{x^* = x_{k+1}}$
    Fin
    $\boldsymbol{k = k + 1}$
Fin

3.2 Les algorithmes génétiques

Cet algorithme essaye de trouver la solution au problème en faisant l'analogie avec l'évolution naturelle d'une population. Cette évolution est celle décrite par Darwin dans « l'origine des espèces » en 1859. Cet algorithme est basé sur le principe suivant :

- Un individu Y est composé par un ensemble de gènes ($\boldsymbol{Y = (y_1, y_2, \dots, y_N)}$), ces gènes sont compris dans l'espace des solutions à notre problème ($\boldsymbol{Y \epsilon [K]}$)
- Un ensemble d'individus est appelé une population
- Chaque individu est plus ou moins adapté à son environnement de par les gènes qu'il a reçus. Ainsi, un individu Y se définit par son niveau d'adaptation à son environnement qui est défini par une fonction coût ($\boldsymbol{f(Y)}$)
- La population considérée à l'itération k et appelée k-ième génération
- Les lois de l'évolution (sélection naturelle) s'appliquent pour définir les changements d'une génération à une autre.

Tout l'enjeu de cet algorithme et de définir comment s'opère le changement de population entre deux générations (à chaque itération). La Figure 6 permet de présenter les principales étapes de cet algorithme.

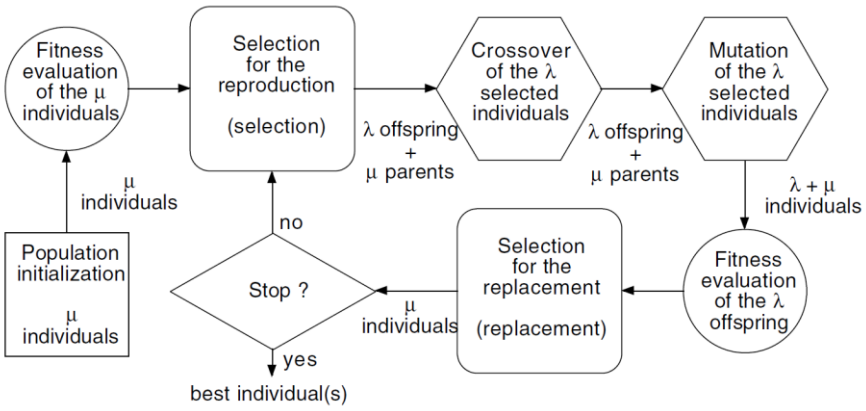


Figure 6 : Schéma d'un algorithme génétique

Plusieurs variantes plus ou moins complexes peuvent exister. Ces variantes proposent notamment des systèmes différents de sélection ou de modification d'une génération à une autre.
Les algorithmes génétiques ont l'avantage de répondre à des problèmes d'optimisation difficile en s'affranchissant de toute analyse de sensibilité au préalable. Ils sont également fiables et très robustes et s'adaptent pour la recherche de variables discrètes ou continues. Par contre, ces algorithmes nécessitent beaucoup de paramètres de réglage tels que, la taille de la population, la taille de l'archivage, des coefficients de diversité… De plus, le nombre d'étapes à respecter le rend très gourmand en temps de calculs ce qui nécessite parfois de paralléliser le programme.

Algorithme génétique

```
P désigne la population
t désigne la génération
début
        t=0
        initialise p(t)
        évaluer p(t)
                Tant que (critère d'arrêt non atteint)
                début
                        t=t+1
                        sélectionner p(t) de p(t-1)
                        recombiner p(t)
                        évaluer p(t)
                fin
fin
```


3.3 La méthode par essaims particulaires

La méthode par essaims de particules (« Particule Swarm Optimization », PSO) a été imaginée en 1995 [Kennedy et al. 95], cet algorithme est issu de l'étude du comportement collectif de déplacement des animaux. Les auteurs de cette méthode ont utilisé la théorie de la sociopsychologie sur le traitement de l'information et les prises de décisions dans des groupes sociaux [Siarry. 11]. Prenons l'analogie souvent faite du comportement collectif des abeilles. Plusieurs colonies d'abeilles sont présentes sur une zone géographique délimitée. Les abeilles ont pour objectif de trouver la localisation où il y a un maximum de fleurs donc de pollen à récolter. Les abeilles vont se déplacer dans l'espace et « sonder » leur environnement pour savoir à chaque position s'il y a présence de fleur. Ainsi, chaque position est plus ou moins intéressante. À chaque changement de position d'une abeille, plusieurs facteurs sont pris en compte. Le premier c'est l'inertie de l'abeille qui se déplace, le second c'est la meilleure position

que l'abeille a trouvée lors de ces recherches et le dernier vient du partage d'information entre les abeilles de la même colonie sur la meilleure position qu'elles ont pu trouver collectivement. Ce comportement permet :

- d'aller chercher des positions « sans motivation » (l'inertie du mouvement),
- que chaque individu puisse scruter l'espace donc augmenter la surface réellement recherchée par les abeilles (comportement individuel)
- que la colonie finisse par trouver la meilleure position, celle qui rapportera le plus à la ruche (comportement collectif)

Si nous remplaçons l'espace physique en 3D des abeilles par un espace multidimensionnel déterminé par le nombre de paramètres inconnus, nous avons les étapes d'optimisation de notre problème.
Ainsi, cet algorithme utilise les données suivantes :

$$\vec{x}_i(it) = f\left(\vec{x}_i(it-1), \vec{v}_i(it-1), \vec{p}_i, \vec{p}_g\right)$$

Où $\vec{x}_i$ représente la position de la particule i, $\vec{v}_i$ sa vitesse, $\vec{p}_i$ sa meilleure position, $\vec{p}_g$ la meilleure position du groupe de particules et it l'itération concernée. Le changement de position des particules s'effectue avec les équations suivantes :

$$\begin{cases} \vec{v}_i(it) = w.\vec{v}_i(it-1) + \phi_1\left(\vec{p}_i - \vec{x}_i(it-1)\right) + \phi_2\left(\vec{p}_g - \vec{x}_i(it-1)\right) \\ \vec{x}_i(it) = \vec{x}_i(it-1) + \vec{v}_i(it-1) \end{cases}$$

Dans ces équations w, ϕ_1 et ϕ_2 sont des pondérations. Les valeurs des ϕ sont déterminées de la façon suivante :

$$\phi = aléa\left[0, \frac{(w+1)^2}{2}\right]$$

Contrairement au précédent, cet algorithme nécessite un seul paramètre de réglage qui est w et qui représente l'inertie de la particule. Classiquement, la valeur de w est égale à 0,6.
Eberhart [Eberhart et al. 01] présente un état de l'art complet sur ce type d'optimisation et Clerc fait une synthèse des travaux [Clerc. 01].

Ci-dessous les grandes étapes de cet algorithme d'optimisation.

Optimisation par essaim de particules

Initialisation x_i, v_i, x_i^{best}, x_g^{best} pour toutes les particules i
Tant que (critère d'arrêt non atteint)
    Pour chaque particule i
        Calcul des ϕ
        Calcul de la vitesse
        Calcul de la position
        Encadrement dans l'espace
        Evaluation de la position avec la fonction coût
        Mise à jour du x_i^{best}
    Fin
    Pour chaque groupe g
        Mise à jour de la meilleure position x_g^{best} dans le groupe g
    Fin
Fin

4. Bibliographie

E. Elbeltagi, T. Hegazy, D. Grierson. Comparison among five evolutionary-based optimization algorithms. Adavcned engineering informatics, Elsevier, 19, 43-53, 2005.

J. Dréo, A. Pétrowski, P. Siarry, E. Taillard. Métaheuristiques pour l'optimisation difficile. Eyrolles, 2003.

C. A. Coello Coello, G. Toscano Pulido, M. Salazar. Handling Multiple Objectives With Particule Swarm Optimization. IEEE Transaction on evolutionary computation, 8, 256-279, 2004.