Introduction aux Méthodes d'Optimisation sans Gradient pour l'Optimisation et le Contrôle en Mécanique des Fluides

R. Duvigneau

INRIA Sophia-Antipolis, Projet OPALE
www-sop.inria.fr/opale

Ecole de printemps OCET
Optimisation et Contrôle des Ecoulements et des Transferts
12 - 17 Mars 2006, Aussois

Table des matières

1 Introduction

1.1 Contexte

On supposera dans l'ensemble de ce cours qu'on cherche à résoudre un problème de la forme suivante :

$$\begin{aligned} &\text{Minimiser } \mathcal{J}(x_c, W(x_c)) \\ &\text{Soumis à } \mathcal{E}(x_c, W(x_c)) = 0 \end{aligned} \tag{1}$$

où $\mathcal{J}$ représente une fonctionnelle coût, x_c le vecteur des variables de contrôle de dimension n et W le vecteur des variables d'état du système. Celui-ci est déterminé par les équations d'état $\mathcal{E}$, considérées comme contraintes pour le problème.

Dans le cadre de l'optimisation et du contrôle des écoulements, ce problème peut designer, par exemple, un problème d'optimisation de forme ou de contrôle optimal. Les équations d'état $\mathcal{E}$ représentent alors le système d'équations régissant l'écoulement, par exemple les équations de Navier-Stokes ou d'Euler et W représente les variables d'écoulement. Pour donner quelques exemples concrêts, un problème typique en optimisation de forme est la réduction de traînée d'un profil d'aile. La fonction de coût prend alors la forme suivante :

$$\mathcal{J}(x_c, W(x_c)) = \int_{\Gamma(x_c)} \overline{\overline{T}}(W(x_c)) \cdot \overrightarrow{n}(x_c) d\Gamma \cdot \overrightarrow{X} \tag{2}$$

où Γ représente la forme du profil paramétrisée par les variables de forme x_c, $\overline{\overline{T}}$ le tenseur des contraintes dépendant des variables d'écoulement W, $\overrightarrow{n}$ la normale au profil et $\overrightarrow{X}$ la direction de l'écoulement amont. Dans le contexte du contrôle optimal des écoulements, la fonction de coût prend typiquement la forme suivante :

$$\mathcal{J}(x_c, W(x_c)) = \int_{T_i}^{T_f} (W(x_c) - W^\star)^2 dt \tag{3}$$

où T_i et T_f sont les bornes d'un intervalle de temps, pendant lequel on souhaite contrôler les variables d'écoulement W pour qu'elles atteignent l'écoulement cible caractérisé par $W^\star$. La formulation du problème (1) est donc suffisamment générale pour les applications qui nous intéressent.

1.2 Limitation des méthodes classiques

Le problème (1) est généralement résolu par une approche itérative convergeant vers la solution du problème $x_c^\star$, les équations d'état étant vérifées à chaque itération du problème d'optimisation (i.e. les équations d'état sont résolues pour chaque vecteur des variables de contrôle x_c). A chaque itération k, un nouveau vecteur des variables de contrôle $x_c^{(k+1)}$ est obtenu à partir *de la valeur de la fonction de coût et de son gradient* et éventuellement d'informations collectées sur le chemin déjà parcouru (voir algorithme (1)).

Il existe de nombreuses méthodes d'optimisation basées sur cet algorithme, correspondant à différentes méthodes de calcul du nouveau vecteur des variables de contrôle à chaque itération. On peut par exemple citer la méthode de plus grande descente, pour laquelle :

$$x_c^{(k+1)} = x_c^{(k)} - \rho \, \nabla \mathcal{J}(x_c, W(x_c)) \tag{4}$$

ou la méthode de Newton (respectivement quasi-Newton), pour laquelle :

$$x_c^{(k+1)} = x_c^{(k)} - \rho \, \mathcal{H}^{-1}(x_c, W(x_c)) \, \nabla \mathcal{J}(x_c, W(x_c)) \tag{5}$$

où ρ est un paramètre numérique et $\mathcal{H}$ la matrice hessienne de la fonction de coût (respectivement une approximation de la matrice hessienne) agissant comme préconditionneur. Toutes ces méthodes sont largement détaillées dans des ouvrages de référence en optimisation numérique[9, 8].

Ces approches ont été appliquées depuis plusieurs années à des problèmes d'optimisation de forme et contrôle optimal des écoulements de complexité croissante. Leur emploi a souvent été couronné de succès, ces méthodes disposant de nombreuses qualités : preuve de convergence, taux de convergence superlinéaire possible (faible nombre d'évaluations nécessaire), faible dépendance vis-à-vis de la dimension du problème n, prise en compte de contraintes complexes, ... Néanmoins, plusieurs difficultés limitent l'usage de ces méthodes :

- Pour certains problèmes, la fonction de coût est non-différentiable par nature ou présente des discontinuités (par exemple les problèmes de type Min Max). Dès lors, ces méthode ne peuvent pas être employées sans aménagement spécifique.

Algorithm 1 Algorithme modèle pour les approches classiques

(0) initialisation
choix d'un vecteur des variables de contrôle initial x_c^0
$k \leftarrow 0$
(1) début de la boucle d'optimisation (itération k)

(2) résolution des équations d'état
obtention de $W(x_c^{(k)})$

(3) calcul de la fonction de coût
obtention de $\mathcal{J}(x_c^{(k)}, W(x_c^{(k)}))$

(4) calcul du gradient de la fonction de coût
obtention de $\nabla\mathcal{J}(x_c^{(k)}, W(x_c^{(k)}))$

(5) calcul du nouveau vecteur des variables de contrôle
obtention de $x_c^{(k+1)}$ à partir de $\mathcal{J}(x_c^{(k)}, W(x_c^{(k)}))$ et $\nabla\mathcal{J}(x_c^{(k)}, W(x_c^{(k)}))$

(6) fin de la boucle d'optimisation
$k \leftarrow k+1$
si convergence atteinte STOP sinon Goto (1)

- Le calcul du gradient de la fonction de coût peut être délicat. En effet, celui-ci peut être évalué par différentes approches :
  - Approximation par différences finies.
    Ce choix est généralement à proscrire, étant donné le nombre important d'évaluations à réaliser (au minimum $n+1$). En outre, de larges erreurs peuvent apparaitre, dépendant du pas de discrétisation. Un pas trop large génère une erreur de troncature et un pas trop petit entraine une erreur d'arrondi. Il est à noter que cette approche est la seule possible lorsque la fonction de coût est évaluée par l'intermédiaire d'un code de calcul commercial, dont les sources sont inconnues.
  - Résolution du problème adjoint[12].
    Cette approche est limitée car elle nécessite l'écriture d'un algorithme dérivé de celui résolvant les équations d'état, dont le développement se révèle au mieux coûteux en temps humain et au pire délicat à réaliser pour des problèmes complexes.
  - Calcul par différentiation automatique[5].
    Cette approche, consistant à générer automatiquement un code évaluant la différentielle de la fonction de coût est expéditive, mais est encore limitée à des problèmes (et des codes de calcul) modérément complexes.

- Un bruitage d'erreur, caractérisé par des irrégularités haute fréquence de la fonction de coût, est généralement observé lorsque la fonction de coût est évaluée par des méthodes numériques complexes (code de calcul). Des études ont montré que ce bruitage est amplifié par les méthodes de discrétisation des EDP d'ordres élevés et un critère de convergence faible. Par suite, ce bruitage peut avoir des conséquences funeste pour l'évaluation du gradient de la fonction de coût et mener l'algorithme d'optimisation à l'échec.

- La fonction de coût peut présenter de multiples minima locaux. Or les méthodes classiques ne sont pas capables de détecter et éviter ceux-ci, convergeant vers le premier minimum trouvé, peut-être caractérisé par une valeur de la fonction de coût médiocre, en comparaison du minimum absolu.

Ces différents points expliquent pourquoi les méthodes classiques sont souvent écartées pour la résolution de problèmes complexes à visées industrielles. Ces différents arguments justifient donc le développement d'algorithmes *ne reposant pas sur l'évaluation du gradient de la fonction de coût*, ni sur la théorie du calcul des variations. Ces méthodes sont généralement appelées *approches boites noires*, bien que ce terme ne signifie nullement qu'on ignore totalement les caractéristiques du système étudié !

1.3 Classification

Les approches de type boites noires peuvent être classées en plusieurs catégories :

- Les approches déterministes :

  - Les méthodes de type simplexe
    Elles consistent à évaluer la fonction de coût en un ensemble de $n+1$ vecteurs des variables de contrôle (formant un simplexe dans $\Re^n$), puis à déplacer cet ensemble dans l'espace des variables de contrôle selon les résultats obtenus. Ces approches ont été proposées par Spendley, Hext et Himsworth en 1962[21], étendues par Nelder et Mead en 1965[16] et revisitées récemment par Dennis et Torczon[6] en 1991.

  - Les méthodes de recherche dans une famille de directions
    Elles consistent à minimiser successivement la fonction de coût dans une famille de n directions, par une suite d'évaluations. Cette approche a été proposée par Rosenbrock en 1960[20] et modifiée par Powell en 1964[17] et par Swann la même année[23].

  - Les méthodes à surface de réponse
    Elles consistent à construire un modèle simple de la fonction de coût, par évaluations de la fonction de coût correspondant à différents vecteurs des variables de contrôle, puis de minimiser ce modèle par une approche classique. Il ne s'agit pas à proprement parler d'une alternative aux méthodes classiques, mais d'une modélisation de la fonction de coût permettant d'utiliser les méthodes à base de gradient.

  - Les méthodes d'interpolation
    Elles consistent à interpoler localement la fonction de coût en différents vecteurs des variables de contrôle, puis à minimiser localement cette interpolation dans une région de confiance. Le résultat permet alors de mettre à jour les supports de l'interpolation, dans une procédure itérative. Ces méthodes ont été proposées par Winfield en 1973[25], puis revisitées par Powell en 1994[18, 19] et d'autres auteurs récemment[3, 4, 14].

- Les approches stochastiques :

  - L'algorithme du recuit simulé
    Celui-ci s'appuie sur une analogie avec le phénomène de cristallisation en métallurgie. Ce processus alterne des cycles de refroidissement lent et de rechauffage (recuit) qui tendent à minimiser l'énergie du materiau. Cette approche a été proposée par Aarts et Korst an 1989[1].

  - Les algorithmes génétiques
    Ils consistent à mimer le processus d'évolution darwinienne, qui tend à produire des individus les plus adaptés possible à un environnement donné. Holland a proposé cette approche en 1975[11], qui a été popularisée par Goldberg en 1989[10].

  - Les algorithme d'essaim
    Ils sont basés sur une analogie avec le comportement collectif d'un groupe d'insectes. Chaque membre du groupe tend à trouver la meilleure source de nourriture, compte tenu de sa mémoire et des communications venant des autres membres. Ces algorithmes ont été mis en oeuvre par Kennedy et Eberhart en 1995[13].

Cette classification n'est pas exhaustive, mais regroupe les principales stratégies employées à l'heure actuelle.

1.4 Calculs parallèles

Les méthodes de type boites noires disposent de moins d'informations que les méthodes classiques, car le gradient de la fonction de coût n'est pas employé lors de la progression vers l'optimum. Par suite, *les méthodes boites noires sont généralement moins efficaces que les méthodes classiques* (en terme de nombre d'évaluations), quand celles-ci peuvent être utilisées.

En revanche, certaines approches boites noires ont une structure algorithmique adaptée à l'emploi des architectures parallèles, dans la mesure où elles ont recours à des évaluations simultanées et indépendantes (*instanciations*) de la fonction de coût. Par suite, ces instanciations peuvent être réparties sur plusieurs processeurs, ce qui se traduit par une réduction significative du temps de calcul réel. En pratique, les approches boites noires peuvent donc se révèler concurrentielles

avec les approches classiques en terme de temps de calcul. Bien qu'il s'agisse d'un aspect purement informatique, les stratégies de calcul parallèle doivent être inclues aux reflexions précédant la construction d'un algorithme moderne. Les deux méthodes analysées dans la suite de ce cours répondent à ces préoccupations.

2 Une méthode déterministe : "Multi-directional Search Algorithm"

2.1 Algorithme basique

La méthode "Multi-directional Search Algorithm" (MSA) développée par Torczon[24], fait partie des méthodes de type simplexe et a été conçue pour une utilisation sur architecture parallèle.

Cette méthode consiste à déplacer un simplexe de $n+1$ sommets dans $\Re^n$ (un triangle dans $\Re^2$, un tétraèdre dans $\Re^3$, ...), chaque sommet représentant un vecteur de n variables de contrôle. L'algorithme débute par une initialisation, consistant à générer l'ensemble des sommets du simplexe $s_0^{(0)}, s_1^{(0)}, \ldots, s_n^{(0)}$, par exemple en perturbant successivement d'une quantité δ les composantes d'un vecteur initial des variables de contrôle $s_0^{(0)} = x_c^{(0)}$:

$$s_i^{(0)} = s_0^{(0)} + \delta e_i \quad \text{avec } e_i^T = [\delta_{1i}, \ldots, \delta_{ni}] \quad i = 1, \ldots, n \tag{6}$$

où δ_{ji} est le symbole de Kronecker. Ensuite, les déplacements du simplexe sont réalisés de manière à faire décroître la fonction de coût évaluée au meilleur des sommets. A chaque itération k, on suppose que $s_0^{(k)}$ est le meilleur sommet du simplexe. L'ensemble des sommets du simplexe est déplacé homothétiquement par rapport à $s_0^{(k)}$:

$$\widetilde{s_i^{(k)}} = (1+\alpha) s_0^{(k)} + \alpha s_i^{(k)} \quad i = 0, \ldots, n \tag{7}$$

avec α généralement pris égal à 1. Ce déplacement est appelé *réflexion*. En cas de succès, i.e. au moins un des sommets obtient une valeur de la fonction de coût inférieure à celle de $s_0^{(k)}$, le simplexe subit alors une *expansion*, destinée à poursuivre la recherche dans les directions considérées :

$$s_i^{(k+1)} = \gamma \widetilde{s_i^{(k)}} + (1-\gamma) \widetilde{s_0^{(k)}} \quad i = 0, \ldots, n \tag{8}$$

avec γ généralement pris égal à 2. Au contraire, en cas d'échec lors de la réflexion, le simplexe subit une *contraction*, pour laquelle les sommets se rapprochent du meilleur sommet $s_0^{(k)}$:

$$s_i^{(k+1)} = \beta \widetilde{s_i^{(k)}} + (1-\beta) \widetilde{s_0^{(k)}} \quad i = 0, \ldots, n \tag{9}$$

avec β généralement pris égal à $-1/2$. Après avoir réalisé une expansion ou une contraction, l'agorithme débute une nouvelle itération. Une vue synthétique de la méthode est donnée par l'algorithme (2), tandis que les mouvements dans $\Re^2$ sont illustrés par la figure (1).

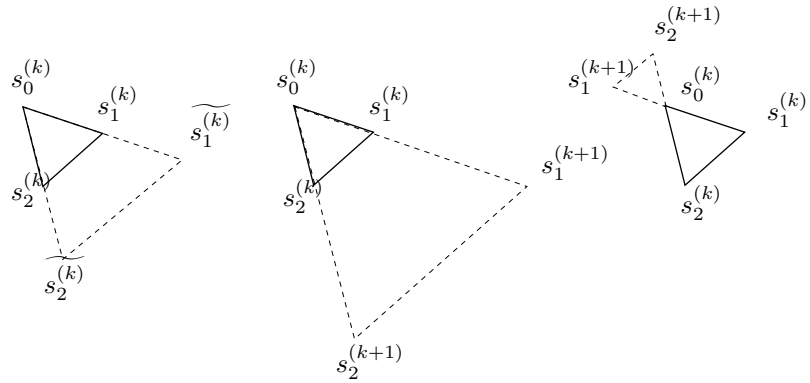

FIG. 1 – Exemple des mouvements de réflexion, expansion et contraction dans $\Re^2$

Le méthode MSA effectue donc une recherche du vecteur des variables de contrôle optimales par une suite de n évaluations simultanées et indépendantes de la fonction de coût. Ces instanciations peuvent être réalisées sur une architecture parallèle comprenant n processeurs. L'algorithme résultant effectue donc $2n$ évaluations de la fonction de coût par itération, pour un coût en terme de temps de calcul de 2 évaluations de la fonction de coût par itération.

Algorithm 2 Algorithme MSA basique

(0) initialisation

initialisation par perturbation $s_i^{(0)} = s_0^{(0)} + \delta e_i \quad i = 1, \ldots, n$
évaluation de la fonction de coût en $s_i^{(0)} \quad i = 0, \ldots, n$
$k \leftarrow 0$

(1) début de la boucle d'optimisation (itération k)

(2) mouvement de réflexion

recherche du meilleur sommet $s_0^{(k)}$
réflexion : $\widetilde{s_i^{(k)}} = (1+\alpha)s_0^{(k)} + \alpha s_i^{(k)} \quad i = 0, \ldots, n$
évaluation de la fonction de coût en $\widetilde{s_i^{(k)}} \quad i = 1, \ldots, n$

(3) si amélioration : mouvement d'expansion

expansion : $s_i^{(k+1)} = \gamma\widetilde{s_i^{(k)}} + (1-\gamma)\widetilde{s_0^{(k)}} \quad i = 0, \ldots, n$
évaluation de la fonction de coût en $s_i^{(k+1)} \quad i = 1, \ldots, n$

(4) sinon : mouvement de contraction

contraction : $s_i^{(k+1)} = \beta\widetilde{s_i^{(k)}} + (1-\beta)\widetilde{s_0^{(k)}} \quad i = 0, \ldots, n$
évaluation de la fonction de coût en $s_i^{(k+1)} \quad i = 1, \ldots, n$

(6) fin de la boucle d'optimisation

$k \leftarrow k+1$
si convergence atteinte STOP sinon Goto (1)

2.2 Extensions pour le calcul parallèle

Si l'utilisateur dispose d'un nombre de processeurs supérieur à n, Torczon propose une extension accélérant encore la convergence. Cette extension consiste à évaluer simultanément au résultat de la réflexion, les futurs mouvements possibles. Par exemple, si l'utilisateur dispose de $3n$ processeurs, on évalue simultanément les sommets du simplexe après réflexion, expansion et contraction, pour un coût en terme de temps de calcul d'une seule évaluation. Parmi ces évaluations, certaines seront inutiles, car le simplexe effectuera soit une expansion, soit une réduction pour l'itération actuelle. Mais ces mouvements étant déjà évalués, le coût final de l'itération en terme de temps de calcul sera seulement une évaluation.

On peut même étendre encore cette approche, en évaluant systématiquement les m itérations futures possibles simultanément. De nombreuses évaluations fourniront des informations inutiles à l'algorithme, mais certaines seront utiles et le temps de calcul sera réduit pour l'utilisateur.

2.3 Résultat théorique

Contrairement à la plupart des approches boites noires, la méthode possède une preuve de convergence. On suppose que $\{s_0^{(k)}\}$ est la suite des meilleurs sommets du simplexe. On définit le domaine :

$$L(s_0^{(0)}) = \{x \in \Re^n : \mathcal{J}(x) \leq \mathcal{J}(s_0^{(0)})\} \tag{10}$$

Pour $y \in \Re^n$, on définit le contour :

$$C(y) = \{x \in \Re^n : \mathcal{J}(x) = \mathcal{J}(y)\} \tag{11}$$

On note $X^\star$ l'ensemble des points stationnaires de $\mathcal{J}$ dans $L(s_0^{(0)})$ On peut ainsi montrer[6] le théorème suivant :

Théorème de convergence de la méthode MSA 2.1 *Si $L(s_0^{(0)})$ est un compact et $\mathcal{J}$ est continûment différentiable sur $L(s_0^{(0)})$, alors il existe une sous-suite de $\{s_0^{(k)}\}$ qui converge vers $x_c^\star \in X^\star$. Par suite, $\{s_0^{(k)}\}$ converge vers $C^\star = C(x_c^\star)$*

dans le sens où :

$$\lim_{k\to\infty}\left[\inf_{x\in C^\star}\|s_0^{(k)}-x\|\right]=0 \tag{12}$$

Ce résultat peut être obtenu à partir de deux points essentiels : les vecteurs $s_0^{(k)}-s_i^{(k)}$ sont linéairement indépendants par construction. Par suite, au moins un de ces vecteurs n'est pas orthogonal à $\nabla\mathcal{J}(s_0^{(k)})$ et produira une direction de descente si $s_0^{(k)}$ n'est pas un point stationnaire. L'algorithme ne peut donc pas converger vers un point non stationnaire. En outre, une suite d'itérations avec échecs génère pour chaque direction une recherche unidimensionnelle de type "backtracking", ce qui assure le résultat de convergence précédent.

2.4 Synthèse

La méthode MSA peut finalement se présenter comme une alternative aux méthodes classiques. Elle permet de s'affranchir du calcul du gradient de la fonction de coût. Par suite, elle est bien adaptée aux problèmes pour lesquels la fonction de coût présente un fort bruitage d'erreur. En revanche, elle converge *a priori* vers un optimum local et n'assure pas de déterminer l'optimum global de la fonction de coût.

3 Une approche stochastique : les algorithmes génétiques

Les algorithmes génétiques sont des méthodes stochastiques basées sur une analogie avec des systèmes biologiques. Ils reposent sur un codage des variables en structures chromosomiques et prennent modèle sur les principes de l'évolution naturelle pour déterminer une solution optimale. Ils ont été initialement développés par Holland [11] et popularisés par Goldberg[10]. Ces algorithmes sont caractérisés par une grande robustesse et possèdent la capacité d'éviter les minima locaux pour effectuer une optimisation globale.

3.1 Principes

Les algorithmes génétiques s'appuient sur un paradigme darwinien de l'évolution génétique d'une population. Dans la nature, les individus d'une population se trouvent en compétition, par exemple lors de la recherche de nourriture ou pour la reproduction. D'après les théories de l'évolution, les individus les plus adaptés survivent majoritairement et ont une descendance. Par suite, les caractéristiques génétiques de ces individus sont transmis à la génération suivante, qui voit potentiellement sa performance augmenter. Le patrimoine génétique de la population s'améliore ainsi de générations en générations.

Un modèle caricatural de cette évolution est mise en oeuvre par les algorithmes génétiques : un procédé de codage est introduit, de manière à générer une structure génétique à partir des variables du problème d'optimisation. On définit ensuite des opérateurs permettant l'évolution du patrimoine génétique de la population d'une génération à l'autre. Ces opérateurs font largement appel à des lois aléatoires. La performance d'un individu, qui détermine sa capacité à survivre, est liée à la valeur de la fonction de coût. On peut définir la performance comme l'inverse où l'opposé de la fonction de coût. La procédure d'optimisation consiste alors à simuler l'évolution d'une population, durant un certain nombre de générations, jusqu'à la détermination d'un individu optimal.

La mise en oeuvre de ces algorithmes a montré leur robustesse et leur capacité à déterminer le minimum global d'un problème. Cependant, le nombre d'évaluations de la fonctionnelle à effectuer est considérablement supérieur au nombre d'évaluations requis par un algorithme classique.

3.2 Algorithme historique

3.2.1 Algorithme

Il existe de nombreux algorithmes génétiques, qui sont généralement des variantes de l'algorithme historique développé par Holland[11]. L'algorithme débute par le choix d'un codage permettant de représenter les variables du problème d'optimisation par une structure génétique. Une population initiale est ensuite créée en générant n_{ind} individus aléatoirement. La performance de ces individus est alors calculée, nécessitant n_{ind} évaluations de la fonction de coût.

A chaque génération, trois opérateurs interviennent, modifiant les caractéristiques génétiques de la population. L'opérateur de *sélection* agit en premier. Son rôle est de choisir les individus de la population qui vont survivre. Le critère déterminant la sélection est lié à la performance des individus. La population après sélection compte toujours n_{ind} individus, certains individus de la population ayant disparu et d'autres ayant été dupliqués. L'opérateur de *croisement* intervient ensuite. Il crée de nouveaux individus en recombinant le patrimoine génétique de la population. Pour cela, les individus

sont appariés aléatoirement, chaque couple de parents créant deux enfants par un échange partiel de chromosomes. A la fin de l'étape de croisement, la population compte n_{ind} individus enfants. Enfin, l'opérateur de *mutation* crée de nouveaux individus en introduisant de nouvelles caractéristiques génétiques dans la population des n_{ind} enfants. Certains chromosomes sont perturbés aléatoirement, de manière à explorer l'espace de recherche. L'algorithme se poursuit alors avec une nouvelle génération.

Cet algorithme historique est illustré par le schéma algorithmique 3 et les différentes étapes sont décrites dans les paragraphes suivants.

Algorithm 3 Algorithme génétique historique

(0) initialisation
choix d'un codage
génération aléatoire de n_{ind} individus
$k \leftarrow 1$

(1) évaluation de la population (génération k)
calcul de la performance des n_{ind} individus

(2) sélection
détermination de n_{ind} survivants

(3) croisement
création de n_{ind} enfants à partir des survivants

(4) mutation
modification de certains enfants

(5) mise a jour
$k \leftarrow k + 1$ Goto (1)

3.2.2 Codage

Les algorithmes génétiques diffèrent des autres méthodes d'optimisation car ils utilisent un codage des variables de contrôle, plutôt que les variables de contrôle elles-mêmes. Le codage désigne le processus qui transforme les variables de contrôle en un *chromosome*.

$$x_c = [(x_c)_1, (x_c)_2, \ldots, (x_c)_n] \in \Re^n \rightarrow [a_1, a_2, \ldots, a_l] \tag{13}$$

Les éléments contituant le chromosome sont les *gènes* . Ils appartiennent à un ensemble appelé *alphabet*. L'algorithme historique emploie un codage binaire qui s'appuie sur l'alphabet $\{0, 1\}$, pour des raisons qui apparaitront plus tard.

Le codage binaire suppose en pratique la définition d'un intervalle admissible pour les variables de contrôle et une discrétisation de cet intervalle. Pour chaque variable de contrôle réelle $(x_c)_i$, on définit une limite inférieure l_i^{inf} et une limite supérieure l_i^{sup} au domaine de variation. En supposant que la discrétisation fournisse une précision de N_i^{cs} chiffres significatifs, le nombre de gènes nécessaire au codage de la variable $(x_c)_i$ est le plus petit entier l_i vérifiant :

$$(l_i^{sup} - l_i^{inf}) 10^{N_i^{cs}} \leq 2^{l_i} - 1 \tag{14}$$

Un chromosome étant constitué des gènes associés à chaque variable de contrôle mis bout à bout, le nombre total de gènes d'un chromosome, caractérisant un individu et un point dans l'espace de recherche est alors :

$$l = \sum_{i=1}^{n} l_i \tag{15}$$

La transformation d'un chromosome en variables réelles peut se faire de manière naturelle par la relation :

$$(x_c)_i = l_i^{inf} + \frac{l_i^{sup} - l_i^{inf}}{2^{l_i} - 1} \sum_{i_{bit}=1}^{l_i} 2^{i_{bit}-1} Ch(i_{bit}) \tag{16}$$

où $Ch(i_{bit})$ représente la valeur binaire, 0 ou 1, du gène i_{bit} associé à la variable $(x_c)_i$. On considère pour illustration un exemple simple, comprenant deux variables de contrôle. Les données sont les suivantes :

$$\begin{aligned} l_1^{inf} &= 0. \quad l_1^{sup} = 7. \quad N_1^{cs} = 0 \\ l_2^{inf} &= -2. \quad l_2^{sup} = 13. \quad N_1^{cs} = 0 \end{aligned} \tag{17}$$

Grâce à la relation 14, on détermine le nombre de gènes nécessaires au codage de chaque variable.

$$l_1 = 3 \quad l_2 = 4 \tag{18}$$

Un chromosome est donc caractérisé par $l = 7$ gènes. On considère, par exemple, le chromosome $[1001101]$ et on cherche à connaitre les variables réelles qui lui sont associées, en appliquant la relation 16. La première variable est caractérisée par le début de chaine $[\mathbf{100}1101]$ et correspond à la valeur

$$(x_c)_1 = 0. + 1.(2^0 \times \mathbf{1} + 2^1 \times \mathbf{0} + 2^2 \times \mathbf{0}) = 1. \tag{19}$$

La seconde variable est caractérisée par la fin de la chaine $[100\mathbf{1101}]$ et a pour valeur

$$(x_c)_1 = -2. + 1.(2^0 \times \mathbf{1} + 2^1 \times \mathbf{1} + 2^2 \times \mathbf{0} + 2^3 \times \mathbf{1}) = 9. \tag{20}$$

3.2.3 Opérateurs génétiques

Après avoir défini un codage, une population initiale de n_{ind} individus est générée aléatoirement. Chaque individu est caractérisé par un chromosome comptant l gènes et représente un point de l'espace de recherche. L'initialisation de la population peut donc être réalisée simplement en effectuant $n_{ind} \times l$ tirages au sort binaires, de manière à générer n_{ind} chromosomes. Le tirage aléatoire doit avoir une loi de probabilité uniforme, de façon à obtenir une occupation homogène de l'espace de recherche.

A chaque génération, on évalue la *performance* de chaque individu de la population, ce qui nécessite n_{ind} évaluations de la fonction de coût. Pour un problème de minimisation, la valeur opposée ou inverse de la fonction de coût peut être choisie. Ensuite, des opérateurs font évoluer les caractéristiques génétiques de la population, de manière à explorer l'espace de recherche et favoriser les caractéristiques des meilleurs individus pour les générations futures.

Sélection Le rôle de l'opérateur de sélection est de filtrer la population de manière à conserver les individus possédant de "bonnes" caractéristiques génétiques. La seule mesure dont on dispose de la qualité d'un individu est sa performance, liée à la valeur de la fonction de coût. Il semble donc naturel de sélectionner les individus pour lesquels la valeur de la fonction de coût est la meilleure et d'éliminer les autres. Néanmoins, cette approche doit être appréhendée avec précaution, puisqu'on souhaite conserver pour les générations suivantes les meilleures caractéristiques génétiques et pas seulement les meilleurs individus. Un individu peut en effet avoir certains gènes présents dans la solution optimale mais une performance médiocre, et son élimination serait néfaste. Cependant, les individus ayant une bonne performance sont plus susceptibles de posséder les bons gènes. On prend ainsi conscience de la notion de *pression de sélection*. Si la pression est trop forte, seul un petit nombre d'individus excellents survivent et la convergence de l'algorithme risque d'être prématurée, conduisant à un optimum local. Si la pression est trop faible, peu d'individus médiocres sont éliminés à chaque génération et la progression de la population est lente. En pratique, les opérateurs doivent tenir compte de cette notion de manière équilibrée.

l'algorithme historique emploie un opérateur appelé *sélection par roulette*. Celle-ci simule le fonctionnement d'une roulette de jeux pour déterminer les individus à sélectionner. A chaque individu correspond une portion de la roue et un lancer de roue est simulé par un tirage aléatoire. On sélectionne alors l'individu dont la portion de roue est désignée. On effectue n_{ind} lancers, de manière à sélectionner n_{ind} individus. Par conséquence, certains individus disparaissent de la population, tandis que d'autres sont dupliqués. Pour que la sélection soit efficace, les individus possédant la meilleure performance doivent être préférentiellement sélectionnés. On utilise donc une roulette "truquée", la portion alouée à chaque individu étant choisie proportionnellement à sa performance. La probabilité p_i pour un individu i de performance f_i d'être sélectionné est :

$$p_i = \frac{f_i}{\sum_{j=1}^{n_{ind}} f_j} \tag{21}$$

Un exemple de sélection par roulette pour quatre individus est fourni par le tableau 1.

individu	performance	probabilité	portion
1	0.565	0.18	$0 \rightarrow 0.18$
2	0.628	0.20	$0.18 \rightarrow 0.38$
3	0.377	0.12	$0.38 \rightarrow 0.50$
4	1.571	0.50	$0.50 \rightarrow 1.00$

No tirage	résultat aléatoire	individu sélectionné
1	0.354	2
2	0.879	4
3	0.567	4
4	0.157	1

TAB. 1 – Exemple de sélection par roulette

Croisement L'opérateur de croisement génère de nouveaux individus à partir des chromosomes des individus sélectionnés. Un échange partiel des chromosomes entre individus est réalisé, produisant donc de nouveaux individus, correspondant à de nouveaux points dans l'espace de recherche. Cependant, aucun gène nouveau n'est introduit dans la population. Par suite, cet opérateur effectue seulement une redistribution des gènes. Les individus sélectionnés possédant certaines caractéristiques génétiques intéressantes, on tente de recombiner celles-ci à travers l'opérateur de croisement.

Lors de cette étape, les individus sont groupés par paire de parents de manière aléatoire. On réalise ensuite un échange partiel des chromosomes avec une probablilité p_c, généralement proche de l'unité. Une forte probabilité favorise l'échange d'informations, mais augmente le risque de perdre de bonnes combinaisons.

L'algorithme historique emploie une technique appelée *coupure à un point*. Il consiste à choisir aléatoirement un point de coupe sur les deux chromosomes parents, puis à échanger les gènes à partir de ce point de coupe pour obtenir deux nouveaux individus enfants. La figure (2) illustre un croisement à un point.

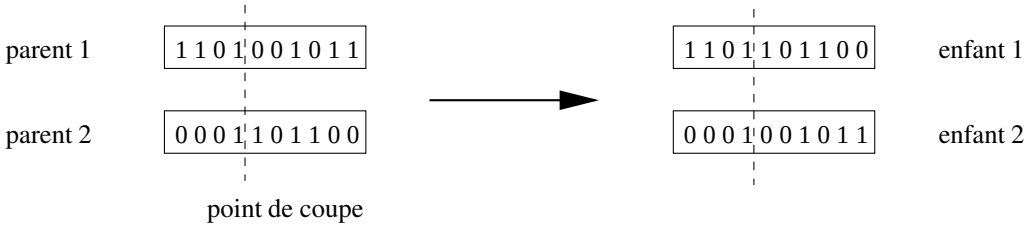

FIG. 2 – Exemple de croisement à un point

Mutation Le rôle de la mutation est d'introduire de nouvelles caractéristiques génétiques, ou de les réintroduire, en modifiant quelques gènes des individus enfants. Pour chaque gène de chaque enfant, une permutation est réalisée avec une probabilité p_m. Cette valeur est généralement faible, de manière à ne pas transformer l'algorithme en une simple recherche aléatoire. La mutation permet d'explorer l'espace de recherche aléatoirement et de réintroduire certains gènes qui ont disparu au cours des générations.

3.3 Quelques résultats théoriques

Les méthodes décrites précédemment sont particulièrement séduisantes car elles reposent sur des techniques simples et des analogies plaisantes et intuitives. Il est en effet confortable de penser qu'il suffit de mimer la nature pour résoudre élégamment un problème d'optimisation. Cependant, une analyse rigoureuse est nécessaire pour comprendre les mécanismes mis en jeu par les algorithmes génétiques, et par la suite percevoir leurs points forts et leurs faiblesses. Quelques résultats théoriques sont fournis dans les paragraphes suivants. On introduit la notion de *schémas*, qui a permis à Holland [11] de construire les premières bases théoriques, notamment le *théorème fondamental des algorithmes génétiques*. On constate ensuite qu'un premier résultat de convergence peut être facilement obtenu par application du théorème du point fixe de Banach[15].

3.3.1 Théorème des schémas

Notion de schémas Au vu des opérateurs intervenant dans l'algorithme génétique historique, on suppose intuitivement que l'algorithme cherche à combiner les "bons" génes présents dans la population. Pour étudier le devenir de quelques gènes en particulier, Holland [11] introduit la notion de schémas, en ajoutant à l'alphabet utilisé pour le codage un symbole $\star$, prenant une valeur quelconque de l'alphabet. Si on considère un codage binaire à l gènes, un schéma est constitué de l symboles appartenant à l'ensemble $\{0, 1, \star\}$. Par exemple, $[01 \star \star 1]$ est un schéma à $l = 5$ symboles, incluant les chromosomes $[01001]$, $[01011]$, $[01101]$, $[01111]$. Si on considère un codage binaire à l gènes, il existe 3^l schémas possibles. Un chromosome particulier appartient à 2^l schémas, puisque chacun de ses gènes peut être remplacé par le symbole $\star$. Par conséquent, une population comptant n_{ind} individus contient entre 2^l et $n2^l$ schémas, selon la diversité de la population.

De manière à qualifier un schéma, on introduit deux propriétés : la *longueur* $\delta(H)$ d'un schéma H est la distance séparant le premier et le dernier gène fixé. Par exemple, le schéma $H = [\star 1 \star \star 1]$ a pour longueur $\delta(H) = 3$. L'*ordre* $o(H)$ d'un schéma H est le nombre de gènes fixés. Pour l'exemple précédent, l'ordre est $o(H) = 2$.

Par la suite, on examine la manière dont ces schémas sont propagés de générations en générations à travers les opérateurs génétiques.

Le théorème fondamental des algorithmes génétiques [11] quantifie la croissance de la représentation d'un schéma donné au cours des générations. On considère un schéma H, de longueur $\delta(H)$ et d'ordre $o(H)$. On examine l'effet des opérateurs de sélection, croisement et mutation de l'algorithme génétique modèle sur ce schéma. On s'intéresse plus particulièrement à l'évolution du nombre $m(H, t)$ de représentants de ce schéma à la génération t.

Sélection L'action de l'opérateur de sélection est relativement simple dans le cas d'une sélection par roulette, pour laquelle la probabilité de sélection d'un individu est proportionnelle à sa performance. D'après 21, sachant que n_{ind} lancers de roulette sont effectués et que $m(H, t)$ représentants du schéma H sont présents dans la population, l'espérance du nombre de représentant du schéma après sélection $m(H, t')$ est :

$$E(m(H, t')) = m(H, t) n_{ind} \frac{f(H)}{\sum_{j=1}^{n_{ind}} f_j} \tag{22}$$

où $f(H)$ est la performance moyenne des représentants du schéma. En introduisant la performance moyenne de la population $\bar{f}$, on obtient :

$$E(m(H, t')) = m(H, t) \frac{f(H)}{\bar{f}} \tag{23}$$

On en déduit que le nombre de représentants d'un schéma après sélection augmente si la performance moyenne de ses représentants est supérieure à la performance moyenne de la population, le taux de croissance étant le rapport des performances. Si on suppose en outre que la performance moyenne des représentants d'un schéma reste au dessus de la performance moyenne de la population d'une quantité $c\bar{f}$ avec c constant, on a :

$$E(m(H, t')) = m(H, t) \frac{\bar{f} + c\bar{f}}{\bar{f}} = (1 + c) m(H, t) \tag{24}$$

Par suite, le nombre de représentants croît de manière exponentielle. L'effet de l'opérateur de sélection est traditionnellement résumé par l'idée suivante : il alloue un nombre exponentiellement croissant (respectivement décroissant) de représentants aux schémas dont la performance moyenne des représentants est supérieure (respectivement inférieure) à la performance moyenne de la population.

Croisement Pour étudier l'impact de l'opérateur de croisement sur la représentation d'un schéma, on suppose qu'un croisement à un point est utilisé. Avec cette hypothèse, tout représentant d'un schéma pour lequel le point de coupe est positionné entre deux gènes fixés est détruit (par souci de simplification, on ignore le cas où deux individus représentant le même schéma se croisent). Par exemple, le schéma $[1 \star \star 0 \star]$ sera détruit si le point de coupe intervient entre le premier et le quatrième gène. Finalement, la probabilité p_{sc} qu'un représentant d'un schéma survive au croisement dépend de la longueur du schéma. Parmi les $l - 1$ positions de coupe, il existe $\delta(H)$ positions provoquant la perte du représentant du schéma. Par suite, la probabilité de survie prend la forme :

$$p_{sc} = 1 - \frac{\delta(H)}{l - 1} \tag{25}$$

Sachant que le croisement est effectué avec une probabilité p_c, l'espérence du nombre de représentants d'un schéma H après sélection et croisement est :

$$E(m(H,t'')) \geq m(H,t)\frac{f(H)}{\bar{f}}(1-p_c\frac{\delta(H)}{l-1}) \tag{26}$$

Le résultat fournit une borne inférieure du nombre de représentants car on a considéré uniquement l'effet destructeur de l'opérateur. L'effet combiné des opérateurs de sélection et croisement est facilement interprétable : la représentation d'un schéma croît exponentiellement, d'un facteur dépendant de la performance moyenne de ses représentants et de la longueur du schéma.

Mutation On examine maintenant les effets de l'opérateur de mutation. Pour qu'un schéma survive à l'opérateur de mutation, il faut que tous les gènes fixés demeurent inaltérés. La probabilité de survie d'un gène étant $1-p_m$, la probabilité qu'un représentant d'un schéma d'ordre $o(H)$ survive est :

$$p_{sc} = (1-p_m)^{o(H)} \approx 1 - p_m o(H) \tag{27}$$

Il faut noter que, là encore, seul l'effet destructeur de l'opérateur de mutation est pris en compte.

Théorème fondamental des algorithmes génétiques 3.1 *L'espérence du nombre de représentants d'un schéma H disposant de $m(H,t)$ représentants à la génération t est en première approximation :*

$$E(m(H,t+1)) \geq m(H,t)\frac{f(H)}{\bar{f}}(1-p_c\frac{\delta(H)}{l-1} - p_m o(H)) \tag{28}$$

Par suite, les schémas courts, d'ordre faible, de performance au dessus de la moyenne voient leur nombre de représentants augmenter exponentiellement au cours des générations.

Interprétation et discussion Le théorème fondamental permet de mieux appréhender le comportement des algorithmes génétiques : plutôt que de chercher une bonne combinaison de gènes, ils simplifient le problème en cherchant à combiner les schémas courts, d'ordre faible, de performance au-dessus de la moyenne, généralement surnommés *briques constructives* (*building blocks*).

Pour expliquer le succès d'une telle approche, Holland [11] met en avant le nombre important d'informations traitées par les algorithmes génétiques à chaque génération. Bien que la population inclue entre 2^l et $n2^l$ schémas à chaque génération, tous ne sont pas traités de manière utile car certains sont détruits. Néanmoins, Holland montre que les algorithmes génétiques sont capables de traiter effectivement un nombre de schémas de l'ordre de n^3 à chaque génération. Ce nombre est largement plus important que le nombre d'évaluations n. Cette propriété est appelée le *parallélisme implicite*. Bien qu'un faible nombre de structures soit évalué, un nombre important d'informations en est retiré et utilisé.

Cette analyse milite pour l'utilisation de l'alphabet binaire, qui possède la plus faible cardinalité possible et permet la représentation du plus grand nombre de schémas par individus. En effet, un individu représente 2^l schémas parmi les 3^l schémas possibles pour le codage binaire. Pour un codage de cardinalité K, un individu représente 2^l schémas parmis les $(K+1)^l$ schémas possibles.

Le théorème fondamental permet également de déterminer les problèmes dont la résolution est délicate par les algorithmes génétiques. Il s'agit des problèmes ne vérifiant pas l'hypothèse de combinaison des briques constructives, pour lesquels la combinaison des schémas courts, d'ordre faible, et de performance au-dessus de la moyenne conduit à des schémas peu performants. on parle alors du phénomène de *déceptivité*. Cette qualification est peu intéressante en pratique car on ignore si le codage choisi conduit à un problème cohérent avec l'hypothèse de combinaison des briques constructives. Elle met néanmoins en lumière l'influence critique du codage dans le succès ou l'échec d'un algorithme génétique.

De nombreuses critiques à ce théorème ont été formulées, soulignant qu'il ne décrit le fonctionnement d'un algorithme génétique que de manière très grossière. Notamment, seuls les effets destructeurs des opérateurs sont comptabilisés, il ne prend pas en compte la variance de la performance parmis les représentants du schéma considéré et tous les schémas représentés par un individu n'ont pas tous la même performance.

3.3.2 Théorème du point fixe

Un résultat de convergence peut être facilement obtenu en appliquant le théorème du point fixe de Banach, pour un algorithme génétique légèrement modifié. On rappelle tout d'abord le théorème du point fixe de Banach :

Théorème du point fixe de Banach 3.1 *Soit (E, d) un espace métrique complet et $f : E \to E$ une application contractante. Alors il existe un point fixe unique $x^\star \in E$ pour f. De plus , quel que soit $x^{(0)} \in E$, la suite $\{x^{(k)}\}$ définie par $x^{(k+1)} = f(x^{(k)})$ converge vers $x^\star$.*

Une application f est contractante de E dans lui-même s'il existe $\lambda \in]0, 1[$ vérifiant $d(f(x), f(y)) \leq \lambda d(x, y)$ pour tout $(x, y) \in E^2$. On applique ce théorème à un algorithme génétique modifié en construisant un espace métrique dont les éléments sont les populations. On montre alors que l'algorithme converge vers une population unique, quelle que soit la population initiale.

Soit E l'ensemble des populations possibles de taille n_{ind}. On définit sur E la distance $d : E \times E \to \Re$ telle que :

$$d(P_1, P_2) = \begin{cases} 0 & \text{si } P_1 = P_2 \\ |1 + M - Eval(P_1)| + |1 + M - Eval(P_2)| & \text{sinon} \end{cases} \tag{29}$$

où $Eval()$ représente la performance moyenne de la population et M une limite supérieure à la performance de chaque individu. On peut facilement établir que (E, d) est un espace métrique. En outre, il est complet puiqu'il est de dimension finie.

On montre que le passage d'une génération à une autre est une application contractante pour un algorithme modifié. On considère un algorithme pour lequel toute nouvelle génération doit présenter une performance moyenne meilleure que celle de la génération précédente. Si ce n'est pas le cas, on recommence au début de la génération courante et on génère une nouvelle population par les opérateurs génétiques jusqu'à vérifier cette condition. On peut montrer dans ces conditions que le passage d'une génération à une autre est une application contractante. En effet, on vérifie alors $Eval(f(P_1)) > Eval(P_1)$ et $Eval(f(P_2)) > Eval(P_2)$ où f est l'application permettant de passer d'une génération à la suivante. Par suite :

$$\begin{aligned} d(f(P_1), f(P_2)) =& |1 + M - Eval(f(P_1))| + |1 + M - Eval(f(P_2))| \\ &< |1 + M - Eval(P_1)| + |1 + M - Eval(P_2)| \\ &< d(P_1, P_2) \end{aligned} \tag{30}$$

Par suite, on peut appliquer le théorème du point fixe de Banach, qui prouve que la suite des populations converge vers une population unique, quelle que soit la population initiale. Cette limite est le point fixe unique de l'application qui permet le passage d'un génération à une autre. Or, une population uniforme dont les individus correspondent au minimum de la fonction de coût est clairement ce point fixe (seule une population uniforme constituée d'individus optimaux ne peut pas générer une nouvelle population de valeur supérieure au sens de Eval(), quels que soient les opérateurs génétiques). Ce théorème permet donc de prouver la convergence d'un algorithme génétique modifié vers l'optimum global.

3.4 La famille des algorithmes génétiques

L'algorithme historique présenté précédemment est une analogie naturelle simple à mettre en oeuvre. Son analyse en terme de schémas propose une description intuitive de ses mécanismes qui aident à sa compréhension. En outre, une légère modification de celui-ci permet d'aboutir à une preuve de convergence vers l'optimum global du problème étudié.

Cependant, la mise en oeuvre pratique de cet algorithme, très général et robuste, met en évidence des taux de convergence généralement médiocres et des résultats variables selon les problèmes étudiés. En effet, il permet de s'attaquer à des problèmes d'origines très variées, mais est souvent mal adapté à la résolution d'un problème particulier. Cette constatation à motivé le développement de nouveaux opérateurs, plus adaptés à la résolution de certains problèmes, donnant ainsi naissance à la *famille des algorithmes génétiques*. Quelques unes de ces améliorations sont décrites dans les paragraphes suivants. Elles conduisent à des algorithmes génétiques "modernes", mieux adaptés aux problèmes qui nous intéressent dans le contexte de la mécanique des fluides.

3.4.1 Codage

Codage Gray Le codage, bien qu'il soit une simple conversion du système de variables, est important dans la mesure où il influe sur le comportement de l'algorithme. En effet, un mauvais choix de codage peut entrainer le phénomène de déceptivité décrit précédemment. Cependant, il est difficile, voire impossible de connaitre *a priori* les qualités d'un codage donné pour un problème spécifique.

Néanmoins, le codage binaire choisit pour l'algorithme historique est généralement un choix médiocre. Pour s'en convaincre, il suffit de constater que deux chromosomes variant d'un unique gène peuvent fournir deux valeurs des variables de contrôle très éloignées. Pour exemple, on considère le codage binaire des entiers entre 0 et 7 : [000] [001] [010] [011] [100] [101] [110] [111]. On constate que les valeurs 3 et 7, ou 1 et 5 correspondent à des chromosomes dont un seul gène diffère. Par suite, deux variables de contrôle très proches pourront avoir des représentations génétiques très

différentes. Cette caractéristique est généralement néfaste à la résolution d'un problème présentant certaines propriétés de régularité.

On peut facilement améliorer le codage binaire en introduisant un codage de type Gray. Une caractéristique de ce codage est que deux valeurs adjacentes ne diffèrent que d'un seul gène. L'algorithme permettant de transformer un codage Gray en codage binaire est fourni par le schéma algorithmique 4. Le codage Gray des entiers entre 0 et 7 devient : [000] [001] [011] [010] [110] [111] [101] [100]. L'utilisation d'un codage Gray est généralement bénéfique, pour la plupart des problèmes dont les variables de contrôle sont réelles.

Algorithm 4 Transformation d'une chaîne Gray $(g_k)_{k=1,\dots,n}$ en binaire $(b_k)_{k=1,\dots,n}$

(0) initialisation
chaîne Gray $(g_k)_{k=1,\dots,n}$
$k \leftarrow 1$
$last \leftarrow 0$

(1) transformation du gène k
si $(g_k = 1)$ alors
si $(last = 0)$ $b_k \leftarrow 1$
sinon $b_k \leftarrow 0$
sinon $b_k \leftarrow last$
$last \leftarrow b_k$

(2) mise a jour
$k \leftarrow k + 1$ Goto (1)

Codage réel Une autre solution pour avoir une représentation génétique proche du problème d'optimisation réel est d'employer le *codage réel* qui consiste à prendre comme chromosome le vecteur des variables de contrôle. L'alphabet a alors une cardinalité infini et le chromosome est le plus court possible (sa longueur est la dimension du problème d'optimisation n). Chaque gène a pour valeur la composante du vecteur des variables de contrôle qui lui est associé. Ce choix semble peu judicieux d'après l'analyse en terme de schémas (cf le nombre de schémas représentés par un individu), il a néanmoins montré de bons résultats en pratique. Notamment, le parallélisme implicite décrit précédemment ne dépend pas de la cardinalité du codage.

3.4.2 Opérateurs

Mise à l'échelle La sélection par roulette a deux inconvénients : tout d'abord, elle permet de traiter uniquement des problèmes pour lesquels la valeur de la performance est positive. Ensuite, elle peut avoir un mauvais comportement lorsque la performance de la population est très hétérogène, ou très homogène[10]. Il survient fréquemment durant les premières générations qu'un individu soit beaucoup plus performant que le reste de la population. Dans ce cas, cet individu a une probabilité d'être sélectionné très forte et de nombreuses copies seront présente dans la population future, ce qui risque de provoquer une uniformisation rapide de celle-ci et une convergence prématurée. Au contraire, lorsqu'on est proche de la convergence, l'écart des performances dans la population est faible. Dans ce cas, les individus ont presque la même probabilité d'être sélectionnés et l'évolution de la population est lente.

La sélection par roulette est donc généralement utilisée avec une mise à l'échelle des performances des individus [10]. On introduit une performance modifiée f' qui dépend linéairement de la performance brute f :

$$f' = af + b \tag{31}$$

Les coefficients a et b sont déterminés à chaque génération de manière à conserver la performance moyenne de la population et fixer un écart entre cette moyenne et la meilleure performance f'_{max}. La figure (3) illustre ce procédé. Ce changement d'échelle permet de contrôler la répartition des probabilités de sélection et par conséquent de maitriser la pression de sélection de l'opérateur.

Partage Lorsqu'une fonction de coût compte plusieurs optima de performances équivalentes, il serait souhaitable de voir s'établir plusieurs sous-populations, chacune tendant vers un optimum. L'algorithme modèle ne permet pas cette évo-

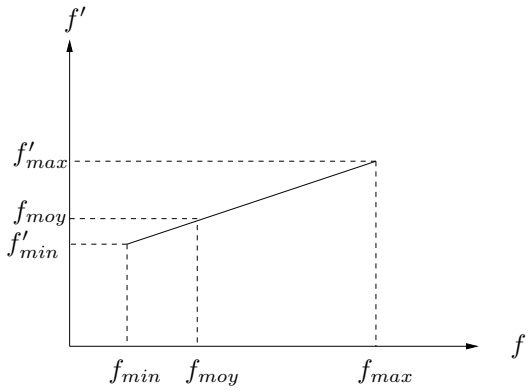

FIG. 3 – Mise à l'échelle

lution [10] car il suffit qu'un optimum apparaisse historiquement légèrement meilleur que les autres pour attirer progressivement la population entière vers lui : lorsque plusieurs individus performants sont regroupés à proximité d'un optimum, il existe une probabilité importante qu'ils survivent et qu'ils se croisent entre eux. Par suite, la génération suivante voit un nombre d'individus croissant au voisinage de cet optimum. Ce phénomène empêche la formation de sous-populations stables, la sous-population possédant les meilleurs individus étant sans cesse croissante, jusqu'à élimination des autres sous-populations. De manière à conserver un équilibre entre les sous-populations et maintenir la diversité à chaque génération, la technique de partage se propose d'handicaper les groupes d'individus trop proches. On introduit à cette fin une *fonction de partage* qui décroit avec la distance d_{ij} séparant deux individus i et j. Une forme standard de fonction de partage est :

$$\begin{array}{c} Sh(d_{ij}) = 1 - \dfrac{d_{ij}}{\sigma} \text{ si } d_{ij} \leq \sigma \\ 0 \text{ sinon} \end{array} \tag{32}$$

La distance d_{ij} peut être considérée dans l'espace des variables de contrôle (espace phénotypique), dans l'espace des chromosomes (espace génotypique) ou encore dans l'espace $\Re$ des performances (espace objectif), suivant les applications. Le coefficient σ permet de contrôler l'extension du partage au sein de la population. La performance f_i de l'individu i intervenant habituellement lors de la sélection est modifiée en :

$$\tilde{f}_i = \frac{f_i}{\sum_{j=1}^{n_{ind}} Sh(d_{ij})} \tag{33}$$

Cette performance modifiée est inférieure ou égale à la performance initiale et est d'autant plus réduite qu'un individu possède des voisins proches. Par conséquent, les individus isolés conservent leur performance initiale, tandis que les individus groupés autour des optima voient leur probabilité de sélection réduite. En pénalisant ceux-ci, on favorise l'émergence et le développement d'autres sous-populations.

Sélection par tournois L'origine des inconvénients de la sélection par roulette provient du choix d'accorder une probabilité de sélection proportionnelle à à la performance. Pour éviter les conséquences de ce choix, ou l'utilisation d'une mise à l'échelle, la sélection par tournois fait intervenir le rang, ou le classement, des individus plutôt que leur performance. On attribut à chaque individu un rang, en les classant par ordre décroissant de leur performance. Plus un individu est de rang faible et plus il a une forte probabilité d'être sélectionné. La sélection est donc déterminée par comparaison des performances plutôt que par la valeur de la performance elle-même.

La sélection par tournois consiste à simuler n_{ind} tournois, un individu étant sélectionné à chaque fois. A chaque tournoi, on prend au hasard n_{part} participants parmi la population. Le participant ayant le rang le plus faible est alors sélectionné. Le choix du nombre de participants permet de contrôler la pression de sélection : plus le nombre de participants est élevé, plus la pression est forte. Un exemple de sélection par tournoi à deux participants est donné par le tableau 2.

Elitisme La notion d'élitisme est introduite de manière à préserver la structure du meilleur individu à chaque génération. Ainsi, l'élitisme empêche qu'un individu particulièrement performant disparaisse au cours de la sélection, par manque de "chance", ou que ses "bonnes" combinaisons soient brisées par l'opérateur de croisement ou mutation

L'élitisme permet de remédier à ce problème. Après évaluation de la performance des individus à une génération t donnée, le meilleur individu de la génération $t-1$ est réintroduit dans la population si aucun des individus de la génération

individu	performance	rang
1	0.565	3
2	0.628	2
3	0.377	4
4	1.571	1

participants (aléatoire)	individu sélectionné
1 3	1
1 2	2
3 4	4
2 3	2

TAB. 2 – Exemple de sélection par tournoi

t n'est meilleur que lui. Par cette technique, l'évolution de la fonction de coût est monotone. On peut montrer que l'élitisme améliore généralement la performance de l'algorithme lorsque la fonction de coût possède un unique optimum, mais peut la dégrader lorsque de nombreux optima sont présents. Cette technique favorise finalement la recherche locale.

Croisement uniforme Lors d'un croisement uniforme, chaque gène des individus enfants est créé en copiant le gène correspondant d'un parent, choisi grâce à un masque. Pour chaque couple de parents, un masque binaire est généré aléatoirement. Pour chaque gène, lorsque le masque a pour valeur 1, le gène du premier enfant est la copie du gène du premier parent. Lorsqu'il a pour valeur 0, le gène du premier enfant est la copie du gène du second parent. La figure (4) illustre cette méthode.Cet opérateur de croisement favorise les recombinaisons complexes de gènes entre les individus appariés.

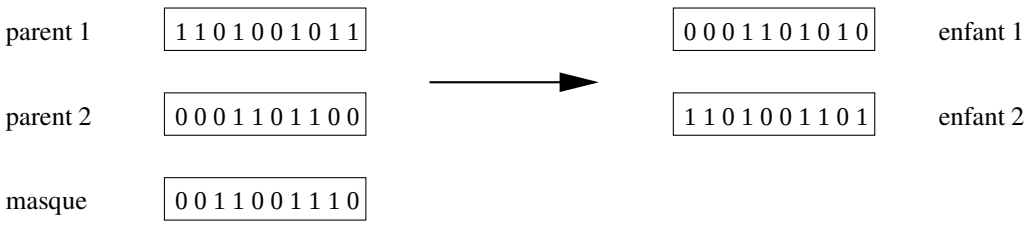

FIG. 4 – Exemple de croisement uniforme

Croisement heuristique Cet opérateur a été conçu pour tirer parti du codage réel et essayer d'améliorer la recherche locale en s'inspirant des méthodes de descente. Pour cela, étant donné deux parents caractérisés par des vecteurs des variables de contrôle x_c^1 et x_c^2, de performances f_1 et f_2 supposées ordonnées dans l'ordre décroissant, on génère deux enfants en effectuant une translation issue de x_c^1, de direction $\overrightarrow{x_c^2 x_c^1}$ et de longueur $\rho \|\overrightarrow{x_c^2 x_c^1}\|$ où $\rho \in [0,1]$ est un paramètre aléatoire. l'opérateur est illustré par la figure (5). On espère ainsi que la direction $\overrightarrow{x_c^2 x_c^1}$ est localement une direction de descente et que les enfants auront une performance supérieure à celle de leurs parents.

Mutation non-uniforme La mutation binaire n'étant pas adaptée à un codage réel, un nouvel opérateur est créé pour ce codage. C'est en outre l'occasion d'introduire un paramètre permettant de mieux contrôler la mutation. Le gène i ayant pour valeur la i-ème composante du vecteur des variables de contrôle $(x_c)_i$, celle-ci est modifiée par mutation de la manière suivante :

$$(x_c)_i' = \begin{cases} (x_c)_i + (l_i^{sup} - (x_c)_i)\,\rho\,(1-\frac{t}{T})^\beta & \text{si } rand = 1 \\ (x_c)_i - ((x_c)_i - l_i^{inf})\,\rho\,(1-\frac{t}{T})^\beta & \text{si } rand = 0 \end{cases} \tag{34}$$

où $rand$ est un tirage aléatoire binaire, décidant du signe de la perturbation, $\rho \in [0,1]$ est un tirage aléatoire, décidant de l'amplitude de la perturbation. Le dernier terme assure une décroissance de celle-ci au cours des générations, t représentant

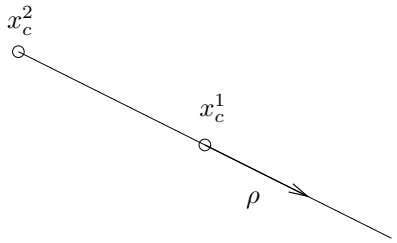

FIG. 5 – Croisement heuristique

la génération actuelle et T le nombre maximal de générations. β est un paramètre numérique généralement pris égal à deux. Cet opérateur de mutation permet donc d'obtenir de larges perturbations durant les premières générations, favorisant le recherche globale, puis des perturbations plus faibles lors des dernières générations, favorisant la recherche locale.

3.5 Optimisation multicritère

Les algorithmes génétiques sont une véritable philosophie de résolution plus qu'un n-ième optimiseur. Par suite, ils sont mieux adaptés à la résolution de certains problèmes qu'un simple optimiseur. L'optimisation multicritère est un exemple typique pour lequel les algorithmes génétiques apportent une méthodologie originale et innovante.

Tout au long des chapitres précédents, on cherchait à atteindre un point optimal selon un critère unique. Cependant, dans le domaine des sciences de l'ingénieur, l'optimalité est rarement définie par un seul critère. Ainsi, il existe généralement un ensemble d'indications, devant être intégrées lors des prises de décision. Le but des paragraphes qui suivent est de décrire les différentes approches destinées à traiter un problème multicritère, et souligner l'apport des algorithmes génétiques dans ce domaine.

3.5.1 Définitions

Problème Un problème d'optimisation multicritère consiste à optimiser simultanément plusieurs fonctions de coût, représentant chacune un critère de décision. Formellement, le problème peut s'écrire :

$$\text{minimiser } \mathcal{J}(x) = (\mathcal{J}_1(x), \ldots, \mathcal{J}_p(x)) \quad x \in \Re^n \tag{35}$$

Les fonctions de coût $\mathcal{J}_1(x), \ldots, \mathcal{J}_p(x)$ ne possèdent généralement pas un minimum commun. En conséquence, on ne peut généralement pas déterminer une solution unique à un problème multicritère. La solution est composée d'un ensemble de points représentant des compromis entre les différents critères.

Optimalité au sens de Pareto De manière à préciser la nature des points solution, on est amené à définir le concept de *dominance*. Un point x domine un point y si x est aussi performant que y pour tous les critères et est strictement meilleur pour au moins un critère :

$$x_c^1 \text{ domine } x_c^2 \Leftrightarrow \exists i \ \mathcal{J}_i(x_c^1) < \mathcal{J}_i(x_c^2) \text{ et } \mathcal{J}_j(x_c^1) \leq \mathcal{J}_j(x_c^2) \ \forall j \neq i \tag{36}$$

Lorsqu'on résout un problème d'optimisation multicritère, on cherche à obtenir *l'ensemble des points non-dominés*, appelé *front de Pareto*. En effet, ces points ne sont pas comparables entre eux, dans la mesure où il n'est pas possible de déterminer dans cet ensemble un point dominant un autre. Ils représentent donc des compromis entre les différents critères. Pour illustrer cette notion, on considère l'exemple à deux critères représenté par la figure (6). Le point x_c^1 est dominé par x_c^5 et domine x_c^4. En revanche, il n'est ni meilleur ni pire que (x_c^2 et x_c^3. Finalement, l'espace objectif peut être réparti en trois zones : la zone en gris foncé correspond aux points qui dominent x_c^1, celle en gris clair aux points dominés par x_c^1, la zone blanche représentant les points indifférents. Un exemple de front de Pareto est donné par la figure (7).

3.5.2 Méthode des pondérations

Les algorithmes étudiés dans les chapitres précédents ne peuvent prendre en compte qu'une unique fonction de coût. Pour traiter un problème multicritère, l'idée vient naturellement de construire une fonction de coût composite, représentative des différents critères, puis de la minimiser à l'aide d'un des algorithmes déjà décrits. Une approche classique consiste à introduire une pondération des différents critères :

$$\text{minimiser } \mathcal{J}_{pond} = \sum_{i=1}^{p} \omega_i \mathcal{J}_i \quad \sum_{i=1}^{p} \omega_i = 1 \tag{37}$$

Le choix des pondérations détermine l'importance relative des critères, chaque optimisation permettant de déterminer un point du front de Pareto. Outre le fait que de nombreux calculs doivent être réalisés pour obtenir une représentation du front, cette méthode a un inconvénient majeur : certains points du front de Pareto ne peuvent pas être atteints lorsque le front n'est pas convexe. Pour mettre en évidence ce phénomène, on considère le cas à deux paramètres exposé à la figure (8), pour lequel on cherche à minimiser la fonction de coût $\mathcal{J} = \omega_1 \mathcal{J}_1 + \omega_2 \mathcal{J}_2$. En écrivant $\mathcal{J}_2 = -\frac{\omega_1}{\omega_2}\mathcal{J}_1 + \frac{1}{\omega_2}\mathcal{J}$, on constate que le point atteint par la minimisation se situe au point de tangence du front et de la droite de pente $s = -\frac{\omega_1}{\omega_2}$, dont l'ordonnée à l'origine des abscisses est la plus faible possible. Par conséquent, pour toute pondération appliquée, les points du front sur la figure (8) entre x_c^2 et x_c^3 ne peuvent être obtenus par cette méthode. Lorsque $s < s_2$, les points atteints se situent à gauche de x_c^2 et lorsque $s > s_2$, ils sont à droite de x_c^3. les points de la partie concave du front ne peuvent pas être atteints. Cette approche se révèle finalement très limitée dans son champ d'application.

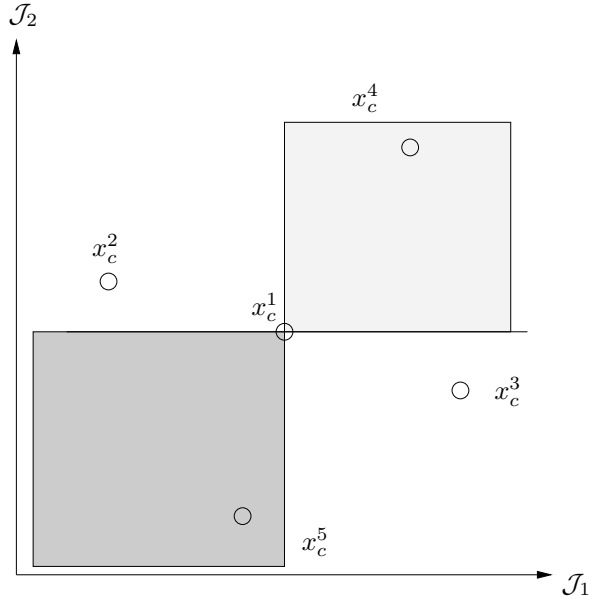

FIG. 6 – Notion de dominance

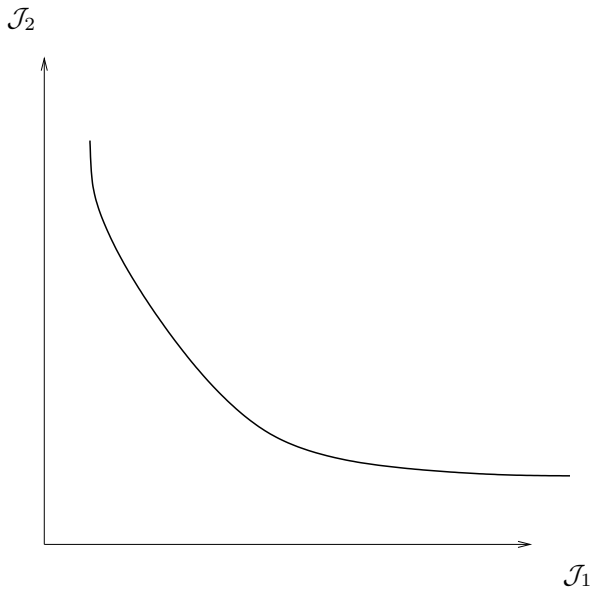

FIG. 7 – Front de Pareto

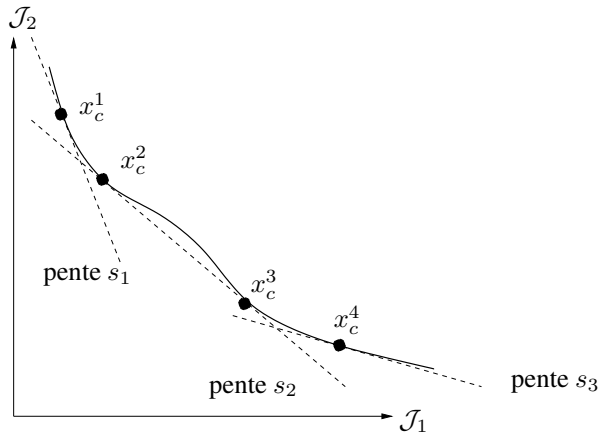

FIG. 8 – Méthode de pondération

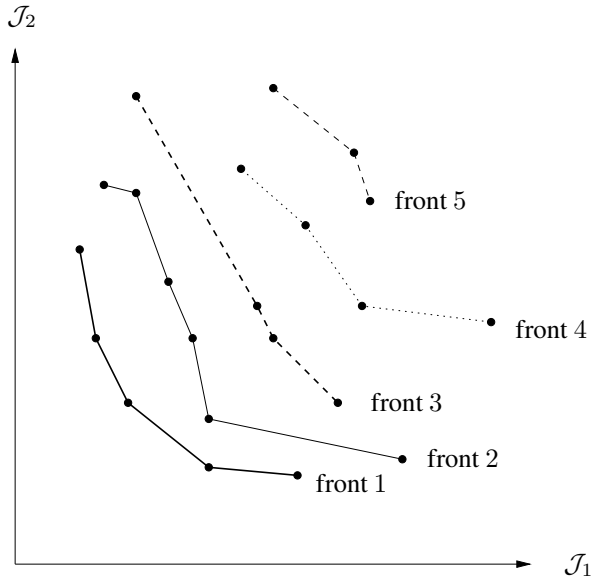

FIG. 9 – Classement par fronts

3.5.3 Approches génétiques

Contrairement aux algorithmes déterministes limités à la méthode des pondérations, les algorithmes génétiques se prêtent tout particulièrement à l'optimisation multicritère, dans la mesure où la solution du problème est composée d'un ensemble de points. Différentes stratégies ont été développées, pour lesquelles l'opérateur de sélection est modifié afin de tendre vers une population répartie le long du front de Pareto. On décrit dans ce cours l'approche "Non-Dominated Sorting Genetic Algorithm" (NSGA)[22].

Cette méthode propose d'attribuer aux individus une performance biaisée, dépendant de la distance qui les sépare du front de Pareto. A chaque génération, avant sélection, la population est classée par *fronts*. Les individus non-dominés constituent le front 1. Parmi les individus restants, les non-dominés forment le front 2, et ainsi de suite ... Un exemple est donné par la figure (9). Ensuite, une performance biaisée est attribuée à chaque individu, dépendant du front auquel il appartient. Les individus constituant le premier front se voient attribuer une même performance élevée. Ensuite, une stratégie de partage entre les individus du front est réalisée, suivant la distance qui les sépare dans l'espace phénotypique. Cette mesure favorise la diversité et permet l'étalement de la population le long du front de Pareto. Les individus du second front sont ensuite traités de la même manière, avec une performance initiale inférieure à la plus basse performance

attribuée dans le front précédent. La procédure se poursuit ainsi jusqu'à ce qu'une performance biaisée soit associée à chaque individu de la population. La sélection s'opère ensuite par un processus classique, en considérant comme critère la performance biaisée. Cette méthode est particulièrement simple à intégrer à un algorithme unicritère, puisqu'il suffit d'insérer la procédure d'attribution de la performance biaisée.

3.6 Synthèse

Les algorithmes génétiques représentent une philosophie de recherche générale, robuste, capable de traiter des problèmes présentant de nombreux optima locaux et des problèmes multicritères. Ils souffrent néanmoins d'une difficulté à aborder des problèmes réels et nécessitent un nombre d'évaluations de la fonction de coût très important.

4 Applications

Les méthodes décrites précédemment sont mises en oeuvre pour deux problèmes typiques en optimisation et contrôle des écoulements : l'optimisation de forme d'une aile 3D en régime transsonique (équations d'Euler) et l'optimisation des caractéristiques d'un jet synthétique pour le contrôle du décrochage d'un profil (équations de Navier-Stokes instationnaires en moyenne de Reynolds).

4.1 Optimisation de forme d'une aile en régime transsonique

On souhaite optimiser la forme d'une aile 3D réaliste pour un régime transsonique ($M_\infty = 0.83$). Ce cas test est défini précisément dans [2]. Les variables de contrôle sont au nombre de vingt et la fonction de coût est une fonction composite des coefficients de portance C_L et de traînée C_D :

$$\mathcal{J} = \frac{C_D}{C_{D_0}} + 10^4 \max(0, 0.999 - \frac{C_L}{C_{L_0}}) \tag{38}$$

C_{L_0} et C_{D_0} représentent les coefficients de portance et traînée pour un profil correspondant au NACA 0012, utilisé comme forme initiale pour l'algorithme déterministe. Le but de l'optimisation est donc de diminuer la traînée en conservant une portance fixe à 0.1% près. Les variables de l'écoulement sont obtenues par résolution des équations d'Euler (discrétisation MUSCL, reconstructions TVD, intégration temporelle par méthode de Newton) sur un maillage non structuré d'environ 30 000 noeuds.

Cet exercice est mené en utilisant la méthode MSA et divers algorithmes génétiques. On compare notamment les résultats obtenus à partir d'un codage réel et un codage binaire (120 gènes), pour différents opérateurs de sélection et croisement et différentes probabilités de mutation. La population compte 60 individus.

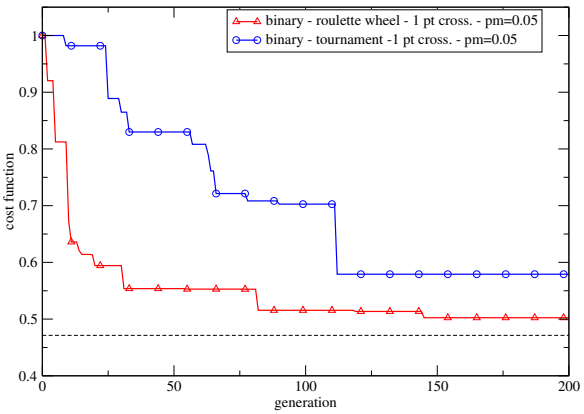

FIG. 10 – Influence de l'opérateur de sélection

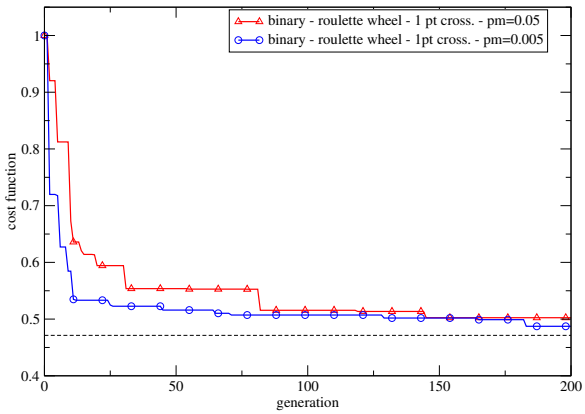

FIG. 11 – Influence de la probabilité de mutation

Les figures (10 - 13) montrent la réduction de la fonction de coût au cours des générations pour les algorithmes génétiques testés. On considère comme algorithme génétique de référence l'algoritme historique (codage binaire, sélection par roulette, croisement à un point) avec $p_c = 0.95$ et $p_m = 0.05$. Sur les figures, la ligne pointillée représente le niveau atteint par la méthode MSA.

La figure (10) montre que le choix d'une sélection par tournoi, avec une pression de sélection plus faible, est préjudiciable. Ce résultat est certainement dû au fait que la contrainte de portance dans la fonction de coût est sévère et exige une sélection élevée pour éliminer les formes qui violent la contrainte. La figure (11) illustre l'influence de la probabilité de mutation : une probabilité trop élevée peut être néfaste, dans la mesure où la recherche aléatoire domine l'algorithme au dépend de la recombinaison des bons individus. La figure (12) montre que de meilleurs résultats peuvent être obtenus en utilisant un codage binaire. L'influence de la mutation est alors cruciale : une probabilité plus élevée favorise la recherche globale durant les premières génération et la recherche locale durant les dernières générations. Enfin, la figure (13) montre l'intérêt du croisement heuristique, inspiré des méthodes de descente, pour améliorer la recherche locale.

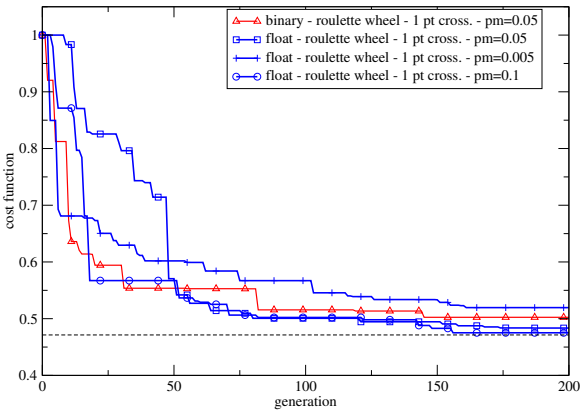

FIG. 12 – Influence de la probabilité de mutation pour un codage réel

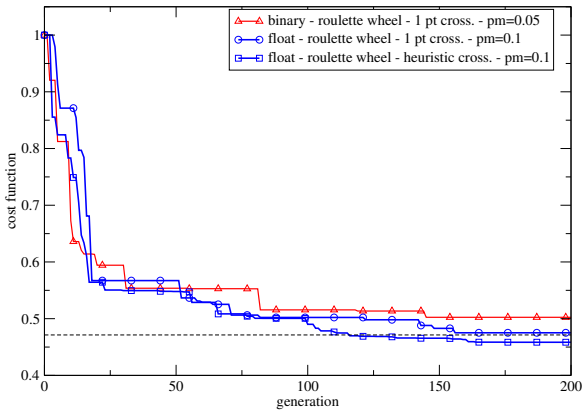

FIG. 13 – Influence de l'opérateur de croisement pour un codage réel

La figure (14) montre une comparaison des résultats obtenus par le meilleur des algorithmes génétiques testés et la méthode MSA, en terme de temps de calcul normalisé (rapport du temps de calcul global sur le temps de calcul d'une analyse). L'algorithme génétique a été executé sur 60 processeurs (n_{ind}) et la méthode MSA sur 20 processeurs (n). On constate que la méthode MSA converge vers un optimum local, tandis que l'algorithme génétique atteint un optimum de meilleure valeur. L'effort de calcul est considérablement supérieur pour l'algorithme génétique, qui nécessite 12000 évaluations de la fonctionnelle contre 4000 pour la méthode MSA.

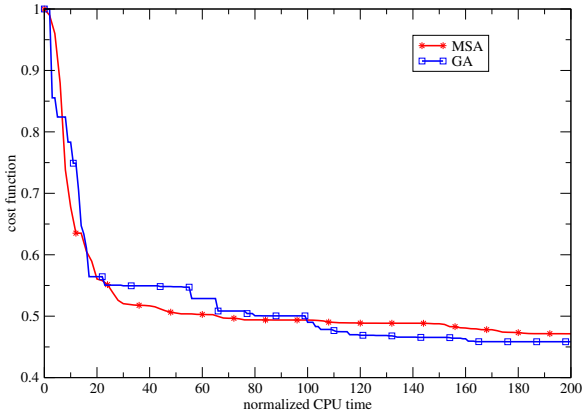

FIG. 14 – Comparaison méthodes MSA - AG

Les figures (15 - 16) comparent les formes obtenues à la base et à l'extrémité de l'aile. On constate que même si les formes ont des caractéristiques similaires, elles sont fort différentes. Ce qui prouve que l'algorithme génétique a été capable de trouver une région de l'espace de recherche correspondant à de meilleures performances (recherche globale), tandis que la méthode MSA a convergé vers la première région optimale trouvée (recherche locale). Les formes optimales sont caractérisées par un affinement du profil et une modification de la forme de l'extrados au niveau de l'onde de choc.

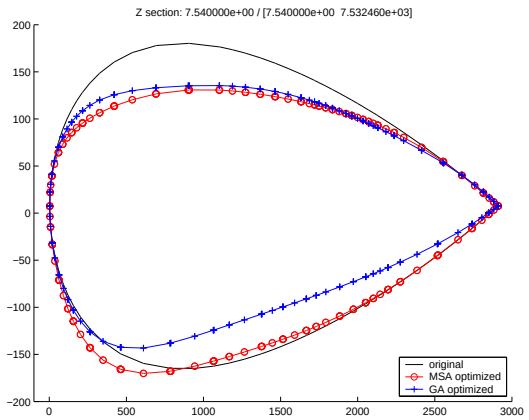

FIG. 15 – Comparaison des profils à la base de l'aile

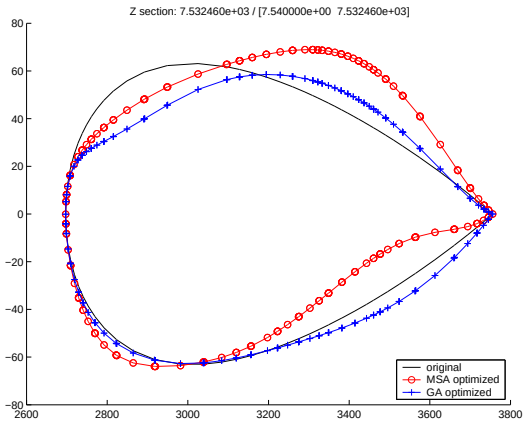

FIG. 16 – Comparaison des profils à l'extrémité de l'aile

Les figures (17-19) représentent le nombre de Mach pariéral pour la forme initiale et les formes optimales trouvées par la méthode MSA et l'algorithme génétique. On constate que la réduction de traînée à portance constante se traduit en grande partie par la réduction de l'intensité de l'onde choc générée sur l'extrados. Pour chaque forme optimale trouvée, des différences significatives sont néanmoins observées en ce qui concerne les écoulements, ce qui n'est pas étonnant au vu des caractéristiques de formes optimales.

4.2 Optimisation d'un jet synthétique pour le contrôle du décrochage

On s'intéresse dans ce paragraphe à l'optimisation des caractéristiques d'un jet synthétique (fréquence, intensité, direction) pour le contrôle du décrochage d'un profil NACA 0012. Seules trois variables de contrôle sont donc considérées. Les variables d'écoulement sont obtenues par résolution des équations de Navier-Stokes instationnaires en moyenne de Reynolds, pour un écoulement incompressible (couplage SIMPLE, schémas du 2nd ordre en temps et espace) avec une modélisation de la turbulence de type proche paroi (modèle $k-\omega$ SST). Le maillage compte environ 90 000 moeuds. La fonction de coût est l'opposé du coefficient de portance moyen au cours du temps. Chaque simulation est menée sur un intervalle de temps assez long pour que les effets transitoires disparaissent. On réalise une optimisation pour plusieurs incidences du profil, dans le but de repousser l'angle de décrochage et augmenter la portance maximale du profil. Des précisions pourront être trouvées dans [7].

Etant donné la quantité importante de calculs à mener (calculs instationnaires), seule la méthode MSA a été testée. La figure (20) montre l'évolution du coefficient de portance en fonction de l'incidence sans jet, pour un profil avec un jet et des paramètres initiaux (tirés d'expériences) et avec un jet et des paramètres optimisés. Les variations représentent l'amplitude de la portance au cours du temps. On constate que la portance maximale a été considérablement augmentée durant l'optimisation (+32% par rapport au cas initial). En outre, le comportement linéaire du coefficient de portance est conservée jusqu'au décrochage, qui a été repoussé de $18°$ à $22°$. Les figures (21-22) représentent les lignes de courant à l'instant de portance maximale pour les paramètres initiaux et optimisés. On observe que l'optimisation des caractéristiques du jet permet d'obtenir un écoulement pour lequel le décollement se produit plus en aval et les zones de décollement sont plus réduites que pour les paramètres initiaux.

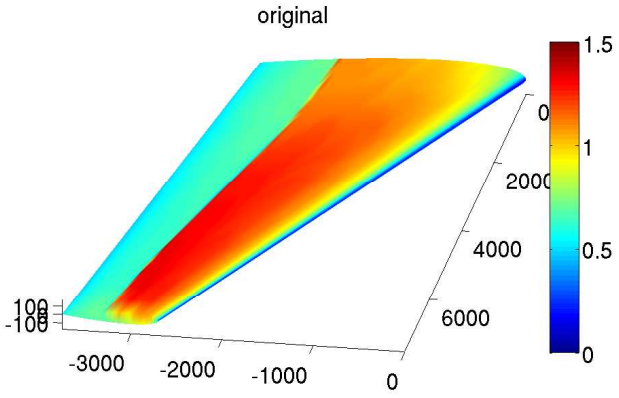

FIG. 17 – Nombre de Mach à la paroi pour la forme initiale

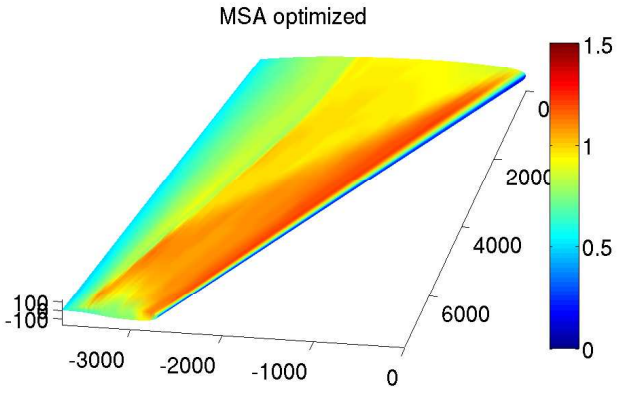

FIG. 18 – Nombre de Mach à la paroi pour la forme optimisée par la méthode MSA

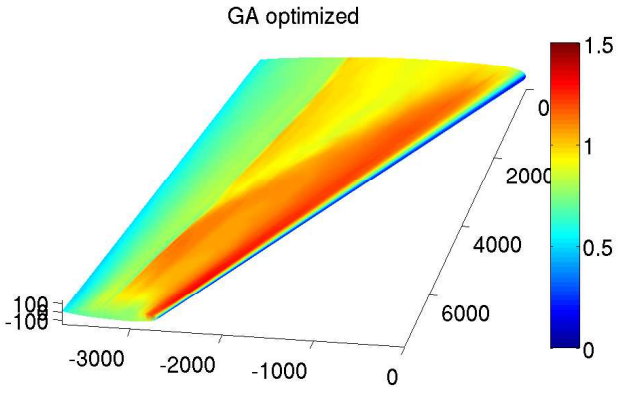

FIG. 19 – Nombre de Mach à la paroi pour la forme optimisée par l'algoritme génétique

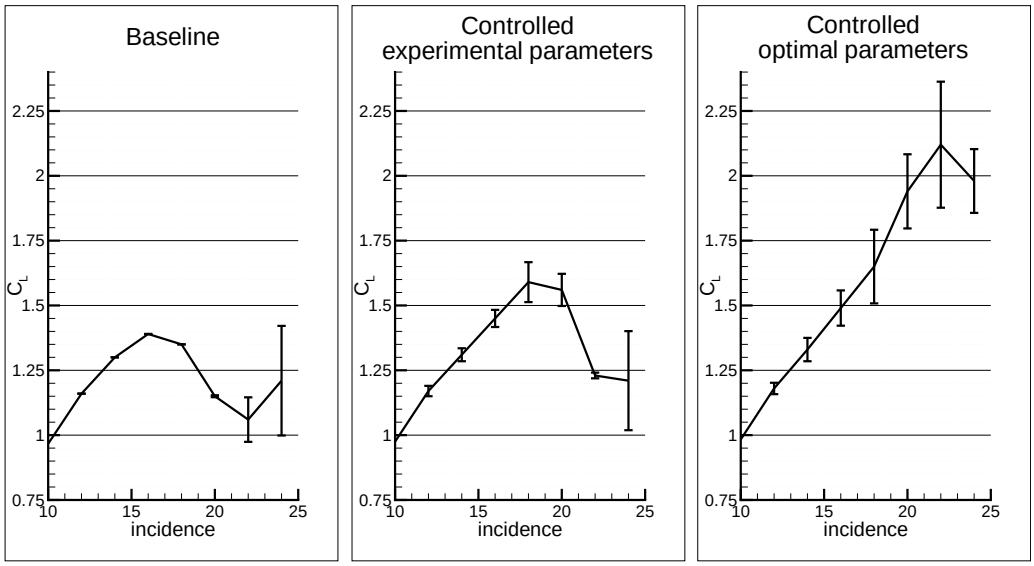

FIG. 20 – Evolution du coefficient de portance en fonction de l'incidence, sans jet (gauche), avec jet et des paramètres initiaux (centre), avec jet et paramètres optimisés (droite)

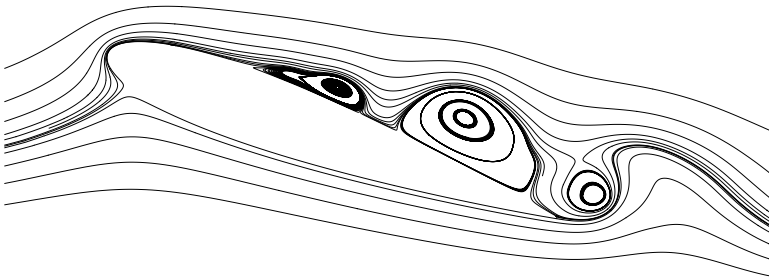

FIG. 21 – Lignes de courant à l'instant de portance maximale pour les paramètres initiaux

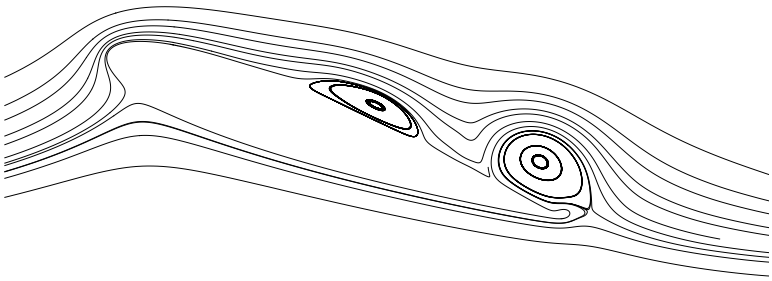

FIG. 22 – Lignes de courant à l'instant de portance maximale pour les paramètres optimisés

5 Conclusion

Les approches de type boites noires sont aujourd'hui utilisées dans de nombreux domaines. Elles sont intéressantes lorsque les méthodes classiques utilisant le gradient de la fonction de coût ne peuvent être utilisées ou fournissent des résultats médiocres (convergence locale). Les méthodes boites noires sont caractérisées par une grande robustesse et la capacité de réaliser une optimisation globale pour les approches stochastiques. Elles ont une structure leur permettant d'utiliser les architectures parallèles, ce qui leur permet de rester compétitives en terme de temps de calcul.

Parmi celles-ci, les algorithmes génétiques représentent une philosophie de recherche très générale, offrant la possibilité de résoudre des problèmes avec de nombreux optima locaux ou multicritères. Malgré les modifications apportées à l'algorithme historique, ils ont encore des difficultés à s'adapter à des problèmes réels, notamment ceux présentant de nombreuses contraintes. Un autre inconvénient pratique des algorithmes génétiques est le nombre important de paramètres à définir pour aboutir à un algorithme performant. Récemment, de nouvelles approches, comme les *stratégies d'évolution*[15], qui tentent de dépasser les limitations des algorithmes génétiques en introduisant d'avantage de flexibilité dans leur définition, ont obtenu des résultats prometteurs.

Bibliographie

Références générales

[1] E.H.L. Aarts and K. Korst. *Simulated annealing and Boltzmann machines*. John Wiley, 1989.

[2] M. Andreoli, A. Janka, and J.A. DÃl'sidÃl'ri. Free-form deformation parameterization for multilevel 3d shape optimization in aerodynamics. INRIA Research Report 5019, November 2003.

[3] A. Conn, K. Scheinberg, and Ph.L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming*, 79 :397–414, 1997.

[4] A. Conn, K. Scheinberg, and Ph.L. Toint. the convergence of derivative free methods for unconstrained optimization, 1997.

[5] F. Courty, A. Dervieux, B. Koobus, and L. Hascoet. Reverse automatic differentiation for optimum design : from adjoint state assembly to gradient computation. Technical Report 4363, Rapport de Recherche INRIA, January 2002.

[6] J.E. Dennis and V. Torczon. Direct search methods on parallel machines. *SIAM Journal of Optimization*, 1(4) :448–474, 1991.

[7] R. Duvigneau and M. Visonneau. Optimization of a synthetic jet actuator for aerodynamic stall control. *Computers and Fluids*, 35 :624–638, July 2006.

[8] R. Fletcher. *Practical Methods of Optimization*. John Wiley & Sons, 1987.

[9] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, 1981.

[10] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison Wesley Company Inc., 1989.

[11] J. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press, 1975.

[12] A. Jameson, L. Martinelli, and N. A. Pierce. Optimum aerodynamic design using the Navier-Stokes equation. *Theorical and Computational Fluid Dynamics*, 10 :213–237, 1998.

[13] J. Kennedy and R.C. Eberhart. Particle swarm optimization. In *1995 IEEE International Conference on neural networks, Perth, Australia*, 1995.

[14] M. Marazzi and J. Nocedal. Wedge trust region methods for derivative free optimization. *Mathematical Programming*, 91(2) :289–305, 2002.

[15] Z. Michalewics. *Genetic algorithms + data structures = evolutionary programs*. AI series. Springer-Verlag, New York, 1992.

[16] J. A. Nelder and R. Mead. A simplex method for function minimization. *Computer Journal*, 7 :308–313, 1965.

[17] M.J.D. Powell. An efficient method for finding the minimum of a function of several variables without calculating derivatives. *The Computer Journal*, 7, 1964.

[18] M.J.D. Powell. A direct search optimization method that models the objective and constraint functions by linear interpolation. in S. Gomez and J-P. Hennart, editors, Advances in Optimization and Numerical Analysis, pp 51–67, printed in the Netherlands, 1994.

[19] M.J.D. Powell. A direct search optimization method that models the objective by quadratic interpolation. Presentation at the 5th Stockholm Optimization Days, 1994.

[20] H. H. Rosenbrock. An automatic method for finding the greatest or least value of a function. *Computer Journal*, 3 :175–184, 1960.

[21] W. Spendley, G. R. Hext, and F. R. Himsworth. Sequential application of simplex designs in optimisation and evolutionnary operation. *Technometrics*, 4 :441–461, 1962.

[22] N. Srinivas and K. Deb. Multiobjective optimization using non-dominated sorting in genetic algorithm. *Evolutionary Computing*, 3(2) :221–248, 1995.

[23] W. H. Swann. Report on the development of a new direct search method of optimization. Technical Report Research Note 64/3, ICI Central Instrument Laboratory, 1964.

[24] V. Torczon. *Multi-Directional Search : A Direct Search Algorithm for Parallel Machines*. PhD thesis, Houston, TX, USA, 1989.

[25] D. Wienfield. Function minimization by interpolation in a data table. *Journal of the Institute of Mathematic Applications*, 12 :339–347, 1973.

Travaux du projet Opale

[26] J. Cagnol and J.-P. Zolesio. *Control Theory for Partial Differential Equations*. Chapman Hall/CRC, Boca Raton, Florida, 2005.

[27] M.C. Delfour and J.-P. Zolésio. *Shapes and geometries*. Advances in Design and Control. Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, 2001. Analysis, differential calculus, and optimization.

[28] A. Dervieux, N. Marco, C. Held, and B. Koobus. Hierarchical Principles and Preconditioning for Optimum Design and Identification. In J. Périaux *et al*, editor, *Innovative Tools for Scientific Computation in Aeronautical Engineering*, Handbooks on Theory and Engineering Applications of Computational Methods. CIMNE (Barcelona), 2001.

[29] J.-A. Désidéri. *Modèles discrets et schémas itératifs. Application aux algorithmes multigrilles et multidomaines*. Editions Hermès, Paris, 1998. (352 p.).

[30] J.-A. Désidéri. Hierarchical optimum-shape algorithms using embedded Bézier parameterizations. In Yu. Kuznetsov, P. Neittanmäki, and O. Pironneau, editors, *Numerical Methods for Scientific Computing, Variational Problems and Applications*, Barcelona, 2003. CIMNE.

[31] J.-A. Désidéri and A. Janka. Multilevel shape parameterization for aerodynamic optimization – application to drag and noise reduction of transonic/supersonic business jet. In P. Neittaanmäki, T. Rossi, S. Korotov, E. Onate, J. Périaux, and D. Knörzer, editors, *European Congress on Computational Methods in Applied Sciences and Engineering, ECCOMAS 2004*, Jyväkyla, Finland, 24-28 July 2004.

[32] J.-A. Désidéri and J.-P. Zolésio. Inverse shape optimization problems and application to airfoils. *Control and Cybernatics*, 34(1), 2005.

[33] J.-A. Désidéri, J.-P. Zolésio, B. Abou El Majd, and A. Janka. Inverse Shape Optimization Model Problems and Multi-Level Geometrical Optimization Methods. In *International Conference on Control, PDEs and Scientific Computing*, Beijing, PR China, September 13-16 2004. Science Press Beijing-New York. to appear.

[34] A. Habbal. Boundary layer homogenization for periodic oscillating boundaries. *Control and Cybernetics*, 30(3) :279–301, 2001.

[35] A. Habbal. A topology nash game for tumoral anti-angiogenesis. *Journal of Structural Multidisciplinary Optimization*, 30(5) :404–412, November 2005.

[36] Z. Tang, J.-A. Désidéri, and J. Périaux. Multi-Objective Inverse Problem in Aerodynamics using Adjoint Method and Game Theory. In *Conference on Control, PDEs and Scientific Computing*, Beijing, China, September 13-16 2004. Science Press Beijing-New York. to appear.