Titre: Title:	Algorithme intelligent d'optimisation d'un design structurel de grande envergure
Auteur: Author:	Stéphane Dominique
Date:	2009
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Dominique, S. (2009). Algorithme intelligent d'optimisation d'un design structurel de grande envergure [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/159/

UNIVERSITÉ DE MONTRÉAL

ALGORITHME INTELLIGENT D'OPTIMISATION D'UN DESIGN STRUCTUREL DE GRANDE ENVERGURE

STÉPHANE DOMINIQUE
DÉPARTEMENT DE GÉNIE MÉCANIQUE
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIAE DOCTOR (Ph.D.)
(GÉNIE MÉCANIQUE)
OCTOBRE 2009

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

ALGORITHME INTELLIGENT D'OPTIMISATION D'UN DESIGN STRUCTUREL DE GRANDE ENVERGURE

présentée par: DOMINIQUE Stéphane
en vue de l'obtention du diplôme de : Philosophiae Doctor
a été dûment acceptée par le jury d'examen constitué de :

VO Huu Duc, Ph.D., président
TRÉPANIER Jean-Yves, Ph. D., membre et directeur de recherche
VADEAN Aurelian, Doct., membre
FRANÇOIS Vincent, Doct., membre

DÉDICACE

À ma conjointe Anne-Marie, pour son soutient dévoué tout au long de mes études graduées, ainsi qu'à mes parents Claudette et Marc-André, pour m'avoir inculqué les qualités nécessaires à la réalisation de ce travail.

REMERCIEMENTS

L'auteur souhaite remercier Pratt & Whitney Canada pour son soutien financier, et plus particulièrement ses superviseurs François Roy et Ghislain Plante pour leur soutien technique et administratif continu au cours de ces cinq dernières années.

L'auteur souhaite aussi remercier son directeur de travaux, Jean-Yves Trépanier, pour son appui respectueux et son apport considérable au contenu de la présente thèse.

Aussi, il serait injuste de passer sous silence l'appui financier du gouvernement canadien, par le biais du CRSNG, sans qui ce travail n'aurait pas pu voir le jour.

Ensuite, l'auteur aimerait remercier tous ses collègues, à la fois ceux de chez Pratt & Whitney Canada et ceux de l'École Polytechnique de Montréal, d'avoir collaboré à l'agréable atmosphère de travail nécessaire à la complétion de ce travail.

Pour terminer, l'auteur aimerait aussi souligner l'apport de tous les membres extraordinaires de son cercle social (famille et amis) qui ont contribué à lui procurer la motivation nécessaire pour progresser lorsque l'exécution de la tâche devenait plus aride que prévu.

RÉSUMÉ

L'implémentation d'un système automatisé d'aide à la décision en design et d'optimisation structurelle peut donner un avantage significatif à toute industrie œuvrant dans le domaine du design de pièces mécaniques. En effet, en fournissant des idées de solutions au designer ou en améliorant les solutions existantes pendant qu'il n'est pas au travail, ce système lui permet de réduire le temps de conception, ou encore d'explorer davantage de possibilités de design dans un même délai de réalisation.

Cette thèse présente une approche originale permettant l'automatisation d'un processus de design basée sur le raisonnement par cas (RPC), mieux connu sous l'appellation « Case-Based Reasoning » ou CBR.

Cette approche a été développée avec l'optique de nécessiter le moins de ressources possible pour l'entretien et le fonctionnement du système. Elle nécessite toutefois une quantité appréciable de ressources lors de l'implémentation, et convient par conséquent davantage aux problèmes de design de grande envergure pour lesquels on envisage à moyen terme de répondre à plusieurs ensembles de spécifications différentes.

Dans un premier temps, le processus de RPC utilise une banque de données contenant toutes les solutions antérieures connues aux problèmes de design similaires. Ensuite, une sélection de solutions de la banque de données est choisie en comparant les spécifications actuelles du problème avec celles des solutions antérieures. Un réseau de neurones substitut est alors utilisé pour produire une solution en adaptant les solutions antérieures aux spécifications actuelles.

Les solutions émergeant du RPC servent ensuite à générer chacune un îlot de solutions initiales pour un algorithme génétique œuvrant lors de la phase de raffinement du processus.

Grâce au principe du raffinement progressif, l'algorithme débute en utilisant uniquement les variables les plus importantes du problème. Puis, à mesure que l'optimisation progresse, les variables sont introduites graduellement jusqu'à l'utilisation de toutes les variables possibles à la fin de l'optimisation.

L'algorithme génétique utilisé est un algorithme spécifiquement développé dans le cadre de cette thèse pour les problèmes d'optimisation de design de pièces mécaniques. Il se nomme *Algorithme génétique à évolution de noyaux territoriaux* (AGENT), et est essentiellement un algorithme génétique à nombres réels quadrillant le domaine en utilisant une heuristique qui empêche la génération de nouveaux individus dans les zones trop rapprochées des solutions ayant déjà été explorées. Cette interdiction s'amenuise ou s'agrandit graduellement en cours d'optimisation pour permettre à l'algorithme une exploration plus globale ou plus locale lorsque nécessaire.

De plus, un nouvel opérateur de recherche nommé *Opérateur de substitution* a été ajouté, qui incorpore aux opérateurs de recherche réguliers la prédiction d'un réseau de neurones substitut à la fonction coût.

Dans cette thèse, l'approche de RPC suggérée et l'AGENT sont testés sur une série de problèmes tests de nature académique, ainsi que sur le problème du design industriel d'un disque de rotor de turbine à gaz servant à propulser un aéronef.

Ces résultats sont comparés aux résultats obtenus pour les mêmes problèmes par certaines des autres méthodes d'optimisation les plus populaires, notamment (selon le problème étudié) un algorithme à gradient, un algorithme génétique binaire, un algorithme génétique à nombres réels, divers algorithmes génétiques utilisant des opérateurs de croisements à parents multiples, divers algorithmes génétiques à évolution

différentielle, l'algorithme de Hookes & Jeeves, ainsi que l'algorithme POINTER du logiciel I-SIGHT 3.5.

Au regard des comparaisons effectuées, on constate que l'AGENT se classe de façon très compétitive, offrant le meilleur résultat pour 5 des 6 problèmes d'optimisation avec contraintes étudiés et offrant la meilleure performance pour les problèmes issus du générateur de paysages gaussiens à valeur maximum du lot. Pour le problème du disque de turbine à gaz, l'AGENT a procuré un disque plus léger en moyenne de 4.3% que le meilleur algorithme de comparaison (POINTER).

Une restriction de l'AGENT est sa performance moindre dans les problèmes académiques hautement multimodaux d'optimisation non contrainte, pour lesquels il offre une performance plutôt passable pour sa complexité d'implémentation.

En conclusion, au regard des résultats préliminaires obtenus, le processus de RPC décrit dans cette thèse, en combinaison avec l'AGENT, semble être un candidat très solide pour automatiser et accélérer la résolution de problèmes de design structurel de pièces mécaniques, permettant potentiellement de réduire le coût des designs préliminaires industriels.

ABSTRACT

The implementation of an automated decision support system in the field of design and structural optimisation can give a significant advantage to any industry working on mechanical designs. Indeed, by providing solution ideas to a designer or by upgrading existing design solutions while the designer is not at work, the system may reduce the project cycle time, or allow more time to produce a better design.

This thesis presents a new approach to automate a design process based on Case-Based Reasoning (CBR), in combination with a new genetic algorithm named *Genetic Algorithm with Territorial core Evolution* (GATE).

This approach was developed in order to reduce the operating cost of the process. However, as the system implementation cost is quite expensive, the approach is better suited for large scale design problem, and particularly for design problems that the designer plans to solve for many different specification sets.

First, the CBR process uses a databank filled with every known solution to similar design problems. Then, the closest solutions to the current problem in term of specifications are selected. After this, during the adaptation phase, an artificial neural network (ANN) interpolates amongst known solutions to produce an additional solution to the current problem using the current specifications as inputs.

Each solution produced and selected by the CBR is then used to initialize the population of an island of the genetic algorithm. The algorithm will optimise the solution further during the refinement phase.

Using progressive refinement, the algorithm starts using only the most important variables for the problem. Then, as the optimisation progress, the remaining variables are gradually introduced, layer by layer.

The genetic algorithm that is used is a new algorithm specifically created during this thesis to solve optimisation problems from the field of mechanical device structural design. The algorithm is named GATE, and is essentially a real number genetic algorithm that prevents new individuals to be born too close to previously evaluated solutions. The restricted area becomes smaller or larger during the optimisation to allow global or local search when necessary.

Also, a new search operator named *Substitution Operator* is incorporated in GATE. This operator allows an ANN surrogate model to guide the algorithm toward the most promising areas of the design space.

The suggested CBR approach and GATE were tested on several simple test problems, as well as on the industrial problem of designing a gas turbine engine rotor's disc.

These results are compared to other results obtained for the same problems by many other popular optimisation algorithms, such as (depending of the problem) gradient algorithms, binary genetic algorithm, real number genetic algorithm, genetic algorithm using multiple parents crossovers, differential evolution genetic algorithm, Hookes & Jeeves generalized pattern search method and POINTER from the software I-SIGHT 3.5.

Results show that GATE is quite competitive, giving the best results for 5 of the 6 constrained optimisation problem. GATE also provided the best results of all on problem produced by a Maximum Set Gaussian landscape generator. Finally, GATE provided a

disc 4.3% lighter than the best other tested algorithm (POINTER) for the gas turbine engine rotor's disc problem.

One drawback of GATE is a lesser efficiency for highly multimodal unconstrained problems, for which he gave quite poor results with respect to its implementation cost.

To conclude, according to the preliminary results obtained during this thesis, the suggested CBR process, combined with GATE, seems to be a very good candidate to automate and accelerate the structural design of mechanical devices, potentially reducing significantly the cost of industrial preliminary design processes.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES ET ABRÉVIATIONS

AE :	Algorithme évolutionnaire
AG :	Algorithme génétique
AGED :	Algorithme génétique à évolution différentielle
AGED-RL :	Algorithme génétique à évolution différentielle avec recherche locale
AGENT :	Algorithme génétique à évolution de noyaux territoriaux
AGO :	Algorithme génétique ordinaire
AGP2 :	Algorithme génétique proportionnel 2
AGR :	Algorithme génétique à nombres réels
AGR-NT :	Algorithme génétique à nombres réels utilisant les noyaux territoriaux
BEM :	Méthode d'extension des frontières
BES :	Méthode d'extension des frontières avec sélection étendue
BLX :	Opérateur de croisement de type « Blend »
BP :	« Backpropagation » (apprentissage par rétropopagation)
CBR :	« Case-based reasonning »
GPS :	« Generalized pattern search »
HJ :	Algorithme de Hookes & Jeeve
LMBP :	Apprentissage par rétropopagation de Levenberg & Marquardt
MCODE :	Algorithme mimétique à évolution différentielle co-évolutionnaire
MSG :	Générateur de paysages Gaussiens à valeur maximale du lot
NLP :	Programmation non-linéaire
PA :	Pénalité adaptative
PCX :	Opérateur de croisement centré sur les parents
PD :	Pénalité dynamique
PM :	Pénalité de mort
PMC :	Pénalité de mémoire comportementale
PR :	Pénalité de recuit simulé
PS :	Pénalité statique
QP :	Programmation quadratique

RNA :	Réseau de neurones artificiel
RPC :	Raisonnement par cas
SEQP :	Programmation quadratique séquentielle avec contraintes d'égalité
SIQP :	Programmation quadratique séquentielle avec contraintes d'inégalité
SPR :	Supériorité des points réalisables
SPX :	Opérateur de croisement simplexe
SQP :	Programmation quadratique séquentielle
SVM :	« Support vector machine »
UNDX :	Opérateur de croisement à distribution normale unimodale

LISTE DES SYMBOLES

Caractères arabes

$\vec{a}$ Vecteur contenant les valeurs des paramètres du modèle

$a_{max,i}$ Borne supérieure pour le $i^{ème}$ paramètre du modèle paramétrique

$a_{min,i}$ Borne inférieure pour le $i^{ème}$ paramètre du modèle paramétrique

$\vec{C}_i$ $i^{ème}$ enfant produit par un opérateur de recherche évolutionnaire

C_s Constante de raffinement de $S^z_{\min}$ (100 par défaut)

C_v Constante de convergence utilisé par l'heuristique du taux d'application variable des opérateurs de recherche

$\vec{d}$ Vecteur de distance euclidienne pondérée entre les spécifications d'une solution et celles du problème

D^g Distance totale cumulée entre les paramètres d'une population à la génération g

$DP_m(\vec{x})$ Pénalité à la valeur adaptative due aux pénalités de mort pour une contrainte *m*

E_d Extension de l'hyperespace de design

E Module de Young d'un matériau

Eff Efficacité d'un algorithme (ratio entre sa solution et la meilleure solution connue)

E_{max} Nombre maximal d'évaluations accordées à la phase de raffinement

E_{tot} Extension de l'hyperespace de design occupée par les territoires des individus de **Ptot**

Ev_{max} est le nombre total d'évaluations de $F(\vec{x})$ permises pour une optimisation

f L'une des deux constantes de l'opérateur de recherche de l'AGED

$\overline{F}$ Valeur adaptative moyenne d'une population d'individus

$F(\vec{x})$ Fonction coût du problème, aussi nommée valeur adaptative dans le cas des AG

$\tilde{F}(\vec{x})$ Estimation de $F(\vec{x})$ par un modèle substitut

f_c Force de rétention des croyances

F_{max} Valeur adaptative maximale d'une population d'individus

f_{rc} Force de rétention des croyances d'un algorithme culturel

FS_i facteur de sécurité au $i^{ème}$ point de discrétisation pour le problème du disque

g Numéro de la génération actuelle

$G(\vec{x})$ Fonction d'analyse du problème (fonction coût, sans les contraintes)

$g_m(\vec{x})$ Valeur de la transgression de la $m^{ème}$ contrainte

$H(\vec{x})$ Pénalité à la valeur adaptative due aux transgressions de contraintes

$h_m(\vec{x})$ $m^{ème}$ contrainte d'inégalité du problème

I_M Intervalle entre deux migrations d'un AG insulaire

$imax_j$ Numéro i de l'individu de $\vec{P}_{i,j}$ ayant la meilleure valeur adaptative pour le $j^{ème}$ îlot de population

k_{ile} Nombre d'îlots de l'AG et nombre de solutions initiales de l'AGENT

$L_{b,i}$ Borne supérieure pour la spécification i

M Nombre de paramètres de design pour un modèle paramétrique

M_{Flex} Moment de flexion

N Nombre de variables du problème d'optimisation

N_c Nombre de contraintes d'inégalité d'un problème

N_d nombre de points de discrétisation du modèle paramétrique de disque, moins 1

N_M Nombre d'individus émigrant de chaque population d'un algorithme insulaire

N_{max} Nombre d'individus conservés au maximum dans **Ptot** pour l'AGENT

N_p Nombre de spécifications du modèle paramétrique

N_{pop} Nombre d'individu au total dans un îlot d'un algorithme génétique

N_{RN} Nombre d'individus substitués par l'opérateur de substitution

N_s Nombre de solutions au total dans la banque de données du problème

N_t Nombre de variables de sortie du modèle paramétrique

N_{tot} Nombre d'individus compris dans la population totale de l'AGENT

P Matrice population d'un algorithme génétique et population active de l'AGENT

$\vec{p}$ Vecteur de spécifications du problème

p_c L'une des deux constantes de l'opérateur de recherche de l'AGED

$\mathbf{P^g}$ Matrice des $P^g{}_{ij}$

$P^g{}_{ij}$ est $j^{ème}$ paramètre du $i^{ème}$ individu de la population active à la génération g.

$P_{i,j}$ $i^{ème}$ codon du $j^{ème}$ individu d'une population

$\vec{P}_{i,j}$ Vecteur $\vec{x}$ du $i^{ème}$ individu de la population du $j^{ème}$ îlot.

p_{moyen} Pas moyen de mutation d'un gène

p_n Nombre d'optimums générés pour un problème par le MSG

Pr_j Probabilité de reproduction du $j^{ème}$ individu lors de la sélection par roulette

Ptot Population totale de l'AGENT

R_N Nombre maximal d'individus utilisés pour l'entraînement en temps réel du réseau dans l'AGENT. Cet entraînement est effectué chaque R_N évaluations.

$\vec{S}$ Vecteur des sensibilités de la valeur adaptative par rapport à $\vec{x}$

$S_{r,i}$ Importance relative de la i^{ème} variable

$s_{inf,i}$ Borne inférieure pour la variable x_i

$S^0_{\min}$ Seuil minimal initial d'acceptation pour $S_{r,i}$

$S^z_{\min}$ Seuil minimal d'acceptation pour $S_{r,i}$ lors du $z^{ème}$ raffinement

$s_{sup,i}$ Borne supérieure pour la variable x_i

t Épaisseur d'un élément du disque

$\vec{t}$ Vecteur des variables de sortie du modèle paramétrique

T_i Température au $i^{ème}$ point de discrétisation

T_{CV} Temps de convergence de l'algorithme

T_{RN} Nombre de génération utilisant un modèle substitut à la fonction F($\vec{x}$) entre deux générations utilisant la véritable valeur de la fonction

U Nombre total de contraintes du modèle paramétrique

$U_{b,i}$ Borne inférieure pour la spécification i

v Coefficient de pénalité proportionnelle à la distance entre le point et le domaine réalisable

$\vec{w}$ Vecteur des poids accordés aux spécifications du problème

$\vec{x}$ Vecteur des paramètres adimensionnels du modèle paramétrique

$\vec{x}_{ini}$ Vecteur $\vec{x}$ associé à une solution initiale d'un algorithme d'optimisation

$\vec{x}_{ini,j}$ Vecteur $\vec{x}$ associé à la $j^{ème}$ solution initiale d'un algorithme d'optimisation

$\vec{y}_{z,j}$ Paramètres de $\vec{x}$ choisis pour solution j à la phase de raffinement z

z Numéro de la phase de raffinement actuelle

z_{max} Nombre maximal de phases de raffinement requises

Caractères grecs

α Constante de dilatation thermique d'un matériau

α_c Taux d'application de l'opérateur de croisement d'un AG

α_e Nombre d'applications de l'opérateur d'élitisme d'un AG

α_{ec} Taux d'application de l'opérateur de recherche d'un espace de croyance

α_m Taux d'application de l'opérateur de mutation d'un AG

δ Rayon du territoire des individus dans l'AGENT

δ_0 Rayon initial du territoire des individus de l'AGENT

ΔD^g Différence de distance totale cumulée entre deux générations

δ_g Territoire des individus de l'AGENT à la génération g

δ_{min} Constante représentant le rayon territorial minimal permis par l'algorithme

ε_r Allongement radial d'un matériau

ε_θ Allongement tangentiel d'un matériau

γ_c^0 Créativité de l'AGENT

υ Coefficient de poisson d'un matériau

$\Phi_{i.}$ Identifiant du modèle paramétrique associé à l'individu numéro i

ρ Masse volumique d'un matériel

σ_{eff} Contrainte effective

$\sigma_{eff,i}$ Contrainte effective calculée à la $i^{ème}$ station de discrétisation

σ_{Flex} Contrainte de flexion

$\sigma_{Flex,i}$ Contrainte de flexion au $i^{ème}$ point de discrétisation

σ_{max} Contrainte effective maximale admise

$\sigma_{max,i}$ Contrainte effective maximale admise à la $i^{ème}$ station de discrétisation

σ_r Contrainte radiale

$\sigma_{r,i}$ Contrainte radiale au $i^{ème}$ point de discrétisation

σ_θ Contrainte tangentielle

$\sigma_{\theta,i}$ Contrainte tangentielle au $i^{ème}$ point de discrétisation

Θ_1 Borne inférieure de la variable aléatoire de l'opérateur de croisement de l'AGENT

Θ_2 Borne inférieure de la variable aléatoire de l'opérateur de croisement de l'AGENT

ψ Constante retirée de $DP_m(\vec{x})$ pour une contrainte non satisfaite

LISTE DES ANNEXES

Chapitre 1.
INTRODUCTION

Ce projet de recherche est né d'un besoin industriel énoncé par Pratt & Whitney Canada; son département de structures rotatives visait à réduire le temps requis par le procédé de design structurel en phase conceptuelle de ses rotors de turbines à gaz.

En effet, lorsqu'un client ou un autre département demande s'il est possible ou non de construire un moteur ou un rotor ayant certaines spécifications précises, il est impératif de pouvoir répondre ainsi que de savoir donner les dimensions approximatives d'un tel rotor dans les plus brefs délais afin de ne pas perdre un avantage compétitif par rapport aux autres entreprises.

De plus, le design issu de la phase conceptuelle servira de solution initiale pour les étapes plus avancées du design. Il est donc primordial que la solution trouvée soit faisable et de bonne qualité, car un design initial de piètre qualité obligera plusieurs itérations additionnelles très coûteuses par la suite.

Cette thèse présente donc une approche permettant l'automatisation de certaines phases d'un processus de design adapté aux problèmes de design structurel d'envergure industrielle. Malgré le besoin très particulier qui a généré le projet, il faut noter que l'approche développée se veut applicable à tout problème de dimensionnement en ingénierie, voire même à tout problème de design de pièces mécaniques.

Cette approche repose sur deux volets distincts et complémentaires qui ont été développés et constituent les contributions principales de ce travail :

- D'une part, il s'agit d'automatiser au maximum les tâches les plus répétitives effectuées par le designer, de façon à ce qu'il puisse utiliser le temps gagné pour

améliorer son design ou tout simplement réduire son délai de réponse. Le premier objectif de recherche consiste donc à développer et à appliquer un système efficace d'aide à la décision en design de pièces mécaniques.

- D'autre part, il s'agit d'améliorer les outils de conception et de raffinement du design, afin d'aider le designer à obtenir une solution de meilleure qualité découlant d'un effort de conception similaire. Le second objectif de recherche consiste donc à développer et à appliquer des outils d'optimisation de design existants.

Essentiellement, cette thèse est séparée en trois parties. Chacune des deux premières parties (chapitres 2-3 et chapitres 4-5) couvre respectivement le travail développé dans chacun des deux principaux volets de recherche. La troisième partie (chapitres 6-7) présente une série de cas tests académiques et un exemple d'application typique pour valider la procédure décrite dans les deux premières parties. Le reste de cette introduction décrira plus en détail le travail effectué pour chacun des deux volets de recherche.

Volet I : Automatisation du processus de design et systèmes d'aide à la décision

Avec l'état actuel de la science, il serait utopique de penser remplacer le jugement humain par un algorithme de design. Toutefois, grâce à des modèles de dimensionnement intelligents, il n'est pas impensable d'aspirer à produire automatiquement (par interpolation ou extrapolation) un nouveau design décent si l'on dispose déjà de plusieurs expériences de designs similaires ou d'une excellente connaissance de la théorie se rapportant au domaine de design.

L'implémentation de telles méthodes pourrait fournir un avantage concurrentiel en procurant rapidement au designer une solution de départ avec laquelle travailler, ou

encore en explorant le domaine pour une solution nouvelle en dehors de ses heures de travail.

Parmi les méthodes d'automatisation qui sont actuellement utilisées, seuls les processus basés sur le raisonnement par cas (RPC, mieux connu sous le nom de CBR pour « Case-Based Reasoning ») sont capables à la fois d'utiliser le savoir provenant d'expériences de design antérieurs tout en s'adaptant aux différences potentielles de forme et de spécifications des designs ultérieurs. Dans le cas d'un design de pièces mécaniques conçues pour un domaine de pointe tel que le génie aérospatial, cette versatilité est essentielle afin de ne pas avoir à refaire l'implémentation du système chaque fois que la technologie et la forme de la pièce évoluent.

Dans le cadre de ce premier volet de recherche, la théorie du RPC sera donc appliquée. Le RPC est un processus de design utilisant les modèles paramétriques de solutions connues pour produire un nouveau design répondant à de nouvelles spécifications. En général, le RPC consiste en une boucle de trois phases bien distinctes :

- Recherche de solutions similaires
- Adaptation des solutions au problème actuel
- Entreposer la solution pour un usage ultérieur

Il s'agit d'un processus bien connu ayant fait l'objet de nombreuses recherches. La plupart des applications actuelles du RPC fournissent au designer soit les solutions connues les plus rapprochées des conditions actuelles, soit une solution interpolée en utilisant un modèle statistique ou un réseau de neurones artificiels (Zhang, Louvieris & Petrou, 2007), soit une solution potentiellement différente à l'aide d'un algorithme d'optimisation, généralement génétique (Avramenko & Kraslawski ,2008).

L'un des problèmes des processus fournissant uniquement une solution connue ou interpolée, surtout dans le cas du dimensionnement de pièces mécaniques complexes pour lesquelles on dispose d'une maigre banque de solutions, est qu'il est souvent difficile d'effectuer l'interpolation de façon précise, ou même de prévoir efficacement quel sera le meilleur modèle paramétrique à conserver parmi les solutions environnantes.

Les processus de RPC utilisant un algorithme génétique résolvent ce problème en explorant le domaine de design pour découvrir de nouvelles solutions, et acquièrent donc une connaissance suffisante du domaine même si la banque de solutions initiales est faible. Toutefois, puisque la recherche qu'ils effectuent est très aléatoire, ces processus de RPC disposent de la limitation d'être très coûteux en temps de calcul pour les systèmes de grande envergure.

L'objectif spécifique de la recherche effectuée dans ce volet est donc d'appliquer un processus de RPC de façon à tirer avantage des algorithmes d'optimisation, tout en réduisant autant que possible le coût d'utilisation du processus.

L'originalité principale développée dans ce premier volet tient dans l'ajout d'une phase de raffinement automatisée à l'intérieur du processus de RPC.

La phase de raffinement consiste en un algorithme d'optimisation génétique utilisant chacune des solutions environnantes proposées par le RPC comme épicentre d'une population de solutions initiales. Ces solutions centrales sont générées normalement par le RPC en utilisant un réseau de neurones artificiels (RNA) comme interpolant. En bref, l'originalité principale du processus tient du fait qu'il utilisera à la fois les capacités d'interpolation des réseaux de neurones (pour faire usage du savoir à priori) et les capacités de recherche des algorithmes génétiques (pour acquérir du savoir en temps réel).

De cette façon, l'aspect d'exploration aléatoire de l'algorithme génétique s'en trouve réduit au domaine situé dans les environs des solutions suggérées par le RNA et l'algorithme d'optimisation converge plus rapidement.

En principe, une fois le processus implémenté, le designer n'aura plus qu'à raffiner manuellement chaque solution sortante pour déterminer laquelle est la plus valable. Le processus de dimensionnement dans son ensemble, à l'exception de la création de nouveaux modèles paramétriques et de la validation des solutions, s'en trouvera donc automatisé. De plus, puisque les algorithmes génétiques fonctionnent habituellement à partir de solutions initiales aléatoires, le système pourra commencer à fonctionner même s'il dispose d'une banque de données vierge, à condition toutefois que le designer ait au moins inclus une première solution aléatoire avec son modèle paramétrique.

Le chapitre 2 de ce document présente une revue de l'état actuel de la science du design et des systèmes automatisés d'aide à la décision en design.

L'approche développée est décrite plus en détail au chapitre 3 de cette thèse, tandis que le chapitre 6 présentera des cas tests académiques validant certains aspects originaux du processus. L'application du processus entier pour le problème de dimensionnement d'un disque de rotor de turbine à gaz sera décrite dans la troisième partie de la thèse, au chapitre 7.

Volet II : Raffinement du design et optimisation

Lorsque l'on dispose d'un modèle paramétrique pour un design, il suffit alors de modifier la valeur de ces paramètres pour modifier la forme ou la composition de l'objet à produire.

Si l'on dispose en plus d'un modèle analytique pour mesurer la capacité de l'objet à accomplir sa fonction, il ne reste alors plus qu'à changer la valeur des paramètres pour

tenter d'obtenir le meilleur design possible. C'est ce que l'on appelle le raffinement du design, ou encore son optimisation.

À ce stade, le problème de design est très bien défini, et la littérature regorge de possibilités quant à la façon de le résoudre. Ce problème d'optimisation, rarement facile à résoudre, présente d'ailleurs un intérêt majeur pour bon nombre d'ingénieurs œuvrant dans un domaine industriel.

En effet, les problèmes de designs pratiques possèdent en général une très grande quantité de variables, un domaine non convexe ainsi qu'un comportement discontinu et non linéaire, ce qui rend difficile et coûteuse l'application de plusieurs méthodes d'optimisations actuelles.

Plusieurs avenues sont actuellement disponibles pour tenter de trouver des solutions aux problèmes d'optimisation de design structurel ayant un grand nombre de variables. Parmi les avenues explorées, on trouve entre autres les algorithmes d'optimisation à gradient, les algorithmes évolutionnaires (AE) et les algorithmes de type « generalized pattern search » (GPS).

Ces algorithmes d'optimisation disposent toutefois des limitations majeures suivantes :

- Robustesse : les algorithmes à gradient sont difficilement applicables dans le cas de problèmes discontinus, bruités ou multimodaux.
- Rapidité : les AE requièrent un temps de calcul très élevé pour arriver à leur fin pour les problèmes de grande envergure. Ceci est dû au caractère très aléatoire de leurs opérateurs de recherche, qui utilisent peu l'information connue sur la topologie de la fonction pour produire les nouvelles solutions.
- Créativité : les algorithmes à gradient et les GPS ne disposent d'aucunes capacité de recherche aléatoire, et donneront par conséquent toujours le même résultat

face au même problème, ce qui est somme toute contraire au but recherché en design.

Tel que mentionné dans la description du premier volet de recherche, les algorithmes de design, du moins ceux utilisant le RPC, utilisent généralement des AE de type génétique.

L'objectif spécifique de recherche du second volet de cette thèse est donc de développer et de valider un AE génétique capable de s'appuyer sur l'information topologique connue de la fonction coût pour converger plus rapidement que les algorithmes génétiques actuels dans le cas des problèmes de design de pièces mécaniques. Cet algorithme sera alors inclus dans le processus de design issu du premier volet de recherche.

Dans cette optique, le chapitre 4 de cette thèse présente d'abord une revue de l'état actuel de la science en optimisation structurelle.

Par la suite, le chapitre 5 présente un AE original de type génétique dénommé AGENT (Algorithme Génétique avec Évolution de Noyaux Territoriaux) qui a été développé dans l'optique d'effectuer aussi efficacement que possible une recherche globale pour un problème d'optimisation structurelle de grande envergure. L'efficacité de cet algorithme sera comparée à celle d'autres algorithmes d'optimisation populaires pour divers problèmes de nature académique au chapitre 6.

Finalement, en troisième partie de ce document, les deux contributions majeures de cette thèse (soit le processus de RPC et l'AGENT) seront appliquées conjointement au problème de design d'un disque de rotor de turbine à gaz au chapitre 7.

Chapitre 2.
ÉTAT ACTUEL DE LA SCIENCE EN DESIGN DE PIÈCES MÉCANIQUES

Ce chapitre décrit l'état actuel de la science en design, et plus particulièrement en design de pièces mécaniques et en système d'aide à la décision au design.

Premièrement, la section 2.1 donnera une définition plus détaillée du design.

Ensuite, la section 2.2 décrira les principaux processus de design que l'on retrouve dans la littérature concernant le design de produits et de pièces mécaniques.

En troisième lieu, la section 2.3 décrira les trois principaux types de processus d'aide à la décision en design, qui sont en fait des automatisations de certaines des étapes du design.

Finalement, le processus de design suggéré dans cette thèse est un processus utilisant le raisonnement par cas (RPC), qui est un processus d'aide à la décision en design. La section 2.4 décrira les différents processus de RPC que l'on retrouve dans la littérature, ainsi que les avantages et les limitations du RPC.

2.1 Définition du design

En général, les ingénieurs en mécanique entreprennent le design d'une nouvelle pièce pour améliorer les fonctionnalités d'une pièce existante ou pour créer une nouvelle entité disposant de nouvelles fonctions. Une définition adéquate du design de pièces mécaniques a été donnée par Dym & Levitt (1991), pouvant se traduire ainsi :

« Le design est la génération et l'évaluation systématique et intelligente des spécifications d'une entité dont la forme et la fonction permettent d'atteindre les objectifs énoncés et de satisfaire les contraintes données ».

De façon plus abrégée, on peut simplement dire que le design, pour une pièce mécanique, est la recherche d'une nouvelle forme adaptée à sa fonction.

Au tout début d'un processus de design, le designer fait face à un ou plusieurs besoins. Ces besoins sont traduits plus explicitement en un but, qui est séparé en spécifications du design et en objectifs (Roozenburg & Eekels, 1995).

Une spécification est un but qui doit absolument être atteint pour que le design soit valide. Un objectif, au contraire, est un but qui est souhaitable, mais que l'objet final peut accomplir en tout ou en partie sans affecter la faisabilité du design (par exemple, une spécification peut dicter à un rotor une durée de vie de 5000 heures, alors qu'un objectif peut être de réduire le poids ou le coût du même rotor).

Les tâches de design peuvent être classifiées en trois différents types : les tâches routinières, les tâches innovatrices et les tâches créatives (Brown & Chandrasekaran, 1985; Gero, 1990).

Les tâches routinières impliquent généralement de choisir tels quels un objet ou plusieurs composantes de façon à accomplir les buts du design.

Les tâches innovatrices de design, quant à elles, impliquent plutôt d'ajuster l'état ou les dimensions d'un objet pour satisfaire de nouvelles spécifications du client ou du milieu d'opération de l'objet (Braha & Maimon, 1998).

Finalement, les tâches créatives consistent en un design introduisant de nouvelles variables et pour lequel l'espace de design est modifié. L'ensemble des solutions possibles d'un design créatif est donc inconnu (Avramenko & Kraslawski, 2008).

La recherche effectuée dans le cadre de cette thèse s'attaquera plus particulièrement aux tâches de design de type innovatrices, qui constituent la majorité des tâches de design en ingénierie. En effet, comme l'affirme Kolonder (1993), il est très rare que les designers démarrent un projet sans partir d'une solution ou d'une expérience de design antérieure. D'ailleurs, Lansdown (1987) va même plus loin en affirmant que l'innovation en design est généralement le résultat de modifications incrémentales d'idées existantes plutôt que de nouvelles idées.

2.2 Modèles généraux de design

Comme l'affirment Avramenko & Kraslawski (2008), plusieurs articles discutent du design de façon générale, et la plupart déclarent que le processus de design est un processus par étapes, itératif et évolutionnaire. Cette section discute de plusieurs modèles généraux s'appliquant à toutes les sous-disciplines du design.

Modèle de Cross

L'un des modèles les plus simples de design que l'on puisse retrouver est le modèle de Cross (2000), qui est constitué uniquement de 4 étapes (voir figure 2.1).

D'abord, durant la phase d'exploration, les objectifs et les spécifications du design sont définis. Ensuite, une ébauche de design est créée dans la phase de génération, afin que son respect des spécifications et objectifs soit évalué dans la phase d'évaluation. Après une ou plusieurs itérations, lorsque l'évaluation est positive, la phase de communication implique la constitution de la documentation et la mise en œuvre des moyens de production du concept.

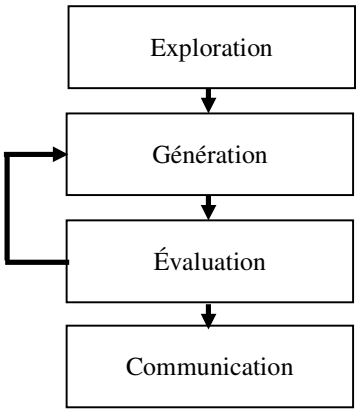


Figure 2.1 : Modèle général de design de Cross

Afin de clarifier davantage le processus de design, plusieurs chercheurs ont proposé des schémas plus détaillés que celui de Cross. Les figures 2.2 et 2.3 illustrent certains des schémas les plus utilisés dans la littérature.

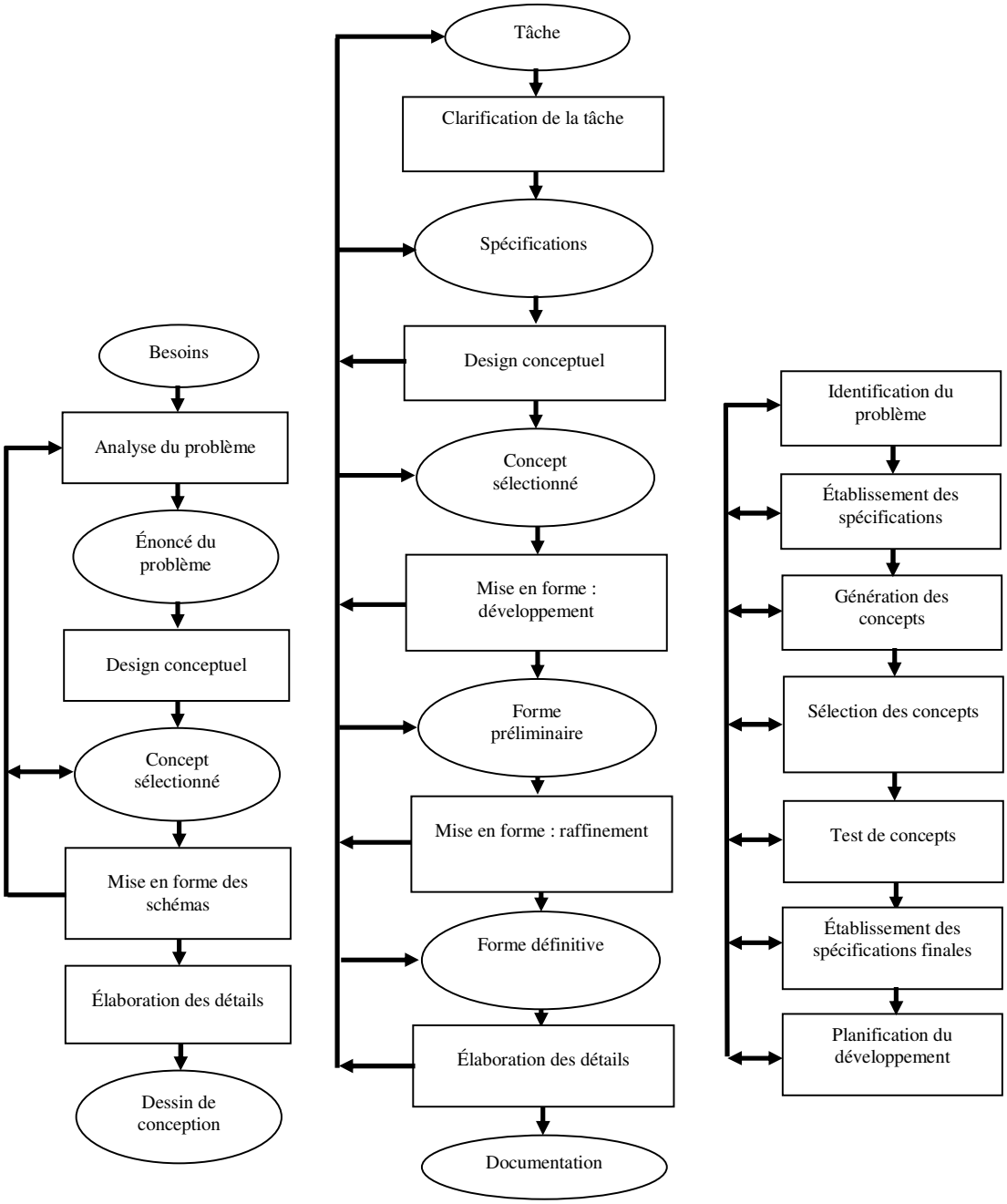
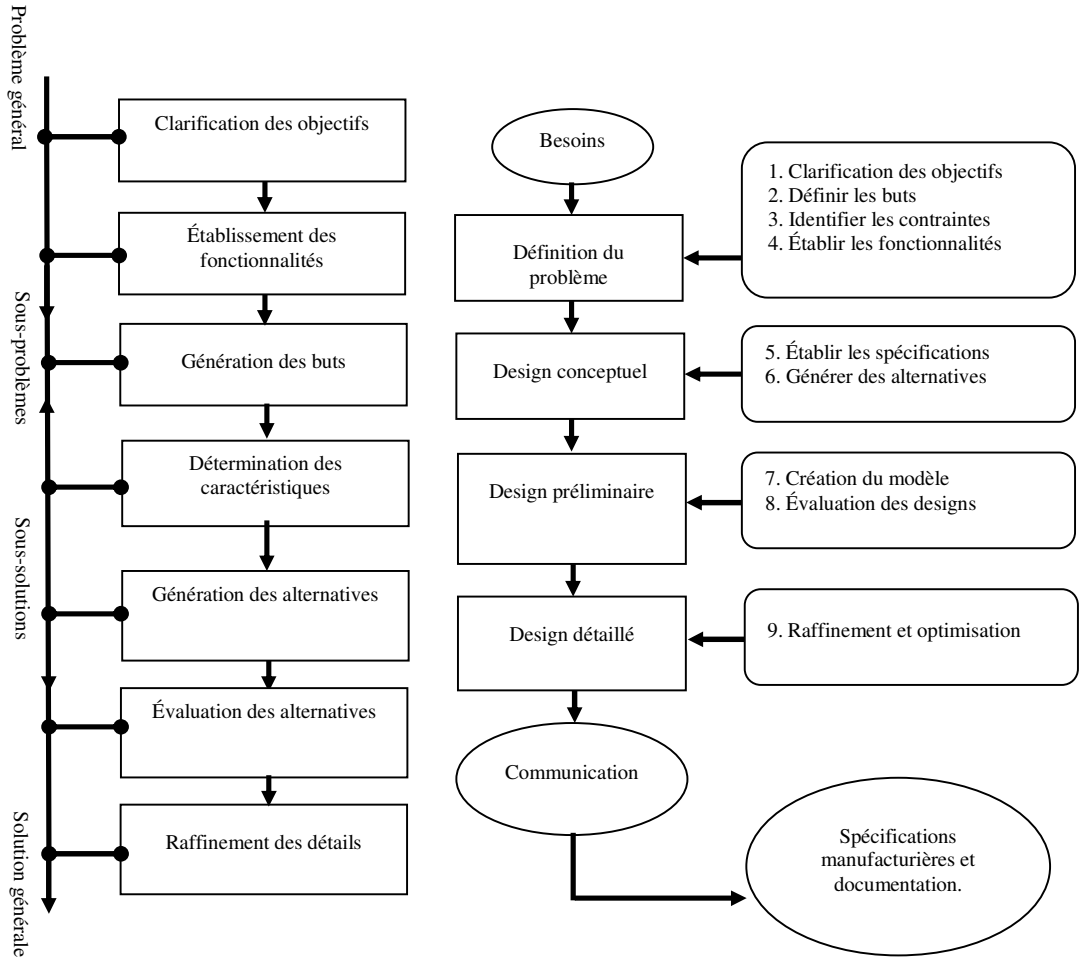


Figure 2.2 : Modèles génériques de design



D. Modèle modifié de Cross (2000) **E. Modèle de Dym & Little (2004)**

Figure 2.3 : Modèles génériques de design (suite)

Modèles plus détaillés

L'une des différences majeures que l'on peut retrouver dans les modèles plus détaillés de la littérature est la présence d'une tâche de design destinée à élaborer les buts, objectifs et spécifications du problème.

Dans le modèle de French (figure 2.2 A), cette phase se nomme « Analyse du problème ». Dans le modèle de Pahl & Beitz (figure 2.2 B), elle est appelée « Clarification de la tâche ». Le modèle de Ulrich & Eppinger (figure 2.2 C) la sépare en 2 tâches distinctes, soit l'identification du problème (pour le but et les objectifs) et l'établissement des spécifications (pour les spécifications du produit). Le modèle modifié de Cross (figure 2.3 D) sépare cette tâche en 4, soit les tâches de « Clarification des objectifs », « Établissement des fonctionnalités », « Génération des buts » et « Détermination des caractéristiques ». Finalement, le modèle de Dym & Little (figure 2.3 E) sépare cette tâche approximativement de la même façon que le modèle modifié de Cross; c'est-à-dire en 4 sous-tâches distinctes impliquant de clarifier les objectifs, définir le but, définir les contraintes et établir les fonctionnalités de l'objet.

Similarités entre les modèles

Comme le modèle modifié de Cross (figure 2.3 D) le suggère explicitement, la tâche complète de design consiste à mettre en œuvre des idées permettant de passer graduellement d'un problème général et abstrait à une solution finale très spécifique et détaillée.

Quel que soit le modèle de design utilisé, on peut donc distinguer huit tâches à accomplir pour un design de pièces mécaniques.

1 – Analyser le problème

La première tâche à accomplir consiste à identifier les besoins du client et à clarifier la nature exacte du problème.

2 – Définir les fonctionnalités

Une fois le problème bien défini, il est possible de déterminer les fonctionnalités requises pour qu'une entité réponde bien aux besoins du design actuel. Par exemple, une

fonctionnalité peut être de permettre à un passager de se déplacer d'un point A à un point B.

3 – Définir les objectifs

À partir des fonctionnalités requises, le designer doit définir le but poursuivi par le design. Chacun des énoncés spécifiques (quantitatif ou qualitatif) d'un but constitue un objectif. Un objectif peut être par exemple de produire un objet de poids minimal, ou encore de lui procurer un aspect esthétique.

4 – Définir les spécifications

Les spécifications sont des critères précis que l'entité doit absolument rencontrer pour que le design atteigne son but ou ses objectifs. Par exemple, la vie utile de l'objet peut être fixée à un nombre d'heures minimales voulues par le client.

5 – Design de concepts

Cette étape consiste à générer ou proposer des modèles de solution répondant aux spécifications données à l'étape 4 tout en tendant vers la direction donnée par les objectifs de l'étape 3.

6 – Évaluer les concepts

Lors de cette étape, les divers modèles de solution conçus à l'étape 5 sont évalués et comparés afin de déterminer s'ils satisfont suffisamment les objectifs de design et afin de choisir si on doit revenir au design de concepts ou si l'un des concepts peut convenir comme concept final de solution.

7 – Raffinement et détails

Cette étape consiste à élaborer davantage le concept choisi précédemment, à l'optimiser autant que possible et à régler tous les petits détails nécessaires à sa mise en œuvre.

8 – Communication

La dernière étape du design consiste à établir la procédure exacte de manufacture et à documenter l'entité produite de façon à permettre la suite du développement de la pièce.

Essentiellement, en observant les modèles suggérés pour le design dans la littérature, il est possible de constater que la plupart identifient globalement ces mêmes huit tâches à accomplir lors d'un processus de design, avec une nomenclature variant sensiblement d'un cas à l'autre.

Afin de mieux démontrer les similarités entre les diverses méthodologies de design, la table 2.1 illustre en détail les différentes tâches requises lors d'un processus de design, ainsi que l'étape à laquelle ces tâches sont effectuées dans chacun des modèles génériques des figures 2.2 et 2.3.

Table 2.1. Étape à laquelle chacune des 9 tâches fondamentales du design est effectuée pour divers modèles de processus de design.

TÂCHE DE DESIGN	Cross	French	Pahl & Beitz
Analyser le problème	Exploration	Analyse du problème	Clarification de la tâche
Définir les fonctionnalités	Exploration	Analyse du problème	Clarification de la tâche
Définir les objectifs	Exploration	Analyse du problème	Clarification de la tâche
Définir les spécifications	Exploration	Analyse du problème	Clarification de la tâche
Design de concepts	Génération	Design conceptuel	Design conceptuel
Évaluer les concepts	Évaluations	Mise en forme des schémas	Mise en forme : développement
Raffinement et détails	Génération	Élaboration des détails	Élaboration des détails
Communication	Communication	Élaboration des détails	Élaboration des détails

TÂCHE DE DESIGN	Ulrich & Eppinger	Cross modifié	Dym & Little
Analyser le problème	Identification du problème	Clarification des objectifs	Définition du problème
Définir les fonctionnalités	Identification du problème	Établissement des fonctionnalités	Définition du problème
Définir les objectifs	Établissement des spécifications	Génération des buts	Définition du problème
Définir les spécifications	Établissement des spécifications	Détermination des caractéristiques	Design conceptuel
Design de concepts	Génération de concepts	Génération des alternatives	Design conceptuel
Évaluer les concepts	Sélection de concepts et Tests de concepts	Évaluation des alternatives	Design préliminaire
Raffinement et détails	Établissement des spécifications finales	Raffinement des détails	Design détaillé
Communication	Planification du développement	Raffinement des détails	Communication

2.3 Systèmes d'aide à la décision en design

Avec l'état actuel de la technologie, il est plutôt impensable de remplacer le jugement humain en automatisant entièrement un processus de design. Toutefois, dans le but d'accélérer le processus, de réduire les risques d'erreur et de simplifier la tâche à accomplir par le designer, plusieurs chercheurs se sont efforcés de confectionner des systèmes permettant d'automatiser certaines des tâches les plus répétitives du design, ou encore de suggérer au designer des solutions avec lesquelles il peut amorcer son travail. Ces systèmes sont nommés des systèmes d'aide à la décision en design.

Avramenko & Kraslawski (2008) suggèrent de classifier en trois catégories distinctes les systèmes d'aide à la décision les plus utilisés actuellement pour le design : les systèmes d'aide à la décision algorithmiques, les systèmes d'aide à la décision basés sur le raisonnement par induction du savoir (aussi appelés systèmes experts), et les systèmes d'aide à la décision basés sur le raisonnement par cas.

Les sections qui suivent décrivent plus en détail chacune de ces catégories de système d'aide au design.

2.3.1 Système algorithmique

Un système algorithmique est une procédure très spécifique à un domaine qui apporte une solution satisfaisante à un problème de design en un nombre fini d'itérations. Les prémisses principales de ce genre de système sont que les spécifications initiales du design soient constantes et très bien définies, et que le domaine de design soit suffisamment bien connu pour que des critères précis existent pour évaluer la qualité de tous les designs possibles.

Les méthodes généralement employées par les systèmes algorithmiques sont la recherche exhaustive, la recherche rapide, et la programmation mathématique (Avramenko & Kraslawski, 2008).

Les stratégies de recherches exhaustives regroupent entre autres les méthodes avares (mieux connues sous le nom « greedy »), la programmation dynamique et la séparation et évaluation (« branch and bound »). Ces méthodes génèrent une énorme quantité de concepts à évaluer pour les problèmes complexes. Leur application est par conséquent limitée aux problèmes les plus simples. Le lecteur intéressé pourra consulter Siddal (1982), Dasgupta (1989) et Chandrasekaran (1990) pour plus de détail sur le sujet.

L'alternative aux méthodes de recherches exhaustives est la recherche rapide, qui utilise des règles arbitraires pour limiter l'espace de recherche de ces méthodes. Le désavantage inhérent aux méthodes de recherche rapide est que la solution recherchée au problème peut fort possiblement se situer hors de l'espace de recherche.

Finalement, la programmation mathématique implique de développer un modèle mathématique fixe ayant une fonction objectif à optimiser et un ensemble de contraintes qui représente les limitations de ressources (Siddall, 1982 ; Braha & Maimon, 1998 ; Gani, 2004).

Les méthodes algorithmiques disposent généralement de l'avantage de procurer rapidement une solution acceptable au problème de design, mais ont comme inconvénient une absence totale de flexibilité et donc un coût considérable d'entretien pour le système.

2.3.2 Systèmes experts

Les systèmes experts sont des systèmes d'aide à la décision qui utilisent des règles établies par des experts humains ou bien des algorithmes d'acquisition et de traitement de données pour produire un résultat adéquat pour le problème.

Un système expert dispose généralement de la structure affichée à la figure 2.4 (Avramenko & Kraslawski, 2008).

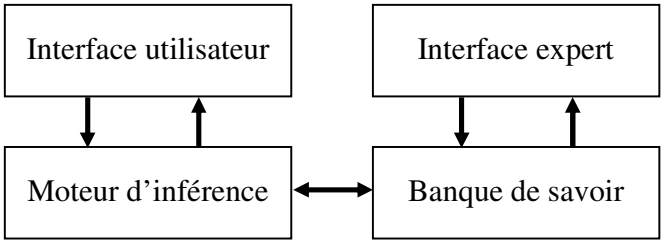


Figure 2.4 : Système expert

Ainsi, l'utilisateur envoie des requêtes au moteur d'inférence, qui détermine les résultats appropriés à retourner selon les règles de design appropriées. Le moteur d'inférence est donc responsable d'élaborer des règles de décision selon les données contenues dans la banque de savoir.

Les hypothèses formant ces règles de design peuvent être vérifiées ou modifiées à même la banque de savoir par les experts, permettant d'inculquer au procédé un raisonnement imitant celui d'un expert humain du domaine.

Un lecteur intéressé par les systèmes experts peut se référer à Tong & Sriram (1992), Nakayama & Tanaka (1999) et Li & Kraslawski (2004), pour plus d'information sur le sujet.

Système expert versus système algorithmique

L'un des avantages du système expert par rapport au système algorithmique est de ne pas nécessiter de modélisation exacte du domaine du problème. Le système expert est donc généralement moins coûteux d'implémentation. En effet, puisque le système expert est fondé majoritairement sur des méthodes empiriques et des heuristiques, il est possible pour un expert de concevoir le système sans avoir à se soucier de générer une simulation mathématique complète de l'opération de la pièce mécanique en question.

De plus, l'introduction d'heuristiques spécifiques au domaine bien précis de design permet généralement d'alléger considérablement la quantité de calculs informatiques requis pour arriver à une solution, réduisant ainsi aussi le coût d'opération du système.

Par contre, cette façon de procéder implique aussi que, contrairement à l'approche algorithmique, le système ne dispose d'aucune garantie quant à sa capacité de générer une solution adéquate au problème. Qui plus est, cette approche limite aussi l'exploration du domaine, et donc la possibilité de découvrir de nouveaux designs valables que l'expert n'aurait pas envisagés à prime abord.

2.3.3 Raisonnement par cas

Le dernier des trois grands types de systèmes d'aide à la décision est le système de raisonnement par cas (RPC).

Le raisonnement par cas (RPC, ou aussi appelé CBR pour Case-Based-Reasonning) est un processus général qui s'applique bien au design automatisé d'objets industriels (Maher, Balachandran & Zhang, 1995 ; Mendes, Mosley & Watson, 2002). Les fondements du RPC ont été créés au début des années 80 par Schank (1982) et son groupe de recherche. Depuis, de nombreuses applications du RPC ont été instaurées, notamment dans les domaines du diagnostic médical, du design et de la prédiction de

fissures. Bien que l'historique de RPC dépasse le cadre de ce document, le lecteur intéressé est invité à consulter Aamodt & Plaza (1994) pour plus de détails sur le sujet.

Le RPC est un processus s'inspirant de l'apprentissage humain, à savoir que chaque fois qu'un nouveau problème est rencontré, la solution antérieure à un problème similaire peut s'avérer un bon point de départ pour ce nouveau problème. Un processus de design fondé sur le RPC suit habituellement la procédure de la figure 2.5 (Hunt, 1995).

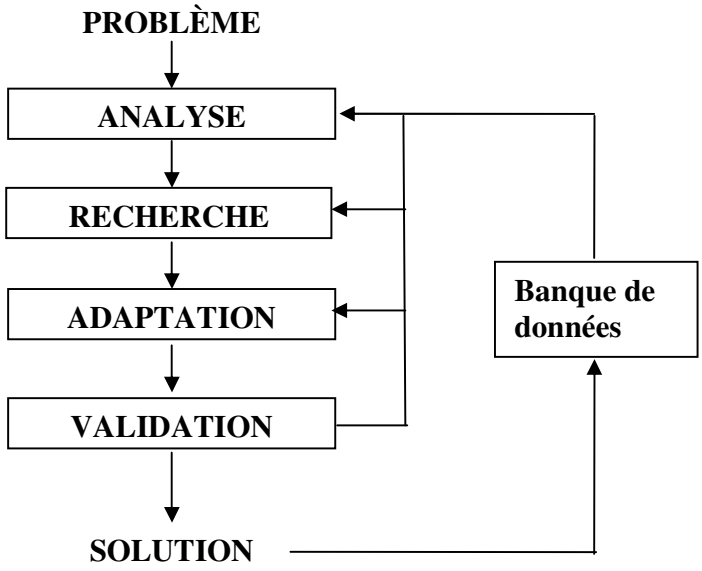


Figure 2.5 : Design utilisant le raisonnement par cas

Le processus de design proprement dit s'amorce donc par la *phase d'analyse* du problème. Le designer établit alors une liste des buts et des spécifications précises, comme dans le cas d'un processus de design régulier.

Ceci accompli, le designer vérifie dans ses connaissances et dans les bases de données auxquelles il a accès afin de trouver une liste de designs antérieurs capables de s'acquitter de fonctionnalités similaires à celles qu'il a énoncées lors de son analyse. Il s'agit de la phase *recherche*, qui consiste donc à former une banque d'au moins une

solution possible pour chaque fonctionnalité requise. La phase de recherche permet aussi d'obtenir la procédure de design ayant été accomplie pour les designs antérieurs, de façon à réduire le temps du processus de design actuel (par exemple, on peut disposer déjà d'un modèle paramétrique capable de satisfaire nos besoins).

Ensuite, le designer s'interroge sur sa capacité à mettre en œuvre les solutions de sa recherche pour répondre aux fonctionnalités dictées par son analyse. Il s'agit de la phase d'*adaptation*. En général, pour le dimensionnement d'une pièce mécanique, cette étape signifie la modification des paramètres de la pièce, ainsi que la mise en place d'une méthodologie pour mesurer l'efficacité d'un design et sa faisabilité selon les spécifications du problème. Après la phase d'adaptation, le système dispose donc d'une solution adaptée à son problème.

Finalement, lorsque l'adaptation est terminée, la phase *validation* consiste à vérifier la validité du modèle de solution obtenu, que ce soit grâce à une expertise théorique ou encore par la mise en œuvre d'un prototype du système. Si la solution n'est pas valide, le procédé peut être relancé de nouveau, ou alors l'usager peut réparer manuellement la solution et terminer la phase de validation. La solution sortant de ce système peut alors être mise en œuvre et ajoutée à la banque de données du problème pour servir lors d'un futur problème similaire.

Le chapitre 3 décrira plus en détail chacune des phases du processus de design utilisant le RPC utilisé dans cette thèse.

RPC versus système algorithmique

Par rapport aux systèmes de RPC, les systèmes algorithmiques disposent de l'inconvénient d'être extrêmement rigides. En effet, si les spécifications ou les paramètres de la pièce mécanique changent d'un design à un autre, le designer utilisant un processus de RPC n'aura qu'à insérer un nouveau modèle paramétrique de solution dans le système et le design pourra se poursuivre normalement. Avec un système

algorithmique, le système en entier devra être modifié pour tenir compte de ces ajustements.

Ensuite, les systèmes de type RPC disposent de l'avantage d'acquérir de la connaissance à mesure que surviennent les nouveaux problèmes de design, allégeant chacune des tâches subséquentes. Un système algorithmique, au contraire, devra recommencer chacune des nouvelles tâches de design à neuf.

D'un autre côté, les atouts du système de RPC se fondent uniquement sur la connaissance provenant de solutions antérieures à des problèmes similaires. Ainsi, dans le cas où l'on fait face à un problème de design complètement nouveau, ou pire encore, à un type de problèmes que l'utilisateur n'envisage pas de résoudre à nouveau, le RPC ne dispose pas vraiment d'avantages tangibles sur la méthode algorithmique, et sera plus coûteux d'implémentation.

Le système d'aide à la décision de type RPC est donc plus approprié pour les cas où l'on dispose déjà de solutions à des problèmes de design similaires, ou pour les cas où l'on envisage à priori la possibilité de résoudre à nouveau le même type de problèmes de design.

RPC versus système expert

Les systèmes experts disposent de la même limitation que le RPC, à savoir qu'il est nécessaire pour le système de posséder de l'information sur certains design antérieurs afin de pouvoir profiter pleinement du système.

Toutefois, chaque fois qu'une variation importante survient sur le problème de design, le RPC s'ajustera automatiquement, alors que les règles définissant le système expert devront être revues entièrement.

De plus, puisque le système de RPC est conçu pour apprendre automatiquement les nouvelles informations, sa maintenance est très simple par rapport à celle d'un système expert.

Aussi, la structure du RPC est plus favorable aux problèmes complexes et à la gestion d'un nombre élevé de variables et de solutions, cas pour lesquels le nombre élevé d'interactions entre les composantes fait que le système expert risque de produire des règles de design incomplètes ou erronées

D'un autre côté, les systèmes experts disposent d'une structure qui leur permet de se spécialiser et d'être extrêmement efficaces pour un type de problème bien précis, et sont donc davantage indiqués que le RPC si l'on s'attend à ce que les problèmes de design auxquels l'utilisateur se trouvera confronté varient peu avec le temps.

2.4. Choix d'un algorithme d'aide à la décision en design de pièces mécaniques

Suite à la discussion précédente, il est possible de suggérer quelques guides pour le choix d'un type d'algorithme d'aide à la décision pour un problème de design de pièces mécaniques.

D'abord, on constate que le nombre de problèmes de design similaires mais différents que l'on s'attend à devoir résoudre influence beaucoup notre décision. En effet, un système algorithmique sera incapable de s'adapter à plusieurs problèmes de design comportant des différences, alors qu'un système expert s'y adaptera difficilement et qu'un système de type RPC s'y adaptera aisément.

Ensuite, la connaissance à priori du domaine est aussi un critère de choix très important. Si une très bonne connaissance à priori du domaine est disponible, le système expert performera de façon exemplaire, de même que le RPC si la connaissance inclus les

paramètres de diverses solutions de design antérieures. Par contre, si aucune connaissance n'est disponible, le système algorithmique sera un meilleur candidat. La figure 2.6 résume cette discussion.

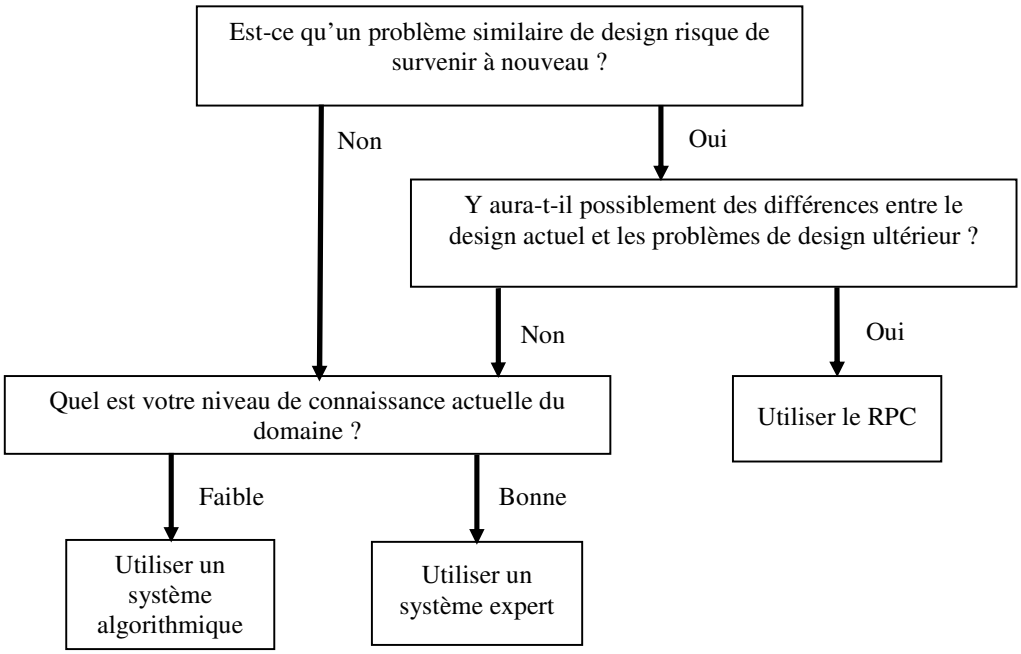


Figure 2.6 : Choix d'un système d'aide à la décision pour un problème de design

2.5. Champs d'application du RPC

Le système d'aide à la décision étudié dans cette thèse est un système de type RPC. En effet, le problème de design d'un rotor de turbine à gaz auquel s'est intéressé l'auteur est un problème de design récurrent ; à chaque fois qu'un nouveau moteur est dimensionné, le design d'un nouveau modèle de rotor doit être effectué. De plus, le design de pièces mécaniques destinées à l'industrie aéronautique est un secteur de pointe en constant changement, de telle sorte qu'il est peu probable qu'un rotor produit plusieurs années auparavant dispose de la même exacte paramétrisation qu'un rotor fraichement sorti du

processus de design. Dans un tel cas, suivant la discussion de la section 2.4, un processus de type RPC semble tout à fait convenir à la situation.

Le chapitre suivant explique donc en détail le processus de design qui a été implémenté dans le cadre de cette recherche. Notez que, malgré que le problème du rotor de turbine à gaz soit le problème ayant généré la recherche contenue dans cette thèse, le champ d'application de la méthode décrite au chapitre 3 s'applique à tout problème de design récurrent, et en particulier à ceux dont les paramètres et les spécifications de design sont amenés à varier au moins légèrement d'un problème à l'autre. Ceci inclut donc généralement toutes les pièces mécaniques pour lesquelles on recherche une performance croissante d'année en année (comme les pièces d'automobiles), ou encore celles que l'on envisage utiliser dans différents environnements ou adapter à différentes autres composantes.

Chapitre 3.

LE PROCESSUS DE DESIGN

Ce chapitre décrit le processus de design proposé pour atteindre le premier objectif de recherche global de cette thèse, qui est de développer et d'appliquer un système efficace d'aide à la décision en design de pièces mécaniques.

Suite au raisonnement décrit au chapitre 2, la procédure de design qui sera utilisée est un processus de RPC. L'élément d'originalité du processus décrit ici est que le processus disposera à la fois d'une adaptation neurale et d'un raffinement lors duquel un algorithme génétique est utilisé afin d'optimiser le résultat de la phase d'adaptation. Le fondement de la procédure proposée a été inspiré par Liu (1996) et par Jonhson & Sushil (2005), qui ont utilisé le résultat du RPC pour ensemencer la population initiale d'un algorithme génétique ordinaire et par Zhang, Louvieris & Petrou (2007), qui a cité divers chercheurs suggérant l'usage d'un réseau de neurones artificiels pour adapter les solutions aux spécifications actuelles.

Le RPC est un processus de design que l'on pourrait qualifier de paresseux, à savoir qu'aucun paramètre n'ayant pas été pensé à prime abord par le concepteur des modèles de solutions choisis par le système ne pourra être implémenté dans le processus. Par exemple, si un modèle paramétrique utilise des arcs de cercle pour définir une géométrie, le RPC ne pourra pas automatiquement utiliser de splines pour définir cette même géométrie, du moins pas sans la création manuelle d'un nouveau modèle par le designer.

Malgré cet inconvénient, le RPC trouve tout de même maintes applications pratiques en ingénierie. En effet, lorsqu'une industrie résout successivement plusieurs problèmes de design similaires (comme c'est le cas dans l'exemple d'un rotor de turbine à gaz), le problème de design revient souvent à résoudre uniquement un problème de choix de

modèle et de redimensionnement. Une entreprise voulant adapter ses pièces aux spécifications des clients aura donc tout intérêt à utiliser un système de design de ce type pour éviter aux designers de refaire chaque fois le même travail répétitif. Dans un tel cas, le temps récupéré par un tel processus pourra servir à des tâches moins évidentes à automatiser, comme la création de nouveaux modèles paramétriques et la validation des solutions.

La figure 3.1 illustre les bénéfices de l'application proposée dans ce document en comparant les tâches devant être effectuées par le designer et par le système informatique lors de son implémentation. On constate que les spécifications de la phase d'analyse sont généralement fournies par le client (ou alors ce sont des bornes fournies par les autres départements qui travaillent sur le même design). On observe aussi que le processus proposé peut automatiquement sélectionner des solutions initiales, les adapter aux spécifications, choisir les paramètres d'optimisation et enfin optimiser la solution. Il ne reste alors plus pour le designer que les tâches les plus cruciales du design, soit la création des nouveaux modèles de pièces et la validation des résultats du RPC pour la solution provenant des modèles actuels.

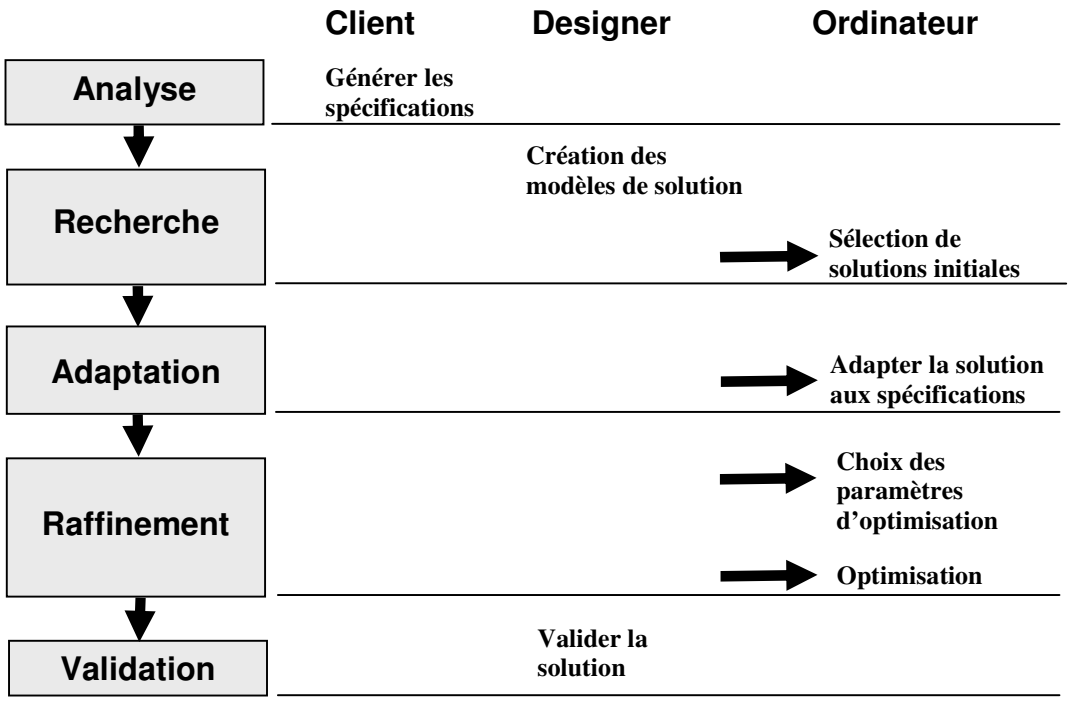


Figure 3.1 : Automatisation des tâches de design à l'aide d'un système informatique de type RPC.

Le reste de cette section présente donc plus en détail chacune des phases du processus.

3.1 Analyse

La phase d'analyse, qui débute le processus de RPC, est une étape importante qui permet de classifier ou quantifier les fonctionnalités du problème actuel de façon à mieux pouvoir reconnaître les solutions connues qui se rapprochent le plus de la solution actuellement recherchée.

3.1.1 Choix des spécifications

Pendant cette phase, il importe de choisir les spécifications du problème. Les spécifications doivent représenter toutes les fonctionnalités majeures requises pour l'objet, et sont des valeurs en général fournies par le client.

Lors de l'implémentation du système de design de type RPC, il est important de tenter de penser d'emblée à toutes les spécifications possiblement fournies par les clients pour le type de pièces souhaité.

Lors d'un design particulier, chacune de ces spécifications devra avoir une valeur bien définie, qui peut être une valeur numérique entière, discrète ou même une chaîne de caractère.

Le vecteur de spécifications sera nommé $\vec{p}$ et sa longueur N_p.

3.1.2 Choix des poids pour chaque spécification

Pour un design particulier, chaque spécification p_i devra être accompagnée d'un poids w_i, qui est une valeur numérique entre 0 et 1 indiquant l'importance de cette spécification aux yeux du client. Si le client ne précise rien pour p_i, $w_i = 0$. Autrement, le client ou le designer accordera la valeur $w_i = 1$ à la (les) spécification(s) la(les) plus essentielle(s), et pondèrera les autres entre 0 et 1 selon leur importance relative.

Ce choix influencera par la suite la sélection des solutions initiales au problème. De plus, si l'utilisateur le désire, un modèle paramétrique pourrait utiliser $\vec{w}$ pour pondérer la fonction coût lors de l'optimisation.

3.2 Phase recherche

Lors de cette phase, le système explore la banque de données contenant toutes les solutions connues, et sélectionne les solutions ayant les spécifications les plus rapprochées de celles du problème actuel.

3.2.1 Création de nouveaux modèles paramétriques

Si le designer désire apporter des améliorations à l'un ou l'autre des modèles, ou alors tout simplement tester l'application d'un nouveau modèle paramétrique, il doit le faire au tout début de cette phase, avant de lancer le reste du processus. Bien sûr, en pratique, puisque le but du RPC est souvent d'obtenir rapidement une estimation initiale, il n'est pas rare que le processus soit amorcé tel quel pour vérifier la qualité de la solution que l'on peut obtenir avec les modèles actuels avant de le redémarrer à neuf en incluant les éventuels nouveaux modèles.

Afin d'éviter de minimiser les éventuelles failles dans le modèle et de faciliter l'optimisation, il est important d'effectuer la paramétrisation du problème aussi rigoureusement que possible. Selon l'expérience acquise lors des essais effectués par l'auteur au cours de ses travaux, une bonne façon de créer un modèle pour le processus de RPC est de rendre adimensionnels et de relativiser autant de paramètres que possible. Par exemple, les figures 3.2a et 3.2b illustrent deux paramétrisations pour un quadrilatère ACDB.

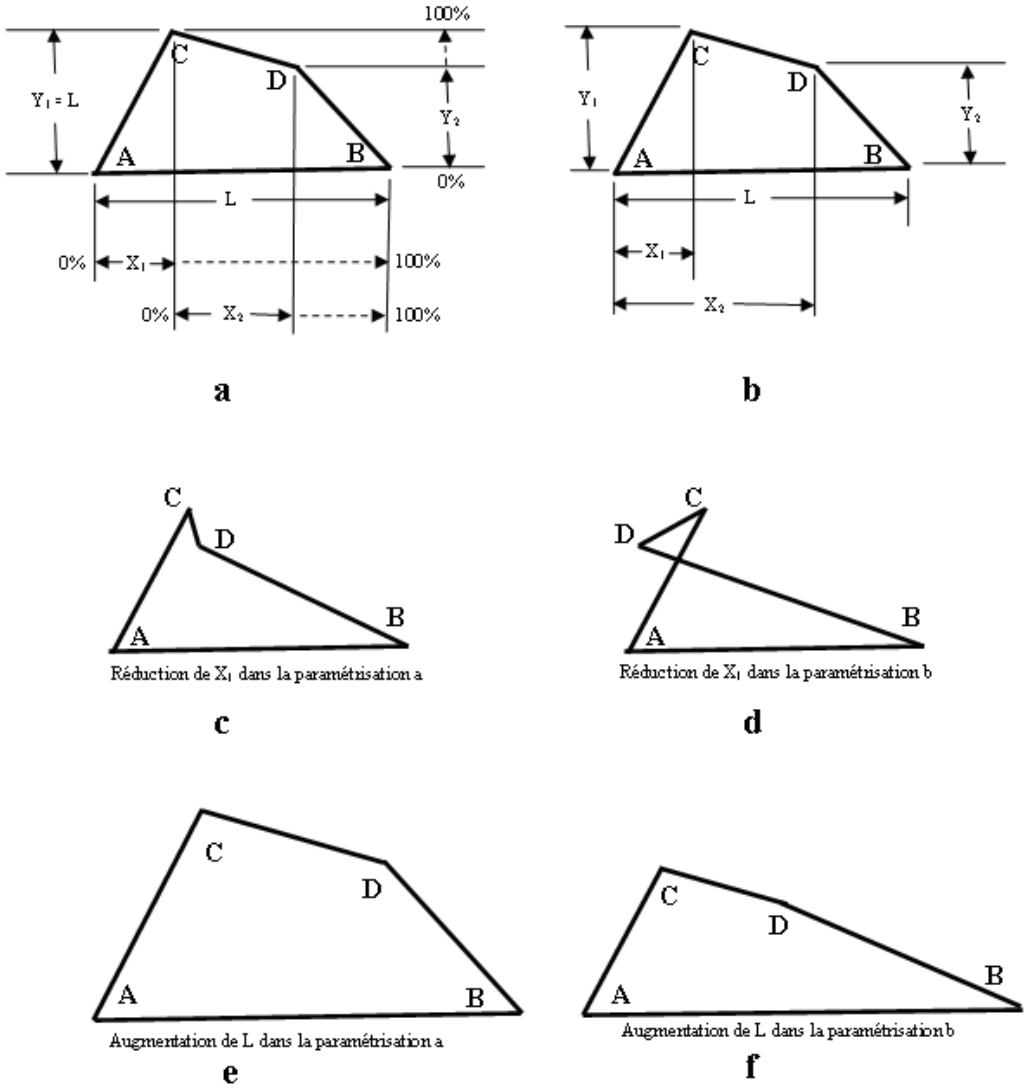


Figure 3.2 : Comparaison entre une paramétrisation relative a, et absolue b.

Dans la figure 3.2a, on observe que la position horizontale des points C et D est définie par X_1 et X_2 comme étant respectivement un pourcentage de L et de la composante horizontale de la droite CB. La position verticale de C est définie par Y_1 qui est une proportion de L, et la position verticale de D est un pourcentage Y_2 de la composante verticale du segment AC.

Dans la figure 3.2b, la position des points C et D est plutôt simplement définie par les coordonnées (X_1, Y_1) et (X_2, Y_2) centrées à l'origine A.

En principe, les deux paramétrisations offrent la même flexibilité de design. Toutefois, si un algorithme modifie aléatoirement la valeur de certains de ces paramètres, on constate que la paramétrisation 3.2a, qui ne comporte qu'une seule variable absolue *L*, est plus robuste que la paramétrisation 3.2b. Par exemple, supposons qu'un optimiseur modifie X_2 grandement à la baisse. Les figures 3.2c et 3.2d seraient respectivement obtenues, selon la paramétrisation utilisée. On constate que la figure D n'est plus un quadrilatère valide pour le design de pièces mécaniques, et son module d'analyse risque fort de produire une erreur de calcul due au croisement des segments AC et BD.

En fait, chaque fois que l'on place un nouveau point ou une nouvelle entité (point, courbe, etc.) dans un design, il importe de se demander dans quelle limite il est préférable que cette entité demeure. S'il est possible de cerner ces bornes, il est alors possible de poser comme paramètre les dimensions relatives entre ces bornes. Si ces dimensions ne sont pas bornées dans toutes les directions (comme par exemple la position verticale du point C sur la figure 3.2a), il importe alors de se demander si la dimension ne pourrait pas être adimensionnalisée en la divisant par une autre dimension appropriée. Un autre problème survient alors : qu'est-ce qu'une dimension appropriée ?

Dans un problème de design structurel, une dimension appropriée devrait être une dimension qui procure aux paramètres une signification physique pour le problème actuel. Dans l'exemple de la figure 3.2a**,** la position verticale du point C est définie comme une proportion de *L***,** avec l'implication que tout changement de la variable absolue *L* redimensionnera aussi la hauteur du quadrilatère et, dans ce cas-ci, provoquera d'ailleurs une modification de l'échelle du quadrilatère en entier. Est-ce souhaitable ? Cela dépend beaucoup de l'application de la pièce. Si les proportions de la pièce restent généralement constantes selon son application mais que c'est uniquement sa taille qui

change, alors la paramétrisation est appropriée. Afin d'illustrer le phénomène, les figures 3.2e et 3.2f démontrent l'impact d'une élongation de la longueur *L* pour chacune des paramétrisations respectives 3.2a et 3.2b.

Si la hauteur de la pièce et sa longueur ont tendance à être indépendantes selon l'utilisation, alors rendre adimensionnel le paramètre positionnant verticalement le point C en le divisant par *L* est probablement une moins bonne idée que de laisser le paramètre absolu Y_1. En effet, dans un tel cas, tout changement de *L* par un algorithme d'optimisation obligerait alors cet algorithme à modifier également Y_1 pour conserver sa solution optimale, créant une interdépendance entre les variables qui ajoute une difficulté non nécessaire pour l'optimiseur.

On constate donc qu'il n'est pas nécessairement aisé de produire une paramétrisation qui minimisera les failles du modèle mathématique du système tout en réduisant autant que faire se peut les difficultés d'un éventuel algorithme d'optimisation. Pour cette raison, et aussi parce qu'il est difficile de reproduire artificiellement la créativité humaine lors de la mise en œuvre d'un nouveau modèle paramétrique, cette étape du design devra être effectuée manuellement. La figure 3.3 synthétise toutefois la discussion contenue dans les paragraphes précédents à propos des bonnes pratiques de paramétrisation.

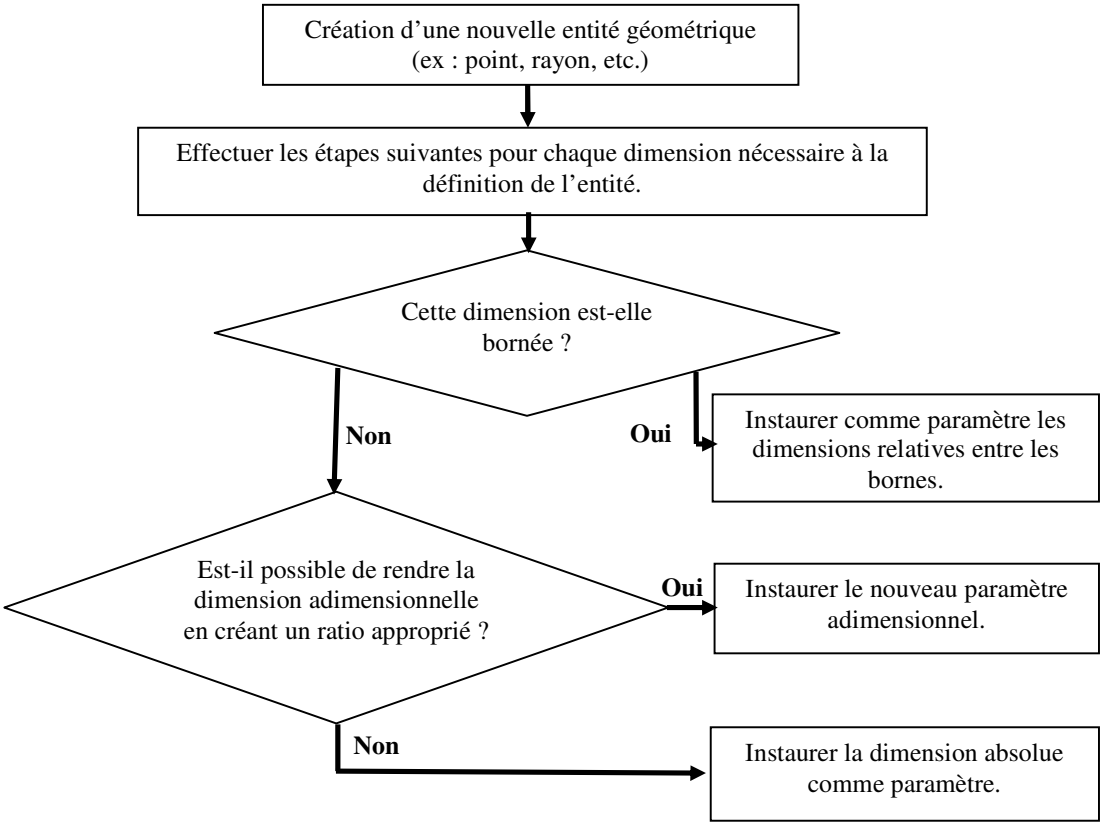


Figure 3.3 : Création d'une paramétrisation appropriée pour un algorithme d'optimisation de design structurel.

Un lecteur intéressé par les bonnes pratiques de paramétrisation en design structurel pourra se référer à Kim (2003) et à Choi & Kim (2005) pour un exposé plus approfondi sur le sujet.

Une fois le modèle paramétrique créé, il importe ensuite de l'introduire dans la banque de données de solutions existantes pour le design afin de permettre à l'algorithme de RPC de l'utiliser pour le design. Puisque le RPC utilisera d'abord les modèles ayant les spécifications $\vec{p}$ les plus rapprochées du problème actuel, le nouveau modèle potentiel doit être inséré temporairement dans la banque de données avec les spécifications exactes du problème actuel afin de s'assurer que le RPC le choisira comme solution

potentielle. Une fois le design effectué, il importe toutefois de ne laisser que la véritable solution au problème dans la banque de données.

Paramètres non numériques (chaînes de caractères)

La figure 3.3 n'est évidemment valide que pour les paramètres numériques, les chaînes de caractère n'ayant pas à être bornées ou adimensionnalisées. Lors de la création du modèle, l'utilisateur devra toutefois se questionner avant de créer un paramètre non numérique.

En effet, tout au long du processus de design, les paramètres non numériques, qui représentent en général des modifications majeures de la solution, ne seront ni interpolés ou extrapolés, ni modifiés, ni optimisés pour une solution donnée. Une fois la solution initiale choisie lors de la phase recherche, ces paramètres seront pratiquement considérés comme des constantes du modèle.

Si le paramètre ne représente pas un changement drastique de la solution, l'utilisateur peut plutôt décider de numériser le paramètre (par exemple en affectant un nombre entier à chaque valeur possible du paramètre) afin de laisser les algorithmes de recherche et d'optimisation modifier à leur gré cette valeur.

Création de la fonction coût F($\vec{x}$), des variables de sortie $\vec{t}$ et gestion des contraintes pour l'entraînement d'un modèle substitut.

Lors de la création d'un modèle paramétrique pour ce processus de RPC, il importe aussi de définir un modèle d'analyse capable de quantifier à quel point une solution utilisant ce modèle permet de bien accomplir ses objectifs.

Cette fonction d'analyse G($\vec{x}$) pourra être analytique, ou bien numérique à l'aide de logiciels d'analyse par éléments finis. La construction de cette fonction d'analyse est très dépendante du type particulier de design et dépasse le cadre de ce document.

Toutefois, afin d'obtenir la vraie fonction coût F($\vec{x}$) du problème, on doit ajouter l'effet des contraintes pour ce design particulier. En effet, généralement, la différence entre deux problèmes de design similaires du RPC est l'ensemble de contraintes qui est utilisé.

Plusieurs méthodes de gestion de contraintes existent dans la littérature. Parmi les plus utilisées, on trouve les fonctions de mérites et la peine de mort (Yeniay, 2005). En général, ces méthodes modifient la valeur de la fonction coût par une autre fonction qui dépend de la valeur des contraintes. La valeur de la pénalité produite par les contraintes est en principe directement reliée aux spécifications $\vec{p}$ du problème en plus d'être reliée à la valeur des paramètres d'une solution. Cette fonction sera donc nommée H($\vec{x}, \vec{p}$). La fonction coût devient alors :

$$F(\vec{x}, \vec{p}) = G(\vec{x}) + H(\vec{x}, \vec{p}) \qquad (3.1)$$

Le choix de la meilleure façon de générer H($\vec{x}, \vec{p}$) est très dépendant du type de problème à résoudre (Yeniay, 2005). Les méthodes dites à peine de mort appliquent une pénalité fixe et n'ont pas tendance à ramener la solution d'un optimiseur dans le domaine réalisable (voir annexe B.1.2). À l'inverse, les fonctions de mérite appliquent une pénalité qui dépend du niveau d'effraction de la contrainte. Ces fonctions de mérite ont donc tendance à laisser aller la solution dans le domaine irréalisable si le coefficient de pénalité n'est pas géré correctement ou que certaines zones de G($\vec{x}$) disposent d'un gradient extrêmement élevé. Le processus de RPC proposé utilisera une combinaison hybride originale entre ces deux types de méthodes et calculera H($\vec{x}, \vec{p}$) grâce à l'équation 3.2 présentée ci-dessous :

$$H(\vec{x}, \vec{p}) = \sum_{m=1}^{U} \left(vg_m(\vec{x}, \vec{p}) + DP_m(\vec{x}, \vec{p}) \right) \qquad (3.2)$$

où $DP_m(\vec{x}, \vec{p}) = -\psi$ si $g_m(\vec{x}, \vec{p}) \neq 0$

$DP_m(\vec{x}, \vec{p}) = 0$ si $g_m(\vec{x}, \vec{p}) = 0$

$g_m(\vec{x}, \vec{p}) = 0$ si la contrainte $h_m(\vec{x}, \vec{p}) \geq 0$ est satisfaite.

$g_m(\vec{x}, \vec{p}) = h_m(\vec{x}, \vec{p})$ si $h_m(\vec{x}, \vec{p}) < 0$

v est un coefficient de pénalité, initialement 100

ψ est une pénalité de mort, 10^{50} par défaut

U est le nombre total de contraintes du modèle

Plusieurs façons de gérer v sont proposées dans la littérature (Yeniay, 2005). La plupart nécessitent de spécifier un ou plusieurs paramètres adaptés au problème. Puisque l'on désire avoir un algorithme qui automatise au maximum le procédé, et donc qui requiert le moins de paramètres possible, l'approche proposée est celle de Hadj-Alouane & Bean (1997), qui adapte v à l'état actuel de l'optimisation. La constante v variera donc comme suit :

Initialement, $v = 100$.

Au début de chaque génération :

Si toutes les solutions contenues dans la population enfreignent au moins une contrainte :

$$v = 10\, v$$

Si aucune des solutions n'enfreint de contraintes :

$$v = v / 10$$

Si $v > \psi$ alors

$$v = \psi$$

Ainsi, dès qu'une contrainte n'est pas respectée, l'optimiseur évalue la solution comme étant une mauvaise solution grâce à la pénalité de mort. Toutefois, si aucune des solutions ne satisfait les contraintes, l'algorithme aura tendance à revenir doucement vers le domaine réalisable grâce aux pénalités décroissantes $v\, g_m(\vec{x}, \vec{p})$.

Cette approche est validée grâce à l'application d'un exemple analytique simple à l'annexe B.1.2. Les résultats obtenus seront comparés avec d'autres méthodes de gestions de contraintes existantes dans la littérature à la section 6.2.

À ce stade du développement du modèle, il est aussi de mise de déterminer certaines variables de sortie $\vec{t}$ importantes pour le calcul des contraintes par un modèle substitut. Ce choix sera discuté plus en détail dans la section 3.4.4.

Pour un problème de design particulier, le vecteur $\vec{p}$ est considéré constant, et par conséquent l'appellation F($\vec{x}$) sera utilisée au lieu de F($\vec{x}, \vec{p}$) dans les sections concernées.

3.2.2 Sélection des solutions de départ pour le processus

À cette étape, le processus de RPC doit sélectionner les modèles de solutions les plus plausibles pour le design actuel. Ces solutions sont considérées être celles dont les spécifications sont les plus rapprochées des spécifications du problème actuel.

La littérature regorge de possibilités sur la façon exacte de calculer la distance entre deux ensembles de spécifications. Bogaerts & Leake (2004) citent les plus communes comme étant la distance euclidienne (*ued*), la distance euclidienne pondérée (*wed*), la mesure maximale (*mms*) et l'erreur quadratique moyenne (*msd*).

Parmi ces méthodes, la distance euclidienne pondérée semble être celle donnant les résultats les plus satisfaisants lorsque la pondération est effectuée de manière experte (Mendes, Mosley & Watson, 2002). Dans le cas d'un processus répété de design de pièces comme celui du design de rotor de turbine à gaz, il est souvent possible d'affecter un expert designer à la tâche qui saura définir adéquatement le vecteur de pondération $\vec{w}$ des spécifications. Cette méthode sera donc celle qui sera utilisée dans ce document.

Notez qu'il existe aussi des méthodes capables d'approximer automatiquement $\vec{w}$ si l'on dispose d'une banque de données suffisante (Bogaerts & Leake, 2004). Par contre, pour être efficaces, ces méthodes nécessitent qu'une quantité importante de données soient disponibles dans la banque de donnée du problème, et ne donnent pas nécessairement un résultat adapté au client et au problème de design actuel. Le choix des pondérations sera donc tout de même laissé à l'utilisateur, et correspondra au vecteur $\vec{w}$ défini lors de la phase d'analyse du processus de design.

Distance euclidienne pondérée

À ce stade du processus de RPC, il importe de calculer le vecteur $\vec{d}$ contenant les distances entre les spécifications actuelles et celles de chacune des solutions de la banque de données.

En utilisant la distance euclidienne pondérée, le calcul de la distance pour une solution bien particulière *j* devient :

$$d_j = \sqrt{\sum_{i=1}^{N_s} w_i \, dist\left(p_{i,j}, p_i\right)^2} \qquad (3.3)$$

avec $dist\left(p_{i,j}, p_i\right) = \left(\frac{p_{i,j} - p_i}{U_{b,i} - L_{b,i}}\right)$ pour $p_i \in \mathbb{R}$

$dist\left(p_{i,j}, p_i\right) =$

0 si $p_{i,j} = p_i$

1 si $p_{i,j} \neq p_i$ pour $p_i \notin \mathbb{R}$

où :

$p_{i,j}$ est la $i^{ème}$ spécification de la solution j.

p_i est la $i^{ème}$ spécification du problème actuel.

$U_{b,i}$ et $L_{b,i}$ sont respectivement les bornes supérieures et inférieures définies pour la spécification i.

N_s est le nombre total de spécifications

Afin d'éviter qu'une spécification prenne moins ou plus de poids que celui qui lui est accordé, il est important de veiller à ce que le terme adimensionnel $\left(\frac{p_{i,j} - p_i}{U_{b,i} - L_{b,i}}\right)$ puisse varier de 0 à 1 pour toutes les spécifications i du problème actuel. Les bornes $U_{b,i}$ et $L_{b,i}$ doivent donc être définies comme étant respectivement la plus grande et la plus petite valeur plausible de p_i.

Pour un i donné, le processus définira donc $U_{b,i}$ comme étant la valeur maximale parmi les $p_{i,j}$, incluant $p_{i.}$ De même $L_{b,i}$ est égal à la valeur minimale de $p_{i,j}$ pour tout j, incluant p_i.

Pour les variables n'étant ni entières, ni réelles (telles que les chaînes de caractère), la valeur de $dist(p_{i,j}, p_i)$ sera simplement considérée égale à 0 si la spécification est la même que celle du problème actuel, ou égale à 1 autrement.

Si une spécification est inexistante pour une certaine solution, la formule 3.3 ne produit pas de résultat. Dans un tel cas, Bogaerts & Lake (2004) suggèrent plusieurs façons de mesurer $dist(p_{i,j}, p_i)$ pour la spécification manquante. Dans le cas d'un problème de design, une spécification pour laquelle le designer n'a pas donné de valeur signifie généralement que la solution ne répond tout simplement pas à cette spécification (par exemple, si la spécification est le matériel de recouvrement d'une ailette de rotor alors que la solution envisagée n'a pas de recouvrement). Le processus actuel utilisera donc la façon la plus simple de mesurer $dist(p_{i,j}, p_i)$, qui est d'appliquer la valeur 1 par défaut pour $dist(p_{i,j}, p_i)$ lorsque $p_{i,j}$ n'existe pas. De cette façon, les solutions qui répondent à la spécification seront privilégiées par rapport à celles pour lesquelles cette spécification est manquante.

Solutions initiales

Une fois $\vec{d}$ connu, il importe alors simplement de déterminer le nombre k_{ile} de solutions initiales voulues par l'utilisateur. Les solutions j ayant les $(k_{ile}-1)$ plus petites distances d_j seront alors choisies comme solutions initiales pour la suite du processus.

Notez qu'un nombre égal à $k_{ile}-1$ solutions existantes (au lieu de k_{ile}) est choisi à l'aide de la distance euclidienne pondérée. La raison pour ceci est qu'un réseau de neurones sera utilisé dans la phase d'adaptation pour générer la $k_{ile}^{ième}$ solution initiale au problème. La section suivante traitera de ce processus.

3.3 Phase d'adaptation

La phase d'adaptation est la phase d'un design de type RPC durant laquelle les solutions sélectionnées dans la phase de recherche doivent être adaptées aux spécifications actuelles.

L'adaptation est possiblement la phase la plus difficile d'un processus de RPC. En fait, la plupart des applications du RPC ne comprennent que peu ou pas de phase d'adaptation (Leake, 2000), réduisant le RPC à un système de recherche de solutions antérieures dans une banque de données. Dans de tels cas, l'adaptation est laissée à un utilisateur humain, et le RPC sert uniquement à lui fournir du matériel initial avec lequel travailler.

Afin de tenter d'améliorer cette lacune, Whar & Babka (1998) suggèrent l'utilisation d'un réseau de neurones qui pourrait modifier efficacement les solutions pour tenir compte des nouvelles spécifications. La procédure de design proposée s'inspire de cette théorie et utilise un réseau de neurones artificielles de type perceptron (voir annexe A) dans la phase d'adaptation du design.

Avramenko & Kraslawski (2008), quant à eux, énoncent que le processus de design est évolutionnaire et, qu'en ce sens, un algorithme génétique convient parfaitement à l'adaptation du design. Dans la thèse actuelle, l'auteur considère important de séparer en deux phases distinctes les méthodes utilisant le savoir acquis a priori sur le domaine de design et les méthodes produisant un résultat à partir du savoir à acquérir en temps réel lors du design. Aussi, l'approche utilisant un algorithme évolutionnaire est implémentée dans le processus de design, mais plutôt dans une nouvelle phase de raffinement, qui a été greffée à la sortie de la phase d'adaptation.

La méthode consistant en l'utilisation d'un algorithme d'optimisation ou d'un réseau de neurones pour transformer une solution lors de la phase d'adaptation se nomme

« adaptation de transformation » (Zhang, Louvieris & Petrou, 2007) et n'est pas originale en soi. Par contre, l'élément d'originalité de cette section de la thèse tient de l'utilisation successive des deux méthodes (réseau de neurones et optimisation), ainsi que de l'insertion du concept de raffinement progressif (section 3.4.2) dans le processus de RPC pour choisir les variables qui seront intéressantes pour l'algorithme d'optimisation.

Utilisation de réseaux de neurones de type perceptron dans la phase adaptation

L'annexe A.3 de ce document présente les réseaux de neurones de type perceptron, qui sont utilisés dans cette phase d'adaptation du design.

En principe, chaque solution existante est constituée d'un modèle paramétrique, ainsi que d'une valeur précise pour chacun des paramètres du modèle. L'idée ici est donc d'entraîner un réseau de neurones à identifier la valeur résultante pour chacun des paramètres d'un modèle, en utilisant les spécifications comme variables d'entrée.

Selon la théorie des réseaux de neurones, pour ce genre d'application, il serait possible d'utiliser un seul réseau pour déterminer la valeur de chacune des variables de sortie (voir figure 3.4).

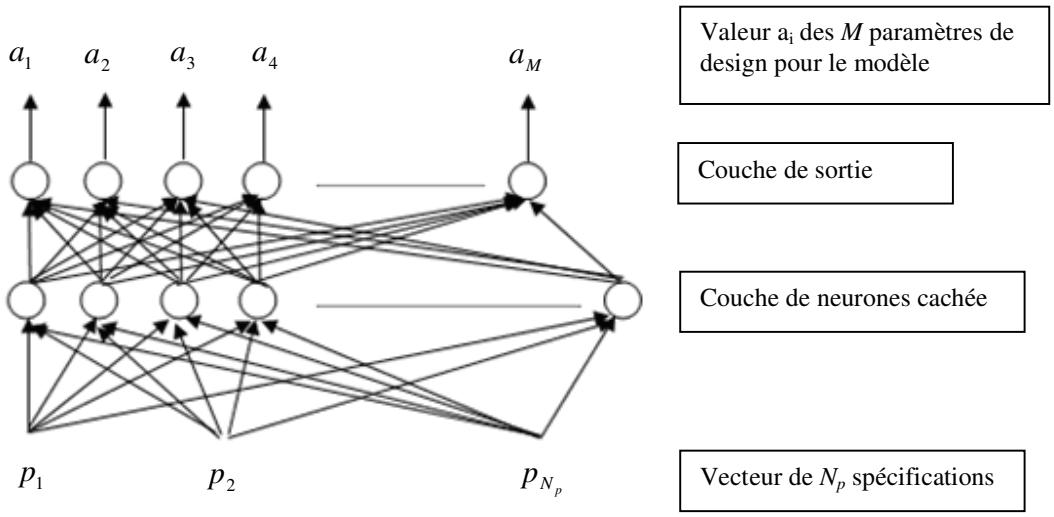


Figure 3.4 : Application d'un seul perceptron pour un modèle

Par contre, pour un système de grande envergure, cette façon de procéder implique une topologie de réseau de neurones qui n'est pas nécessairement adaptée à la complexité de chacune des sorties du réseau. Par exemple, si une sortie varie de façon linéaire avec un seul des paramètres d'entrée, un réseau contenant un seul neurone pourrait parfaitement convenir à cette sortie. Par contre, une autre sortie beaucoup plus complexe pourrait nécessiter une couche de neurones cachés plus dense pour être estimée de façon adéquate. De plus, lorsqu'un algorithme d'entraînement tentera de réduire l'erreur quadratique moyenne d'un réseau, s'il y a plusieurs paramètres de sortie, l'erreur moyenne diminuera, mais il est fort possible que la qualité de l'évaluation de certains paramètres s'en trouve amoindrie.

Ainsi, afin de produire un entraînement parfaitement adapté à chaque sortie du réseau, nous tenterons une nouvelle approche dans laquelle un réseau de neurones indépendant est créé puis entraîné pour chacune des sorties du réseau. La figure 3.5 illustre la structure des réseaux de neurones employés, alors que la figure 3.6 illustre les liens utilisés entre ces réseaux pour générer une solution au problème d'adaptation.

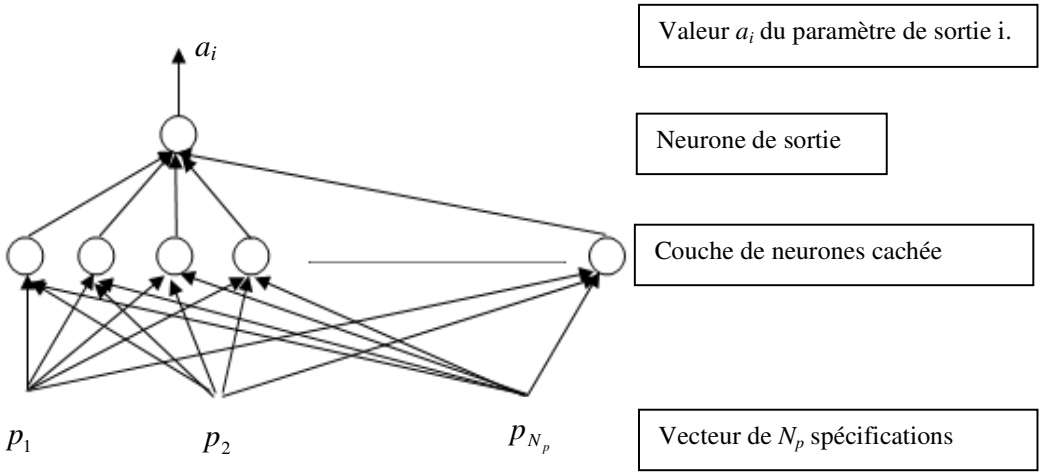


Figure 3.5 : Application d'un perceptron pour une seule variable de sortie

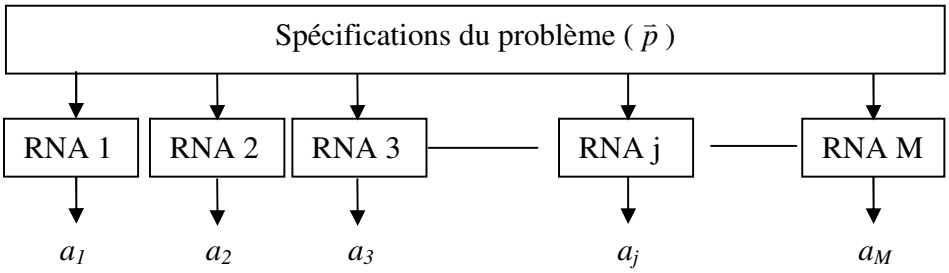


Figure 3.6 : Utilisation d'un RNA par variable de sortie pour générer les valeurs des paramètres d'un modèle.

La section 7.12 de ce document validera le potentiel de cette dernière approche en comparant l'erreur finale obtenue pour un même nombre de passes d'apprentissage pour le problème de design d'un disque de turbine à gaz selon l'utilisation d'un seul perceptron (figure 3.4) ou de plusieurs (figure 3.6).

Évidemment, puisqu'il s'agit d'un modèle numérique, le perceptron ne pourra fournir de valeur que pour les paramètres à valeurs entières ou réelles. Pour une solution estimée par le réseau de neurones, la valeur des paramètres non numériques sera alors posée

comme étant égale à la valeur de ces paramètres pour une solution spécifique de la banque de données. Cette solution spécifique sera choisie comme étant la solution la plus rapprochée (selon l'équation 3.3) parmi celles utilisant le même modèle paramétrique.

Notez aussi que le perceptron ne peut fournir que des nombres réels. Ainsi, si le paramètre a_i est un nombre entier, il sera nécessaire d'arrondir la solution au nombre le plus près pour obtenir sa véritable valeur estimée (Hagan, Demuth & Beale, 2002).

La solution neurale qui sera évaluée utilisera le réseau de neurones estimant $\vec{a}$ pour le modèle paramétrique de la solution la plus rapprochée des spécifications actuelles. Si ce réseau n'existe pas, par exemple parce que la solution la plus rapprochée est la seule solution connue utilisant ce modèle, c'est le réseau du modèle de la $2^{ème}$ solution la plus éloignée qui servira. Si aucune des solutions sélectionnées ne dispose d'un modèle paramétrique associé à un réseau de neurones (par exemple parce que la banque de données est encore pratiquement vierge), la solution neurale sera remplacée par la $k_{ile}^{\ ième}$ solution la plus rapprochée des spécifications actuelles.

Entraînement du réseau

Les réseaux de neurones utilisés dans cette phase s'entraînent en utilisant toutes les solutions ayant des modèles similaires contenues dans la banque de données comme patrons d'entraînement.

Chaque solution doit être enregistrée avec les spécifications qui l'ont engendrée, de telle sorte que le réseau de neurones pourra utiliser la structure de la figure 3.6 pour un modèle paramétrique donné.

L'entraînement est effectué en suivant la technique mentionnée à l'annexe A.4.

3.4 Phase de raffinement

La phase de raffinement qui a été ajoutée au processus de RPC est en quelque sorte une simple extension de la phase d'adaptation. En effet, il s'agit toujours d'adapter les solutions initiales trouvées à de nouvelles spécifications. Toutefois, l'auteur a jugé bon de séparer les deux phases pour bien mettre l'emphase sur le caractère distinct de chacune : à savoir que la phase d'adaptation utilise le savoir acquis antérieurement sur le problème pour adapter la solution, alors que la phase de raffinement tente de découvrir un savoir additionnel qui lui permettra d'améliorer cette même solution.

Une phase de raffinement est constituée de quatre étapes bien précises :

1. préparer les paramètres
2. sélectionner un ensemble de paramètres variables
3. optimiser la valeur de ces paramètres.
4. retourner à l'étape 2 si le raffinement n'est pas terminé, ou fin.

L'algorithme de raffinement est conçu pour utiliser successivement de plus en plus de paramètres variables à mesure qu'il raffine le design en question, d'où la boucle de retour sur les étapes 2 et 3.

3.4.1 Préparation des paramètres

Pour faciliter la tâche des réseaux de neurones et réduire les possibilités d'erreur numérique en cours d'optimisation, il est recommandé de rendre adimensionnels tous les paramètres potentiellement choisis comme variables par l'algorithme de façon à ce que leur valeur varie entre 0 et 1.

Pour ce faire, l'utilisateur devra spécifier pour chacun des paramètres i ses bornes maximales $a_{max,i}$ et minimales $a_{min,i}$ possibles. Notez que cette tâche sera beaucoup

moins complexe à effectuer si le modèle paramétrique inclut d'emblée plusieurs variables adimensionnelles (voir section 3.2.1).

Le reste de la phase de raffinement utilisera donc la représentation adimensionnelle $\vec{x}$ des paramètres de valeur $\vec{a}$.

$$x_i = \frac{a_{\max,i} - a_i}{a_{\max,i} - a_{\min,i}} \qquad (3.4)$$

3.4.2 Sélection d'un sous-ensemble de paramètres variables

Les systèmes de grande envergure sont des systèmes possédant une grande quantité de variables. Bien qu'il soit difficile de définir exactement ce que constitue une grande quantité de variables pour un problème d'optimisation non linéaire en design mécanique, on peut parler facilement d'un nombre se situant entre quelques dizaines et plusieurs centaines de variables à déterminer.

L'un des principaux problèmes inhérents aux problèmes d'optimisation de grande envergure est l'énorme quantité de calculs requis pour pouvoir trouver une solution. Dans le cas de la plupart des algorithmes, on sait que la durée de l'optimisation augmente de façon exponentielle avec le nombre de variables en présence.

Toutefois, lorsque l'on dispose d'une grande quantité de variables pour un problème, il est rare que la totalité de ces variables soit également utile pour réduire la fonction coût.

Il est donc proposé (Yang & Honavar, 1999) de poser constantes toutes les variables pour lesquelles la sensibilité de la fonction coût est inférieure de plusieurs ordres de grandeur à celle des d'autres variables.

Une fois ceci fait, il est alors théoriquement possible de lancer une optimisation avec les variables restantes. Cette optimisation sera en principe moins coûteuse que ne l'aurait été l'optimisation du problème entier puisque le domaine de solutions s'en trouve restreint. Ensuite, suivant le principe du raffinement progressif (Kim & Weck, 2005), il est possible d'utiliser la solution obtenue comme point initial d'un nouveau problème où seront incorporées graduellement les autres variables.

Raffinement progressif

Dans leur article, Kim & Weck (2005) utilisent un algorithme génétique à longueur de chromosome variable afin d'optimiser la topologie d'un pont.

S'inspirant des algorithmes énoncés dans l'article de Ryoo & Hajela (2004), ils créent un algorithme évolutionnaire optimisant d'abord une topologie, puis allongeant progressivement la longueur d'un chromosome en y ajoutant des paramètres afin de raffiner la solution obtenue. La figure 3.7 illustre certains des résultats qu'ils ont obtenus.

L'avantage de procéder ainsi, plutôt que de partir avec une solution initiale aléatoire et d'optimiser de front une énorme quantité de variables, est de trouver d'abord grossièrement une solution initiale avec quelques variables plus importantes, puis, partant de cette solution, d'ajouter graduellement des variables en augmentant la longueur du chromosome jusqu'à obtenir une solution adéquate. On évite donc des évaluations inutiles de la fonction coût qui auraient eu lieu en raffinant à prime abord une solution encore grossière.

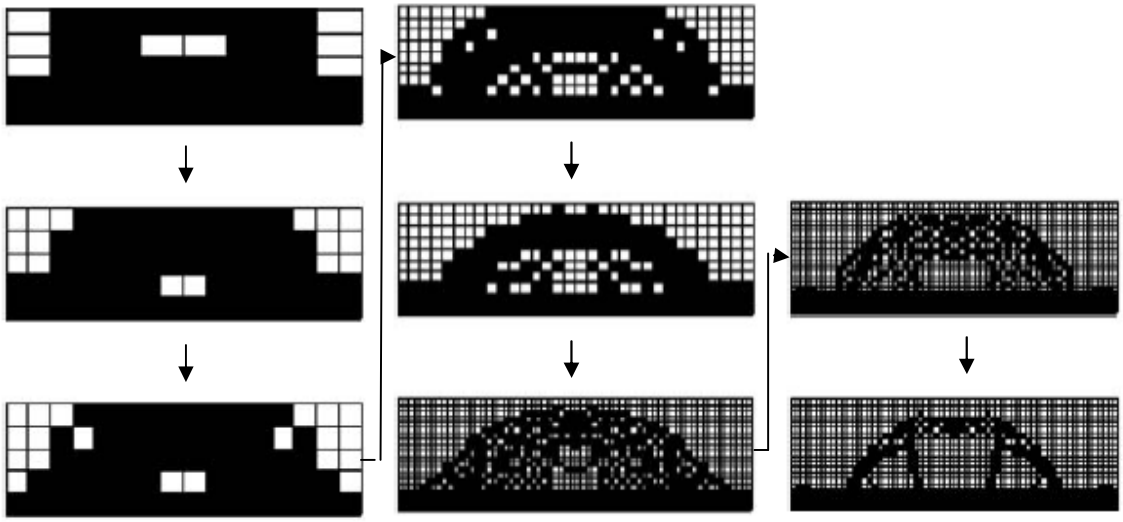


Figure 3.7 : Optimisation de la topologie d'un pont par raffinement progressif (Kim & Weck 2005)

Les bons résultats qu'ils obtiennent rejoignent notre discussion précédente à savoir qu'un algorithme sera plus performant s'il utilise d'abord les paramètres les plus importants pour effectuer des ajustements grossiers avant d'effectuer des ajustements plus fins à l'aide des autres paramètres.

La section 7.10 de ce document présente une série de tests visant à valider cette approche pour la méthodologie de design de type RPC qui est utilisée.

Calcul de la sensibilité de la fonction coût pour chaque variable

L'une des principales méthodes que l'on retrouve dans la littérature pour éliminer des variables est d'étudier la sensibilité de la fonction coût autour d'un certain point d'optimisation et ensuite de définir un seuil sous lequel on considérera que la variable donnée n'introduit que du bruit dans la fonction.

Par exemple, en évaluant certains points de la fonction, on peut trouver un modèle de régression linéaire pour remplacer la fonction coût et ignorer toutes les variables dont le coefficient est sous un certain seuil (Bosman & Thierens, 2005).

Dans le même ordre d'idée, plusieurs chercheurs utilisent les SVM (support vector machine) pour tracer deux hyperplans qui séparent les variables ayant une influence moindre des variables ayant une plus grande influence sur la fonction coût (Graf, Smola & Borer, 2003).

Une autre façon de réduire le nombre de variables dans un système est de déterminer si cette variable a ou non de l'influence sur le système grâce à un réseau neuronique (Yang & Honavar, 1999 ; Guo & Uhrig, 1992 ; Hornyak & Monostori, 1997).

Étant en quelque sorte un modèle statistique, un réseau de neurones est alors utilisé comme modèle local pour le problème en cours, et la sensibilité estimée S_i de la fonction coût F($\vec{x}$) pour un paramètre x_i peut être évaluée par différence finie à l'aide de l'équation suivante :

$$S_i = \left| \frac{\tilde{F}(\vec{x}) - \tilde{F}(\vec{x} + \Delta x_i)}{\Delta x_i} \right| \tag{3.5}$$

où $\tilde{F}$ ($\vec{x}$) est la valeur de la fonction coût telle qu'estimée par le RNA substitut.

Si aucun RNA n'est disponible pour la fonction F($\vec{x}$), par exemple parce qu'elle n'a jamais encore été évaluée, la sensibilité sera posée par défaut égale à 1 pour tous les paramètres numériques. De cette façon, tous les paramètres seront sélectionnés comme variables par l'algorithme lors de la première phase de raffinement et les évaluations qui y seront effectuées serviront à entraîner le réseau pour les phases subséquentes. L'option

est toutefois laissée à l'utilisateur d'utiliser alors plutôt la véritable fonction coût F($\vec{x}$) du système.

Si le paramètre x_i n'est pas un caractère numérique, il sera d'emblée considéré comme une constante par l'optimiseur et la valeur correspondante de S_i sera donc de 0.

Notez que tout modèle substitut capable de bien approximer F($\vec{x}$) pourrait convenir pour cette phase du processus de design. Toutefois, puisque le processus actuel entraîne déjà un réseau de neurones pour servir de substitut à F($\vec{x}$) pour l'algorithme d'optimisation, ce même réseau pourra être utilisé à la fois pour la sélection et l'optimisation des paramètres. Il s'avère donc judicieux de l'utiliser pour ces deux cas afin de ne pas perdre le calcul effectué lors de son apprentissage.

Sélection des Δx_i

Le choix des Δx_i est un facteur crucial permettant de calculer correctement l'importance de chacune des variables pour le problème actuel. En effet, si la topographie de la fonction est cyclique ou dispose de plateaux, un pas mal choisi pourrait afficher une sensibilité près de 0, même si le paramètre est important (voir figure 3.8).

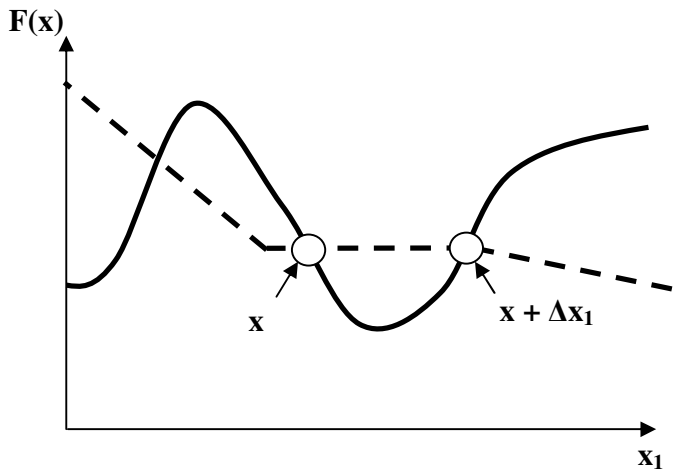


Figure 3.8 : Erreur d'approximation de S_1 suite à un mauvais choix de Δx_1

Il importe donc ici de se rappeler que l'on désire évaluer $\vec{S}$ uniquement pour pouvoir classifier l'importance relative des paramètres numériques du système. Ainsi, même si la valeur calculée de $\vec{S}$ n'est pas exacte, l'important est que S_i soit grand si F($\vec{x}$) varie beaucoup lors d'une variation de x_i.

Puisque l'on dispose d'un modèle substitut permettant d'évaluer rapidement plusieurs valeurs de la fonction coût, il est alors possible de réduire le risque de sous-évaluer l'importance d'un paramètre en sélectionnant plusieurs Δx_i différents et en utilisant uniquement la plus grande valeur de S_i obtenue à l'aide de ces Δx_i.

Par défaut, l'algorithme de raffinement sépare le domaine de x_i en 11 parties distinctes (0, 0.1, 0.2… 1), puis calcule et choisit les Δx_i permettant d'atteindre ces points afin de calculer la sensibilité S_i pour cette variable. Bien sûr, le nombre exact et la teneur des Δx_i qui devraient être utilisés dépendent beaucoup de la nature du problème résolu. Pour cette raison, l'algorithme de design utilisé permet alternativement de choisir manuellement le nombre et la valeur numérique maximale des Δx.

Calcul de l'importance relative $\vec{S}_r$ de chaque x_i

La sensibilité S_i pour une variable est un nombre variant de 0 à l'infini. Il peut donc s'avérer difficile de bien visualiser l'impact relatif de chacune des variables.

Pour cette raison, l'importance relative, qui est en fait la sensibilité relative $S_{r,i}$ de chaque variable sera calculée avant d'effectuer le choix de l'ensemble de variables destinées à l'optimisation. $S_{r,i}$ peut être évaluée à l'aide de l'équation suivante :

$$S_{r,i} = 100\left(\frac{S_i}{\max_{i=1}^{M}(S_i)}\right) \tag{3.6}$$

Ainsi, $S_{r,i}$ est une mesure variant de 0 à 100, où 0 dénote un paramètre n'ayant aucune influence sur la fonction coût, alors que 100 représente le paramètre ayant la plus grande sensibilité de tous les x_i.

Choix d'un ensemble de variables parmi les paramètres existants

Une fois les importances relatives des paramètres calculées, il suffit ensuite de définir un seuil minimal d'importance S_{min}. Tous les paramètres x_i disposant d'une importance relative $S_{r,i} \leq S_{min}$ ne seront considérés que comme du bruit pour $F(\vec{x})$ et seront donc considérés constants par l'algorithme d'optimisation ultérieur.

Encore une fois, le choix idéal de S_{min} dépend beaucoup du problème actuel et une liberté de modifier les valeurs par défaut est laissée à l'utilisateur. Par défaut, il est suggéré de permettre à S_{min} de filtrer initialement tous les paramètres pour lesquels $S_{r,i}$ est plus faible que l'importance relative maximale de deux ordres de grandeurs et plus.

Nous verrons plus tard que l'algorithme de raffinement effectuera une boucle afin d'inclure de plus en plus de paramètres dans l'optimisation. Ainsi, si on dénote $S_{\min}^{z}$ comme le seuil minimal d'importance requis pour un paramètre afin qu'il soit considéré variable lors de la phase de raffinement z, on obtient :

$$S_{\min}^{z} = \frac{S_{\min}^{z-1}}{C_s} \quad \text{pour } z = 1,2\ldots z_{max}\text{-}1 \qquad (3.7)$$

$$S_{\min}^{z} = 0 \quad \text{pour } z = z_{max}$$

où $S_{\min}^{0} = 100$

C_s est une constante définie par l'utilisateur, et est égale à 10^2 par défaut.

On observe alors que, lors de la dernière phase de raffinement, seuls les paramètres ayant une influence nulle sur la fonction coût sont exclus de l'optimisation.

Notez que, tel que mentionné précédemment, toutes les sensibilités sont posées égales à 1 dans le cas où aucun RNA substitut n'est disponible lors de la première phase de raffinement. Dans un tel cas, il est alors important de poser $S^1_{\min} = S^0_{\min}$ pour ne pas accélérer inutilement l'insertion de variables dans le procédé. En effet, puisque les importances $S_{r,i}$ ne seront alors pas calculées lors de la première phase de raffinement, le seuil $S^0_{\min}$ estimé par l'utilisateur ne sera pas non plus utilisé lors de cette phase. On l'utilisera donc plutôt à la seconde phase de raffinement, d'où l'utilisation de $S^1_{\min} = S^0_{\min}$.

À partir de ce point, nous dénoterons par $\vec{y}_{z,j}$ le vecteur contenant tous les paramètres de $\vec{x}$ ayant été choisis comme des variables lors de la phase *z* du raffinement, à partir de la solution numérotée j.

3.4.3 Optimisation numérique des paramètres

L'optimisation numérique étant une science en soi à part du processus de design, l'algorithme évolutionnaire qui a été développé pour cette application sera présenté en détail dans le deuxième volet de cette thèse (au chapitre 5). Pour l'instant, il suffit de savoir qu'il s'agit d'un algorithme évolutionnaire de type génétique.

Cette portion du document s'intéressera donc plus précisément à l'initialisation et l'utilisation de l'information extraite à la sortie de cet algorithme génétique.

Initialisation de l'algorithme

À ce stade, nous disposons d'une série de k_{ile} solutions initiales, chacune étant représentée par un vecteur de paramètres $\vec{x}_{ini}$. Pour chacune de ces solutions *j*,

l'algorithme applique la procédure de calcul définie dans la section 3.4.2 et obtient un ensemble de variables représentées par $\vec{y}_{z,j}$.

L'initialisation d'un algorithme génétique régulier implique de définir une population initiale aléatoire dont la taille est définie par l'utilisateur. La valeur de chaque variable y est choisie aléatoirement dans son domaine possible (variant ici de 0 à 1 pour chaque x_i numérique).

Toutefois, pour un problème industriel de grande envergure, la plupart de ces solutions initiales seront possiblement hors du domaine réalisable, et il est fort possible qu'un effort considérable de calcul doive être mis en œuvre avant même d'avoir une solution au problème capable d'égaler la performance des solutions initiales envisagées par l'utilisateur.

Pour guider l'algorithme dans ses premières itérations, il s'agit alors d'utiliser les solutions sortant du processus de RPC afin d'ensemencer la population initiale de l'algorithme génétique. Cette approche a été proposée et implémentée par Ramsey & Grefenstette (1993), puis par Jonhson & Sushil (2005) avec un succès considérable, et semble une façon prometteuse de permettre le raffinement de problèmes de structure industriels. L'originalité du développement présenté ici tient du fait qu'au lieu d'utiliser simplement les solutions du RPC comme individus initiaux d'un algorithme génétique, on s'en sert plutôt comme éléments centraux d'un îlot de solutions, évitant ainsi que les solutions d'apparence moins prometteuses issues d'une piètre adaptation ne soient rapidement éliminées avant d'avoir acquis leur plein potentiel. Les solutions obtenues par cette approche seront comparées avec celles obtenues par l'approche de Jonhson & Sushil (2005) pour le problème de design d'un disque de turbine à gaz à la section 7.13.

Ainsi, même si chaque solution utilise un modèle paramétrique potentiellement différent, cette façon de faire permet d'effectuer des recherches autour de chacune des

solutions et éventuellement de les comparer entre elles, délaissant petit à petit les solutions les moins prometteuses pour raffiner davantage les solutions les plus susceptibles de donner des bons résultats.

Soit

j le numéro identifiant une solution initiale retenue pour le problème

$\vec{P}_{i,j}$ le membre numéro *i* de la population de l'îlot *j*

$\vec{P}_{i,j}$ est un vecteur contenant une valeur valide pour tous les paramètres contenus dans $\vec{x}_{ini,j}$. Par exemple si $\vec{x}_{ini,1}$ = [x_1, x_2, x_3], $\vec{P}_{i,1}$ sera un vecteur contenant une valeur possible pour x_1, x_2 et x_3.

Le nombre d'îlots de l'algorithme sera égal à k_{ile}. À l'intérieur de ces îlots, chacun des individus sera initialisé de la façon suivante :

$$\vec{P}_{1,j} = \vec{x}_{ini,j} \tag{3.8}$$

$$\vec{P}_{i,j} = \text{mutation}(\vec{x}_{ini,j}) \text{ pour } i = 2,3...N_{pop}$$

Le nombre d'individus par îlot est un nombre N_{pop} défini par l'utilisateur.

L'opérateur de mutation employé pour le cas actuel de RPC est décrit à la section 5.5.1. Pour l'instant, il suffit de savoir qu'il s'agit d'un opérateur modifiant aléatoirement un ou plusieurs des paramètres de $\vec{x}_{ini,j}$ inscrit dans $\vec{y}_{z,j}$ et laissant constant les autres paramètres.

La figure 3.9 illustre un exemple d'initialisation des îlots de l'algorithme génétique. Notez que les limites circulaires des îlots indiqués sur la figure ne sont pas des limites réelles puisque l'opérateur de mutation peut en théorie permettre à un individu de se retrouver à n'importe quel endroit de l'espace de design. Toutefois, elles illustrent les situations les plus plausibles pour la population initiale et représentent bien l'effet de l'initialisation de l'algorithme de RPC.

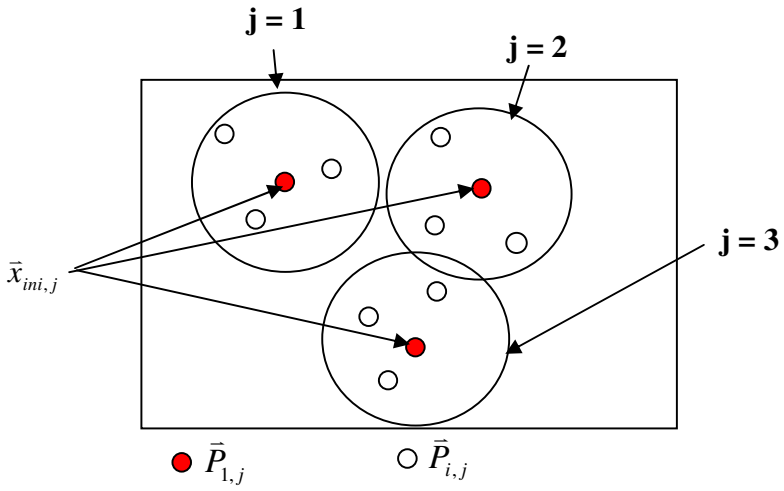


Figure 3.9 : Initialisation de la population d'un algorithme génétique pour k_{ile} = 3 et N_{pop} = 4.

Utilisation de l'information extraite à la sortie de l'algorithme génétique.

Une fois que l'algorithme d'optimisation a convergé vers une série de solutions (ou épuisé le nombre d'évaluations qui était alloué par l'utilisateur pour cette phase de raffinement), le processus effectue la démarche suivante :

1. S'il s'agissait de la dernière phase de raffinement, présenter chacune des solutions générées en vue de la phase de validation. Sinon, passer à 2.
2. Trouver une valeur $imax_j$ pour chaque îlot, qui est la valeur de i représentant la meilleure solution $\vec{P}_{i,j}$.
3. Poser $\vec{x}_{ini,j} = \vec{P}_{i\max_j,j}$ pour $j = 1,2\ldots k_{ile}$

4. $S_{\min}^{z+1} = \frac{S_{\min}^{z}}{10^2}$
5. $z = z + 1$
6. Passer à la phase subséquente de raffinement, en sautant l'étape de préparation des variables (pour aller directement au calcul des sensibilités relatives).

Ainsi, on remplace chaque noyau d'îlot par un nouveau noyau, qui est en fait le meilleur individu de cet îlot, et on réinitialise l'algorithme en utilisant cette fois davantage de paramètres.

Sachant que, pendant l'évolution de l'algorithme génétique, certains individus peuvent migrer d'un îlot à un autre (voir sections D.6.2 et 5.7), on obtient un algorithme qui délaisse lors de chaque raffinement les modèles de solutions les moins prometteurs pour se concentrer sur les solutions disposant de davantage de potentiel.

De plus, puisque les importances relatives des paramètres sont recalculées à chaque phase de raffinement, seuls les paramètres les plus importants dans cette portion de l'espace de design sont utilisés chaque fois. Métaphoriquement, ceci revient en quelque sorte à effectuer un calcul de gradient après un certain nombre d'itérations, afin de choisir la nouvelle direction de recherche.

3.4.4 Entraînement d'un modèle substitut pour F($\vec{x}$)

Pour que le raffinement suggéré fonctionne correctement, un RNA doit être entraîné comme modèle substitut pour chacun des modèles paramétriques de design. Une fois entraîné, l'information de ce modèle pourra être réutilisée lors d'un design futur, à condition que l'architecture du réseau ait été conçue correctement.

En effet, on se rappelle que F($\vec{x}$) est en fait F($\vec{x}, \vec{p}$), et dépend des spécifications du problème. Si on entraîne bêtement un réseau à reconnaître une valeur de F($\vec{x}, \vec{p}$) pour différentes valeurs de $\vec{x}$, le savoir acquis ne sera plus valide dès que l'on sera en présence d'un nouveau problème ayant de nouvelles spécifications.

De même, puisque la valeur en un point de H($\vec{x}, \vec{p}$) dépend de v (équation 3.2), qui varie au cours de l'optimisation, deux évaluations de F($\vec{x}$) à deux moments différents sont susceptibles de fournir deux valeurs très différentes si la solution est hors du domaine réalisable, nuisant considérablement aux efforts de convergence du réseau de neurones si celui-ci s'entraîne à l'aide de valeurs de F($\vec{x}$) utilisant des v différents.

Ainsi, il est plutôt conseillé d'entraîner le réseau à générer G($\vec{x}$) qui est la fonction coût sans contraintes du système. Lors de la génération du modèle, l'usager détermine le nombre de variables de sortie dont il a besoin pour calculer ses contraintes. Le vecteur contenant ces variables se nomme le vecteur $\vec{t}$ et sa longueur est N_t. L'usager devra alors choisir parmi les deux options suivantes pour chacune des contraintes i du système:

1. Si le calcul analytique de l'infraction $h_i(\vec{x}, \vec{p})$ n'est pas coûteux, le calculer analytiquement.
2. Si le calcul $h_i(\vec{x}, \vec{p})$ est coûteux, entraîner un réseau de neurones à estimer $h_i(\vec{x}, \vec{p})$, ou du moins les valeurs $\vec{t}$ permettant de calculer aisément $h_i(\vec{x}, \vec{p})$.

Par exemple, si $h_i(\vec{x}, \vec{p})$ est une fonction simple, telle que $h_i(\vec{x}, \vec{p}) = x_1 - p_1 + x_2$, un réseau de neurones ne sera pas requis. Toutefois, si la contrainte $i = 1$ spécifie que la température maximale du design ne doit pas dépasser $p_1 = 1000^oC$ et que cette température est calculée par un calcul complexe par éléments finis, le choix 2, qui implique l'entraînement d'un réseau de neurones, serait plus approprié. Il importerait

alors d'entraîner le réseau à déterminer la température maximale t_1 du système, puis d'appliquer $h_1(\vec{x}, \vec{p}) = p_1 - t_1$.

Chaque variable de $\vec{t}$ nécessitera un RNA entraîné spécifiquement pour retrouver cette variable. Pour pouvoir entraîner un tel réseau, il sera important d'enregistrer les valeurs de $\vec{t}$ obtenues lors de l'utilisation de la véritable fonction coût. Ces réseaux seront donc entraînés à l'aide des paramètres d'entrée $\vec{x}$ des solutions évaluées ainsi que des valeurs de $\vec{t}$ correspondant à ces solutions. L'architecture recommandée est celle de la figure 3.10.

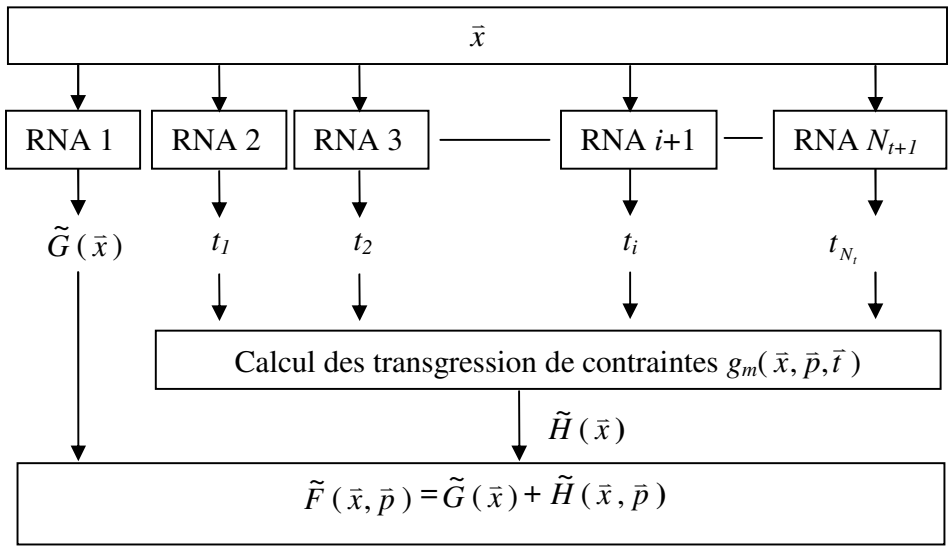


Figure 3.10 : Estimation de F($\vec{x}$) à l'aide d'une série de RNA substituts

Donc, à chaque fois qu'il est discuté d'entraîner le RNA de raffinement dans ce rapport, on fait alors référence à l'entraînement de tous les réseaux de la figure 3.10, utilisant tous les $\vec{x}$ comme patron d'entrée et respectivement les valeurs de G($\vec{x}$) ou de t_i comme patrons de sortie, selon le cas.

Lors de l'évaluation de $\tilde{F}(\vec{x})$ par un modèle substitut, l'algorithme évalue d'abord $\tilde{G}(\vec{x})$ et $\vec{t}$, puis lance le calcul des transgressions de contraintes $g_m(\vec{x},\vec{p},\vec{t})$ que l'utilisateur a programmé avec le modèle paramétrique. Ce calcul fournit la valeur de $\tilde{H}(\vec{x})$ grâce à l'équation 3.2 et permet l'évaluation finale de $\tilde{F}(\vec{x})$.

Cette approche sera validée à la section 7.11 de cette thèse en comparant l'erreur moyenne obtenue à la sortie du réseau par cette méthode avec l'erreur obtenue par un RNA entraîné pour approximer directement $F(\vec{x})$ lors du problème de design d'un disque de turbine à gaz.

3.5 Phase de validation

Cette phase, simple mais vitale au bon fonctionnement du RPC, implique que l'utilisateur vérifie les solutions générées par chaque îlot de l'algorithme, les modifie selon ses désirs et juge s'il dispose d'une solution devant être inscrite ou non dans la banque de données de solutions du problème.

Si le designer juge une solution valide, ses valeurs de $F(\vec{a},\vec{p})$, $\vec{a}$, $\vec{p}$ et $\vec{t}$ seront alors inscrites dans la banque de données pour être utilisées lors des phases de recherche ultérieures. Lorsqu'une nouvelle solution est inscrite dans la banque de données, le RNA de la phase d'adaptation correspondant au modèle paramétrique de cette solution devra être entraîné de nouveau.

De plus, toutes les évaluations réelles de $F(\vec{a},\vec{p})$, qu'elles aient abouties ou non à un résultat valide, sont enregistrées à part dans la banque de données afin de pouvoir servir à l'usager ou à l'entraînement éventuel des RNA substituts pour le modèle.

Cette étape termine une itération du processus RPC. À ce stade, l'utilisateur peut décider de relancer le processus entier, de relancer uniquement la phase de raffinement ou de considérer le processus terminé pour de bon.

3.6 Algorithme de design résultant

La figure 3.11 présente le schéma entier du processus proposé, et par conséquent synthétise les discussions des sections 3.1 à 3.5.

L'algorithme débute avec le problème de design qui se trouve en haut à gauche, et se termine avec la phase de validation située près du coin inférieur droit de la figure. Les cases blanches sont des parties intégrantes du processus même, alors que les cases grisées sont surtout illustrées pour rappeler les interactions entre le processus et les différents modules externes tels que les RNA et la banque de données.

Bien que la plupart des actions du processus soient automatisées par l'algorithme, un certain nombre restent à accomplir manuellement par l'usager, soit :

- Entrer $\vec{p}$ et $\vec{w}$ lors de la phase d'analyse
- Créer les nouveaux modèles paramétriques.
- Choisir k_{ile} lors de la phase de recherche.
- Définir les bornes a_{min} et a_{max} pour chaque paramètre des $\vec{x}_{ini,j}$.
- Déterminer le nombre d'évaluations E_{max} et de phases de raffinement z_{max} requises
- Valider les résultats et choisir ceux qui iront dans la banque de données.

De plus, les options suivantes lui sont laissées pour altérer le fonctionnement par défaut de l'algorithme :

- Utiliser $F(\vec{x})$ au lieu de $\tilde{\tilde{F}}(\vec{x})$ pour le calcul de $\vec{S}$.
- Définir la valeur de C_s
- Définir les paramètres de l'AGENT (voir chapitre 5)
- Entraînement manuel de l'un ou l'autre des réseaux de neurones
- Réinitialisation de l'un ou l'autre des réseaux de neurones
- Désactivation de l'un ou l'autre des réseaux de neurones
- Gérer manuellement les solutions de la banque de données (modification, ajout ou suppression).

Le chapitre 6 présente quelques problèmes analytiques simples qui serviront à valider certains aspects originaux de ce processus. L'algorithme entier sera appliqué au problème industriel du design d'un disque de rotor de turbine à gaz au chapitre 7.

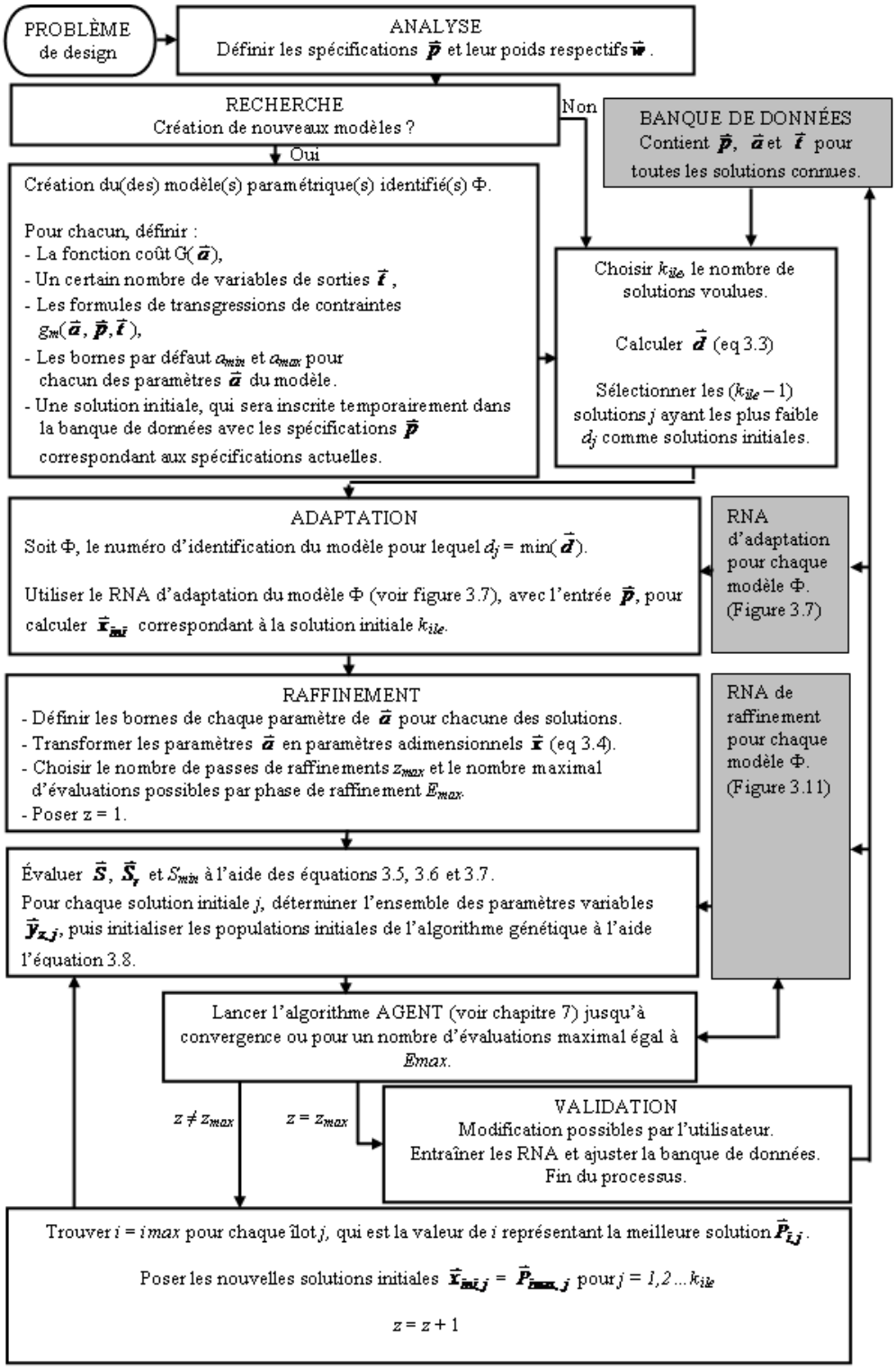


Figure 3.11 : Algorithme de RPC appliqué au design structurel

Chapitre 4.

ÉTAT ACTUEL DE LA SCIENCE EN OPTIMISATION STRUCTURELLE

Le second volet de recherche de cette thèse s'intéresse aux techniques d'optimisation des problèmes de design structurel. Ce chapitre fera donc une brève revue des principales techniques utilisées dans le domaine aux sections 4.1 à 4.3, avant de discuter de celles qui seront les plus utiles pour le problème actuel à la section 4.4.

Dans un de leurs articles, Wolpert & Macready (1995) énoncent le « No-Free-Lunch Theorem », qui est une preuve théorique illustrant qu'aucune méthode d'optimisation n'est en moyenne meilleure qu'une autre pour résoudre l'ensemble des problèmes existants. Ainsi, le gain en efficacité qu'une méthode peut avoir sur une autre pour un problème qui lui est simple est perdu lors de la résolution d'un problème qui lui sera plus aride. En effet, par exemple, aucun algorithme n'est véritablement meilleur que la recherche aléatoire dans le cas d'un problème hautement discontinu et bruité. Cette preuve théorique forte, bien que décourageant la recherche d'un algorithme général pouvant convenir à tous, invite toutefois à approfondir la recherche d'un algorithme convenant mieux au type de problème qui nous intéresse pour un domaine de la science en particulier.

Ainsi, nous ne nous intéressons pas ici à un découvrir un algorithme capable de résoudre tous les problèmes, mais plutôt à sélectionner un algorithme adapté à la résolution des problèmes bien particuliers d'optimisation du design structurel de pièces mécaniques en phase conceptuelle.

De façon générale, le modèle mathématique d'un système peut être illustré tel que sur la figure 4.1 :

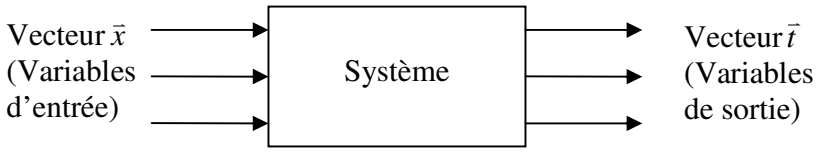


Figure 4.1 : Un modèle mathématique quelconque

L'optimisation consiste donc à déterminer l'ensemble de paramètres $\vec{x}$ donnant les valeurs les plus souhaitables pour $\vec{t}$.

Dans le cas de l'optimisation d'un design structurel de pièces mécaniques, les paramètres définissent généralement les états de forme (par exemple une longueur) et de composition (par exemple le matériau) d'une structure. Les critères seront à la fois donnés par l'environnement (par exemple la température) et par les fonctionnalités requises pour le produit (par exemple le nombre d'heures de fonctionnement).

De façon générale, la littérature définit trois classes de problèmes d'optimisation de design structurel : les problèmes de dimensionnement, de forme et de topologie (Papadrakis, Lagaros, Tsompanakis & Plevris, 2001).

Initialement, l'optimisation structurelle s'intéressait surtout aux problèmes *de dimensionnement*, comme optimiser l'épaisseur des plaques et des coques, et la forme en coupe des poutres d'un dessin 2D.

L'étape suivante est *l'optimisation de forme*, qui optimise à la fois les dimensions et les conditions limites de la structure.

Ensuite, on se rend compte que malgré les optimisations de dimensions et de formes, on obtiendra un résultat peu optimal si la paramétrisation du problème n'est pas adéquate. Certains chercheurs proposent donc des méthodologies pour changer la topologie (ou la paramétrisation) en cours de route, ce que l'on nomme *optimisation de la topologie* (Burns, 2002 ; Gillman, 2005).

La figure 4.2 illustre ces trois différents types de problèmes affectés à l'optimisation structurelle de deux hypothétiques pylônes soutenant un câble, le but étant de minimiser la masse m_i :

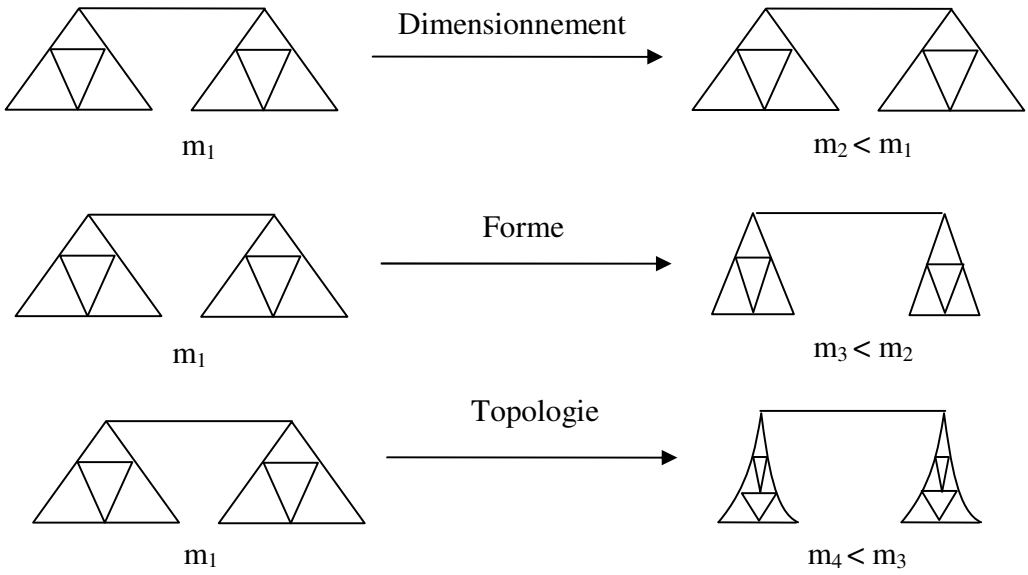


Figure 4.2 : Les trois grands types de problèmes d'optimisation structurelle

Peu importe le type de problème d'optimisation auquel on s'adresse, une fois la topographie choisie, le problème sera de déterminer le vecteur de valeurs d'entrée donnant les meilleures valeurs de sortie possibles.

Mathématiquement, on peut poser le problème d'optimisation comme suit :

$$\textit{Minimiser } F(\vec{x}) \tag{4.1}$$

Sujet à $h_i(\vec{x}) \geq 0 \quad \forall i \in \{1,2...N_c\}$

$s_{inf,i} \geq x_i \geq s_{sup,i} \quad \forall i \in \{1,2...N\}$

où $F(x)$ est la fonction coût du système

$h_i(x)$ représentent N_c contraintes d'inégalités

$s_{inf,i}$ sont les bornes inférieures des N variables x_i

$s_{sup,i}$ sont les bornes supérieures des N variables x_i

Cette formulation implique de définir une fonction coût unique pour un système. Dans le cas d'un problème multi-objectifs (par exemple l'optimisation de l'efficacité et du coût), on devra alors pondérer chacun des objectifs et construire une fonction coût dépendante de ces critères ou utiliser un front de Pareto, qui constitue en quelque sorte l'ensemble des solutions offrant les meilleurs compromis entre les différents objectifs (voir Das, 1997). La section 3.2.1 de ce document traite plus en détail de la création d'une fonction coût pour le type de problèmes de design auquel on s'intéresse dans ce document.

La littérature mentionne de nombreuses recherches ayant été accomplies dans le but de résoudre le système d'équation 4.1 depuis les années 70. On encourage le lecteur à consulter les travaux de Vanderplaats (1999), Papadrakis, Lagaros, Tsompanakis & Plevris (2001) et Burns (2002) pour une revue plus complète sur ce sujet.

Parmi les avenues les plus explorées, on retrouve les algorithmes à gradient, les AE et les méthodes de types GPS. Les sections 4.1 à 4.3 qui suivent présentent une brève introduction à chacune de ces méthodes, suivies d'un aperçu des avantages et inconvénients de chacune d'entre elles.

4.1 Algorithmes à gradients

Les algorithmes dits « à gradients » sont parmi les algorithmes les plus utilisés actuellement pour résoudre itérativement ce système. On les nomme ainsi parce qu'ils se fient sur l'information contenue dans les dérivées de la fonction coût pour choisir la direction de recherche la plus prometteuse pour le prochain pas.

Les algorithmes à gradients disponibles dans la littérature capables de résoudre un problème d'optimisation sont classés en différentes catégories selon le type de fonction coût et de contraintes dont on dispose. La figure 4.3 illustre la hiérarchie des algorithmes à gradients disponibles dans la littérature.

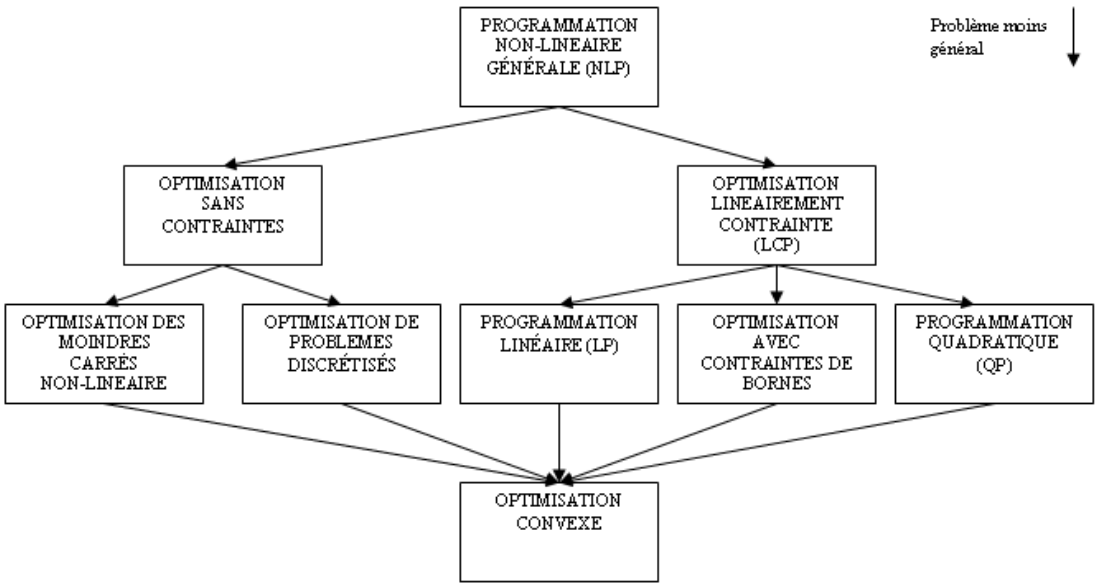


Figure 4.3 : Classification des algorithmes d'optimisation

En pratique, les problèmes d'optimisation structurelle sont formulés sous la forme de problèmes de programmation non linéaire générale (aussi nommée NLP pour Non-Linear Programming) parce qu'ils disposent de contraintes et d'une fonction coût toutes non linéaires (Gould, Orban & Toint, 2004).

Vu son importance en sciences appliquées, ce domaine de recherche a acquis une vaste littérature. Plusieurs auteurs ont tenté d'en synthétiser l'essence dans des livres (Gill, Murray & Wright, 1981 ; Dennis Schnabnel, 1983 ; Fletcher, 1987a ; Bertsekas, 1995 ; Bonnans, Gilbert, Lemarechal & Sagastizabal, 1997 ; More & Toraldo, 1991 ; Nocedal & Wright, 1999), dans des comptes-rendus de conférences (Coleman, 1990 ; Hager, Hearn & Pardalos 1994 ; Spedicato, 1994 ; Leone, Murli, Pardalos & Toraldo, 1998 ; Yuan, 1998 ; Di pillo & Murli 2003), et dans des articles divers (Fletcher, 1987b ; Noceldal, 1992 ; Wright, 1992 ; Boggs & Tolle, 1995 ; Lewis & Overton, 1996 ; Conn, Gould & Toint, 1998 ; Powell, 1998 ; Nash, 2000 ; Marazzi & Nocedal, 2001 ; Todd, 2001 ; Forsgren, Gill & Wright, 2002, Gould, 2003 ; Gould, Orban, & Toint, 2004 ; Gould & Toint, 2004).

Bien que nous n'en discutions pas ici, le lecteur intéressé pourra retrouver l'historique de ces recherches dans Gould & Toint (2004).

L'étude de problèmes moins généraux (et principalement les problèmes de programmation quadratique) reste très pertinente pour le domaine, car ces problèmes moins généraux constituent souvent des sous-problèmes créés par les algorithmes conçus pour résoudre les problèmes de NLP.

Bien qu'une littérature riche et intéressante soit disponible à propos de toutes les classes de problèmes existants, nous nous contenterons ici de décrire rapidement l'état de la science pour les algorithmes à gradients se rapportant aux problèmes NLP de grande envergure, puisque ce type de problème est généralement celui résolu lors de l'optimisation structurelle d'un design de pièce mécanique.

Programmation quadratique séquentielle (SQP)

La méthode de loin la plus utilisée pour résoudre les problèmes NLP est la programmation séquentielle quadratique (SQP).

La SQP consiste globalement à résoudre le problème d'optimisation en résolvant successivement une série de problèmes de programmation quadratique (QP) bien posés.

On pourrait globalement classer la SQP en deux catégories distinctes, soit la programmation quadratique séquentielle avec contraintes d'égalité (SEQP) et la programmation quadratique séquentielle avec contraintes d'inégalité (SIQP).

Les méthodes SEQP utilisent le principe d'ensembles actifs, qui stipule que, au point optimal, certaines contraintes d'inégalité $h(x)$ sont actives ($h(x) = 0$) et les autres sont tout simplement respectées sans contraindre la solution ($h(x) > 0$). Ces algorithmes tentent de prédire l'ensemble de contraintes dites actives et n'utilisent que ces contraintes pour construire un modèle de QP à résoudre.

La SIQP résout le problème non linéaire entier, incluant toutes les contraintes même celles inactives. La matrice Hessienne des sous-problèmes est alors une approximation du Hessien du Lagrangien entier, où toutes les contraintes sont linéarisées.

La discussion faite dans Murray & Wright (1992), Goldsmith (1999) et Gill, Murray & Saunders (2002) semble démontrer que la SIQP est inapte à converger aussi rapidement que la SEQP pour les problèmes de grande envergure. Plus de détails sur ces méthodes peuvent être retrouvés dans Robinson (1974), Hock & Schittkowski (1981), Boggs, Kearsley & Tolle (1999a), Boggs, Kearsley & Tolle (1999b), Fletcher & Leyffer (2002) ainsi que Gould, Orban & Toint, (2004).

Avantages et inconvénients des méthodes à gradient.

Comme on peut le constater dans la discussion précédente, les algorithmes à gradients se fondent fortement sur l'information topographique environnante pour choisir le prochain pas d'optimisation. Ceci leur donne l'avantage de converger généralement plus rapidement que les autres méthodes vers l'optimum local le plus rapproché.

Toutefois, pour des applications industrielles qui possèdent souvent des discontinuités, des endroits non évaluables pour la fonction coût et un nombre considérable d'optimums locaux, leur convergence vers une solution intéressante est très loin d'être garantie si la solution initiale de l'algorithme n'est pas suffisamment rapprochée de la solution recherchée.

De plus, dans ces cas complexes, le gradient de la fonction doit être calculé par différence finie, avec un pas défini par l'utilisateur. Puisque l'utilisateur ne dispose pas toujours de beaucoup d'informations sur la topographie de la fonction coût, il est parfois très difficile de définir ce pas adéquatement pour chaque variable sans de lourds ajustements préalables. Étant donné que la qualité de la solution générée par ces algorithmes dépend fortement de la justesse du calcul du gradient, on peut donc prétendre que leur utilisation est généralement plutôt couteuse pour les problèmes industriels.

4.2 Algorithmes de type « Generalized Pattern Search »

Les méthodes de types GPS constituent un type de méthodes d'optimisations qui n'a pas besoin d'évaluer le gradient de la fonction pour converger.

La structure générale d'un algorithme GPS est la suivante :

1 – Déterminer un ensemble de directions de recherche pour l'évaluation de la fonction.

2 – Évaluer la fonction coût pour un sous-ensemble de directions et de pas de recherche
3 – Si possible, trouver un point où la valeur de la fonction coût est moindre qu'au point actuel.
4 – Si aucun meilleur point n'est trouvé, raffiner la grille et recommencer jusqu'à un certain critère de convergence.

Les algorithmes GPS explorent donc en quelque sorte une grille de points autour de l'optimum actuel. Par exemple, la figure 4.4, tirée de Audet & Orban (2004a), représente une possibilité de raffinement de la grille de recherche après une itération infructueuse (pas de meilleures solutions trouvées).

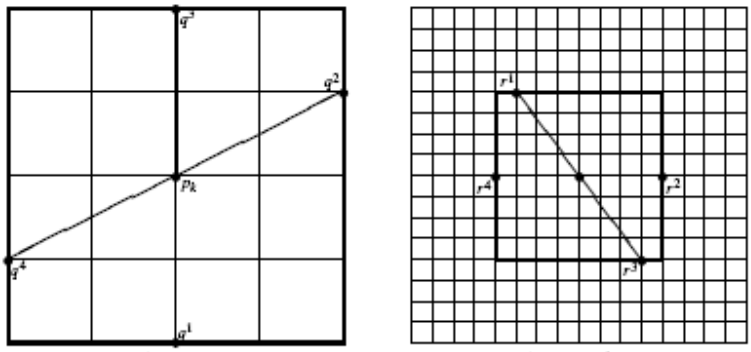


Figure 4.4 : Raffinement de la grille de recherche d'un GPS

Plusieurs variantes de GPS existent dans la littérature (entre autres HJ de Hooke & Jeeves, 1961 et MADS dans Audet & Orban, 2004a et Audet & Orban, 2004b). Les principales différences entre ces méthodes sont la façon de sélectionner le prochain mouvement exploratoire, la méthode pour mettre à jour les directions de recherche et la méthode pour modifier la taille de la grille.

Avantage des algorithmes de type GPS sur les algorithmes à gradient

Pour un problème de grande envergure bruité, multimodal et possiblement discontinu, les GPS disposent de plusieurs avantages par rapport aux algorithmes à gradient.

- *Exploration plus globale*

  Puisqu'ils explorent une surface finie du domaine plutôt que de trouver le premier minimum dans une direction donnée, les GPS sont moins susceptibles que les algorithmes à gradient d'être piégés dans un optimum local causé par le bruit ambiant ou par des variables de moindre importance.

- *Coût moindre*

  Puisqu'ils n'ont pas besoin de calculer le gradient à chaque point de la fonction, les GPS ont tendance à être moins coûteux en temps de calcul qu'un algorithme à gradient pour les topologies complexes.

- *Robustesse*

  Puisqu'ils travaillent sur une grille de solutions plutôt que sur une seule solution particulière, les algorithmes GPS peuvent converger même s'ils rencontrent certains points où la fonction ne peut pas être évaluée.

- *Parallèlisation*

  Puisqu'ils évaluent un certain nombre de points sur la grille simultanément, les GPS se prêtent bien au calcul en parallèle sur plusieurs ordinateurs.

Ainsi, pour le type de problèmes qui nous intéresse, il est fort à parier que les GPS donneront plus souvent de meilleurs résultats que les algorithmes à gradient.

4.3 Algorithmes évolutionnaires (AE)

Une solution de plus en plus populaire aux problèmes d'optimisation est l'utilisation d'algorithmes évolutionnaires. Voici une définition relativement large des algorithmes évolutionnaires :

Les algorithmes évolutionnaires (AE) sont des algorithmes qui utilisent les idées et s'inspirent de l'évolution naturelle pour obtenir une solution à un problème donné (Yao, 2005).

La plupart des algorithmes évolutionnaires actuellement utilisés en ingénierie sont basés sur la démarche suivante (voir figure 4.5) :

1. Générer une population initiale P(0)
2. Évaluer la valeur adaptative (souvent appelée « fitness ») de chaque individu
3. Sélectionner des parents selon la valeur adaptative.
4. Appliquer des opérateurs de recherche aux parents et produire la génération P(i+1)
5. Terminer l'algorithme si la solution satisfait le critère de convergence. Sinon, retour à l'étape 2.

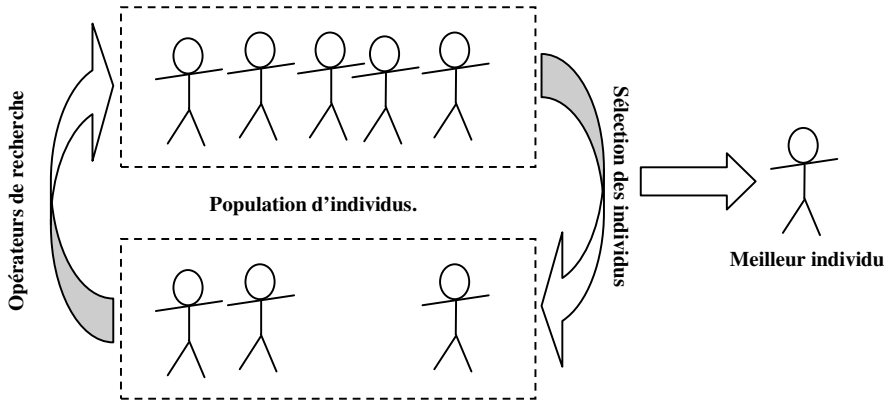


Figure 4.5 : Schéma d'un algorithme évolutionnaire

Les algorithmes évolutionnaires les plus utilisés sont sans contredit les algorithmes génétiques. Vu leur importance pour la présente recherche, ces algorithmes sont décrits plus en détail à l'annexe D de ce document.

Avantages et inconvénients des AE par rapport aux algorithmes à gradient

Si l'on compare le procédé des AE avec celui d'un algorithme d'optimisation à gradient plus conventionnel, on constate peu de différences (Yao, 2005). Le schéma d'un algorithme à gradient est en effet très similaire à celui d'un AE (voir figures 4.5 et 4.6) :

1. Générer une solution initiale
2. Sélectionner la solution la plus récente ou une autre plus acceptable.
3. Appliquer une perturbation (s'inspirant du gradient de la fonction près de la solution actuelle) pour générer une autre solution
4. Terminer si la solution satisfait le critère de convergence, sinon, retour à l'étape 2.

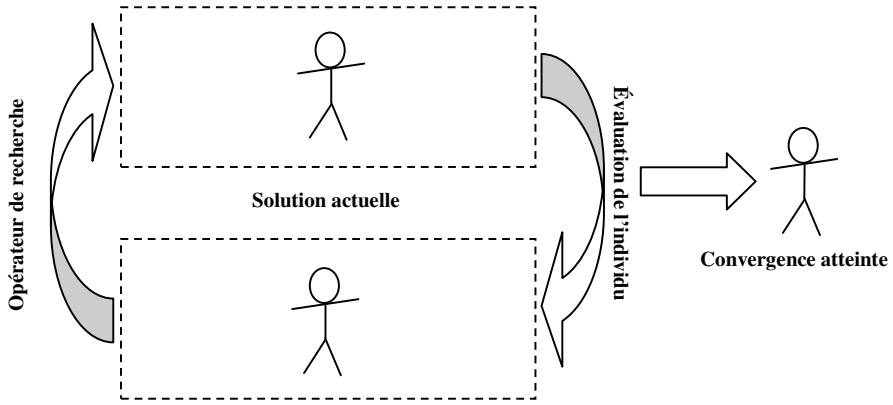


Figure 4.6 : Schéma d'un algorithme d'optimisation à gradient

Quelle est la différence ? L'algorithme évolutionnaire utilise une population, donc plusieurs solutions en parallèles. De plus, puisque les AE s'inspirent de l'évolution naturelle, ses opérateurs de recherche sont généralement davantage aléatoires, ce qui permet d'explorer une plus grande partie du domaine réalisable, au prix d'une vitesse de convergence réduite.

Chance accrue d'éviter les optimums locaux.

Travailler sur plusieurs solutions en parallèle avec des opérateurs aléatoires réduit la possibilité de tomber sur un optimum local (Srinivas & Patnaik, 1994 ; Yao, 2005). Il faut noter, toutefois, que le biais introduit dans les algorithmes génétiques conduit tout de même souvent à une convergence prématurée vers un optimum local (Reed, 1967).

Coût du calcul accru

Si l'on dispose d'une excellente approximation de la solution globale au problème posé, travailler simultanément sur plusieurs autres solutions et utiliser des opérateurs aléatoires ne fait que rallonger considérablement le temps de calcul.

Les algorithmes évolutionnaires sont donc davantage utiles dans les cas suivants :

- On en sait très peu sur la solution attendue
- Le domaine est très bruité, discontinu ou multimodal.
- On cherche une solution nouvelle et différente au problème actuel.

Pour le cas de l'optimisation du design de pièces mécaniques, le domaine sera effectivement discontinu ou du moins multimodal dans la plupart des cas. De plus, la possibilité de générer des solutions éloignées de la solution actuelle est très attrayante pour une application conçue dans le but de suggérer des designs à un ingénieur. Ainsi, pour le problème actuel, nous privilégierons les AE sur les algorithmes à gradient.

Les AE les plus populaires pour les problèmes d'optimisation sont les algorithmes génétiques (AG). Vu la grande importance des AG dans le cadre de cette recherche, ils sont décrits plus en détail à l'annexe D de ce document.

Comparaison entre les AE et les algorithmes GPS.

Pour un problème de grande envergure bruité et multimodal, il n'est pas toujours clair de savoir à l'avance le meilleur algorithme à utiliser entre le GPS et les AE. Si le domaine est discontinu, les AE semblent être une meilleure alternative.

Avantages du GPS par rapport aux AE

- Meilleure convergence

Audet & Dennis (2003) ont prouvé qu'un GPS allait converger vers un point stationnaire sous l'hypothèse d'un domaine continûment différentiable, alors que la convergence des algorithmes évolutionnaires est plus difficile à prouver. Le GPS semble donc plus apte à donner un résultat viable que les AE.

- *Efficacité supérieure*

Puisqu'ils utilisent une méthode de recherche mieux dirigée et évaluant un nombre fini de points, les GPS convergeront généralement plus rapidement que les AE.

Désavantage du GPS par rapport aux AE

- *Créativité moindre*

Wretter & Wright (2003) ont comparé les résultats entre un GPS et un AE pour plusieurs cas tests, et semblent démontrer que vu le caractère aléatoire de certains de leurs opérateurs, les AE disposent d'une meilleure capacité pour trouver un optimum global, surtout si le domaine exploré est discontinu.

4.4 Choix d'un type d'algorithme d'optimisation pour l'algorithme de design RPC

À ce stade, il importe de se souvenir que l'objectif global de ce volet de recherche est d'appliquer, ou de développer et de valider, un algorithme d'optimisation adapté au problème du design de pièces mécaniques.

Ainsi, l'algorithme sera confronté à des problèmes de design structurel en ingénierie, qui sont généralement :

- Discontinus à cause de la gestion de contraintes et de la paramétrisation qui implique souvent plusieurs variables discrètes (par exemple les variables impliquant le choix des matériaux ou de la topologie du modèle).
- Multimodaux à cause du nombre élevé de variables.
- Hautement non linéaires à cause des principes physiques analysés.

Et puisque l'algorithme sera utilisé dans le but de fournir une aide aux designers, ses objectifs seront doubles. Selon le cas, le designer voudra :

- Raffiner la valeur des paramètres d'une solution connue afin d'améliorer ses

performances.

- Explorer le domaine dans le but de trouver des solutions inconnues au problème.

Le second des deux objectifs énumérés ci-dessus convient certes mieux à un AE, alors que le premier pourrait convenir à un algorithme à gradient, à un algorithme GPS ou un AE, selon la nature de la fonction coût ainsi que la distance entre la solution connue et la solution finale.

Enfin, d'un point de vue industriel, pour le design conceptuel d'une pièce, on désire un algorithme qui :

- Sera utilisable sans nécessiter beaucoup d'ajustements préalables.
- Sera stable en cours d'utilisation.
- Est adapté au calcul sur plusieurs ordinateurs en parallèle

Ces considérations, surtout celles de la stabilité et du calcul en parallèle, écartent le choix de l'algorithme à gradient pour le cas considéré. Le choix entre AE et GPS est plus difficile à réaliser.

D'un côté, l'aspect aléatoire des AE est plutôt attrayant. Plusieurs lancées successives d'un AE ne donneront pas nécessairement le même résultat, et permettront à l'algorithme d'infiltrer des zones d'apparence non prometteuses possiblement abandonnées par le GPS. Avec un AE, l'algorithme procurera donc une série possible de solutions au designer si celui-ci dispose du temps nécessaire.

Par contre, le GPS, qui explore d'abord le domaine globalement, puis raffine petit à petit son exploration, est plus efficace dans sa recherche et sera capable de trouver plus rapidement ces nouveaux filons de design.

Ce concept de raffinement est d'ailleurs à la base de la plupart des algorithmes hybrides existants dans la littérature (Renders & Flasse 1996, Chambers, Lehman & Dowla 2007) : on désire d'abord trouver des zones prometteuses, puis les exploiter plus en détail (voir figure 4.7). Il serait donc aussi possible d'utiliser un algorithme hybride AE – GPS ou AE – gradient pour explorer le domaine puis raffiner la solution obtenue selon les besoins de l'utilisateur. Toutefois, ces algorithmes ont tendances à être plus complexes d'utilisation pour l'usager. En effet, en plus d'avoir à ajuster les paramètres de chacun des algorithmes composant l'hybride, le concepteur doit décider jusqu'à quel point il laisse l'AE explorer et à partir de quand le problème est considéré comme un raffinement local.

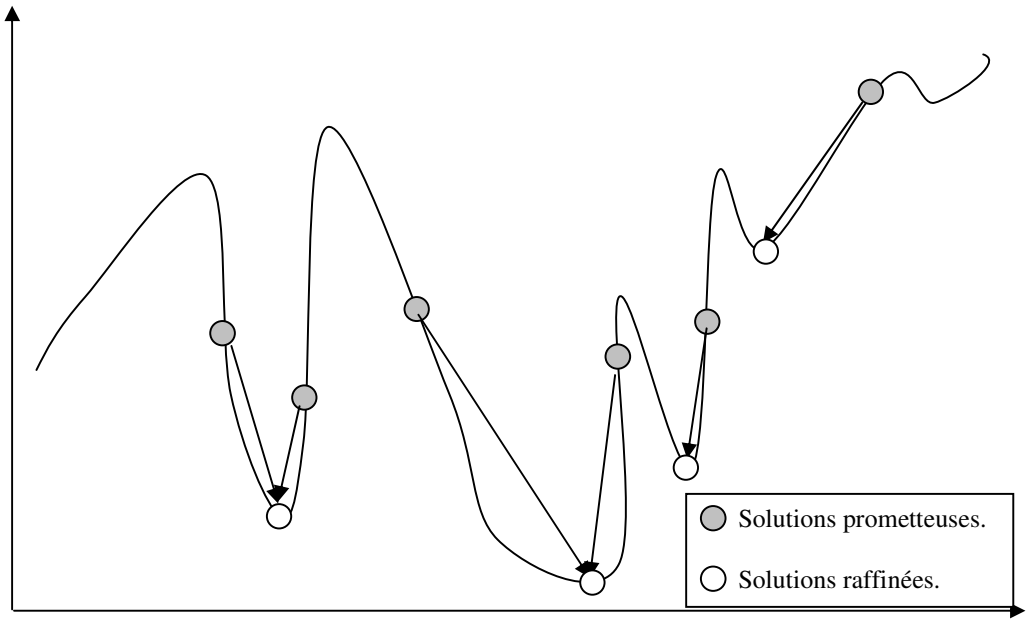


Figure 4.7 : Concept de raffinement des algorithmes hybrides

Ainsi, si on résume la discussion précédente, on désirerait idéalement avoir un algorithme qui réponde aux trois critères suivants :

1. Il doit posséder certains opérateurs aléatoires afin de pouvoir explorer graduellement de nouvelles portions du domaine si on lui en laisse le temps ou si on le lance à nouveau (comme dans le cas d'un AE).
2. Il doit explorer d'abord le domaine globalement avant de raffiner progressivement son pas d'exploration (comme dans le cas d'un GPS).
3. Il serait aussi souhaitable que l'algorithme dispose d'opérateurs de recherche qui utilisent leur connaissance de la topologie de la fonction (comme dans le cas des algorithmes à gradient) afin de guider l'exploration et de permettre une accélération de la convergence lorsque l'on s'approche de l'optimum.

L'algorithme choisi pour répondre au critère 1 est un algorithme évolutionnaire de type génétique. Toutefois, puisque les algorithmes génétiques existants ne répondent pas convenablement aux critères 2 et 3, un nouvel algorithme a été développé et validé dans le cadre de cette recherche. Cet algorithme se nomme l'algorithme génétique à évolution de noyaux territoriaux (AGENT), et sera décrit plus en détail au chapitre 5.

Comme son nom l'indique, il s'agit d'un algorithme génétique, et s'inspire fortement de plusieurs algorithmes génétiques tirés de la littérature. L'annexe D décrit plus en détail les algorithmes génétiques, et plus particulièrement ceux ayant inspiré et ayant contribué à valider les concepts de l'AGENT.

Chapitre 5.
ALGORITHME GÉNÉTIQUE À ÉVOLUTION DE NOYAUX TERRITORIAUX (AGENT)

Ce chapitre décrit l'algorithme développé lors de cette recherche, ainsi que les principales heuristiques originales qu'il utilise, soit les noyaux territoriaux et l'opérateur de substitution.

La création de l'AGENT s'inspire des trois critères de développement mentionnés à la section 4.4 :

1. L'algorithme doit posséder certains opérateurs aléatoires afin de pouvoir explorer graduellement de nouvelles portions du domaine si on lui en laisse le temps ou si on le lance à nouveau (comme dans le cas d'un AE).
2. Il doit explorer d'abord le domaine globalement avant de raffiner progressivement son pas d'exploration (comme dans le cas d'un GPS).
3. Il serait aussi souhaitable que l'algorithme dispose d'opérateurs de recherche qui utilisent leur connaissance de la topologie de la fonction (comme dans le cas des algorithmes à gradient) afin de guider l'exploration et de permettre une accélération de la convergence lorsque l'on s'approche de l'optimum.

De plus, l'AGENT a été conçu dans une optique d'automatisation de processus, c'est-à-dire qu'il tente de réduire autant que faire se peut l'apport de l'usager à l'algorithme.

L'AGENT est un AG utilisant des opérateurs de croisement et de mutation semblables à ceux que l'on retrouve dans la littérature pour un encodage réel (voir section D.3), répondant ainsi au critère 1 défini ci-dessus. Une heuristique nommée « Noyaux territoriaux » s'inspirant des grilles de raffinement des GPS lui a été ajoutée afin de répondre aussi au critère 2 (voir section 5.1). Finalement, afin d'accélérer sa

convergence, il fera aussi usage d'un opérateur de recherche nommé *opérateur de substitution* (voir section 5.4.3) qui utilisera un réseau de neurones faisant office de modèle substitut à la fonction coût. Ce modèle lui permettra de tirer avantage des connaissances acquises sur la fonction coût afin de mieux choisir les prochaines évaluations, répondant ainsi aussi au critère numéro 3.

Cet algorithme a été conçu dans l'optique de produire un résultat robuste peu importe le type de fonction coût, potentiellement calculable en parallèle, nécessitant peu d'ajustements mais permettant à l'utilisateur de définir aisément ses priorités entre l'exploration du domaine et le raffinement local. L'AGENT est plutôt lourd de calcul et d'implémentation pour un AG, et convient par conséquent davantage aux problèmes pour lesquels la résolution de la fonction coût est plutôt coûteuse pour l'usager. De plus, étant d'abord et avant tout un algorithme d'exploration globale disposant des caractéristiques des AE, l'AGENT n'est pas aussi apte à raffiner une solution autour d'un optimum que ne le serait par exemple un algorithme à gradient régulier. Par contre, les tests effectués dans le cadre de cette recherche semblent démontrer qu'il s'agit d'un algorithme potentiellement très compétitif dans les cas d'optimisation avec contraintes (voir section 6.2), ou d'optimisation de problèmes dont la topologie se rapproche de celle des problèmes d'optimisation de pièces mécaniques (voir section 6.1 et chapitre 7)

5.1 Principes généraux des noyaux territoriaux

L'analogie évolutive des AG veut que l'on compare les valeurs des paramètres des solutions possibles à la génétique d'autant d'individus. Les individus ayant les gènes les plus adaptés survivent et sont aptes à former la génération suivante.

Dans cette optique, on peut concevoir les noyaux territoriaux comme un autre critère de sélection naturelle pour les individus. En effet, si trop d'individus en développement occupent le même territoire, même s'ils possèdent des gènes potentiellement très

adaptés, il y a une bonne chance que plusieurs de ces individus meurent de faim ou soient expulsés par d'autres individus déjà matures avant d'avoir pu exprimer leur plein potentiel.

Ainsi, si l'on considère le domaine de la fonction coût comme un énorme territoire possible pour une population, on peut concevoir que les individus auront plutôt tendance à se répartir le territoire équitablement avant d'être forcés de cohabiter sur les mêmes zones. Ce constat est à la source de l'heuristique des noyaux territoriaux que nous présentons ici.

Supposons que l'on assigne à chaque individu de la population totale **Ptot** un territoire de chasse minimal ayant un rayon δ. On assume que l'individu réside au milieu de son territoire et qu'il ne tolèrera aucun intrus dans ce rayon minimal. Aussi, tout individu en développement pénétrant dans cette zone se verra détruit avant d'avoir pu exprimer son plein potentiel (c'est-à-dire avant que sa valeur adaptative ne soit évaluée). Selon la distance euclidienne, un individu défini par le vecteur de paramètres $\vec{x}$ sera dans le territoire de l'un ou l'autre des individus *i* de **Ptot** s'il ne respecte pas l'équation 5.1 :

$$\sum_{j=1}^{N}\left(x_j - Ptot_{i,j}\right)^2 \geq \delta^2 \quad \text{pour } i \in \{1,2,...,Ntot\} \tag{5.1}$$

où *Ntot* est le nombre total d'individus de la population **Ptot**

N est le nombre de paramètres composant une solution

Dans un algorithme à noyaux territoriaux, aucun individu n'ayant été évalué n'est écarté de la population totale **Ptot**, de telle sorte que celle-ci est conservée en mémoire et s'accroît généralement à chaque génération. Toutefois, comme dans les autres AG, seule une portion de taille fixe de cette population participera à la reproduction. Cette population sera nommée **P**, la population active. La taille de la population **P** est une

constante N_{pop} définie par l'utilisateur. Dans l'AGENT, **P** sera constituée uniquement des N_{pop} meilleurs individus de **Ptot**, soit ceux ayant la plus haute valeur adaptative.

À mesure que les divers territoires possibles se peuplent, la compétition se fait plus forte, et le rayon δ que les individus sont aptes à défendre diminue (le choix d'un rayon δ approprié sera discuté plus en détail à la section 5.2). Ainsi, certains territoires en bordure du territoire des individus deviennent disponibles pour les nouveaux arrivants. La figure 5.1 illustre ce phénomène pour une population totale passant de 2 à 6 individus. Notez que les points foncés sur la figure représentent les nouveaux individus évalués alors que les points blancs représentent **Ptot**.

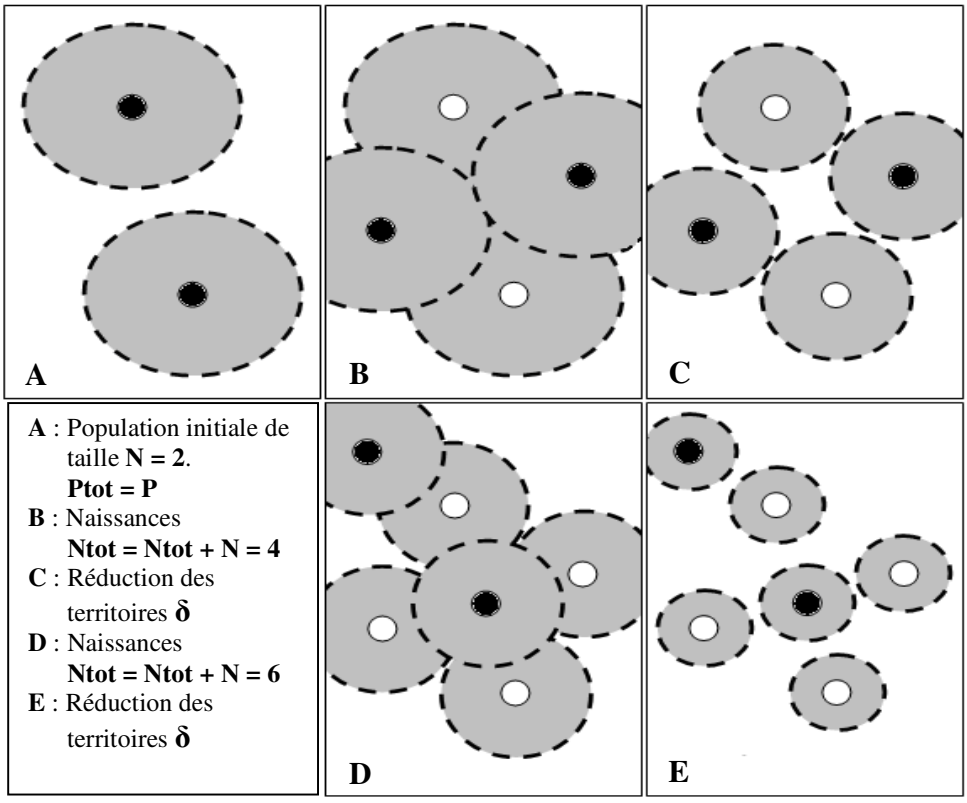


Figure 5.1 : Évolution d'une population de N_{pop} = 2 individus

Aussi, il faut savoir que rien n'empêche les nouveaux individus d'apparaître dans le territoire d'un individu existant. Les noyaux territoriaux stipulent que cet individu ne doit pas être évalué, pas qu'il ne peut pas être généré par les opérateurs de recherche des AG (voir figure 5.2).

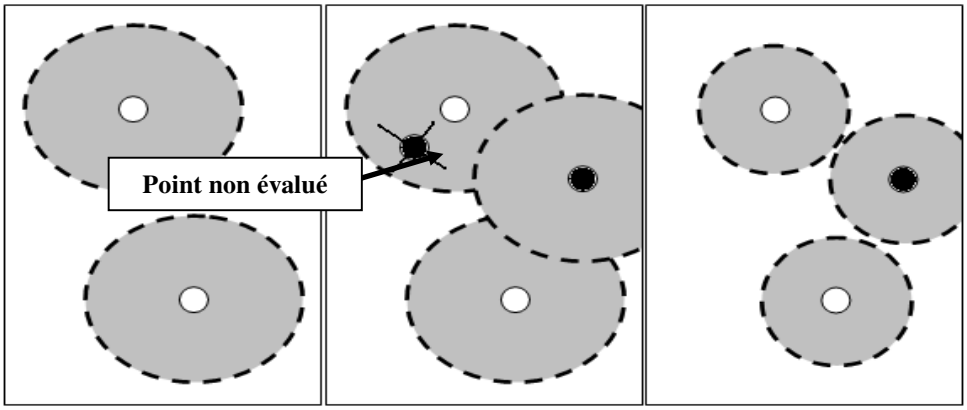


Figure 5.2 : Destruction d'un intrus sous-développé dans le territoire.

En bref, les noyaux territoriaux sont une heuristique imitant en quelque sorte le mode de raffinement des algorithmes de type GPS pour sélectionner la longueur des pas d'un algorithme évolutionnaire. L'algorithme est effectivement incapable d'explorer les environs d'une solution actuelle tant qu'il n'a pas exploré suffisamment les territoires environnants cette solution.

À première vue, il pourrait sembler que cette pratique est susceptible de ralentir la convergence de l'algorithme en ne lui permettant pas d'explorer de façon approfondie les régions les plus prometteuses. Ceci est en partie vrai en fin d'algorithme, lorsque l'algorithme devrait tenter davantage de raffiner une solution près d'un optimum que d'en trouver des nouvelles. La section 5.5.3 explique comment un réseau de neurones artificiels a été employé pour alléger ce problème.

Toutefois, le reste du temps, c'est plutôt l'inverse qui se produit : cette façon d'opérer empêche l'algorithme de gaspiller certaines évaluations à raffiner certaines solutions

encore immatures en choisissant des points trop près les uns des autres, procurant à l'algorithme de meilleures capacités de recherche globale.

5.2 Choix de la distance δ

Jusqu'à maintenant, nous avons vu les concepts clés des noyaux territoriaux, à savoir que chaque individu évalué devient lui-même le noyau d'un territoire qui est interdit aux autres individus émergents. Le rayon du territoire δ agit donc comme un critère ajustant le pas possible pour l'AG. Le choix d'une valeur adéquate de δ est par conséquent cruciale pour le bon fonctionnement de l'algorithme.

Intuitivement, on comprend qu'un bon choix de δ devrait s'appuyer sur trois facteurs :

1. *Ntot* (Le nombre total de solutions évaluées jusqu'à maintenant)

2. Le degré de créativité désiré par l'utilisateur

3. L'état de convergence de l'algorithme

Le premier de ces facteurs provient du fait que plus un espace de design est exploré, plus les territoires interdits recouvrent d'espace et plus on a de chances d'être déjà à proximité d'un optimum intéressant. Ainsi, à mesure que *Ntot* augmente, on devrait diminuer δ pour permettre à l'algorithme de raffiner le design et ouvrir de nouveaux espaces aux nouveaux individus.

Le second facteur implique les besoins précis de l'utilisateur. En effet, selon les ressources dont il dispose, il pourra vouloir explorer plus complètement le domaine ou alors permettre à l'algorithme de converger plus rapidement, amener un choix respectivement plus élevé ou plus bas de δ.

La méthode qui suit suggère une solution mathématique pouvant convenir aux deux premiers facteurs énoncés. Le troisième facteur, moins trivial, sera traité un peu plus loin.

Si toutes les N variables sont adimensionnelles (variant de $x_{i,min}$ = 0 à $x_{i,max}$ =1), l'hyperespace de design aura un domaine dont l'extension (le volume à N dimensions) sera la suivante :

$$E_d = \prod_{i=1}^{N} \left(x_{i,\max} - x_{i,\min} \right) = 1 \qquad (5.2)$$

Ensuite, si l'on pose les hypothèses que les territoires ne se recoupent pas et ne dépassent pas des bornes 0 et 1 des variables de $\vec{x}$, on pourrait démontrer que l'extension E_{tot} occupée par les hypersphères des territoires autour de chacun des noyaux est la suivante :

$$E_{tot} = k_1 \cdot Ntot \cdot \delta^N \qquad (5.3)$$

où

$$k_1 = \frac{\pi^{\frac{N}{2}}}{(N/2)!} \qquad \text{si N est pair}$$

$$k_1 = \frac{2^{2k_2} (k_2!) \pi^{\frac{N-1}{2}}}{2k_2!} \qquad \text{si N est impair}$$

$$k_2 = \frac{N}{2} + \frac{1}{2}$$

Sachant qu'en réalité, il est fort possible que les territoires de l'algorithme se recoupent ou dépassent les bornes des variables, ce volume devient l'espace maximal théorique possiblement occupé par les territoires pour une population de taille *Ntot*.

Définissons maintenant une nouvelle constante adimensionnelle que l'on appellera γ_c^0 : la créativité de l'algorithme.

$$\gamma_c^0 = \frac{E_{tot}}{E_d} \tag{5.4}$$

γ_c^0 est alors une valeur mesurant la proportion maximale d'espace possiblement occupée par les territoires autour des individus de **Ptot**. En théorie, plus cette proportion est grande, plus l'algorithme devra produire des individus éloignés des individus actuels afin que ceux-ci puissent être évalués. Par contre, si cette valeur excède 1, il existe une chance théorique pour que l'espace entier soit interdit et que l'algorithme ne puisse plus choisir la moindre valeur acceptable.

Afin que l'usager puisse définir le niveau de créativité voulu pour l'algorithme, γ_c^0 deviendra l'un des paramètres à définir pour l'algorithme, par une valeur se situant théoriquement entre 0 et 1, servant à calculer le rayon initial δ_0 des territoires.

Si on combine les équations 5.2, 5.3 et 5.4 et que l'on isole δ, on obtient la relation 5.5 décrivant le rayon que l'on utilisera comme rayon initial pour générer le territoire autour de chaque noyau territorial.

$$\delta_0 = \left(\frac{\gamma_c^0}{k_1 Ntot} \right)^{1/N} \tag{5.5}$$

Le choix de γ_c^0 suggéré par défaut est de 0,2. Nous reviendrons plus tard sur l'influence du choix de γ_c^0 dans l'AGENT.

Notez que si une population utilise plusieurs modèles paramétriques, par exemple parce qu'une solution prometteuse a immigré d'un îlot utilisant un autre modèle, le nombre de

paramètres N peut varier d'un individu à l'autre. Un tel cas ne devrait pas se présenter ni souvent, ni longtemps en pratique, car le meilleur modèle détruira éventuellement tous ses compétiteurs. Dans un tel cas, de toute façon, l'algorithme n'utilisera pas les noyaux territoriaux autour des solutions utilisant un modèle pour détruire une solution issue d'un autre. La distance δ_0 calculée variera alors simplement selon le modèle.

5.2.1 Estimation de l'état de convergence de l'algorithme

Le troisième facteur influant sur le choix de δ, un peu moins intuitif, va un peu plus loin que faire varier uniquement δ selon *Ntot*. En effet, si la population est en convergence vers un point précis, il est possible que le domaine environnant soit très densément occupé, et donc que la totalité des nouveaux individus d'une génération tombent dans une zone interdite. Dans un tel cas, aucun d'entre eux ne sera évalué, *Ntot* n'augmentera pas et l'algorithme stagnera sur place. Ainsi, lorsque l'algorithme semble vouloir converger, il est important de réduire δ afin de le laisser raffiner ses solutions. À l'inverse, si on sent que l'algorithme est toujours en exploration, on peut vouloir augmenter δ pour lui permettre de progresser plus rapidement vers une autre destination.

Le critère $F_{\max} - \overline{F}$, suggéré par Srinivas & Patnaik (1994) pour évaluer la convergence (voir section D.6.3) d'un AG, est intéressant pour quantifier cette convergence aux environs d'un optimum ou pour évaluer la qualité actuelle de la solution d'un AG. Par contre, puisqu'il s'appuie uniquement sur la valeur adaptative des solutions et non sur la valeur des paramètres, il est possible, dans un environnement multimodal, que ce critère annonce une convergence accrue même lorsque l'on découvre la présence d'un nouvel optimum.

La figure 5.3 illustre une situation où ce critère annoncera une estimation erronée de la convergence. Supposons qu'entre deux itérations, la population $\mathbf{P_1}$ évolue et devient la population $\mathbf{P_2}$. Sur l'exemple de cette figure, il est possible de constater que $F_{\max} - \overline{F}$

est plus petit pour la population $\mathbf{P_2}$ que pour la population $\mathbf{P_1}$ (la distance verticale entre les points I_2, I_3 et I_4 constituant $\mathbf{P_2}$ est en moyenne plus petite que la distance verticale entre les individus I_1, I_2 et I_3 constituant $\mathbf{P_1}$). Selon le critère, ceci signifierait une population qui converge vers un optimum. Pourtant, la découverte du point I_4, qui est très éloigné selon l'axe **x** des autres individus de la population, indique au contraire qu'une exploration globale est davantage de mise que le raffinement de la solution près des individus de $\mathbf{P_1}$. Intuitivement, on comprend donc que la distance entre les variables (selon l'axe **x**) d'une population peut être davantage éloquente que la différence entre leur valeur adaptative (selon l'axe **F**) si l'on s'intéresse à savoir s'il existe d'autres potentiels optimums intéressants ou non.

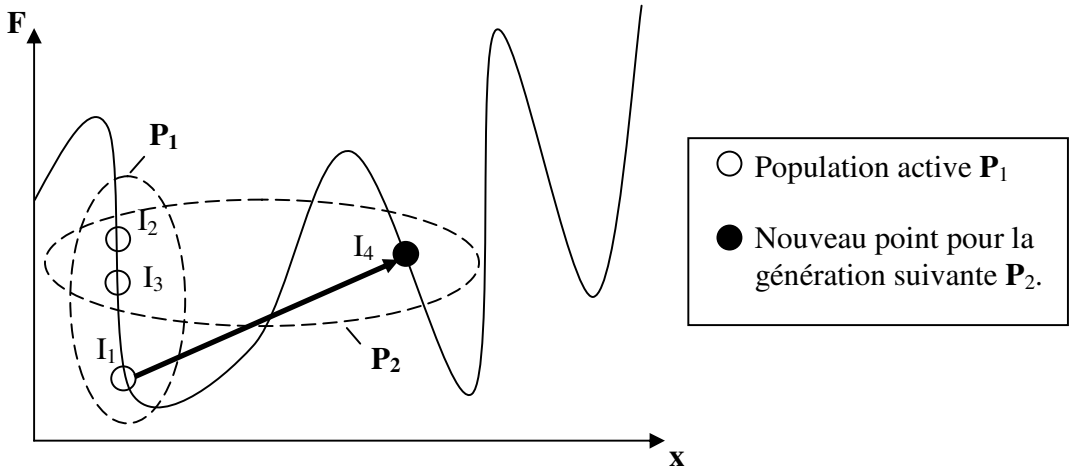


Figure 5.3 : Estimation erronée de la convergence de $\mathbf{P_2}$ par l'estimateur suggéré.

Puisque l'on s'intéresse à la convergence dans l'optique de déterminer si une recherche globale est encore nécessaire ou si l'on doit raffiner les solutions actuelles, il est donc intéressant d'utiliser plutôt un outil qui dépend de la distance entre les points de la population pour quantifier cette convergence, soit la distance totale cumulée entre tous les paramètres de tous les individus de la population :

$$D^g = \sqrt{\sum_{i=1}^{N_{pop}} \sum_{k=1}^{N_{pop}} \sum_{j=1}^{N} \left(P^g{}_{ij} - P^g{}_{kj} \right)^2} \qquad (5.6)$$

où D^g est la distance totale cumulée entre les paramètres d'une population à la génération g.

$P^g{}_{ij}$ est le paramètre *j* de l'individu *i* de la population active à la génération g.

La relation 5.6 n'est évidemment valide que si deux individus appartiennent au même modèle paramétrique. Si un certain $P^g{}_{ij}$ ne possède pas les mêmes paramètres que $P^g{}_{kj}$, la distance entre les deux individus devra être posée maximale pour que l'algorithme converge convenablement. Puisque les paramètres varient de 0 à 1, on posera alors $\sum_{j=1}^{N} \left(P_{ij}^g - P_{kj}^g \right)^2 = N$, où N sera le nombre de paramètres du modèle ayant le plus de paramètres entre le modèle de $P^g{}_{ij}$ et celui de $P^g{}_{kj}$.

En principe, à l'optimum, $D^g = 0$. Ensuite, plus les points sont rapprochés les uns des autres, plus D^g sera petit. Puisque l'on ne connaît pas réellement la topologie exacte des fonctions que l'on désire étudier, l'ordre de grandeur de cette valeur ne veut toutefois rien dire en soi. En effet, ce que l'on désire observer d'abord et avant tout, c'est l'évolution de D^g entre deux générations, soit :

$$\Delta D^g = D^g - D^{g-1} \qquad (5.7)$$

5.2.2 Utiliser ΔD^g pour choisir δ_g

δ_g est le rayon autour des noyaux territoriaux à la génération *g*.

Dans le contexte de l'évolution autour de noyaux territoriaux, si ΔD^g est positif pour un *g* donné, on comprend que l'algorithme a découvert certains points intéressants plus éloignés de l'optimum actuel, et que sa recherche globale n'est donc pas terminée (comme dans l'exemple de la figure 5.3). Il est donc suggéré alors de poser un δ_g plus grand que δ_{g-1} pour favoriser l'exploration globale. Par contre, afin que les territoires interdits n'en viennent pas à occuper une portion trop importante du domaine de design, on limite δ_g à la taille maximale δ_0.

Si $\Delta D^g = 0$, cela veut généralement signifier que la population $\mathbf{P^g}$ est exactement la même que la population $\mathbf{P^{g-1}}$, et donc que les opérateurs de recherche actuels ont été incapables de produire le moindre individu acceptable pour $\mathbf{P^g}$. Ceci peut avoir plusieurs causes. D'abord, puisque l'AGENT ne remplace un individu de sa population active que si un meilleur individu est découvert, cela peut vouloir dire que l'on se situe déjà très près de l'optimum recherché. Ensuite, cela peut aussi vouloir dire que la plupart des individus générés pour la génération *g* sont tombés dans un territoire interdit et ont été exterminés avant évaluation. Finalement, puisque les opérateurs de recherche sont aléatoires, cela peut aussi tout simplement être de la malchance, particulièrement si la taille de la population est trop faible pour la complexité du problème. Dans un cas comme dans l'autre, afin d'assurer la convergence de l'algorithme, il importe ici de réduire le rayon des territoires δ_g pour permettre à l'algorithme de découvrir de nouvelles solutions.

Enfin, si $\Delta D^g < 0$, la distance totale entre les points de l'algorithme diminue, ce qui veut dire que la recherche locale a réussi à produire un nouveau point intéressant. Ainsi, on

peut considérer que le pas δ_{g-1} était approprié pour la population actuelle, et δ_g devrait conserver approximativement le même ordre de grandeur.

Conséquemment, à la lumière de la discussion précédente, l'équation 5.8 est suggérée pour le choix d'un rayon territorial δ_g.

$$\delta_g = \left(\frac{\gamma_c^g}{k_1 Ntot} \right)^{1/N} \tag{5.8}$$

Avec

$$\gamma_c^g = \frac{\gamma_c^{g-1}}{\gamma_c^0} \quad \text{si} \quad \left(\Delta D^g > 0\right) \cap \left(\frac{\gamma_c^{g-1}}{\gamma_c^0} < \gamma_c^0 \right)$$

$$\gamma_c^g = \gamma_c^0 \quad \text{si} \quad \left(\Delta D^g > 0\right) \cap \left(\frac{\gamma_c^{g-1}}{\gamma_c^0} \geq \gamma_c^0 \right)$$

$$\gamma_c^g = \gamma_c^{g-1} \gamma_c^0 \quad \text{si} \quad \Delta D^g = 0$$

$$\gamma_c^g = \gamma_c^{g-1} \quad \text{si} \quad \Delta D^g < 0$$

De cette façon, la créativité initiale γ_c^0 définie par l'utilisateur (entre 0 et 1) sert à la fois à déterminer le territoire initial des noyaux territoriaux et la rapidité avec laquelle ces territoires évoluent. On peut donc saisir l'intuition que plus γ_c^0 est faible, plus l'algorithme convergera rapidement vers un optimum sur son chemin, alors que si γ_c^0 est grand, l'algorithme raffine beaucoup moins vite sa grille et devient un meilleur explorateur global. L'utilisateur sera donc libre de spécifier à sa guise la créativité voulue pour l'AGENT selon les ressources dont il dispose.

Afin de ne pas fournir d'informations erronées à l'algorithme, on posera D^0 égal à une constante plus grande que $N_{pop}{}^2 N$, de façon à ce que $D^1 < D^0$ et donc que $\gamma_c^1 = \gamma_c^0$. Par conséquent, un faux état de convergence n'affectera pas le calcul de δ^1 lors de la première itération.

5.3 Schéma général d'un AGENT

L'AGENT est un algorithme qui s'inspire un peu du fonctionnement de l'AGED (voir section D.4) en ce qui a trait à la gestion de sa population, c'est-à-dire qu'il sélectionne un groupe d'individus pour constituer sa population active, et ne les remplace que s'il découvre une meilleure solution.

Pour cette raison, l'AGENT est garanti de toujours converger vers un point au moins aussi bon que le point initial, ce qui évite d'avoir recours à des opérateurs tels que l'élitisme ou de se soucier de l'effet destructeur de certains opérateurs des AG réguliers. La figure 5.4 illustre le schéma général du processus de l'AGENT.

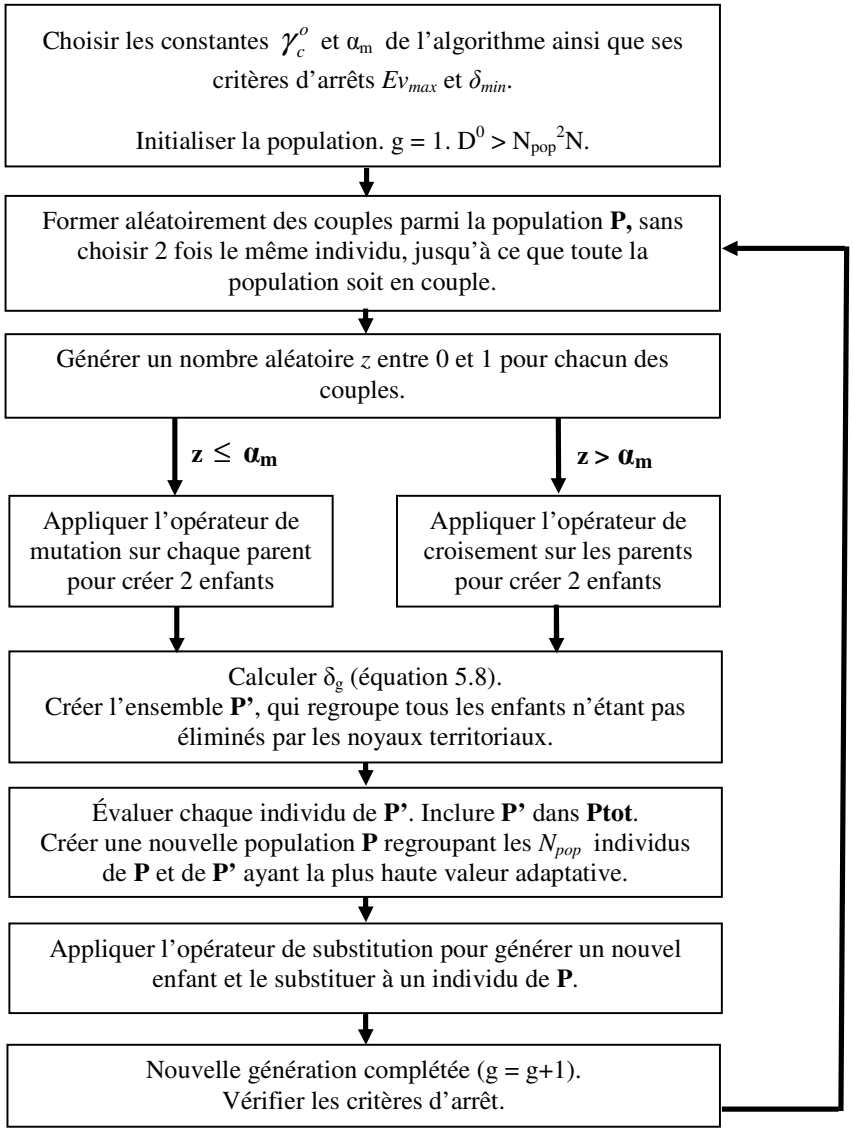


Figure 5.4 : Schéma général de l'algorithme AGENT.

Les sections qui suivent présentent plus en détail chacune des étapes de ce procédé.

5.4 Initialisation de la population de l'AGENT

L'AGENT initialise sa population grâce à la méthode décrite précédemment à la section 3.4.3 de ce document. En bref, l'un des individus de chacune de ses populations (il y

aura en effet plus d'une population si l'algorithme est insulaire) correspondra exactement à une solution donnée par l'utilisateur. Tous les autres individus de chaque population seront alors générés en appliquant l'opérateur de mutation de l'AGENT sur la solution initiale connue pour cette population (équation 3.8).

À ce stade, les noyaux territoriaux ne devraient pas encore être activés : c'est-à-dire que tous les individus seront considérés viables et évalués, même s'ils se situent dans le territoire d'un autre. Toutefois, puisque l'AGENT pourra alors parfois utiliser un pas de mutation exactement égal à δ_0 (voir section 5.5.1), il est important de s'assurer que chacun des individus générés est différent des autres individus générés avant lui. Dans le cas contraire, il faudra lui appliquer à nouveau l'opérateur de mutation jusqu'à ce qu'il diffère des autres individus. Cette exception assure une certaine diversité initiale favorable à l'exploration de l'algorithme, surtout dans le cas des problèmes ayant très peu de variables et une population élevée.

5.5 Opérateurs de recherche pour l'AGENT

Les opérateurs de recherche de l'AGENT sont la mutation, le croisement et la substitution. Le fonctionnement des deux premiers opérateurs est très similaire à celui des opérateurs d'un AG régulier, mais il est tout de même important d'y inclure quelques différences pour tenir compte de la présence des noyaux territoriaux.

D'abord, on ne désire pas d'enfants qui soient de copies conformes de leurs parents. En effet, de tels enfants se retrouveraient automatiquement dans le territoire de leurs parents et seraient aussitôt éliminés. Il est donc souhaitable de s'assurer d'appliquer l'un ou l'autre des opérateurs de recherche. Ainsi, le seul taux d'application des opérateurs qui sera défini est α_m, le taux de mutation, entre 0 et 1.

Avant d'appliquer les opérateurs de recherche, l'algorithme génère une variable aléatoire z entre 0 et 1. Si $z \leq \alpha_m$, les deux enfants seront créés par mutation respective des deux parents. Sinon, les deux enfants seront issus du croisement de leurs parents.

Finalement, la substitution est un opérateur de recherche particulier créé spécialement pour l'AGENT. Cet opérateur utilise un modèle substitut à la fonction F($\vec{x}$) pour remplacer un individu de **P** à chaque génération. La substitution a pour objectif de permettre une interpolation locale de la fonction coût afin de tenter de rapprocher l'algorithme de l'optimum réel, malgré l'interdiction des noyaux territoriaux.

5.5.1 Opérateur de mutation

L'opérateur de mutation sera principalement utilisé avec l'objectif de participer à la phase de recherche globale et de permettre à l'algorithme d'esquiver les optimums locaux.

L'AGENT utilise pratiquement le même opérateur de mutation, qu'un AGR (voir section D.3.1). Par contre, plutôt que d'être une valeur fixe définie par l'utilisateur, le pas moyen de mutation p_{moyen} sera défini en proportion du rayon des territoires δ_g selon l'équation 5.9. Ainsi, la taille des territoires des noyaux territoriaux devient un indicateur de l'ordre de grandeur du pas des mutations de l'algorithme.

$$p_{moyen} = \delta_g \tag{5.9}$$

Aussi, pour un problème de design structurel, il est souvent difficile de prévoir les interrelations existantes entre les diverses variables de la fonction. Une mutation régulière utilisant une perturbation gaussienne a beaucoup de chances de produire un résultat faisant varier toutes les variables à la fois, et il s'avère donc plus difficile de

raffiner la solution d'un problème où les variables sont indépendantes à l'aide de ces perturbations.

Afin de convenir à un plus grand nombre de problèmes, l'opérateur de mutation de l'AGENT utilisera donc le même principe que l'opérateur de recherche d'un AGED (voir D.4), à savoir que chaque variable aura une certaine probabilité p_c d'être modifiée chaque fois. La valeur optimale de cette probabilité p_c dépend beaucoup de la topologie de la fonction et ne peut pas réellement être connue intuitivement à prime abord. Ursem & Vadstrup (2003) énoncent que la valeur $p_c = 0,2$ semble convenir à une grande variété de problèmes dans l'AGED. Ainsi, afin de couvrir le maximum de situations, chacune des variables *i* du parent $\vec{P}_1$ aura une probabilité de subir une mutation Gaussienne selon l'équation 5.10.

$$\begin{aligned} C_{1i} &= P_{1i} + \delta_g \sqrt{2} Erf^{-1}(2z_1 - 1) \quad \text{si} \quad z_2 \le 0,2 \\ C_{1i} &= P_{1i} \quad \text{autrement} \end{aligned} \tag{5.10}$$

où C_{1i} est le $i^{\text{ème}}$ paramètre du nouvel individu $\mathbf{C_1}$ émergeant de la mutation

z_1 et z_2 sont deux nombres aléatoires réels, différents pour chaque variable *i*, compris entre 0 et 1.

Une fois ces mutations effectuées, il importe ensuite de s'assurer que la solution ait au moins quitté le territoire du parent duquel elle est issue. Pour ce faire, on doit vérifier que la distance entre $\vec{P}_1$ et $\vec{C}_1$ satisfait l'équation 5.11.

$$\sqrt{\sum_{i=1}^{N_{pop}} (C_{1i} - P_{1i})^2} \geq \delta_g \qquad (5.11)$$

Si cette équation n'est pas satisfaite, l'action suivante devra être effectuée par l'algorithme :

$$C_{1z_3} = P_{1z_3} + z_4 \delta_g \qquad (5.12)$$

où z_3 est une variable aléatoire entière comprise entre 1 et *N*.

z_4 est soit une variable aléatoire qui est égale soit à 1, soit à -1.

Ainsi, on s'assure que l'individu quitte le noyau territorial de son parent. Autrement, une malchance lors de l'application des opérateurs de mutation aurait davantage de chance de contribuer à une convergence prématurée de l'AGENT en ne créant aucun individu apte à explorer le domaine.

Notez qu'à chaque fois qu'une nouvelle valeur est attribuée pour un paramètre de C_{1i}, il est important de vérifier que cette valeur reste à l'intérieur des bornes permises pour le paramètre (généralement entre 0 et 1, puisque les paramètres sont adimensionnels). Sinon, on doit poser C_{1i} égal à la borne qu'il enfreint. Ce confinement peut nuire à la création de nouveaux individus viables en ramenant certains individus à l'intérieur des noyaux territoriaux, surtout si l'optimum est près des frontières, mais il est évidemment nécessaire pour assurer le bon fonctionnement de l'algorithme.

La figure 5.5 résume l'application de l'opérateur de mutation de l'AGENT.

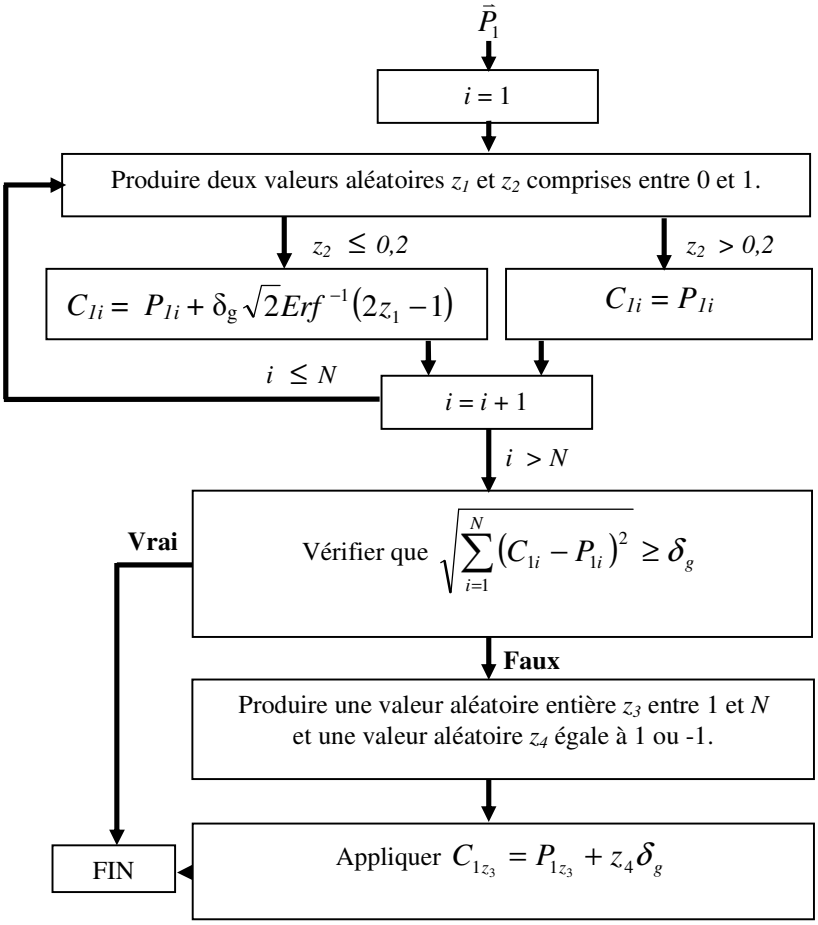


Figure 5.5 : Opérateur de mutation de l'AGENT

5.5.2 Opérateur de croisement

L'opérateur de croisement est employé par l'AGENT avec l'objectif de contribuer à la convergence de l'algorithme aux environs d'un optimum.

Comme dans le cas de l'opérateur de mutation, l'opérateur de croisement employé par l'AGENT ressemble beaucoup à l'opérateur équivalent des AGR (voir équation D.3.1). Il s'agit en effet de produire deux enfants situés sur un vecteur reliant les deux parents.

Toutefois, cet opérateur dispose d'une variation majeure sur l'opérateur des AGR. Au lieu de choisir aléatoirement entre 0 et 1 la valeur de θ représentant la contribution respective θ et 1-θ de chaque parent à un enfant donné, l'AGENT génère, si possible, une valeur de θ permettant aux enfants de naître à l'extérieur des territoires interdits par leurs parents. Pour ce faire, l'algorithme déterminera une borne inférieure Θ_1 et une borne supérieure Θ_2 pour la valeur de θ. La figure 5.6 illustre l'emplacement du point $\vec{C}_1$ selon la valeur de θ, ainsi que l'emplacement voulu pour les bornes Θ_1 et Θ_2. Les cercles autour des individus $\vec{P}_1$ et $\vec{P}_2$ représentent les limites actuelles de leur territoire interdit.

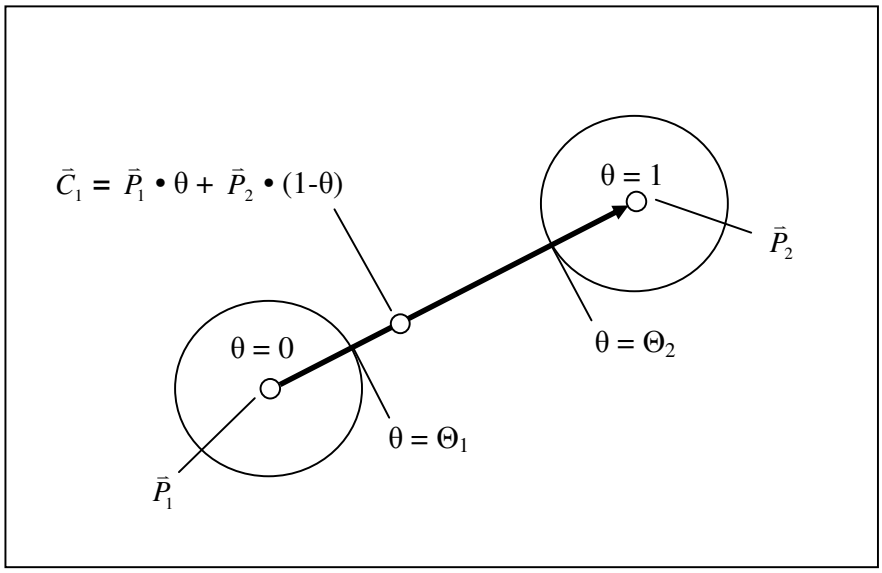


Figure 5.6 : Position de $\vec{C}_1$ selon la valeur de θ

Les bornes sont donc calculées selon les relations suivantes :

$$\Theta_1 = \frac{\delta_g}{\sqrt{\sum_{i=1}^{N} \left(P_{2,i} - P_{1,i}\right)^2}} \tag{5.13}$$

$$\Theta_2 = 1 - \Theta_1 \tag{5.14}$$

À ce stade, il est fort possible que les territoires autour de $\vec{P}_1$ et $\vec{P}_2$ couvrent la totalité du vecteur sur lequel peut naître $\vec{C}_1$. Cette situation se produit lorsque la relation suivante n'est pas satisfaite :

$$\Theta_1 \leq \Theta_2 \tag{5.15}$$

La figure 5.7 illustre le cas limite, où $\Theta_1 = \Theta_2$.

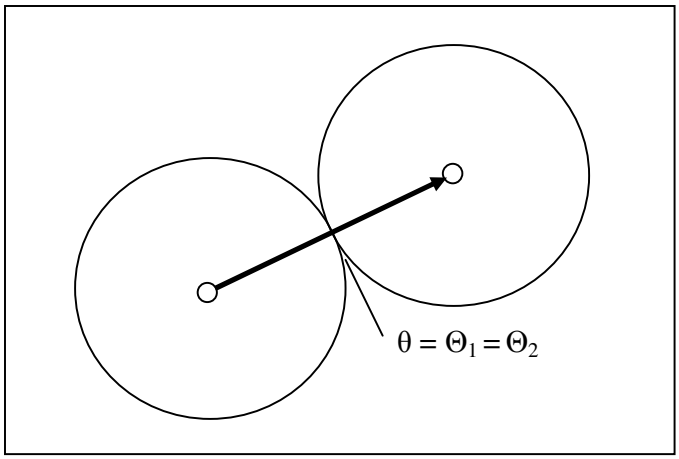


Figure 5.7 : Cas limite à partir duquel la création de $\vec{C}_1$ ne sera plus possible.

Dans la situation où l'équation 5.15 n'est pas satisfaite, le croisement n'est pas possible puisque les enfants seraient automatiquement détruits. Dans un tel cas, afin que les enfants $\vec{C}_1$ et $\vec{C}_2$ aient une chance d'être valides, l'algorithme appliquera plutôt l'opérateur de mutation sur chacun des parents pour les confectionner.

De même, si les deux parents ne disposent pas des mêmes paramètres (par exemple parce qu'ils proviennent d'un autre îlot de solutions utilisant un modèle paramétrique différent), aucun croisement ne sera possible, et on appliquera plutôt l'opérateur de mutation.

Si des bornes valides sont découvertes, il ne reste alors plus qu'à appliquer l'équation suivante pour terminer le croisement :

$$\vec{C}_1 = (1-\theta)\cdot \vec{P}_1 + (1-\theta)\cdot \vec{P}_2$$
$$\vec{C}_2 = (1-\theta)\cdot \vec{P}_2 + (1-\theta)\cdot \vec{P}_1 \qquad (5.16)$$

où $\vec{P}_1$ et $\vec{P}_2$ sont les deux vecteurs parents

θ est un nombre aléatoire comprit entre Θ_1 et Θ_2

$\vec{C}_1$ et $\vec{C}_2$ sont les deux enfants résultants

La figure 5.8 illustre le schéma d'application de l'opérateur de croisement de l'AGENT.

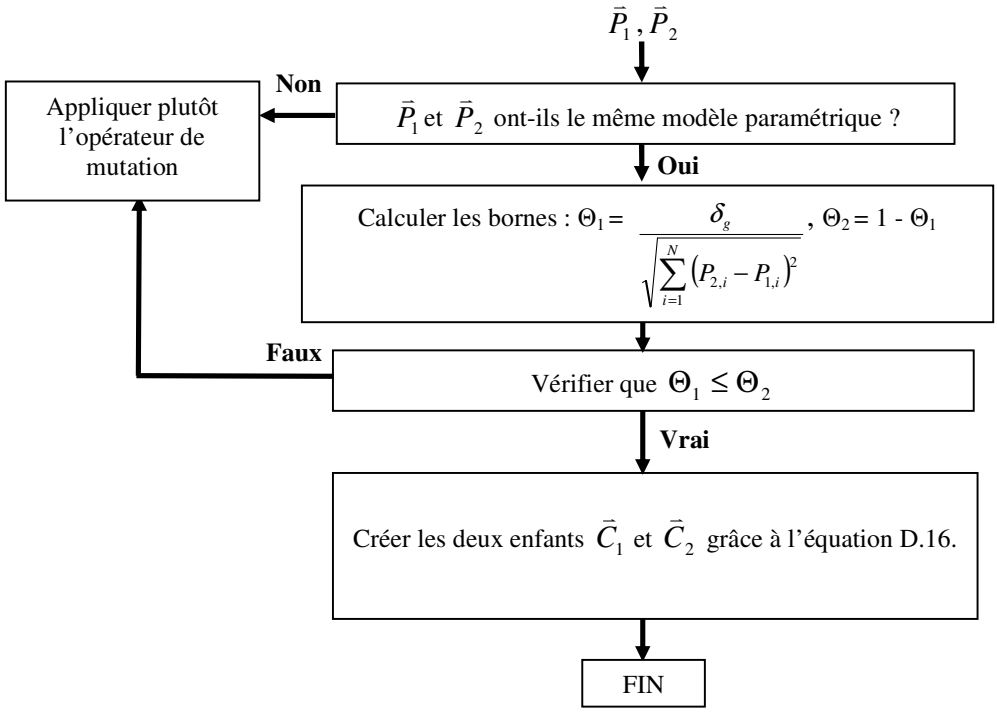


Figure 5.8 : Opérateur de croisement de l'AGENT

5.5.3 Opérateur de substitution

Une façon intéressante de visualiser l'application d'un modèle substitut de F($\vec{x}$) est de considérer cette application comme un opérateur de recherche pour l'AG. En effet, comme dans le cas des mutations et des croisements, le modèle substitut est alors

simplement utilisé entre deux évaluations de F($\vec{x}$) pour choisir le(s) prochain(s) individu(s) à évaluer.

L'opérateur de recherche suggéré, que l'on pourrait nommer *opérateur de substitution*, suit la démarche de la figure 5.9.

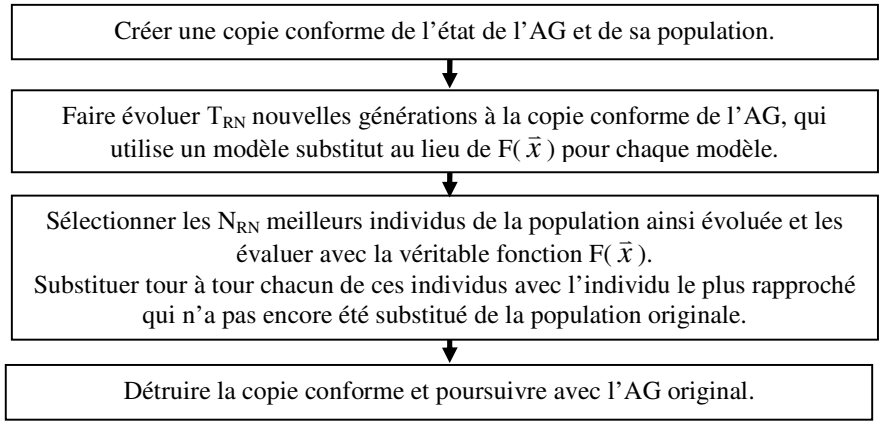


Figure 5.9 : Opérateur de substitution appliqué à un AG régulier.

Cet opérateur de recherche s'inspire de la méthode de Ratle (1998). Cette dernière méthode devient alors un cas particulier de l'opérateur de substitution pour lequel le nombre d'individus substitués est égal à la population entière de l'algorithme (c'est-à-dire que $N_{RN} = N_{pop}$).

Les enfants **C** produits par la substitution ne devront pas substituer aléatoirement n'importe quels individus de la population **P** de l'algorithme. Ceci pourrait avoir comme effet secondaire de provoquer une convergence prématurée de l'algorithme. Chaque enfant devra plutôt remplacer l'individu de **P** qui n'a pas encore été substitué avec lequel il a la plus faible distance euclidienne.

Application de l'opérateur de substitution à l'AGENT

Dans l'AGENT, l'objectif principal de l'opérateur de substitution est de permettre à l'algorithme de passer outre les noyaux territoriaux actuels et de raffiner sa recherche autour des optimums suspectés. Par contre, pour éviter de nuire à la recherche globale de l'algorithme, la substitution dans l'AGENT utilisera un nombre N_{RN} d'enfants égal à 1. Dans le cas contraire, plusieurs points très rapprochés pourraient se substituer aux points de la population et aller à l'encontre de la philosophie des noyaux territoriaux, qui se veut de répartir équitablement les points sur le domaine. En utilisant $N_{RN} = 1$, l'opérateur de substitution effectue simplement une translation de l'un des points vers l'optimum le plus probable du problème actuel (voir figure 5.10).

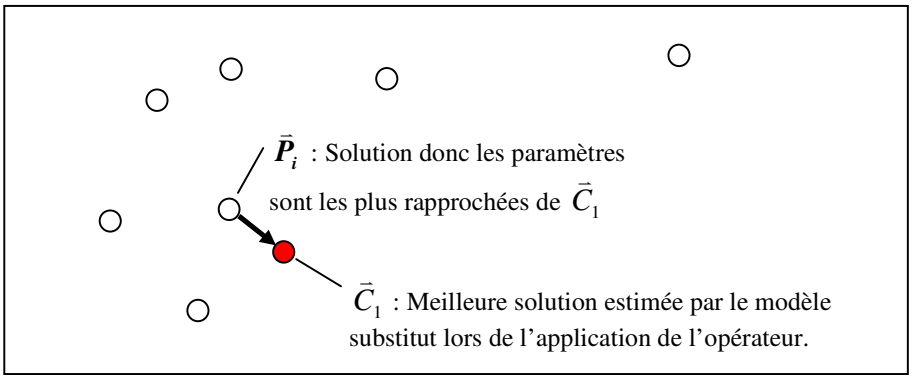


Figure 5.10 : Opérateur de substitution pour $N_{RN} = 1$, $\vec{P}_i$ est remplacé par $\vec{C}_1$.

Finalement, l'enfant ne devra être substitué que s'il s'avère un meilleur candidat que l'individu qu'il devrait en principe remplacer. Ceci évite l'effet destructeur d'une mauvaise interpolation et assure la convergence de l'algorithme vers un point au moins aussi bon que la solution initiale.

Ainsi, on doit :

Remplacer $\vec{P}_i$ par $\vec{C}_1$, où i est l'individu pour lequel

$$\sum_{j=1}^{N}\left(P_{i,j}-C_{1,j}\right)^2 = \min_{i=1}^{N_{pop}}\left[\sum_{j=1}^{N}\left(P_{i,j}-C_{1,j}\right)^2\right] \qquad (5.17)$$

si et seulement si F($\vec{P}_i$) < F($\vec{C}_1$).

Évidemment, si l'individu $\vec{P}_i$ ne dispose pas du même modèle paramétrique que $\vec{C}_1$, la relation 5.17 ne sera pas valide. Dans un tel cas, afin que cet individu ne soit pas remplacé, il est suggéré de poser la distance $\sum_{j=1}^{N}\left(P_{i,j}-C_{1,j}\right)^2 = N$.

Maintenant, on se rappelle que le but ultime de l'opérateur de substitution dans l'AGENT est de permettre à l'algorithme d'utiliser la connaissance qu'il a acquise sur la topologie de la fonction pour raffiner la solution autour des optimums probables à coût minimum.

Pour ce faire, il est important de désactiver les mécanismes favorisant l'exploration globale de la fonction, puisque ceux-ci sont justement destinés à empêcher l'algorithme de raffiner les optimums en cours.

Lors de la création de la copie conforme de l'algorithme, nous génèrerons les valeurs suivantes pour la copie :

Ptot^{RN} = Ptot
P^{RN} = P
$\gamma_c^{g,RN} = 0$
$\alpha_m{}^{RN} = 0$
où **RN** signifie l'appartenance au clone de l'algorithme.

On constate que, plutôt que de poser $\gamma_c^{o,\,\mathrm{RN}} = \gamma_c^o$, on pose $\gamma_c^{o,\,\mathrm{RN}} = 0$. Ainsi, le rayon territorial δ_g des individus de $\mathbf{Ptot}^{\mathbf{RN}}$ de la copie conforme de l'algorithme sera toujours évalué égal à 0 (voir équation 5.8) et les rayons territoriaux n'élimineront plus les nouveaux individus.

Ensuite, on sait que l'opérateur de mutation est l'opérateur qui est davantage utilisé pour explorer le domaine, alors que l'opérateur de croisement est généralement davantage relié à la convergence de l'algorithme. En posant $\alpha_m = 0$, on empêche les mutations (qui auraient de toute façon un pas de 0 puisque $\delta_g = 0$) et on laisse le clone de l'algorithme utiliser uniquement les croisements (et donc des combinaisons linéaires) de ses individus pour découvrir les points les plus prometteurs suggérés par le modèle substitut.

Vu la façon dont l'opérateur de croisement est conçu, les croisements impossibles (dus à deux parents utilisant des modèles paramétriques différents) généreront quand même des mutations avec un pas nul, créant des copies conformes de leurs parents. Cette situation n'est toutefois pas réellement problématique, les individus provenant de modèles différents n'ayant pas tendance à cohabiter longtemps dans une même population.

La figure 5.11 illustre le schéma de l'opérateur de substitution, tel qu'appliqué à l'AGENT. Par défaut, l'opérateur utilise $T_{RN} = 10$.

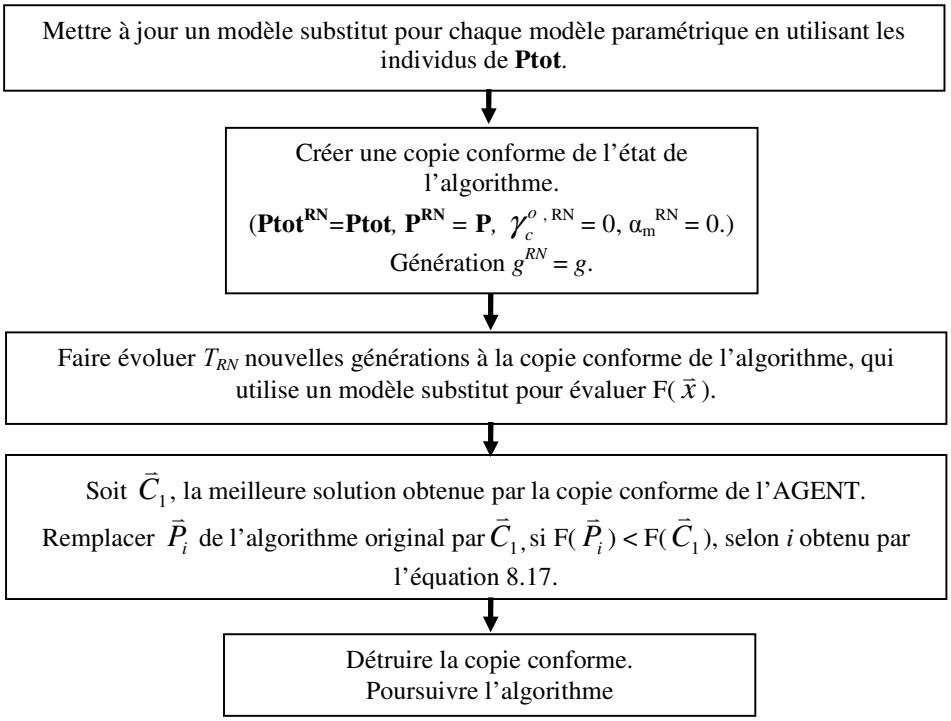


Figure 5.11 : Opérateur de substitution appliqué à l'AGENT

Dans les faits, on peut supposer que le meilleur choix pour N_{RN} est très dépendant de la qualité de l'approximation fournie par le modèle substitut à F($\vec{x}$) et de la topologie de la fonction. En effet, N_{RN} est le nombre d'enfants générés et évalués par cet opérateur à chaque génération. Si le modèle substitut est de piètre qualité, la plupart de ces évaluations ne donneront pas de résultats intéressants et ne feront que ralentir l'algorithme en question. Ainsi, dans le cas où l'usager estime que le modèle substitut n'est pas apte à représenter correctement la fonction, la possibilité lui est laissée de poser $N_{RN} = 0$ et ainsi de désactiver l'opérateur de substitution.

L'approche actuelle sera validée pour le type de problème qui nous intéresse à la section 6.1.3, où l'on comparera l'efficacité de l'opérateur de substitution utilisé dans l'AGENT ($N_{RN} = 1$) avec l'utilisation du modèle substitut suggérée par Ratle (1998) ($N_{RN} = N_{pop}$).

Mise à jour des modèles substituts

Afin de profiter des évaluations de F($\vec{x}$) effectuées en cours d'algorithme, il est important d'entraîner le RNA substitut avant l'application de l'opérateur de substitution.

Cet entraînement sera effectué selon la méthode décrite à l'annexe A (section A.4), en utilisant pour ce faire tous les individus de la population totale **Ptot**.

Notez qu'une mise à jour en temps réel d'un réseau de neurones peut s'avérer couteuse en temps de calcul. Aussi les options suivantes sont-elles possibles dans l'implémentation :

1 – Désactiver la capacité d'apprentissage en temps réel de l'algorithme. Cette option est utile si l'utilisateur juge le modèle substitut suffisant dans son état actuel.

2 – Permettre uniquement l'entraînement en temps réel du réseau grâce aux R_N points les plus récents de **Ptot** et ce, uniquement après chaque R_N évaluations. Ceci permet d'alléger la durée de l'entraînement, tout en continuant d'améliorer les performances du réseau localement aux environs de la solution actuelle. Cette option est particulièrement utile si l'on s'attend à un nombre d'évaluations extrêmement élevé.

3 – Désactiver complètement l'opérateur de substitution de l'algorithme. Cette option est utile pour effectuer une optimisation rapide si aucun RNA n'est disponible pour le modèle actuel, et que l'utilisateur ne désire pas consacrer de temps à son entraînement.

Par contre, rappelons-nous que nous nous intéressons particulièrement à des problèmes pour lesquels l'évaluation de la fonction coût elle-même est très couteuse. Dans de tels cas, il s'avère souvent intéressant de peaufiner au maximum notre modèle substitut en cours de calcul pour maximiser les chances de cibler correctement le meilleur point à évaluer.

5.6 Critères d'arrêt

L'AGENT dispose de deux critères d'arrêt possibles.

Le premier critère est le nombre d'évaluations total permis par l'usager. L'équation 5.18 illustre ce critère d'arrêt.

$$Ev = Ev_{\max} \tag{5.18}$$

où Ev est le nombre d'évaluations de F($\vec{x}$) effectuées depuis le lancement de l'algorithme.

Ev_{max} est le nombre total d'évaluations de F($\vec{x}$) permises par l'utilisateur.

Dès qu'une demande d'évaluation de F($\vec{x}$) est soumise par l'algorithme, il est important de vérifier que le nombre total d'évaluations Ev de la fonction coût n'est pas déjà égal à Ev_{max}. Si c'est le cas, la fonction devrait retourner une erreur (l'individu ne devrait pas être évalué). Ainsi, même si une génération commande plus d'évaluations de F($\vec{x}$) que le nombre permis, l'algorithme aura utilisé exactement le nombre voulu d'évaluations à la fin de cette génération. Le critère d'arrêt donné par l'équation 5.18 sera alors respecté.

Le second critère d'arrêt tente de prédire si la convergence de l'algorithme est suffisante pour les besoins de l'utilisateur. Par exemple, si les paramètres de l'algorithme semblent se trouver à moins de 0,001 de l'optimum, et que l'utilisateur désire une précision de plus ou moins 0,001 sur ses paramètres, il serait approprié d'arrêter l'algorithme. Avec l'AGENT, il peut s'avérer plutôt difficile de savoir la distance entre les points actuels et un optimum. Par contre, nous disposons du rayon δ_g des territoires autour des noyaux territoriaux. Cette distance diminue à mesure que l'algorithme converge, et représente en quelque sorte une approximation de l'ordre de grandeur du pas de mutation des individus. Un critère d'arrêt approprié est donc celui de l'équation 5.19, par lequel

l'utilisateur peut décider du rayon minimal requis par l'algorithme avant de considérer sa convergence. Il est toutefois important de constater que, puisque l'opérateur de croisement existe et que les mutations sont de dimensions aléatoires, δ_{min} ne représente que très approximativement la précision de l'algorithme. Une pratique suggérée serait donc de poser le rayon minimal δ_{min} égal à un nombre d'un ordre de grandeur inférieur à la précision voulue sur le résultat.

$$\delta_g \leq \delta_{\min} \tag{5.19}$$

où δ_{min} est une constante déterminée par l'utilisateur, représentant le rayon territorial minimal permis par l'algorithme.

Si plusieurs modèles paramétriques possédant divers δ_g différents sont présents dans la population, tous les δ_g devront satisfaire au critère d'arrêt de l'équation 5.19 pour que le critère d'arrêt soit confirmé.

En bref, lorsque l'algorithme arrive au stade où il doit vérifier ses critères d'arrêt, il vérifie si l'une ou l'autre des équations 5.18 et 5.19 est satisfaite. Si tel est le cas, l'AGENT propose les solutions contenues dans la population **P** de chacun des îlots et termine son activité.

5.7 L'AGENT utilisé comme algorithme insulaire

Comme la plupart des algorithmes génétiques, l'AGENT est capable d'incorporer diverses heuristiques, dont l'heuristique des algorithmes insulaires. Dans un tel cas, chaque îlot sera considéré comme une population indépendante, disposant de sa propre population totale **Ptot**, de sa propre population active **P** et de son propre rayon territorial $\delta_{g..}$ Notez que dans un tel cas, toutes les populations devront satisfaire à l'un des critères d'arrêt pour que l'algorithme prenne fin.

L'un des inconvénients des algorithmes insulaires est que l'utilisateur doive spécifier la valeur des paramètres de migration I_M et N_M (voir section D.6.2). Dans les faits, ce que l'on désire en utilisant cette heuristique est, principalement, de séparer les populations le temps que celles-ci trouvent chacun une solution mature. Ceci promeut la diversité en empêchant que la meilleure solution ne tue dans l'œuf toutes les autres solutions prometteuses. Une fois la maturité atteinte, on désire provoquer une migration pour permettre à la meilleure solution résultante de se répandre dans les autres populations pour éviter de perdre d'inutiles évaluations à raffiner les solutions les moins viables.

Vu le caractère aléatoire des AE et le fait que la topologie de F(**x**) soit inconnue, il est impossible de prévoir ce que sera une bonne valeur de l'intervalle de migration I_M pour le problème. Par contre, pour un AGENT, on dispose déjà d'une certaine information sur l'état de convergence d'une population. En effet, supposons que pour l'une des populations de l'algorithme insulaire, on découvre que $\Delta D^g = \Delta D^{g-1}$. On pourra alors supposer que cette population est en bonne voie de convergence (voir section 5.2.2) et sera alors prête à la migration.

L'approche proposée dans ce document est donc de générer avant l'application des opérateurs de recherche à chaque génération un ensemble $\mathbf{P_M}$ de populations en état de convergence (i.e. des populations pour lesquelles $\Delta D^g = \Delta D^{g-1}$). Si $\mathbf{P_M}$ compte plus d'une population, chaque population active de $\mathbf{P_M}$ échangera l'un de ses individus choisi aléatoirement avec chacune des autres populations actives de $\mathbf{P_M}$.

Le pseudo-code utilisé pour effectuer cette migration est le suivant :

Pour i allant de 1 jusqu'à k_{ile}.
    Pour j allant de (i+1) jusqu'à k_{ile}
        Générer deux nombres entiers aléatoires z_1 *et* z_2 *entre 1 et* N_{pop}.
        $P_{M,i}$ *échange son individu* z_1 *contre l'individu* z_2 *de* $P_{M,j}$

où k_{ile} est le nombre d'îlots de l'algorithme insulaire.

$\mathbf{P_M}$ regroupe toutes les populations pour lesquelles $\Delta D^g = \Delta D^{g-1}$.

$\mathbf{P_{M,i}}$ est la $i^{\text{ème}}$ population de $\mathbf{P_M}$.

De cette façon, l'utilisateur n'aura pas à se soucier du choix de I_M et de N_M, et la fréquence à laquelle les individus seront échangés entre les populations sera conséquente à la convergence de l'algorithme plutôt que d'agir de façon fixe et non informée.

Cette approche sera validée sur un problème académique à la section 6.1.4.

5.7.1 Problème disposant de plusieurs modèles paramétriques

Nous avons mentionné à plusieurs reprises dans ce chapitre certaines exceptions au fonctionnement normal de l'algorithme dues à l'utilisation d'un modèle paramétrique différent pour certains des individus habitant une même population.

Puisque l'algorithme utilise N_{pop} mutations d'un même individu pour générer sa population initiale, chaque individu partageant une même population initiale sera construit à l'aide du même modèle paramétrique.

Par contre, si plusieurs solutions disposant de modèles différents sont utilisées pour générer la population des différents îlots, chacun de ces îlots aura son propre modèle paramétrique.

Ensuite, une fois que les migrations auront commencé, certains individus issus d'un modèle paramétrique seront amenés à côtoyer des individus issus d'un modèle différent dans une même population. Afin de clarifier la démarche à effectuer lors de certaines

exceptions causées par ce phénomène, les règles particulières à l'application de plusieurs modèles paramétriques sont énumérées ci-dessous :

- Chaque individu $\vec{P}_i$ doit être conservé en mémoire avec le numéro Φ_i du modèle paramétrique auquel il appartient.
- Un rayon territorial δ_g différent devra être calculé pour chaque modèle de solution existant dans **Ptot**, puisque δ_g dépend du nombre de paramètres N du modèle. Lorsque l'on évalue si un individu doit être éliminé avant évaluation à cause des noyaux territoriaux, seuls les noyaux émergeant des individus de **Ptot** disposant du même modèle paramétrique que l'individu devraient être envisagés.
- La distance euclidienne entre deux individus $\vec{P}_i$ et $\vec{P}_j$ disposant d'un modèle paramétrique différent est N, soit le nombre de paramètres du modèle Φ_i ou Φ_j qui dispose du plus grand nombre de paramètres. Ceci correspond en fait à une distance maximale de 1 entre chacune des paires de paramètres pour ces modèles. De cette façon, les ΔD^g signifieront avec raison une divergence ($\Delta D^g > 0$) si un nouveau modèle paramétrique commence à se répandre dans la population et une convergence ($\Delta D^g < 0$) si un modèle paramétrique minoritaire est éliminé de la population.
- Aucun opérateur de croisement n'est valide entre deux parents $\vec{P}_i$ et $\vec{P}_j$ si $\Phi_i \neq \Phi_j$. L'opérateur de mutation sera alors plutôt utilisé sur chacun des parents.
- Lors de l'application de l'opérateur de substitution, un RNA différent devrait être utilisé pour évaluer la valeur adaptative de chacun des modèles Φ_i.
- Lors du calcul du critère d'arrêt $\delta_g < \delta_{min}$, si plusieurs δ_g existent, tous doivent satisfaire le critère pour que celui-ci soit considéré comme atteint.

5.8 AGENT appliqué à des problèmes de nature académique

On se rappelle que l'AGENT a été conçu pour générer des solutions dans le cas où la fonction coût est plutôt lourde de calcul, comme c'est typiquement le cas des problèmes industriels d'analyse de pièces mécaniques. Puisque l'idée est de produire rapidement un design valable en phase conceptuelle, le nombre d'évaluations permises ne sera généralement pas très élevé, de l'ordre de quelques centaines à quelques dizaines de milliers selon le problème étudié.

Toutefois, puisque les instances dans la littérature où l'on retrouve à la fois le modèle paramétrique détaillé et les résultats d'une optimisation pour de tels problèmes sont rares, l'AGENT a d'abord été comparé avec les autres algorithmes génétiques émergents en utilisant une série de problèmes tests retrouvés régulièrement dans la littérature pour comparer entre eux les divers algorithmes d'optimisation (voir section 6.3).

Ces problèmes ont été conçus de façon à illustrer le type de problèmes pour lesquels les algorithmes sont efficaces et pour lesquels ils présentent des lacunes. Par contre, ces problèmes sont aussi très peu couteux à évaluer, ce qui fait que les résultats que l'on retrouve communément utilisent dans les environs de 300 000 évaluations de fonction.

Puisque l'AGENT conserve par défaut tous les individus ayant été évalués avec la véritable fonction $F(\vec{x})$ dans N_{pop}, il devrait donc évaluer plusieurs centaines de milliers de noyaux territoriaux à chaque nouvelle évaluation de fonction lors de ces tests, ce qui s'avèrerait extrêmement fastidieux en temps de calcul si l'on désire effectuer chaque problème de façon répétée afin d'obtenir une bonne approximation de la moyenne et de l'écart type des résultats.

Ainsi, pour les cas où la fonction coût est plutôt simple à évaluer et que le nombre d'évaluations est immense (comme c'est le cas lors de l'utilisation de cas tests

académiques typiques tels que la fonction de Rosenbrock ou de Schwefel), l'option est laissée à l'utilisateur de poser une limite maximale N_{max} à la population, qui indiquera le nombre maximal d'individus conservés dans **Ptot.** Les individus conservés seront alors simplement les N_{max} individus les plus récents, qui sont généralement les individus pour lesquels il existe la plus grande probabilité qu'un nouvel individu partage le territoire.

Par défaut, dans les tests de ce document, lorsqu'il n'est pas mentionné que N_{max} est utilisé, sa valeur sera posée égale à ∞^+.

Le chapitre suivant énonce quelques problèmes académiques validant les performances de l'AGENT en comparaison avec certains autres algorithmes évolutionnaires présents dans la littérature.

Chapitre 6.
PROBLÈMES TESTS ACADÉMIQUES

Cette section de la thèse s'intéresse aux problèmes tests de nature académique explorés lors de la création et de la validation de l'algorithme AGENT décrit au chapitre 5.

La section 6.1 présente d'abord les problèmes issus du générateur de paysage gaussien à valeur maximale du lot, qui génère aléatoirement des problèmes ayant des caractéristiques similaires aux problèmes de design de pièces mécaniques. Ces problèmes ont servi de tests préliminaires pour déterminer les différentes caractéristiques requises par l'AGENT, et par conséquent l'AGENT y performe de façon très compétitive.

On se rappelle que l'AGENT performait aussi de façon très compétitive dans le cas des problèmes d'optimisation avec contraintes académiques typiques (voir section 6.2). Dans l'optique de bien mettre en relief les forces et les faiblesses de l'AGENT, la section 6.3 compare plutôt ses performances à celles de différents algorithmes populaires pour une suite de problèmes académiques simples sans contraintes.

6.1 Générateur de paysage gaussien

Les générateurs de paysages gaussiens (Gaussian landscape generator) sont des outils fréquemment employés dans la littérature pour comparer les performances de divers algorithmes génétiques (Stuckman, 1998 ; De Jong, Potter & Spears , 1997 ; Morrison & De Jong, 1999 ; Michalewicz, Deb, Shmidt & Stidsen, 2000 ; Gaviano, Kvasov & Lera, 2003).

Celui que nous allons employer dans ce chapitre a été proposé dans Yuan & Gallagher (2003) et Yuan & Gallagher (2006) et se nomme le générateur de paysages gaussiens à valeur maximale du lot (ou MSG pour Maximum Set Gaussian landscape generator). Ce MSG a été utilisé plusieurs fois dans la littérature (Yuan & Gallagher, 2004 ; Gong, Nakamura & Tamaki, 2005). Il est particulièrement intéressant pour notre problème de design structurel puisqu'il génère aléatoirement des fonctions ressemblant beaucoup aux problèmes standards de design. En effet, elles sont non-linéaires, multimodales, légèrement discontinues et possèdent des zones où la valeur de la fonction est presque nulle (pour simuler la portion du domaine irréalisable ou produisant des erreurs) ainsi que d'autres zones où la fonction coût réagit de façon plus continue avec les variables d'entrée.

Le principe de base du MSG est de générer aléatoirement un ensemble pondéré de fonctions gaussiennes à variables multiples et, lorsqu'un point est évalué, de présenter comme solution la plus grande valeur estimée à l'aide de l'ensemble de ces fonctions. L'objectif de l'optimisation sera alors de maximiser cette valeur.

Chacune des fonctions gaussiennes est calculée par l'équation suivante :

$$F(\vec{x}) = \max_{i=1}^{p_n} \left[w_{g,i} \exp\left(-\frac{1}{2N} (\vec{x} - u_i)^T \Sigma_i^{-1} (\vec{x} - u_i) \right) \right] \tag{6.1}$$

où $w_{g,i}$ est le poids de la $i^{ème}$ fonction gaussienne du problème

$\vec{x}$ est le vecteur de paramètres du problème

N est le nombre de dimensions du problème

p_n est le nombre total de fonctions gaussiennes du problème

u_i est le vecteur contenant les moyennes de la $i^{ème}$ fonction gaussienne

Σ_i^{-1} est la matrice de covariance inverse générée aléatoirement pour la $i^{ème}$ fonction gaussienne.

Ainsi, en utilisant le MSG, il est possible de générer des fonctions ayant un nombre choisi d'optimums p_n dont les valeurs sont déterminées par les poids aléatoires $w_{g,i}$.

La figure 6.1 présente un exemple de fonction où $p_n = 3$ et $N = 1$. On peut y distinguer les trois gaussiennes constructrices, ainsi que la courbe $F(x)$ passant par la plus haute valeur parmi ces trois fonctions.

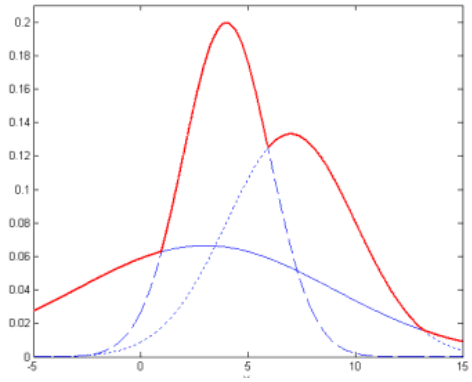


Figure 6.1 : Exemple de paysage gaussien où $p_n = 3$ et $N = 1$.

La figure 6.2 , quant à elle, illustre un exemple légèrement plus complexe de paysage pour lequel $p_n = 20$ et $N = 2$. On peut déjà constater la présence de pics de solutions prometteuses, entourées par certaines zones à très faible gradient.

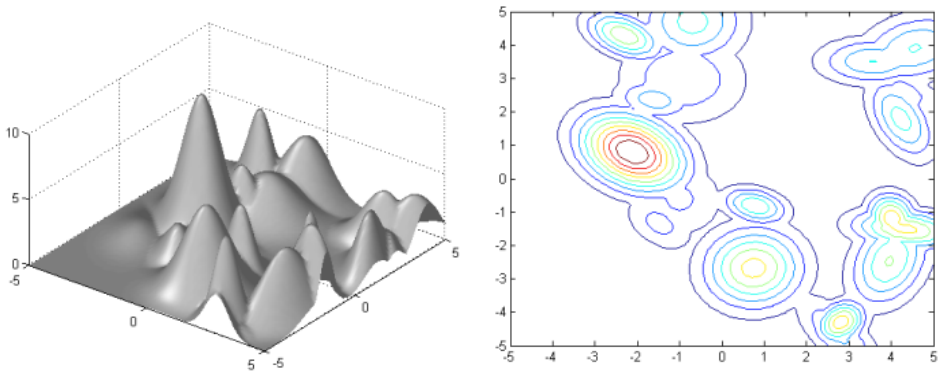


Figure 6.2 : Exemple de paysage gaussien où $p_n = 20$ et $N = 2$.

Le lecteur intéressé pourra trouver plus de détail sur la confection du modèle dans Yuan & M. Gallagher (2003).

6.1.1 Comparaison entre l'AGENT et d'autres algorithmes populaires pour le MSG

En utilisant le MSG, il est possible de générer aléatoirement une série de cas tests permettant de comparer l'efficacité de l'AGENT à celle des autres algorithmes génétiques décrits dans l'annexe D, soient l'AGO, le AGR, le AGED et le AGP2.

Afin de bien représenter l'historique de développement de l'AGENT, les résultats d'un autre algorithme, nommé l'AGR-NT, seront aussi compilés dans cette section. L'AGR-NT est le premier algorithme développé dans le cadre de cette recherche qui utilisait le principe des noyaux territoriaux. Essentiellement, les résultats obtenus en intégrant les diverses heuristiques décrites dans l'annexe D dans l'AGR-NT ont servi de guide pour déterminer les heuristiques à utiliser pour la confection de l'AGENT.

L'AGR-NT est un AGR tel que décrit dans la section D.3, mais utilisant le pas de mutation moyen décrit par l'équation 5.5.1 et le principe des noyaux territoriaux avec un rayon de territoire $\delta = \frac{p_{moyen}}{2}$ pour éliminer les solutions trop rapprochées des solutions actuelles avant leur évaluation.
Le pas moyen utilisé pour la perturbation gaussienne par l'AGR-NT est le suivant

$$p_{moyen} = p_{initial} \frac{\left(Ev_{\max} - Ev\right)}{Ev_{\max}} \tag{6.2}$$

où $p_{initial}$ est le pas initial des mutations de l'algorithme
Ev_{max} est le nombre maximal d'évaluations pour ce test
Ev est le nombre actuel d'évaluations effectuées

Méthodologie de tests

Les problèmes tests utilisés disposent respectivement de N = 2, 4, 6, 8, 10, 15 et 25 paramètres, les nombres les plus élevés représentant plutôt bien la gamme de complexité des problèmes réels en phase conceptuelle. Le nombre d'optimums utilisé est N = 20, avec la valeur la plus élevée parmi les $w_{g,i}$ égale à 1, et les autres choisies aléatoirement entre 0 et 0,8.

Un total de 10 problèmes test a été généré pour chaque nombre de dimensions. Chaque problème a ensuite aussi été résolu 10 fois par chacun des 5 algorithmes génétiques (AGO, AGR, AGP2, AGED et AGR-NT) pour chacune des combinaisons d'heuristiques présentées à la table 6.1.

Fondamentalement, il s'agit d'utiliser toutes les combinaisons d'heuristiques possibles pour chaque algorithme parmi les heuristiques présentées à l'annexe D, soit les algorithmes culturels, les algorithmes insulaires et le taux variable d'application des variables. Notez que puisque l'AGED n'utilise pas les opérateurs de mutation et de croisement, le taux variable d'application des opérateurs ne sera pas utilisé sur lui.

Une fois les résultats compilés pour les cinq algorithmes mentionnés plus haut et leurs différentes combinaisons d'heuristiques, les caractéristiques permettant de mieux performer pour le problème du MSG ont été isolées et assemblées de façon à créer l'AGENT. Chacun des problèmes générés par le MSG a ensuite été résolu 10 fois par l'AGENT pour valider sa création. Le raisonnement conduisant à la construction de l'AGENT sera détaillé plus loin dans cette section et à la section 6.1.2.

Table 6.1. Combinaisons d'heuristiques utilisées

Numéro identifiant la combinaison	Algorithme insulaire	Algorithmes culturels	Taux variable d'application des opérateurs	
1				
2	X			
3		X		
4	X	X		
5			X	Ne s'appliquent pas à l'AGED.
6	X		X	
7		X	X	
8	X	X	X	

Les constantes utilisées pour chacune des heuristiques sont définies dans la table 6.2. Elles correspondent aux valeurs utilisées dans la littérature d'où elles sont issues (voir chapitre 5).

Table 6.2. Paramètres utilisés par les heuristiques

Algorithme insulaire	$k_{île} = 4$ $I_M = 10$ $N_M = 2$
Algorithme culturel	$\alpha_{ec} = 0.4$ $f_{rc} = 0.2$
Taux d'application variable des opérateurs (ces valeurs substituent les taux propres à l'algorithme lorsque l'heuristique est utilisée)	$\alpha_m = 0.95$ $\alpha_c = 0.95$

Les paramètres suivants sont utilisés par les algorithmes génétiques, aussi choisis selon les suggestions générales des articles d'où ils proviennent (voir annexe D et chapitre 5), et sont définis dans la table 6.3.

Table 6.3. Paramètres utilisés par les différents algorithmes génétiques

AGO	$\alpha_m = 0.05$ $\alpha_c = 0.95$
AGR	$\alpha_m = 0.05$ $\alpha_c = 0.95$ Pas moyen de mutation = 0.5
AGED	$F = 0.35$ $p_c = 0.35$
AGP2	$\alpha_m = 0.05$ $\alpha_c = 0.95$ Longueur d'un chromosome = 100 fois le nombre de paramètres.
AGR-NT	$\alpha_m = 0.8$ $\alpha_c = 0.5$ $p_{initial} = 1$
AGENT	$\gamma_c^o = 0.2$ $\alpha_m = 0.5$

Puisque le but de cette étude est de comparer entre eux l'efficacité propre des algorithmes et de leurs opérateurs de recherche, aucun des algorithmes n'utilisera de réseau de neurones substitut à la fonction coût pour la durée de ce test. Ainsi, afin de bien comparer tous les algorithmes sur la même base, l'opérateur de substitution a été retiré temporairement de l'AGENT pour la résolution de ces problèmes.

Tous les algorithmes utilisent une population totale de 80 individus (ou de 4 fois 20 individus dans le cas des algorithmes insulaires). Les individus de la population initiale sont générés aléatoirement à l'intérieur des bornes de la fonction pour chaque variable (soit entre -5 et 5). Le nombre d'évaluations total a été fixé égal à 5000. Ce nombre d'évaluations a été choisi pour mieux représenter la réalité des problèmes de nature industriels. En effet, pour des modèles très complexes, parfois multidisciplinaires et faisant appel à des calculs d'éléments finis, chaque évaluation peut être très coûteuse pour l'utilisateur, prenant un temps de calcul variant de quelques secondes à quelques heures selon la nature du problème et la technologie dont il dispose.

Résultats

Les résultats sont résumés sur les figures 6.3 à 6.9. Les indicateurs de performances utilisés sont le temps de convergence sur l'axe horizontal et l'efficacité sur l'axe vertical. L'efficacité, variant entre 0 et 1, est une mesure de la valeur adaptative moyenne obtenue par l'algorithme par rapport à cette de l'optimum théorique. Le temps de convergence, quant à lui, est une mesure du nombre d'évaluation requises pour atteindre 95% de la valeur adaptative finale (voir annexe B.1.1 pour une définition de l'efficacité et du temps de convergence).

Idéalement, on désire obtenir un point le plus à gauche et en haut possible de la figure, c'est-à-dire un algorithme à la fois rapide et capable de donner une solution de qualité.

Chacun des points de la figure représente la moyenne des 100 tests effectués par un algorithme pour un nombre de variables donné et pour l'une des huit combinaisons d'heuristique pouvant lui être associée. Chaque figure représente les résultats pour les problèmes ayant un certain nombre de variables fixe. Les figures successives représentent donc des problèmes de plus en plus complexes pour l'optimiseur.

Puisque l'on s'intéresse à construire un bon algorithme de recherche globale, on s'intéressa d'abord au point le plus élevé dans la figure (celui ayant la meilleure efficacité), et ensuite à celui le plus à gauche parmi ceux ayant une efficacité acceptable. Les algorithmes qui sont pointés sur les différentes figures sont nommés selon le nom de l'algorithme et le numéro de la combinaison d'heuristique (voir tables 6.3 et 6.1) qui les définit. Par exemple, AGR-NT-2 signifie l'algorithme AGR-NT utilisant la combinaison d'heuristique 2, soit les algorithmes insulaires.

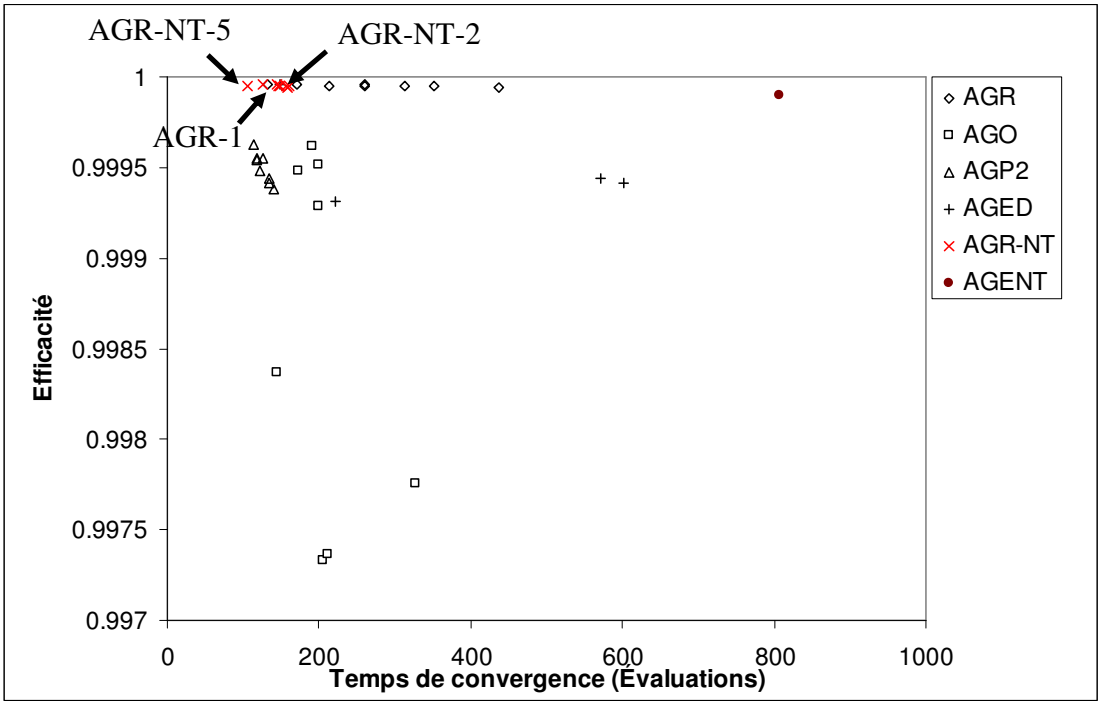


Figure 6.3 : Résultats pour N = 2 variables.

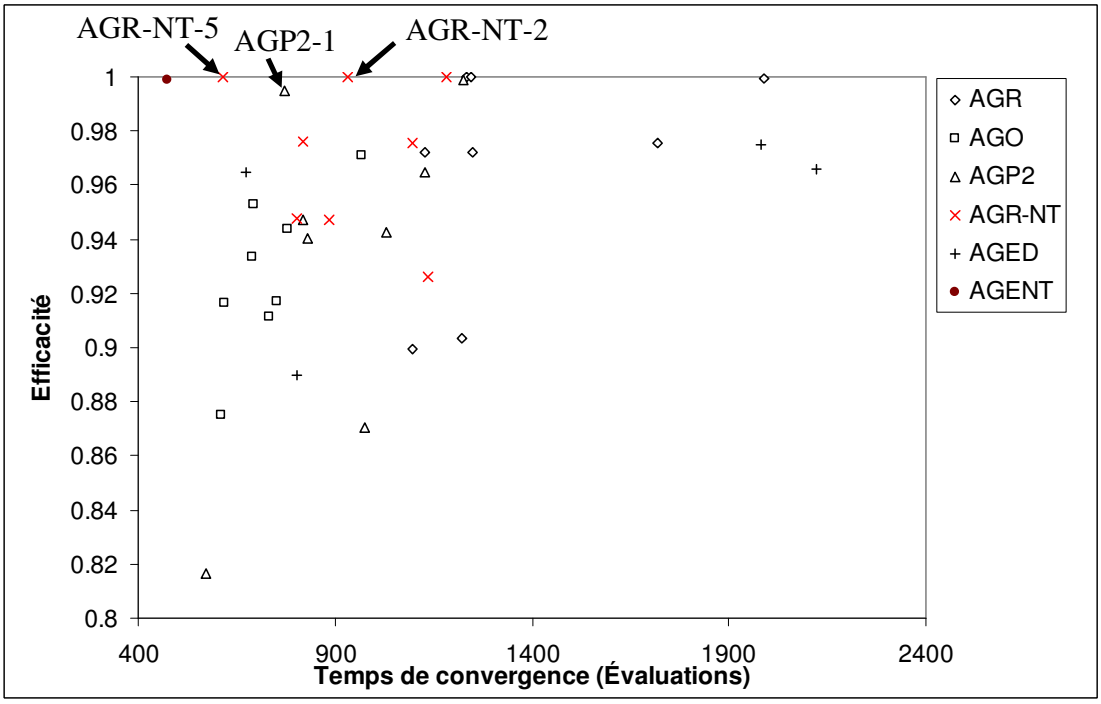


Figure 6.4 : Résultats pour N = 4 variables.

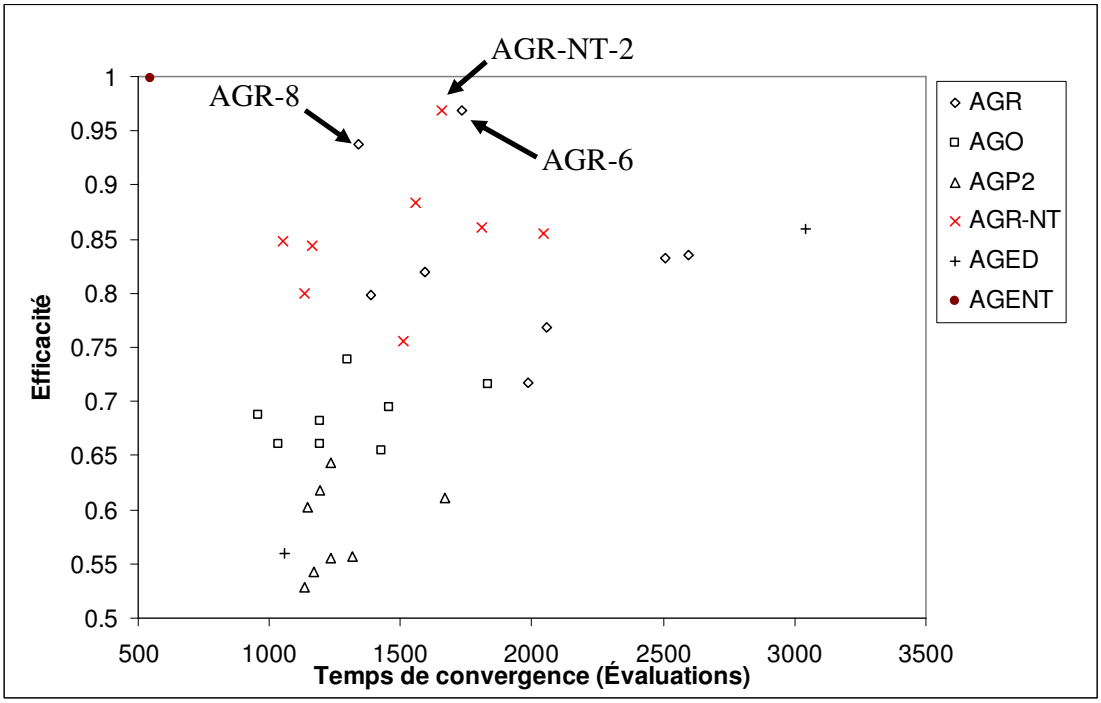


Figure 6.5 : Résultats pour N = 6 variables.

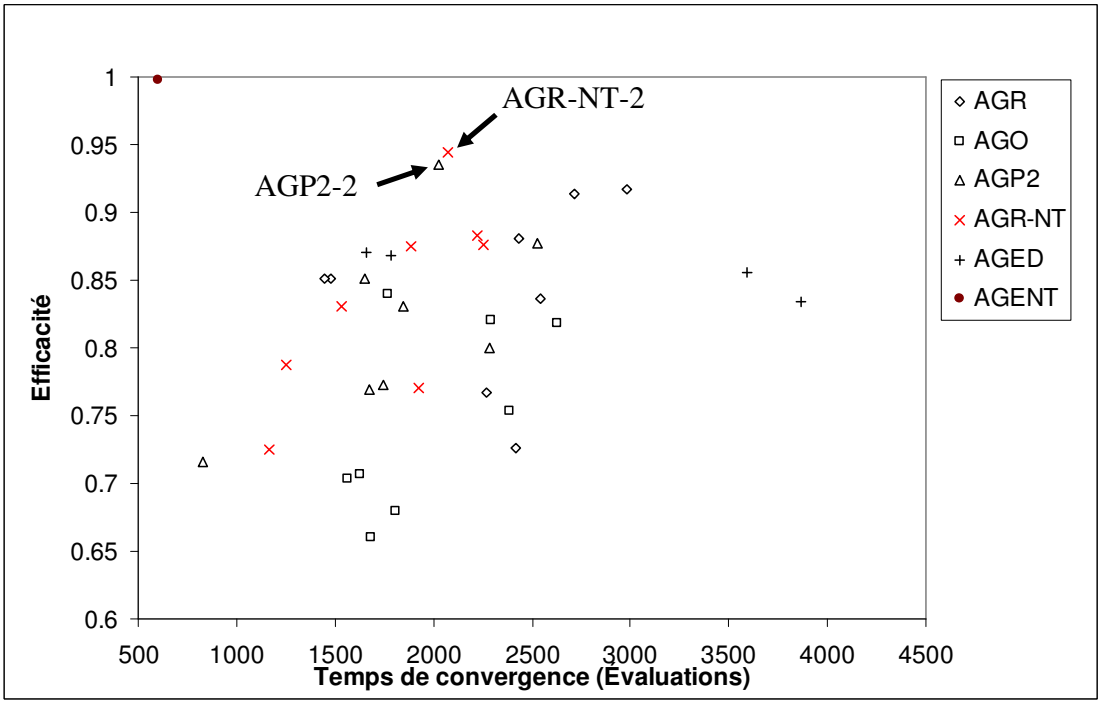


Figure 6.6 : Résultats pour N = 8 variables.

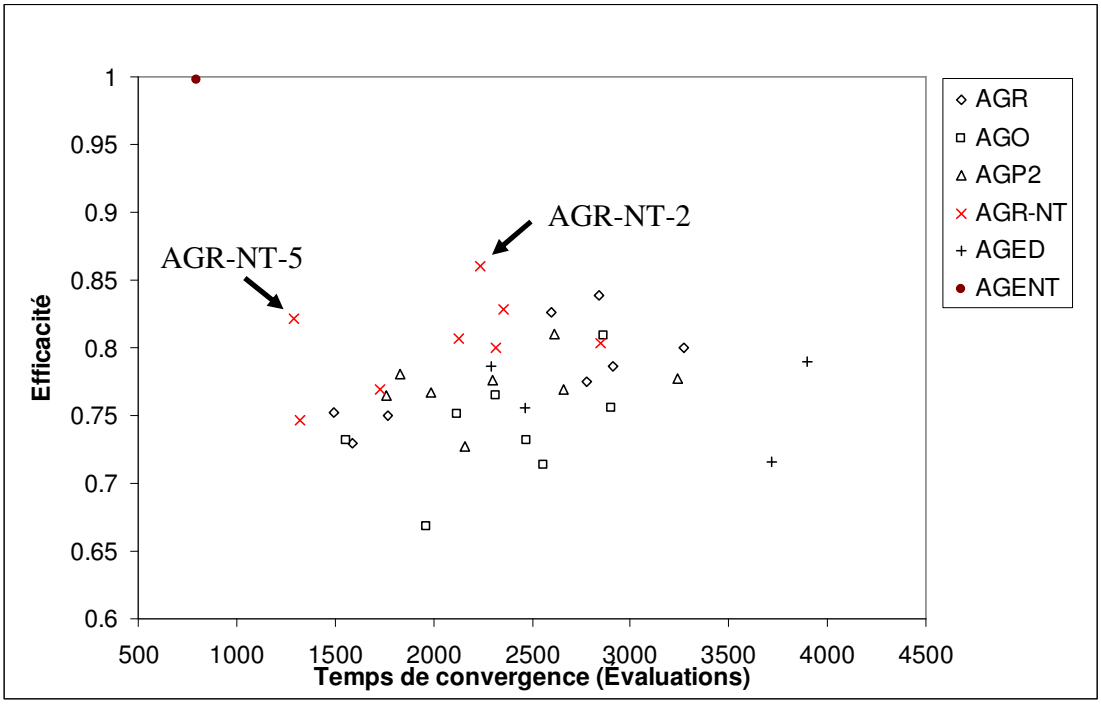


Figure 6.7 : Résultats pour N = 10 variables.

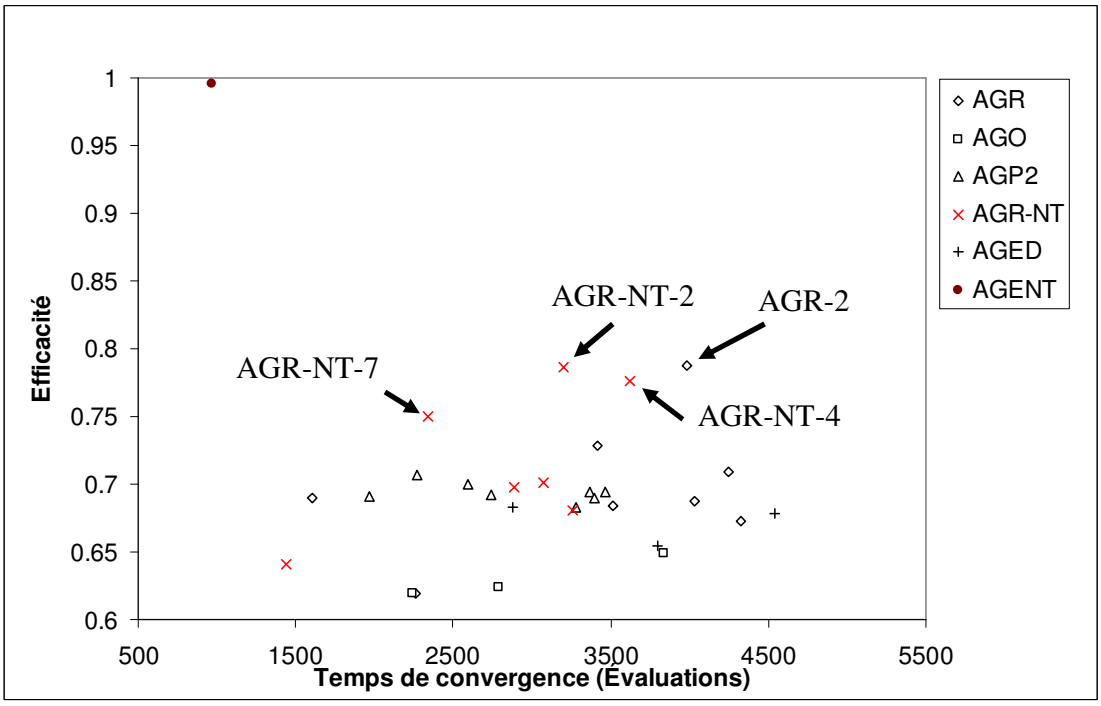


Figure 6.8 : Résultats pour N = 15 variables.

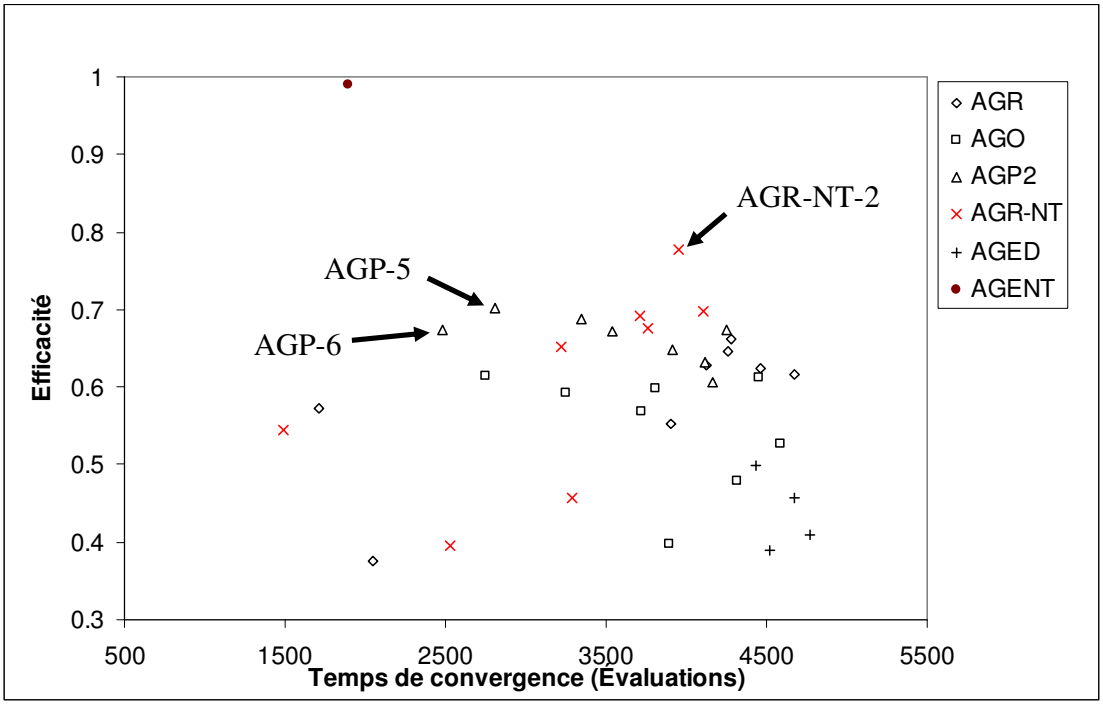


Figure 6.9 : Résultats pour *N* = 25 variables.

Notes sur les résultats de l'AGENT

En observant les figures 6.3 à 6.9, on remarque que l'AGENT performe de façon exemplaire pour tous les problèmes sauf celui à N = 2 variables, où il offre une efficacité raisonnable, mais un temps de convergence plus lent que les autres algorithmes.

À ce stade, le lecteur est invité à se rappeler que l'AGENT a été développé ultérieurement à partir des résultats obtenus pour les autres algorithmes illustrés sur les figures 6.3 à 6.9. Ces résultats ont été utilisés de façon à maximiser la performance de l'AGENT sur les problèmes spécifiques de MSG. Ainsi, en gardant à l'esprit le « No-Free-Lunch Theorem» (chapitre 4), le fait que les performances de l'AGENT surpassent les performances des autres algorithmes pour les problèmes de MSG étudiés ne signifie en aucun cas que l'AGENT est supérieur aux autres algorithmes étudiés, mais simplement qu'il performe mieux pour le type de problème pour lequel il a été conçu (la section 6.3 fait d'ailleurs ressortir une classe de problème pour lesquels l'AGENT offre

des performances passables). L'hypothèse de l'auteur est donc qu'un algorithme performant bien pour les problèmes de MSG offrira aussi de bonnes performances dans le cas des problèmes d'optimisation de la structure de pièces mécaniques, qui semblent à prime abord posséder une topologie similaire à celle des problèmes issus du MSG. Cette hypothèse sera validée sur un problème d'optimisation du design de disque de rotor de turbine à gaz à la section 7.14. Le reste de la section 6.1 explique les observations à propos des performances des divers autres algorithmes ayant contribué au développement de l'AGENT.

Résultats de l'AGO

Parmi les autres algorithmes étudiés, on observe que l'AGO et l'AGED sont ceux ayant procuré le moins de résultats intéressants. Pour l'AGO, le résultat était escompté. En effet, les mutations d'un AGO ne renversent qu'un seul codon d'un génome binaire, et ne sont donc pas particulièrement aptes à explorer un domaine constitué de nombres réels.

Résultats de l'AGED

Pour l'AGED, la problématique est un peu plus complexe. En observant les résultats, on constate que, selon le nombre de variables impliquées, l'AGED-2, qui utilise 4 îlots de population performe en moyenne 5-20% moins bien que l'AGED-1, qui n'en utilise qu'un seul. On peut donc supposer que, puisque les opérateurs de recherche de l'AGED n'imposent pas de mutations aléatoires (elles procurent plutôt un pas provenant d'un croisement différentiel), les performances de l'AGED pour la recherche globale dépendent beaucoup de la diversité des valeurs contenues dans la population initiale de l'algorithme. L'AGED-2 disposant de quatre populations moins diversifiées, il performe donc moins bien que l'AGED-1, qui dispose d'une seule population, mais quatre fois plus grande.

De même, on observe aussi que l'AGED semble se classer de moins en moins bien par rapport aux autres algorithmes à mesure que le nombre de variables augmente. Ceci est dû au fait que plus la complexité du problème augmente, plus il est difficile de posséder une grande diversité de valeurs adéquate pour chacune des variables dans la population. Ces résultats semblent donc indiquer que, pour les problèmes de grande envergure, l'AGED ne serait pas un candidat très approprié, puisque la population initiale nécessaire à avoir une bonne diversité initiale de gènes serait énorme.

Résultats de l'AGP2

Ensuite, on observe que les résultats utilisant l'AGP2 se classent de façon plutôt concentrée dans chacune des figures, mais que leur efficacité est parfois compétitive, parfois non, et que cela ne semble pas être influencé par le nombre de variables en présence (en effet, on observe de bons résultats pour 4, 8 et 25 variables, alors que les résultats sont de piètre qualité pour $N = 6$ ou 15 variables).

Pour comprendre le phénomène, il s'agit alors de s'intéresser de plus près à la composition du génome dans l'AGP2 (voir D.5). Ce que l'on constate, c'est que l'algorithme produit des valeurs de variables étant égales à des nombres fractionnaires, dont le numérateur et le dénominateur sont des nombres entiers. Ainsi, si la solution optimale du problème possède des valeurs qui sont très rapprochées d'un nombre fractionnaire simple pour chacune des variables (par exemple $x_i = 0{,}5$), l'AGP2 trouvera très rapidement une excellente solution, alors que si les problèmes générés aléatoirement disposent de solutions moins facilement constructibles par des fractions de nombres entiers, les résultats de l'AGP2 seront davantage passables.

Étant donné que l'on désire un algorithme capable d'une bonne recherche globale peu importe la valeur numérique des variables à l'optimum, et aussi parce que l'AGP2 requiert la manipulation de génomes dont la taille s'accroit de façon considérable avec le nombre de variables (2500 codons pour 25 variables dans le problème actuel), cet

algorithme ne semble pas particulièrement adapté aux problèmes de design structurel de taille industrielle.

Résultats de l'AGR et de l'AGR-NT

Les résultats démontrent aussi des performances acceptables pour l'AGR peu importe la dimensionnalité du problème étudié. Cet algorithme, ayant déjà fait ses preuves dans le cas de design structurel de taille industrielle (voir Song & Keane, 2005), semble capable de s'échapper des optimums locaux et de trouver la meilleure solution au problème si le nombre d'évaluations requises lui est alloué. Par contre, puisque le pas de mutation de l'algorithme est indépendant de l'état de convergence de la population, il s'avère en général plutôt long pour l'algorithme de découvrir l'optimum dans un premier temps, et ensuite relativement coûteux aussi de générer une mutation suffisamment petite pour permettre à l'AGR de raffiner la solution près de l'optimum trouvé.

En effet, on observe que, par rapport à l'AGR-NT, qui contrôle le pas de ses mutations, l'AGR donne généralement des résultats ayant une efficacité comparable mais un temps de convergence sensiblement plus long. Pour illustrer ce phénomène, la figure 6.10 illustre un exemple à deux dimensions de topologie gaussienne, et une évolution hypothétique mais représentative de la trajectoire de la meilleure solution de l'AGR par rapport à celle de l'AGR-NT.

Sur les figures 6.10 A et B, chaque point noir représente la solution de l'algorithme à certains moments de l'optimisation. Le point à côté duquel est inscrit *Départ* représente la solution initiale de l'algorithme, et le point intitulé *Arrivée* représente la solution finale à la fin de l'algorithme. Le cercle pointillé autour de chacun des points représente l'emplacement le plus probable de génération du point suivant par mutation. Plus un point potentiel se situe loin de cette ligne (que ce soit à l'intérieur ou à l'extérieur du cercle), plus les chances que ce point soit généré par mutation sont faibles.

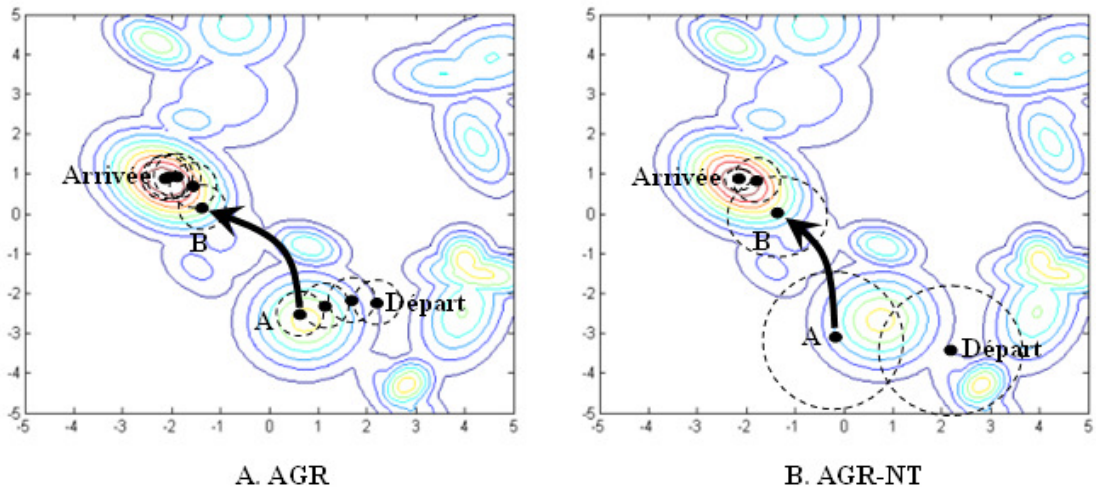


Figure 6.10 : Évolution d'algorithmes d'optimisation sur un problème de MSG typique.

On observe sur la figure 6.10 A qu'au début de l'algorithme, l'AGR tend à utiliser plusieurs évaluations inutilement afin de progresser vers un optimum local situé à proximité du point de départ. Ensuite, puisque la majorité des individus de l'algorithme finissent par se retrouver près de cet optimum local près de A, les chances que survienne un croisement ou une mutation à partir d'un point près de A vers un point B situé près de l'optimum global sont très minces.

Dans le cas de l'AGR-NT, on voit sur la figure 6.10 B que le pas moyen de déplacement des individus est beaucoup plus grand au début de l'algorithme. Ainsi, ses chances de repérer un point prometteur près de l'optimum global (de passer du point A au point B) sont beaucoup supérieures à celle de l'AGR.

Bien sûr, cette affirmation ne serait pas véridique si le pas d'avancement de l'AGR avait été choisit plus grand. Par contre, puisque les paramètres suggérés dans littérature pour l'AGR font que l'algorithme produit près de vingt fois plus de croisements que de mutations, le pas d'avancement de l'algorithme dépend beaucoup de l'emplacement de

ses individus et peu des choix de l'utilisateur. Par malchance, les individus de cet algorithme ont tendance à se regrouper rapidement près du meilleur individu de la population puisque c'est en général celui-là même qui produit la meilleure descendance (il est plus près de l'optimum).

Observons maintenant la figure 6.11, qui représente un agrandissement de l'évolution hypothétique des algorithmes près de l'optimum. On constate que l'algorithme devrait en principe effectuer des pas de plus en plus petits pour continuer à converger. Les mutations de l'AGR, qui utilisent un pas moyen constant, ne peuvent donc pratiquement plus aider une fois que l'algorithme est à une distance située à plusieurs ordres de grandeur plus petite que ce pas de l'optimum. L'AGR en est donc réduit à utiliser ses croisements, et dépend en conséquence beaucoup de la configuration de sa population pour déterminer s'il lui est possible ou non de continuer rapidement sa progression.

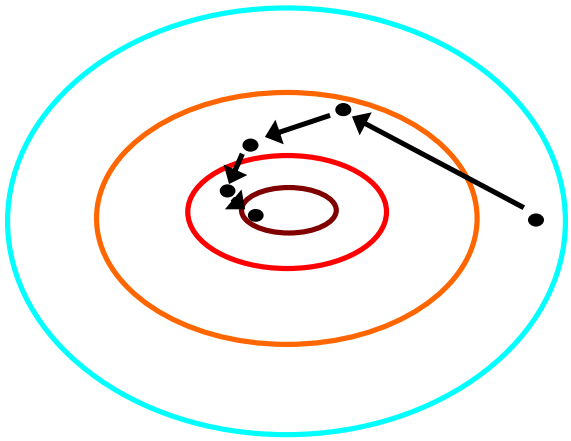


Figure 6.11 : Trajectoire typique du meilleur individu d'un AG à l'approche de l'optimum

Les croisements limitent la recherche de l'algorithme aux zones comprises entre les parents. Les parents font donc office de drapeaux indiquant à l'AG les régions potentiellement prometteuses. Par contre, limiter la recherche aux régions entre les parents sélectionnés limite beaucoup l'exploration globale du domaine. Ainsi, au lieu de supposer que tout le domaine ne contenant pas de parents est peu prometteur, ne serait-il

pas plus exact de supposer que les seules zones peu prometteuses sont celles dans les environs de piètres individus ayant déjà été évalués ?

C'est ce constat qui a amené l'idée des noyaux territoriaux. En effet, les mutations, elles, ne limitent pas réellement la recherche de l'algorithme. L'idée derrière les noyaux territoriaux est donc de concentrer les recherches autour des solutions prometteuses (les nouveaux individus utilisent uniquement la population active comme point de départ), mais sans négliger les zones où aucune solution n'a encore été évaluée (toutes les solutions antérieures servent à déterminer le territoire interdit aux nouveaux arrivants).

Pour ce qui est de l'AGR-NT, son pas de mutation diminue à mesure que l'algorithme progresse. Ceci lui permet d'explorer globalement le domaine au début de l'optimisation et de raffiner graduellement sa recherche autour des optimums trouvés. Si le pas est géré correctement, l'algorithme peut donc progresser vers l'optimum par mutations sans avoir à dépendre de la position des individus de sa population dans l'espace de résolution. Les cercles pointillés sur la figure 6.12 illustrent l'évolution du pas de mutation de l'AGR-NT à mesure que l'algorithme progresse vers l'optimum.

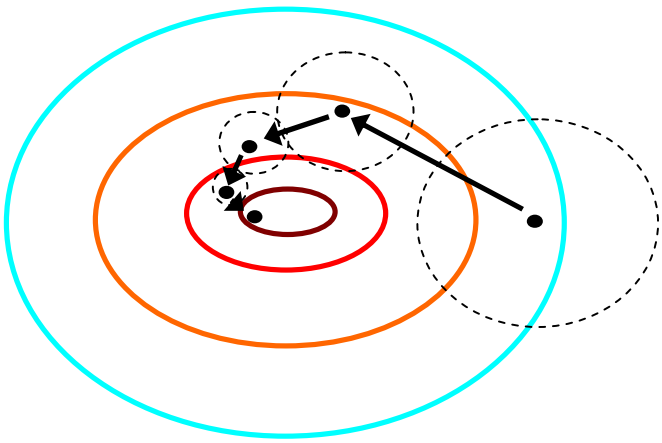


Figure 6.12 : Évolution du pas de l'AGR-NT à l'approche de l'optimum

Les résultats de l'AGR-NT sur les figures 6.3 à 6.9 démontrent le potentiel des noyaux territoriaux, se classant de façon très compétitive, peu importe le nombre de variables impliquées ou la structure de la solution au problème. Dans tous les cas étudiés, l'AGR-NT a procuré les solutions donnant en moyenne la meilleure efficacité par rapport aux autres algorithmes, à l'exception des problèmes à 6 et à 15 dimensions pour lesquels l'AGR a donné une solution plus efficace par le faible écart de 0,1%, mais en payant pour ce faire un prix de respectivement aux environs de 4% et 20% supérieur en terme de temps de convergence.

Par contre, l'AGR-NT, dispose aussi de plusieurs facteurs nuisant à sa convergence vers un optimum adéquat. D'abord, le pas des mutations de l'AGR-NT dépend du ratio entre le nombre d'évaluations qui ont été effectuées jusqu'à maintenant et le nombre total d'évaluations permises (équation 6.2) plutôt que de dépendre de l'état réel de convergence de l'algorithme. Ainsi, même si aucune solution décente n'est trouvée, ou que la topologie de la fonction semble promettre de meilleurs résultats un peu plus loin, l'algorithme raffinera tout de même continuellement ses pas, convergeant possiblement vers un optimum de piètre qualité. Cette lacune explique d'ailleurs la majeure portion de la différence de performance entre l'AGR-NT et l'AGENT. En effet, on se rappelle que ce dernier dispose d'un mécanisme permettant d'adapter le pas d'exploration des individus selon l'état de convergence de la fonction (voir équation 5.8).

Ensuite, les mutations de l'AGR-NT, comme celles de l'AGR, modifient systématiquement toutes les variables à la fois. Dans la réalité, il est rare que toutes les variables soient interdépendantes, et certaines directions de recherches n'utilisant pas systématiquement toutes les variables pourraient parfois s'avérer plus intéressantes que la mutation gaussienne qui modifie la plupart du temps toutes les variables par une valeur dont l'ordre de grandeur est également important. Encore une fois, l'AGENT inclut un mécanisme pour corriger ce défaut, inspiré de l'AGED, permettant de sélectionner aléatoirement les variables affectées par la mutation (voir section 5.5.1).

6.1.2 Choix des heuristiques ayant amené la construction de l'AGENT

Ce sont les défauts de l'AGR-NT, discutés précédemment, qui ont inspiré la création de l'AGENT tel que décrit dans le chapitre 5. De plus, les résultats de l'AGR-NT, plutôt prometteurs au départ, ont servi de fondation pour choisir le fonctionnement et les heuristiques contenues dans l'AGENT.

On constate, en observant les résultats, que l'AGR-NT-2, utilisant simplement l'heuristique des algorithmes insulaire, semble se classer chaque fois parmi les algorithmes les plus compétitifs. La figure 6.13 illustre l'efficacité moyenne de l'AGR-NT pour chaque combinaison d'heuristiques selon le nombre de variables du problème. On constate qu'à faible nombre de dimensions, l'efficacité de l'AGR-NT-2 se situe légèrement sous celle de l'AGR-NT-1 et de l'AGR-NT-5. Par contre, dès que le problème dépasse 4 variables, l'AGR-NT-2 semble être l'algorithme produisant les meilleures solutions.

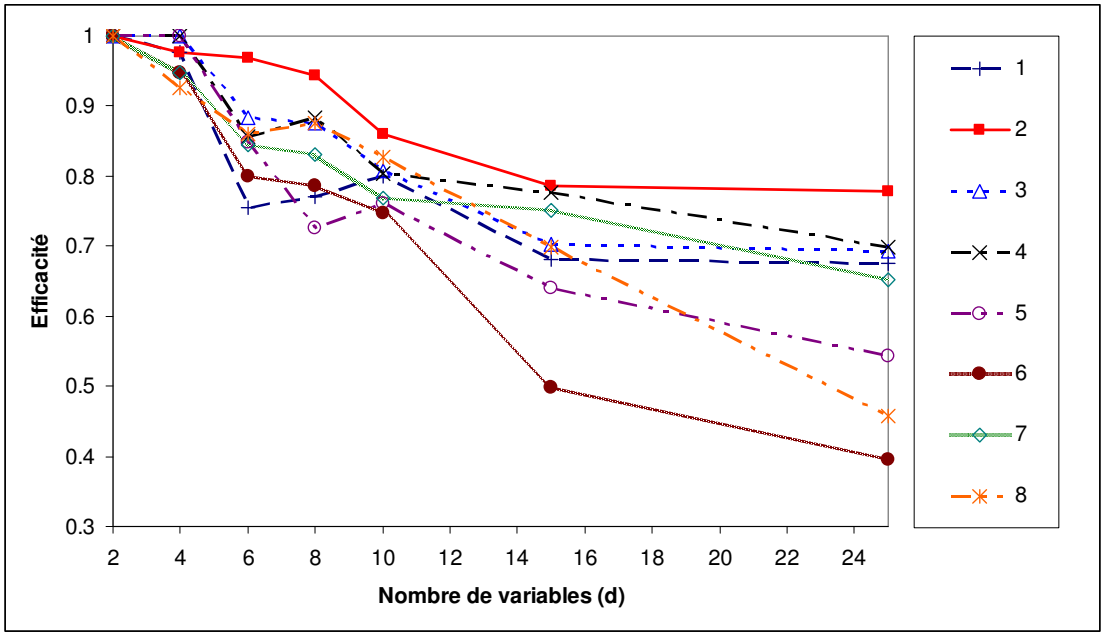


Figure 6.13 : Comparaison de l'efficacité des combinaisons d'heuristiques de l'AGR-NT

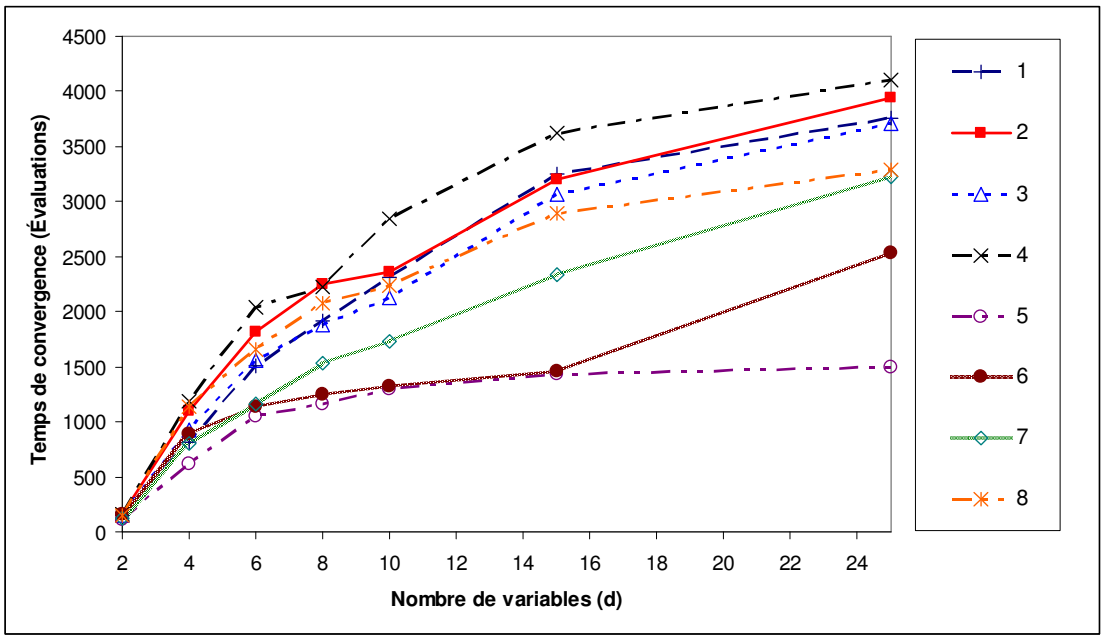


Figure 6.14 : Comparaison du temps de convergence des combinaisons d'heuristiques de l'AGR-NT

La figure 6.14, quant à elle, illustre le temps de convergence pour ces mêmes combinaisons d'heuristiques. On observe cette fois l'effet inverse, à savoir que l'AGR-NT-2 est l'un des algorithmes les plus lents à converger du lot. Toutefois, l'excellent temps de convergence obtenu pour les algorithmes donnant de piètres résultats signifie simplement que ces heuristiques procurent une convergence prématurée de l'algorithme vers un optimum local de faible qualité. Dans les faits, puisque le temps de convergence est défini le temps requis pour atteindre 95% de l'efficacité finale, il est fort probable qu'un algorithme ayant une grande efficacité ait trouvé une solution au moins aussi bonne au problème qu'un algorithme d'efficacité moindre en un même temps, même si son temps de convergence est inscrit comme supérieur. Le temps de convergence est donc davantage un outil permettant de comparer entre eux les algorithmes ayant une efficacité similaire.

À cet effet, on n'observe pas réellement de différence importante entre les temps de convergence des algorithmes les plus efficaces, soient l'AGR-NT-1, l'AGR-NT-2,

l'AGR-NT-3 et l'AGR-NT-4, si ce n'est que l'AGR-NT-2 converge en moyenne environ 5% moins rapidement que l'AGR-NT-3 et l'AGR-NT-4 et 5% plus rapidement que l'AGR-NT-1, qui est son plus proche compétiteur en efficacité.

La différence d'efficacité, toutefois, est très intéressante. L'AGR-NT-2 donnant en moyenne des solutions environ 10% meilleures que ses plus proches compétiteurs, on peut donc supposer que le taux d'application variable des opérateurs provoque une convergence prématurée de l'algorithme, alors que les espaces de croyances, agissant un peu comme des modèles substituts linéaires à la fonction coût, ne font en fait que suggérer des mauvaises solutions à l'algorithme et ne contribue pas véritablement à sa convergence pour le problème étudié.

Ainsi, puisque l'on s'intéresse d'abord et avant tout à construire un algorithme capable d'automatiser la recherche globale lors d'un processus de design, seuls les algorithmes insulaires semblent être intéressants pour l'algorithme génétique qui les utilise.

Pour les raisons décrites dans ce chapitre, l'AGENT a été construit sur le modèle d'un AGR, utilisant toutefois un opérateur de substitution additionnel, un pas de mutation variable selon l'état de convergence de l'algorithme, un opérateur de croisement légèrement différent et une mutation affectant un nombre aléatoire de variables. Suite aux bons résultats obtenus pour l'AGR-NT, l'AGENT utilisera l'heuristique des algorithmes insulaires, et appliquera la nouvelle heuristique développée, soit les noyaux territoriaux, de la façon décrite au chapitre 5.

À la suite de la discussion précédente, on peut conclure que l'AGENT semble s'avérer un algorithme compétitif dans le cas d'un problème ayant les mêmes caractéristiques que les problèmes de paysages gaussiens, soient une topologie multimodale légèrement discontinue et des variations plutôt douces de la solution selon la variation des paramètres.

6.1.3 Avantages de l'opérateur de substitution pour l'AGENT

Tel que mentionné à la section 5.5.3, l'AGENT utilise un opérateur de substitution utilisant un réseau de neurones pour remplacer un nombre N_{RN} d'individus de la population par d'autres individus potentiellement prometteurs à proximité des individus remplacés.

L'approche suggérée généralement dans la littérature (voir Ratle, 1998) est un cas particulier de cet opérateur pour lequel $N_{RN} = N_{pop}$, la taille de la population de l'algorithme. Cette section s'intéresse donc à justifier le choix de $N_{RN} = 1$ utilisé par l'AGENT, en comparant les résultats obtenus pour le problème de paysage gaussien en utilisant différents N_{RN}.

On se rappelle que les résultats présentés précédemment aux sections 6.1.1 et 6.1.2 pour le problème de MSG n'utilisaient pas de réseaux de neurones substituts à la fonction coût. Il importe donc de savoir à prime abord si l'utilisation d'un réseau de neurones peut s'avérer intéressante ou non pour l'algorithme. La figure 6.15 compare l'efficacité obtenue pour l'AGENT sans opérateur de substitution avec les résultats obtenus en utilisant la même méthodologie qu'à la section 6.1.1, mais avec respectivement des opérateurs de substitution tels que suggérés pour l'AGENT ($N_{RN} = 1$) ou par Ratle (1998) ($N_{RN} = N_{pop}$).

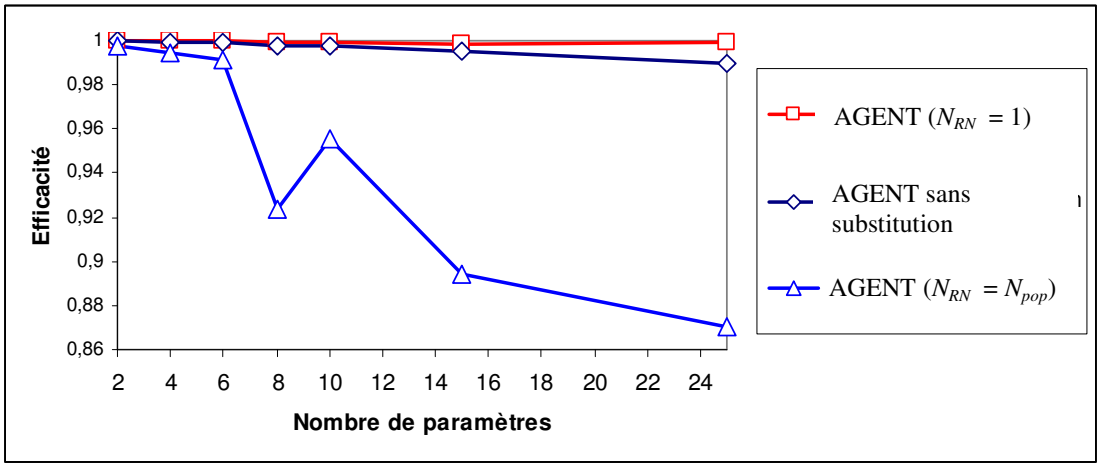


Figure 6.15 : Comparaison de l'efficacité moyenne des AGENT selon leur utilisation de l'opérateur de substitution.

On constate que pour le problème du MSG, l'AGENT obtient en moyenne les meilleurs résultats en utilisant un opérateur de substitution avec $N_{RN} = 1$, qui effectue simplement une translation d'un point de la population vers le point le plus prometteur suggéré des environs. En effet, lorsque $N_{RN} = N_{pop}$, N_{pop} individus sont évalués à chaque itération par l'opérateur de substitution, provoquant plusieurs évaluations inutiles de points si le réseau de neurones n'est pas très précis, comme c'est le cas lorsque le nombre de paramètres augmente pour un même nombre d'évaluations. On constate d'ailleurs une discontinuité inhabituelle des résultats pour $N_{RN} = N_{pop}$ lorsque $N = 8$ paramètres, probablement due au fait que plusieurs des 10 problèmes générés automatiquement pour $N = 8$ dispose d'une topologie plus difficile à reproduire par les additions de fonction sigmoïdales des perceptrons, diminuant de ce fait la qualité des réseaux de neurones.

La figure 6.16, quant à elle, présente le temps de convergence requis pour atteindre 95% de l'efficacité finale pour les problèmes gaussiens présentés selon l'AGENT utilisé. On y observe qu'en plus d'être plus efficace, l'AGENT utilisant un opérateur de substitution avec $N_{RN} = 1$ est aussi plus rapide de convergence, le rendant meilleur sur tous les aspects pour les problèmes de paysages gaussiens étudiés.

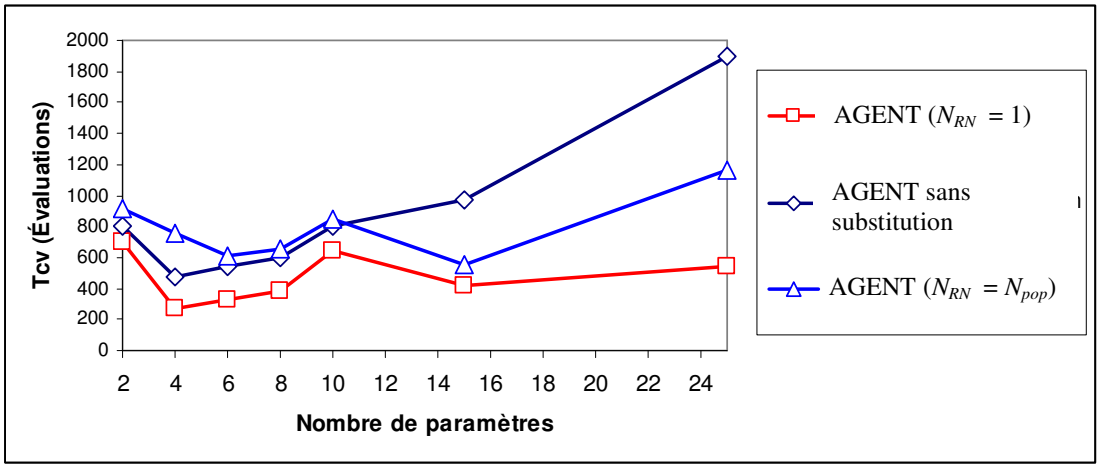


Figure 6.16 : Comparaison du temps de convergence moyen des AGENT selon leur utilisation de l'opérateur de substitution.

En effet, l'AGENT utilisant N_{RN} = 1 profite du savoir des réseaux de neurones substitut lorsque celui-ci est intéressant, tout en ne perdant qu'une seule évaluation par génération lorsque les réseaux ne donnent pas de résultats concluants. La figure 6.17 compare l'évolution de deux résultats typique selon que l'AGENT utilise ou non son opérateur de substitution pour N = 15 variables. Sans substitution, l'AGENT n'est ni plus ni moins qu'un algorithme de recherche aléatoire, d'où la progression en escalier observée. Toutefois, avec un réseau de neurones substitut, l'utilisation adéquate du savoir accumulé sur la topologie de la fonction peut s'avérer bénéfique dans le cas de l'exploration de fonctions ne présentant pas trop de discontinuités comme c'est le cas des paysages gaussiens, d'où la progression plus lisse observée lors des 300 premières évaluations.

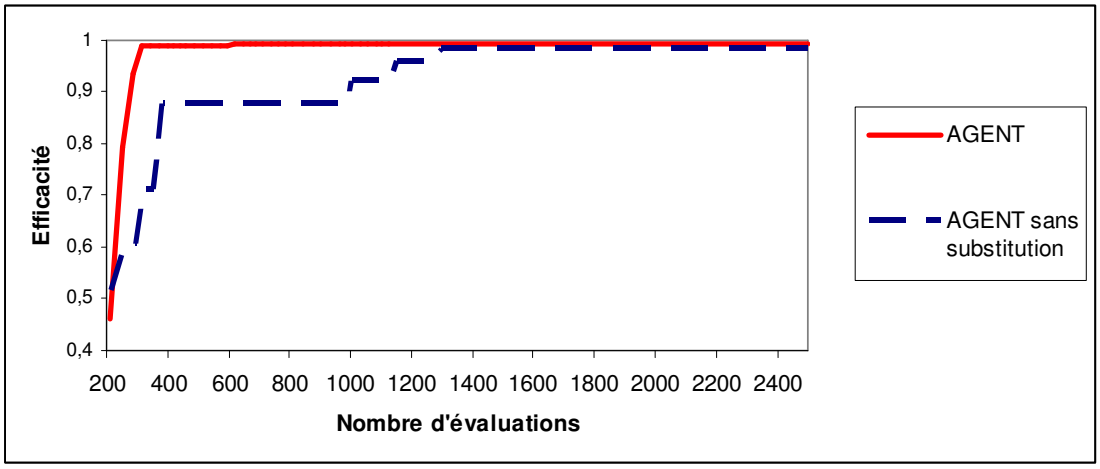


Figure 6.17 : Comparaison de deux courbes de résultats de l'AGENT, selon qu'il utilise ou non l'opérateur de substitution.

6.1.4 Utilisation des algorithmes insulaires pour l'AGENT

Tel que décrit à la section 5.7, l'AGENT utilisera une adaptation des algorithmes insulaires dont l'avantage principal est de ne pas nécessiter l'ajustement des deux paramètres I_M et N_M.

Dans les tests de la section 6.1.1, l'AGENT utilisait plutôt l'algorithme insulaire directement de la façon prescrite par la littérature (Cantu-Pax, 1998). Cette section de la thèse s'intéressera donc à démontrer que l'adaptation proposée pour les algorithmes insulaires donne au moins d'aussi bons résultats que l'utilisation régulière des algorithmes insulaires pour l'AGENT, mais en nécessitant l'ajustement de deux paramètres en moins.

Les figures 6.18 et 6.19 comparent respectivement les efficacités et les temps de convergence des résultats obtenus par la méthodologie de la section 6.1.1 pour l'AGENT utilisant soit des algorithmes insulaires tels que décrits pour l'AGENT à la section 5.7, soit des algorithmes insulaires tels que décrits dans la littérature par Cantu-Pax (1998) en posant les valeurs typiques des paramètres $I_M = 10$ et $N_M = 2$.

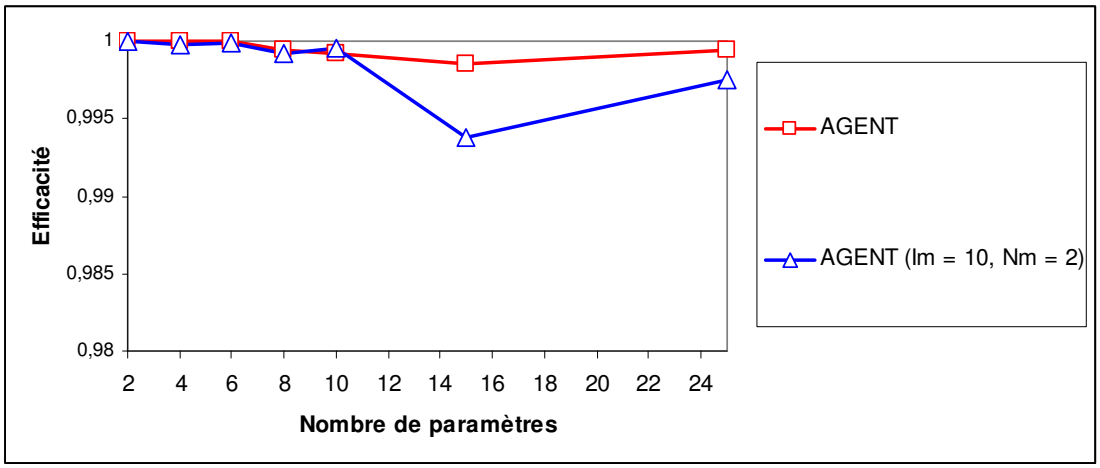


Figure 6.18 : Efficacité moyenne de l'AGENT selon le type d'algorithme insulaire utilisé

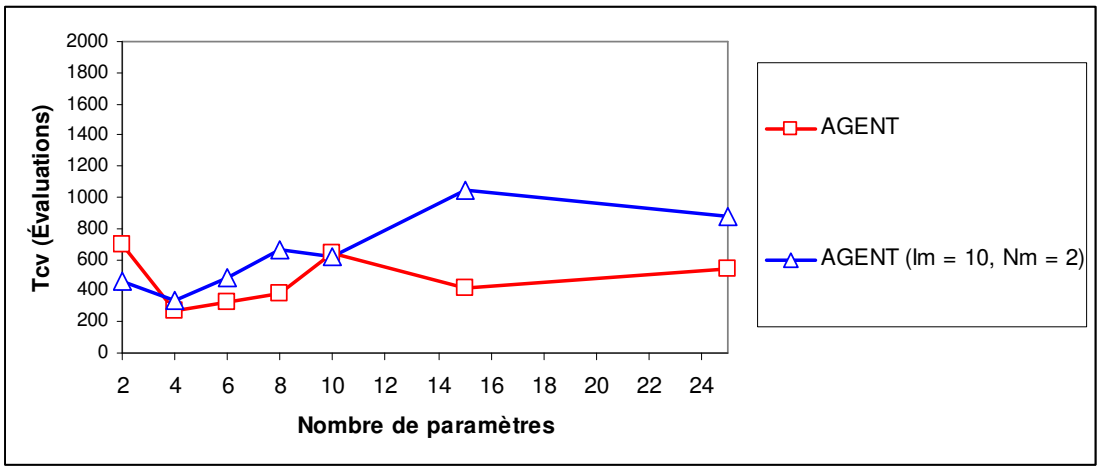


Figure 6.19 : Temps de convergence moyen de l'AGENT selon le type d'algorithme insulaire utilisé

On y constate essentiellement peu de différences, sinon que l'AGENT utilisant la version sans paramètres des algorithmes insulaires semble un peu plus efficace et un peu plus rapide lorsque le nombre de paramètres s'accroit.

D'ailleurs, de simples tests statistiques démontrent que pour tous les nombres de dimensions, les résultats utilisant la version originale proposée pour gérer les migrations entre les différents îlots sont, avec 95% de certitude, au moins aussi efficaces, lorsqu'ils ne le sont pas davantage, que les résultats utilisant les paramètres I_M = 10 et N_M = 2 communément employés pour les algorithmes insulaires dans la littérature.

En conclusion, la méthode décrite à la section 5.7 pour gérer les migrations entre les îlots des algorithmes insulaires semble compétitive pour les problèmes s'apparentant aux paysages gaussiens, tout en apportant l'avantage de disposer de deux paramètres en moins à définir pour l'utilisateur, améliorant de ce fait l'aspect automatique du procédé.

6.2 Comparaison entre l'AGENT et d'autres algorithmes génétiques pour des cas tests avec contraintes de nature académique.

Dans leur article, Liu, Ma, Zhang & Zhou (2005) décrivent deux méthodes populaires de gestion des contraintes, ainsi que les résultats de ces méthodes pour plusieurs cas tests.

Les méthodes de gestion de contraintes utilisées sont :

- la pénalité statique (PS), où la pénalité *v* est une constante.
- les algorithmes co-évolutionnaires (CO) , qui utilisent deux îlots de population, l'un pour lequel la valeur adaptative est la fonction coût sans contraintes G($\vec{x}$), l'autre pour lequel la valeur adaptative est en fait uniquement fonction des contraintes H($\vec{x}$).

Chacune de ces deux méthodes a été testée pour un algorithme génétique régulier de type AGR et pour un algorithme génétique de type AGED.

De plus, un nouvel algorithme nommé « algorithme mimétique à évolution différentielle co-évolutionnaire » (MCODE), qui applique une mutation gaussienne à une population entière lorsqu'elle reste inchangée durant un certain nombre d'itérations, est comparé avec les autres.

Ainsi, cet article présente un total de cinq algorithmes gérant différemment les contraintes, soit PS-AGR, PS-AGED, CO-AGR, CO-AGED et MCODE.

Dans un autre article, Michalewicz & Shoenauer (1997) présentent plusieurs méthodes de gestion de contraintes bien connues pour un algorithme génétique régulier de type AGR. Ces méthodes sont :

- la peine de mort (PM), telle que décrite à la section 3.1.
- la pénalité statique (PS), où v est conservé constant.
- la pénalité dynamique (PD), où la pénalité v est une fonction qui augmente à mesure que l'algorithme converge.
- la pénalité adaptative (PA), qui est la fonction de mérite utilisant v utilisée dans ce rapport (section 3.1).
- la pénalité de recuit simulé (PR), où v est une fonction qui augmente à mesure que l'algorithme converge, et où l'algorithme se réinitialise autour de la meilleure solution après chaque évolution.
- la méthode de gestion des contraintes à mémoire comportementale (PMC), où la valeur adaptative est d'abord uniquement une pénalité qui s'applique d'abord à une seule contrainte, jusqu'à ce qu'un certain nombre d'individus satisfassent cette contrainte. Ensuite, on réinitialise l'algorithme autour des individus satisfaisant cette contrainte, on ajoute une contrainte et on recommence jusqu'à ce qu'un certain nombre de solutions faisables existent pour toutes les contraintes. À ce stade, la valeur adaptative inclut aussi la fonction coût en plus des pénalités et l'optimisation se résout normalement.

- la méthode de supériorité des points réalisable (SPR), où la pénalité est ajustée de façon à ce que les points irréalisables aient toujours une valeur adaptative inférieure à celle des points réalisables.

Le lecteur intéressé pourra trouver le détail sur chacune des méthodes de gestion de contrainte dans les articles concernées. Pour les besoins de cette thèse, nous nous contenterons d'affirmer que ces méthodes ont été testées sur les problèmes suivants :

Problème G_0 :

Minimiser : $G(x) = -a_1 - a_2$

Sujet à : $a_2 - 2a_1^4 + 8a_1^3 - 8a_1^2 - 2 \le 0$

$a_2 - 4a_1^4 + 32a_1^3 - 88a_1^2 - 96a_1 - 36 \le 0$

Avec : $0 \le a_1 \le 3, 0 \le a_2 \le 4$

Optimum : $\vec{a}^* = (2{,}32952024 ; 3{,}17849288),\ G^* = -5{,}508013271$

Problème G_1

Minimiser : $G(\vec{a}) = 5a_1 + 5a_2 + 5a_3 + 5a_4 - 5\sum_{i=1}^{4} a_i^2 - \sum_{i=5}^{13} a_i$

Sujet à : $2a_1 + 2a_2 + a_{10} + a_{11} \le 10$ $\quad$ $2a_1 + 2a_3 + a_{10} + a_{12} \le 10$

$2a_2 + 2a_3 + a_{11} + a_{12} \le 10$ $\quad$ $-8a_1 + a_{10} \le 0$

$-8a_2 + a_{11} \le 0$ $\quad$ $-8a_3 + a_{12} \le 0$

$-2a_4 - a_5 + a_{10} \le 0$ $\quad$ $-2a_6 - a_7 + a_{11} \le 0$

$-2a_8 - a_9 + a_{12} \le 0$

Avec : $0 \le a_i \le 1$ $\quad$ pour i = 1, 2…9

$0 \le a_i \le 100$ $\quad$ pour i = 10,11,12

$0 \le a_{13} \le 1$

Optimum : $\vec{a}^* = (1;1;1;1;1;1;1;1;1;3;3;3;1)$, $G^* = -15$

Problème G_6

Minimiser : $G(\vec{a}) = (a_1 - 10)^3 + (a_2 - 20)^3$

Sujet à : $(a_1 - 5)^2 + (a_2 - 5)^2 - 100 \geq 0$

$-(a_1 - 6)^2 - (a_2 - 5)^2 + 82{,}81 \geq 0$

Avec : $13 \leq a_1 \leq 100$ $\quad$ $0 \leq a_2 \leq 100$

Optimum : $\vec{a}^* = (14{,}095 ; 0{,}84296)$ $\quad G^* = -6961{,}81381$

Problème G_7

Minimiser : $G(\vec{x}^*) = a_1^2 + a_2^2 + a_1 a_2 - 14a_1 - 16a_2 + (a_3 - 10)^2 + 4(a_4 - 5)^2 +$

$(a_5 - 3)^2 + 2(a_6 - 1)^2 + 5a_7^2 + 7(a_8 - 11)^2 + 2(a_9 - 10)^2 +$

$(a_{10} - 7)^2 + 45$

Sujet à : $105 - 4a_1 - 5a_2 + 3a_7 - 9a_8 \geq 0$

$-3(a_1 - 2)^2 - 4(a_2 - 3)^2 - 2a_3^2 + 7a_4 + 120 \geq 0$

$-10a_1 + 8a_2 + 17a_7 - 2a_8 \geq 0$

$-a_1^2 - 2(a_2 - 2)^2 + 2a_1 a_2 - 14a_5 + 6a_6 \geq 0$

$8a_1 - 2a_2 - 5a_9 + 2a_{10} + 12 \geq 0$

$-5a_1^2 - 8a_2 - (a_3 - 6)^2 + 2a_4 + 40 \geq 0$

$3a_1 - 6a_2 - 12(a_9 - 8)^2 + 7a_{10} \geq 0$

$-0.5(a_1 - 8)^2 - 2(a_2 - 4)^2 - 3a_5^2 + a_6 + 30 \geq 0$

Avec : $-10.0 \leq a_i \leq 10.0$ $\quad$ pour i = 1,2,…,10

Optimum : $\vec{a}^* = (2{,}171996 ; 2{,}36368 ; 8{,}773926 ; 5{,}095984 ; 0{,}9906548 ;$

$1{,}430574 ; 1{,}321644 ; 9{,}828716 ; 8{,}280092 ; 8{,}375927)$

$G^* = 24{,}3062091$

Problème G_9

Minimiser : $G(\vec{x}) = (a_1 - 10)^2 + 5(a_2 - 12)^2 + a_3^4 + 3(a_4 - 11)^2 + 10a_5^6 + 7a_6^2 + a_7^4 - 4a_6a_7 - 10a_6 - 8a_7$

Sujet à : $127 - 2a_1^2 - 3a_2^4 - a_3 - 4a_4^2 - 5a_5 \geq 0$

$282 - 7a_1 - 3a_2 - 10a_3^2 - a_4 + a_5 \geq 0$

$196 - 23a_1 - a_2^2 - 6a_6^2 + 8a_7 \geq 0$

$-4a_1^2 - a_2^2 + 3a_1a_2 - 2a_3^2 - 5a_6 + 11a_7 \geq 0$

Avec : $-10.0 \leq a_i \leq 10.0$ pour i = 1 à 7

Optimum : $\vec{a}^* = (2{,}330499 ; 1{,}951372 ; -0{,}4775414 ; 4{,}365726 ; -0{,}6244870 ; 1{,}038131 ; 1{,}594227)$

$G^* = 680{,}6300573$

Problème G_{10}

Minimiser : $G(\vec{a}) = a_1 + a_2 + a_3$

Sujet à : $1 - 0{,}0025(a_4 + a_6) \geq 0$

$1 - 0{,}0025(a_5 + a_7) \geq 0$

$1 - 0{,}01(a_8 - a_5) \geq 0$

$a_1a_6 - 833{,}33252a_4 - 100a_1 + 83333{,}333 \geq 0$

$a_2a_7 - 1250a_5 - a_2a_4 + 1250a_4 \geq 0$

$a_3a_8 - 1250000 - a_3a_5 + 2500a_5 \geq 0$

Avec : $100 \leq a_1 \leq 10000$

$1000 \leq a_i \leq 10000$ pour i = 2,3,

$10 \leq a_i \leq 1000$ pour i = 4 à 8

Optimum : $\vec{a}^* = (579{,}3167 ; 1359{,}943 ; 5110{,}071 ; 182{,}0174 ; 295{,}5985 ; 217{,}9799 ; 286{,}4162 ; 395{,}5979)$

$$G^* = 7049{,}330923$$

Résultats

La table 6.4 affiche les algorithmes qui ont été testés sur chacun des problèmes et les résultats moyens obtenus après 50 lancements de l'algorithme. La figure 6.20, quant à elle, affiche la différence moyenne obtenue entre la valeur adaptative finale et l'optimum. Les algorithmes ont utilisé une population de N_{pop} = 40 et un nombre d'évaluations maximum égal à 300 000.

Seuls les résultats pour l'AGENT ont été obtenus par l'auteur dans le cadre de cette thèse. Les autres résultats sont tirés de Michalewicz & Shoenauer (1997) et de Liu, Ma, Zhang & Zhou (2005). Les résultats inscrits pour l'AGENT ont été obtenus avec la même population et le même nombre d'évaluations que les autres algorithmes. Les valeurs utilisées pour ses paramètres sont celles suggérées par défaut dans cette thèse (soit $\gamma_c^{\,0} = 0{,}2$ et $\alpha_m = 0{,}5$), ainsi qu'une taille de population totale limitée à N_{max} = 5000.

Table 6.4. Résultats moyens obtenus pour les diverses méthodes de gestion de contraintes

	G_0	G_1	G_6	G_7	G_9	G_{10}
Optimum	-5,508	-15,000	-6961,814	24,306	680,630	7049,331
AGENT	-5,508	-15,000	-6961,814	24,332	680,725	7182,145
PS-AGR	-5,954	-13,7597	-7233,572	--	685,9112	--
PS-AGED	-5,562	-14,9583	-6967,597	--	683,0213	--
CO-AGR	-5,634	-14,9971	-6962,332	--	681,2550	--
CO-AGED	-5,508	-15	-6961,403	--	680,7413	--
MCODE	-5,508	-15	-6961,587	--	680,6759	--
PM	--	-14,999	--	27,116	681,771	--
PS	--	-15,002	--	29,258	681,262	2449,798
PD	--	-15,000	--	26,905	681,111	4213,497
PR	--	-15,000	--	24,418	680,718	8206,151
PMC	--	-15,000	--	--	681,175	8271,292
SPR	--	-15,000	--	22,932	682,682	2101,411

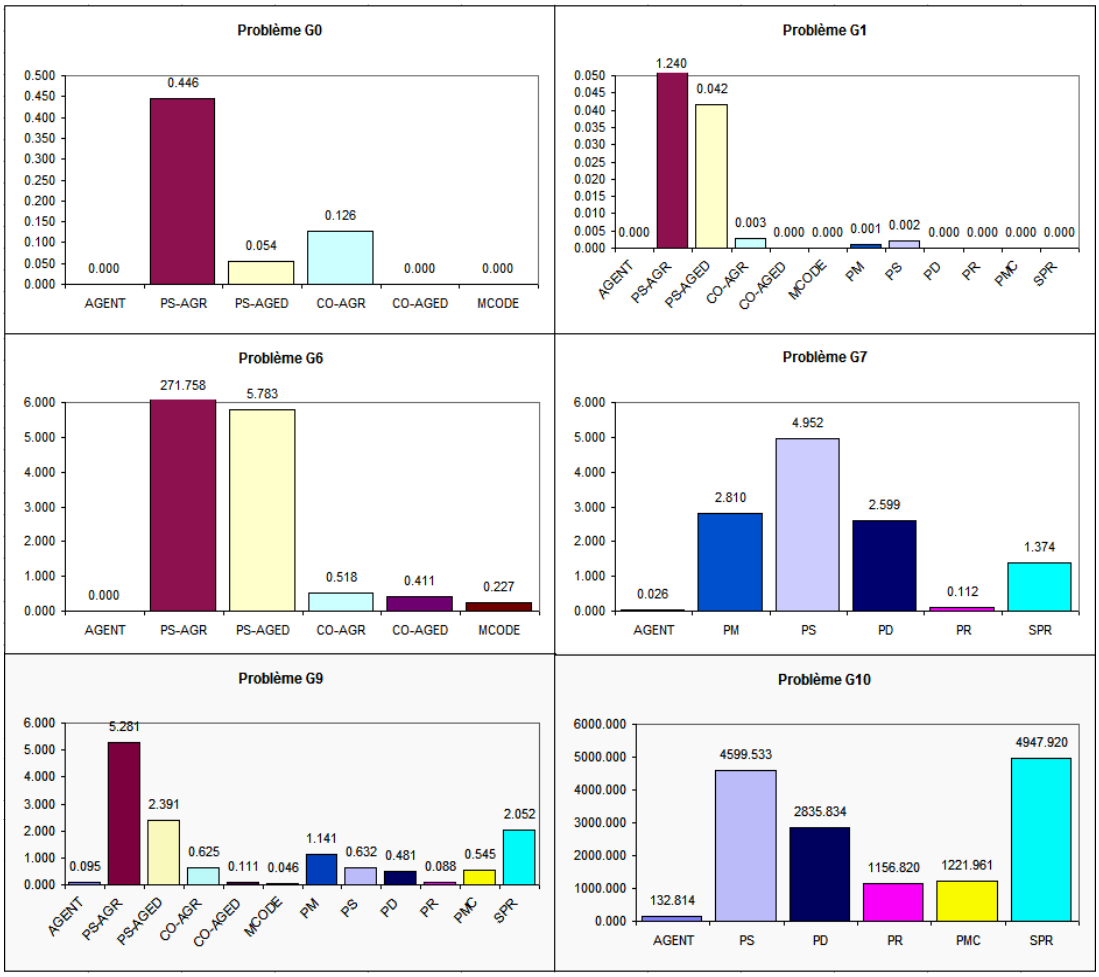


Figure 6.20. Éloignement moyen de la valeur adaptative optimale pour chacun des problèmes

Sur la figure 6.20, on peut constater que l'AGENT performe de façon très compétitive pour les problèmes d'optimisation avec contraintes.

En effet, il a donné les meilleurs résultats pour tous les problèmes, à l'exception du problème G_9 où il s'est classé 3ème, derrière les algorithmes MCODE et PR qui l'ont devancé respectivement de la faible valeur de 0,049 et de 0,07.

Pour les problèmes simples G_0 et G_1, l'AGENT n'est pas le seul à avoir donné une solution à moins de 0,0005 de la valeur optimale, et il se classe donc à égalité avec CO-AGED et MCODE pour le problème G_0, et à égalité avec CO-AGED, MCODE, PD, PR, PMC et SPR pour le problème G_1.

Par contre, pour les problèmes G_6, G_7 et G_{10}, l'AGENT a donné chaque fois un résultat supérieur à tous les autres résultats pris dans la littérature, laissant supposer que l'AGENT, avec sa méthode originale hybride de gestion de contraintes, est un candidat potentiellement intéressant pour la résolution de problèmes d'optimisation avec contraintes.

La section suivante présente d'autres séries de cas tests démontrant les performances de l'AGENT, dans le cas cette fois de l'optimisation non contrainte. L'AGENT est aussi appliqué à un problème contraint de taille industrielle au chapitre 7.

6.3 Comparaison entre l'AGENT et d'autres algorithmes génétiques pour des cas tests sans contraintes de nature académique.

On se rappelle que la section 6.2 présente plusieurs cas tests avec contraintes dans lesquels l'AGENT performe de façon très compétitive. De plus, la section 6.1 de ce document démontre d'excellents résultats pour l'AGENT pour les problèmes issus du MSG.

Dans la section 6.3.1, afin d'évaluer aussi les faiblesses de l'AGENT, l'algorithme a été comparé à plusieurs autres algorithmes génétiques populaires pour divers cas tests typiques sans contraintes de nature académiques.

La section 6.3.2 discute de l'une des différences majeures entre l'AGENT et les autres algorithmes évolutionnaires, qui est une évolution basée sur la mutation au lieu du croisement. On y explique pourquoi cela affecte les résultats de l'AGENT pour les cas tests typiquement utilisés en optimisation non contrainte. Plusieurs cas tests, toujours tirés de la littérature, y sont présentés pour lesquels le biais conféré aux autres algorithmes dans les cas tests typiques est transféré du côté de l'AGENT, qui y performe alors de façon supérieure.

6.3.1 Comparaison pour une série de tests typiques

Bien qu'une multitude d'algorithmes soient disponibles dans la littérature, l'auteur s'est limité à une étude comparative comprenant les algorithmes plus communément utilisés, soient les AGR (voir section D.3), les AGED (voir D.4) et certains de leurs dérivés récents.

Pour les AGR, Ortiz-Boyer, Hervás-Martínez & Gracia-Pedrajas (2005) comparent entre autres les résultats obtenus pour des AGR utilisant les opérateurs de croisement UNDX,

BLX et SBX à un nouvel opérateur de croisement basé sur la distribution statistique des individus nommé CIXL2. Dans le CIXL2, les nouveaux individus prometteurs transmettent les mesures de dispersion et de localisation à leurs enfants, de façon à ce que le pas et la direction des croisements soient davantage contrôlés.

Dans un autre article, Noman & Iba (2008) comparent les résultats obtenus pour l'AGED régulier et pour un algorithme que l'on appellera AGED-RL (algorithme génétique à évolution différentielle avec recherche locale), qui recherche la meilleure contribution à appliquer pour chacun des parents à un enfant lors d'une opération de croisement plutôt que de choisir cette contribution de façon arbitraire. Ce dernier algorithme est d'ailleurs, parmi les algorithmes aperçus dans la littérature dans le cadre de cette thèse, l'un de ceux ayant produit les meilleurs résultats pour les problèmes d'optimisation sans contraintes

Ces comparaisons ont été effectuées sur un ensemble de fonctions tests à 30 variables tirées de l'ensemble très connu conçu par Eiben & Bäck (1997), auxquelles s'ajoutent la fonction de Rosenbrock (1960) et la fonction de Schwefel (1981). La table 6.5 présente les diverses fonctions tests utilisées.

Table 6.5. Fonctions tests pour N = 30 variables

Fonction	Définition	Multi-modale	Séparable
Sphère	$F_{Sph}(\vec{x}) = \sum_{i=1}^{N} x_i^2$ $x_i \in [-5{,}12;5{,}12]$ $x^* = (0,0,...,0) \qquad F_{Sph}{}^* = 0$	Non	Oui
Double somme de Schwefel	$F_{SchDS}(\vec{x}) = \sum_{i=1}^{N} \left(\sum_{j=1}^{i} x_j \right)^2$ $x_i \in [-65{,}536;65{,}536]$ $x^* = (0,0,...,0) \qquad F_{SchDS}{}^* = 0$	Non	Non
Rosenbrock	$F_{Ros}(\vec{x}) = \sum_{i=1}^{N-1} \left[100\left(x_{i+1} - x_i^2\right)^2 + (x_i - 1) \right]$ $x_i \in [-2{,}048;2{,}048]$ $x^* = (1,1,...,1) \qquad F_{Ros}{}^* = 0$	Oui (si N >4)	Non
Rastrigin	$F_{Ras}(\vec{x}) = 10N + \sum_{i=1}^{N} \left(x_i^2 - 10\cos(2\pi x_i) \right)$ $x_i \in [-5{,}12;5{,}12]$ $x^* = (0,0,...,0) \qquad F_{Ras}{}^* = 0$	Oui	Oui
Schwefel	$F_{Sch}(\vec{x}) = 418{,}9829N + \sum_{i=1}^{N} x_i \sin\left(\sqrt{\lvert x_i \rvert}\right)$ $x_i \in [-512{,}03;511{,}97]$ $x^* = (-420{,}9687,-420{,}9687,...,-420{,}9687) \qquad F_{Sch}{}^* = 0$	Oui	Oui
Ackley	$F_{Ack}(\vec{x}) = 20 + e - 20\exp\left(-0{,}2\sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} \right) - \exp\left(\frac{1}{N}\sum_{i=1}^{N} \cos(2\pi x_i) \right)$ $x_i \in [-30;30]$ $x^* = (0,0,...,0) \qquad F_{Ack}{}^* = 0$	Oui	Non
Griewangk	$F_{Gri}(\vec{x}) = 1 + \sum_{i=1}^{N} \frac{x_i^2}{4000} - \prod_{i=1}^{N} \cos\left(\frac{x_1}{\sqrt{i}} \right)$ $x_i \in [-600;600]$ $x^* = (0,0,...,0) \qquad F_{Gri}{}^* = 0$	Oui	Non

Les résultats moyens et l'écart type obtenus pour chacun des algorithmes sont illustrés dans la table 6.6 après 20 lancements de 300 000 évaluations chacun. Les paramètres de chacun des algorithmes testés sont décrits plus en détail dans les articles correspondants, soit Ortiz-Boyer, Hervás-Martínez & Gracia-Pedrajas (2005) pour l'AGR-UNDX, l'AGR-BLX, l'AGR-SBX et l'AGR-CIXL2, ainsi que dans Noman & Iba (2008) pour l'AGED et l'AGED-RL. Quant aux paramètres utilisés par l'AGENT, ce sont ceux par défaut pour cet algorithme, décrits dans la table 6.3, avec 4 îlots contenant chacun de 40 individus et N_{max} = 5000. Pour les fonctions multimodales, afin de promouvoir la recherche globale, la valeur de créativité de l'AGENT a toutefois été posée à γ_c^o = 0,5. Aucun de ces algorithmes n'utilise de modèle substitut à la fonction. Ainsi, afin de comparer les algorithmes sur une même base, notez que l'opérateur de substitution, donnant l'avantage d'un modèle substitut à l'AGENT, a été retiré temporairement de l'algorithme pour effectuer ces tests.

Table 6.6. Comparaison entre la solution moyenne obtenue par l'AGENT et par divers algorithmes pour un ensemble de fonctions test à 30 variables.

Fonction	AGENT	AGR-UNDX	AGR-CIXL2	AGR-BLX	AGR-SBX	AGED	AGED-RL
Sphère	2,016E-26 ± 5,367 E-24	2,91E-05 ± 1,43E-05	6,365E-16 ± 2,456E-16	3,257E-16 ± 1,396E-16	1,645E-12 ± 8,874E-13	5,73E-17 ± 2,03E-16	1,75E-31 ± 4,99E-31
Double somme de Schwefel	2,038E-03 ± 1,108E-03	2,080E+01 ± 7,216E+00	1,995E-03 ± 2,280E-03	9,332E-03 ± 1,086E-02	2,033E-01 ± 1,966E-01	--	--
Rosenbrock	2,710E+01 ± 4,082E00	2,840E+01 ± 3,606E-01	2,494E+01 ± 1,283E+00	2,923E+01 ± 1,723E+01	2,775E+01 ± 9,178E+01	5,20E+01 ± 8,56E+01	4,52E+00 ± 1,55E+01
Rastrigin	2,129E+01 ± 1,710E+01	1,107E+02 ± 1,242E+01	2,919E+00 ± 1,809E+00	2,189E+00 ± 1,417E+00	1,419E+01 ± 3,704E+00	2,55E+01 ± 8,14E+00	2,14E+01 ± 1,23E+01
Schwefel	2,570E+03 ± 3,456E+02	8,050E+03 ± 3,741E+02	6,410E+02 ± 2,544E+02	3,695E+02 ± 1,595E+02	1.104E+03 ± 3,353E+02	4,90E+02 ± 2,34E+02	4,70E+02 ± 2,96E+02
Ackley	6,341E-04 ± 2,561E-04	3,551E-02 ± 1,224E-02	1,378E-08 ± 5,677E-09	4,207E-08 ± 1,713E-08	5,335E-06 ± 1,453E-06	1,37E-09 ± 1,32E-09	1,66E-15 ± 0,00E-00
Griewangk	1,109E-05 ± 7,554E-05	7,837E-02 ± 4,438E-02	1,525E-02 ± 1,387E-02	3,760E-02 ± 2,874E-02	2,196E-02 ± 1,874E-02	2,66E-03 ± 5,73E-03	2,07E-03 ± 5,89E-03

On constate qu'en général, pour ces problèmes d'optimisation sans contraintes, l'AGENT performe correctement, mais n'est pas du tout supérieur aux autres algorithmes étudiés. Son classement pour chacun des problèmes est le suivant :

- Sphère : 2nd — derrière AGED-RL
- Double somme de Shwefel : 2nd — derrière AGR-CIXL2
- Rosenbrock : 3ème — derrière AGR-CIXL2 et AGED-RL
- Rastrigin : 4ème — derrière AGR-SBX, AGR-CIXL2 et AGR-BLX
- Schwefel : 6ème — derrière AGR-SBX, AGR-CIXL2, AGED, AGED-RL et AGR-BLX
- Ackley : 6ème — derrière AGR-SBX, AGR-CIXL2, AGR-BLX, AGED et AGED-RL
- Griewangk : 1er

Fonctions unimodales

Selon les résultats ci-dessus, l'AGENT performe de façon acceptable par rapport aux autres algorithmes dans le cas des fonctions unimodales comme la sphère ou la double somme de Schwefel, se classant second dans chaque cas.

Les bonnes performances de l'AGENT, malgré qu'il ne soit pas à la base conçu pour les fonctions unimodales, s'expliquent par le fait que l'AGENT imite les algorithmes de type GPS en raffinant progressivement son pas d'avancement à mesure que le nombre d'évaluations augmente et que la topologie de la fonction le requiert. Ainsi, par rapport aux autres algorithmes qui n'ont pas de procédé pour raffiner leur pas d'avancement (c'est-à-dire tous les algorithmes proposés excepté l'AGED-RL), chacune de ses évaluations dispose d'une plus grande probabilité de se rapprocher de l'optimum une fois parvenu très près de celui-ci.

Fonctions multimodales séparables

L'AGENT semble disposer de piètres performances par rapport aux autres algorithmes dans le cas des fonctions multimodales et dont les variables sont séparables comme dans le cas des fonctions de Rastrigin et de Schwefel.

Il est possible de comprendre ce phénomène en examinant la topologie des fonctions multimodales étudiées. Par exemple, dans le cas de la fonction de Rastrigin (voir figure 6.21), on observe une quantité extrêmement élevée d'optimums répartis uniformément sur le domaine, entre lesquels la fonction coût varie drastiquement.

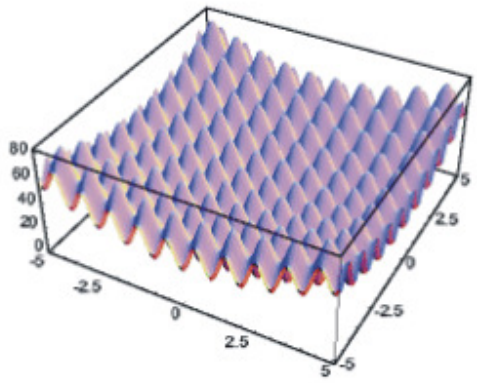
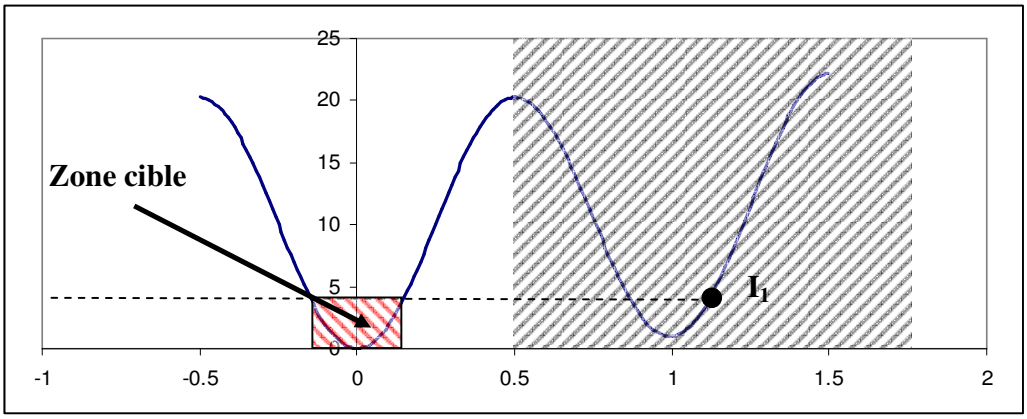


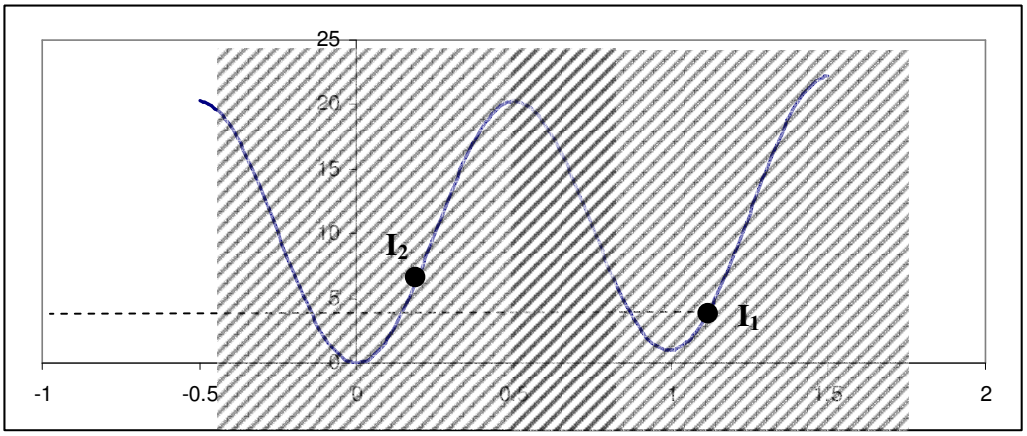
Figure 6.21 : Fonction de Rastrigin à 2 dimensions

Pour des optimums si nombreux, si escarpés et si rapprochés les uns des autres, l'algorithme trouve d'abord l'un des optimums près du centre, et est ensuite limité dans sa recherche par les noyaux territoriaux.

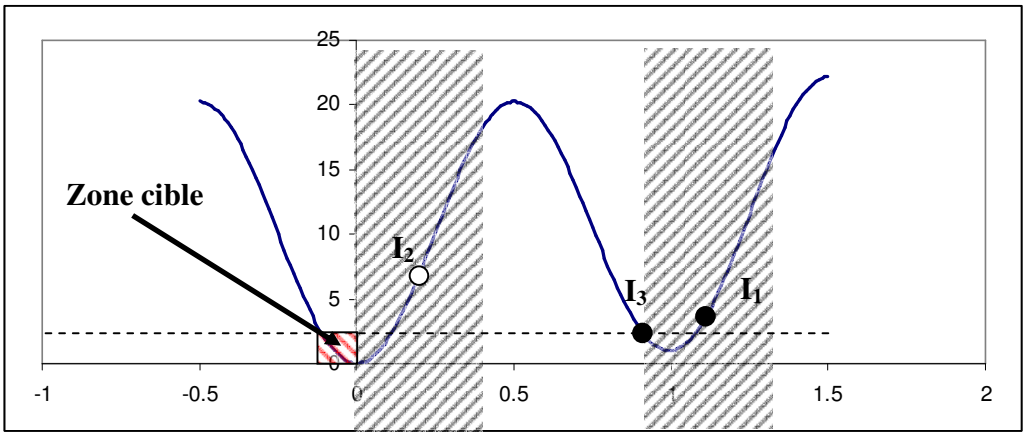
Cette situation est illustrée à la figure 6.22, qui représente la fonction de Rastrigin en une seule dimension aux environs de l'optimum. Sur la figure 6.22 A, le point $\mathbf{I_1}$ représente un point appartenant à la population active **P** de l'AGENT. La zone hachurée autour de $\mathbf{I_1}$ représente la limite de son territoire. Aucun individu ne peut naître dans cette zone.



A



B



C

Figure 6.22 : Problème de progression de l'AGENT pour la fonction de Rastrigin

Dans l'AGENT, les limites à gauche et à droite de la zone hachurée représentent aussi le pas le plus probable de mutation. Plus on s'éloigne de ces limites, plus la probabilité de générer un nouveau point est faible. Afin que le nouveau point généré soit assurément inclus dans la population active, il doit apparaître dans la zone cible désignée sur la figure. On comprend donc que la probabilité que ce point soit généré est très faible. La situation la plus probable est donc que le point généré ne se situe pas dans la zone cible. La figure 6.21 B illustre $\mathbf{I_2}$, une localisation possible de génération. On constate alors que, selon le principe des noyaux territoriaux, le territoire autour du point généré empêche toute apparition de nouveaux individus dans la zone cible pour un certain temps.

Après plusieurs générations, une fois les territoires de chasse raffinés, une portion de la zone cible devient à nouveau atteignable. Toutefois, à ce stade, puisque $\mathbf{I_1}$ dispose d'une meilleure valeur adaptative que $\mathbf{I_2}$, il est fort probable que ce $\mathbf{I_2}$ ne fasse plus partie de la population active **P**, et donc que les nouveau-nés émergent plutôt aux environs de $\mathbf{I_1}$. L'individu $\mathbf{I_3}$ de la figure 6.22 C représente une localisation très probable de génération du nouvel individu. L'individu $\mathbf{I_3}$ dispose d'une excellente valeur adaptative, et sera alors inclus dans la population active. Par contre, on comprend qu'à ce stade, puisque les pas de mutation suivent une probabilité gaussienne, il est très peu probable qu'une mutation suffisamment importante se produise pour atteindre la nouvelle zone cible, et ainsi de suite.

Pour ce type de problèmes, l'algorithme converge donc généralement très rapidement vers des valeurs près d'un optimum local de qualité moyenne, mais a ensuite beaucoup de difficulté à repérer l'optimum global, et se contente donc uniquement de raffiner la solution près du meilleur optimum découvert au départ.

Les autres algorithmes testés, quant à eux, ne disposent pas de telles restrictions sur leur recherche. De plus, dans le cas des AGR, le pas de mutation de l'algorithme est défini

par l'opérateur. Puisque, pour la fonction de Rastrigin, la distance entre chacun des optimums est constante (les minimums sont espacés de 1 dans la direction de chacune des variables), le pas de mutation utilisé (0,8 dans la plupart des cas) augmente considérablement les probabilités de génération d'un individu aux environs d'un nouvel optimum lors de chacune des mutations, permettant à ces algorithmes de mieux performer pour le problème en question.

Dans la réalité, les problèmes de design structurel de pièces mécaniques reviennent généralement davantage à trouver des dimensions minimales pour lesquelles la pièce peut tout de même supporter sa charge. Ainsi, il existe toujours plusieurs optimums dus aux différentes configurations possibles de la pièce, mais la topologie risque davantage de posséder des zones évasées ressemblant à celle des topologies gaussiennes étudiées à la section 6.1, où l'AGENT performe de façon exemplaire, qu'à celle des problèmes étudiés dans cette section.

De plus, dans le cas d'un problème de design de pièces mécaniques, un optimum se situant sur un pic extrêmement escarpé de la fonction voudrait aussi signifier un facteur de sécurité très faible pour la pièce. En effet, la moindre variation de ses paramètres viendrait alors signifier une chute dramatique dans ses performances, ce qui implique que les optimums plus difficiles à retracer par l'AGENT ne sont pas nécessairement d'excellents optimums à considérer dans une situation de design réelle.

Finalement, afin d'expliquer davantage les piètres performances de l'AGENT dans ce cas précis, nous verrons à la section 6.3.2 que le fait que l'optimum se situe au centre du domaine confère un biais aux autres algorithmes qui nuit aux performances de l'AGENT.

Fonctions multimodales non-séparables

Pour ce qui est des fonctions multimodales non-séparables, on peut constater que l'AGENT offre de piètres performances dans le cas de la fonction d'Ackley, terminant en 6ème place sur les 7 algorithmes testés. Puisque le pas entre chacun des optimums de cette fonction est constant dans la direction de chacune des variables, l'AGENT dispose des mêmes défauts pour la résolution de cette fonction que pour la résolution de la fonction de Rastrigin, discutés précédemment.

Par contre, on remarque aussi que l'AGENT performe plutôt bien dans le cas des autres fonctions multimodales non-séparables, terminant même premier pour la fonction de Griewangk et troisième pour Rosenbrock.

L'explication du bon fonctionnement de l'AGENT dans ces cas précis n'est pas encore très bien définie. L'hypothèse de l'auteur est que les opérateurs de recherche des autres algorithmes tendent à privilégier des déplacements plutôt constants des individus dans les directions orthogonales des variables, ce qui nuit à leur progression pour ces cas plus complexes.

6.3.2 Effet de la localisation de l'optimum dans le domaine

Un autre problème de l'AGENT pour cet ensemble de tests est que la topologie converge toujours vers un optimum situé au centre du domaine (à l'exception de la fonction de Schwefel). Les autres algorithmes utilisent en général surtout les opérateurs de croisement pour générer de nouveaux individus. Ces opérateurs de croisement ont tendance à produire des individus situés dans un espace borné par l'emplacement des parents, procurant un biais positif vers l'optimum situé au centre du domaine. Puisque les opérateurs de recherche de l'AGENT se fondent davantage sur les mutations qui poussent l'algorithme à explorer des portions de domaine plutôt situées à une certaine distance des individus actuels, l'AGENT ne profite pas autant que les autres algorithmes de ce biais, et est par conséquent moins performant pour ce type de problèmes.

En réalité, toutefois, dans le cas de design structurel de pièces mécaniques, on désire généralement respecter les contraintes et réduire le poids des pièces. Selon que l'on enfreigne ou non une contrainte, il est au moins aussi probable que l'optimum se situe à la borne inférieure ou supérieure d'une série de variables qu'au centre de celle-ci. L'algorithme utilisé devra donc être apte à performer adéquatement dans chacun des cas.

Dans leurs articles, Tsutsui (2000) et Tsuitsui & Goldberg (2001) soulignent le problème de l'optimum centré à l'origine et ont modifié certaines fonctions tests pour obtenir un optimum situé sur la borne inférieure des variables du domaine. La table 6.7 écrit les fonctions que nous utiliserons dans cette thèse, tirés de ces articles.

Table 6.7. Fonctions tests de Tsutsui (2000)

Fonction	Définition	Δx	Position de l'optimum
F_3	$F_3 = \sum_{i=1}^{5} ent(x_i)^{**}$ $x_i \in [-5{,}12;5{,}11]$ $x_i^* \in [-5{,}12;5{,}00[\quad F_3^* = -30$	0,01	Coin
M-Rastrigin	$F_{Ras}(\vec{x}) = 200 + \sum_{i=1}^{20} \left(x_i^2 - 10\cos(2\pi x_i)\right)$ $x_i \in [0;5{,}11]$ $x^* = (0,0,...,0) \quad F_{Ras}^* = 0$	0,01	Coin
M-Sphere	$F_{Sph}(\vec{x}) = \sum_{i=1}^{20} x_i^2$ $x_i \in [0;5{,}12]$ $x^* = (0,0,...,0) \quad F_{Sph}^* = 0$	0,01	Coin
Rastrigin	Voir Table 6.5 avec $N = 20$	0,01	Centre
Sphere	Voir Table 6.5 avec $N = 20$	0,01	Centre

** ent(x_i) est la partie entière de x_i

Plutôt que de comparer la qualité finale des résultats comme dans la table 6.6 **Tsutsui** (2000) compare plutôt le nombre d'évaluations requises avant que toutes les variables disposent d'une distance maximale de Δx (décrite dans la table 6.7) avec l'optimum. Les

algorithmes qu'il compare sont un algorithme génétique à nombres réels (AGR) ainsi que différentes méthodes d'extension des frontières nommées BEM et BES qui ont été développées pour corriger le problème des algorithmes génétiques avec les optimums situés dans les bornes du domaine (voir D.3.1 et Tsutsui, 2000, pour plus de détails).

La table 6.8 illustre la comparaison entre les résultats moyens obtenus par l'AGENT après 20 lancements et ceux donnés par Tsutsui (2000) et Tsutsui & Goldberg (2001). Les paramètres utilisés par l'AGENT sont les même que ceux utilisés à la section 6.3.1. Notez que, lorsque les articles ont présenté différents échantillonnages ou variantes pour les méthodes proposées, seuls les meilleurs résultats sont inscrits dans la table 6.8.

Table 6.8. Nombre d'évaluations requises pour atteindre l'optimum des fonctions tests

Fonction	AGENT	GA	BEM	BES
F_3	2122 ±170	14 559,5 ±1049,0	7701,4 ±826,9	6830,6 ±924,3
M-Rastrigin	52 027 ±5669	414 271,4 ±14 248,8	317 736,6 ±14 410,5	143 400,1 ±12 239,7
M-Sphere	11 423 ±318	64 772,4 ±584,6	39 804,3 ±737,5	32 809,9 ±436,2
Rastrigin	> 300000 *	101 852,2 ±16 584,0	103 884,5 ±15 618,7	108 087,2 ±21 464,1
Sphere	23 103 ±636	38 383,1 ±764,3	38 665,4 ±613,9	37 113,6 ±1037,5

* : Optimum non atteint après 300 000 évaluations.

En observant les résultats obtenus pour l'AGENT dans les tables 6.6 et 6.8, on constate que celui-ci obtient un résultat largement supérieur à celui des autres algorithmes étudiés pour tous les problèmes où l'optimum se situe sur une borne située dans un coin du domaine. Ce constat est véridique même pour le problème de Rastrigin, pour lequel l'AGENT performe pourtant de façon médiocre lorsque l'optimum est situé au centre du domaine (pour des raisons expliquées en 6.3.1).

Ainsi, on peut en conclure que l'AGENT, qui évolue beaucoup par mutations, et plus apte à trouver les optimums en bordure du domaine que les autres algorithmes utilisant principalement le croisement comme moyen de progression.

Ce qu'il faut retenir, toutefois, c'est que, puisque les algorithmes utilisant des opérateurs de croisement autres que l'opérateur standard de l'AGR sont aptes à compétitionner avec l'AGENT sur une bonne variété de problèmes académiques, des recherches futures visant à remplacer l'opérateur de croisement de l'AGENT par un opérateur à parents multiples pourraient éventuellement s'avérer profitables.

Le prochain chapitre compare de nouveau les performances de l'AGENT avec celles d'autres algorithmes, mais cette fois pour le cas d'un problème de design de pièces mécaniques de nature industrielle.

Chapitre 7.
AUTOMATISATION DU DESIGN PRÉLIMINAIRE D'UN DISQUE DE ROTOR DE TURBINE À GAZ

Ce dernier volet de la thèse présente le problème du design d'un disque de rotor de turbine à gaz de poids minimal. Ce problème représente plutôt bien le niveau de complexité et la structure des problèmes les plus simples auxquels un ingénieur pourra être confronté dans un contexte industriel, et est par conséquent un excellent cas test pour un algorithme d'automatisation de design voulant performer adéquatement durant la phase conceptuelle du design.

La figure 7.1 présente un disque de turbine à gaz. Le problème actuel s'intéressera au dimensionnement général du disque, mais sans inclure l'attache de l'ailette.

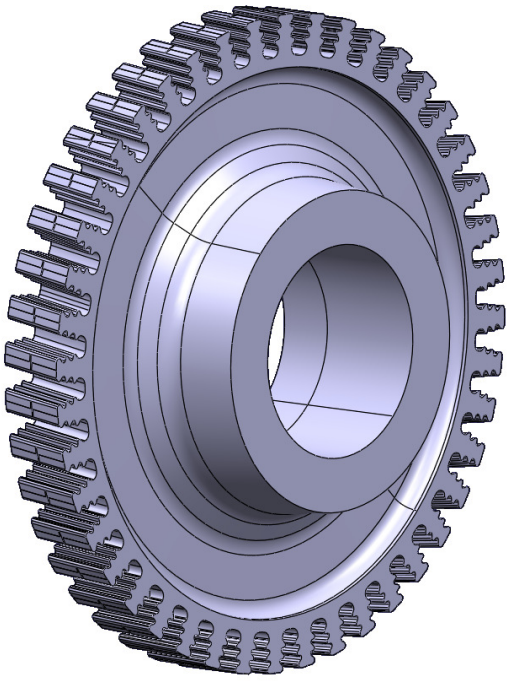


Figure 7.1 : Disque de rotor de turbine à gaz

La figure 7.2 présente une vue en coupe de la portion de disque considérée, le plan de coupe étant parallèle à l'axe du moteur.

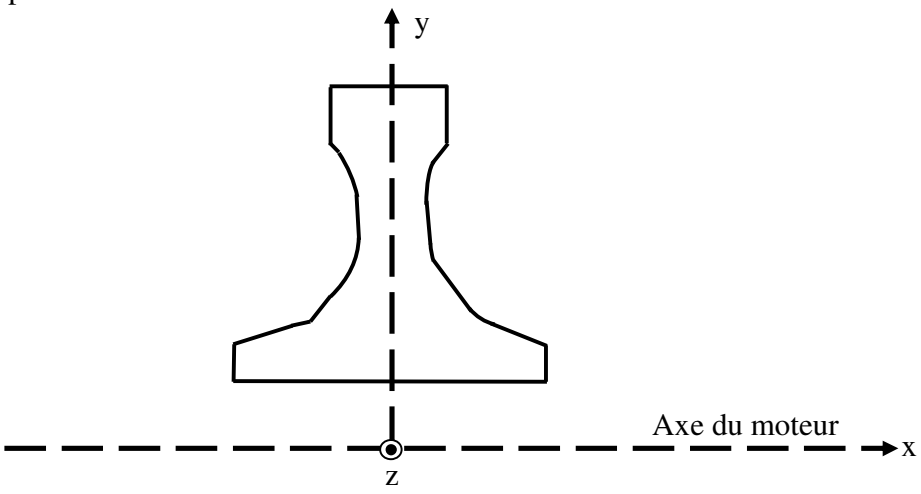


Figure 7.2 : Vue en coupe du disque (attachement des ailettes non incluses).

7.1 Spécifications

Les spécifications du problème proviennent de deux sources. D'abord, le client désire une vie utile et une efficacité de moteur minimale bien définies pour ses conditions d'opération. Ces spécifications, traitées ensuite par un département d'aérodynamique, produisent des contraintes mécaniques, géométriques et thermiques bien précises pour le modèle.

Peu importe le modèle utilisé, au moment où le problème de design structurel doit être résolu, les spécifications de la table 7.1 sont donc fournies au designer.

Table 7.1. Spécifications pour le problème du disque

Spécification	Nom	Unités
p_1	Rayon à l'alésage	m
p_2	Rayon de la jante	m
p_3	Largeur de jante	m
p_4	Excentricité de la traction des ailettes	m
p_5	Traction des ailettes	N
p_6	Vitesse de rotation	RPM
p_7	Température d'opération	^{o}C
p_8	Vie Utile	heures

Les spécifications p_1 à p_4 sont des considérations géométriques du problème, et sont illustrées sur la figure 7.3, ainsi que la direction de la force p_5 et la direction de rotation à la vitesse p_6.

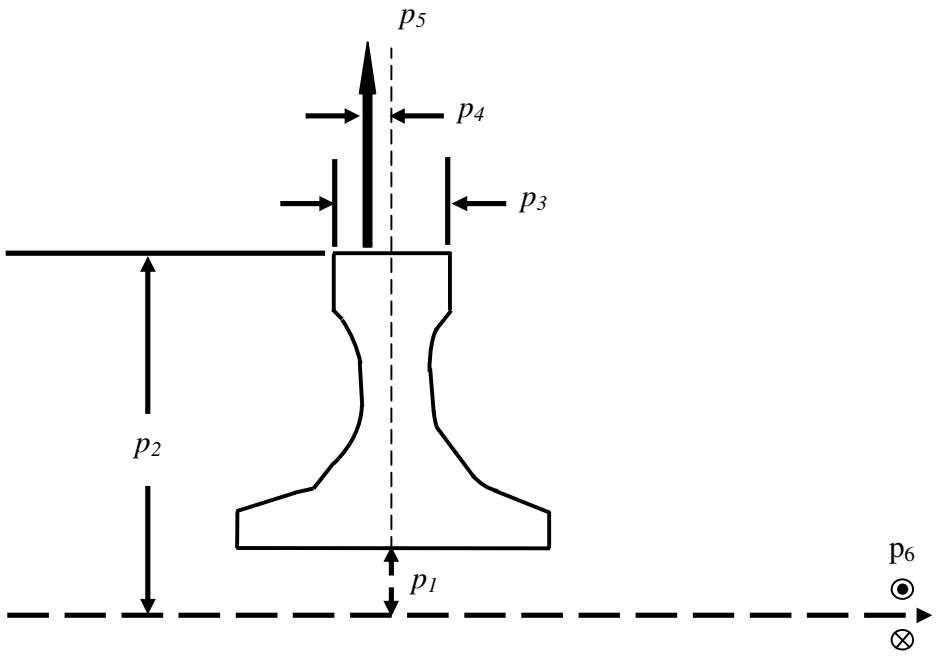


Figure 7.3 : Spécifications du problème de disque

Les deux autres spécifications sont la température p_7 du gaz circulant dans le moteur en condition d'opération normale et le nombre d'heures p_8 pour lequel les ailettes du moteur disposent de moins de 1% de chances de rupture.

7.2 Modèles paramétriques

Deux modèles paramétriques plutôt similaires seront utilisés pour générer les solutions initiales connues pour le problème.

7.2.1 Modèle symétrique à 3 cols

Le premier est celui de la figure 7.4. Les disques produits par ce modèle disposent d'un plan de symétrie transversal en plus de l'axe de symétrie donné par l'axe du moteur. Essentiellement, les paramètres ont été choisis pour positionner cinq points entre la base et le sommet du disque. De ces points, le plus bas et le plus élevé ont respectivement la même dimension axiale que la base et le sommet du disque, alors que les trois points centraux (les cols) peuvent être positionnés de façon à soustraire du matériel entre ces deux points.

De plus, autant pour des obligations de fabrication que pour relaxer les contraintes, chacun des cols est recouvert d'un arrondi tangent aux deux droites qui se rejoignent au col en question.

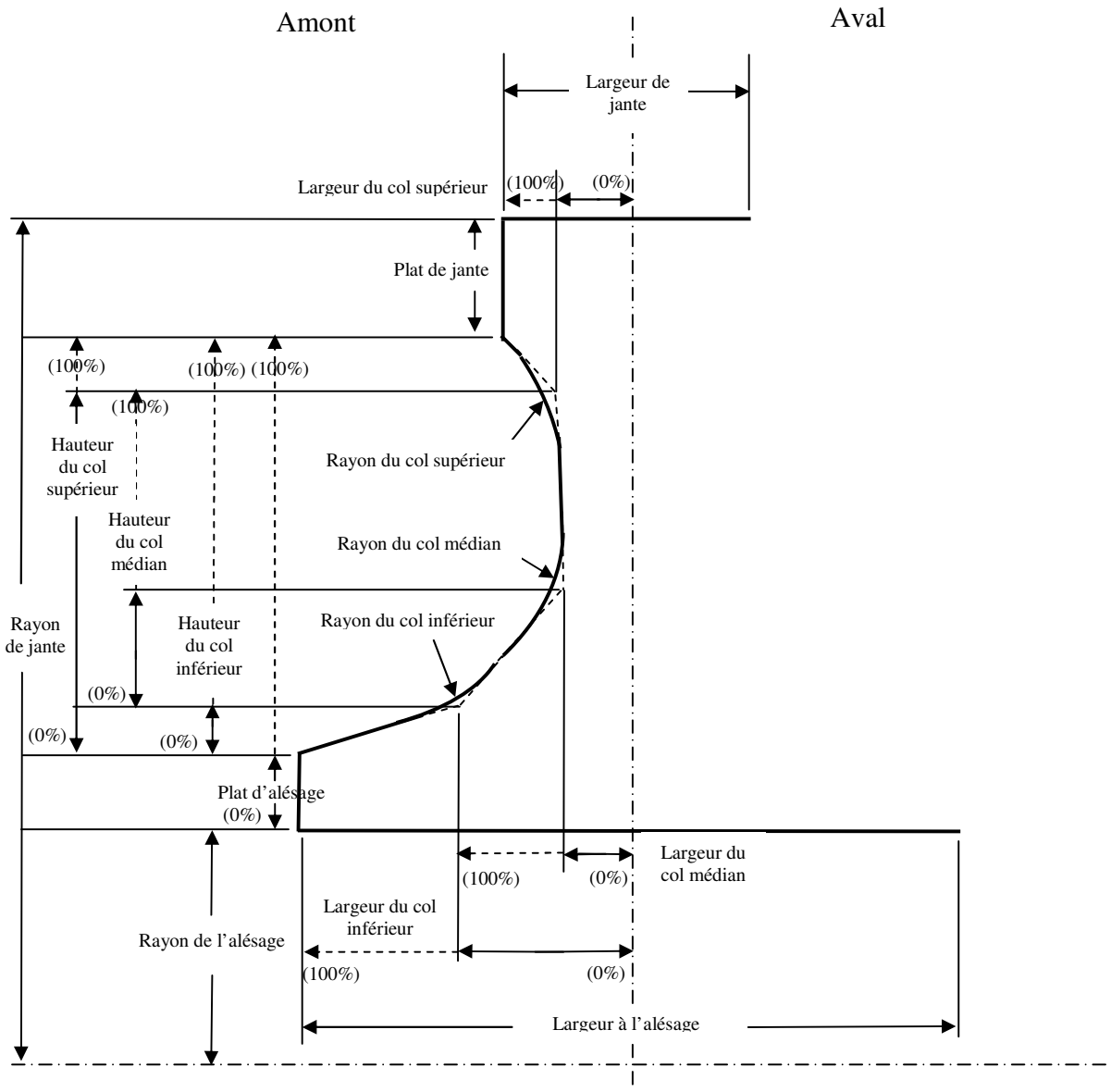


Figure 7.4 : Modèle de disque symétrique à 3 cols

Au total, 16 paramètres sont nécessaires pour définir une solution à l'aide de ce modèle. Trois de ces paramètres (le rayon de l'alésage, le rayon de jante et la largeur de jante) seront donnés par les spécifications du problème (voir table 7.1). Le designer aura donc

à déterminer une valeur pour chacun des 13 paramètres restants afin de produire une solution pour ce modèle paramétrique. Ces paramètres sont identifiés dans la table 7.2.

Table 7.2. Paramètres du modèle de disque symétrique à 3 cols

Paramètres	Nom	Unités
a_1	Rayon de l'alésage (= p_1)	m
a_2	Rayon de jante (= p_2)	m
a_3	Largeur de jante (= p_3)	m
a_4	Largeur à l'alésage	m
a_5	Plat d'alésage	m
a_6	Largeur du col inférieur	%
a_7	Hauteur du col inférieur	%
a_8	Rayon du col inférieur	m
a_9	Largeur du col médian	%
a_{10}	Hauteur du col médian	%
a_{11}	Rayon du col médian	m
a_{12}	Largeur du col supérieur	%
a_{13}	Hauteur du col supérieur	%
a_{14}	Rayon du col supérieur	m
a_{15}	Plat de jante	m
a_{16}	Matériau de construction	--

Tous les paramètres sont des nombres réels, à l'exception du dernier, le matériau de construction, qui est représenté par un nombre entier (0 ou 1, selon que l'on utilise respectivement le matériau 1 ou le matériau 2).

7.2.2 Modèle asymétrique à 3 cols

Ce modèle de solution est très similaire au modèle symétrique à trois cols présenté précédemment, mais comporte la différence majeure de ne pas posséder de plan de symétrie radial.

Ainsi, la plupart des paramètres existants dans le modèle symétrique devront être définis à la fois pour la moitié du disque située en amont et la moitié du disque située en aval de l'axe du moteur, donnant un total de 28 paramètres à spécifier par l'utilisateur pour

produire une solution au problème, en incluant les trois paramètres donnés par les spécifications.

La figure 7.5 illustre les paramètres du modèle asymétrique. Notez que tous les paramètres pour lesquels il est spécifié « (amont) » disposent d'un second paramètre décrivant la même dimension en aval, noté « (aval) ».

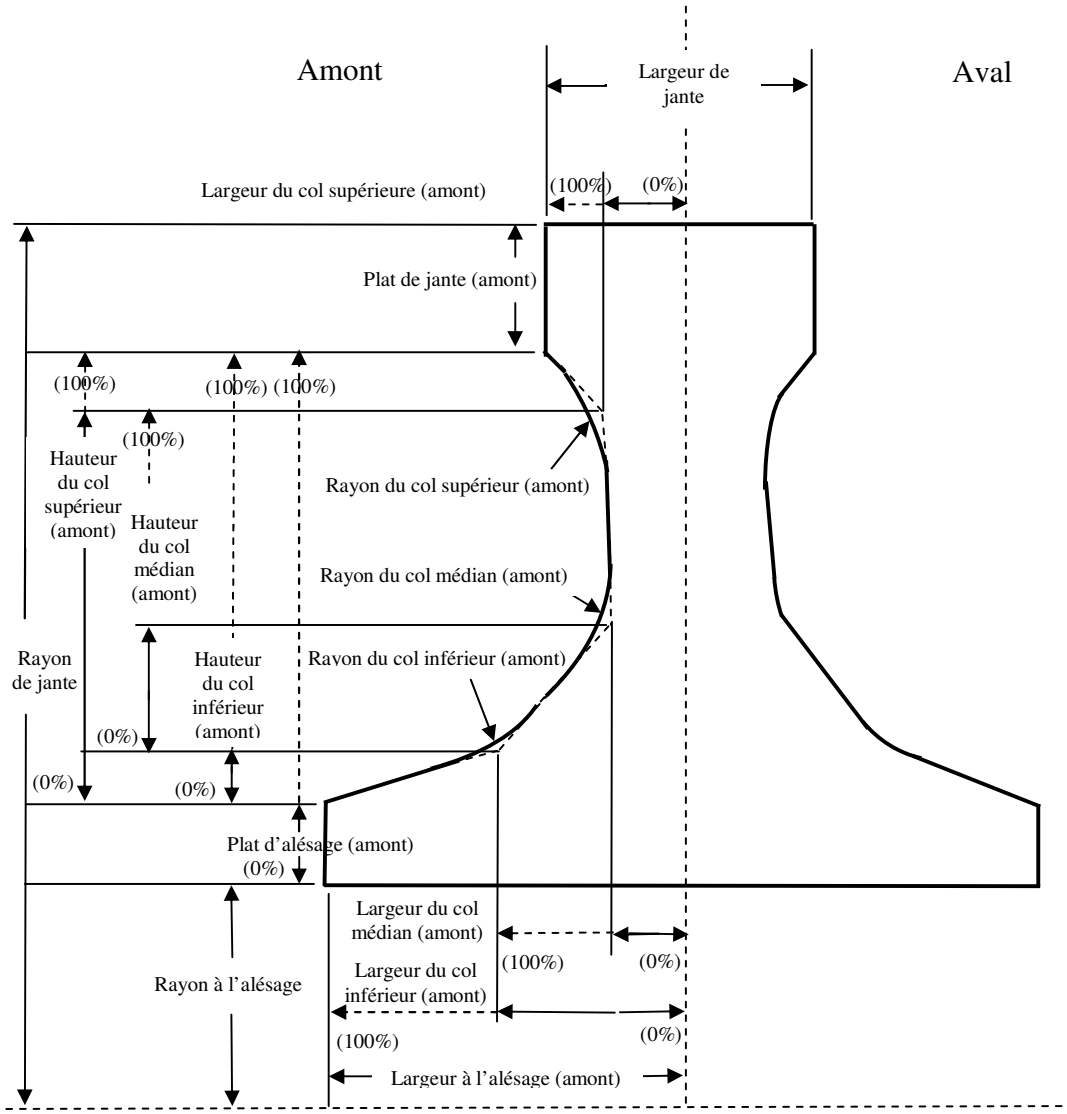


Figure 7.5 : Modèle de disque asymétrique à 3 cols

Essentiellement, on constate que le modèle asymétrique est apte à reproduire toutes les solutions produites par le modèle symétrique en plus d'être capable de générer les disques asymétriques. Toutefois, pour diverses raisons de calculs et de fabrication, le coût total de l'utilisation du modèle asymétrique est plus élevé que l'utilisation du modèle symétrique. Ce désavantage sera inclus lors du calcul de la fonction coût du problème (voir section 7.4). Les paramètres du modèle asymétrique sont énumérés dans la table 7.3.

Table 7.3. Paramètres du modèle de disque asymétrique à 3 cols

Paramètres	Nom	Unités
a_1	Rayon de l'alésage (= p_1)	m
a_2	Rayon de jante (= p_2)	m
a_3	Largeur de jante (= p_3)	m
a_4	Largeur à l'alésage (amont)	m
a_5	Plat d'alésage (amont)	m
a_6	Largeur du col inférieur (amont)	%
a_7	Hauteur du col inférieur (amont)	%
a_8	Rayon du col inférieur (amont)	m
a_9	Largeur du col médian (amont)	%
a_{10}	Hauteur du col médian (amont)	%
a_{11}	Rayon du col médian (amont)	m
a_{12}	Largeur du col supérieur (amont)	%
a_{13}	Hauteur du col supérieur (amont)	%
a_{14}	Rayon du col supérieur (amont)	m
a_{15}	Plat de jante (amont)	m
a_{16}	Largeur à l'alésage (aval)	m
a_{17}	Plat d'alésage (aval)	m
a_{18}	Largeur du col inférieur (aval)	%
a_{19}	Hauteur du col inférieur (aval)	%
a_{20}	Rayon du col inférieur (aval)	m
a_{21}	Largeur du col médian (aval)	%
a_{22}	Hauteur du col médian (aval)	%
a_{23}	Rayon du col médian (aval)	m
a_{24}	Largeur du col supérieur (aval)	%
a_{25}	Hauteur du col supérieur (aval)	%
a_{26}	Rayon du col supérieur (aval)	m
a_{27}	Plat de jante (aval)	m
a_{28}	Matériau de construction	--

7.3 Contraintes du problème

En supposant que les bornes des variables soient instaurées correctement (il est important de faire particulièrement attention aux bornes supérieures des rayons des cols), la seule contrainte géométrique restante est celle stipulant que le col supérieur doit se trouver au-dessus du col inférieur, soit :

Contrainte h_1 : $a_{13} - a_7 \geq 0$

Quant aux contraintes de résistance mécanique, on doit respecter les équations suivantes :

Contrainte h_{i+1} : $FS_i \geq 1{,}02$ pour $i \in \{1,2...N_d\}$

FS_i est le facteur de sécurité, calculé comme étant le ratio entre la contrainte mécanique effective au point de discrétisation i du disque et la contrainte maximale admissible par le matériau en ce point précis. Nous verrons plus tard que l'analyse discrétisera le disque en N_d points. Les facteurs de sécurité seront fournis par la relation 7.1.

$$FS_i = \frac{\sigma_{\max,i}}{\sigma_{eff,i}} \tag{7.1}$$

où $\sigma_{eff,i}$ est la contrainte mécanique effective au point i.

$\sigma_{max,i.}$ est la contrainte mécanique maximale admissible par le matériau au point i.

L'analyse permettant de calculer $\sigma_{eff,i}$ et $\sigma_{max,i.}$ est décrite dans la section 7.6.

Enfin, pour le modèle asymétrique, on doit aussi ajouter la contrainte en aval :

Contrainte h_{2+N_d} : $a_{25} - a_{19} \geq 0$

Bornes des variables a_1 à a_3

Notez qu'il a été mentionné que les paramètres a_1, a_2 et a_3 sont directement égaux aux spécifications du problème. Ces paramètres ne sont donc pas des variables du problème à proprement dites puisqu'elles seront posées constantes tout au long de sa résolution. Par contre, puisqu'elles sont nécessaires à l'évaluation des contraintes et du poids du disque, elles ont été incluses dans les paramètres afin que la fonction coût sans contraintes soit G($\vec{a}$) et non G($\vec{a}, \vec{p}$). De cette façon, le RNA substitut à la fonction coût disposera de toutes les données nécessaires à une estimation correcte de G($\vec{a}$) sans devoir modifier la structure de l'algorithme d'apprentissage, qui n'inclut dans le moment que les variables et non les spécifications (voir section 3.4.4).

Ensuite, les bornes supérieures et inférieures des variables a_1 à a_3 seront posées comme suit :

$$a_{max,i} = a_{min,i} = p_i \qquad \text{pour} \quad i \in \{1,2,3\}$$

Ainsi, la sensibilité pour ces paramètres sera toujours calculée égale à 0 (voir équation 3.5), et aucun de ces paramètres ne sera par conséquent sélectionné comme variable pour l'algorithme d'optimisation lors des divers raffinements, ce qui permettra leur utilisation comme paramètres sans alourdir inutilement les calculs.

7.4 Fonction coût des modèles

L'objectif du problème est de dimensionner un disque de poids minimal capable de supporter la charge centrifuge exercée par son propre poids et par celui des ailettes qu'il supporte.

Pour le modèle symétrique, la fonction coût sera donc la suivante :

$$F(\vec{a}) = G(\vec{a}) + \sum_{m=1}^{1+N_d} \left(vg_m(\vec{a}, \vec{p}) + DP_m(\vec{a}, \vec{p}) \right) \tag{7.2}$$

où $G(\vec{a})$ est le poids du disque

Les autres paramètres sont les mêmes que ceux donnés par l'équation 3.2 à la section 3.2.1, les contraintes étant fonction de la charge centrifuge à supporter (voir section 7.6).

Pour un disque asymétrique, un coût additionnel de 1% doit être ajouté pour tenir compte des pertes dues aux procédés d'analyse et de fabrication plus complexes impliqués lors du design avancé. La fonction coût suivante sera donc appliquée lors de l'utilisation d'un modèle asymétrique :

$$F(\vec{a}) = 1{,}01\, G(\vec{a}) + \sum_{m=1}^{2+N_d} \left(vg_m(\vec{a}, \vec{p}) + DP_m(\vec{a}, \vec{p}) \right) \tag{7.3}$$

7.5 Variables de sortie t du modèle

Tel que mentionné à la section 3.4.4, il est important de déterminer un ensemble de variables de sortie permettant aux réseaux de neurones de calculer les transgressions de contraintes indépendamment du calcul de la fonction $G(\vec{a})$. Dans cet exemple, nous utiliserons donc les variables données par la relation 7.4.

$$t_i = \mathrm{FS_i} \qquad \text{pour} \quad i \in \{1,2,3\} \tag{7.4}$$

Ceci permettra le calcul direct des contraintes par le RNA sans avoir besoin d'effectuer l'analyse entière.

7.6 Analyse du modèle paramétrique

Pour les besoins du calcul du poids et des contraintes mécaniques du disque, celui-ci sera discrétisé en N_d + 1 points (x_i ,y_i) répartis uniformément entre l'alésage et l'extrémité supérieure de la jante selon la dimension radiale du disque (voir figure 7.6).

N_d + 1 points (x'_i ,y'_i) seront également calculés en aval de l'écoulement, à la même hauteur que les points en amont ($y' = y$). Pour le modèle symétrique, on aura $\vec{x}\,' = -\vec{x}$.

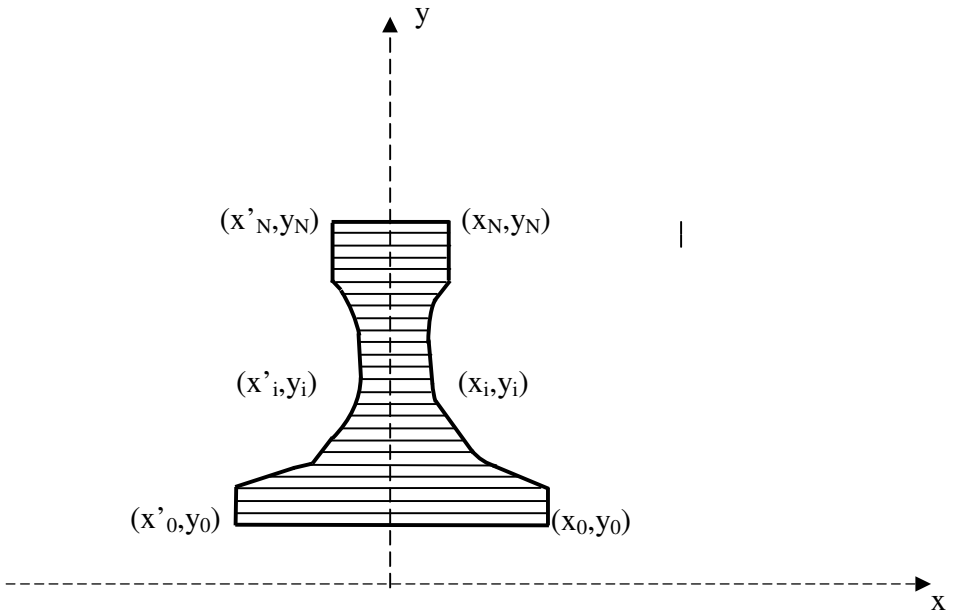


Figure 7.6 : Discrétisation du disque

La mathématique permettant de calculer l'emplacement de ces points à l'aide des paramètres des figures 7.4 et 7.5 n'inclut que des fonctions géométriques simples

(calculs de points sur des droites et des arcs de cercle), mais est suffisamment aride pour alourdir inutilement le texte. Elle ne sera donc pas présentée ici.

7.6.1 Poids d'une solution

Une fois l'emplacement de ces points déterminé, il est possible d'approximer le poids du disque par la méthode des trapèzes, comme s'il s'agissait d'une série de disques minces concentriques, selon l'équation suivante :

$$G(\vec{a}) = \rho\pi \sum_{i=0}^{N_d-1} (y_{i+1} + y_i)(y_{i+1} - y_i)(x_i - x'_i) \qquad (7.5)$$

où ρ est la masse volumique du matériel utilisé.

7.6.2 Contraintes tangentielles et radiales

Le calcul des contraintes tangentielles et radiales est un peu moins simple. La figure 7.7 illustre le total des forces radiales impliquées sur un élément infinitésimal du disque (pour N_d qui tend vers l'infini) dans le plan de coupe *yz*. On obtient alors l'équation 7.6 pour l'équilibre des forces selon l'axe radial.

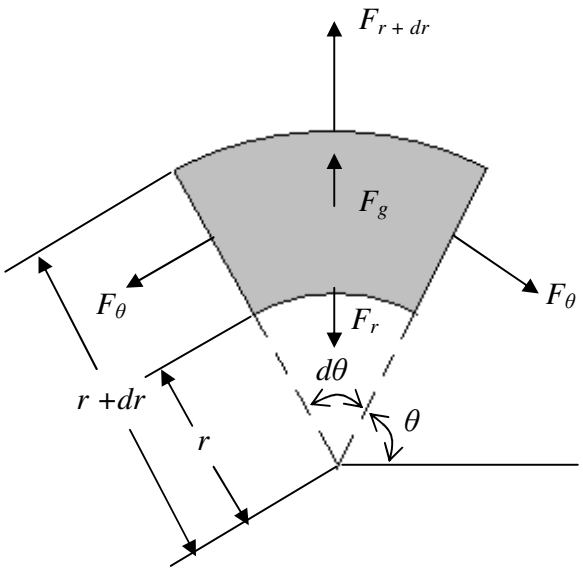


Figure 7.7 : Élément infinitésimal du disque

$$F_{r+dr} - F_r - 2F_\theta \sin\left(\frac{d\theta}{2}\right) = -F_g - P_r \qquad (7.6)$$

où F_r et F_{r+dr} sont les forces radiales exercées sur l'élément

F_θ sont les forces tangentielles exercées sur l'élément

θ est l'angle où est situé l'élément

$d\theta$ est l'angle couvert par l'élément

r est le rayon où est situé l'élément

dr est le rayon couvert par l'élément

F_g est la force centrifuge due à la masse même de l'élément

P_r est la force externe appliquée sur l'élément (égale à p_5)

En terme de contraintes, on peut définir les forces comme étant :

$$F_r = rt\sigma_r d\theta \tag{7.7}$$

$$F_\theta = \sigma_\theta dr\left(t + 0.5\frac{dt}{dr}dr\right)$$

$$F_{r+dr} = \left(\sigma_r + \frac{d\sigma_r}{dr}dr\right)\left(t + \frac{dt}{dr}dr\right)(r + dr)d\theta$$

$$F_g = \rho t r^2 \omega^2 d\theta dr$$

où σ_r est la contrainte radiale exercée sur l'élément

σ_θ est la contrainte tangentielle exercée sur l'élément

t est l'épaisseur de l'élément ($t_i = x_i - x'_i$)

ω est la vitesse angulaire de l'élément (égale à p_6)

En remplaçant les relations 7.7 dans 7.6, on obtient la relation d'équilibre 7.8.

$$P_r + \frac{d}{dr}(r\sigma_r t) - \sigma_\theta t + \rho\omega^2 r^2 t = 0 \tag{7.8}$$

Ensuite, puisque l'on discrétise le problème en $N_d + 1$ sections, on peut remplacer les différentielles par des différences finies. Ainsi, à une certaine station i, on aura :

$$\left(\frac{P_{r,i} + P_{r,i-1}}{2}\right) + \left(\frac{r_i t_i \sigma_{r,i} - r_{i-1} t_{i-1} \sigma_{r,i-1}}{r_i - r_{i-1}}\right) - \left(\frac{\sigma_{\theta,i} t_i + \sigma_{\theta,i-1} t_{i-1}}{2}\right) + \rho\omega^2\left(\frac{r_i^2 t_i + r_{i-1}^2 t_{i-1}}{2}\right) = 0 \tag{7.9}$$

Partant de cette relation, il est possible d'obtenir une relation linéaire pour σ_θ et σ_r comme suit :

$$A_{11}\sigma_{r,i} + A_{12}\sigma_{\theta,i} = B_1 \tag{7.10}$$

Avec :

$A_{11} = 1$

$$A_{12} = \frac{r_{i-1} - r_i}{2r_i}$$

$$B_1 = \frac{r_{i-1}t_{i-1}\sigma_{r,i-1}}{r_i t_i} + \frac{(r_i - r_{i-1})t_{i-1}\sigma_{\theta,i-1}t_{i-1}}{2r_i t_i} - \frac{\rho\omega^2(r_i - r_{i-1})}{2r_i}\left(r_i^2 + \frac{r_{i-1}t_{i-1}}{t_n}\right) - \left(\frac{r_i - r_{i-1}}{2r_i t_i}\right)(P_i + P_{i-1})$$

Les allongements ε_r et ε_θ, en coordonnées polaires axisymétriques, sont reliés aux déplacements radiaux *u* par les équations :

$$\varepsilon_r = \frac{du}{dr} \qquad \varepsilon_\theta = \frac{u}{r} \tag{7.11}$$

On obtient donc l'équation de compatibilité suivante :

$$\frac{d}{dr}(r\varepsilon_\theta) = \varepsilon_r \tag{7.12}$$

En utilisant de nouveau une différence finie, on pourrait réécrire la relation 7.12 comme suit :

$$\varepsilon_{\theta,i} - \left(\frac{r_i - r_{i-1}}{2r_i}\right)\varepsilon_{r,i} = \left(\frac{r_i - r_{i-1}}{2r_i}\right)\varepsilon_{r,i-1} + \frac{r_{i-1}\varepsilon_{\theta,i-1}}{r_i} \tag{7.13}$$

De façon générale, si l'on pose l'hypothèse d'une plasticité nulle (une déformation plastique rendrait de toute façon le disque inutilisable), la loi de Hooke définit les relations suivantes entre les allongements et les contraintes :

$$\varepsilon_r = \frac{\sigma_r - \upsilon_i \sigma_\theta}{E_i} + \alpha_i \Delta T \tag{7.14}$$

$$\varepsilon_\theta = \frac{\sigma_\theta - \upsilon_i \sigma_r}{E_i} + \alpha_i \Delta T$$

où α_i est le coefficient de dilatation thermique du matériau

ΔT est la différence de température dans la section

υ_i est le coefficient de poisson du matériau

E_i est le module de Young du matériau (son coefficient d'élasticité)

Pour le cas qui nous intéresse, le disque n'est soutenu que par un essieu sur l'une de ses extrémités, et les spécifications ont été choisies de façon à tenir compte de la dilatation thermique en cours d'opération. Ainsi, à l'équilibre, le gradient de température et les contraintes qu'il génère seront considérés nuls ($\Delta T = 0$.)

En remplaçant les relations de l'équation 7.14 dans 7.13, et en effectuant quelques manipulations algébriques, on peut obtenir la relation linéaire suivante :

$$A_{21}\sigma_{r,i} + A_{22}\sigma_{\theta,i} = B_2 \tag{7.15}$$

Avec :

$$A_{21} = \frac{\delta + \upsilon_i}{E_i}$$

$$A_{22} = \frac{1 + \upsilon_i \delta}{E_i}$$

$$B_2 = \alpha_i \Delta T (1 - \delta) = 0$$

$$\delta = \frac{r_i - r_{i-1}}{2r_i}$$

Pour les besoins de ce cas test, les constantes α, υ et E du matériau du disque seront estimées constantes pour un matériau donné, peu importe la température locale.

Avec les équations 7.10 et 7.15, on obtient un simple système de deux équations linéaires à deux inconnues que l'on peut résoudre simplement par la règle de Cramer :

$$\sigma_r = \frac{B_1 A_{22} - B_2 A_{12}}{\det(\mathbf{A})} \qquad \sigma_\theta = \frac{B_2 A_{11} - B_1 A_{21}}{\det(\mathbf{A})} \tag{7.16}$$

où det(**A**) est le déterminant de la matrice **A**.

On connait donc maintenant les valeurs des contraintes à une certaine station i en fonction de la contrainte à la station i - 1 (sachant que i = 0 à l'alésage et $i = N_d$ à l'extrémité de la jante). Aussi, sachant que l'alésage est une extrémité libre, et que seules les ailettes exercent une force sur l'extrémité de la jante, on dispose des conditions limites suivantes :

$$\sigma_{r,0} = 0 \tag{7.17}$$

$$\sigma_{r,N_d} = \frac{p_5}{2\pi p_2 a_3}$$

où $\sigma_{r,0}$ et σ_{r,N_d} sont respectivement les contraintes radiales aux stations i = 0 et i = N_d

Il est possible de résoudre le problème itérativement en posant successivement des valeurs de $\sigma_{\theta,0}$ de plus en plus susceptible de produire le résultat escompté pour $\sigma_{r,N}$. Un

estimé initial suffisant est $\hat{\sigma}_{\theta,0} = \sigma_{r,N_d}$. L'algorithme de la figure 7.8 permet alors de résoudre le problème (notez que $\hat{\sigma}$ représente une valeur estimée de σ).

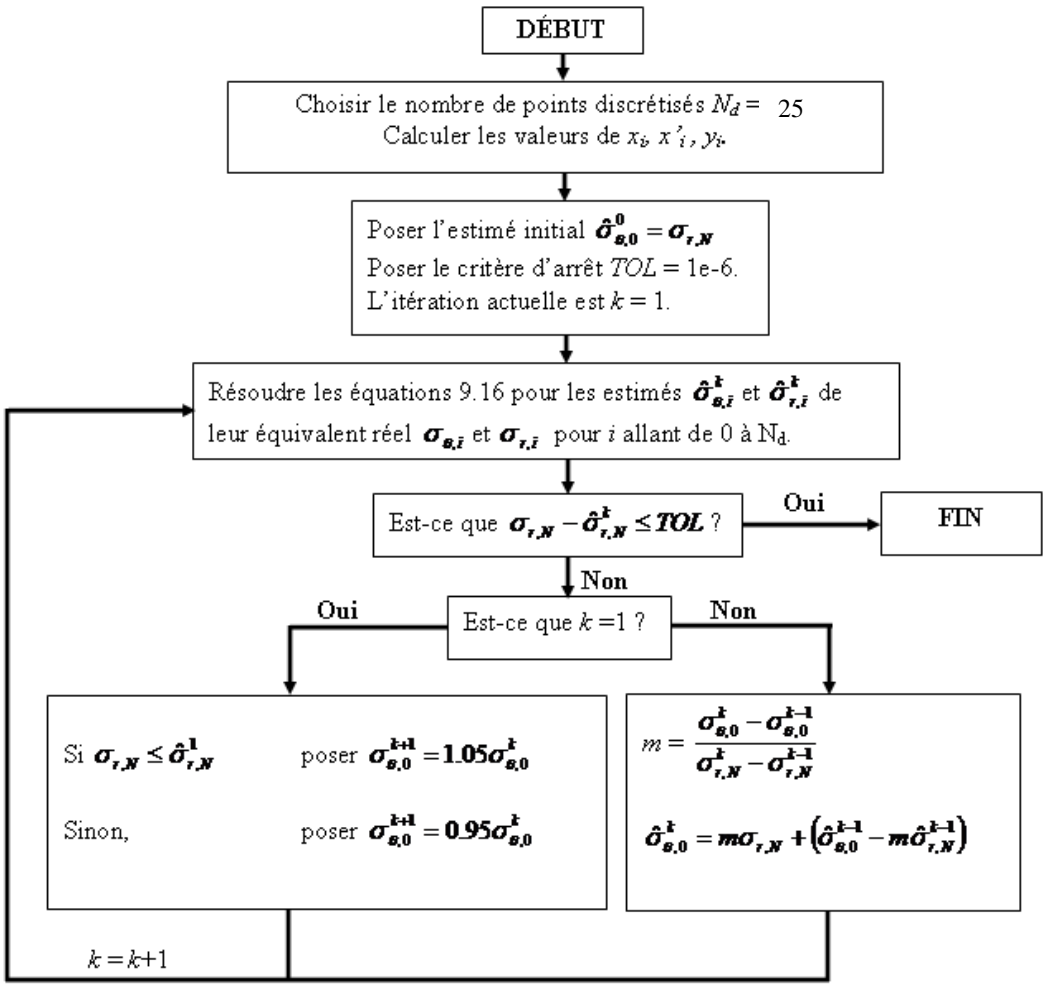


Figure 7.8 : Calcul des estimés pour les contraintes $\sigma_{\theta,i}$ et $\sigma_{r,i}$

7.6.3 Contraintes axiales

Si les spécifications incluent une excentricité de la force centrifuge exercée par les ailettes (i.e. $p_4 > 0$), ou si le disque est asymétrique, un moment de flexion sera possiblement induit, provoquant une contrainte axiale additionnelle dans le moteur.

À une station *i* donnée, le moment de flexion est approximé par l'équation 7.18.

$$M_{Flex} = p_5\left(p_4 - \frac{x_i + x'_i}{2} \right) \tag{7.18}$$

La contrainte de flexion résultante est alors :

$$\sigma_{Flex} = \frac{M_{Flex}\left(\frac{x_i - x'_i}{2} \right)}{I} \tag{7.19}$$

Avec $$I = \frac{\pi r_i \left(x_i - x'_i \right)^3}{6}$$

où I est le moment d'inertie approximatif d'une section du disque.

7.6.4 Contrainte effective

Une fois la distribution de contraintes calculées, on peut déterminer la contrainte effective totale en un certain point selon l'équation suivante :

$$\sigma_{eff,i} = \sqrt{\sigma_{r,i}^2 - \sigma_{r,i}\sigma_{\theta,i} + \sigma_{\theta,i}^2 + \sigma_{flex,i}^2} \tag{7.20}$$

Contraintes maximales et paramètres des matériaux

La contrainte maximale possible en un point du disque sans en provoquer la rupture dépend du matériau utilisé et de la température en ce point précis. Par souci de simplification, nous utiliserons une relation empirique pour évaluer la température en un point plutôt que de résoudre les véritables équations de conduction de la chaleur.

La distribution de température utilisée sera fonction de la position radiale y_i du point et de la température d'opération p_7 du moteur selon la relation 7.21. Par exemple, la figure 7.9 illustre cette distribution entre le rayon à l'alésage et le rayon de jante pour $p_7 = 1100\ ^0$C.

$$T_i = 260 + 0{,}5\frac{(p_7 - 260)}{1 + \exp\left(-5 + \frac{10i}{N_d}\right)} \tag{7.21}$$

où T_i est la température au point de discrétisation i

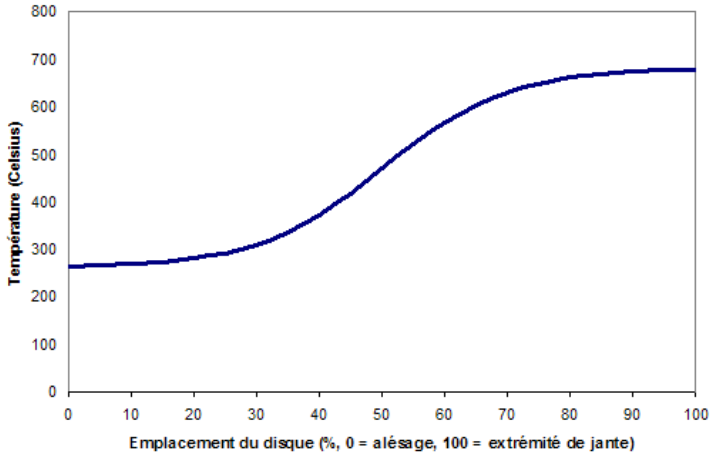


Figure 7.9 : Température du métal du disque pour une température d'opération du moteur de 1100 ^{o}C.

Maintenant, il ne reste qu'à déterminer la contrainte maximale supportable pour le matériau donné à cette température pour une durée de vie de p_8 heures.

Pour ce problème, nous supposerons deux choix de matériaux fictifs, soit *M1* et *M2*. Les propriétés de ces matériaux sont données dans la table 7.4 pour les vies utilisées dans ce problème, soit $p_8 = 5000$ ou 10000 heures.

Table 7.4. Propriété des matériaux M1 et M2

	Matériau *M1*	Matériau *M2*
E (MPa)	206×10^3	210×10^3
υ	0,3	0,3
ρ (kg/m^3)	8010	8080
σ_{max} (MPa) pour p_8 = 5000 heures	$\sigma_{max,i} = 8T_i^4 x10^{-10} - 2T_i^3 x10^{-6} + 9T_i^2 x10^{-4} - 2{,}64T_i \, x10^{-3} + 1115$	$\sigma_{max,i} = 3T_i^4 x10^{-10} - 9T_i^3 x10^{-7} + 3T_i^2 x10^{-4} - 7{,}9T_i \, x10^{-3} + 962$
σ_{max} (MPa) pour p_8 = 10000 heures	$\sigma_{max,i} = 0{,}92 \, \sigma_{5000}$ Où σ_{5000} est la valeur de $\sigma_{max,i}$ calculée pour p_8 = 5000 heures.	$\sigma_{max,i} = 0{,}88 \, \sigma_{5000}$ Où σ_{5000} est la valeur de $\sigma_{max,i}$ calculée pour p_8 = 5000 heures.

Il est alors possible d'évaluer en chaque point pour chaque matériau $\sigma_{max,i}(p_7, p_8)$ et donc de calculer les effractions de contraintes selon l'équation 7.2. La figure 7.10 illustre le comportement de chacun des matériaux selon la température pour une vie utile de 5000 heures.

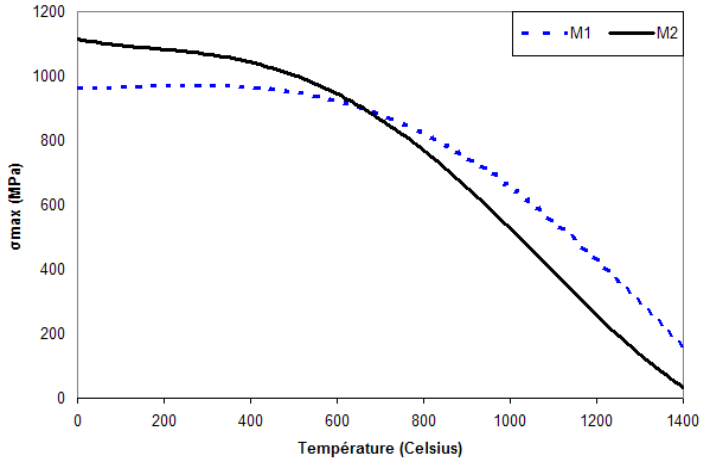


Figure 7.10 : Comportement des matériaux M1 et M2 selon la température.

7.7 Solutions initiales connues au problème

Afin de pouvoir profiter du procédé de RPC présenté au chapitre 3, il importe de connaître plusieurs solutions initiales au problème. Dans un cas réel, ces solutions initiales seraient en fait constituées des designs antérieurs.

Pour le cas test actuel, ces solutions ont été générées en lançant l'algorithme AGENT avec ses paramètres par défaut (table 6.3) à partir d'une population aléatoire de $k_{île} = 4$ îlots de $N_{pop} = 40$ individus pour un nombre d'évaluations égal à 50 000.

Dans un premier temps, par souci de limiter le nombre de solutions initiales requises pour le modèle, les trois spécifications géométriques ont été posées constantes pour toutes les solutions initiales :

$p_1 = 5{,}08$ cm
$p_2 = 15{,}36$ cm.
$p_3 = 2{,}54$ cm.

La table 7.5 montre la valeur des autres spécifications pour lesquelles une solution initiale a été calculée. Les solutions 1 à 16 sont constituées de toutes les combinaisons possibles pour deux valeurs distinctes de chaque spécification p_5, p_6, p_7 et p_8. La solution 17 est un point intermédiaire entre toutes ces spécifications. Selon la théorie sur les plans d'expérience, ces points devraient donner aux RNA de la phase d'adaptation la possibilité de concevoir un modèle au moins linéaire pour interpoler ou extrapoler une solution pour laquelle $p_1 = 5{,}08$ cm., $p_2 = 15{,}36$ cm., $p_3 = 2{,}54$ cm. et $p_4 = 0$ cm., en utilisant le modèle de disque symétrique.

Table 7.5. Valeur des spécifications des solutions initiales

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
1	0	177 928x50	25000	1100	5000	8,758
2	0	177 928x50	25000	1100	10000	9,995
3	0	177 928x50	25000	1500	5000	10,148
4	0	177 928x50	25000	1500	10000	11,741
5	0	266 893x50	25000	1100	5000	13,011
6	0	266 893x50	25000	1100	10000	13,362
7	0	266 893x50	25000	1500	5000	14,560
8	0	266 893x50	25000	1500	10000	14,840
9	0	177 928x50	30000	1100	5000	10,928
10	0	177 928x50	30000	1100	10000	12,272
11	0	177 928x50	30000	1500	5000	12,392
12	0	177 928x50	30000	1500	10000	15,052
13	0	266 893x50	30000	1100	5000	15,838
14	0	266 893x50	30000	1100	10000	17,491
15	0	266 893x50	30000	1500	5000	17,776
16	0	266 893x50	30000	1500	10000	20,344
17	0	222 411x50	27500	1300	5000	12,082
18	0,00508	222 411x50	27500	1300	5000	16,888
19	0,00508	177 928x50	25000	1100	5000	12,261
20	0,00508	266 893x50	30000	1500	10000	12,836

Ensuite, les solutions 18 à 20 fournissent quelques exemples de solutions pour lesquels $p_4 > 0$. Puisque cette excentricité p_4 produit une contrainte de flexion dans le disque si le centre de masse de chaque disque mince utilisé dans l'analyse ne se situe pas exactement à $\frac{x_i + x_i^{'}}{2} = p_4$, ces solutions utilisent toutes le modèle paramétrique asymétrique.

Puisqu'il n'y a que peu de solutions utilisant le modèle asymétrique, il y a de fortes chances que le RNA de la phase d'adaptation ne soit pas apte à produire une solution de départ très convenable, et ce sera donc à l'algorithme d'optimisation d'effectuer le plus gros du travail.

Les valeurs obtenues pour chacun des paramètres pour ces vingt solutions initiales sont fournies à l'annexe C.

7.8 Problèmes à résoudre

À ce stade, tout notre problème est défini convenablement, à l'exception des bornes des variables, des spécifications actuelles, de leurs poids $\vec{w}$.

Les valeurs des spécifications qui seront utilisées pour les différents cas tests sont inscrites dans la table 7.6, ainsi que les poids qui leur sont accordés. Notez que ces spécifications ont été choisies de façon arbitraire et ne représentent pas une application réelle de turbine à gaz.

Table 7.6. Spécifications des cas tests étudiés

Spécification	Poids w_i	Cas test #1	Cas test #2	Cas test #3	Cas test #4
p_1	0,1	0,0508	0,0508	0,0381	0,0508
p_2	0,1	0,1536	0,1536	0,1270	0,1536
p_3	0,1	0,0254	0,0254	0,0381	0,0254
p_4	1	0	0	0	0,01
p_5	0,1	26 000	32 000	27 500	30 000
p_6	0,1	200 000x50	280 000x50	222 411x50	266 893x50
p_7	0,1	1200	1600	1300	1100
p_8	0,1	5000	10000	5000	5000

Nous avons d'abord choisi le poids $w_4 = 1$ en sachant que c'est la spécification p_4 qui gère principalement le choix entre les deux modèles paramétriques suggérés. Ensuite, puisque notre connaissance à priori du problème ne laisse pas penser que l'une ou l'autre des spécifications dispose de plus de valeur qu'une autre, leurs poids ont tous été posés égaux à 0,1.

Le cas test #1 dispose de spécifications conscrites à l'intérieur du domaine borné par les spécifications données par les solutions 1 à 20. Puisque $p_4 = 0$ pour ce problème, la solution pourra en principe être interpolée plutôt efficacement grâce au RNA, et la solution initiale de l'AGENT devrait être déjà plutôt bonne.

Le cas test #2, quant à lui, dispose de spécifications sortant des bornes des solutions 1 à 20. Il nous servira à vérifier si le RNA peut être aussi un bon extrapolant.

Le cas test #3 propose de modifier les spécifications p_1 à p_3, qui sont constantes pour toutes les solutions initiales connues. Le RNA ne pourra alors pas l'interpoler correctement et la phase d'adaptation ne devrait alors pas donner d'avantages majeurs à l'algorithme pour cette solution.

Le dernier cas test proposé (#4) utilise une valeur de $p_4 > 0$ et offre donc la possibilité d'utilisation du modèle de disque asymétrique. À nouveau, puisque peu de solutions initiales sont disponibles pour ce modèle, la phase d'adaptation ne devrait pas accorder d'avantages déterminants pour ces problèmes.

La table 7.7 et 7.8 décrivent respectivement les bornes choisies pour chacune des variables pour le modèle symétrique et le modèle asymétrique de disque lors de la résolution de ces cas tests.

Table 7.7. Bornes utilisées pour les variables du modèle symétrique de disque

Paramètre	$a_{min,i}$	$a_{max,i}$	Paramètre	$a_{min,i}$	$a_{max,i}$
a_1	p_1	p_1	a_9	1	100
a_2	p_2	p_2	a_{10}	0	100
a_3	p_3	p_3	a_{11}	0,000127	0,00762
a_4	0,0508	0,2032	a_{12}	1	100
a_5	0,00127	0,00508	a_{13}	40	100
a_6	1	100	a_{14}	0,000127	0,00762
a_7	0	60	a_{15}	0,00254	0,0127
a_8	0,000127	0,00762	a_{16}	0	1

Table 7.8. Bornes utilisées pour les variables du modèle asymétrique de disque

Paramètre	$a_{min,i}$	$a_{max,i}$	Paramètre	$a_{min,i}$	$a_{max,i}$
a_1	p_1	p_1	a_{15}	0,00254	0,0508
a_2	p_2	p_2	a_{16}	0,0254	0,1016
a_3	p_3	p_3	a_{17}	0,000254	0,0254
a_4	0,0254	0,1016	a_{18}	1	100
a_5	0,000254	0,0254	a_{19}	0	60
a_6	1	100	a_{20}	0,000127	0,00508
a_7	0	60	a_{21}	1	100
a_8	0,000127	0,00508	a_{22}	0	100
a_9	1	100	a_{23}	0,000127	0,00508
a_{10}	0	100	a_{24}	1	100
a_{11}	0,000127	0,00508	a_{25}	40	100
a_{12}	1	100	a_{26}	0,000127	0,00508
a_{13}	40	100	a_{27}	0,00254	0,0508
a_{14}	0,000127	0,00508	a_{28}	0	1

Les cas tests seront d'abord résolus 20 fois par l'algorithme AGENT présenté au chapitre 6, utilisant le procédé complet de RPC décrit dans le chapitre 3. Les paramètres utilisés pour le processus de RPC et l'AGENT sont décrits dans la table 7.9

Table 7.9. Paramètres utilisés pour la résolution des cas tests

Paramètre	Valeur	Paramètre	Valeur
γ_c^o	0,2	α_m	0,5
z_{max}	2	C_z	100
$k_{île}$	6	N_{pop}	40
E_{max}	50 000	δ_{min}	$1x10^{-8}$
N_{RN}	0	N_{max}	5000

Avec N_{RN} = 0, on constate que l'opérateur de substitution n'est pas inclus dans l'AGENT pour la plupart des calculs de ce chapitre, pour deux raisons majeures :

- D'abord, l'opérateur de substitution n'a été implémenté que très récemment dans l'AGENT. La plupart des résultats avaient déjà été obtenus avant son implémentation.
- Actuellement, le prototype de l'AGENT est programmé dans un langage facilitant le développement mais alourdissant les calculs (le programme est en Visual Basic). Vu le nombre élevé d'évaluations de fonction (50 000) et le

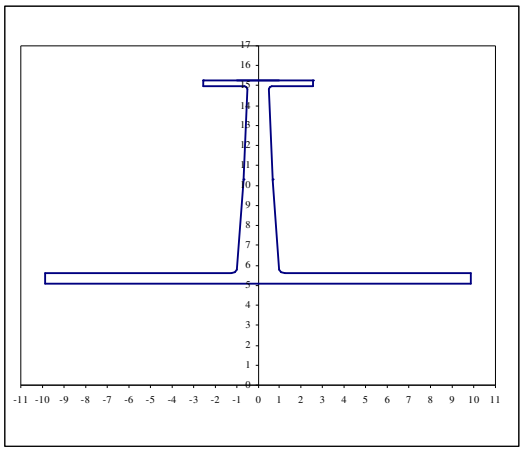
lancement de l'algorithme 20 fois pour chacun des tests, relancer l'intégralité des tests en utilisation la substitution requerrait davantage de ressources.

La section 7.11 comparera toutefois les résultats obtenus avec ou sans l'opérateur de substitution pour le cas test #1.

Les résultats moyens et les écarts types obtenus sont énumérés dans la table 7.10, et la forme résultante des meilleurs disques obtenus est illustrée sur les figures 7.11 à 7.14. L'échelle sur les figures est de 1 : 1 cm.

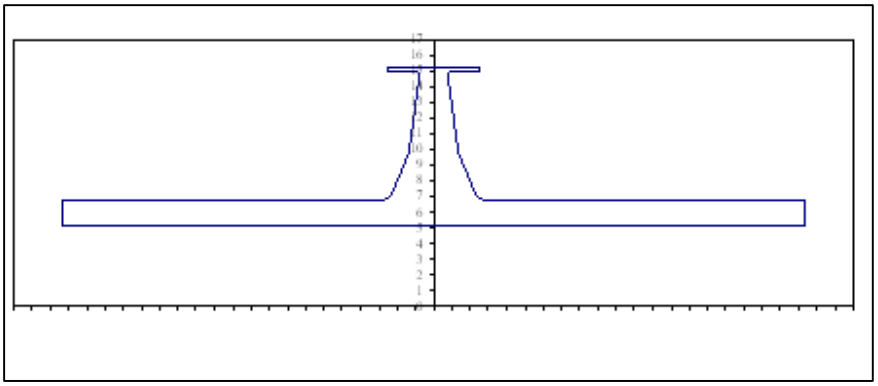
Table 7.10. Poids moyens obtenus pour les cas tests

Cas test #1	Cas test #2	Cas test #3	Cas test #4
10,290 ±0,080	33,030 +0,094	9,131 ±0,183	28,990 ±0,919



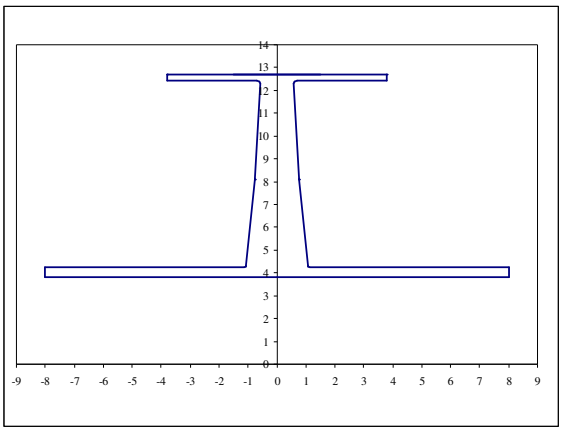
x_1	0,0508	x_7	5,5912E-06	x_{13}	0,99994195
x_2	0,1524	x_8	0,00240697	x_{14}	0,00142139
x_3	0,0508	x_9	0,33457115	x_{15}	0,00254384
x_4	0,1976	x_{10}	0,50040078	x_{16}	0
x_5	0,00127	x_{11}	0,00562733	Poids	10,228
x_6	0,10252	x_{12}	0,19470039		

Figure 7.11 : Meilleur disque obtenu pour le cas test #1



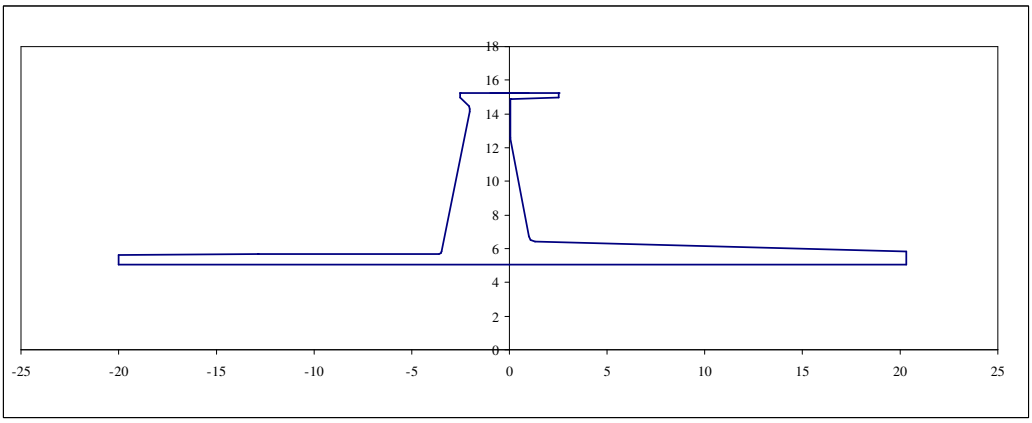
x_1	0,0508	x_7	0	x_{13}	0,99948878
x_2	0,1524	x_8	0,00530264	x_{14}	0,00215403
x_3	0,0508	x_9	0,38332361	x_{15}	0,00257522
x_4	0,40551213	x_{10}	0,35905905	x_{16}	0
x_5	0,00423364	x_{11}	0,00541499	Poids	32,911
x_6	0,11944796	x_{12}	0,29127947		

Figure 7.12 : Meilleur disque obtenu pour le cas test #2



x_1	0,0381	x_7	0,00052	x_{13}	1
x_2	0,127	x_8	0,00085654	x_{14}	0,00128676
x_3	0,0762	x_9	0,40170226	x_{15}	0,00260259
x_4	0,16030705	x_{10}	0,47204528	x_{16}	0
x_5	0,00127	x_{11}	0,00695321	Poids	7,820
x_6	0,13227411	x_{12}	0,14868929		

Figure 7.13 : Meilleur disque obtenu pour le cas test #3



x_1	0,0508	x_{11}	0,00169617	x_{21}	0,01
x_2	0,152	x_{12}	0,78041637	x_{22}	0,72543016
x_3	0,0508	x_{13}	0,92808669	x_{23}	0,00374126
x_4	0,20020731	x_{14}	0,00363812	x_{24}	0,01
x_5	0,00132798	x_{15}	0,00254	x_{25}	0,99
x_6	0,6427072	x_{16}	0,2032	x_{26}	0,0001
x_7	0,0100	x_{17}	0,0019	x_{27}	0,0025
x_8	0,0069	x_{18}	0,0500	x_{28}	0
x_9	0,1374	x_{19}	0,0672	Poids	24,514
x_{10}	0,0000	x_{20}	0,1313		

Figure 7.14 : Meilleur disque obtenu pour le cas test #4

7.9 Apport de la phase d'adaptation

Afin de bien démontrer la contribution de la phase d'adaptation, qui fournit une série de solutions initiales connues ainsi qu'une solution supplémentaire grâce à son RNA, les cas tests seront aussi résolus 20 fois en désactivant complètement ce RNA (c'est-à-dire que $k_{île}$ solutions initiales sont utilisées pour initialiser les populations des divers îlots au lieu de $k_{île} - 1$ solutions et une solution additionnelle provenant du RNA).

Le cas où aucune phase d'adaptation n'est utilisée a aussi été testé. Les solutions initiales de l'algorithme y ont alors été générées aléatoirement.

La table 7.11 décrit la valeur adaptative initiale de chacun des cas tests. Dans le cas de l'utilisation de la phase d'adaptation avec ou sans RNA, cette valeur correspond à celle donnée par la meilleure solution initiale connue et choisie (ou estimée par le RNA) pour ce problème. Pour le cas de l'initialisation aléatoire des solutions, la valeur considérée correspond à la meilleure valeur adaptative donnée après initialisation aléatoire de la population du problème. La valeur inscrite dans le tableau est alors la moyenne de cette valeur adaptative mesurée à chaque lancement de l'algorithme. On se rappelle que ψ est une constante représentant une pénalité de mort pour la solution (1×10^{50} par défaut). Le nombre de ψ ajoutés à une solution est donc une indication du nombre de contraintes violées par la solution donnée.

Table 7.11. Valeur adaptative initiale selon le type d'adaptation utilisé

	Cas test #1	**Cas test #2**	**Cas test #3**	**Cas test #4**
Solution initiale aléatoire	44,880	$377{,}55 + 9\psi$	50,412	$156{,}801 + \psi$
Phase d'adaptation sans RNA	13,011	$232{,}564 + 23\psi$	9,575	$371{,}687 + 23\psi$
Phase d'adaptation complète	11,083 (11,083)	$122{,}240 + 3\psi$ $(122{,}240 + 3\psi)$	10,170 (22,135)	$371{,}687 + 23\psi$ $(386{,}687 + 23\psi)$

Pour la phase d'adaptation utilisant un RNA, la valeur adaptative de la solution proposée par le RNA est décrite entre parenthèses. On constate, tel qu'attendu, que cette valeur est la meilleure solution initiale pour les cas tests #1 et #2, démontrant que les capacités d'interpolations et d'extrapolations des RNA sont en effet aptes à fournir une meilleure solution initiale à l'algorithme lorsque l'on dispose de suffisamment de solutions connues au problème.

Pour les cas tests #3 et #4, le RNA ne dispose pas des informations nécessaires à une estimation adéquate des résultats. En effet, pour le cas test #3, aucune information n'est contenue dans les solutions initiales sur le comportement du système lorsque $p_{1,} p_2$ et p_3 varient. Pour le cas test #4, le manque d'informations est dû au manque de solutions initiales pour ce modèle; seulement trois solutions initiales sont disponibles pour évaluer le comportement des vingt-huit variables du modèle asymétrique.

La table 7.12 décrit les résultats moyens et les écarts types obtenus après 20 lancements de l'algorithme pour chacun des cas tests selon l'utilisation d'une phase d'adaptation entière, d'une phase d'adaptation n'utilisant pas de RNA substitut, ou utilisant d'une optimisation partant plutôt simplement d'une solution initiale aléatoire.

Table 7.12. Comparaison entre les résultats obtenus selon l'utilisation de la phase d'adaptation.

	Cas test #1	**Cas test #2**	**Cas test #3**	**Cas test #4**
Solution initiale aléatoire	10,831 ±0,289	33,847* ±1,331*	9,294 ±0.180	27,272 ±1,01
Phase d'adaptation sans RNA	10,421 +-0,092	33,504 ±0,209	8,428 ±0,102	28,798 ±0,617
Phase d'adaptation complète	10,290 ±0,080	33,030 +0,094	9,131 ±0,183	28,990 ±0,919

* : Certaines solutions irréalisables obtenues. Seules les solutions réalisables ont été compilées pour produire le résultat de la table.

L'hypothèse sous-jacente à l'utilisation d'une phase d'adaptation pour initialiser un algorithme génétique est qu'une meilleure solution initiale procure aussi une meilleure solution finale pour le processus. Les résultats de la table 7.11 et de la table 7.12 semblent confirmer cette hypothèse. En effet, pour les cas tests #1 et #2, où le RNA procure une solution initiale avantageuse, les algorithmes utilisant une phase d'adaptation complète performent en moyenne respectivement 5,3% et 2,5% mieux qu'un algorithme initialisé aléatoirement.

Pour le cas test #4, la solution initiale procurée par le RNA est de piètre qualité mais remplace uniquement une solution quelconque du problème et ne nuit pas significativement aux performances de l'algorithme par rapport à l'utilisation d'une phase d'adaptation sans RNA. Par contre, pour ce cas test, la mauvaise qualité des solutions initiales est telle que les solutions aléatoires initiales fournissent en moyenne un meilleur point de départ que les solutions initiales connues. Dans un tel cas, la phase d'adaptation nuit à l'algorithme au lieu de l'aider, procurant en moyenne une solution finale plus massive de 4,3%.

Le cas test #3, quant à lui, a aussi été choisi spécialement pour illustrer une lacune de la méthode actuelle. Dans ce cas particulier, la meilleure solution initiale connue au problème (qui est la $6^{ème}$ plus distance en terme de spécifications) est remplacée par la solution de mauvaise qualité donnée par le RNA lors de la phase d'adaptation. Cette situation est possible dans un cas comme celui-ci puisque $k_{île}$ est exactement égal à 6. Donc, pour ce cas test particulier, le RPC avec RNA procure tout de même un résultat plus avantageux d'environ 1,8%, alors que le même processus sans RNA aurait plutôt procuré, quant à lui, une solution plus légère de près de 10,3% par rapport à une initialisation aléatoire de l'algorithme.

On peut donc tirer de ces tests les conclusions suivantes :

- Si l'on dispose d'une banque de données suffisante, un processus utilisant le RPC pour initialiser un algorithme génétique semble avantageux par rapport à une initialisation aléatoire des individus de l'algorithme.
- Un processus de RPC utilisant un RNA substitut n'est avantageux que si le nombre et la constitution des solutions initiales pouvant servir à l'entraînement du réseau est suffisante pour que le RNA produise une solution de meilleure qualité que les solutions initiales existantes. Dans le cas contraire, la mauvaise estimation risque au contraire de nuire à la qualité de la solution générée.

7.10 Apport du raffinement progressif

La méthode décrite à la section 3.4.2, visant à insérer progressivement les variables les plus importantes d'abord dans l'optimisation, sera étudiée ici pour ce problème en comparant les résultats obtenus pour chacun des cas tests selon que l'on utilise tous les paramètres à la fois ($z_{max} = 1$) ou deux niveaux de raffinement ($z_{max} = 2$).

Puisque l'on considère ne posséder aucun RNA substitut à la fonction coût au départ, le processus utilisera plutôt la véritable fonction $F(\vec{x})$ pour évaluer les importances initiales des paramètres.

La table 7.13 illustre les résultats moyens obtenus après 20 lancements de chacun des algorithmes à divers nombres d'évaluations pour chacun des problèmes #1 à #4.

Table 7.13. Résultats moyens des cas tests effectués pour le raffinement progressif

	Cas test #1		Cas test #2		Cas test #3		Cas test #4	
Nombre d'évaluations	z_{max} = 1	z_{max} = 2	z_{max} = 1	z_{max} = 2	z_{max} = 1	z_{max} = 2	z_{max} = 1	z_{max} = 2
5000	12,084	11,551	23,525E+06*	10,862E+06*	10,437	10,114	38,005	37,561
10 000	10,862	10,660	12,223E+06*	3,062E+06*	10,212	9,964	34,430	33,475
25 000	10,543	10,393	34,314	33,514	9,242	9,334	30,691	29,800
50 000	10,256 ±0,126	10,290 ±0,080	34,272 +0,573	33,030 +0,094	8,949 ±0,211	9,131 ±0,183	27,151 ±0,860	28,990 ±0,919

* : Certaines des solutions de cette série de tests sont irréalisables. Pour des besoins de clarté, les pénalités de mort ont été exclues des solutions avant de calculer les moyennes (mais pas les pénalités adaptatives). La valeur représentée est donc une indication de la distance moyenne entre la solution et le domaine réalisable.

On observe que, à l'exception du cas test #2, les résultats donnés après 50 000 évaluations en utilisant le raffinement progressif (z_{max} = 2) sont plutôt décevants. Ceci revient à la discussion effectuée sur le sujet à l'annexe B.1.1, à savoir que lors d'une optimisation à long terme, le nombre élevé d'évaluations effectuées sans utiliser certaines variables nuit à l'exploration globale et est susceptible de provoquer un biais prématuré vers un optimum local.

Pour le cas test #2, on peut supposer que, vu l'étroitesse du domaine réalisable, l'utilisation des variables les plus importantes permet d'atteindre plus rapidement le domaine réalisable et donc de laisser plus d'évaluations pour l'optimisation du poids du disque.

On peut aussi constater que pour une optimisation à court terme (5000 ou 10 000 évaluations), l'utilisation des variables les plus importantes aurait permis pour tous les cas tests d'obtenir une solution finale de meilleure qualité.

Il est donc possible de tirer les conclusions suivantes de ces résultats :

- Le raffinement progressif semble susceptible d'améliorer les résultats de l'AGENT pour les problèmes d'optimisation pour lesquels on dispose de peu d'évaluations par rapport à la complexité du problème.
- Puisqu'elle ne tient pas compte de l'état de convergence de l'algorithme d'optimisation pour déterminer quand il faut insérer les nouvelles variables, la méthode d'implémentation du raffinement progressif étudiée dans cette thèse risque de nuire aux performances de l'AGENT lorsque l'utilisateur dispose de suffisamment de ressources pour laisser l'algorithme faire une exploration globale approfondie.

Le raffinement progressif suggéré ne devrait donc être employé que si l'utilisateur juge le problème très complexe pour le temps dont il dispose pour le résoudre.

7.11 Apport de l'opérateur de substitution

L'opérateur de substitution est un nouvel opérateur de recherche proposé à la section 5.5.3 de cette thèse ayant pour but d'intégrer de façon élégante un RNA substitut à un AG.

Le bénéfice potentiel de cet opérateur a déjà été démontré à la section 6.1.3 pour le cas des problèmes issus du MSG. Cette section vise donc à valider davantage l'utilisation de cet opérateur en observant les bénéfices obtenus pour l'un des cas tests étudiés dans ce chapitre.

Le cas test #1 a donc été résolu 20 fois en utilisant l'opérateur de substitution, avec un nombre maximal de points d'entraînement R_N = 500. Les résultats moyens obtenus sont comparés avec les résultats de l'AGENT sans opérateur de substitution dans la table 7.14.

Table 7.14. Comparaison entre les résultats avec ou sans opérateur de substitution pour le cas test #1.

	Poids final
Processus avec substitution	10,231 ±0,014
Processus sans substitution	10,290 ±0,080

Pour ce cas précis, l'opérateur de substitution a donc permis en moyenne une amélioration de la solution de l'ordre de 0,6%, en plus de réduire considérablement l'écart type sur les résultats.

Ce résultat, secondé par les résultats de la section 6.1.3 tendent à justifier l'ajout d'un modèle substitut à l'AGENT.

Il faut noter toutefois que, pour un nombre égal d'évaluations, le processus utilisant un opérateur de substitution utilise environ deux fois plus de temps pour calculer la solution que le processus n'utilisant pas de modèle substitut. L'auteur se permet donc de rappeler que, malgré l'apport évident des modèles substituts à un AG, ceux-ci devraient surtout être utilisés que lorsque la fonction F($\vec{x}$) est très lourde à calculer. En effet, le temps d'entraînement des RNA est relativement constant pour un même nombre de paramètres. Le ratio entre ce temps d'entraînement et le temps d'évaluation de F($\vec{x}$) est donc un facteur important à prendre en considération avant de décider de l'utilisation ou non de l'opérateur de substitution.

7.12 Utilisation de RNA en parallèle

Phase d'adaptation

La section 3.3 (figure 3.6) présente une architecture de RNA en parallèle permettant d'estimer les valeurs des paramètres $\vec{a}$ au cours de la phase d'adaptation.

L'objectif de cette section est de valider cette approche en la comparant avec approche classique (figure 3.4) qui propose de n'utiliser qu'un seul RNA avec *N* neurones à la sortie.

À cette fin, 20 architectures de RNA ont été entraînées à partir des solutions initiales du problème pour chacune des méthodes selon la procédure d'apprentissage décrite à l'annexe A.4 de cette thèse.

Les figures 7.15 et 7.16 comparent l'erreur quadratique moyenne mesurée à la fin de l'entraînement pour chacune des deux architectures de réseau et pour chaque variable des deux modèles de disque utilisés.

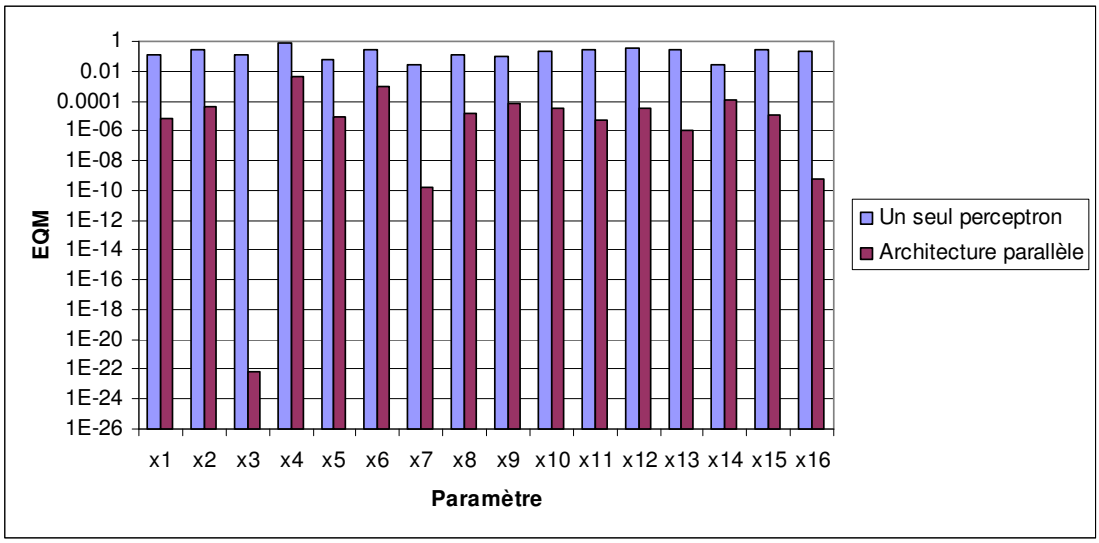


Figure 7.15 : Comparaison des erreurs quadratiques moyennes pour chaque paramètre du modèle de disque symétrique.

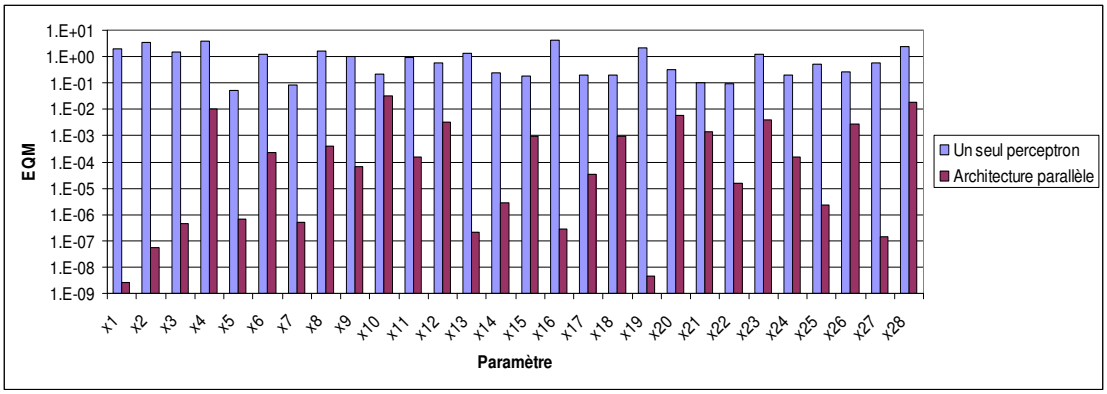


Figure 7.16 : Comparaison des erreurs quadratiques moyennes pour chaque paramètre du modèle de disque asymétrique.

On observe une différence énorme entre le RNA entraînant tous les paramètres simultanément et les RNA entraînant chacun un seul et unique paramètre à sa sortie. Ceci est dû, entre autres, au fait que chacun des paramètres dispose d'une complexité qui lui est propre. Puisque chaque neurone utilisé permet de représenter approximativement une inflexion dans la valeur de la fonction approximée, chaque paramètre nécessite donc un nombre de neurones différents sur la couche cachée, ce qui n'est pas possible avec l'architecture typique du RNA utilisé : le perceptron (voir annexe A).

De plus, on sait que le problème d'entraînement d'un réseau de neurones revient en fait à un problème d'optimisation où l'on minimise l'erreur quadratique sur la valeur de sortie du réseau par rapport aux paramètres. Or, comme dans la plupart des problèmes d'optimisation disposant d'un grand nombre de variables, ces problèmes d'entraînement disposent d'une grande quantité d'optimums locaux. Ainsi, dans le cas de l'entraînement de tous les paramètres simultanément, on se contente de trouver l'optimum donnant la plus petite erreur moyenne à la sortie, alors que, dans le cas de l'entraînement d'un seul paramètre par réseau, chaque réseau choisit l'optimum le plus avantageux pour ce paramètre en particulier.

Il est à noter que, vu la topologie plus complexe du réseau, le temps d'entraînement de la topologie proposée est légèrement supérieur au temps d'entraînement d'un seul RNA pour un même nombre de passes d'entraînement (environ 20% en moyenne pour le cas étudié). Toutefois, la différence d'ordre de grandeur entre les performances des deux architectures semble tout de même justifier l'utilisation d'un RNA en parallèle.

En conclusion, bien que cette architecture de RNA soit plus complexe à mettre en œuvre que le perceptron régulier, elle semble donner une erreur plus faible de plusieurs ordres de grandeur sur ses résultats pour un même nombre de passes d'entraînement.

Modèle substitut

On se rappelle (section 3.4.4), que cette thèse suggère comme modèle substitut une architecture où plusieurs RNA opèrent en parallèle pour générer la valeur de la fonction coût (figure 3.10) au lieu d'un seul réseau donnant directement la valeur de F($\vec{a}$). Afin de valider cette approche, l'erreur relative moyenne a été mesurée pour toutes les solutions suggérées par l'opérateur de substitution de l'AGENT lors des tests de la section 7.11.

À des fins de comparaison, un second RNA substitut a été construit avec les mêmes patrons d'entraînement, utilisant cette fois l'architecture classique constituée d'un seul perceptron. L'erreur relative moyenne donnée par ce réseau a été mesurée pour les mêmes solutions, données par l'opérateur de substitution.

En moyenne, l'architecture en parallèle affiche un écart moyen de 4446% avec la véritable valeur adaptative.

L'architecture classique, quant à elle, affiche en moyenne un écart de 43 301%.

Malgré l'aspect démesuré du pourcentage d'erreur, il faut garder à l'esprit que F($\vec{x}$) inclut une pénalité de mort énorme lorsqu'une contrainte est enfreinte. Ainsi, lorsque les RNA estiment qu'une solution est infaisable alors que ce n'est pas le cas, la valeur de l'erreur atteint une proportion impressionnante. En comparaison, le RNA de l'architecture en parallèle estimant le poids du disque sans contraintes (c'est-à-dire G($\vec{x}$)) affiche une erreur moyenne de 3,97%.

Malheureusement, pour l'architecture classique, il est impossible de dissocier G($\vec{x}$) et H($\vec{x}$) à la sortie du réseau puisque ces valeurs sont estimées à l'intérieur même du RNA. Une comparaison plus poussée n'est donc pas possible avec les données actuelles. Toutefois, la différence d'erreur relative moyenne mesurée, de même que les bons résultats obtenus pour une architecture en parallèle lors de la phase d'adaptation (voir figures 7.15 et 7.16) laissent penser que cette architecture est aussi avantageuse dans le cas du modèle substitut à la fonction, d'où son utilisation dans les sections 6.1.3 et 7.11.

7.13 Avantages d'utiliser les solutions comme centres d'îlots

Puisqu'il n'y a que peu de solutions utilisant le modèle asymétrique, le RNA de la phase d'adaptation n'est pas particulièrement apte à produire une solution de départ convenable pour le cas test #4. Si les valeurs de p_5, p_6 et de p_7 du problème sont trop différentes des valeurs de p_5, p_6 et de p_7 inscrites pour les solutions utilisant ce modèle (18 à 20), le procédé ne générera pas une solution de départ intéressante pour le modèle asymétrique. Un algorithme génétique utilisant la procédure d'initialisation de la population telle que proposée par Johnson & Sushil (2005) (c'est-à-dire un algorithme qui inscrit comme population initiale les solutions générées par le RPC) écartera donc rapidement ce modèle, qui est pourtant très prometteur dans les cas où p_4 est suffisamment plus grand que 0.

En utilisant plutôt les solutions comme éléments centraux d'un îlot de population, l'algorithme disposera tout de même d'un laps de temps intéressant pour raffiner la solution du modèle asymétrique avant que celles-ci n'entrent en compétition avec les solutions du modèle symétrique semblant à prime abord de meilleure qualité.

La table 7.10 compare les résultats obtenus par le procédé actuel selon que l'on initialise la population de l'algorithme de la façon proposée à la section 3.4.3, ou selon la méthode de Johnson & Sushil (2005) en utilisant qu'un seul et unique îlot contenant toutes les 20 solutions initiales, ainsi que certaines solutions aléatoires additionnelles pour compléter.

Table 7.15. Comparaison entre deux méthodes d'initialisation de l'algorithme

	Cas test #1	**Cas test #2**	**Cas test #3**	**Cas test #4**
Initialisation de Johnson & Sushil	10,630 ±0,346	33,452 ±0,536	9,206 ±0,214	**
Phase d'adaptation complète	10,290 ±0,080	33,030 +0,394	9,131 ±0,183	28,990 ±0,919

** : Aucune solution réalisable obtenue au cours des 20 optimisations

Ce que l'on constate en observant les résultats de la table 7.15, c'est que le type d'initialisation utilisé pour l'algorithme dans cette thèse donne en moyenne de meilleurs résultats que l'initialisation de Johnson & Sushil pour chacun des quatre cas tests étudiés, par un écart relatif respectif de 3,2%, 1,3% et 0,8% pour les cas test #1-#3, et par un écart très important dépendant des pénalités de contraintes violées dans le cas tu test #4.

En effet, pour le cas test #4, l'initialisation de Johnson & Sushil appliquée à l'AGENT procure un résultat de très mauvaise qualité. En effet, cette méthode n'ayant pas été construite pour gérer plusieurs modèles de solution différents, elle a vite éliminé le modèle paramétrique asymétrique et donne un résultat utilisant le modèle symétrique alors que ce dernier n'est pas de mise pour ce cas test.

La grande différence entre la méthode proposée dans cette thèse et la méthode de Johnson & Sushil (2005) est que la méthode de cette thèse lie chaque population à une solution initiale (et génère un essaim d'individus autour de cette solution), alors que Johnson & Sushil (2005) lient plutôt chaque individu de leur unique population à une solution différente.

Or, on se rappelle (voir section 7.9) que plus les solutions initiales sont rapprochées des solutions finales de l'algorithme, plus celui-ci est susceptible de fournir une solution intéressante à notre problème. Johnson & Sushil (2005) utilisent davantage de solutions initiales (par exemple, dans le cas étudié, leur méthode en utilise un nombre $N_{pop} = 40$ au lieu de $k_{ile} = 6$). Ainsi, puisque les solutions initiales du problème actuel sont plutôt éloignées les unes des autres, les individus initiaux générés par la méthode proposée dans cette thèse disposent en général d'une meilleure valeur adaptative moyenne au départ et procurent de meilleures chances à l'algorithme d'atteindre une solution intéressante dans les temps prescrits.

Par contre, on se souvient aussi que les solutions les plus rapprochées en terme de spécifications ne sont pas nécessairement toujours les solutions ayant les meilleures valeurs adaptatives pour un problème donné (par exemple, voir la discussion dans la section 7.9 à propos du cas test #3). Ainsi, malgré ses bons résultats, on peut supposer que la méthode de cette thèse, qui utilise moins de solutions initiales, dispose d'un risque plus élevé d'exclure la véritable meilleure solution initiale au problème si celle-ci n'est pas la solution disposant des spécifications les plus rapprochées des spécifications du problème. Si la fonction coût n'est pas très longue à évaluer, un designer utilisant cette méthode aurait donc intérêt à vérifier la valeur adaptative de la plupart des solutions rapprochées connues avant de choisir le nombre $k_{île}$ d'îlots à utiliser.

De plus, puisque la méthode proposée s'appuie au départ sur $k_{île}$ solutions différentes, il est important que le designer qui utilise cette méthode s'assure que tous les modèles qu'il juge compétitifs soient initialement représentés sur au moins l'un des îlots avant de lancer l'algorithme de résolution.

7.14 Comparaison avec certaines méthodes d'optimisations utilisées en industrie

À la connaissance de l'auteur, le logiciel I-SIGHT-FD est un logiciel très répandu en milieu industriel pour résoudre les problèmes de design de pièces mécaniques.

Cette section de la thèse compare donc les résultats obtenus par certains des algorithmes d'optimisation pertinents de ce logiciel avec les résultats de la méthode de design proposée utilisant l'AGENT.

Quatre méthodes d'optimisations comprises dans le logiciel ont été choisies, soit un AG insulaire de type AGR (voir sections D.3.1 et D.6.2), un algorithme de type GPS nommé Hookes & Jeeves (voir section 4.2), un algorithme à gradient de type SQP (voir section 4.1) et un algorithme nommé POINTER, qui est en fait un algorithme hybride utilisant successivement quatre algorithmes afin d'explorer le domaine à la fois globalement et localement. Les algorithmes utilisés par POINTER sont un solveur linéaire, un algorithme à gradient de type SQP, l'algorithme du simplexe de Nelder & Mead et un algorithme génétique.

La version du logiciel qui a été utilisée est I-SIGHT 3.5. Plus de détails sur chacun des algorithmes utilisés peuvent être retrouvés dans la documentation du logiciel en question.

Les algorithmes possédant des opérateurs aléatoires (POINTER et l'AG insulaire) ont été lancés vingt fois pour résoudre le cas test #1 et les résultats moyens ont été comptabilisés. Les autres algorithmes n'ont été lancés qu'une seule fois.

Les paramètres utilisés par l'AG insulaire sont les paramètres par défaut suggérés pour les AGR (voir table 6.3), avec les paramètres de migration correspondant à $I_M = 10$ et $N_M = 2$.

Les valeurs utilisées pour les autres algorithmes sont celles suggérées par I-SIGHT 3.5, à l'exception du pas de l'algorithme à gradient, qui a été fixé à $1x10^{-8}$ puisque la valeur par défaut ne donnait aucune amélioration du résultat.

Le critère d'arrêt a été fixé à la convergence de l'algorithme à moins de $1x10^{-8}$ (pour Hookes & Jeeves et le SQP) ou à l'atteinte de 50 000 évaluations de fonction pour POINTER et l'AG insulaire.

Lorsque requise (c'est-à-dire pour tous les algorithmes sauf l'AG insulaire), la solution initiale fournie à l'algorithme est la solution procurant la meilleure valeur adaptative parmi les solutions initiales connues au problème (voir solution 5 dans l'annexe C). Ainsi, l'algorithme débute le problème comme s'il avait été initialisé par la phase d'adaptation du processus de RPC, mais sans RNA. Ceci est simplement dû au fait qu'au moment de débuter les tests, l'auteur ne disposait pas encore de la solution du RNA d'adaptation du processus.

La seule conséquence de cette initialisation est que pour comparer les divers algorithmes sur une même base, les résultats de l'AGENT utilisant la phase d'adaptation sans RNA devront être utilisés. Pour l'AG insulaire de I-SIGHT, qui est initialisé aléatoirement, une comparaison plus équitable serait plutôt entre lui et les résultats de l'AGENT ayant aussi été initialisé aléatoirement. Ces résultats sont disponibles sur la table 7.12, et

seront réutilisés dans la table 7.16, qui affiche les solutions obtenues par les algorithmes de I-SIGHT 3.5.

Table 7.16. Comparaison entre l'AGENT et quelques algorithmes de I-SIGHT 3.5 pour le cas test #1.

Algorithme de comparaison	Valeur adaptative	Nombres d'évaluations avant la convergence	Valeur adaptative de l'AGENT ayant la même initialisation	Nombres d'évaluations avant la convergence de l'AGENT
SQP	13,003	220	10,421 ±0,092	50 000
Hookes & Jeeves	11,736	613	10,421 ±0,092	50 000
AG insulaire	14,235 ±2,171	50 000	10,831 ±0,289	50 000
POINTER	10,889 ±0.124	50 000	10,421 ±0,092	50 000

SQP

On constate que, tel qu'attendu pour la topographie étudiée, l'AGENT offre un disque de turbine en moyenne plus léger de respectivement 20,0% et 11,2% par rapport aux algorithmes SQP et Hookes & Jeeves, qui n'effectuent pas de recherches globales approfondies.

N'ayant amélioré la solution initiale que de 0,14%, l'algorithme de SQP n'est manifestement pas adapté à un problème de ce type, qui débute dors et déjà avec une solution près d'un optimum.

Hookes & Jeeves

Par contre, l'algorithme de Hookes & Jeeves, bien qu'offrant une solution finale de moins bonne qualité que l'AGENT, réussit à améliorer considérablement la solution très rapidement. En effet, à sa convergence après seulement 613 évaluations, l'algorithme

dispose déjà d'une solution plus légère de 9,8%. À titre comparatif, après ce même nombre d'évaluations, l'AGENT disposait plutôt en moyenne d'une solution plus légère de 0,3% en moyenne. Cet algorithme serait donc à privilégier sur l'AGENT dans un cas où l'utilisateur dispose de très peu d'évaluations pour résoudre son problème.

AG insulaire

Un autre fait impressionnant est l'écart énorme de 23,9% entre les résultats moyens de l'AGENT et de l'AG insulaire. Cet écart justifie la création de l'AGENT, et peut s'expliquer par la lenteur de convergence typique des AG insulaires à nombres réels. En effet, pour un problème de cette envergure, un AG régulier nécessiterait probablement un nombre d'évaluations plus grand de plusieurs ordres de grandeur avant de potentiellement découvrir l'un des meilleurs optimums de la fonction.

POINTER

L'algorithme ayant procuré le meilleur résultat parmi les algorithmes de I-SIGHT 3.5 est l'algorithme hybride POINTER. Malgré cela, ses performances sont de 4.3% inférieures à celles de l'AGENT pour le problème étudié.

POINTER est un algorithme conçu pour être utilisable même par un novice en optimisation; tous ses paramètres sont donc gérés automatiquement, à l'exception du nombre d'évaluations désirées. On peut donc prétendre sans aucun doute que POINTER a été utilisé au maximum de ses capacités pour résoudre le problème de cette thèse.

Toutefois, puisque cette gestion de paramètres ainsi que le fonctionnement détaillé de l'algorithme sont protégés et donc non disponibles à l'auteur, il est difficile de discuter des raisons de la supériorité de l'AGENT sur POINTER dans ce cas particulier.

L'auteur aimerait d'ailleurs tempérer les résultats obtenus en ajoutant que le fait que l'AGENT ait procuré des solutions supérieures pour ce problème de design particulier

ne signifie pas nécessairement que l'AGENT soit supérieur aux autres algorithmes étudiés pour tous les problèmes d'optimisation du design de pièces mécaniques. Par contre, on peut conclure que ces résultats classent l'AGENT comme un excellent candidat potentiel pour résoudre ce type de problème, et qu'ils justifient par conséquent des recherches futures dans cette direction.

Chapitre 8.
CONCLUSION

Tel que discuté dans l'introduction, l'implémentation d'un système automatisé d'aide à la décision en design peut accorder un avantage décisif à une entreprise œuvrant dans le design de pièces mécaniques.

Par contre, les systèmes actuels d'aide à la décision en design disposent de plusieurs restrictions (comme les difficultés d'implémentation et le coût élevé d'utilisation du système) qui limitent leur usage actuel au cas des pièces les plus simples.

Volet I : Automatisation du processus de design et système d'aide à la décision

Le besoin de systèmes d'aide à la décision plus performants a donc généré le premier objectif de cette recherche, qui était d'appliquer un processus de RPC de façon à tirer avantage des algorithmes d'optimisation, tout en réduisant autant que possible le coût d'utilisation du processus.

Dans cette optique, plusieurs améliorations originales ont été proposées pour le processus de RPC :

- **Scinder la phase d'adaptation en une phase d'adaptation et une phase de raffinement.**

  La phase d'adaptation du processus a été scindée et deux phases distinctes, soit une phase d'adaptation utilisant le savoir connu à priori sur le domaine, et une phase de raffinement visant à acquérir et utiliser un savoir supplémentaire sur le domaine de design.

Le processus proposé au chapitre 3 utilise le RPC et un RNA pour générer une série de solutions à la sortie de la phase d'adaptation. Ensuite, un AG nommé l'AGENT utilise une quantité $k_{île}$ de ces solutions comme points centraux pour ses populations et tente d'améliorer ces solutions lors de la phase de raffinement. Les bons résultats obtenus à la section 7.9 démontrent l'efficacité de cette méthode pour le cas étudié.

En effet, l'AGENT s'est montré apte à produire une solution acceptable pour tous les cas tests même lorsque l'on dispose de peu d'informations préalables sur le système. Toutefois, lorsqu'une quantité suffisante d'informations antérieures est disponible, le savoir initial fourni par la phase d'adaptation a permis d'obtenir une solution finale plus légère dans les mêmes délais pour tous les cas étudiés.

Il est à noter, toutefois, que cette approche dispose d'une limitation lorsque très peu d'information est connue sur le domaine (comme dans le cas des tests #3 et #4 du chapitre 7). En effet, lorsque les solutions aléatoires générées sont de meilleure qualité que celles fournies par le processus de RPC, celui-ci ne sert alors qu'à empêtrer le processus en début d'optimisation. De plus, l'utilisation d'un RNA dans un tel cas peut potentiellement nuire à l'efficacité de l'algorithme de raffinement en forçant la mise à l'écart d'une solution connue pour la remplacer par une mauvaise interpolation.

- **Utiliser le raffinement progressif pour écarter dans un premier temps les variables ayant une influence très faible sur la valeur adaptative lors de l'optimisation.**

  Afin d'accélérer la convergence des algorithmes et d'éviter d'avoir à utiliser inutilement certaines variables, la section 3.4.2 de ce document propose une méthode basée sur la sensibilité de $F(\bar{x})$ ou de $\widetilde{F}(\bar{x})$ à chacune des variables pour déterminer l'importance de celles-ci et écarter les variables de moindre importance de

l'optimisation. Ces variables seront ensuite progressivement insérées dans le processus en cours d'optimisation.

Les résultats contenus dans cette thèse à l'annexe B.1 et la section 7.10 démontrent les gains possibles de cette approche pour les cas où l'on ne dispose pas de beaucoup d'évaluations possibles par rapport à la complexité du problème (par exemple dans le cas d'un problème disposant d'une quantité phénoménale de variables, ou d'un problème très long à évaluer). Par contre, le choix des seuils d'importances minimales $S^0_{\min}$ et du moment où il est le plus opportun d'incorporer de nouvelles variables dépend beaucoup de la topologie particulière du problème et des recherches futures seraient nécessaires à la découverte d'une façon plus adéquate de gérer le raffinement progressif. L'intuition de l'auteur est qu'une méthode plus efficace de raffinement progressif devrait évaluer plus fréquemment l'importance relative $\bar{S}_r$ des variables et devrait être basée sur un critère de convergence (par exemple sur ΔD^g) pour choisir en continu le seuil minimal acceptable pour qu'une variable soit incluse dans l'optimisation.

Dans le même ordre d'idée, une amélioration permettant le calcul de l'importance des variables en temps réel au cours de l'optimisation pourrait permettre au système d'évaluer la stabilité de la solution aux environs de l'optimum.

- **Utiliser une méthode de gestion hybride des pénalités à la valeur adaptative données par les transgressions de contraintes.**
  La méthode proposée à la section 3.2.1 utilise à la fois une pénalité constante (dite pénalité de mort) et une pénalité proportionnelle adaptative. Par rapport aux méthodes traditionnelles, cette méthode dispose de l'avantage de fonctionner aisément sans nécessiter véritablement d'ajustements de paramètres, tout en guidant l'optimiseur petit à petit en direction du domaine réalisable lorsqu'aucune solution initiale réalisable n'est connue. Les excellents résultats obtenus dans les sections B.1.2 et 6.2

démontrent l'aspect très compétitif de cette approche pour divers cas tests lors de l'utilisation de l'AGENT.

Volet II : Raffinement du design et optimisation

Les techniques d'optimisation disponibles actuellement disposent ou bien de la limitation d'être très lentes à converger (comme c'est le cas des AG), ou bien d'être incapables de créativité aléatoire (dans le cas des autres algorithmes). Cette créativité, de même que la stabilité des AG, est pourtant voulue dans le cadre d'un processus de design.

L'objectif spécifique de recherche du second volet de cette thèse était donc de développer et de valider un AG capable de s'appuyer sur l'information topologique connue de la fonction coût pour converger plus rapidement que les algorithmes génétiques actuels dans le cas des problèmes de design de pièces mécaniques.

Dans cette optique, l'algorithme génétique à évolution de noyaux territoriaux (l'AGENT) a été développé au chapitre 5 de cette thèse. Cet algorithme est essentiellement un AGR modifié de façon à pouvoir traiter de plusieurs modèles paramétriques différents de solutions simultanément, et à utiliser deux heuristiques originales :

- **L'opérateur de substitution**

  L'opérateur de substitution décrit à la section 5.5.3 est une façon originale et pratique de considérer l'apport d'un modèle substitut à la fonction coût d'un problème sans nuire grandement aux diverses autres fonctionnalités des AG.

  Les résultats des sections 6.1.3 et 7.11 démontrent que cet opérateur semble améliorer les résultats obtenus par l'AGENT pour un même nombre d'évaluations de fonction. Toutefois, l'expérience acquise par l'auteur lors de l'élaboration de cette thèse

suggère qu'un utilisateur devrait être prudent avant d'utiliser cet opérateur de recherche. L'entraînement en temps réel d'un RNA substitut à la fonction coût est une tâche souvent utile, mais plutôt longue en temps de calcul. Pour les problèmes où la valeur de $F(\vec{x})$ peut être évaluée très rapidement, il peut être plus efficace de permettre davantage d'évaluations pour l'algorithme optimisation (ou de le relancer plusieurs fois d'affilée) que de perdre un temps précieux à l'entraînement d'un RNA substitut.

- **Les noyaux territoriaux**

  Les noyaux territoriaux sont le cœur de l'AGENT, et probablement la contribution la plus importante de ce doctorat. Appliquée dans le cadre de l'AGENT, cette heuristique a permis d'améliorer considérablement les performances de l'AGR pour le problème du MSG (voir section 6.1).

  L'AGENT n'utilisant que cette heuristique (c'est-à-dire sans opérateur de substitution) a aussi démontré d'excellentes performances par rapport à tous les autres algorithmes étudiés dans le cas des problèmes d'optimisation avec contraintes (section 6.2).

  D'un autre côté, on se rappelle (voir section 6.3) que l'AGENT offre des performances passables sur les problèmes hautement multimodaux non contraints par rapport aux autres AG utilisant des opérateurs de croisement plus évolués (tels que CIXL-2) que l'opérateur de croisement des AGR réguliers. Une autre avenue de recherche intéressante pourrait être de vérifier si l'utilisation d'un opérateur de croisement évolué à parents multiples pourrait permettre d'améliorer les performances de l'algorithme, à la fois pour les problèmes d'optimisation non contraints et pour les problèmes de design de pièces mécaniques.

Aussi, puisque l'AGENT procure en général une série d'individus mieux répartis les uns des autres que les autres AG, il serait aussi intéressant de vérifier dans divers travaux futurs si un algorithme hybride AGENT – algorithme à gradient pourrait être plus performant que l'AGENT, particulièrement pour les problèmes où le nombre d'évaluations de fonction est très limitée par rapport à la complexité du problème.

Dans le cas du problème de design d'un disque de turbine à gaz (chapitre 7), l'AGENT a permis d'obtenir des résultats très compétitifs, devançant de 3.9% le résultat donné par l'algorithme ayant fourni la meilleure performance parmi les quatre algorithmes testés du logiciel I-SIGHT 3.5, soit l'algorithme POINTER.

À la connaissance de l'auteur, le logiciel I-SIGHT est très utilisé en industrie pour résoudre les problèmes de design de pièces mécaniques. Bien que des travaux futurs soient nécessaires pour voir si l'AGENT performe aussi bien sur d'autres problèmes de design industriel que celui décrit au chapitre 7, les bons résultats obtenus en comparaison avec les algorithmes de ce logiciel permettent tout de même de supposer que l'AGENT, secondé par un processus de RPC, est un solide candidat potentiel pour l'automatisation de problèmes de design d'envergure industriel.

RÉFÉRENCES

Aamodt, A., & Plaza, E. (1993). Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, *7*(1), 39-59.

Aamodt, A., & Plaza, E. (1994). Case-based reasoning: foundational issues, methodological variations, and system approaches. *Artificial Intelligence Communications*, 7, 39–59.

Acan, A., & Tekol, Y. (2005). Performance-based computation lifetimes in genetic algorithms. *Knowledge Incorporation in Evolutionary computation* (pp.195-214). Berlin: Springer.

Andersson, J., & Redhe, M. (2003). Response surface methods and pare to optimization in crashworthiness design [version électronique]. *Proceedings of DETC'03 ASME 2003 Design Engineering Technical Conference in Computers and Information in Engineering Conference* (DETC2003/DAC - 48752). Chicago.

Antonisse, J., (1989). A new interpretation of schema notation that overturns the binary encoding constraint. *Proc. 3rd Int. Conf. Genetic Algorithms and their Applications* (pp. 8-97). San Fransico: Morgan Kaufmann.

Audet, C., & Dennis, J.E. (2004). *Mesh adaptive direct search algorithms for constrained optimization*. Houston: Tech. Rep. TR04-02, Department of Computational and Applied Mathematics, Rice Univeristy.

Audet, C., & Dennis, J.R. (2003). Analysis of generalized pattern search. *SIAM Journals of Optimization*, *13*(3), 889-903.

Audet, C., & Orban, D. (2004). Finding optimal algorithmic parameters using the mesh adaptive direct search algorithm. *SIAM Journal on Optimization*, *17*, 642-667.

Avramenko, Y., & Kraslawski, A. (2008). *Case Based Design: Applications in Process Engineering*. New York: Springer.

Bertsekas, D.P. (1995). *Nonlinear Programming*. Massachusetts: Athena Scientific.

Bogaerts, S., & Leake, D. (2004). Facilitating cbr for Incompletely-described Cases: Distance Metrics for Partial Problem Descriptions. *Advances in case-based reasoning : ECCBR 2004 : European conference on case-based reasoning No7* (pp. 62-76). Berlin: Springer.

Boggs, P.T., Domich, P.D., Rogers, J.E., & Witzgall, C. (1996). An interior point method for general large scale nonlinear programming problems. *SIAM Journal on Optimization*, *9*(3), 755-778.

Boggs, P.T., Kearsley, A.J., & Tolle, J.W. (1999). A global convergence analysis of an algorithm for large scale nonlinear programming problems. *SIAM Journal on Optimization*, *9*(4), 833-862.

Boggs, P.T., Kearsley, A.J. & Tolle, J.W. (1999). A practical algorithm for general large scale nonlinear optimization problems. *SIAM Journal on Optimization*, *9*(3), 755-778.

Boggs, P.T., & Tolle, J.W. (1995). Sequential quadratic programming. *Acta Numerica*, *4*, 1-51.

Bonnans, J.F., Gilbert, J.-Ch., Lemarechal, C., & Sagastizabal, C. (1997). *Optimisation Numérique – Aspects théoriques et pratiques Vol. 27*. Berlin: Springer.

Booker, A.J., Dennis, J.E. Jr., Frank, P.D., Serafini, D.B., Torczon, V., & Trosset, M.W. (1998). A rigorous framework for optimization of expensive function by surrogates. *Structural Optimization*, *17*(1), 1-13.

Bosman, P. A. N., & Thierens, D. (2005). Learning probabilistic models for enhanced evolutionary computation. *Knowledge Incorporation in Evolutionary Computation* (pp. 147-176). Berlin: Springer.

Braha, D. & Maimon, O. (1998). *A Mathematical Theory of Design: Foundations, Algorithms and Applications*. Dordrecht: Kluwer.

Brown, D., Chandrasekaran, B. (1985). Expert System for a class of mechanical design activity. In Gero, J. (éd.), *Knowledge Engineering in Computer-Aided Design*. Amsterdam: Elsevier Science Pub Co.

Burns, S.A. (2002). *Recent Advances in Optimal Structural Design [Version électronique]*. : ASCE Committee on Optimization of the Committee on Electronic Computation. Canfield, R.A. (2004). Multipoint cubic surrogate functions for sequential approximate optimization. *Third ISSMO/AIAA Internet Conference on Approximation in Optimization* (Vol. 27(5), pp. 326-336). Berlin: Springer.

Cantú-Paz, E. (1998). A survey of parallel genetic algorithms. *Calculateurs Parallèles, Réseaux et Systèmes Répartis*, *10*(2), 141-171.

Chambers, D., Lehman, S., & Dwola, F. (2007). Model-based Layer Estimation using a Hybrid Genetic/Gradient Search Optimization Algorithm. *Adaptive Sensor Array Processing Workshop* (pp. UCRL-CONF-231266). Boston: Lawrence Livermore National Laboratory.

Chandrasekaran, B. (1990). Design Problem Solving: A Task Analysis. *AI Magazine*, 11 (4), pp.59-71.

Choi, K.K., & Kim, N.H. (2005). *Structural Sensitivity Analysis and Optimization: Linear Systems*. Berlin: Springer.

Coleman, T.F., & Hubert, L.A. (1989). A direct active ser algorithm for large scale sparse quadratic programs with simple bounds. *Mathematical Programming*, *67*(2), 189-224.

Coleman, T.F., & Li, Y. (1990). *Large Scale Numerical Optimization*. Philadelphie: SIAM.

Conn, A.R., Gould, N. I. M., & Toint, Ph. L. (1998). Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, *25*(2), 433-460.

Conn, A.R., Gould, N. I. M., & Toint, Ph. L. (2000). *Trust-Region Methods*. Philadelphie: SIAM.

Cross, N. (2000). *Engineering Design Methods: Strategies for Product Design*. Chichester: Wiley.

Dagli, C. (1994). *Artificial neural networks for intelligent manifacturing* (1ère éd.). Londre: Chapman & Hall Publisher.

Das, I. (1997). *Nonlinear Multicriteria Optimization and Robust Optimality.* Thèse de doctorat en Mathématiques appliqués, Rice University, Texas, Houston, États-Unis.

Dasgupta, S. (1989). The Structure of Design Processes. In Yovits, M.C. (éd.), *Advance in Computers* (28, pp. 1–67). New York: Academic Press.

De Jong, K.A. (1975). *An analysis of the behaviors of a class genetic adaptive systems* Thèse de doctorat, University of Michigan, Ann Arbor, États-Unis.

De Jong, K.A., Potter, M.A., & Spears, W.M. (1997). Using Problem Generators to Explore the Effects of Epistasis. *Proceedings of the Seventh International Conference on Genetic Algorithms* (pp. 338-345). Michigan: Morgan Kaufmann.

Dennis, J.E., & Schnabnel, R.B. (1983). *Numerical methods for unconstrained optimization and nonlinear equations*. New Jersey: Prentice-Hall.

Di Pillo, G., & Murli, A. (2003). *High Performance Algorithms and Software in Nonlinear Optimization*. Dordrecht: Kluwer Academic Publishers.

Dym, C.L., & Levitt, R.E. (1991). *Knowledge-Based Systems in Engineering*, New York: McGraw-Hill.

Dym C.L., & Little, P. (2004). *Engineering Design: A Project-Based Introduction*. Chichester: Wiley.

Eiben, A.E., & Bäck, Th. (1997). Empirical investigation of multi-parent recombination operators in evolution strategies. *Evolutionary Computation*, *5*(3), 347-365.

Emmerich, M., & Giotis, A., & Ozdemir, M., & Back, T, & Giannakoglou, K. (2002). Metamodel assisted evolution strategies . *Parallel Problem Solving from Nature VII* (pp. 362-370). Berlin: Springer.

Facchinei, F., & Judice, J., & Soares, J. (1998). An active set Newton algorithm for large-scale nonlinear programs with box constraints. *SIAM Journal on Optimization*, *8*(1), 158-186.

Fahlman, S.E. (1988). Faster-learning variations on backpropagation: An empirical study. *Proceedings of the 1988 Connetionist Models Summer School* (pp. 38-51). San Mateo: Morgan Kaufmann Ed.

Falhman, L.E., & Lebiere, C. (1989). The cascade-correlation learning architecture. *Advances in Neural Information Processing System* (pp. 524-532). San Mateo: D.S. Touretzy Eds.

Fletcher, R. (1987). *Practical Methods of Optimization* (2e éd.). Chichester: J. Wiley and Sons.

Fletcher, R. (1987). Recent developments in linear and quadratic programming. In Jacobs, D.A.H., *The state of the art in Numerical Analysis* (pp. 213-243). Oxford: Oxford University Press.

Fletcher, R., & Layffer, S. (2002). Nonlinear programming without a penalty function. *Mathematical Programming*, *91*(2), 239-269.

Forsgren, A, Gill, P.E., & Wright, M.H. (2002). Interior-point methods for nonlinear optimization. *SIAM Review* , *44*(4), 525-597.

French, M.J. (1985). *Conceptual Design for Engineers*. London: Design Council.

Gallagher, M. , & Yuan, B. (2006). A General-Purpose Tunable Landscape Generator. *IEEE Transactions on Evolutionary Computation* (Vol. 10(5), pp. 590-603). Piscataway : IEEE-Inst Electrical Electronics Engineers Inc.

Gani, R. (2004). Chemical product design: challenges and opportunities. *Computers and Chemical Engineering*, 28, 2441–2457.

Gaviano, M., Kvasov, D.E., & Lera, D. (2003). Algorithm 829: Software for Generation of Classes of Test Functions with Known Local and Global Minima for Global Optimization. *ACM Transactions on Mathematical Software*, *29*(4), 469-480.

Gero, J.S. (1990). Desing Prototypes: A Knowledge Representation Schema for Design. *AI Magazine*, 11 (4), 26–36.

Gertz, E.M., Nocedal, J., & Sartenaer, A. (2003). *A starting point strategy for nonlinear interior methods*. Evanston: Technical report OTC 2003/4. Optimization Technology Center.

Gill, P.E., Murray, W., & Saunders, M.A. (2002). *Examples of ill-behaved central paths in convex optimization*. Rocquencourt: Technical report 4179, INRIA.

Gill, P.E., Murray, W., & Wright, M.H. (1981). *Practical Optimization*. Londre: Academic Press.

Goldberg, D.E. (1989). Genetic Algorithms in Search, Optimization and Machine Learning [version électronique]. *Reading MA Addison Wesley*. Illinois: University of Illinois.

Goldsmith, M.J. (1999). *Sequential quadratic programming methods based on indefinite Hessian approximations* Thèse de doctorat en science et ingénierie, Stanford University, California, États-Unis.

Gondzio, J., & Grothey, A. (2003). *Parallel interior point solver for structured quadratic programs: Application to financial planning problems*. Édinbourg: Technical report MS-03-001, School of Mathematics. University of Edinburgh.

Gong, Y., Nakamura, M., & Tamaki, S. (2005). Parallel Genetic Algorithms on Line Topology of Heterogeneous Computing Resources. *Proceedings of 2005 Conference on Genetic and Evolutionary Computation* (pp. 147-1454). Washington: ACM.

Gould, N. I. M. (2003). Some reflections on the current state of active-set and interior point methods for constrained optimization. *SIAG/OPT Views-and-news*, *14*(1), 2-7.

Gould, N. I. M., Orban, D., & Toint, Ph. L. (2004). *Numerical methods for large-scale non linear optimization*. Oxfordshire: CCLRC Technical Report RAL-TR-2004-034.

Gould, N. I. M., & Toint, Ph. L. (2004). How mature is nonlinear optimization. *Applied Mathematics Entering the 21st Century: Invited Talks from the ICIAM 2003 Congress* (pp. 141-161). Philadelphia: SIAM.

Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *Transactions On Systems, Man, And Cybernetics* (Vol. 16(1), pp. 122-128). New-York: Institute of Electrical and Electronics Engineers.

Guillot, M. (2003). *Introduction à l'intelligence artificielle*. Québec: Département de génie mécanique, Université Laval.

Guo, Z., & Uhrig, R.E (1992). Using genetic algorithms to select inputs for neural networks. *Proceedings International Workshop Combinations of Genetic algorithms and Neural Networks (COGANN-92)* (pp. 223-234). Los Alamitos: Whitley and J. D. Schaffer Eds.

Hadj-Alouane, A.B., & Bean, J.C. (1997). A Genetic algorithm for the multiple-choice integer program. *Operations Research*, *45*, 92-101.

Hagan, M., Demuth, H., & Beale, M. (2002). *Neural Network Desing (Electrical engineering)*. Colorado: University of Colorado bookstore.

Hager, W. W., Hearn, W.D., & Pardalos, P.M. (1994). *Large Scale Optimization State of the Art*. Dordrecht: Kluwer Academic Publishers.

Higuchi, T., Tsutsui, S., & Yamamura, M. (2000). Theoretical analysis of simplex crossover for real-coded genetic algorithms. *Parallel Problem Solving from Nature (PPSN-VI)* (pp. 365–374). London: Springer-Verlag.

Hinterding, R. (1995). Gaussian mutation and self-adaption for numeric genetic algorithms. *IEEE International Conference on Evolutionary Computation* (Vol. 1(29), pp. 384-389). New-York: IEEE Press.

Hock, W., & Schittkowski, K. (1981). *Test Examples for Nonlinear Programming Codes.* Berlin: Springer.

Hong, Y.-S., Lee, H., & Tahk, M.-J. (2003). Acceleration of the convergence speed of evolutionary algorithms using multi-layers networks. *Engineering optimization*, *35*(1), 91-102.

Hooke, R., & Jeeves, T.A. (1961). Direct search solutions of numerical and statistical problems. *Journal of the association for computing machinery*, *8*(2), 212-229.

Hornik, K., Stinchcombe, M, & White, H. (1989). Multi-layer feedforward networks are universal aproximators. *Neural Networks*, *2*, 359-366.

Hornyak, J., & Monostori, L. (1993). Feature extraction technique for ANN-based financial forecasting. *Neural Network World*, *7*(4-5), 543-552.

Hunt, J. (1995). Evolutionary Case Based Design. In Waston, I.D. (ed.), *Progress in Case-Based Reasoning LNAI 1020* (pp. 17–31). Berlin: Springer.

Hung, S.L., & Adeli, H. (1994). Parallel genetic/neural network learning algorithm for MIMD shared memory machines. *IEEE Transactions Neural Networks* (Vol. 5, pp. 900-909). New York: Institute of Electrical and Electronics Engineers.

Husken, M., Jin, Y., & Sendhoff, B. (2003). Structure optimization of neural networks for aerodynamic optimization. *Soft Computing Journal*, *9*(1), 21-28.

Hwang, J.-N., You, S.-S., Lay, S.-R., & Jou, I.-C. (1996). What's wrong with a cascaded correlation learning network: A projection pursuit learning perspective. *Transactions on neural networks*, *7*, 278-289.

Jin, Y., Husken, M., Olhofer, M., & Senhoff, B. (2004). Neural networks for fitness aproximation in evolutionary optimization. *Knowledge Incorporation in Evolutionary Computation* (pp. 281-306). Berlin: Springer.

Jin, Y., Olhofer, M., & Senhoff, B. (2002). A framework for evolutionary optimization with approximate fitness functions. *IEEE Transactions on Evolutionary Computation* (Vol. 6(5), pp. 481-494). Portland: Portland State University.

Johansson, E.M., Dowla, F.U., & Goodman, D.M. (1994). Backpropagation learning for multilayer feed-forward neural networks using the conjugate gradient method. *International Journal Neural System*, *2*(4), 291-301.

Johnson, J., & Sushil, J.L. (2005). Case-Initialized Genetic Algorithms for Knowledge Extraction and Incorporation. *Knowledge Incorporation in Evolutionary Computation* (pp. 57-79). Berlin: Springer.

Kim, I.Y., & Weck, O.L. (2005). Variable chromosome length genetic algorithm for progressive refinement in topology optimization. *Structural and Multidisciplinary Optimisation*, *29*, 445-456.

Kintano, H. (1990). Empirical studies on the speed of convergence of neural network training using genetic algorithms. *Proceedings 8th National Conference Artificial intelligence (AAAI-90)* (pp. 789-795). Cambridge: MIT Press.

Kolonder, J.L. (1993). *Case-Based Reasoning*. San Mateo: Morgan Kaufmann.

Lagaros, N.D., & Papadrakakis, M. (2004). Learning improvment of neural networks used in structural optimization. *Advances in engineering software*, *35*(1), 09-25.

Lansdown, J. (1987). The creative aspects of CAD: a possible approach. *Design studies*, 8 (2), 76–81.

Leake, D. (2000). *Case-Based Reasoning: Experiences, Lessons, and Future Directions*. San Jose: AAAI.

Lee, K.Y., El-Sharkawi, Storn, R, & Prince, K. (2008). *Modern heuristic optimization techniques: theory and applications to power systems*. Hoboken: Wiley & Sons inc.-IEEE.

Leone, R.D., Murli, A., Pardalos, P.M., & Toraldo, G. (1998). *High Performance Algorithms and Software in Nonlinear Optimization*. Dordrecht: Kluwer Academic Publishers.

Lewis, A.S., & Overton, M.L. (1996). Eigenvalue optimization. *Acta Numerica*, *5*, 149-190.

Li, X. & Kraslawski, A. (2004). Conceptual Process Synthesis: Past and Current Trends. *Chemical Engineering and Processing*, 43 (5), pp. 589–600.

Lippman, R.P. (1987). An introduction to computing with neural nets. *IEEE ASSP Magazine*, 04-22.

Liu, B., Ma, H., Zhang, X., Liu, B., & Zhou, Y. (2007). A memetic Co-evolutionary Differential Evolution Algorithm for Constrained Optimization. *IEEE Congress on Evolutionary computation* (pp. 2996-3002). Singapour.

Liu, X. (1996). *Combining Genetic Algorithm with Case-based Reasoning for Structural Design* M.S, University of Nevada, Reno, États-Unis.

Maher, L., Balachandran, M., & Zhang, D.-M. (1995). *Case-based reasoning in design*. New Jersey: Erlbaum.

Marazzi, M., & Nocedal, J. (2001). Feasibility directions algorithms for optimization problems with equality and inequality constraints. *Mathematical Programming*, *11*(1), 67-80.

Menczer, F., & Parisi, D. (1992). Evidence of hyperplanes in the genetic learning of neural networks. *Biological Cybernetic*, *66*, 283-289.

Mendes, M., Mosley, N., & Watson, I. (2002). A comparison of case-based reasoning approaches. *Proceedings of the 11th international conference on World Wide Web* (pp. 272-280). New York: ACM.

Michalewicz, Z., Deb, K., Schmidt, M., & Stidsen, T. (2000). Test-case Generator for Nonlinear Continuous Parameter Optimization Techniques. *IEEE Transactions on Evolutionary Computation*, *4*(3), 197-215.

Miller, G.F., Todd, P.M., & Hegde, S.U. (1989). Designing neural networks using genetic algorithms. *Proceedings 3rd International Conference Genetic Algorithms and Their Applications* (pp. 379-384). San Mateo: J.D. Shaffer Ed.

More, J.J., & Toraldo, G. (1991). On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, *5*(3), 590-640.

Morrison, R.W., & De Jong, K.A. (1999). A Test Problem Generator for Non-Stationary Environments. *Proceedings of the 1999 Congress on Evolutionary Computation* (pp. 2047-2053). Washington.

Murray, W., & Wright, M.H. (1992). A sequential quadratic programming algorithm using a incomplete solution of the subproblem. *SIAM Journal on Optimization*, *5*(3), 590-640.

Murty, K.G., & Kabadi, S.N. (1987). Some NP-complete problems in quadratic and nonlinear programming. *Mathematical Programming*, *39*(2), 117-129.

Nakayama, T. & Tanaka, K. (1999). Computer-assisted thermal analysis system founded on case-based reasoning. *Journal of Chemical Information and Computer Science*, 39, pp. 819–832.

Nash, S.G. (2000). A survey of the truncated-Newton method for large scale optimization. *Journal of Computational and Applied Mathematics*, *124*, 45-59.

Nocedal, J. (1992). Theory of algorithms for unconstrained optimization. *Acta Numerica*, *1*, 199-242.

Nocedal, J., & Wright, S.J. (1999). *Large sparse numerical optimization*. Berlin: Springer.

Noman N., & Iba, H. (2008). Accelerating Differential Evolution Using an Adaptive Local Search. *IEEE Transactions on Evolutionary Computation*, *12*(1), 107-125.

Ong, Y.S., Nair, P.B., Keane, A.J., & Wong, K.W. (2005). Surrogate-assisted evolutionary optimization frameworks for high-fidelity engineering design problems. *Knowledge Incorporation in Evolutionary computation* (pp. 57-78). Berlin: Springer.

Ong, Y.S., Nair, P.B., Shi, D.M., & Zhang, Z.K. (2003). Global convergence unconstrained and bound constrained surrogate-assisted evolutionary search in aerodynamic shape design. *Congress on Evolutionary Computation, Special Session on Design Optimization with Evolutionary Computation* (Vol. 3, pp. 1856-1863). Canberra: IEEE Press.

Ono, I., & Kobayashi, S. (1997). A real-coded genetic algorithm for function optimization using unimodal normal distribution crossover. *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA-7)* (pp. 246–253). San Francisco: Morgan Kaufmann.

Pahl, G. & Beitz,W. (1984). *Engineering Design*. London: Design Council Books.

Papadrakis, M., Lagaros, N.D., Tsompanakis, Y., & Plevris, V. (2001). Large Scale Structural Optimization: Computational Methods and Optimization Algorithms. *Archives of Computational Methods in Engineering*, *8*(3), 239-301.

Powell, M.J.D. (1998). Direct search algorithms for optimization calculations. *Acta Numerica*, *7*, 287-336.

Prados, D.L. (1992). Training multilayered neural networks by replacing the least fit hidden neurons. *Proceedings IEEE SOUTHEAST-CONF'92* (Vol. 2, pp. 634-637). New York: IEEE Press.

Ramsey, C., & Grefensttete, J. (1993). Case-based initialization of genetic algorithms. *Proceeding of the Fifth International Conference on Gnetic Algorithms*, 84-91.

Rasheed, K. (2000). An incremental-approximate-clustering approach for developing dynamic reduced models for design optimization . *Proceeding of the Congress on Evolutionary Computation (CEC)* (pp. 986-993). California.

Rasheed, K., Ni, X., & Vattam, S. (2004). Methods for using surrogate models to speed up genetic algorithm optimization: imformed operators and genetic engineering. *Knowledge Incorporation in Evolutionary Computation* (pp. 103-122). Berlin: Springer.

Rasheed, K., Ni, X., & Vattam, S. (2005). Comparison of methods for developing dynamic reduced models for design optimization. *Soft computing journal*, *9*(1), 29-37.

Rasmey, C.L., & Grenfenstette, J. (1993). Case-based initialization of genetic algorithms. *Proceedings of the 5th Conference on Genetic Algorithms* (pp. 89-91). San Mateo: Morgan Kaufmann.

Ratle, A. (1998). Accelerating the convergence of evolutionary algorithms by fitness landscape approximation. *Parallel Problem Solving from Nature V* (pp. 87-96). Berlin: Springer.

Ratle, A. (1998). Accelerating the convergence of evolutionary algorithms by fitness landscape approximation. *Parallel Problem Solving from Nature V* (pp. 87-96). Berlin: Springer.

Ratle, A. (2001). Kriging as surrogate fitness landscape in evolutionary optimization. *Artificial Intelligence for Engineering Design Analysis and Manufacturing*, *15*(1), 37-49.

Reed, J. (1967). Simulation of biological evolution and machine learning. *Journal of Theoretical Biology*, *17*, 319-342.

Renders, J.-M., & Flasse, S.P. (1996). Hybrid methods using genetic algorithms for global optimization. *Transactions On Systems, Man, And Cybernetics* (Vol. 26(2), pp. 243-258). Bruxelles: Université Libre de Bruxelles.

Robinson, S.M. (1974). Perturbed Kuhn-Tucker points and rates of convergence for a class of nonlinear programming algorithms. *Mathematical Programming*, *7*(1), 1-16.

Rosenbrock, H.H. (1960). An automatic method for finding the greatest or least value of a function. *Computer Journal*, *3*, 175-184.

Roy, A., Kim, L.S., & Mkhopadhyay, S. (1993). A polynomial time algorithm for the construction and training of a class of multilayer perceptron. *Neural Networks*, *6*(4), 535-545.

Ryoo, J., & Hajela, P. (2004). Handling variable string lengths in a based structural topology optimization. *Structural and Multidisciplinary Optimisation*, *26*, 318-325.

Schank, R. (1982). *Dynamic Memory: A Theory of Learning in Computers and People*. Cambridge: Cambridge University Press.

Serafini, D.B. (1998). *A framework for managing models in nonlinear optimization of computationnaly expensive functions.* Thèse de doctorat, Rice University, Texas, Houston, États-Unis.

Sexton, R.S, Dorsey, R.E, & Johnson, D.S. (1998). Toward global optimization of neural networks: A comparison of the genetic algorithm and backpropagation. *Decision Support System*, *22*(2), 171-185.

Siddall, J.N. (1982). *Optimal Engineering Design: Principles and Applications*. New York: Dekker.

Sirnivas, M., & Patnaik, L. (1994). Adaptives probabilities of mutation and crossover in genetic algorithms. *Transactions On Systems, Man, And Cybernetics* (Vol. 24(4), pp. 656-667). New York: Institute of Electrical and Electronics Engineers.

Song, W., & Keane, A.J. (2005). An efficient evolutionary optimisation framework applied to turbine blade firtree root local profiles [version électronique]. *Structural and Multidisciplinary Optimization*, *29*(5), 382-390.

Spedicato, E. (1994). *Algorithms for Continuous Optimization: The State of the Art*. Dordrecht: Kluwer Academic Publishers.

Srinivas, M., & Patnaik, L. (1994). Adaptive probabilities of crossover and mutation in genetic algorithms. *Transactions On Systems, Man, And Cybernetics* (Vol. 24(4), pp. 656-667). New York: Institute of Electrical and Electronics Engineers.

Stelmack, M.A., Batill, S.M., & Beck, B.C. (2000). Design of an aircraft brake component using an interactive multidisciplinary design optimization framework. *ASME Journal of Mechanical Design*, *122*(1), 70-76.

Storn, R, & Prince, K. (1995). *Differential Evolution - a Simple and Efficient Adaptive Scheme for Global Optimization over Continuous Spaces*. Berkley: Technical Report TR-95-012, International Computer Science Institute.

Stuckman, B.E. (1988). A Global Search Method for Optimizing Nonlinear Systems. *Transactions on Systems, Man, and Cybernetics*, *18*(6), 965-977.

Sutton, R.S. (1986). Two problems with backpropagation and other steepest-descent learning procedures. *Proceedings 8th Annual Conference Congnitive Science Society* (pp. 823-831). New Jersey: Erlbaum.

Todd, M.J. (2001). Semidefinite optimization. *Acta Numerica*, *10*, 515-560.

Tong, C. & Sriram, D. (eds.) (1992). *Artificial Intelligence in Engineering Design*. Boston: Academic Press.

Tsutsui, S. (2008). Sampling Bias and Search Space Boundary Extension in Real Coded Genetic Algorithms. *IEEE Transactions on Evolutionary Computation* (Vol. 12(1), pp. 107). Las Vegas: Morgan Kaufmann.

Ulrich, K.T. & Eppinger, S.D. (2000). *Product Design and Development*. New York: McGraw-Hill.

Ursem, R. K, & Vadstrup, P (2003). Parameter Identification of Induction Motors Using differential Evolution. *Proceedings of the Fifth Congress on Evolutionary Computation (CEC-2003)* (pp. 790-796). Piscataway: IEEE Press.

Vanderplaats, G.N. (1999). Structural Design Optimization Status and Direction. *Journal of Aircraft*, *36*(1), 11-20.

Vapnik, V (1998). *Statistical Learning Theory*. New-York: John Wiley and Sons.

Whitley, D., Starkweather, T., & Bogart, C. (1990). Genetic algorithms and neural networks: Optimizing connections and connectivity . *Parallel computing*, *14*(3), 347-361.

Widrow, B., & Hoff, M.E. (1960). Adaptative switching circuits. *IRE WESCON Convention Record* (pp. 96-104). San Francisco: IRE.

Willmes, L., Back, T, Jin, Y., & Senhoff, B. (2003). Comparing neural network and kriging for fitness approximation in evolutionary optimization. *(pp. 663-670). Canberra: IEEE Press.*

Wolpert, D.H., & Macready, W.G. (1995). *No Free Lunch Theorems for Search.* : Technical Report SFI-TR-95-02-010, Santa Fe Institute.

Won, K.S., Tapabrata, R., & Tai, K. (2003). A framework for optimization using approximate functions. *Proceedings of the 2003 Congress on Evolutionary Computation CEC2003* (pp. 1520-1527). Canberra: IEEE Press.

Wretter, M., & Wright, J. (2003). Comparison of a generalized pattern search algorithm and a genetic algorithm optimization method. *Proceedings of Eight International IBPSA Conference* (pp. 1401-1408). Eindhoven: Building Simulation.

Wright, M.H. (1992). Interior methods for constrained optimization. *Acta Numerica*, *1*, 341-407.

Wu, A.S., & Garibay, I. (2002). The proportional genetic algorithm: Gene expression in a genetic algorithm. *Genetic Programming and Evolvable Hardware*, *3*, 157-192.

Xu, S., & Grandhi, R.V. (2000). Multipoint approximation development: thermal structural optimization case study. *International Journal for Numerical Methods in Engineering*, *48*, 1151-1164.

Yain-Whar, S., & Babka, O. (1998). Neural Network Supported Adaptation in Case-based Reasoning. *Proceeding of the International ICSC/IFAC Symposium on Neural Computation*, *NC*(98), 931-936.

Yan, W., Zhu, Z., & Hu, R. (1997). Hybrid genetic/BP algorithm and it's application radar target classification. *Proceedings 1997 IEEE National Aerospace and*

Electronics Conference NAECON (Vol. 2, pp. 981-984). New Jersey: IEEE Press.

Yang, J., & Honavar, V. (1998). Feature subset selection using a genetic algorithm. *IEEE Intelligent Systems and Their Applications*, *13*(2), 44-49.

Yang, J-M, Kao, C-Y, & Horng, J-T (1996). Envolving neural induction regular language using combined evolutionary algorithms. *Proceedings 1996 1st Joint Conference Intelligent Systems* (Vol. 1, pp. 162-169). New Jersey: IEEE-TAB Products Group.

Yao, X. (1999). Evolving artificial neural network. *Proceedings of the IEEE*, *87*, 1427-1447.

Yao, X. (2005). A selected introduction to evolutionary computation. *Knowledge Incorporation in Evolutionary computation* (pp. 3-14). Berlin: Springer.

Yeniay, Ö. (2005). Penalty Fuction Methods For Constrained Optimization with Genetic Algorithms. *Mathematical and Computational Applications*, *10*(1), 45-56.

Yuan, B., & Gallagher, M. (2003). On Building a Principled Framework for Evaluating and Testing Evolutionary Algorithms: A Continuous Landscape Generator. *Proceedings of the 2003 Congress on Evolutionary Computation* (pp. 451-458). Australia: The Institute of Electrical and Electronics Engineers.

Yuan, B., & Gallagher, M. (2004). Statistical Racing Techniques for Improved Empirical Evaluation of Evolutionary Algorithms. *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)* (pp. 172-181). Berlin: Springer-Verlag.

Yuan, Y (1998). *Advances in Nonlinear Programming*. Dordrecht: Kluwer Academic Publishers.

Zhang, J., Chung, S.H., Lo, A.W.L., & Hu, B.J. (2005). Fuzzy knowledge incorporation to crossover and mutation. *Knowledge Incorporation in Evolutionary computation* (pp. 123-143). Berlin: Springer.

Zhang, Y., Louvieris, P., & Petrou, M. (2007). Case-Based Reasoning Adaptation for High Dimensional Solution Space. *International Conference on Case-based Reasoning* (pp. 149-163). Berlin: Springer.

Annexe A.
RÉSEAUX DE NEURONES

Pendant que vous lisez ceci, environ 10^{11} neurones interconnectés dans votre cerveau s'activent pour donner un sens à cette phrase, chacun de ces neurones ayant la complexité et pratiquement la vitesse d'un microprocesseur.

Les réseaux de neurones artificiels (RNA) dont nous discuterons ici (plus connus sous l'appellation ANN pour artificial neural network) sont en quelque sorte des modèles mathématiques inspirés de la biologie humaine. Disposant d'une topologie extrêmement simplifiée par rapport à leurs homologues naturels, les RNA n'en sont pas moins capables de résoudre certaines applications de façon remarquable s'ils sont entraînés en conséquence.

En effet, la littérature semble démontrer que leur utilisation pour remplacer un modèle mathématique de grande envergure peut dans plusieurs cas réduire considérablement le temps d'optimisation d'un système (Jin, Husken, Olhofer & Sendhoff, 2004; Rasheed, Ni & Vattam 2004 ; Song & Keane, 2005).

Bien que l'histoire de la science entourant les réseaux neuronaux artificiels dépasse le cadre de ce document, le lecteur intéressé pourra se référer à Hagan, Demuth & Beale (2002) pour plus de détails.

Avant de s'avancer plus loin, il serait utile de définir brièvement ce qu'est un RNA ainsi que les divers types de RNA existants.

A.1 Un neurone

La figure A.1 représente un neurone d'un réseau neuronique artificiel :

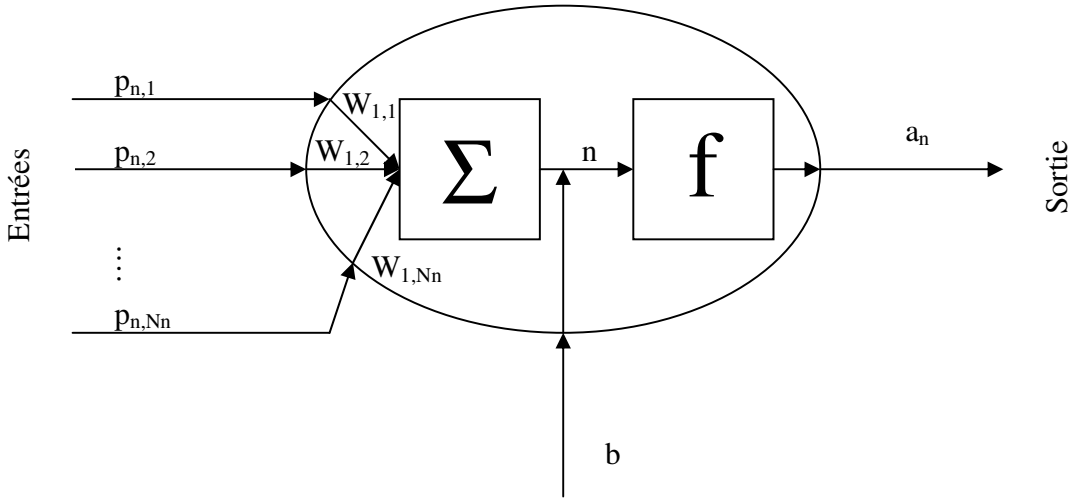


Figure A.1 : Représentation d'un neurone artificiel

Ainsi, un neurone additionne plusieurs entrées, chacune étant multipliée par un certain poids $W_{1,i}$. Ensuite, on ajoute un biais b, puis on passe le résultat dans une certaine fonction de transfert f pour obtenir un scalaire à la sortie.

Mathématiquement, on a :

$$a_n = f\left(\sum_{i=1}^{N_n} p_{n,i} W_{1,i} + b\right) \quad \text{pour } i \in \{1,2...N_n\}$$

où N_n est le nombre d'entrées du neurone.

$p_{n,i}$ est la valeur scalaire de l'entrée i

$W_{1,i}$ est le poids accordé à l'entrée i pour la neurone 1.

b est le biais présent sur le neurone

Ou encore, sous forme vectorielle :

$$a_n = f\left(\vec{W}\vec{p}_n + b\right)$$

où $\vec{W}$ est un vecteur rangée contenant les poids du neurone

$\vec{p}_n$ est le vecteur colonne d'entrée des neurones

b est le biais du neurone

Enfin, si l'on dispose de plusieurs neurones en parallèle ayant chacune le même nombre d'entrées N, on peut définir la relation matricielle suivante :

$$\vec{a}_n = f\ (\mathbf{W}\ \vec{p}_n + \vec{b}\ )$$

où $\mathbf{W}$ est une matrice contenant les vecteurs rangée des poids des neurones

$\vec{b}$ est un vecteur contenant le biais des neurones

En général, lorsqu'un modèle neuronal est mis en place, la fonction de transfert f est fixée, et les paramètres W et $\vec{b}$ sont ajustés par un algorithme de façon à obtenir la sortie voulue an selon les N_n entrées $\vec{p}_n$ du modèle.

La table A.1 illustre les principales fonctions de transfert qui sont utilisées dans les réseaux neuronaux (Hagan, Demuth & Beale, 2002):

Table A.1. Les fonctions de transferts les plus communes pour un neurone

Nom	Relation entrée/sortie
Bornée	$a_n = 0 \quad n < 0$ $a_n = 1 \quad n \geq 0$
Symétrique bornée	$a_n = -1 \quad n < 0$ $a_n = 1 \quad n \geq 0$
Linéaire	$a_n = n$
Linéaire saturée	$a_n = 0 \quad n < 0$ $a_n = n \quad 1 \geq n \geq 0$ $a_n = 1 \quad n > 1$
Linéaire saturée symétrique	$a_n = -1 \quad n < -1$ $a_n = n \quad 1 \geq n \geq -1$ $a_n = 1 \quad n > 1$
Log-sigmoïde	$a_n = \frac{1}{1+e^n}$
Tangente hyperbolique sigmoïde	$a_n = \frac{e^n - e^{-n}}{e^n + e^{-n}}$
Linéaire positive	$a_n = 0 \quad n < 0$ $a_n = n \quad n \geq 0$
Compétitive	$a_n = 1$ pour la neurone du réseau ayant le plus grand n $a_n = 0$ pour les autres

A.2 Les principaux réseaux de neurones

La figure A.2 présente globalement les différentes catégories de réseaux de neurones que l'on peut retrouver dans la littérature (Guillot 2003).

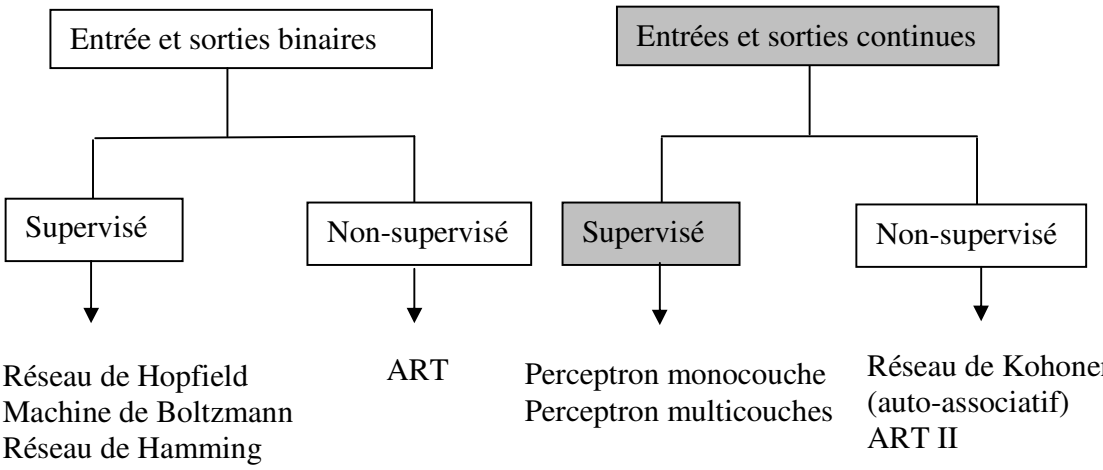


Figure A.2 : Types de réseaux de neurones

On classe dans la catégorie supervisée les réseaux ayant besoin de plusieurs schémas de sorties désirés en fonction des entrées pour pouvoir fournir une réponse adéquate. Les réseaux non-supervisées, quant à eux, n'ont pas besoin de boucle de retour et sont capables d'apprendre à générer eux-mêmes différents schémas de sortie à mesure que les entrées arrivent (par exemple, dans Hagan, Demuth & Beale (2002), un réseau non-supervisé est utilisé pour séparer en trois ensembles les plus différents possibles des objets qui arrivent sans connaître d'avance les paramètres des objets).

La plupart des réseaux de neurones présentés à la figure A.2 sont des mémoires associatives, c'est-à-dire des réseaux dont les neurones peuvent être employés comme entrées ou sorties du réseau.

En général, les réseaux de neurones à mémoire associative sont très utiles lorsqu'il s'agit de reconnaître un certain schéma d'entrée, comme par exemple dans les applications de reconnaissance d'écriture (Hagan, Demuth & Beale, 2002), de reconnaissance de maladie selon les symptômes (Yang & Honavar, 1999) ou tout simplement de classifier certains objets en plusieurs types distincts (comme séparer des fruits de plusieurs formats sur une chaîne de montage).

Toutefois, dans le cadre d'un projet d'optimisation, le réseau de neurones le plus susceptible de nous intéresser est le réseau à entrées/sorties continues avec supervision de type perceptron, qui est davantage un outil de modélisation qu'une mémoire associative.

En effet, dans les problèmes d'optimisation de grande envergure, chaque information requise sur la fonction coût ou sur ses dérivées peut s'avérer très coûteuse, et il peut donc être parfois très avantageux de remplacer le modèle réel par un modèle approximatif.

A.3 Réseaux continus supervisés : Le perceptron

Le perceptron monocouche (disposant d'un seul neurone par sortie) à été proposé comme outil de modélisation par Rosenblatt (voir Lippman, 1987) vers la fin des années 50.

Il s'agit uniquement d'un nombre de neurones égal au nombre de sorties désirées œuvrant en parallèle, chaque neurone étant connecté à chaque entrée. Vers la fin des années soixante, il fut toutefois vertement critiqué par Papert et Minsky en raison du fait qu'il n'était capable de modéliser que les systèmes dont les paramètres de sortie étaient linéairement séparables (Dagli, 1994).

Pour pallier à ce problème, le réseau perceptron multicouches a été proposé (Hagan, Demuth & Beale, 2002), consistant en un nombre $M_{n,i}$ de neurones interconnectés sur plusieurs couches *i* tel qu'illustré à la figure A.3.

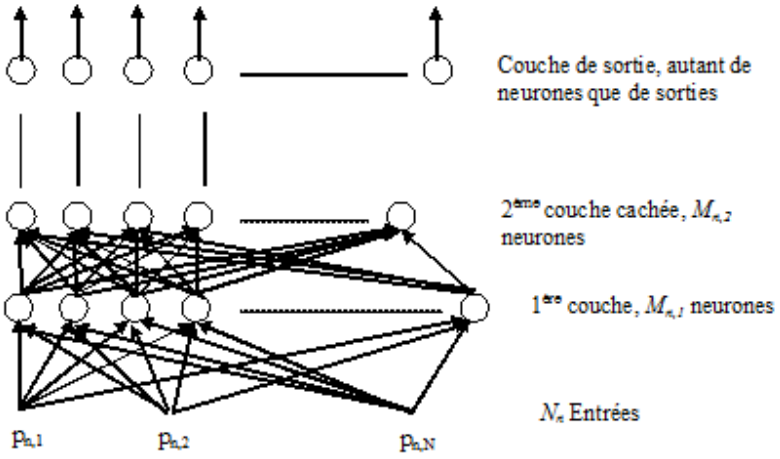


Figure A.3 : Réseau de neurones de type perceptron

En théorie, on peut donc introduire autant de couches cachées que désiré et poser autant de neurones que voulu dans chacune des couches. Toutefois, Hornik, Stinchcombe & White (1989) ont démontré qu'un perceptron avec une seule couche cachée contenant une infinité de neurones constitue en théorie un approximateur universel capable de représenter de façon exacte n'importe quelle fonction, peu importe la continuité et la complexité de celle-ci.

Ainsi, on utilise généralement un perceptron avec une seule couche cachée dans les cas pratiques. L'une des fonctions de transfert les plus fréquemment utilisées pour les neurones des perceptrons est la fonction log-sigmoïde présentée dans la table 1. C'est d'ailleurs un perceptron de ce type (une seule couche caché et fonction de transfert log-sigmoïde) qui sera utilisé dans le reste de ce document.

A.3.1 Entraînement du perceptron

Évidemment, bien qu'un perceptron ait la capacité d'approximer avec succès n'importe quelle fonction, il n'est pas toujours aisé de déterminer les poids **W** et les biais $\vec{b}$ permettant une approximation la plus exacte possible pour le nombre de neurones du perceptron.

De plus, puisque le coût en temps de calcul de l'entraînement du perceptron et de son utilisation dépend directement du nombre de neurones sur la couche cachée, il faut aussi avoir une méthode efficace pour choisir ce nombre de neurones. En effet, un nombre trop petit ne laissera pas assez de degrés de liberté au réseau pour pouvoir modéliser toutes les variations dans la fonction coût, alors qu'un nombre trop grand représentera du temps perdu lors de chacun des calculs subséquents ainsi qu'une possibilité de très mal interpoler entre les points connus comme dans le cas des régressions polynomiales d'ordre très élevé (voir figure A.4).

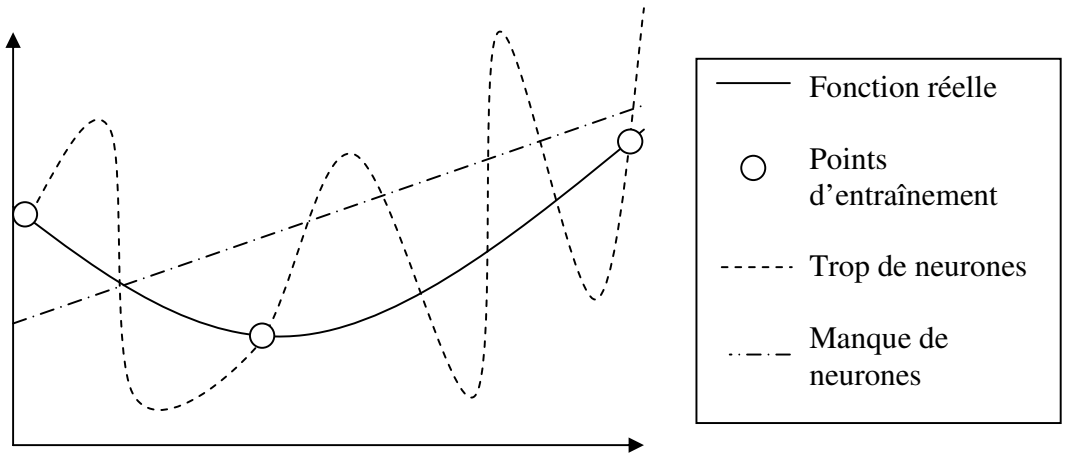


Figure A.4 : Difficultés résultantes d'un mauvais choix du nombre de neurones

Cette section traite des méthodes d'entraînement les plus utilisées pour ajuster les poids et les biais des réseaux de neurones de type perceptron.

A.3.2 Entraînement à propagation inverse (ou BP pour Back Propagation)

Jusqu'à maintenant, plusieurs des utilisations réelles de perceptron que l'on retrouve dans la littérature utilisent l'algorithme BP ou des variantes pour entraîner un réseau dont l'utilisateur a choisi arbitrairement le nombre de neurones (Yao, 1999).

Comme la plupart des méthodes d'entraînement, l'algorithme BP résout le problème d'optimisation suivant :

$$\text{Minimiser} \quad F(\vec{x})$$

où $F(\vec{x})$ est un indicateur de performance du réseau

$\vec{x}$ est un vecteur contenant les valeurs des poids **W** et des biais $\vec{b}$ du réseau.

Les entrées requises pour l'entraînement de BP sont une matrice $\mathbf{P_n}$ contenant les vecteurs colonnes des entrées du réseau $\vec{p}_n$ et une matrice $\mathbf{T_n}$ dont chaque colonne est un vecteur de sorties requises par le réseau $\vec{t}_n$ pour chacun des vecteurs d'entrée $\vec{p}_n$.

En général, l'indicateur de performance du réseau qui est utilisé est une approximation de l'erreur quadratique du réseau pour les données d'entraînement dont on dispose.

$$F(\vec{x}) = E\left(\vec{e}^T\vec{e}\right) = E\left((\vec{t}_n - \vec{a}_n)^T(\vec{t}_n - \vec{a}_n)\right) \qquad \text{(A.1)}$$

où $E(\vec{e}^T\vec{e})$ représente l'espérance de $\vec{e}^T\vec{e}$, et $\vec{e}$ représente l'erreur que le réseau donne

pour chacune des sorties pour lesquels il est entraîné.

On se rappelle que le vecteur $\vec{a}_n$ représente les sorties réelles du réseau et $\vec{t}_n$ les sorties exigées par l'entraînement.

Widrow et Hoff (1960) ont démontré que cet index de performance pour un réseau pouvait être approximé efficacement par :

$$F(\vec{x}) \approx (\vec{t}_n - \vec{a}_n)^T (\vec{t}_n - \vec{a}_n) \qquad \text{(A.2)}$$

Comme nous l'avons vu, dans un perceptron multicouche, on peut calculer la sortie $\vec{a}_{n,m}$ d'une certaine couche $M_{n,i}$ de la façon suivante :

$$\vec{a}_{n,0} = \vec{p}_n$$

$$\vec{a}_{n,m+1} = f_{m+1}\left(W_{m+1}\vec{a}_{n,m} + \vec{b}_{n,m+1}\right) \qquad \text{pour m= 0,1,…, } M_{n,N}$$

$$\vec{a}_n = \vec{a}_{M_{n,N}}$$

où $M_{n,N}$ est le nombre de couches totales du réseau

$\vec{a}_{n,m}, f_m, W_m, \vec{b}_{n,m}$ sont respectivement la sortie, la fonction de transfert, la matrice de poids et le vecteur de biais affectés à la couche m.

Une fois la sortie calculée, il est ensuite possible de rétropropager les sensibilités par rapport à chaque neurone de l'indicateur de performance de la dernière couche vers la première, d'où le nom propagation inverse (ou Back Propagation) de l'algorithme :

$$\vec{s}_M = -2\dot{F}_M(\vec{n}_M)(\vec{t}_n - \vec{a}_n)$$

(A.3)

$$\vec{s}_m = \dot{F}_m(\vec{n}_m)\left(W_{m+1}\right)^T s_{m+1} \qquad \text{pour } m = M_{n,N}\text{-1, } M_{n,N}\text{ -2, ….1}$$

où $\vec{s}_m$ est le vecteur contenant la sensibilité de $\hat{F}(X)$ par rapport à chaque neurone

de la couche m.

$\vec{n}_m = W_m \vec{p}_m + \vec{b}_m$ est le vecteur contenant la sortie de la couche *m* de neurones avant la fonction de transfert.

$$\dot{F}_m(\vec{n}_m) = \begin{bmatrix} \dot{f}_m(n_{m,1}) & 0 & \cdots & 0 \\ 0 & \dot{f}_m(n_{m,2}) & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & \dot{f}_m(n_{m,M_m}) \end{bmatrix}$$

$$\dot{f}_m(n_{m,j}) = \frac{\partial f_m(n_{m,j})}{\partial n_{m,j}}$$

$n_{m,j}$ est la j$^{\text{ième}}$ composante du vecteur $\vec{n}_m$

Après avoir complété ces calculs, nous possédons maintenant la sensibilité relative de l'indicateur de performance du réseau selon chacune des neurones de ce réseau.

L'algorithme BP propose donc simplement de faire un pas selon l'algorithme de la plus forte pente en avançant en direction inverse du gradient approximatif. On peut définir une vitesse d'apprentissage du réseau α_n, qui correspond en fait à une indication de la longueur de pas que l'on désire que l'algorithme effectue.

En général, α_n est prit entre 0 et 1 (souvent aux environs de 0.1), et on met les poids et les biais à jour à l'itération *k* selon les formules suivantes :

$$W_{m,k+1} = W_{m,k} - \alpha_n \vec{s}(\vec{a}_{m-1})^T$$

(A.4)

$$b_{m,k+1} = b_{m,k} - \alpha_n \vec{s}$$

On constate donc que le perceptron entraîné avec BP convergera éventuellement vers un minimum de l'indicateur de performance sous les mêmes conditions que l'algorithme de la plus forte pente.

Si l'on effectue plusieurs passes d'apprentissage (plusieurs itérations de k) avec les différentes données d'entraînement, on pourra alors construire un modèle représentant adéquatement la fonction d'où proviennent les points employés pour l'entraînement.

Ceci, à condition que l'algorithme n'oscille pas à cause d'un pas trop grand, ou qu'il ne converge pas vers un minimum local. Pour un réseau disposant de plusieurs neurones, il devient donc extrêmement important de bien spécifier les poids et les biais initiaux du système.

Variations sur BP

Plusieurs chercheurs proposent des améliorations heuristiques permettant d'améliorer les propriétés de convergence de BP. Bien que cela dépasse le cadre de ce document, un lecteur intéressé pourra se référer à Hagan, Demuth & Beale (2002) pour plus de détails.

A.3.3 Algorithme de Levenberg-Marquardt

L'algorithme de Levenberg-Marquadt (aussi connu sous le nom de LMBP pour Levenberg-Marquadt Back Propagation) est une variation de la méthode de Newton qui a été conçue pour minimiser des fonctions qui sont des sommes du carré d'autres fonctions non-linéaires.

Ceci est particulièrement bien adapté pour un réseau de neurones utilisant l'erreur quadratique moyenne comme indicateur de performance.

Cet algorithme semble très populaire actuellement vu sa vitesse de convergence quadratique et sa robustesse. Grossièrement, l'algorithme tente d'utiliser une direction de Gauss-Newton (quadratique) pour modifier ses paramètres. S'il constate qu'il ne parvient pas à réduire l'indicateur de performance, il change graduellement la direction du pas jusqu'à atteindre la direction de la plus forte pente. Ceci lui procure une convergence presque quadratique si la fonction s'y porte, et lui assure une convergence au moins linéaire si la topologie de la fonction ne permet pas la convergence quadratique.

Soit un échantillonnage de Q données d'entraînement, on définit les vecteurs suivants :

$$\vec{v} = \begin{bmatrix} v_1 & v_2 & \cdots & v_N \end{bmatrix} = \begin{bmatrix} e_{1,1} & e_{2,1} & \cdots & e_{M_1,1} & e_{1,2} & \cdots & e_{M_M,Q} \end{bmatrix}$$

$$\vec{x}^T = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} = \begin{bmatrix} W_{1,1} & W_{2,1} & \cdots & W_{M_1,1} & b_{1,1} & b_{2,1} & \cdots & b_{M_1,1} & W_{1,2} & \cdots & b_{M_M,M} \end{bmatrix}$$

où $\vec{v}$ est le vecteur contenant toutes les erreurs pour tous les patrons d'apprentissage

$\vec{x}$ est un vecteur colonne contenant tous les paramètres modifiables du réseau de

neurone

$W_{i,j}$ est la rangée i de la matrice de poids W pour la couche de neurone j

$b_{k,l}$ est le biais du $k^{\text{ième}}$ neurone de la couche l.

En utilisant pratiquement la même procédure que dans BP pour calculer les sensibilités, on peut alors calculer chaque élément de la matrice Jacobienne J comme étant :

$$J_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial w_{i,j,m}} \text{ pour un poids} \qquad \text{(A.5)}$$

$$J_{h,l} = \frac{\partial v_h}{\partial x_l} = \frac{\partial e_{k,q}}{\partial b_{i,m}} \text{ pour un biais}$$

où k est le k^{ième} élément du vecteur d'erreur.

q est le q^{ième} patron d'apprentissage

i est la i^{ième} rangée de W_m ou de b_m, selon le cas

j est la j^{ième} colonne de la matrice de poids W_m

h et l sont respectivement la h^{ième} et l^{ième} rangée et colonne de la matrice J

On trouve alors la correction à appliquer sur chaque paramètre du réseau en utilisant l'équation de Levenberg-Marquardt (Hagan, Demuth & Beale, 2002) :

$$\Delta\vec{x}_k = -\left[J^T(x_k)J(x_k) + u_k I\right]^{-1} J^T(x_k)v(x_k) \qquad \text{(A.6)}$$

Ici, u_k est divisé par un nombre $\varphi > 1$ (typiquement 10) et le pas est accepté si la somme des erreurs au carré décroit. u_0 sera posé égal à 100 par défaut.

Si la somme des erreurs au carré augmente, u_k est multiplié par φ (i.e. on se rapprochera de la direction de la plus forte pente), le pas est ignoré et on recalcule la nouvelle direction. Ceci garanti que l'on fera toujours diminuer la somme des carrés à chaque itération.

A.3.4 Autres méthodes d'entraînement de réseau

On a vu que l'entraînement d'un réseau de neurones est très similaire à un problème d'optimisation numérique.

Or, comme dans le cas des méthodes à gradient, les algorithmes d'apprentissage traditionnels cherchent surtout à trouver un optimum local, qui peut ou non constituer un modèle adéquat pour notre système. Plus le nombre de paramètres augmente, plus le nombre d'optimums locaux augmentera, ce qui fait que les modèles neuroniques représentant des systèmes de grande envergure seront sujet à tomber dans un optimum d'apprentissage local et à donner de piètres résultats, surtout dans le cas des fonctions non-différenciables et multi-modales (Sutton, 1986 ; Whitley, Starkweatehr & Bogart, 1990).

Pour pallier à ce problème, plusieurs chercheurs (Yao 1999, Lagaros & Papadrakakis, 2004) proposent des algorithmes d'apprentissage basés sur les algorithmes évolutionnaires, disposant d'une meilleure capacité à éviter le bruit et les optimums locaux.

A.3.5 Algorithmes évolutionnaires vs Algorithmes à gradient (BP et ses variations)

Plusieurs divergences sont retrouvées dans la littérature à savoir laquelle des méthodes est la plus adaptée pour résoudre le problème d'entraînement des réseaux neuronaux.

Fahlman (1988) et Johanson, Dowla & Goodman (1991) démontrent pour certains problèmes que les algorithmes évolutionnaires sont sensiblement plus lents à converger que les variantes rapides de BP et les algorithmes de gradient conjugué.

Prados (1992) et Secton, Dorsey & Johnson (1998) au contraire, rapportent que pour leurs problèmes les algorithmes évolutionnaires sont significativement plus rapides et plus stables de convergence.

Comme dans le cas des problèmes d'optimisation réguliers, la tendance générale est de constater que les algorithmes utilisant le gradient sont définitivement plus rapides

lorsque la solution initiale est très près de l'optimum, alors que les algorithmes évolutionnaires sont beaucoup plus efficaces lorsque l'on ne dispose pas d'une solution initiale valable. En réponse à cette constatation, plusieurs chercheurs ont conçu et utilisé des algorithmes hybrides utilisant d'abord un algorithme évolutionnaire pour trouver une solution valable, puis utilisant cette solution comme point initial pour un algorithme à gradient comme BP afin d'entraîner le réseau. Cette approche semble avoir donnée d'excellents résultats dans plusieurs applications (Hung & Adeli, 1994 ;Yang, Kao & Horng, 1996 ; Yan, Zhu & Hu, 1997).

Le seul constat trouvé dans la littérature stipulant le contraire est donnée par Kintano (1990) qui, quand à lui, stipule que les techniques d'optimisation hybride GA-BP (genetic algorithm-back propagation) sont au mieux, aussi efficaces que les variantes rapides de BP dans des réseaux à petite échelle, mais beaucoup moins efficaces dans le cas de réseau à grande échelle.

Toutefois, lorsque l'on se penche sur les articles, il semble que le constat donnant des résultats contradictoires soit davantage dû au fait que certains comparent les algorithmes évolutionnaires rapides avec la version classique de BP, alors que d'autres comparent les versions rapides de BP avec un algorithme évolutionnaire classique. On constate qu'il n'y a pas de véritable gagnant général sur la méthode à utiliser, celle-ci semble être très dépendante des données initiales que l'on possède et du problème à résoudre.

Pour ce projet, le but recherché est d'obtenir un réseau de neurones s'appliquant bien au plus grand nombre de fonctions possibles. Ainsi, lors de la création du réseau, nous allons utiliser un algorithme hybride qui accorde la moitié du temps alloué à l'entraînement à un algorithme génétique simple pour déterminer les valeurs de départ des poids et des biais, et nous utiliserons ensuite l'algorithme à gradient LMBP pour réduire l'erreur lors des entraînements subséquents.

A.3.6 Optimisation de la topographie d'un réseau

Jusqu'ici, nous nous sommes intéressés à déterminer les paramètres d'un réseau neuronal fixe donnant l'erreur quadratique moyenne minimale à la sortie par rapport à ses données d'entraînement.

Or, on sait (Yao, 1999) que plus un réseau de neurones a de neurones cachés, plus il risque d'interpoler piètrement entre les points car il disposera d'une trop grand flexibilité, et moins un réseau a de neurones cachées, moins il sera susceptible de modéliser adéquatement une fonction contenant beaucoup d'irrégularités ou d'inflexions.

Ainsi, pour obtenir un réseau performant, il faut, en plus d'optimiser les paramètres d'un réseau, construire un réseau dont la topologie sera optimale pour le problème en cours.

On se permet ici d'ouvrir une parenthèse pour noter le parallèle entre la construction d'un réseau de neurones et un design structurel. En effet, dans le cas du design structurel, on cherche à optimiser les paramètres pour minimiser une fonction coût. Bien que ceci améliore souvent le design, il ne faut pas perdre de vue qu'optimiser la paramétrisation en plus des valeurs numériques des paramètres ne peut que produire un résultat davantage optimal. Bien sûr, ceci est plus facile à faire dans le cas d'un réseau de neurones que dans un cas de design puisque la topologie de la fonction est très stricte et bien définie.

Dans les dernières années, plusieurs chercheurs ont proposés des algorithmes permettant d'optimiser la topologie des réseaux de neurones. La topologie changeable d'un réseau est constituée du nombre de neurones sur la couche cachée et de la connectivité du réseau, sans oublier la fonction de transfert de chaque neurone.

Jusqu'à maintenant, le design de l'architecture des réseaux est encore pratiquement uniquement effectué par des êtres humains grâce à l'expérience et par une série d'essais et erreurs.

Toutefois, plusieurs algorithmes constructifs et destructifs sont proposés pour la construction automatisée de réseaux de neurones (Fahlman & Lebiere, 1989 ; Roy, Kim & Mkhopadhyay, 1993 ; Hwang, You & Jou, 1996). Par exemple, le réseau de corrélation en cascade ajoute un neurone caché à la fois et optimise le réseau après chaque ajout, jusqu'à atteindre une convergence sur l'erreur quadratique moyenne.

Les réseaux constructifs commencent avec un réseau minimal et ajoute des caractéristiques (connections et neurone) une à une jusqu'à atteindre un critère de convergence. Les réseaux destructifs, à l'opposé, commencent avec un réseau « complet » et enlèvent des caractéristiques une à une tant que l'erreur n'augmente pas de façon remarquable.

On constate, toutefois, que ces méthodes constructives et destructives investiguent seulement une portion réduite des sous-ensembles possibles, plutôt que tous le domaine réalisable. Tel un algorithme à gradient, ils sont susceptibles de tomber dans un optimum local (Yang, Kao & Horng, 1996).

Le problème de la construction d'une architecture neuronale peut être vu comme un problème d'optimisation consistant à trouver le point de performance le plus élevé d'une surface.

Plusieurs caractéristiques de cette surface tendent à démontrer que les algorithmes évolutionnaires seraient les candidats les plus appropriés pour résoudre ce problème. Entre autre (Miller, Todd & Hegde, 1989) :

1 – La surface est non-différenciable puisque le changement dans le nombre de neurones et de connections est discret et peut avoir un effet discontinu sur les performances.
2 – La surface est complexe et bruitée puisque le lien d'une architecture à sa performance est fortement épistatique et dépend de la méthode d'évaluation utilisée.
3 – La surface est multi-modale puisque différentes architectures peuvent avoir la même performance.
4 – La surface est trompeuse puisque des architectures similaires peuvent avoir de très différentes performances.

Toutefois, jusqu'à maintenant, les méthodes évolutionnaires de construction des réseaux de neurones impliquent de résoudre un lot de sous-problèmes d'entraînement à chaque fois, ce qui peut demander un temps de calcul considérable dans le cas des problèmes d'envergure industrielles comme celui que nous allons aborder. Pour cette raison, et parce que la littérature ne semble pas démontrer l'efficacité de ces méthodes pour un problème de grande envergure, les réseaux de neurones de ce document n'utiliserons pas ces méthodes de construction topographique évolutionnaires du réseau.

Il est toutefois avantageux d'utiliser tout de même une méthode de construction topographique du réseau, puisque cela automatise le choix du nombre de neurones du réseau, enlevant ainsi le besoin pour l'utilisateur de le déterminer manuellement.

Les réseaux utilisés se construiront donc grâce une simple méthode constructive, soit la corrélation en cascades (voir figure A.5).

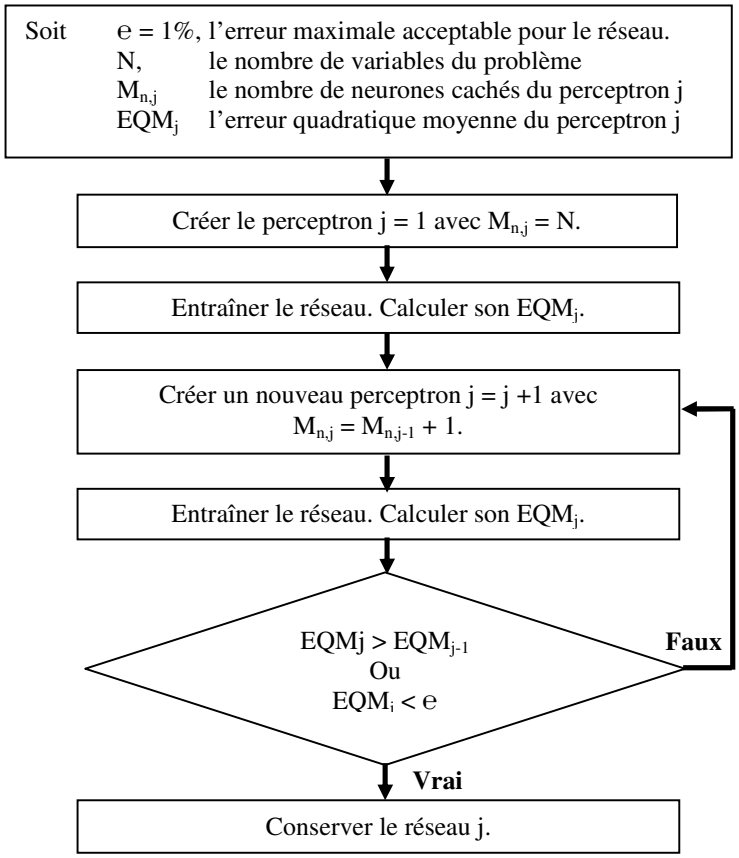


Figure A.5 : Construction d'un réseau par corrélation en cascades

A.4 Choix final des réseaux de neurones

À la lumière de la discussion contenue dans ce chapitre, les choix suivants ont été faits concernant les RNA utilisés par les algorithmes de ce document :

- Ce seront des perceptrons disposant d'une seule couche de neurones cachés (fig. A.3).
- Lors de leur création, les poids et les biais initiaux seront de petites valeurs aléatoires (entre -1 et 1). Ces valeurs seront modifiées à l'aide de 1000 évaluations d'un algorithme génétique régulier à codage réel sans heuristiques

(voir section 5.3, avec $N_{pop} = 40$, $p_{moyen} = 0.5$, $\alpha_c = 0.95$ et $\alpha_m = 0.05$) dans le but de réduire l'erreur quadratique initiale du réseau.

- La suite de leur entraînement se fera à l'aide de l'algorithme LMBP jusqu'à convergence suffisante (0.01% d'erreur par défaut) ou atteinte du temps maximal alloué (500 passes par défaut).
- La construction du réseau se fera à l'aide d'une corrélation en cascades (fig. A.5).

Lorsqu'il y a lieu, les apprentissages en temps réel du réseau se feront à l'aide de l'algorithme LMBP.

Annexe B.
CAS TEST DE LA POUTRE D'ÉPAISSEUR VARIABLE

B.1 Poutre d'épaisseur variable

Cette section décrit un exemple analytique simple mettant en évidence le principe et l'apport du raffinement progressif d'un processus de RPC.

L'exemple est celui de la figure B.1. Il s'agit d'une poutre de section rectangulaire et d'épaisseur variable soumise à une pression verticale variable orientée vers le bas. L'objectif sera de déterminer l'épaisseur (selon l'axe y) de la poutre qui minimisera son poids tout en étant capable de supporter la charge répartie *P(z)*.

Le poids *W* d'une poutre de section rectangulaire est défini par la relation suivante :

$$W = \rho b \int_{x=0}^{L} h(z)dz \qquad \text{(B.1)}$$

où
- ρ est la densité du matériel en kg / m^3.
- b est la largeur de la poutre en m.
- h est l'épaisseur de la poutre en m.
- L est la longueur de la poutre en m.

La largeur *b* de la poutre sera une constante fournie avec le problème. Pour un poids minimal, on désire donc minimiser l'épaisseur de la poutre *h* en chaque point.

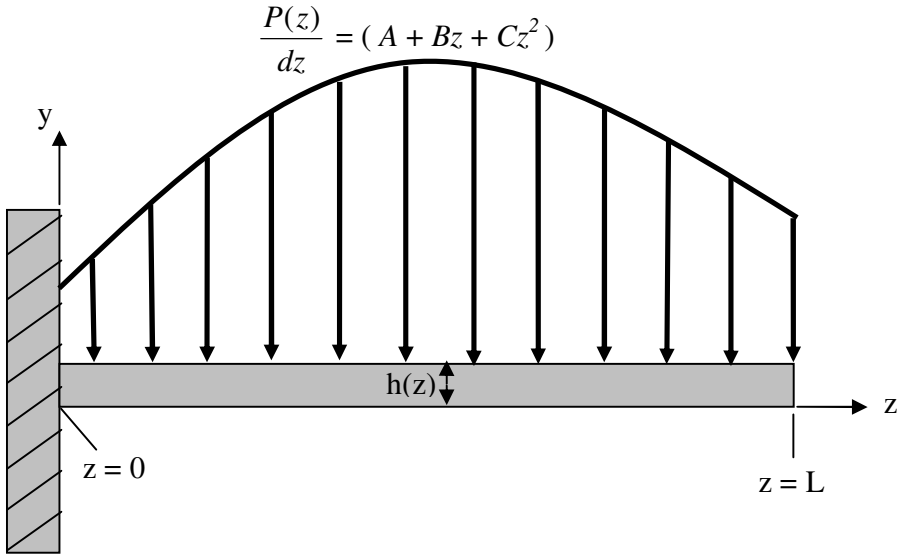


Figure B.1 : Exemple d'une poutre soumise a une force verticale variable

Selon la théorie de la résistance des matériaux, cette force générera en chaque point une contrainte de tension/compression σ_f due à la flexion et un effort tranchant σ_v, les points les plus contraints se situant aux surfaces supérieures et inférieures de la poutre.

La contrainte totale au carré sera donc la suivante :

$$\sigma^2 = \sigma_f{}^2 + \sigma_v{}^2 \qquad \text{(B.2)}$$

En un point $z = z_1$, la contrainte de flexion peut être calculée selon la relation B.3.

$$\sigma_f = \frac{h}{2I_z} \int_{z=z_1}^{L} (z - z_1) P(z) dz \qquad \text{(B.3)}$$

où I_z est le moment d'inertie de la poutre

Pour une poutre de section rectangulaire, $I_z = \frac{bh^3}{12}$.

En supposant que *P(z)* soit représenté par le polynôme $A + Bz + Cz^2$, on obtient, après quelques manipulations de B.3, la relation suivante pour la flexion :

$$\sigma_f(z_1) = \frac{1}{bh^2}\left[\frac{3C\left(L^4 - z_1^{\,4}\right)}{2} + 2\left(L^3 - z_1^{\,3}\right)(B - Cz_1) + 3\left(L^2 - z_1^{\,2}\right)(A - Bz_1) - 6(L - z_1)(Az_1)\right]$$

(B.4)

Quant à l'effort tranchant, sa formulation générale pour une poutre rectangulaire est la suivante :

$$\sigma_v(z_1) = \frac{F_v(z_1)}{bh} \qquad \text{(B.5)}$$

où $F_v(z)$ est la force totale nette exercée parallèlement à une section de poutre au point z.

Par un diagramme des corps libres (DCL), on pourrait démontrer la relation suivante pour cet exemple :

$$F_v(z_1) = \int_{z=z_1}^{L} P(z)dz \tag{B.6}$$

Après quelques manipulations algébriques, on obtient la relation suivante pour l'effort tranchant :

$$\sigma_v(z_1) = \frac{1}{bh}\left[\frac{C}{3}\left(L^3 - {z_1}^3\right) + \frac{B}{2}(L^2 - {z_1}^2) + A(L - z_1)\right] \tag{B.7}$$

Selon le matériel utilisé, nous connaîtrons l'effort maximal admissible pour σ, soit σ_{max}. L'une des contraintes du problème sera alors

$${\sigma_{\max}}^2 \geq {\sigma_f}^2 + {\sigma_v}^2 \tag{B.8}$$

Ou encore, pour un point z_1 donné, en substituant les relations B.4 et B.8 :

$${\sigma_{\max}}^2 \geq \left(\frac{K_1}{h^2}\right)^2 + \left(\frac{K_2}{h}\right)^2 \tag{B.9}$$

Avec

$$K_1 = \frac{1}{b}\left[\frac{3C\left(L^4 - {z_1}^4\right)}{2} + 2\left(L^3 - {z_1}^3\right)(B - Cz_1) + 3\left(L^2 - {z_1}^2\right)(A - Bz_1) - 6(L - z_1)(Az_1)\right]$$

$$K_2 = \frac{1}{b}\left[\frac{C}{3}\left(L^3 - {z_1}^3\right) + \frac{B}{2}(L^2 - {z_1}^2) + A(L - z_1)\right]$$

Puisque le poids varie linéairement avec l'épaisseur h, on voudra trouver l'épaisseur minimale h_{min} en chaque point, et donc résoudre :

$$\sigma_{\max}{}^2 \geq \left(\frac{K_1}{h_{\min}{}^2}\right)^2 + \left(\frac{K_2}{h_{\min}}\right)^2 \tag{B.10}$$

Après quelques manipulations algébriques, on obtient l'expression B.11.

$$h_{\min}{}^4 \sigma_{\max}{}^2 - h_{\min}{}^2 K_2{}^2 - K_1{}^2 = 0 \tag{B.11}$$

À partir de cette expression, on peut aisément en trouver les racines et découvrir que :

$$h_{\min}{}^2 = \frac{K_2{}^2 \pm \sqrt{K_2{}^4 + 4\sigma_{\max}{}^2 K_1{}^2}}{2\sigma_{\max}{}^2} \tag{B.12}$$

L'expression sous la racine sera toujours supérieure à $K_2{}^2$. Ainsi, puisqu'une valeur négative de $h_{min}{}^2$ est impossible, on rejette la solution impossible et on découvre finalement l'expression B.13, qui est l'expression analytique voulue pour h_{min}.

$$h_{\min} = \left(\frac{K_2{}^2 + \sqrt{K_2{}^4 + 4\sigma_{\max}{}^2 K_1{}^2}}{2\sigma_{\max}{}^2}\right)^{1/2} \tag{B.13}$$

Avec

$$K_1 = \frac{1}{b}\left[\frac{3C\left(L^4 - {z_1}^4\right)}{2} + 2\left(L^3 - {z_1}^3\right)(B - Cz_1) + 3\left(L^2 - {z_1}^2\right)(A - Bz_1) - 6(L - z_1)(Az_1)\right]$$

$$K_2 = \frac{1}{b}\left[\frac{C}{3}\left(L^3 - {z_1}^3\right) + \frac{B}{2}(L^2 - {z_1}^2) + A(L - z_1)\right]$$

De plus, pour des besoins de fabrication, on désire que la poutre ait une épaisseur minimale d'au moins 0.04 m, même si une épaisseur plus petite permettrait de soutenir les contraintes. La seconde contrainte du problème sera :

$$h(z) \geq 0{,}04 \qquad \text{(B.14)}$$

Les spécifications du client ont les valeurs suivantes :

A = 100 x10^3 N
B = 50 x10^3 N
C = -80 x10^3 N
L = 1 m
b = 0.1 m
σ_{max} = 100 x10^6 Pa
ρ = 7770 kg / m^3

La figure B.2 illustre la solution théorique pour la poutre de poids minimal supportant cette charge et répondant aux spécifications ci-dessus.

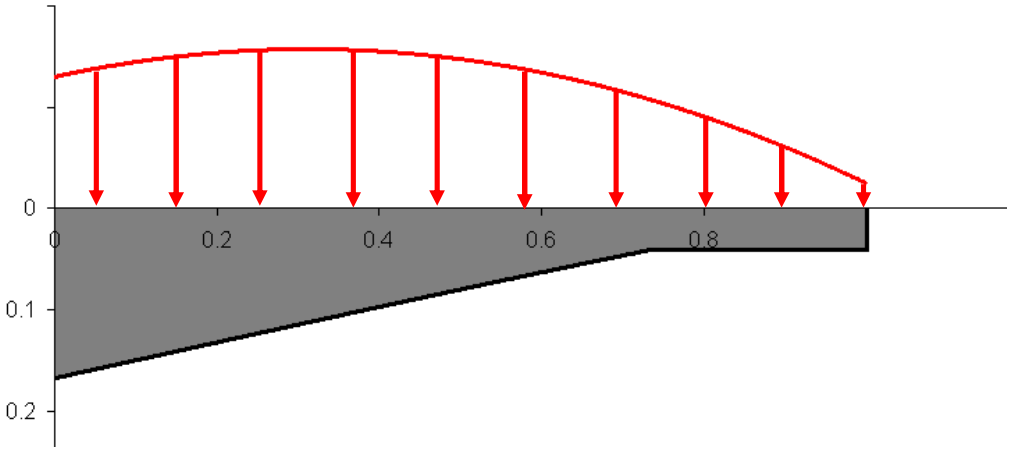


Figure B.2 : Poutre d'épaisseur minimale répondant aux spécifications données. Application d'un algorithme d'optimisation pour le dimensionnement de la poutre

Supposons maintenant que l'on ne connait pas l'expression analytique de la solution et que l'on désire minimiser le poids de la poutre illustrée à la figure B.2 en ne variant que l'épaisseur *h(z)*.

D'abord, il s'agit de créer un modèle paramétrique de solution. Puisque cette section du rapport s'intéresse à illustrer l'impact positif possible du raffinement progressif sur le processus, la paramétrisation sera effectuée en conséquence : c'est-à-dire que cette paramétrisation ne sera pas construite de façon à faciliter la découverte d'une solution, mais plutôt de façon à générer plusieurs variables ayant une influence très différente sur la fonction coût. De plus, cette paramétrisation sera effectuée de façon à reproduire l'environnement multimodal et l'interdépendance des variables propres aux problèmes plus complexes.

Soit D_R, un nombre entier représentant le nombre de paliers de variables différentes pour le problème.

La poutre sera discrétisée en $P_d = 2^{D_R - 1}$ points intermédiaire, dont les épaisseurs h_i seront générées à l'aide d'un vecteur $\vec{a}$ contenant un nombre $N = 2^{D_R} - 1$ variables.

L'épaisseur h_i du $i^{\text{ème}}$ point de la poutre sera définie comme étant :

$$h_i = \sum_{k=0}^{D_R - 1} a_j \tag{B.15}$$

Avec $$j = 2^k + Ent\left(\frac{i-1}{2^{D_R - 1 - k}} \right)$$

où $Ent(x)$ est la partie entière de x

La position z_i des points i de la poutre sera définie par la relation :

$$z_i = L\left(\frac{2i-1}{2^{D_R}} \right) \tag{B.16}$$

Les épaisseurs des extrémités encastrées et libres de la poutre seront respectivement nommées h_0 et h_L, et seront extrapolées à partir des valeurs de h_i selon la relation B.17.

$$h_0 = h_1 - z_1 \frac{h_2 - h_1}{x_2 - x_1} \tag{B.17}$$

$$h_L = h_{P_d} + \left(L - z_{P_d} \right) \frac{h_{P_d} - h_{P_d - 1}}{x_{P_d} - x_{P_d - 1}}$$

Ce que l'on obtient, en fin de compte, est un peu comme si chacune des N variables de $\vec{a}$ était un bloc d'une certaine longueur prédéfinie, et dont l'épaisseur est directement la valeur de la variable a_i en question.

La figure B.3 illustre la disposition des blocs de construction a_i ainsi que la position des points z_i pour un nombre de paliers D_R = 4. On y constate, par exemple, que si l'on désire évaluer l'épaisseur du point i = 4, le calcul à effectuer sera $h_4 = a_1 + a_2 + a_5 + a_{11}$.

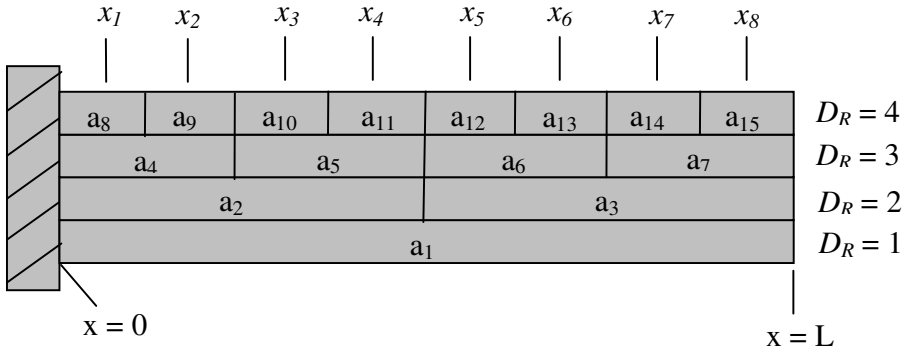


Figure B.3 : Paramétrisation de la poutre et position *x* des points pour D_R = 4.

Maintenant que notre modèle paramétrique a été généré, il ne reste plus qu'à poser le problème d'optimisation lui-même. À partir de ce modèle, il est possible d'évaluer numériquement l'intégrale B.1, qui est en fait la fonction coût G($\vec{a}$) sans contraintes de notre problème :

$$G(\vec{a}) = W \approx b\rho \sum_{i=0}^{P_d} (z_{i+1} - z_i) \left(\frac{h_i + h_{i+1}}{2} \right) \tag{B.18}$$

où $$h_0 = h_1 - z_1 \frac{h_2 - h_1}{x_2 - x_1}$$

$$h_{P_d+1} = h_{P_d} + (L - z_{P_d}) \frac{h_{P_d} - h_{P_d-1}}{x_{P_d} - x_{P_d-1}}$$

Ensuite, dans un modèle réel, il importerait de définir une façon d'évaluer les contraintes du modèle. Pour ce modèle simple, puisque la hauteur minimale h_{min} a déjà été évaluée

en chaque point, on peut directement poser cette valeur comme contrainte du modèle. On dispose donc des contraintes suivantes :

$$h_i \geq h_{\min} \qquad \text{pour i} = 1 \ldots P_D$$

$$h_i \geq 0{,}04 \qquad \text{pour i} = 1 \ldots P_D$$

où h_{min} est la valeur calculée à l'équation B.13.

De plus, on doit poser des bornes pour les variables du vecteur $\vec{a}$. Chaque palier n'aura le droit de modifier positivement ou négativement la hauteur que par la moitié de la hauteur du palier précédent. Le premier palier, quant à lui, pourra varier entre 0 et 0,5 mètres, selon les besoins. On obtient alors les bornes suivantes pour le problème :

$$a_{\min,1} = 0 \qquad a_{\max,1} = 0{,}5$$

$$a_{\min,j} = \frac{-0{,}5}{2^K} \quad a_{\max,j} = \frac{0{,}5}{2^K}$$

où K est le palier de la variable actuelle, que l'on peut aussi définir comme le plus grand entier K pour lequel $j \geq 2^K$.

À ce stade, le problème est entièrement défini. Nous utiliserons la solution initiale suivante comme point initial pour démarrer les éventuelles optimisations :

$$x_j = 0{,}5 \qquad \text{pour } j \in \{0,1,2 \ldots N\}$$

où x_j sont les variables adimensionnelles du problème.

B.1.1 Avantages et inconvénients du raffinement progressif pour le problème

Afin de mettre en évidence les avantages du raffinement progressif, le problème sera résolu plusieurs fois en utilisant différents nombres de phases de raffinement (z_{max} = 1, z_{max} = 2, z_{max} = 4, z_{max} = 8 et z_{max} = 16). Tous ces z_{max} seront utilisé pour divers nombres de paliers de variables D_R = 3, 5 et 7, ce qui implique respectivement un nombre de paramètres N égal à 7, 31 et 127. L'algorithme AGENT (voir chapitre 5) sera utilisé pour résoudre le problème. Étant donné la nature aléatoire des algorithmes génétiques, l'algorithme sera lancé 100 fois pour chaque combinaison de D_R et de z_{max}. Les paramètres utilisés par l'algorithme sont affichés dans la table B.1.

Table B.1. Paramètres utilisés par l'AGENT

Paramètre	**Valeur**
Nombre d'îlots ($k_{île}$)	4
Population (N_{pop})	40
α_m	0,5
γ^0	0,2
Ev_{max}	5000
δ_{min}	0

Afin d'avoir un réseau de neurones capable d'identifier les valeurs des importances relatives initiales des solutions, un RNA a été entraîné au préalable pour agir comme modèle substitut de la fonction. Ce RNA utilise comme patrons d'entraînement la solution initiale du problème, ainsi que deux solutions supplémentaire pour chacune des variables obtenues en remplaçant successivement la valeur de la variable dans la solution initiale par sa borne maximale et sa borne minimale.

Étant donné la nature discontinue donnée à la forme de la poutre par la solution, le point optimal pour chaque palier D_R ne sera pas exactement le même que le véritable point optimal théorique. Néanmoins, le ratio entre l'optimum théorique et chacune des solutions trouvées pourra tout de même servir d'indicateur de performance de l'algorithme pour les problèmes étudiés. L'optimum théorique a été posé à

$F_{Théorique}^{*}$ = -76.477, qui est la meilleure solution obtenue par les algorithmes d'optimisation lors de sa résolution.

Déterminons maintenant deux indicateurs pour comparer les performances des algorithmes utilisés. Le premier indicateur mesure donc *Eff*, ou l'efficacité de l'algorithme, que l'on peut décrire comme la qualité des solutions qu'il génère. Il est calculé grâce à la relation B.19.

$$Eff = \frac{F^{*}}{F_{Théorique}^{*}} \tag{B.19}$$

où F^{*} est la meilleure valeur adaptative découverte par l'algorithme d'optimisation pour ce problème.

$F_{Théorique}^{*}$ est la meilleure valeur adaptative connue pour ce problème.

Le second indicateur mesure le T_{CV}, le temps de convergence de l'algorithme. Cet indicateur nous donne une idée des ressources requises par un algorithme avant de générer sa solution finale. L'indicateur est calculé comme étant égal au nombre d'évaluations nécessaires à l'algorithme pour atteindre 95% de son amélioration finale.

La relation B.20 donne le calcul effectué pour l'évaluation de $T_{CV.}$

$$F^{CV} = F^{0} + 0{,}95\left(F^{*} - F^{0}\right) \tag{B.20}$$

où F^{0} est la valeur adaptative initiale du meilleur individu de l'algorithme.

T_{CV} est le temps nécessaire (en nombre d'évaluations) au meilleur individu de l'algorithme pour atteindre une valeur de valeur adaptative plus grande que F^{CV}.

Résultats

Les figures B.4, B.5 et B.6 illustrent différentes formes typiques de poutres obtenues pour D_R = 3, 5 et 7. Plus D_R augmente, plus l'algorithme dispose de liberté pour modeler la poutre, et devrait donc être davantage capable de représenter fidèlement l'optimum théorique. Malgré ceci, on observe que plus D_R est élevé, plus la solution s'éloigne de cet optimum théorique.

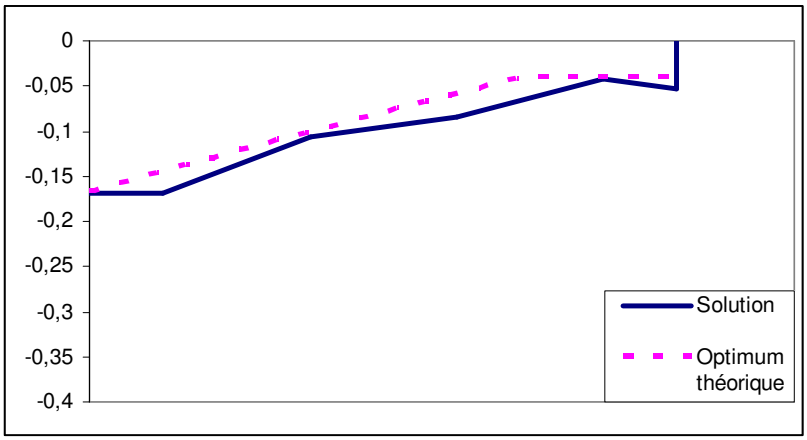


Figure B.4 : Résultat typique obtenu pour D_R = 3 après 5000 évaluations

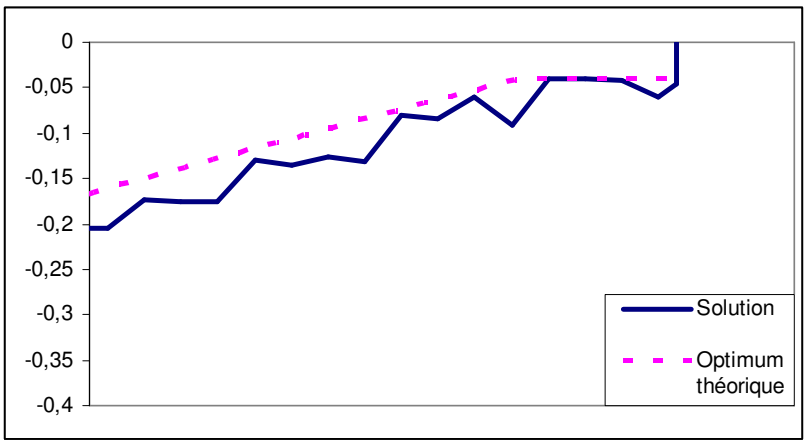


Figure B.5 : Résultat typique obtenu pour D_R = 5 après 5000 évaluations

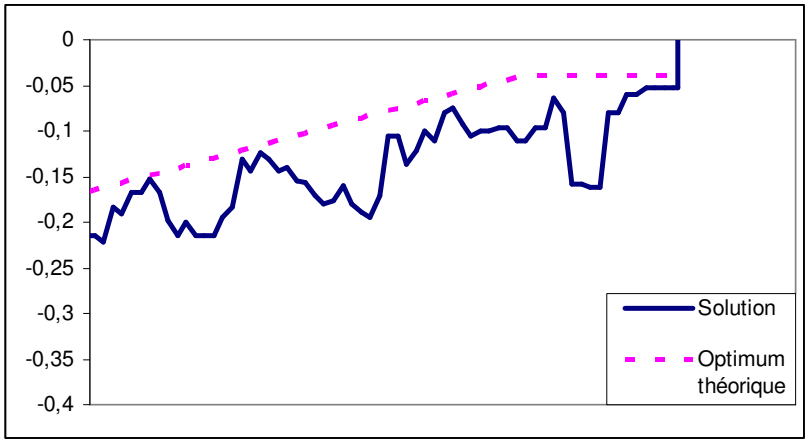


Figure B.6 : Résultat typique obtenu pour $D_R = 7$ après 5000 évaluations

Ceci est dû au fait qu'un algorithme de recherche global tel que l'AGENT tente d'évaluer plusieurs combinaisons de variables plutôt que de foncer vers l'optimum le plus rapproché. Pour des problèmes complexes tels que les problèmes à $D_R = 5$ et $D_R = 7$, l'algorithme doit trouver la combinaison optimale sur un nombre total respectif de 31 et de 127 variables, ce qui est impossible dans le temps imparti de 5000 évaluations pour le problème. En effet, sur 127 variables, explorer uniquement les points bornant le domaine ne requière pas moins de $2^{127} \approx 1{,}70141 \times 10^{38}$ évaluations ! Pourtant, il s'agit bel et bien des dimensions de problèmes que l'on peut s'attendre à vouloir gérer dans un cas industriel. Une géométrie réaliste de pièce mécanique demande une quantité respectable de variables, et chaque évaluation d'une fonction réelle est généralement très coûteuse, ce qui oblige les algorithmes d'optimisation à rechercher la meilleure solution dans les temps impartis, plutôt que de rechercher le véritable optimum global.

Ces résultats préliminaires fournissent aussi l'intuition d'une règle d'or en optimisation de pièces mécaniques, surtout en phase conceptuelle : il est important de travailler avec des modèles aussi simples que possible si l'on veut un procédé de design efficace.

Les figures B.7, B.8 et B.9 comparent l'efficacité moyenne et le temps de convergence moyen des résultats obtenus pour respectivement D_R = 3, 5 et 7 paliers selon le nombre de phases de raffinement z_{max} utilisées.

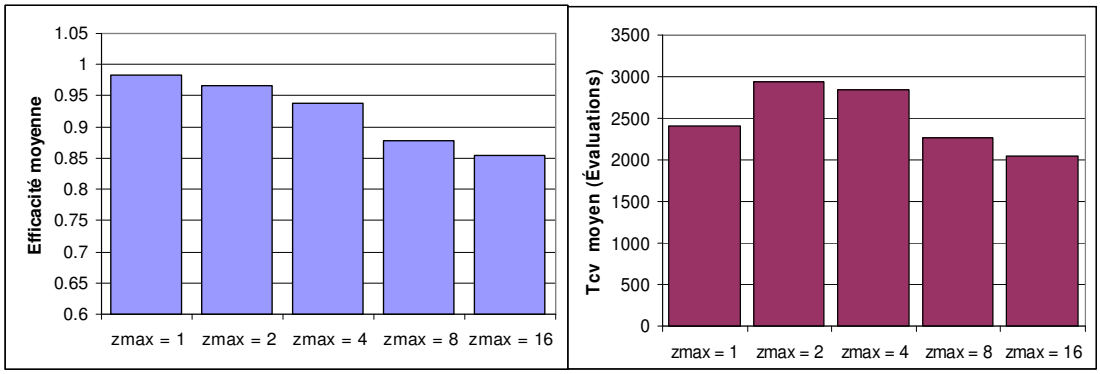


Figure B.7 : Efficacité moyenne et temps de convergence moyen pour D_R = 3

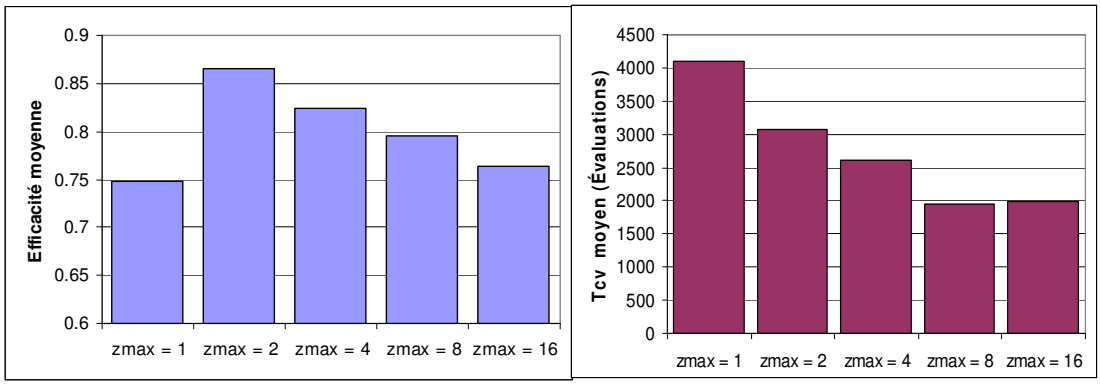


Figure B.8 : Efficacité moyenne et temps de convergence moyen pour D_R = 5

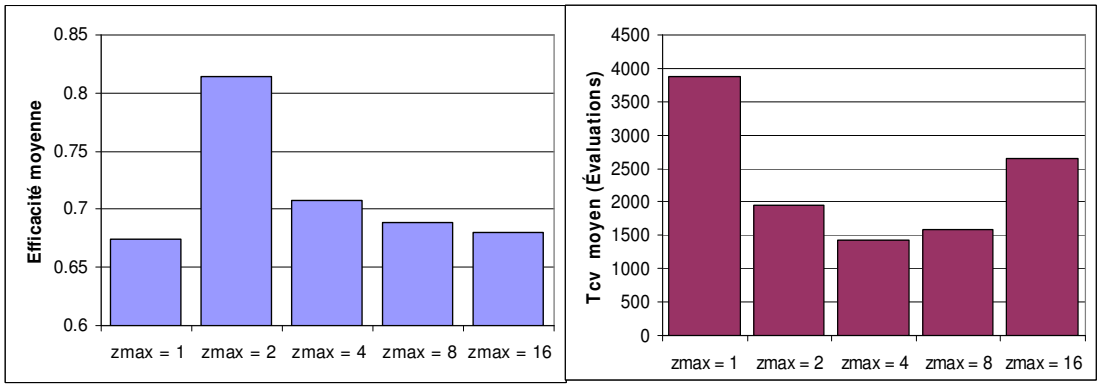


Figure B.9 : Efficacité moyenne et temps de convergence moyen pour D_R = 7

On peut y constater que le choix de z_{max} influence énormément la qualité des résultats.

Effet du raffinement progressif pour $D_R = 3$

Sur la figure B.7, on observe que, pour le problème le moins complexe à 7 paramètres, le raffinement progressif ne fait en fin de compte que nuire à la qualité de la solution.

La figure B.10 illustre l'influence mesurée pour les paramètres aux environs de la solution initiale. La valeur par défaut C_z = 100 a été utilisée par les phases de raffinement. Ainsi, pour les algorithmes où $z_{max} > 1$, l'ensemble initial des paramètres est composé de tous les paramètres pour lesquels l'importance relative est au-dessus de la valeur 1, soit x_1, x_3 et x_6 et x_7.

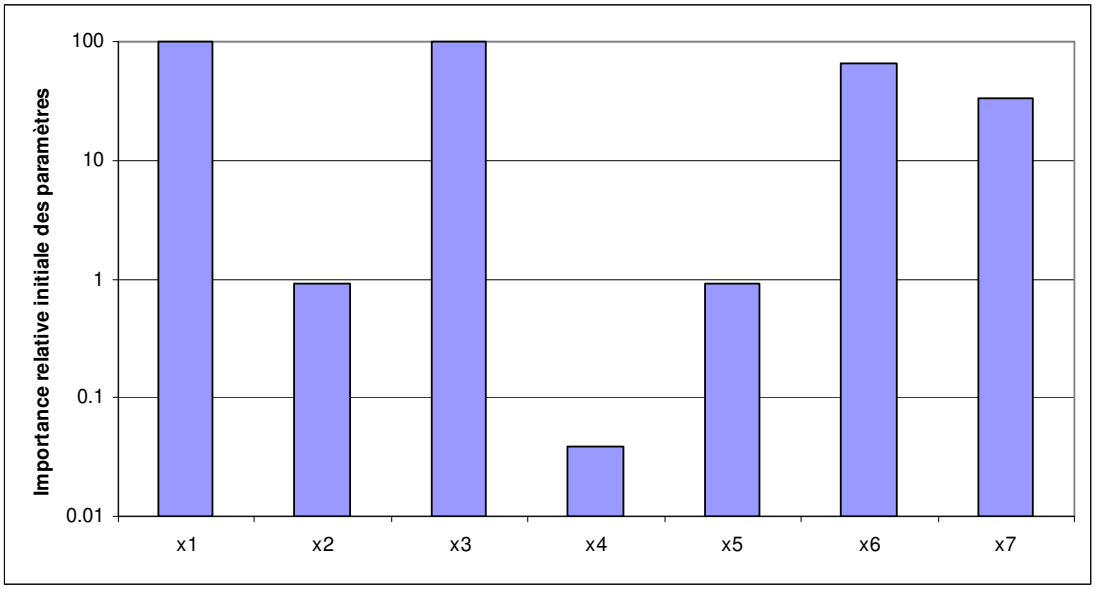


Figure B.10 : Importance relative initiale des paramètres pour $D_R = 3$.

Lors des phases successives de raffinement, en considérant que l'importance relative des paramètres conserve approximativement son ordre de grandeur, tous les paramètres seront utilisés. Ainsi, pour $z_{max} > 1$, l'effet obtenu est qu'un certain nombre d'évaluations de l'algorithme (égal à 5000 divisé par z_{max}) est effectué en n'utilisant que les quatre

paramètres les plus importants à la base, alors que les évaluations suivantes utilisent tous les paramètres.

Sur la figure B.7, on peut voir que le temps de convergence moyen de l'algorithme pour D_R = 3 et z_{max} = 1 se situe aux environs de 2400 évaluations. Si z_{max} = 2, l'algorithme utilisant toutes les variables aura donc pratiquement convergé avant même que la seconde phase de raffinement ne soit atteinte. Pour un problème si simple, l'ajout d'une seule phase de raffinement ne fait donc que ralentir la progression de l'algorithme, en le forçant à ajuster pendant 2500 évaluations des variables qui n'en ont pas besoin d'autant pour atteindre une valeur adéquate.

De plus, on se souvient que chaque fois que l'on atteint une nouvelle phase de raffinement, on doit réinitialiser les populations de chaque îlot de l'algorithme génétique. Pour les paramètres utilisés, cette initialisation génère donc 4x20 solutions aléatoires (qui ne tiennent pas compte des noyaux territoriaux) à chaque fois qu'une nouvelle phase de raffinement est enclenchée. Ainsi, plus z_{max} est élevé, plus le nombre de solutions générées de façon purement aléatoire est élevé, ce nombre s'élevant jusqu'à 15x4x20 = 1200 évaluations pour z_{max} = 16. Dans de telle conditions, il est facile de comprendre que si le problème dispose de peu de paramètres ayant une importance relative similaire comme c'est le cas ici, ajouter des phases de raffinement ne fait que nuire au fonctionnement normal de l'algorithme en le transformant en simple opérateur de recherche aléatoire, détériorant donc son efficacité et son temps de convergence.

Effet du raffinement progressif pour D_R = 5 et 7

Les histogrammes des figures B.11 et B.12 illustrent l'importance relative des paramètres pour respectivement 31 et 127 paramètres. On observe que pour D_R = 5, seuls 10 des 31 paramètres seront utilisés dans la première phase de raffinement si z_{max} > 1, alors que 22 des 127 paramètres seront considérés variables pour D_R = 7 et z_{max} > 1.

Si les ordres de grandeurs des importances relatives se maintiennent, toutes les variables seront incluse pour z_{max} = 2 si D_R = 5, alors que toutes les variables ne seront incluses qu'à z_{max} = 3 pour D_R = 7.

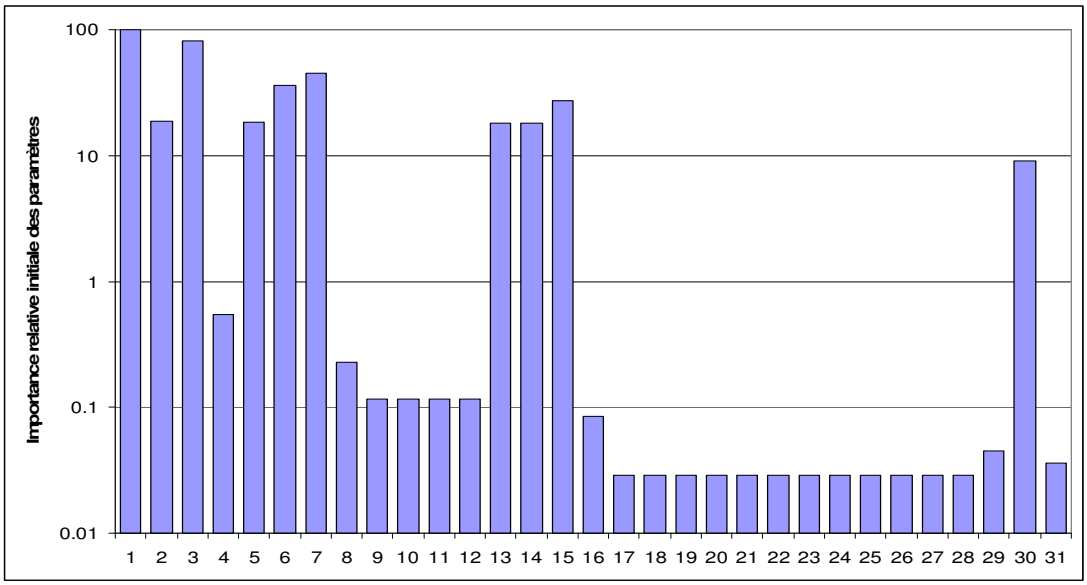


Figure B.11 : Importance relative initiale des paramètres pour D_R = 5.

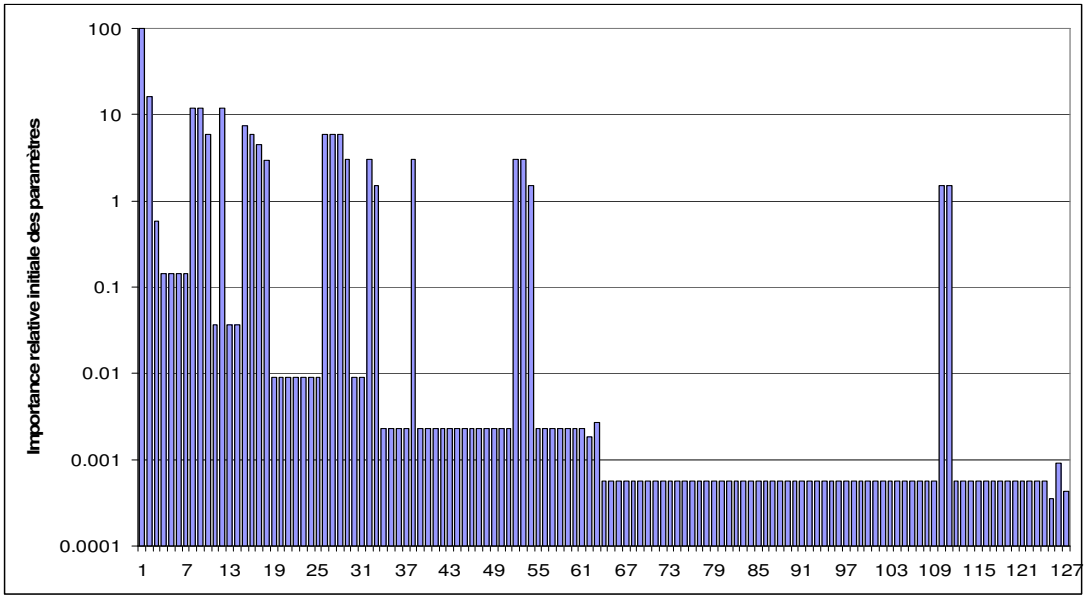


Figure B.12 : Importance relative initiale des paramètres pour D_R = 7.

Pour des problèmes ayant autant de variables, un optimiseur génétique régulier aurait de la difficulté à trouver rapidement une solution acceptable. En effet, pour des problèmes simples à 30 variables, la plupart des articles utilisent un nombre d'évaluations de l'ordre de $3x10^6$ (Boyer, Hervás-Martínez & Gracia-Pedrajas, 2005 ; Noman & Iba, 2008).

Les résultats des figures B.8 et B.9 démontrent donc que, en utilisant $z_{max} > 1$, on obtient non seulement une solution de meilleure qualité, mais aussi plus rapidement, que si l'on utilisait un algorithme sans raffinement progressif utilisant directement toutes les variables ($z_{max} = 1$).

Par contre, on constate aussi dans les deux cas que la meilleure solution est obtenue en utilisant $z_{max} = 2$. Ceci est dû au fait que dans chaque cas, lorsque toutes les variables sont en cause, l'algorithme perd énormément de temps à évaluer des points qui ont été raffinés à l'aide de variables peu importantes, alors qu'une meilleure amélioration serait possible en utilisant uniquement les variables plus importantes.

Ainsi, en 2500 évaluations, qui constitue le point où l'algorithme change de phase de raffinement pour $z_{max} = 2$, l'algorithme n'a pas encore eu réellement le temps de faire converger les variables de la première phase de raffinement au mieux de leur valeur. Puisque ces variables sont celles influençant le plus la valeur de la fonction coût, les performances de l'algorithme s'en trouvent ensuite amoindries. Pour $z_{max} > 2$, le nombre d'évaluations laissées à l'algorithme pour travailler uniquement avec ces paramètres est réduit encore davantage, ce qui amène les performances moindres de l'algorithme que l'on peut constater sur les graphiques pour z_{max} croissant au-dessus de 2 avec $D_R = 5$ et $D_R = 7$.

Par contre, il est intéressant de constater que, même si l'on pousse z_{max} à des valeurs énormes comme $z_{max} = 16$, l'apport à l'algorithme de la première phase de raffinement

avec peu de variable compense encore la perte d'efficacité due à la réinitialisation de la population entre chaque phase de raffinement. Donc, même en utilisant le raffinement progressif à l'aide de paramètres C_z et z_{max} instaurés sans connaissance initiale du problème, le raffinement progressif continue d'améliorer les performances de l'algorithme pour les problèmes à grand nombre de paramètres.

En bref, pour les problèmes de grande envergure pour lesquels on désire une solution rapide, il semble bien qu'un algorithme utilisant le raffinement progressif soit parfois apte à donner de meilleurs résultats que l'utilisation simple d'un algorithme génétique. Toutefois, on peut aussi constater que les valeurs optimales pour z_{max} et C_z dépendent grandement de la nature du problème à résoudre et de l'ordre de grandeur de différence entre les diverses importances relatives des paramètres.

Conséquemment, la méthode de raffinement progressif présentée dans ce document a fait ses preuves pour améliorer significativement la qualité des solutions fournies par les algorithmes génétiques pour ce problème, mais uniquement pour les cas où le problème dispose d'un grand nombre de paramètres par rapport au nombre d'évaluations dont on dispose, et davantage encore si l'utilisateur dispose de suffisamment de connaissances sur le problème pour bien définir les valeurs de z_{max} et de C_z. Une avenue intéressante pour une recherche future pourrait donc être de trouver une meilleure façon de sélectionner un ensemble de paramètres adéquats selon leur importance relative, ainsi que de trouver une meilleure méthode pour choisir le nombre d'évaluations permises lors de chaque phase de raffinement.

L'apport du raffinement progressif avec $z_{max} = 2$ et $C_z = 100$ sera à nouveau démontré au chapitre 7, cette fois pour le problème du design d'un disque de rotor de turbine à gaz.

B.1.2 Validation de la méthode hybride de gestion des contraintes utilisée

À la section 3.2.1, il a été proposé d'utiliser à la fois une pénalité de mort et une fonction de mérite adaptative pour tenir compte des contraintes dans la fonction coût de l'optimisation. À la connaissance de l'auteur, les optimiseurs utilisent plutôt parfois l'une, parfois l'autre de ces méthodes, plutôt que les deux simultanément. Aussi, cette section tente-t-elle d'utiliser le problème de la poutre afin de démontrer l'avantage d'utiliser une combinaison de fonction de mérite adaptative et de peine de mort lors du calcul de l'effet des contraintes sur la fonction coût.

Le problème sera résolu à l'aide de l'algorithme AGENT (voir chapitre 5) en utilisant les valeurs par défaut de l'algorithme comme paramètres et z_{max} = 1. Le nombre de paliers de variables D_R sera posé égal à 3 (7 variables), et le nombre d'évaluations maximal possible sera de 3000, que la section B.1.1 semble démontrer comme un temps suffisant pour que la solution $D_R = 3$ ait convergée.

L'algorithme utilisera la fonction coût telle que définie par l'équation 3.2. Dans le cas d'un algorithme génétique, la valeur de la fonction coût est aussi appelée la valeur adaptative (ou « fitness »). L'algorithme sera lancé 100 fois en utilisant uniquement les fonctions de mérite ($\psi = 0$), 100 fois en utilisant uniquement les pénalités de mort ($v = 0$) et finalement 100 fois en utilisant à la fois une fonction de mérite et une pénalité de mort.

Afin de bien démontrer l'effet escompté, les contraintes du problème doivent être actives dès le départ. Le point initial du problème sera donc modifié afin de se trouver plutôt dans le domaine irréalisable du problème. La poutre de volume zéro suivante sera donc utilisée comme solution initiale au problème :

$$a_j = 0 \quad \text{pour } j \in \{1,2\ldots N\}$$

De plus, dans le but de réduire la portion réalisable du domaine, les bornes des variables ont été ajustées comme suit :

$$a_{\min,1} = 0 \qquad a_{\max,1} = 0{,}14$$

$$a_{\min,j} = \frac{-0{,}14}{2^K} \qquad a_{\max,j} = \frac{0{,}14}{2^K}$$

où K est le palier de la variable actuelle, que l'on peut aussi définir comme le plus grand entier K pour lequel $j \geq 2^K$.

Finalement, dans le but d'augmenter les chances que les solutions au problème se situent hors du domaine réalisable, la créativité initiale de l'AGENT a été posée égale à $\gamma_c^0 = 0.001$.

La figure B.13 illustre le nombre de solutions irréalisables obtenues à la sortie de l'algorithme en utilisant soit uniquement une pénalité de mort $\psi = 500$, soit une fonction de mérite pour laquelle v est initialisé à la valeur 100, ou soit une méthode hybride additionnant ces deux pénalités.

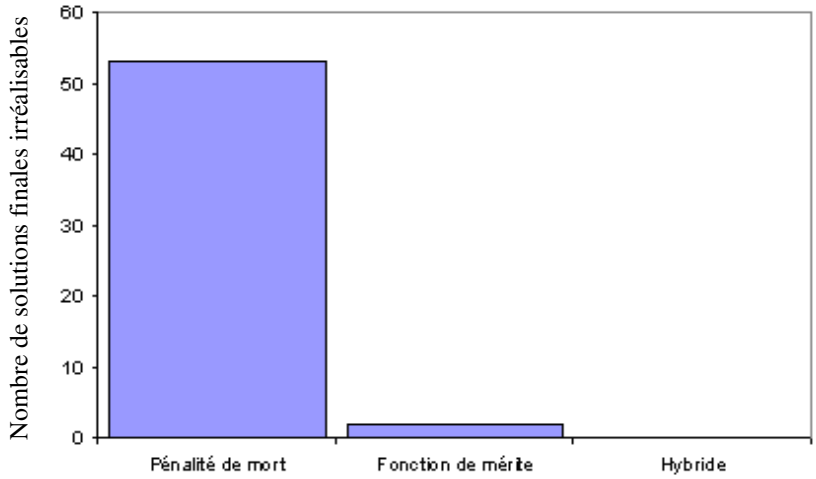


Figure B.13 : Nombre de solutions irréalisables (sur 100) selon le type de pénalité utilisé.

On constate d'abord qu'à partir d'une solution si loin dans le domaine irréalisable, la méthode utilisant des pénalités de mort est incapable de retrouver la moindre solution satisfaisant les contraintes 53 fois sur 100.

Ce phénomène peut être expliqué en observant l'évolution du poids de la poutre en cours d'optimisation (voir figure B.14). Effectivement, puisque la valeur adaptative de la fonction est pénalisée également pour toute la population, peu importe l'importance de la transgression de chaque contrainte, l'algorithme juge plus profitable de réduire le poids de la poutre à 0 que de chercher en direction d'un poids plus élevé au cas où l'une ou l'autre des contraintes pourrait disparaître.

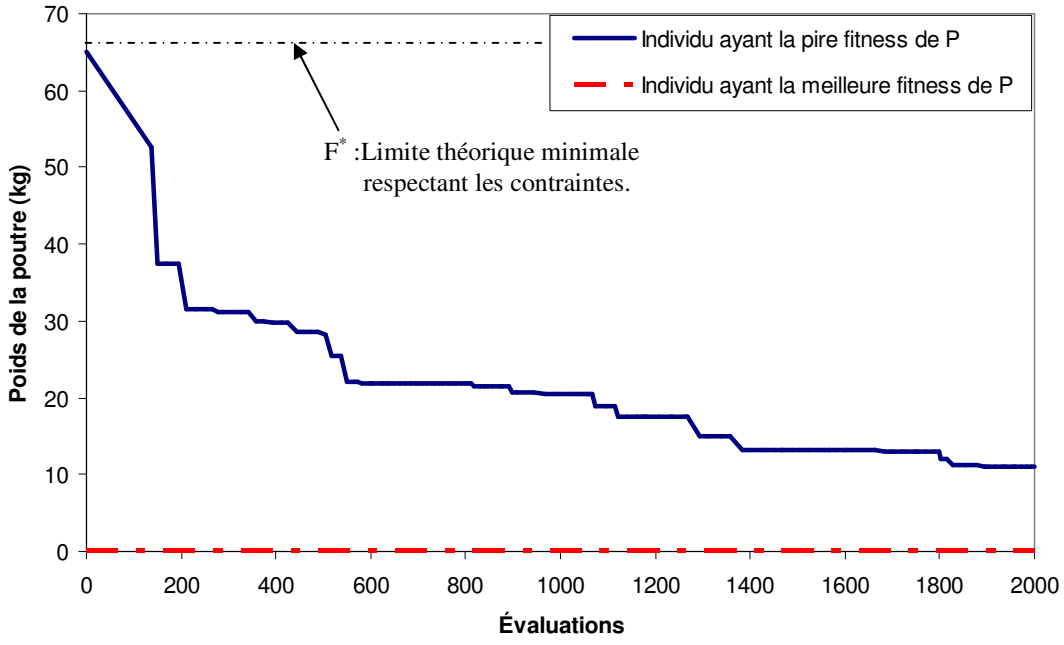


Figure B.14 : Évolution du poids de la poutre lors de la création d'une solution irréalisable par la méthode des pénalités de mort.

Dans un tel cas, les heuristiques encouragent donc la recherche en direction inverse du domaine réalisable, et seul l'aspect aléatoire des algorithmes génétiques peut ramener

une solution dans la portion du domaine ne transgressant pas les contraintes, ce qui explique la grande proportion de solution irréalisables obtenues.

Dans le cas des fonctions de mérite, seules deux solutions sont sorties irréalisables de l'algorithme génétique. Le phénomène les ayant produites est alors très différents. La figure B.15 illustre la progression de la valeur adaptative de la fonction au cours de l'optimisation. Les zones où la fonction est hors du domaine réalisable sont délimitées par les régions grises. On peut y constater que dans un tel cas, au contraire des solutions utilisant la pénalité de mort, l'algorithme trouve plusieurs solutions réalisables très rapidement au cours de l'optimisation.

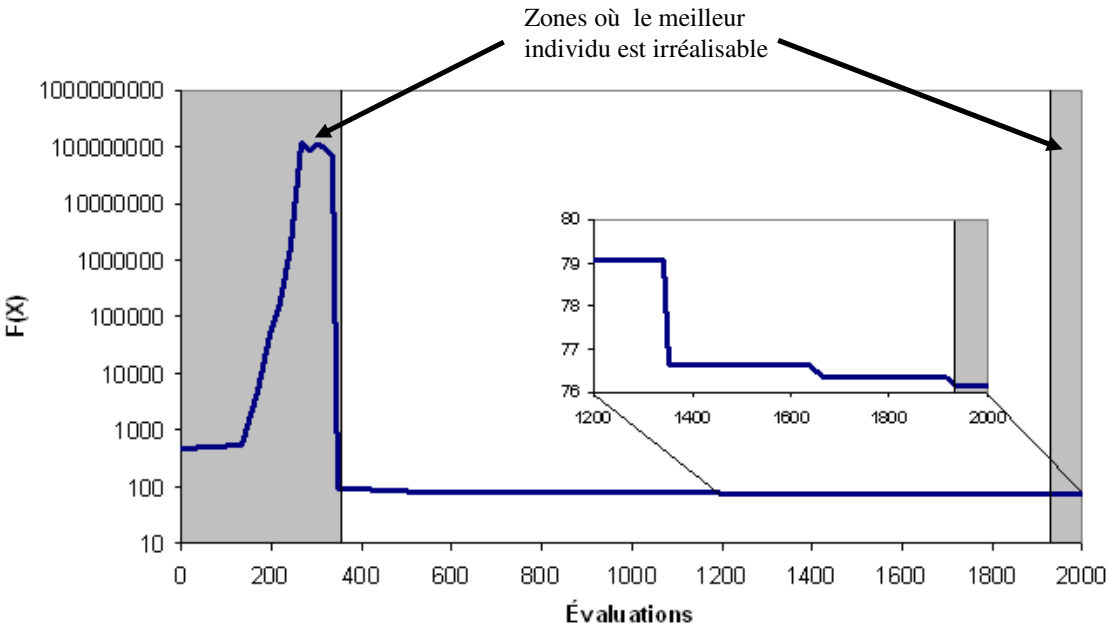


Figure B.15 : Évolution de la fonction coût lors de la création d'une solution irréalisable par la méthode des fonctions de mérite.

En effet, l'utilisation d'une pénalité qui s'accroit avec la distance entre la solution et le domaine réalisable génère un gradient positif en direction du domaine réalisable, ce qui

pousse les heuristiques de recherche de l'algorithme à favoriser cette direction de recherche.

Par contre, puisque cette pénalité est proportionnelle à la distance entre la solution et le domaine réalisable, et qu'elle diminue lorsque l'algorithme dispose uniquement de solutions réalisables, elle devient très faible lorsque l'algorithme est en état avancé de convergence, de façon à ce que si certains points légèrement hors des frontières sont sélectionnés par l'algorithme, il n'est pas impossible que ceux-ci disposent quand même d'une meilleure valeur adaptative que les autres points, même s'ils sont irréalisables.

Face à ce phénomène, on pourrait être tenté de limiter la diminution de la constante v. Par contre, puisque la définition exacte d'une faible valeur de v dépend beaucoup de la nature du problème étudié, il serait alors nécessaire d'ajouter un paramètre supplémentaire au procédé pour définir cette valeur, valeur qui ne serait pas nécessairement évidente à définir par un utilisateur qui connait peu le problème en question.

Nous avons donc tenté une approche hybride additionnant les deux pénalités, avec l'optique qu'elle pourrait à la fois permettre de ramener la population de l'algorithme vers le domaine réalisable, tout éliminant les possibilités de provoquer une solution irréalisable une fois l'état de convergence atteint. Sur la figure B.13, on constate en effet qu'aucune solution irréalisable n'a été produite par cette approche sur les 100 essais effectués.

Un désavantage potentiel de cette méthode est qu'elle pourrait potentiellement limiter l'algorithme à une portion du domaine réalisable qu'il aurait de la difficulté à quitter. En effet, la littérature (Yeniay, 2005) semble indiquer que pour plusieurs problèmes, il peut être avantageux de laisser l'algorithme dépasser temporairement les bornes du domaine

réalisable afin qu'il évite certains piètres optimums locaux causés par l'intersection de quelques unes des contraintes.

Dans notre cas, ce phénomène est amoindri par le fait que l'algorithme AGENT utilisé est un explorateur global plutôt efficace utilisant certains opérateurs de recherche aléatoires capable de demander l'évaluation de certains points loin en dehors des contraintes malgré un gradient défavorable. Les figures B.16 et B.17 illustrent respectivement les efficacités (équation B.19) et les temps de convergence (équation B.20) obtenus lors des tests.

Sur chaque figure, les lignes centrales des boîtiers représentent la moyenne des résultats, alors que les bornes des boîtiers représentent un écart type au dessus et en dessous de cette moyenne. Les lignes supérieures et inférieures représentent, quant à eux, les déciles supérieurs et inférieurs des résultats. Notez qu'afin de mieux représenter les tests sur une même base de comparaison, seuls les tests ayant produits des résultats n'enfreignant aucune contrainte ont été compilés pour produire ces figures.

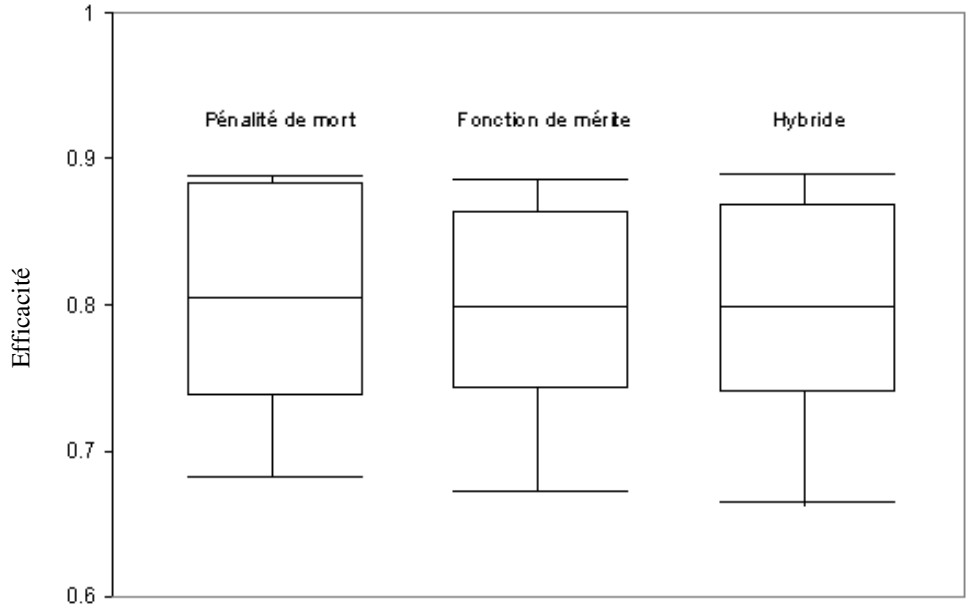


Figure B.16 : Efficacité de l'algorithme selon le type de pénalités utilisées

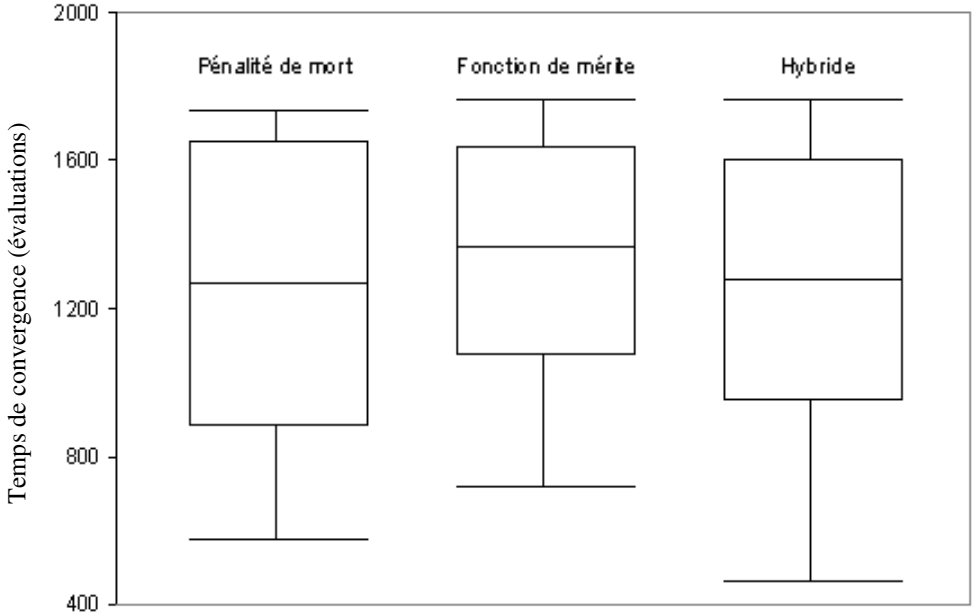


Figure B.17 : Temps de convergence selon le type de pénalités utilisées

En ce qui a trait à la qualité de la solution produite par les différentes méthodes (aussi appelée « efficacité », figure B.16), on constate bien peu de différences, hormis un avantage minime pour la peine de mort et une distribution moins concentrée pour la méthode hybride. D'ailleurs, quelques simples tests statistiques utilisant la loi de Student nous permettent d'affirmer avec une probabilité de plus de 95% que l'efficacité moyenne des trois méthodes est identique lorsqu'elles produisent une solution réalisable.

Pour ce qui est du temps de convergence (figure B.17), la méthode utilisant uniquement les fonctions de mérites semble être moins rapide pour résoudre ce problème. En effet, son temps de convergence moyen est d'environ 1360 itérations, comparativement à 1280 pour la méthode hybride, et 1271 pour la méthode utilisant uniquement les pénalités de mort. Les tests statistiques permettent d'affirmer avec plus de 95% de certitude que la méthode des pénalités de mort et la méthode hybride ont le même temps de convergence, alors qu'il est à 80% certain que la méthode utilisant les fonctions de mérite aient un temps de convergence plus lent d'environ 100 évaluations.

En conséquent, pour le problème présenté, on peut prétendre que la méthode hybride de gestion des contraintes (additionnant une pénalité constante de mort à une fonction de mérite adaptative proportionnelle à la distance entre la solution et le domaine réalisable) donne d'aussi bons résultats, au moins aussi rapidement, que les deux autres méthodes. De plus, elle permet d'éviter la génération de solutions irréalisables pour le problème test par l'algorithme d'optimisation.

Annexe C.

SOLUTIONS INITIALES POUR LE PROBLÈME DU DISQUE

Cette annexe présente les solutions initiales générées pour les 4 cas tests proposés au chapitre 7. Ces solutions initiales sont le résultat d'une optimisation utilisant les paramètres par défaut de l'AGENT et 50 000 évaluations. Ces optimisations ont été lancées à partir d'une population initiale aléatoire de disques utilisant un modèle paramétrique symétrique pour les solutions 1 à 17 et un modèle asymétrique pour les solutions 18 à 20 (voir détails dans le chapitre 7). Notez que l'échelle de la figure est de 1 : 2,54 cm.

SOLUTION 1

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
1	0	177 928x50	25000	1100	5000	8,758

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0015	x_{14}	0,0003
x_3	0,0508	x_9	0,3475	x_{15}	0,0025
x_4	0,1559	x_{10}	0,5124	x_{16}	0
x_5	0,0013	x_{11}	0,0050		
x_6	0,1069	x_{12}	0,1739		

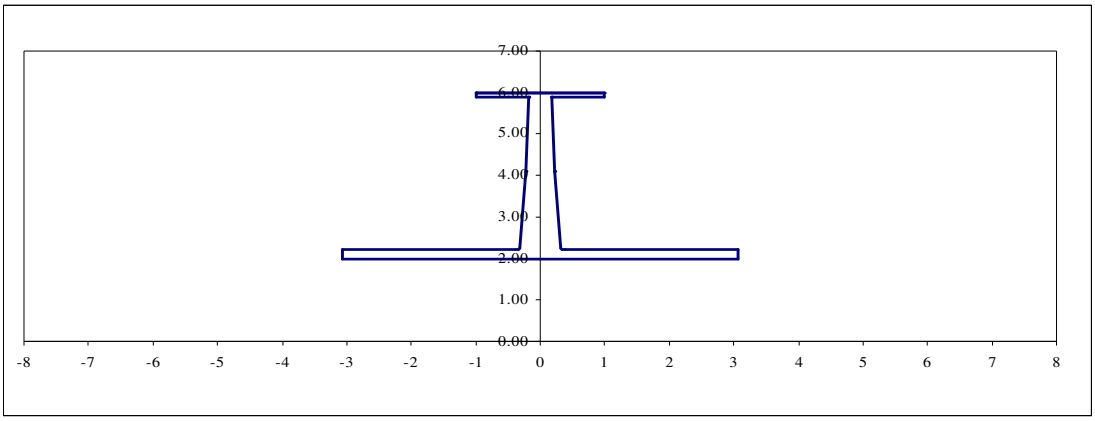


Figure C.1 : Solution initiale 1

SOLUTION 2

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
2	0	177 928x50	25000	1100	10000	9,995

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0001	x_{14}	0,0008
x_3	0,0508	x_9	0,2034	x_{15}	0,0025
x_4	0,1818	x_{10}	0,0000	x_{16}	0
x_5	0,0013	x_{11}	0,0049		
x_6	0,3029	x_{12}	0,1856		

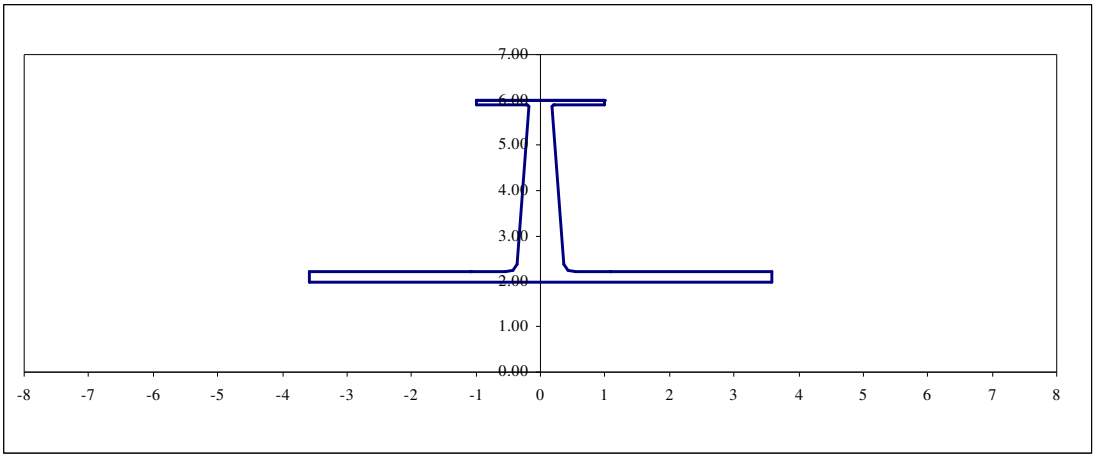


Figure C.2 : Solution initiale 2

SOLUTION 3

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
3	0	177 928x50	25000	1500	5000	10,148

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0022	x_{14}	0,0005
x_3	0,0508	x_9	0,4246	x_{15}	0,0025
x_4	0,1955	x_{10}	0,1314	x_{16}	0
x_5	0,0013	x_{11}	0,0021		
x_6	0,1564	x_{12}	0,1633		

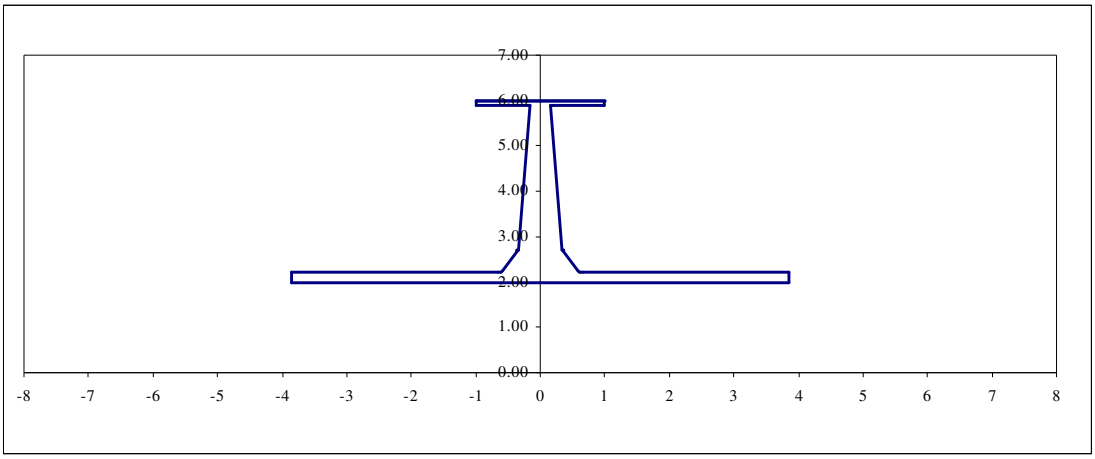


Figure C.3 : Solution initiale 3

SOLUTION 4

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
4	0	177 928x50	25000	1500	10000	11,741

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0063	x_{14}	0,0004
x_3	0,0508	x_9	0,6186	x_{15}	0,0025
x_4	0,1648	x_{10}	0,0020	x_{16}	0
x_5	0,0021	x_{11}	0,0014		
x_6	0,1793	x_{12}	0,1774		

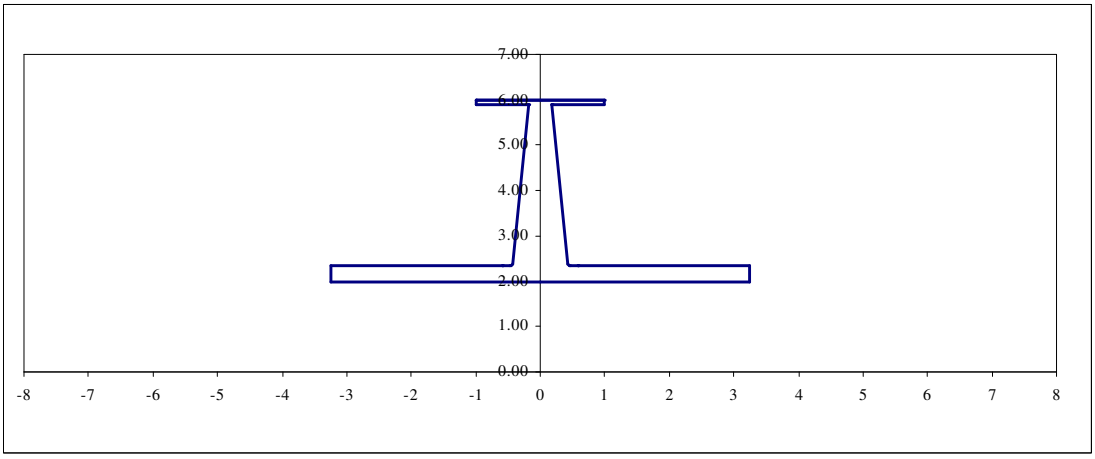


Figure C.4 : Solution initiale 4

SOLUTION 5

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
5	0	266 893x50	25000	1100	5000	13,011

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0038	x_{14}	0,0006
x_3	0,0508	x_9	0,5738	x_{15}	0,0025
x_4	0,1638	x_{10}	0,2587	x_{16}	0
x_5	0,0021	x_{11}	0,0002		
x_6	0,1536	x_{12}	0,2526		

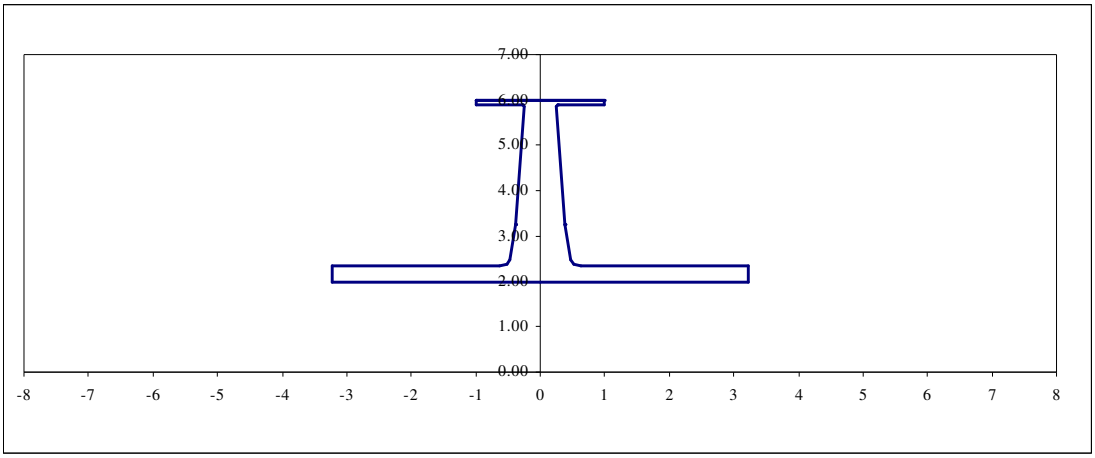


Figure C.5 : Solution initiale 5

SOLUTION 6

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
6	0	266 893x50	25000	1100	10000	13,362

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0034	x_{14}	0,0057
x_3	0,0508	x_9	0,0878	x_{15}	0,0025
x_4	0,2050	x_{10}	0,0000	x_{16}	0
x_5	0,0013	x_{11}	0,0074		
x_6	0,8024	x_{12}	0,2707		

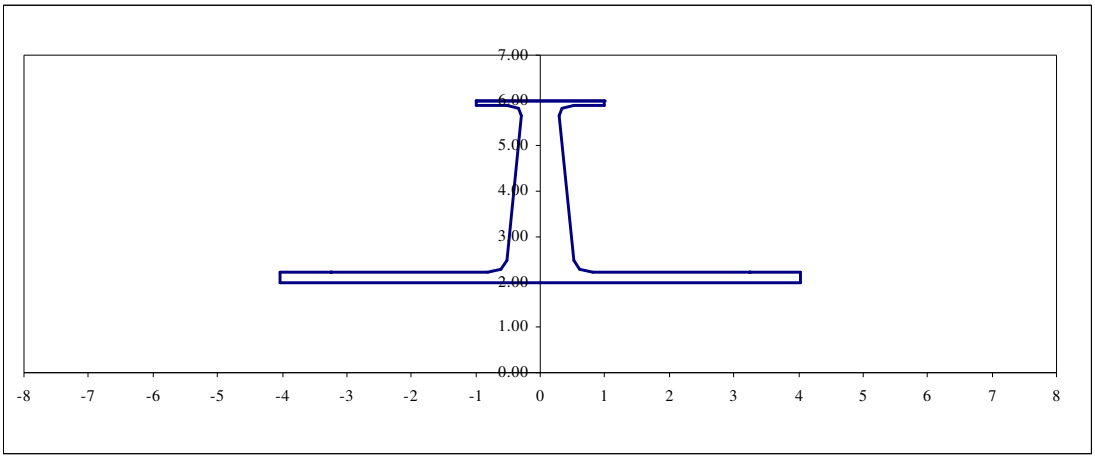


Figure C.6 : Solution initiale 6

SOLUTION 7

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
7	0	266 893x50	25000	1500	5000	14,560

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0031	x_{14}	0,0024
x_3	0,0508	x_9	0,3307	x_{15}	0,0025
x_4	0,3023	x_{10}	0,0000	x_{16}	0
x_5	0,0013	x_{11}	0,0003		
x_6	0,2180	x_{12}	0,2323		

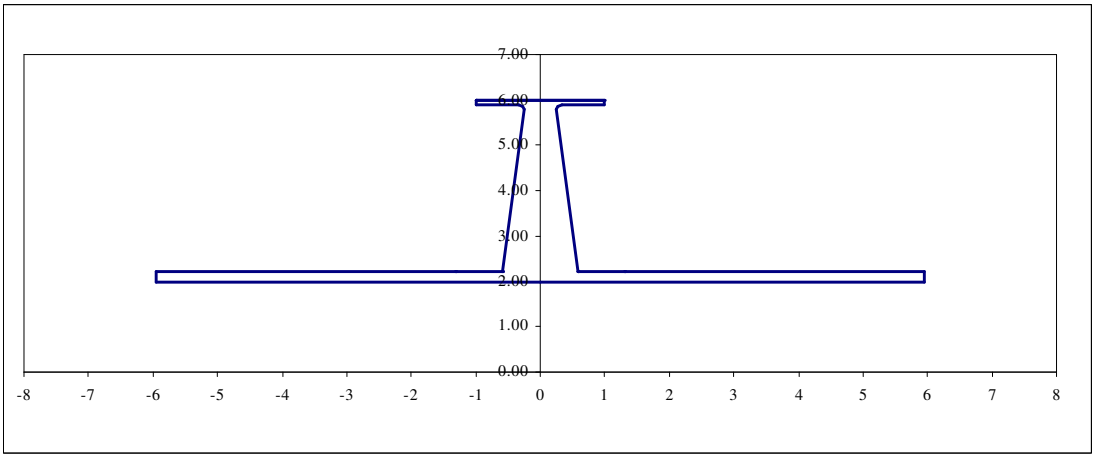


Figure C.7 : Solution initiale 7

SOLUTION 8

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
8	0	266 893x50	25000	1500	10000	14,840

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0076	x_{14}	0,0028
x_3	0,0508	x_9	0,1156	x_{15}	0,0025
x_4	0,2745	x_{10}	0,0000	x_{16}	0
x_5	0,0013	x_{11}	0,0004		
x_6	0,6285	x_{12}	0,2539		

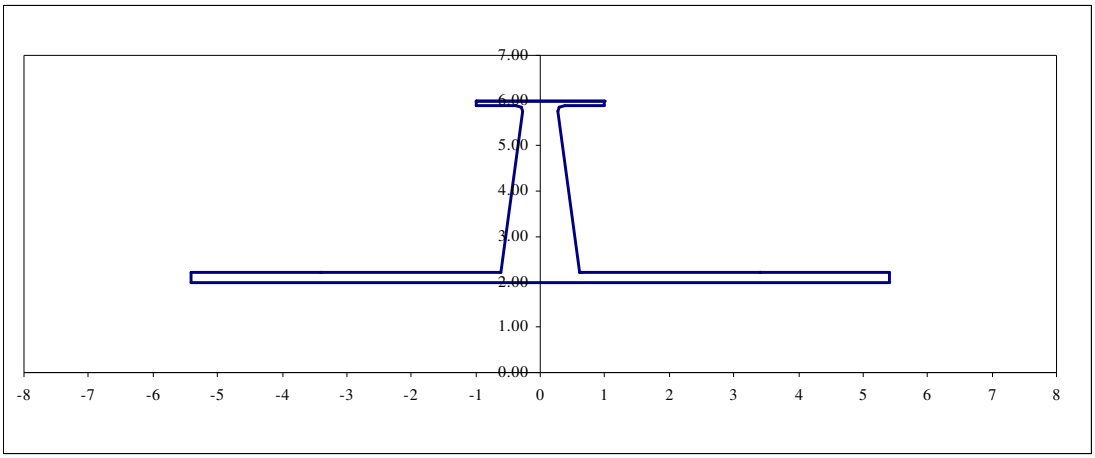


Figure C.8 : Solution initiale 8

SOLUTION 9

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
9	0	177 928x50	30000	1100	5000	10,928

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0009	x_{14}	0,0018
x_3	0,0508	x_9	0,0640	x_{15}	0,0025
x_4	0,2229	x_{10}	0,0000	x_{16}	0
x_5	0,0013	x_{11}	0,0040		
x_6	0,9211	x_{12}	0,1733		

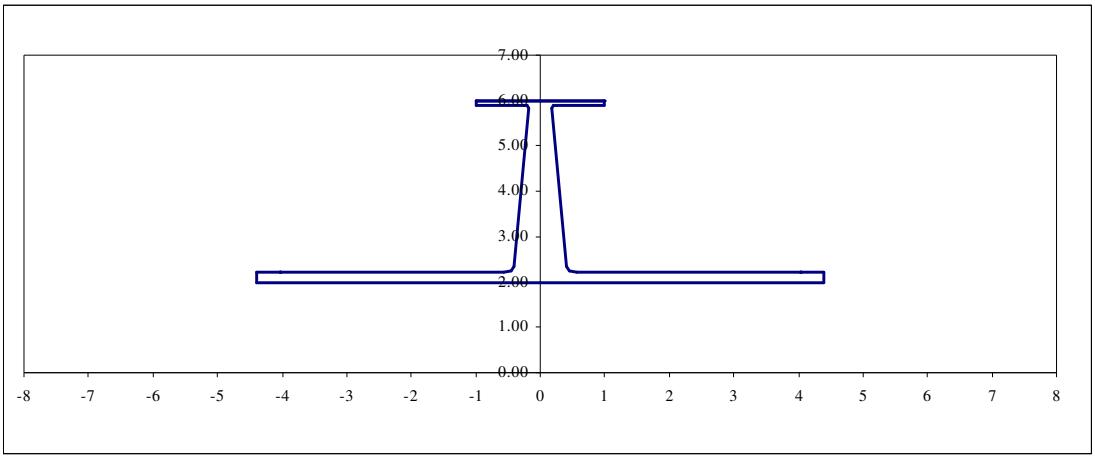


Figure C.9 : Solution initiale 9

SOLUTION 10

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
10	0	177 928x50	30000	1100	10000	12,272

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0058	x_{14}	0,0024
x_3	0,0508	x_9	0,6156	x_{15}	0,0025
x_4	0,2644	x_{10}	0,2979	x_{16}	0
x_5	0,0013	x_{11}	0,0032		
x_6	0,0953	x_{12}	0,1921		

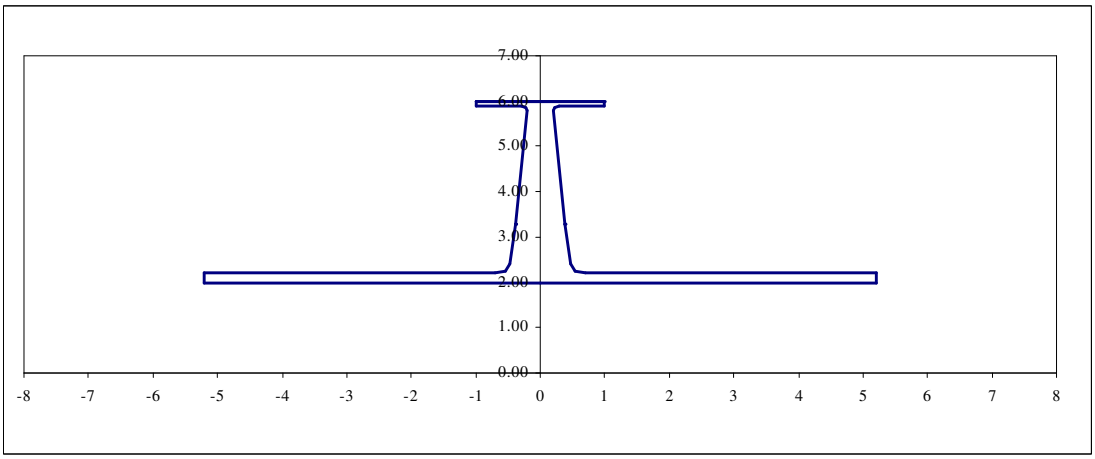


Figure C.10 : Solution initiale 10

SOLUTION 11

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
11	0	177 928x50	30000	1500	5000	12,392

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0003	x_{14}	0,0011
x_3	0,0508	x_9	0,2035	x_{15}	0,0025
x_4	0,3281	x_{10}	0,5944	x_{16}	0
x_5	0,0013	x_{11}	0,0058		
x_6	0,0765	x_{12}	0,1798		

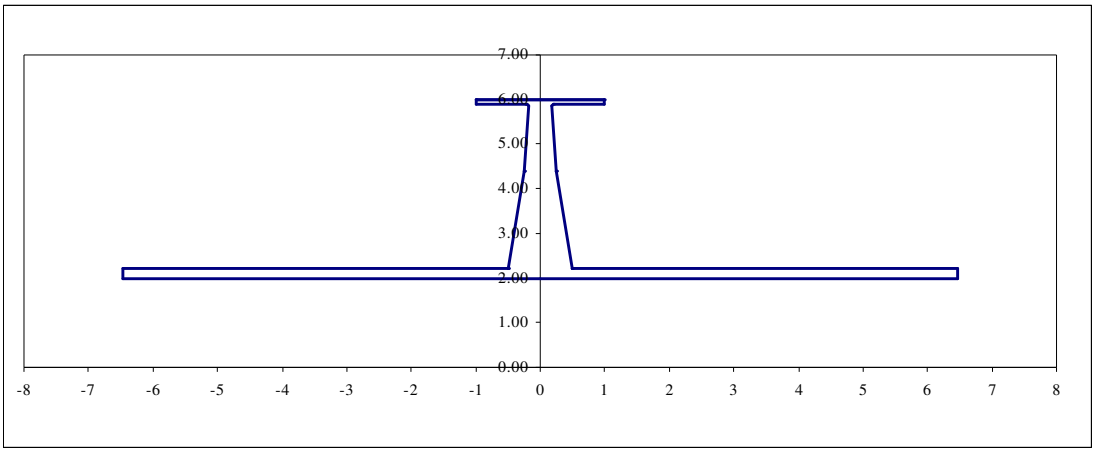


Figure C.11 : Solution initiale 11

SOLUTION 12

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
12	0	177 928x50	30000	1500	10000	15,052

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0006	x_{14}	0,0001
x_3	0,0508	x_9	0,1704	x_{15}	0,0025
x_4	0,3652	x_{10}	0,0000	x_{16}	0
x_5	0,0013	x_{11}	0,0018		
x_6	0,3830	x_{12}	0,1818		

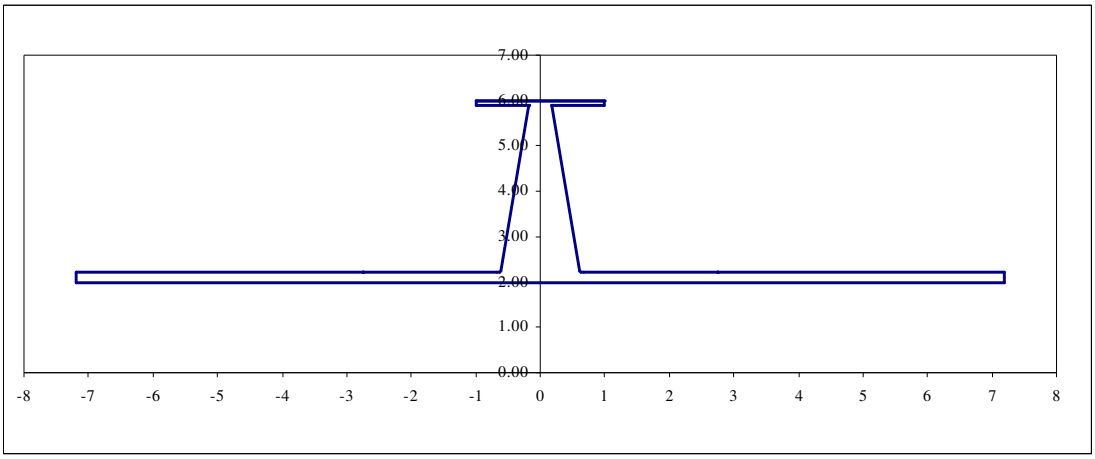


Figure C.12 : Solution initiale 12

SOLUTION 13

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
13	0	266 893x50	30000	1100	5000	15,838

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0058	x_{14}	0,0046
x_3	0,0508	x_9	0,3588	x_{15}	0,0025
x_4	0,2312	x_{10}	0,5469	x_{16}	0
x_5	0,0021	x_{11}	0,0049		
x_6	0,1282	x_{12}	0,2632		

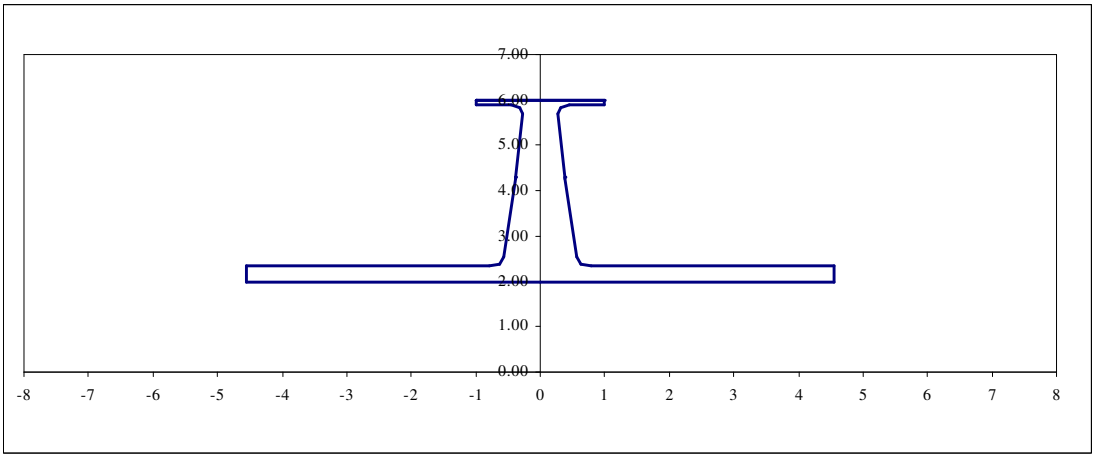


Figure C.13 : Solution initiale 13

SOLUTION 14

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
14	0	266 893x50	30000	1100	10000	17,491

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0004	x_{14}	0,0005
x_3	0,0508	x_9	0,4496	x_{15}	0,0025
x_4	0,3834	x_{10}	0,3349	x_{16}	0
x_5	0,0013	x_{11}	0,0044		
x_6	0,1088	x_{12}	0,2835		

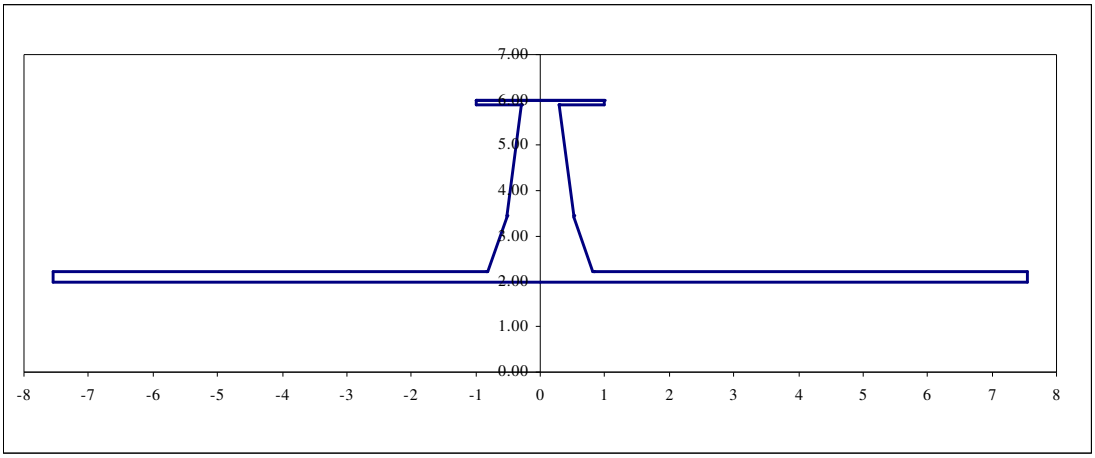


Figure C.14 : Solution initiale 14

SOLUTION 15

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
15	0	266 893x50	30000	1500	5000	17,776

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0004	x_{14}	0,0022
x_3	0,0508	x_9	0,3557	x_{15}	0,0025
x_4	0,2973	x_{10}	0,4133	x_{16}	0
x_5	0,0021	x_{11}	0,0029		
x_6	0,1255	x_{12}	0,2526		

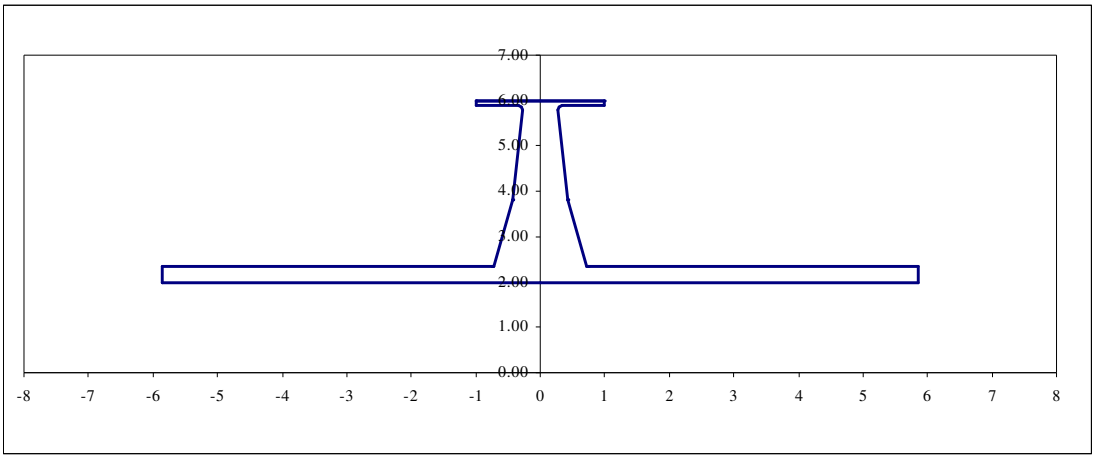


Figure C.15 : Solution initiale 15

SOLUTION 16

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
16	0	266 893x50	30000	1500	10000	20,344

Valeur des variables					
x_1	0,0508	x_7	0,0000	x_{13}	1,0000
x_2	0,1524	x_8	0,0018	x_{14}	0,0016
x_3	0,0508	x_9	0,2555	x_{15}	0,0025
x_4	0,3575	x_{10}	0,5292	x_{16}	0
x_5	0,0021	x_{11}	0,0031		
x_6	0,1169	x_{12}	0,2777		

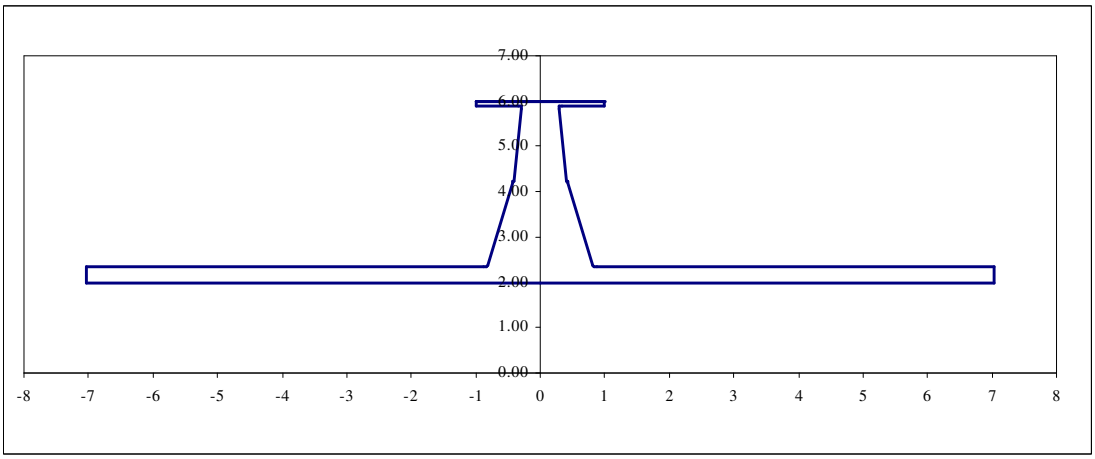


Figure C.16 : Solution initiale 16

SOLUTION 17

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
17	0	222 411x50	27500	1300	5000	12,082

Valeur des variables					
x_1	2,000	x_7	0,000	x_{13}	1,000
x_2	6,000	x_8	0,034	x_{14}	0,022
x_3	2,000	x_9	0,337	x_{15}	0,100
x_4	9,911	x_{10}	0,467	x_{16}	0
x_5	0,050	x_{11}	0,280		
x_6	0,102	x_{12}	0,215		

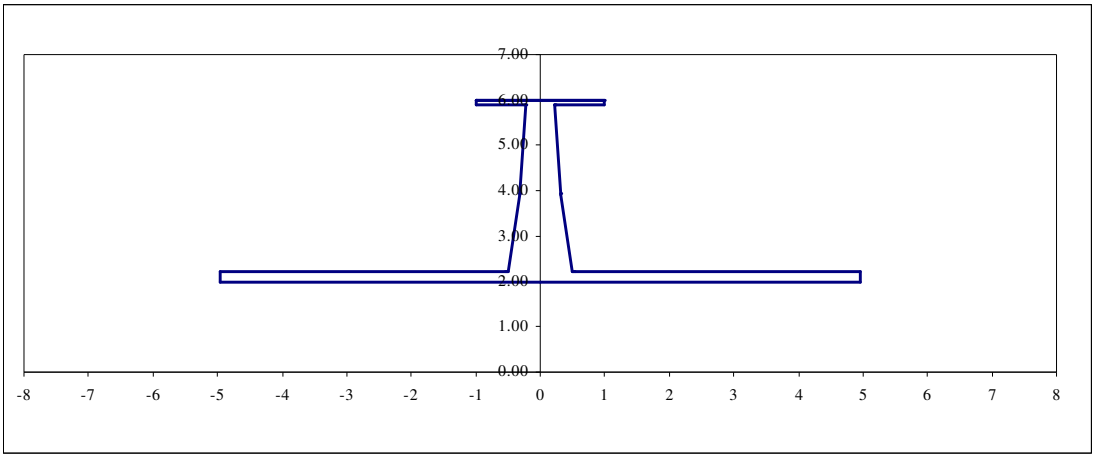


Figure C.17 : Solution initiale 17

SOLUTION 18

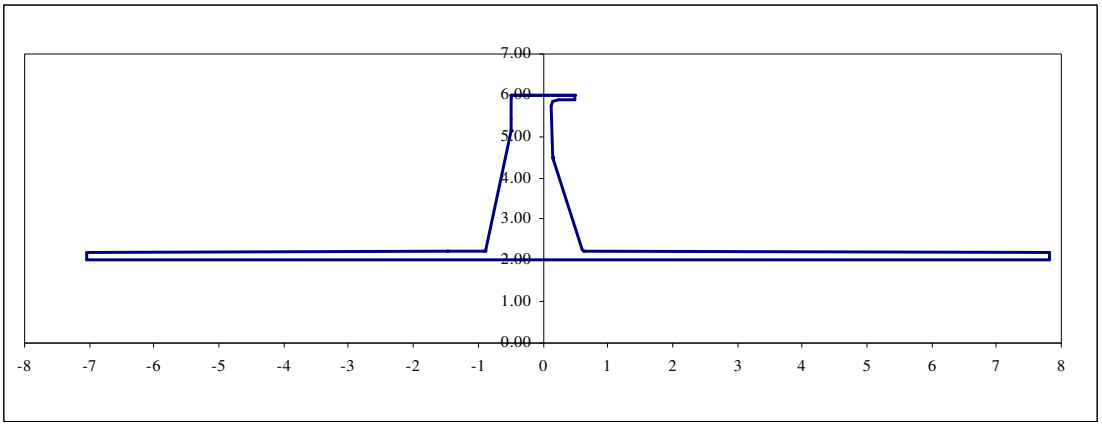
Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
18	0,00508	222 411x50	27500	1300	5000	16,888

Valeur des variables					
x_1	0,0508	x_{11}	0,0009	x_{21}	0,0364
x_2	0,1524	x_{12}	1,0000	x_{22}	0,6194
x_3	0,0254	x_{13}	0,9104	x_{23}	0,0009
x_4	0,1790	x_{14}	0,0013	x_{24}	0,2547
x_5	0,0013	x_{15}	0,0143	x_{25}	0,9900
x_6	0,2093	x_{16}	0,1987	x_{26}	0,0029
x_7	0,0100	x_{17}	0,0013	x_{27}	0,0025
x_8	0,0001	x_{18}	0,0787	x_{28}	0
x_9	0,3916	x_{19}	0,0100		
x_{10}	0,0000	x_{20}	0,0169		

Figure C.18 : Solution initiale 18

SOLUTION 19

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
19	0,00508	177 928x50	25000	1100	5000	12,261



Valeur des variables					
x_1	0,0508	x_{11}	0,0035	x_{21}	0,0788
x_2	0,1524	x_{12}	1,0000	x_{22}	0,0000
x_3	0,0254	x_{13}	0,4757	x_{23}	0,0049
x_4	0,0945	x_{14}	0,0041	x_{24}	0,0100
x_5	0,0013	x_{15}	0,0025	x_{25}	0,9900
x_6	0,2121	x_{16}	0,1534	x_{26}	0,0045
x_7	0,0100	x_{17}	0,0013	x_{27}	0,0025
x_8	0,0046	x_{18}	0,5114	x_{28}	1
x_9	0,0100	x_{19}	0,0100		
x_{10}	0,8474	x_{20}	0,1405		

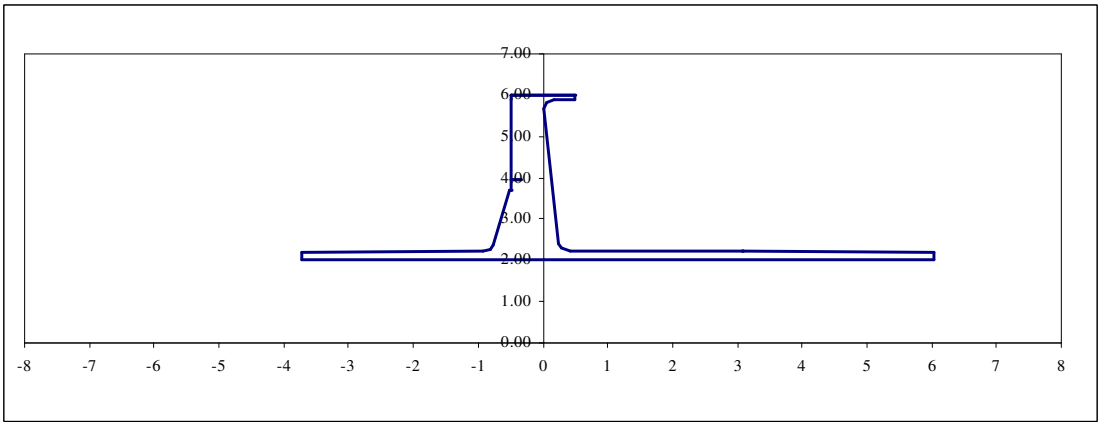


Figure C,19 : Solution initiale 19

SOLUTION 20

Solution	p_4 Excentricité	p_5 Traction	p_6 Vitesse	p_7 Température	p_8 Vie utile	$F(x)$ Poids
20	0,00508	266 893x50	30000	1500	10000	12,836

Valeur des variables					
x_1	0,0508	x_{11}	0,0020	x_{21}	0,2416
x_2	0,1524	x_{12}	1,0000	x_{22}	0,4949
x_3	0,0254	x_{13}	0,4593	x_{23}	0,0048
x_4	0,1103	x_{14}	0,0041	x_{24}	0,0100
x_5	0,0021	x_{15}	0,0123	x_{25}	0,9763
x_6	0,2012	x_{16}	0,0772	x_{26}	0,0049
x_7	0,0100	x_{17}	0,0016	x_{27}	0,0025
x_8	0,0024	x_{18}	0,1168	x_{28}	0
x_9	0,7116	x_{19}	0,0100		
x_{10}	0,0000	x_{20}	0,2000		

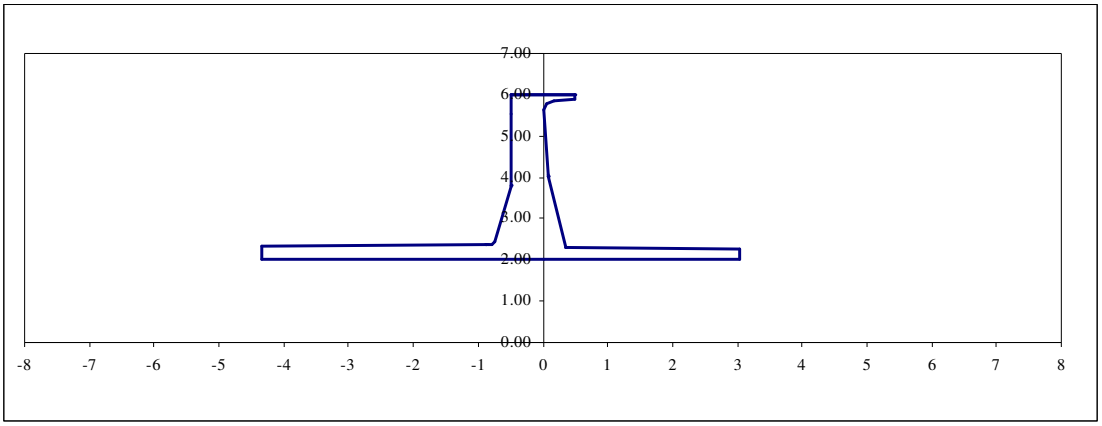


Figure C,20 : Solution initiale 20

Annexe D.
ALGORITHMES GÉNÉTIQUES

Ce chapitre décrit plus en détail les algorithmes génétiques (AG), qui constituent le cœur de ce volet de recherche. Les concepts généraux décrits dans ce chapitre servent de base pour le chapitre 5, où l'algorithme original développé dans ce volet (l'AGENT) sera défini plus en détail.

La section D.1 définit les AG de façon plus générale, alors que les sections D.2 à D.5 décrivent les principaux algorithmes ayant inspiré le développement de l'AGENT. La section D.6 présente ensuite certaines des principales heuristiques applicables aux AG, et la section D.7 termine le chapitre en abordant les méthodes d'application d'un modèle substitut à la fonction coût d'un AG retrouvées dans la littérature.

D.1 Définition d'un génome

Les algorithmes génétiques sont sans aucun doute le type d'AE le plus connu et le plus utilisé en ingénierie. Ils utilisent une série d'information (codons) regroupés en gènes pour former le génome de chaque individu d'une population.

Afin de mieux visualiser les concepts ci-dessus, supposons, par exemple, que l'on désire coder la solution à un problème disposant des variables a_1, a_2 et a_3. On pourrait définir une population de N_{pop} individus, chacun disposant d'un génotype tel qu'illustré à la figure D.1.

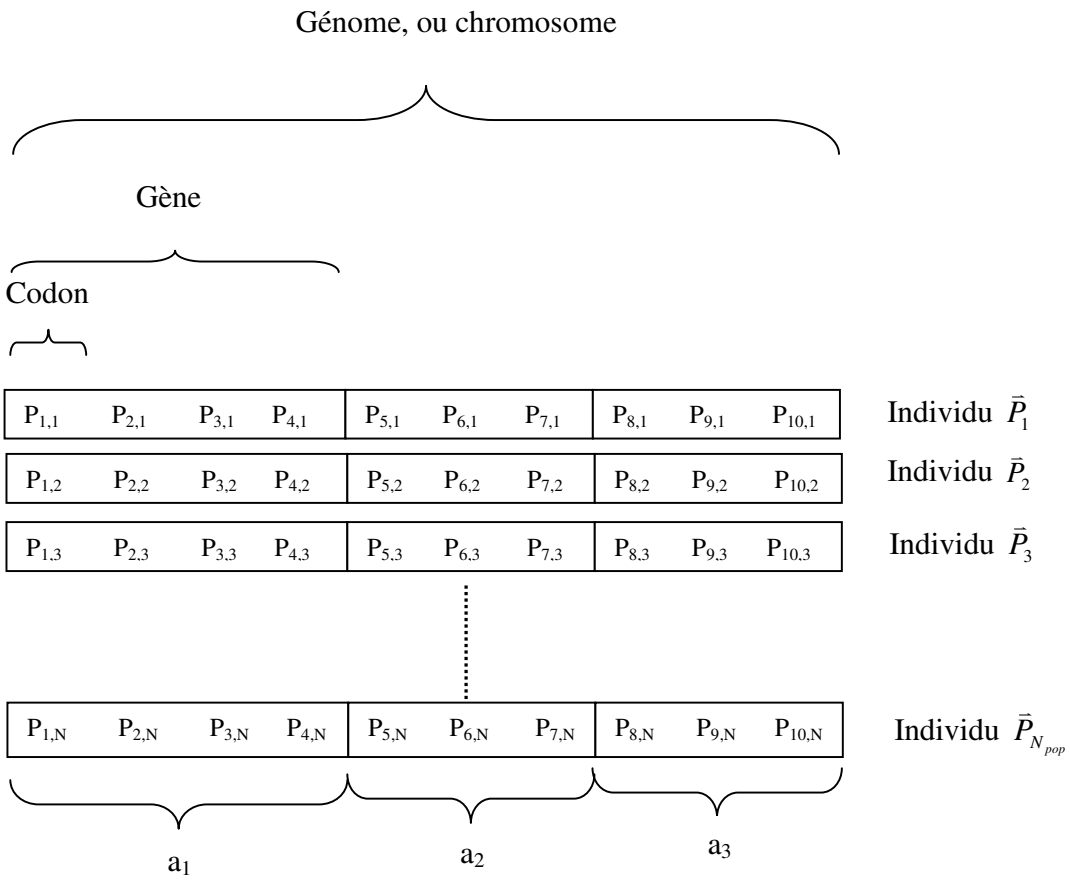


Figure D.1 : Définition du génome d'une population de N_{pop} individus à 3 variables.

Ici, chaque codon $P_{i,j}$ peut prendre des valeurs bien distinctes que l'on doit définir au début de l'algorithme. Les codons les plus fréquemment rencontrés dans la littérature sont les codons binaires et les codons à virgule flottante.

La valeur de chaque variable a_1, a_2 et a_3 peut être calculée pour un individu j particulier comme une fonction des codons lui correspondant. Ainsi, dans l'exemple de la figure D.1, pour un individu j, $a_1 = f(P_{1,j}, P_{2,j}, P_{3,j}, P_{4,j})$, $a_2 = f(P_{5,j}, P_{6,j}, P_{7,j})$ et $a_3 = f(P_{8,j}, P_{9,j}, P_{10,j})$.

Comme on le voit, une fois le génome d'une population bien défini, il est alors possible de convertir aisément les solutions à un problème donné en individus génétiques et vice versa. On peut alors concevoir un algorithme génétique, qui est en fait un algorithme évolutionnaire utilisant le concept de génomes.

Les sections suivantes présentent quatre AG très présents dans la littérature, soit l'algorithme génétique ordinaire (AGO), l'algorithme génétique à code réel (AGR), l'algorithme génétique à évolution différentielle (AGED) et l'algorithme génétique proportionnel (AGP2). Ces algorithmes sont dignes de mention pour deux raisons majeures. D'abord, la plupart des concepts ayant inspiré les mécanismes de l'AGENT sont tirés des concepts inhérents à ces algorithmes génétiques particuliers. Ensuite, puisque l'algorithme AGENT qui a été développé lors de cette recherche est un AG, ils lui serviront de base de comparaison pour mesurer son efficacité sur les problèmes dont la topologie est similaire à celle des problèmes de design structurel de pièces mécaniques.

D.2 AGO

Le premier des algorithmes que nous décrirons plus en détail ici est l'algorithme génétique ordinaire (AGO). Cet algorithme, qui utilise le codage binaire pour les variables du problème, est, d'un point de vue historique, le précurseur et l'inspiration de la plupart des algorithmes génétiques existants.

Le type de codage utilisé par l'AGO a pour intérêt de permettre de créer des opérateurs de croisement et de mutation simples. C'est également en utilisant ce type de codage que les premiers résultats de convergence théorique ont été obtenus.

Malgré les nombreux développements qui se sont produits depuis sa création, l'AGO présente toujours un intérêt majeur car il est utilisé comme base pour certaines

heuristiques et comme outil de comparaison pour les nouveaux algorithmes émergents (Lee & El-Sharkawi 2008).

La figure D.2 représente l'AGO dans son ensemble. Les mécanismes de cet algorithme sont décrits plus en détail dans cette section.

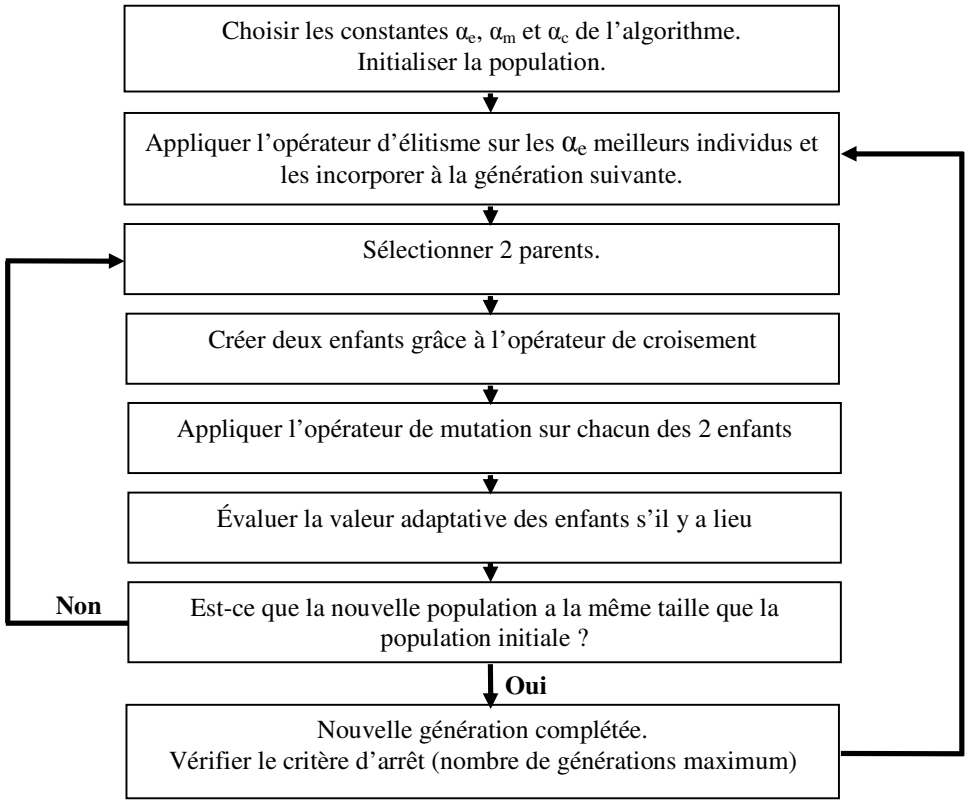


Figure D.2 : Représentation schématique d'un AGO

D.2.1 Codage binaire

Avec un codage binaire, chaque codon $P_{i,j}$ ne peut prendre que la valeur 0 ou la valeur 1. Chaque gêne définissant une variable en nombre binaire, on peut alors retrouver la valeur réelle d'un paramètre a_i de l'individu j grâce à l'expression D.1.

$$a_i = \frac{\sum_{k=R1}^{R2} 2^{(k-R1)} P_{k,j}}{U_{b,j} - L_{b,j}} \qquad \text{(D.1)}$$

où $P_{i,j}$ $\in \{0,1\}$, est le codon i du génome de l'individu j

$R1$ est l'emplacement du premier codon définissant a_i dans $P_{i,j}$

$R2$ est l'emplacement du dernier codon définissant a_i dans $P_{i,j}$

$U_{b,i}$ et $L_{b,i}$ sont respectivement les bornes supérieures et inférieures de a_i

D.2.2 Sélection des parents

Il existe plusieurs façons de sélectionner deux parents pour créer les enfants de la génération future (Goldberg, 1989).

L'une des méthodes les plus populaires est la sélection par roulette (Acan & Tekol, 2005), qui représente assez bien la réalité biologique en accordant plus de chances de se reproduire aux individus les mieux adaptés. Dans ce processus, chaque individu se voit acquérir une probabilité Pr_j de se reproduire définie par l'équation D.2.

$$Pr_j = \frac{F(P_j)}{\sum_{i=1}^{N_{pop}} F(P_i)} \qquad \text{(D.2)}$$

où $F(P_j)$ est la valeur adaptative de l'individu j de la population P.

N_{pop} est le nombre total d'individus parmi la population.

Si on les plaçait bouts à bouts, ces segments de probabilités Pr donneraient donc une probabilité totale de 1. L'idée derrière cette distribution est de reproduire le principe de tirage aléatoire utilisé dans les roulettes de casinos avec une structure linéaire. Pour choisir un parent, un nombre aléatoire Z entre 0 et 1 est donc généré, puis on détermine le vainqueur comme étant :

$$j = j \text{ minimum pour lequel } \; Z \leq \sum_{i=1}^{j} \Pr_j \qquad \text{(D.3)}$$
$$\text{avec } j \in \{1,2,3...\}$$

D.2.3 Opérateurs de recherche de l'AGO

Les opérateurs de recherche des AGO sont la mutation, le croisement et l'élitisme. Lors du passage d'une génération à l'autre, l'élitisme est d'abord appliqué, puis successivement des croisements et des mutations jusqu'à ce que la prochaine génération ait la même taille que la génération actuelle.

Élitisme

N'ayant pas réellement son équivalent biologique, l'élitisme consiste à conserver une copie intégrale des α_e individus les plus adaptés et de les incorporer directement dans la génération suivante. Cette pratique a pour but d'aider la convergence de l'algorithme en empêchant des opérations potentiellement destructives d'affecter les meilleurs individus.

Croisement

Le croisement est l'opérateur employé pour représenter l'accouplement entre plusieurs individus de la même espèce. Il consiste à sélectionner aléatoirement deux parents. Ensuite, il existe une probabilité α_c (généralement aux environs de 0,95) qu'un croisement ait lieu entre ces parents.

Si un croisement a lieu, on doit choisir aléatoirement deux points de section dans leur chromosome. Ensuite, la portion de chromosomes entre ces deux points de section est échangée d'un individu à l'autre. De cette façon, les portions de gènes prometteurs peuvent se répandre dans une population et faire éventuellement converger l'algorithme. La figure D.3 illustre un exemple d'application de croisement pour un OGA.

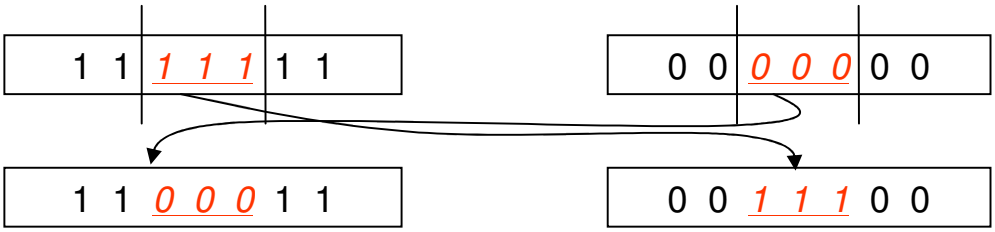


Figure D.3 : Exemple de croisement pour un OGA. Les codons 3 à 5 sont croisés.

Les deux enfants résultants subissent ensuite l'opérateur de mutation avant de se retrouver dans la génération suivante.

Mutation

La mutation est l'opérateur employé pour représenter l'évolution naturelle d'une espèce vers une autre espèce plus adaptée à son milieu. Elle consiste à modifier aléatoirement le génome d'un individu, en sélectionnant un des codons et en inversant sa valeur (voir figure D.4).

Soit α_m, le taux d'application de l'opérateur de mutations (généralement faible, environ 0,05). Pour chaque enfant généré à la sortie d'un croisement, l'algorithme dispose d'une probabilité α_m de lui appliquer l'opérateur de mutation. Si il a été modifié, l'individu résultant de cette étape doit recalculer sa valeur adaptative, et il se retrouve ensuite dans la génération suivante.

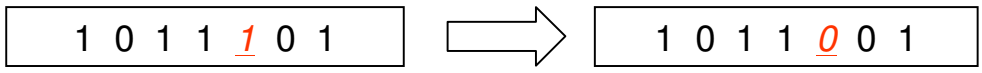


Figure D.4 : Exemple de mutation pour un OGA. Le 5ème codon est inversé.

D.3 AGR

L'algorithme génétique à valeurs réelles (AGR) suit le même schéma que l'AGO (voir figure D.2), mais utilise plutôt des valeurs réelles pour encoder ses variables. Ainsi, la valeur de chacune des variables est directement reliée à un codon du chromosome, selon l'équation D.4.

$$a_i = P_{i,j} \tag{D.4}$$

Les codons ont alors une signification davantage physique que dans le cas de l'AGO.

D.3.1 Opérateurs de recherche de l'AGR

Mutation

Dans le cas d'un AGR, la mutation n'est pas simplement l'inversion d'un bit sur le chromosome, il s'agit plutôt d'effectuer un pas déterminé de façon aléatoire dans la direction de chacune des variables.

Ainsi, on effectue généralement ici une mutation aléatoire selon une distribution de probabilité donnée. La distribution utilisée pour la perturbation est généralement Gaussienne (Hinterding, 1996), et correspond à un pas dont la grandeur est définie par la relation D.5.

$$\Delta a_i = \mathrm{p}_{\mathrm{moyen}} \sqrt{2} Erf^{-1}\left(2Z_i - 1\right) \tag{D.5}$$

où Δa_i est le pas résultant pour la variable *i* de la mutation.

$\mathrm{p}_{\mathrm{moyen}}$ est le pas moyen de mutation, tel que défini par l'utilisateur.

Z_i est une variable aléatoire variant de 0 à 1 pour chaque variable *i*.

L'un des problèmes de ce type de mutation est que l'on doive choisir p_{moyen} adéquatement pour le problème en cours si l'on désire obtenir de bons résultats. Nous verrons à la section 5.4.1 comment l'AGENT remédie à ce problème.

Croisement

Dans l'AGR, l'opérateur de croisement ne consiste pas en un échange de codons d'une solution à une autre. En effet, pour un problème ayant des variables continues, un tel échange n'aurait pas véritablement de sens physique.

L'opérateur de croisement utilisé est généralement différentiel, comme celui de l'équation D.6.

$$\begin{aligned} \vec{C}_1 &= (1-\theta)\vec{P}_1 + (1-\theta)\vec{P}_2 \\ \vec{C}_2 &= (1-\theta)\vec{P}_2 + (1-\theta)\vec{P}_1 \end{aligned} \qquad \text{(D.6)}$$

où $\vec{P}_1$ et $\vec{P}_2$ sont les deux vecteurs parents

θ est un nombre aléatoire entre 0 et 1, différent pour change enfant

$\vec{C}_1$ et $\vec{C}_2$ sont les deux enfants résultants

Plusieurs des nouveaux algorithmes génétiques émergents sont des AGR disposant de nouveaux opérateurs de croisement, réputés procurer de meilleurs résultats que l'opérateur décrit ci-dessus pour certains genres de problèmes.

Parmi les croisements les plus utilisés, on retrouve le croisement unimodal à distribution normale (nommé UNDX pour unimodal normal distribution crossover), qui, au lieu d'utiliser uniquement deux parents, en utilise plutôt plusieurs et produit des descendants selon une distribution normale autour du centre de masse de ces parents (Ono & Kobayashi, 1998 ; Deb, Joshi & Anand, 2002). Cet opérateur étant réputé pour donner de piètres résultats lorsque l'optimum se situe près des bornes des variables, Tsutsui

(2000) et Tsutsui & GoldBerg (2001) proposent deux méthodes d'application de l'opérateur, nommées BEM pour «Boundary Extension by Mirroring » et BES pour «Boundary Extension with extended Selection », corrigeant partiellement le problème.

Un autre opérateur de croisement très utilisé est l'opérateur de croisement simplexe (SPX), qui produit des enfants selon une distribution uniforme de probabilité conscrite dans un espace dont les bornes sont décrites par les parents (Higuchi, Tsutsui & Yamamura, 2000).

Deb, Joshi & Anand (2002) proposent, quant à eux, proposent un opérateur de croisement dont la distribution de probabilité est centrée autour des parents, nommé PCX pour « parent centric recombination operator », qui est inspiré d'un autre opérateur populaire centré sur les parents nommé BLX pour « blend crossover ». La figure D.5, inspirée d'une figure de cet article, démontre d'ailleurs la différence de distribution entre les enfants produits par les croisements UNDX, SPX ou BLX pour un choix de trois parents formant le même triangle quelconque.

Figure D.5 : Comparaison entre différents opérateurs de croisement pour les AGR.

Pour ces opérateurs à parents multiples, Ting & Büning (2003), sans proposer de changement direct aux opérateurs de croisements, proposent plutôt une méthode

nommée TMPGA (pour « Tabou Multi-Parents Genetic Algorithm », qui sélectionne les parents en s'inspirant de la recherche taboue pour prévenir les parents trop similaires d'être choisis et de produire en quelque sorte ce qu'ils appellent de l'inceste numérique.

De façon générale, les AGR sont réputés avoir donné de meilleurs résultats dans le cas de problèmes disposant de variables continues (Antonisse, 1989 ; Menczer & Parisi, 1992), et sont donc en principe de meilleurs candidats que l'AGO pour ce projet de recherche.

Dans cette thèse, l'opérateur de croisement régulier des AGR (équation D.6) sera utilisé pour produire les résultats préliminaires ayant mené à la construction de l'AGENT (voir section 6.1). Par contre, les performances de l'AGENT seront comparées à celles des AGR utilisant respectivement les opérateurs UNDX, SPX et BLX et les méthodes BES, BEM à la section 6.3 pour divers problèmes de nature académique dont les résultats seront tirés des articles correspondants.

D.4 AGED

Un des algorithmes qui gagne de plus en plus en popularité est l'algorithme génétique à évolution différentielle (AGED). Créé par Storn & Price (1995), cet algorithme a depuis été testé dans de nombreux cas sur des problèmes réels avec succès (Ursem & Vadstrup, 2003).

Tel que l'AGR, cet algorithme encode les variables à l'aide de nombres réels. Toutefois, au lieu d'utiliser des opérateurs aléatoires pour générer de nouveaux candidats, l'AGED utilise une proportion des différences vectorielles entre deux solutions pour générer la perturbation menant à l'individu suivant. De cette façon, le pas et la direction des perturbations suivent davantage la topologie de la fonction et l'algorithme est guidé plus rapidement vers un optimum éventuel.

La figure D.6 représente schématiquement l'algorithme AGED.

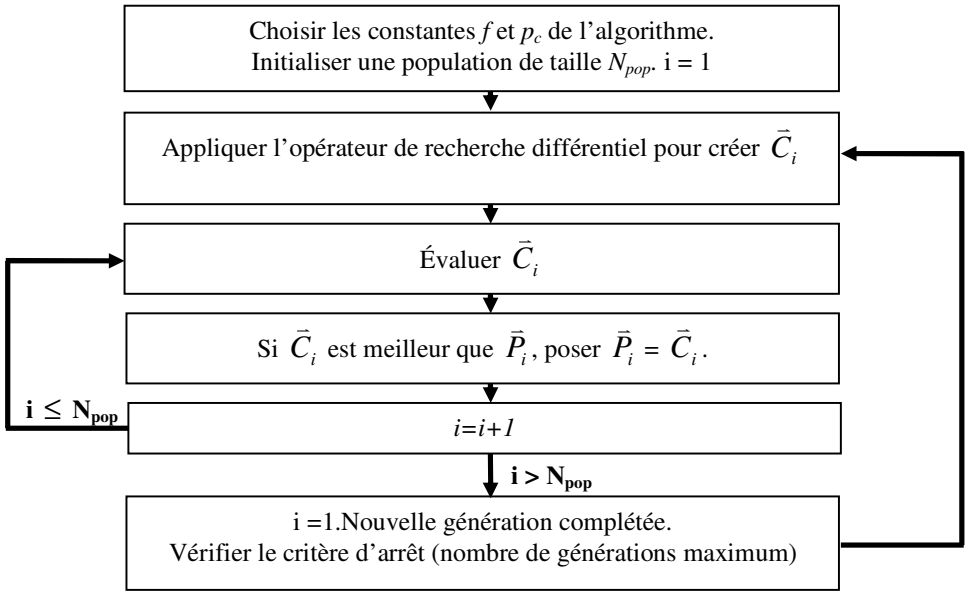


Figure D.6 : Représentation schématique d'un AGED

Ainsi, chaque opération améliore l'un des individus ou est immédiatement rejetée par le procédé. De cette façon, on s'assure que l'algorithme convergera toujours vers une meilleure solution.

D.4.1 Opérateur de recherche différentiel

L'opérateur de recherche différentiel nécessite 2 paramètres f et p_c à modifier manuellement. Appliqué à un individu $\vec{P}_i$, l'opérateur suit les étapes suivantes :

1. Sélectionner 3 parents différents $\vec{P}_1$, $\vec{P}_2$ et $\vec{P}_3$ au hasard, tous différents de P_i.
2. Générer $\vec{C}_1 = \vec{P}_1 + f(\vec{P}_2 - \vec{P}_3)$

3. Pour chaque variable de $\vec{C}_1$, tester si Z < p_c, où Z est un nombre de 0 à 1 généré aléatoirement à chaque fois. Si Z < p_c , cette variable sera modifiée.
4. Pour toutes les variables *j* à modifier, $C_{1,j} = C_{1,j}$
5. Pour toutes les autres variables, $C_{1,j} = P_{i,j}$

La solution résultante $\vec{C}_1$ se situe donc à l'une des extrémités de l'hypercube défini entre les extrémités $\vec{P}_i$ et $\vec{C}_1$. De cette façon, l'algorithme est à même de modifier plus d'une variable à la fois et donc d'utiliser l'interdépendance entre certaines variables si nécessaire pour quitter l'optimum local, sans toutefois devoir nécessairement les utiliser toutes, de façon à faciliter la tâche de l'algorithme pour les problèmes ayant des variables linéairement séparables.

Dans un article récent, Noman et Iba (2008) proposent une adaptation de l'AGED que nommée AGED-RL pour « algorithme génétique à évolution différentielle avec recherche locale » (ou LSDE pour « Local Search Differential Evolution »). Comme son nom l'indique, cet algorithme possède la particularité d'utiliser une recherche locale pour déterminer le pas idéal à appliquer lors des croisements de l'AGED. Bien que la présente thèse utilisera l'AGED tel quel pour produire les résultats préliminaires ayant amené la construction de l'AGENT, les performances de l'AGENT seront tout de même comparées à celles de l'AGED-RL pour certains problèmes de nature académiques dont les résultats sont donnés par Noman et Iba (2008).

D.5 AGP2

L'un des algorithmes plutôt récents apparaissant dans la littérature est l'algorithme génétique proportionnel AGP2 (Wu & Garibay, 2002). Le but de l'AGP2 est de pousser à l'extrême l'indépendance de la position des variables dans le génome.

En effet, en utilisant un opérateur de croisement régulier sur un AGO, les codons qui se croisent ont tendance à être ceux des variables situées l'une à coté de l'autre sur le chromosome, ce qui fait que les directions de recherche sont limitées aux variables qui se côtoient. Par exemple, si une valeur combinée des gènes de a_2 et a_4 est prometteuse dans le génotype [a_1 a_2 a_3 a_4], la seule façon de modifier a_2 et a_4 simultanément par croisement régulier est d'inclure aussi un gène parasite a_3. L'AGP2 propose donc, plutôt que de coder dans un chromosome fixé les diverses variables, d'y coder des petits incréments/décréments qui ont un effet indépendamment de leur position dans le chromosome. En quelque sorte, l'AGP2 représente mieux la réalité biologique et le véritable ADN que les autres algorithmes présentés dans ce chapitre.

À chaque variable est assignée une étiquette positive et une étiquette négative. La valeur finale de la variable est alors basée sur le nombre de fois où l'étiquette apparaît dans le gène plutôt que sur la valeur de cette étiquette.

La valeur adimensionnelle du paramètre a_j est donnée par

$$x_j = \frac{P_{char,j}}{P_{char,j} + N_{char,j}} \tag{D.7}$$

où $P_{char,j}$ est le nombre d'étiquettes positives représentant le paramètre
$N_{char,j}$ est le nombre d'étiquettes négatives représentant le paramètre

Par exemple, pour deux variables x_1 et x_2 si les lettres minuscules (x_1 et x_2) représentent une valeur positive d'étiquette et que les lettres majuscules (X_1 et X_2) sont des étiquettes négatives, le chromosome [x_1 x_1 x_1 x_2 x_2 x_1 X_1] donne x_1 = 4/5 et x_2 = 2/2.

L'opérateur de croisement de cet algorithme est le même que dans le cas d'un AGO. Pour la mutation, il s'agit plutôt de remplacer l'une des étiquettes du chromosome aléatoirement par une autre.

Cette approche dispose d'une caractéristique qui semble très intéressante à première vue. Lors des croisements, les combinaisons de gènes ayant contribué à l'amélioration des individus vont se retrouver tout naturellement côte à côte. L'algorithme capture donc en quelque sorte les interdépendances entre les variables dans son encodage et augmente les chances que les croisements bénéfiques se produisent à nouveau.

D.6 Heuristiques

Puisqu'un algorithme génétique n'est à la base ni plus ni moins qu'un algorithme de recherche aléatoire, plusieurs chercheurs ont tenté de créer des règles leur permettant de converger plus aisément ou plus rapidement vers un optimum. Ces règles, dont la pertinence est parfois difficile à prouver mathématiquement, sont appelées des heuristiques et n'en demeurent pas moins efficaces lorsqu'elles sont appliquées sur un type de problème bien particulier.

Il existe énormément d'heuristiques applicables aux algorithmes génétiques. Cette section décrit trois de ces heuristiques qui ont été testées sur les AG dans le cadre de cette recherche. Une quatrième heuristique (les noyaux territoriaux) a été développée dans le cadre de cette recherche et sera présentée avec l'algorithme AGENT dans le chapitre 5 de ce document.

D.6.1 Espaces de croyances (algorithme culturel)

Une façon attrayante de représenter la connaissance du domaine dans les algorithmes génétiques est d'introduire un espace de croyances pour la population, qui guide son cheminement (voir figure D.7).

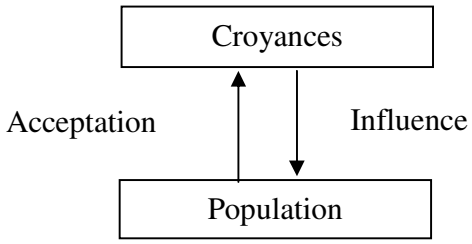


Figure D.7 : Influence d'un espace de croyances sur les individus d'une population

Les croyances sont en fait des opérations, ou des segments d'individus, considérés prometteurs, qui sont utilisés pour se croiser avec la population lors de la génération de nouveaux individus. De cette façon, lorsqu'une modification sur un individu s'avère fructueuse, on l'incorpore dans les croyances et elle sera disponible pour être utilisée à nouveau lors d'une évolution future. Ce type d'algorithme génétique est aussi appelé algorithme culturel, et est réputé pour donner des résultats compétitifs avec les autres algorithmes génétiques pour plusieurs applications (Reynolds, 1999 ; Franklin & Bergerman, 2000 ; Becerra & Carlos, 2005).

Création de l'espace de croyances

Dans les faits, conserver un espace de croyance revient un peu à conserver de l'information sur la topologie de la fonction. Lorsqu'une mutation ou un croisement donne une amélioration considérable de la valeur adaptative d'un individu, on peut alors entreposer l'effet de l'opération sur l'individu dans l'espace de croyance, avec une mesure de l'amélioration obtenue par son utilisation. Cette mesure d'amélioration de la

fonction coût $\Delta F(\vec{x})$ correspond plus ou moins au gradient de la fonction suite à cette opération.

Ensuite, chaque fois qu'une croyance est utilisée, sa mesure de $\Delta F(\vec{x})$ est ajustée pour tenir compte du nouveau résultat obtenu selon une équation ressemblant à la suivante :

$$\Delta F(\vec{x}) = (1 - f_c)\Delta F^{k}(\vec{x}) + f_c \Delta F^{k-1}(\vec{x}) \qquad \text{(D.8)}$$

où $\Delta F(\vec{x})$ est la force de la croyance.

$\Delta F^{k}(\vec{x})$ est le gradient obtenu par la plus récente application de cette croyance.

$\Delta F^{k-1}(\vec{x})$ est la force de la croyance avant son application la plus récente.

f_c mesure la force de rétention des croyances, définie par l'utilisateur (0,2 par défaut).

Après chaque itération, on ne conserve qu'un nombre prédéfini (souvent égal à la taille de population) de croyances. Ainsi, toutes celles dont la mesure d'amélioration de $\Delta F(\vec{x})$ est faible par rapport aux autres sont détruites. On permet alors aux croyances les plus prometteuses de subsister et on efface celles qui ne donnent plus de bons résultats, afin de bien cerner la topologie locale aux alentours de la population.

Application de l'opérateur de croyances

Chaque fois que l'on choisit un opérateur de recherche pour les parents, on introduit une probabilité α_{ec} (généralement aux environs de 0,4) d'utiliser plutôt l'opérateur de recherche de l'espace de croyances au lieu des opérateurs de recherche réguliers de cet algorithme. Cet opérateur choisit alors une croyance au hasard parmi l'ensemble, et l'applique sur le parent en question afin d'obtenir son enfant.

La figure D.8 illustre le schéma d'un algorithme génétique auquel on a incorporé un espace de croyances :

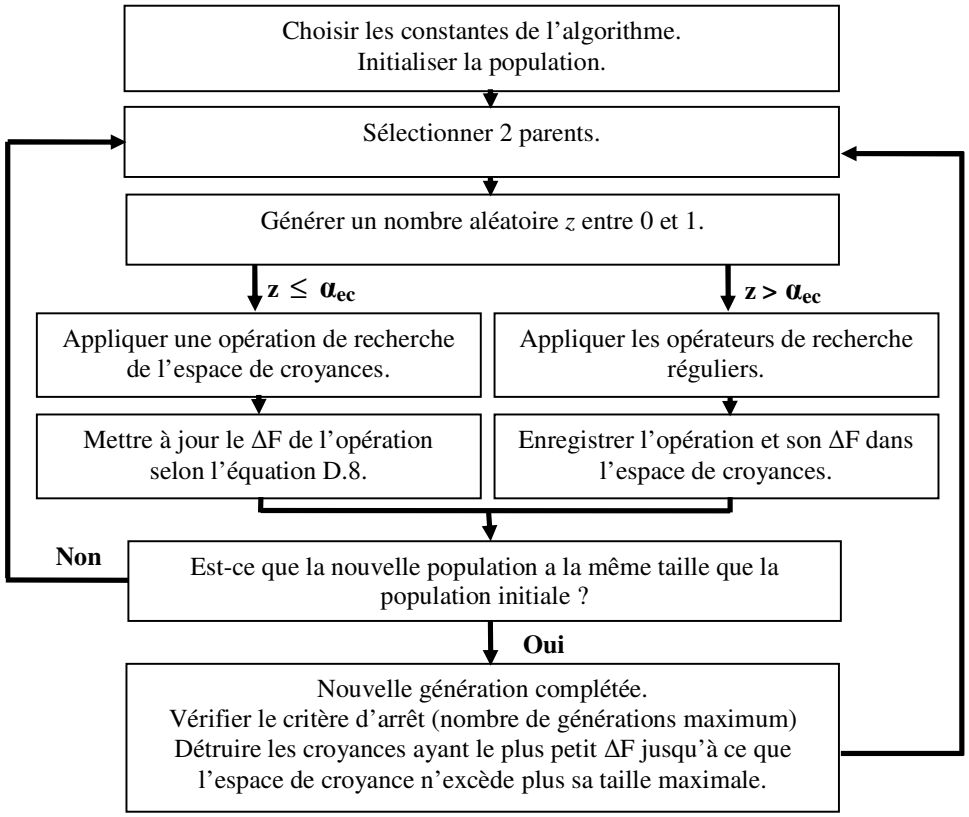


Figure D.8 : Représentation schématique d'un algorithme culturel

À première vue, cette approche semble très intéressante en combinaison avec un algorithme de type AGP2 en ce sens que, près de la convergence, elle permettrait de capturer de l'information sur les gradients de la fonction coût lors de l'application des opérateurs de croisement. Par exemple, si ajouter les étiquettes x_1x_2 dans le génome accroit considérablement la valeur adaptative, on pourra entreposer x_1x_2 dans l'espace de croyance et réutiliser cette opération plus tard pour continuer à faire progresser les variables dans la même direction.

D.6.2 Îlots (algorithme insulaire)

Cette heuristique a été développée suite à la constatation que plusieurs AG fonctionnant en parallèle donnaient souvent de meilleurs résultats qu'un seul AG disposant d'une plus grande population (Cantu-Pax, 1998).

En effet, comme dans le cas de tous les algorithmes disposant d'un biais vers les meilleures solutions, les individus des AG sont susceptibles de converger vers un optimum local à proximité des solutions actuelles. L'idée derrière l'heuristique des îlots est donc de protéger certains des individus de l'algorithme de ce centre d'attraction, afin de permettre à l'AG de continuer d'explorer globalement, même lorsque certains autres individus sont en phase plus avancée de convergence.

Pour ce faire, l'heuristique consiste principalement à séparer les individus de la population totale sur plusieurs îlots indépendants. Lors de chaque itération, chaque individu de chaque îlot ne subit d'interactions et de croisement qu'avec les individus partageant son îlot. Pour ainsi dire, l'AG traite alors le problème comme s'il était constitué de plusieurs AG en parallèle.

Toutefois, lors du traitement de solutions en parallèle, une fois que chaque îlot est en phase de convergence plus avancée, plusieurs évaluations de fonction peuvent être perdues pour développer des solutions qui sont assurément moins prometteuses que la meilleure solution actuellement connue. Afin de réduire cet effet, après un certain nombre fixé d'itérations, on permet à quelques-uns des individus de chaque îlot de migrer vers les autres îlots où ils pourront répandre leurs gênes si ceux-ci sont meilleurs que ceux présents à leur destination. Ceci permettra donc à long terme la convergence de l'algorithme en entier vers le même optimum.

Malgré les résultats prometteurs des AG utilisant des îlots, l'un des problèmes reste qu'il est nécessaire d'ajuster deux paramètres pour les faire fonctionner. Ces paramètres sont les suivants :

- I_M : Intervalle entre les migrations (en nombre de générations).
- N_M : Nombre d'individus de chaque îlot participant à la migration.

Cette heuristique est utilisée dans l'algorithme AGENT. L'utilisation de ces paramètres y est donc discutée plus en détail, à la section 5.6. La figure D.9 illustre le schéma d'un AG utilisant des îlots.

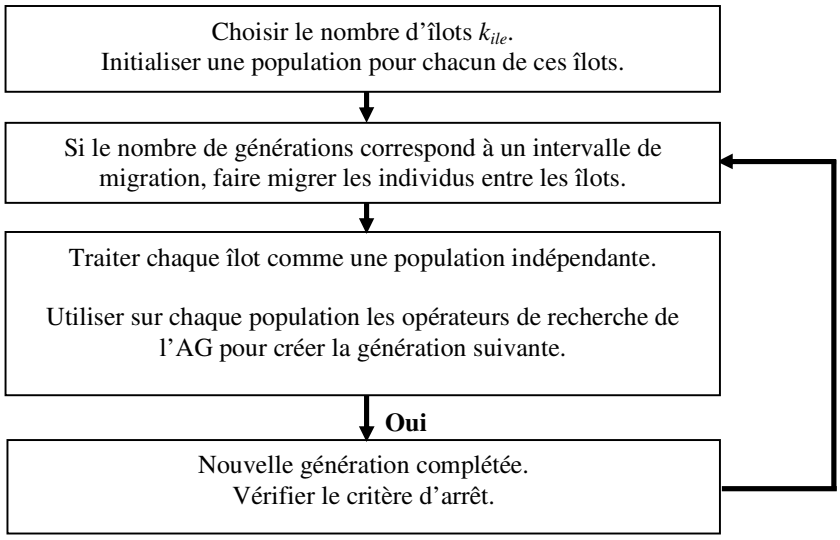


Figure D.9 : Représentation schématique d'un algorithme insulaire

D.6.3 Taux d'application variable des opérateurs

En général, chaque parent dispose d'une certaine probabilité de se voir appliquer chaque opérateur de recherche pour produire sa descendance.

Jusqu'à maintenant, le taux d'application des divers opérateurs est très souvent fixé par l'utilisateur selon son expérience et par la méthode des essais et erreurs. Toutefois, De Jong (1975), Grefenstette (1986), Goldberg (1989) et Zhang, Chung, Lo & Hu (2005) démontrent que le choix du taux d'application de chacun des opérateurs est très important car il peut affecter de façon critique le comportement et la performance de l'algorithme. Ces articles proposent certains guides généralisés pour choisir des taux adéquats, même que Grefenstette (1986) propose un second niveau d'algorithme génétique qui sert uniquement à sélectionner les meilleurs taux pour les opérateurs. Bien que cette méthode donne de bons résultats à la sortie de l'algorithme, elle est relativement dispendieuse en temps de calcul et ne sera pas utilisée ici.

Fondamentalement, un algorithme génétique régulier utilise majoritairement deux opérateurs pour obtenir deux qualités d'optimisation différentes.

Premièrement, l'opérateur de mutation est un opérateur aléatoire qui confère une certaine créativité à l'algorithme. C'est principalement cet opérateur qui permet de passer outre les optimums locaux et donc qui permet de meilleures solutions novatrices au système actuel. Toutefois, un algorithme génétique basé uniquement sur la mutation serait en quelque sorte guidé uniquement par le hasard, et donc très lent à converger.

L'opérateur de croisement, quant à lui, réutilise les portions de gênes donnant de bons résultats afin de faire converger l'algorithme. Cet opérateur confère donc la qualité de rapidité de convergence à l'algorithme.

Srinivas & Patnaik (1994) résument assez bien dans leur article les idées sous-jacentes aux choix des probabilités de croisement et de mutation dans les algorithmes, proposant même un algorithme génétique adaptif dans lequel ces probabilités dépendent de la valeur adaptative de la solution.

Sachant que l'opérateur de croisement est davantage lié à la propriété de convergence, et que l'opérateur de mutation permet davantage d'explorer le domaine, Zhang, Chung, Lo & Hu (2005) proposent une méthode basée sur la logique floue pour déterminer les valeurs des taux de croisement et de mutation selon l'état de l'optimisation. Lorsque l'optimisation est à son état initial, le taux de mutation est posé élevé alors que le taux de croisement est posé plus bas. À mesure que l'algorithme converge, le taux de mutation décroit alors que le taux de croisement croit. Ceci permet d'explorer plusieurs avenues différentes du problème initialement, tout en convergeant rapidement lorsqu'une solution valable a été déterminée.

Malgré l'aspect prometteur de ces propositions, la grande difficulté dans chaque cas est de trouver un estimateur convenable indiquant le niveau de convergence de la population actuelle. L'estimateur suggéré par Srinivas & Patnaik (1994) est $F_{\max}(\vec{x}) - \overline{F}(\vec{x})$, l'écart entre la valeur adaptative maximale et la valeur adaptative moyenne de la population. En effet, lorsque cette valeur tend vers 0, l'algorithme aura entièrement convergé. À l'inverse, si cette valeur tend vers un grand nombre, l'algorithme sera encore en phase d'exploration.

L'application de taux variable des opérateurs qui sera testée utilisera la constante de convergence C_v, qui peut ainsi être définie par :

$$C_v = 1 - K_1 \qquad \text{(D. 9)}$$

Avec $K_1 = \dfrac{F_{\max}(x) - \overline{F}(x)}{F_{\max,init}(x) - \overline{F}_{init}(x)}$ si $F_{\max}(\vec{x}) - \overline{F}(\vec{x}) \leq F_{\max,init}(\vec{x}) - \overline{F}_{init}(\vec{x})$

$K_1 = 1$ autrement.

où $F_{\max}(\vec{x}) - \overline{F}(\vec{x})$ est l'écart entre la valeur adaptative maximale et la valeur adaptative moyenne

de la population.

$F_{\max,init}(\vec{x}) - \overline{F}_{init}(\vec{x})$ est la valeur de $F_{\max}(\vec{x}) - \overline{F}(\vec{x})$ lors de l'initialisation de l'AG.

Ainsi, C_v varie de 0 lorsque l'AG est à son état initial à 1 lorsque l'AG a complètement convergé. Selon la discussion faite dans ci-dessus, on suggère donc les taux d'application suivants pour les opérateurs de recherche :

$$\alpha'_m = \alpha_m (1 - C_v) \qquad \text{(D.10)}$$
$$\alpha'_c = \alpha_c (C_v - 1)$$

où α_m et α_c sont respectivement les valeurs maximales choisies par l'usager pour le taux de mutation et le taux de croisement (0.95 par défaut).
α'_m et α'_c sont respectivement les taux de mutation et de croisement applicables pour la population pour une génération ayant la convergence C_v donnée lors de l'utilisation de cette heuristique.

D.7 Application d'un modèle substitut à la fonction coût pour un AG

En général, pour les systèmes de grande envergure, ce sont les diverses évaluations de F($\vec{x}$) qui utilisent la majeure partie du temps de calcul requis pour un problème. Puisque chaque vérification de la fonction coût est très coûteuse, il peut être avantageux de remplacer certaines évaluations de la fonction coût par un modèle mathématique (Booker, Dennis, Frank, Serafini & Torczon, 1998 ; Serafini, 1998 ; Stelmack, Batill & Beck, 2000 ; Xu & Grandhi, 2000 ; Canfield, 2004 ; Anderson & Redhe, 2003).

La littérature propose une variété de techniques pour créer des modèles mathématiques aptes à remplacer la fonction coût. Voir Vapnik (1998), Ong, Lum, Nair, Shi & Zhang

(2003), Ong, Nair, Keane, & Wong (2005) et Bosman & Thierens (2005) pour des exposés plus approfondis sur le sujet.

Parmi les modèles très populaires, on retrouve entre autres les modèles de régression polynomiale d'ordre peu élevé (Smith & Mason, 1997) et les modèles Bayesien (krigeage) (Ratle, 1998 ; Ratle, 2001 ; Emmerich, Giotis, Multlu Ozdemir, Back & Glannakoglou, 2002 ; Willmes, Back, Jin & Sendhoff, 2003). Aussi, la littérature nous propose l'utilisation de réseaux de neurones pour remplacer certaines évaluations de la fonction (Rasheed, 2000 ; Jin, Olhofer & Sendhoff 2002 ; Rasheed, Ni & Vattam, 2005 ; Hong, Lee & Tahk, 2003 ; Husken, Jin & Sendhoff, 2005 ; Won, Tapabrata & Tai, 2003).

Puisque l'on s'intéresse à obtenir un algorithme capable d'apprendre en temps réel la topologie de la fonction coût pour un problème de design de grande envergure, les bons résultats de Song & Keane (2005), obtenus lors du design d'une fixation d'ailette sur un rotor de turbine à gaz, suggèrent que les réseaux de neurones artificiels sont une solution appropriée pour le problème actuel.

Par contre, lorsque l'on utilise un modèle mathématique dans un algorithme évolutionnaire, il faut non seulement savoir lequel utiliser, mais aussi quand l'utiliser. L'une des méthodes les plus simples est d'utiliser le modèle pour évaluer la valeur adaptative de la population entière, puis d'utiliser une fois à chaque T_{RN} générations la vraie fonction F($\vec{x}$) pour vérifier et corriger le modèle (Ratle, 1998). Pour cette méthode, T_{RN} est un nombre entier défini par l'utilisateur, généralement égal à 10.

Une autre méthode populaire consiste à n'évaluer avec la vraie fonction que les points ayant le plus amélioré la qualité de la solution. On peut donc par exemple évaluer exactement les R points les plus prometteurs, ou évaluer avec la vraie fonction tous les points améliorant de plus d'une certaine quantité le meilleur point obtenu lors des

dernières évaluations de la fonction (Emmerich, Giotis, Multlu Ozdemir, Back & Glannakoglou, 2002).

Song & Keane (2005), évaluent plutôt la qualité du modèle substitut en chaque point, et proposent d'utiliser la vraie fonction uniquement lorsque le modèle possède une erreur divergeant de plus d'un certain critère (typiquement trois écarts-types).

Dans le cas qui nous intéresse, on désire créer un processus d'application de modèle substitut qui se prête bien à la résolution en parallèle de F($\vec{x}$). Ainsi, pour une étape de calcul donnée, il est important de savoir d'avance le nombre d'évaluations que l'on doit faire afin de ne pas perdre de ressources en laissant certains nœuds du réseau d'ordinateurs libres pendant que les autres calculent. La méthode de Ratle (1998) semble particulièrement bien adaptée à ce type de calculs, et les bons résultats qu'il a obtenus laissent penser que cette méthode conviendrait adéquatement pour le processus décrit dans ce document. La section 5.4.3 présente une adaptation de cette méthode qui permet de l'insérer aisément dans l'AGENT.