Numéro d'ordre : 3930

THÈSE

présentée devant

l'Université Louis Pasteur, Strasbourg I

pour obtenir le grade de :

Docteur de l'Université Louis Pasteur, Strasbourg I

Spécialité Informatique

par

Michel SALOMON

Sujet de la thèse :

Étude de la parallélisation de méthodes heuristiques d'optimisation combinatoire. Application au recalage d'images médicales.

Soutenue publiquement le 11 décembre 2001 devant le jury composé de :

Mme Christine GRAFFIGNE Professeur à l'Université René Descartes, Paris V		Rapporteur externe
M. Denis TRYSTRAM Professeur à l'Institut National Polytechnique de Grenoble		Rapporteur externe
M. Jerzy KORCZAK Professeur à l'Université Louis Pasteur, Strasbourg I		Rapporteur interne
M. Guy-René PERRIN Professeur à l'Université Louis Pasteur, Strasbourg I	Directeur de thèse	Examinateur
M. Fabrice HEITZ Professeur à l'Université Louis Pasteur, Strasbourg I	Codirecteur de thèse	Examinateur

Remerciements

Je remercie tout particulièrement Guy-René Perrin *et Fabrice* Heitz *pour m'avoir encadré, ainsi que pour leurs nombreux conseils, suggestions et encouragements.*

Je remercie Jerzy Korczak *pour m'avoir fait l'honneur de présider le jury de thèse.*

Je remercie Christine Graffigne *et Denis* Trystram *d'avoir accepté d'être les rapporteurs de cette thèse.*

Merci également à tous les membres de l'équipe Image et Calcul Parallèle Scientifique du Laboratoire des Sciences de l'Image, de l'Informatique et de la Télédétection (LSIIT) pour la bonne ambiance qu'ils contribuent à créer, propice à la réflexion.

Finalement, je tiens à remercier ma famille pour son soutien constant tout au long de mes études.

Table des matières

Introduction générale

Un problème d'optimisation combinatoire est un problème qui peut s'exprimer par une fonction (dite de coût) avec ou sans contraintes, à minimiser ou maximiser sur un ensemble de définition fini ou dénombrable. C'est le cas de nombreux problèmes, dans des domaines d'applications très variés, qu'ils soient scientifiques ou techniques. Pour illustrer ce propos on peut citer des problèmes académiques tels que la coloration de graphes ou le sac à dos multidimensionnel, ainsi que des applications réelles comme le positionnement de composants dans la conception de circuits imprimés, la définition de réseaux de radio-émetteurs ou la restauration d'images. Aussi, l'optimisation combinatoire est un domaine qui fait l'objet de recherches intenses.

Tous les problèmes d'optimisation n'ont bien entendu pas le même degré de difficulté, celui-ci étant surtout lié à la dimension de l'espace de recherche et au « paysage » de la fonction à optimiser (nombre de minima, dérivabilité, etc). En conséquence, de multiples algorithmes d'optimisation ont été développés, certains étant plus adaptés à des problèmes présentant certaines caractéristiques, tandis qu'ils échouent sur des problèmes où d'autres méthodes d'optimisation sont adéquates, et inversement. Ces algorithmes d'optimisation peuvent être classés de différentes manières. Nous distinguons ainsi les classes suivantes : algorithmes d'optimisation locale/algorithmes d'optimisation globale. Alors que les algorithmes de la première classe sont piégés par le premier minimum qu'ils rencontrent ou sont handicapés par la taille de l'espace de recherche, les algorithmes de la seconde classe ne présentent pas ces inconvénients et permettent de trouver une solution « proche » de l'optimum global. En revanche, les algorithmes de la première classe convergent plus rapidement que ceux de la seconde, tout en ayant un coût calculatoire moindre. En définitive, il n'existe pas de meilleur algorithme d'optimisation en termes de performances, que ce soit au niveau de la qualité des résultats ou des temps de calcul, et ceci indépendamment du problème considéré.

Le travail que nous présentons a essentiellement deux objectifs. D'une part l'étude d'algorithmes d'optimisation globale dans le cadre d'un problème particulier, d'autre part l'évaluation de l'adéquation de ces algorithmes à certains modes de parallélisation. Du fait d'une collaboration entre le LSIIT (ULP/CNRS) et l'Institut de Physique Biologique (ULP-Hôpitaux Universitaires de Strasbourg/CNRS), initiée fin 1997 dans le cadre du Programme Pluri-Formation « Analyse et Synthèse Multi-images », nous avons choisi comme champ d'étude le recalage en imagerie médicale, plus précisément le recalage rigide et le recalage déformable (tout deux denses, i.e. basés sur les niveaux de gris, en 3D). Dans ce contexte, le but du recalage est la mise en correspondance des structures anatomiques afin de suivre, par exemple, l'évolution de lésions ou d'évaluer l'efficacité d'un traitement. De nombreuses méthodes de recalage en imagerie médicale

consistent à minimiser une fonction de coût, ou fonction de similarité exprimant la similitude au niveau des voxels (ou des pixels en 2D) des images que l'on cherche à recaler entre elles. Or les fonctions de similarité classiques en recalage d'images sont non linéaires, irrégulières et présentent de nombreux minima locaux. Pour résoudre ce problème d'optimisation difficile, il est donc nécessaire de recourir à un algorithme d'optimisation globale.

Parmi ce type d'algorithmes, nous avons étudié la recherche tabou, ainsi que plusieurs algorithmes stochastiques obtenus par modélisation de phénomènes physiques ou biologiques : recuit simulé et ses variantes continues (notamment l'*Adaptive Simulated Annealing* et l'équation de la diffusion), et des algorithmes évolutionnaires. Un inconvénient majeur de ces algorithmes est leur coût calculatoire, qui croît suivant la dimension du problème considéré et sa difficulté ; une mise en œuvre parallèle s'impose donc. Dans le cadre du recalage d'images médicales, la parallélisation de ces algorithmes est motivée essentiellement par deux raisons : une réduction significative du temps de calcul, car la rapidité d'une méthode de recalage est un facteur primordial si on envisage une utilisation en routine clinique ; assurer la pérennité de la méthode de recalage en permettant de traiter des données plus importantes qu'il n'est possible en séquentiel (des images ayant une résolution plus fine, c'est-à-dire passer de 128^3 voxels à 256^3, voire plus). Mis à part la parallélisation des algorithmes évoqués précédemment, nous avons également regardé le comportement d'algorithmes hybrides parallèles qui s'appuient sur des concepts introduits par différents algorithmes d'optimisation. En outre, afin de pouvoir traiter indifféremment le recalage d'images monomodales (provenant de la même modalité, i.e. source d'acquisition) ou d'images multimodales, l'algorithme de recalage doit être « idéalement » indépendant de la fonction de coût. En effet, d'une part il existe une variété de fonctions de similarité plus ou moins complexes suivant la nature du recalage qu'elles permettent de traiter. D'autre part, la transformation recherchée pour effectuer le recalage ne doit pas être limitée au niveau du nombre de degrés de liberté. Or ces deux contraintes rendent caduque toute parallélisation spécifique au problème considéré : on s'oriente tout naturellement vers les parallélisations génériques.

En écho aux premières implantations SIMD de plusieurs algorithmes d'optimisation globale, notamment du recuit simulé et au vu, pour certains autres (algorithmes évolutionnaires) de leur comportement en séquentiel, nous nous sommes en particulier intéressé au parallélisme de données. À l'opposé de l'autre grand mode de parallélisation qu'est le parallélisme de contrôle, qui repose sur la vision d'un programme comme des parties séquentielles pouvant être exécutées simultanément, le parallélisme de données, ou data-parallélisme consiste à appliquer en parallèle un même traitement à des données distribuées sur les différents processeurs. Il s'agit donc d'un modèle de programmation et un des objectifs de cette thèse est aussi de voir comment ce modèle de programmation « résiste » à ces algorithmes par rapport aux travaux originels sur architecture SIMD. En filigrane, l'implantation d'algorithmes data-parallèles sur une machine MIMD nous permet donc d'étudier la pertinence de ce mode de parallélisation lorsqu'il n'y pas d'adéquation entre le modèle de programmation et le modèle d'exécution.

En réalité on voit que cette thèse est à facettes multiples, mais peut se résumer globalement comme étant une étude sur l'adéquation d'algorithmes d'optimisation globale au problème du recalage d'images médicales, en même temps que leur adéquation à une mise en œuvre dans le modèle de programmation data-parallèle.

La thèse est structurée en trois parties.

La première partie est consacrée à la présentation de différents algorithmes d'optimisation globale. Divers aspects de chaque algorithme sont abordés, en particulier sur le plan théorique lorsqu'un modèle mathématique permettant de modéliser l'algorithme existe (preuve de la convergence, choix des paramètres, etc). Pour les algorithmes d'une même famille, la description est faite dans un cadre commun afin de mettre en évidence les différences et les similitudes entre chaque approche. Sur le plan des parallélisations nous avons essayé de faire le panorama le plus complet possible (par exemple, dans le cas du recuit simulé nous présentons l'approche par recuits multi-températures, par essais multiples, massivement parallèle, etc).

Dans la seconde partie, nous faisons une introduction générale à la problématique du recalage en imagerie médicale 3D mono et multimodale. Un état de l'art succinct permet au lecteur de se faire une idée de la difficulté posée par le recalage dans ce domaine. Nous décrivons ensuite plus précisément les deux problèmes de recalage que nous avons retenus, à savoir le recalage rigide et le recalage déformable. La transformation rigide permet de faire un recalage global, tandis que la transformation déformable prend en compte des variations qui sont locales. À noter que, comme nous le verrons, la méthode de recalage déformable considérée s'appuie sur une modélisation paramétrique hiérarchique de la déformation continue, ce qui permet de réduire fortement sa complexité calculatoire par rapport à d'autres méthodes de recalage déformable.

La troisième partie traite de l'objectif de la thèse, c'est-à-dire de l'étude de l'adéquation d'algorithmes d'optimisation globale à la problématique du recalage et à leur mise en œuvre data-parallèle. Tout d'abord, dans le cas du recalage rigide, pour chaque algorithme retenu nous menons une étude de sa version séquentielle. Cela nous permet d'une part, d'évaluer pour chaque algorithme son aptitude à résoudre ce problème de recalage et, d'autre part, de déterminer le mode de parallélisation le plus adéquat. Ceci se traduit par la validation pour le recalage rigide et pour certains algorithmes, d'approches data-parallèles existantes ou mises en œuvre ici, à notre connaissance, pour la première fois. Chaque étude d'un algorithme séquentiel ou parallèle porte sur ses performances, tant au niveau de la qualité des recalages produits que des temps de calcul, mais également sur l'influence de ses divers paramètres. À la lumière des résultats précédents, pour le recalage déformable, nous avons restreint le domaine d'étude pour le recalage déformable de manière à privilégier deux algorithmes : d'un côté un algorithme évolutionnaire, l'évolution différentielle qui induit la parallélisation la plus performante dans le cas du recalage rigide, de l'autre l'équation de la diffusion. Ce dernier algorithme se caractérise, dans le cadre de la méthode de recalage déformable considérée, par des calculs réguliers et locaux lors de l'optimisation des variables, ce qui du fait de leur très grand nombre, laisse envisager un parallélisme de données massif qui pourrait être mis en œuvre très efficacement sur une machine SIMD qui possèderait un très grand nombre de processeurs. Toutefois, comme ce modèle d'architecture parallèle n'est plus d'actualité, nous présentons une implantation sur machine MIMD. Les différents algorithmes parallèles qui sont issus de ce travail ont été appliqués et validés sur des images médicales 3D du cerveau, de taille 128^3 et 256^3 voxels, obtenues par résonance magnétique. Par exemple, dans le cas du recalage déformable, la parallélisation massive permet de traiter des images 256^3 voxels, alors qu'en séquentiel sur une station de travail on est pour l'instant limité à des images 128^3. De plus, le gain calculatoire qu'elle présente permet d'accélérer de manière appréciable le traitement des images 128^3, et laisse entrevoir une utilisation future de la méthode de recalage déformable sur des images 256^3 en routine clinique.

Première partie

Méthodes heuristiques d'optimisation combinatoire et leurs parallélisations : État de l'art

Introduction

Comme nous l'avons déjà souligné les problèmes se modélisant comme un problème d'optimisation multidimensionnelle sont très divers, les exemples sont d'ailleurs légion. L'un des plus typique est le problème du voyageur de commerce [72] (désigné par TSP pour *Traveling Salesman Problem*). Ce dernier consiste à trouver un circuit de longueur minimale partant d'une ville et passant exactement une fois par les autres villes, pour finalement revenir à celle de départ. La difficulté de résolution du TSP vient du fait que la taille de l'espace de recherche croît de façon exponentielle avec le nombre n de villes, puisqu'on a $\frac{(n-1)!}{2} \approx \frac{1}{2}\sqrt{2\pi(n-1)}\left(\frac{n-1}{e}\right)^{n-1}$ circuits possibles (la position de départ est arbitraire et on ne tient pas compte de l'ordre des villes). Le TSP est très utilisé pour évaluer des algorithmes d'optimisation car il modélise de manière simplifiée de nombreux problèmes d'ordonnancement de tâches.

Formellement tout problème d'optimisation sans contraintes peut s'exprimer de la manière suivante : *étant donné un ensemble Ω (l'espace de recherche) de configurations x du problème à résoudre et une fonction de coût C, trouver la configuration x' qui soit de coût minimal*

$$C\left(x'\right) = \min\left\{C(x) \mid x \in \Omega\right\}.$$

Cette configuration x' peut ne pas être unique. D'autre part il est à noter que dans certains problèmes et algorithmes on parlera plutôt de fonction d'énergie (notée E) que de fonction de coût. Notons également qu'un problème de maximisation peut s'exprimer sous la forme d'une minimisation : $\max\{C(x) \mid x \in \Omega\} = -\min\{-C(x) \mid x \in \Omega\}$. On dira qu'une configuration x' est un minimum local s'il existe un voisinage de taille $\epsilon \in \mathbb{R}$, $\epsilon > 0$, défini par $V(x') = \{\, x \in \Omega \mid \|x - x'\| < \epsilon\}$ tel que pour toute configuration $x \in V(x')$ on a $C\left(x'\right) \leq C\left(x\right)$. Enfin une configuration de coût minimal sera appelée un minimum global.

Lorsque le problème est formalisé par un ensemble de contraintes, et que l'ensemble Ω est fini ou dénombrable, mais avec un cardinal ne permettant pas de l'énumérer en un temps « raisonnable », alors le problème d'optimisation est dit combinatoire. Dans ce contexte, résoudre un problème d'optimisation combinatoire consiste donc à trouver la configuration optimale dans le sens de C sur l'ensemble des configurations de Ω satisfaisant les contraintes du problème. Il existe différentes classes de problèmes d'optimisation combinatoire, dont la classe NP (pour *Non deterministic Polynomial*), le lecteur intéressé trouvera des informations plus complètes dans l'ouvrage [98] ; parmi les problèmes de cette classe, les plus difficiles sont dits NP-complet. Intuitivement, résoudre un problème NP-complet peut être vu comme le parcours d'un arbre dont le nombre de nœuds (un nœud correspond à faire un choix pour une variable de la configuration) croît exponentiellement suivant sa hauteur qui est polynomiale.

De nombreux algorithmes ont été proposés pour résoudre des problèmes d'optimisation. Différentes classifications ont été proposées : algorithmes (ou méthodes) déterministes/algorithmes stochastiques ; algorithmes de recherche locale/algorithmes de recherche globale ; ou encore la distinction méthodes exactes/méthodes approchées. Nous distinguons les classes suivantes :

① *Les algorithmes d'optimisation locale*
Dans cette classe nous mettons tout algorithme qui est piégé par le premier minimum qu'il rencontre ou qui ne permet pas d'obtenir une solution approchée de l'optimum global en raison de la trop grande cardinalité de l'espace de recherche. La descente du gradient ou le simplex sont des exemples de tels algorithmes.

② *Les algorithmes d'optimisation globale*
Tout algorithme qui n'est pas sensible aux minima locaux et qui permet d'obtenir une solution « proche » de l'optimum en évitant d'explorer systématiquement l'espace de recherche. Ces algorithmes sont forts variés, comprenant notamment des techniques stochastiques obtenues par modélisation de phénomènes physiques ou biologiques, très populaires dans la communauté scientifique : algorithmes génétiques, recuit simulé, automates cellulaires, etc.

Par conséquent, résoudre un problème d'optimisation combinatoire avec un algorithme d'optimisation locale risque d'aboutir à une solution sous-optimale très éloignée de l'optimum recherché (du moins sans pré-traitement par une méthode globale). Aussi, avons nous décidé de nous intéresser en particulier aux algorithmes suivants :

- Le recuit simulé, un algorithme défini par Kirkpatrick *et al.* [69] dans le cadre d'un problème de placement de composants sur un circuit imprimé. Il s'appuie sur un cadre mathématique bien défini : la modélisation markovienne et la décomposition en cycles, ce qui a permis de définir notamment des conditions suffisantes de convergence. Des méthodes liées au recuit simulé sont également étudiées (équation de la diffusion par exemple [4, 44]).
- Les algorithmes évolutionnaires (algorithmes génétiques [60], stratégies d'évolution [94, 113] et évolution différentielle [120]) qui sont basés sur la modélisation de l'évolution biologique.
- La recherche tabou, une méta-heuristique introduite par Glover [45, 46].

Cependant pour certains problèmes d'optimisation combinatoire ces algorithmes nécessitent, pour converger, un temps de calcul qui croît rapidement avec le nombre de variables à optimiser et la complexité du « paysage » de la fonction d'énergie, aboutissant à des implantations séquentielles qui sont trop lentes. Pour accélérer la vitesse de calcul deux voies sont généralement étudiées, tout d'abord un travail direct sur les paramètres de l'algorithme, l'algorithme lui-même et la fonction de coût ; et en second lieu la parallélisation. On se tourne tout naturellement vers la parallélisation du fait de son caractère plus général et surtout en raison des apports suivants :

- réduction importante du temps de calcul, permettant d'accélérer fortement le traitement ;
- possibilité de résolution de problèmes de taille plus importante que dans le cas séquentiel ;
- amélioration de la qualité de la solution finale ;
- possibilité d'implantation de nouveaux algorithmes reposant sur les méthodes séquentielles.

On distingue deux catégories de parallélisation :

① *les parallélisations liées au problème*, i.e. la décomposition fonctionnelle (parallélisation des fonctions sous-jacentes à l'algorithme, la fonction de coût par exemple), et le partitionnement des données (division de l'espace de recherche, distribution des variables à optimiser) ;

② *les parallélisations génériques*, qui reviennent à paralléliser l'algorithme d'optimisation.

Afin d'obtenir des résultats généraux on s'oriente vers les parallélisations génériques et les hybrides parallèles de certaines des techniques présentées (algorithme génétique/recuit simulé).

Chapitre 1

Recuit simulé et méthodes liées

1.1 Origine du recuit simulé

Le recuit simulé [2, 69] fut obtenu par analogie avec le phénomène thermodynamique de recuit des métaux, ou avec le processus de refroidissement aboutissant à la cristallisation d'un liquide. Initialement le métal est porté à très haute température, puis il est refroidi progressivement :

- à haute température les atomes sont très agités de telle sorte que toutes les configurations atomiques sont équiprobables ;
- à basse température les atomes s'organisent pour aboutir à une structure atomique parfaite proche de l'état d'énergie minimale.

Le refroidissement doit être très lent pour ne pas rester bloqué dans un minimum local : à chaque instant le métal doit se trouver dans un état de quasi-équilibre thermique permettant de sortir du minimum local en atteignant un autre état d'équilibre associé à la température courante. Si le refroidissement est trop rapide on obtient un état qui au sens mathématique est une solution sous-optimale.

Pour simuler le phénomène de recuit des métaux on utilise un résultat de la mécanique statistique. Soit Ω un ensemble de configurations atomiques possibles et E une fonction d'énergie définie sur Ω. La probabilité pour que le système soit dans un niveau d'énergie $E(X)$ associé à la configuration X est donné à la température T par la distribution de Boltzmann :

$$P(X) = \frac{1}{Z_T} \exp\left(-\frac{E(X)}{K_B T}\right), Z_T \text{ constante de normalisation et } K_B \text{ constante de Boltzmann.}$$

Metropolis *et al.* [83] définirent un algorithme permettant de simuler le comportement du système considéré à une température T, tout en respectant la distribution de Boltzmann. Au départ le système se trouve dans une configuration arbitraire, il passe ensuite dans une suite de configurations obtenues par réarrangement élémentaire de la configuration courante par l'intermédiaire de la procédure de transition suivante : $P(Y \mid X) = \min\left[1, \exp\left(-\frac{E(Y)-E(X)}{T}\right)\right]$

Les premiers à avoir eu l'idée d'utiliser cet algorithme pour résoudre un problème d'optimisation furent Kirkpatrick *et al.* [69]. Ils remplacèrent la fonction d'énergie par la fonction à minimiser, une configuration atomique par une solution admissible du problème et introduisirent d'autre part un schéma de température évoluant d'une valeur haute vers une valeur basse.

Bien qu'originellement le recuit simulé fut introduit pour des problèmes discrets, il peut être utilisé pour résoudre des problèmes continus en discrétisant l'espace de recherche. Cependant, pour traiter le cas continu, différents algorithmes reposant sur les principes sous-jacents du recuit simulé furent développés par la suite :

- les algorithmes *Classical Simulated Annealing* (CSA) et *Fast Simulated Annealing* (FSA) [124] qui sont dérivés directement du recuit simulé discret ;
- la méthode dite *Stochastic Approximation with Smoothing* (SAS) [122] ;
- l'algorithme basé sur l'équation de la diffusion, proposé par Geman *et al* [44] ;
- le recuit simulé adaptatif (*Adaptive Simulated Annealing* (ASA), appelé initialement le *Very Fast Simulated Re-annealing* (VFSR)) introduit par Lester Ingber [63].

L'intérêt du recuit simulé ou de toute autre méthode dérivée, par rapport à une recherche locale, est dû au fait que ces algorithmes acceptent de façon probabiliste des configurations d'énergie plus élevée, ce qui permet de ne pas rester piégé par un minimum local. La probabilité d'acceptation d'une configuration est d'autant plus élevée que T est grand.

Nous allons tout d'abord présenter le recuit simulé « classique » dans le cas discret. Ensuite nous aborderons le cas continu : nous nous intéresserons plus particulièrement à l'équation de la diffusion et au recuit simulé adaptatif. Tous les algorithmes seront présentés en considérant un problème de minimisation.

1.2 Cas discret

1.2.1 Modélisation du recuit simulé par un processus markovien

Considérons un espace de configurations Ω et une fonction d'énergie (ou de coût dans le vocabulaire de l'optimisation) $E : \Omega \to \mathbb{R}$ que l'on cherche à minimiser sur Ω. On définit un système de voisinage V pour tout $x \in \Omega$ auquel on associe une matrice stochastique de transition $Q = \left[q(x,y) \mid (x,y) \in \Omega^2\right]$ telle que :

$$V(x) = \{y \in \Omega \mid q(x,y) > 0\} \tag{1.1}$$

On suppose que Q est symétrique et irréductible, i.e. que $q(x,y) = q(y,x)$ et qu'il existe pour tout $(x,y) \in \Omega^2$ une suite finie d'états $x_0, ..., x_K \in \Omega$ tels que $x_0 = x, x_K = y$ et

$$q(x_0,x_1) \times q(x_1,x_2) \times ... \times q(x_{K-1},x_K) > 0 \tag{1.2}$$

On se donne également une séquence de nombres $(T_n)_{n\in\mathbb{N}}$ appelés températures et vérifiant :

$$T_n \geq T_{n+1} \ \ \forall n \geq 0 \text{ et } \lim_{n\to+\infty} T_n = 0 \tag{1.3}$$

Considérons comme procédure de transition l'algorithme de Metropolis [83]. L'algorithme du recuit simulé construit alors une suite $X_0, X_1, ..., X_n$ de configurations aléatoires à valeurs dans Ω où X_0 est choisie arbitrairement ; chaque nouvelle configuration est obtenue en deux phases :

① *Phase d'exploration* : on tire aléatoirement une configuration $Y_n \in V(X_n)$ suivant la loi

$$P(Y_n = y \mid X_n) = q(X_n, y) \tag{1.4}$$

② *Phase d'acceptation* : la configuration Y_n est acceptée avec la probabilité

$$P(X_{n+1} = Y_n \mid X_0, ..., X_n) = \min\left[1, \exp\left(-\frac{E(Y_n) - E(X_n)}{T_n}\right)\right] \text{ (Metropolis)}, \quad (1.5)$$

dans le cas contraire Y_n est rejetée et $X_{n+1} = X_n$.

La suite $(X_n)_{n\in\mathbb{N}}$ correspond à l'évolution d'une chaîne de Markov inhomogène contrôlée par le schéma de température $(T_n)_{n\in\mathbb{N}}$, et dont la matrice de transition $P_{T_n} = \left[p_{T_n}(x,y) \mid (x,y) \in \Omega^2\right]$ où $p_{T_n}(x,y) = P(X_{n+1} = y \mid X_n = x)$, vérifie :

$$p_{T_n}(x,y) = \begin{cases} 0 & \text{si } y \notin V(x) \\ q(x,y) \cdot \exp\left(-\frac{[E(y)-E(x)]^+}{T_n}\right) & \text{si } y \in V(x), [d]^+ = \sup(0,d) \\ 1 - \sum_{z\in\Omega, z\neq x} p_{T_n}(x,z) & \text{si } y = x \end{cases} \quad (1.6)$$

L'hypothèse de symétrie et d'irréductibilité de la matrice Q implique que la chaîne $(X_n)_{n\in\mathbb{N}}$ est irréductible et récurrente, admettant à température constante T comme unique distribution invariante la distribution de Gibbs :

$$\lim_{n\to+\infty} P_T(X_n = x) = \frac{1}{Z_T} \exp\left(-\frac{E(x)}{T}\right) \text{ avec } Z_T = \sum_{x'\in\Omega} \exp\left(-\frac{E(x')}{T}\right) \quad (1.7)$$

La distribution de Gibbs peut également être simulée par d'autres algorithmes que l'algorithme de Metropolis, notamment l'échantillonneur de Gibbs [43]. Ce dernier définit une procédure de transition très intéressante dans le cas où les configurations $X_n, n \in \mathbb{N}$ sont des champs (ou vecteurs) de variables aléatoires markoviennes :

$$X_n = (X_{n,1}, ..., X_{n,s}, ..., X_{n,N}) = (x_{n,1}, ..., x_{n,s}, ..., x_{n,N}) = x_n, \text{ avec } N \in \mathbb{N}^+, x_n \in \Omega \quad (1.8)$$

s est l'index des variables, on dira aussi que s est un site ; l'espace de recherche Ω est de la forme :

$$\Omega = \Omega_1 \times ... \times \Omega_N, \text{ où } x_{n,s} \in \Omega_s \quad (1.9)$$

Dans ce contexte, il est avantageux de choisir comme nouvelle configuration X_{n+1} une configuration Y_n qui ne diffère de la configuration X_n qu'en un seul site s. Dans le cas markovien, le calcul de la variation d'énergie ne fait alors intervenir que le voisinage du site s (les configurations qui diffèrent au niveau de ce site) :

$$\begin{aligned} P(Y_{n,s} = y_{n,s} \mid X_{n,r} = x_{n,r}; r \neq s) &= \frac{P(Y_{n,s} = y_{n,s}, X_{n,r} = x_{n,r}; r \neq s)}{P(X_{n,r} = x_{n,r}; r \neq s)} \\ &= \frac{\frac{1}{Z_T} \exp\left(-\frac{E(x_{n,1},...x_{n,s-1},y_{n,s},x_{n,s+1},...,x_{n,N})}{T}\right)}{\sum_{{y'}_{n,s}\in\Omega_s} \frac{1}{Z_T} \exp\left(-\frac{E(x_{n,1},...x_{n,s-1},{y'}_{n,s},x_{n,s+1},...,x_{n,N})}{T}\right)} \\ &= \frac{\exp\left(-\frac{E(Y_n)}{T}\right)}{\sum_{\{Y'_n = y'_n \in \Omega | {y'}_{n,s}\in\Omega_s, {y'}_{n,r} = x_{n,r}; r\neq s\}} \exp\left(-\frac{E(Y'_n)}{T}\right)} \end{aligned} \quad (1.10)$$

L'échantillonneur de Gibbs utilise la probabilité conditionnelle (1.10) pour passer d'une configuration X_n à une configuration $X_{n+1} = Y_n$:

① Choix d'un site s.

② Tirage aléatoire d'une étiquette $y_{n,s}$ pour le site s suivant la distribution conditionnelle locale $P(Y_{n,s} = y_{n,s} \mid X_{n,r} = x_{n,r}; r \neq s)$.

Les sites sont alors balayés de manière aléatoire, ou suivant un ordre établi.

Suivant la façon dont sont balayés les sites, deux matrices de transition sont possibles pour la chaîne de Markov engendrée par l'échantillonneur de Gibbs. On a ainsi $P_{T_n} = \left[p_{T_n}(x,y) \mid (x,y) \in \Omega^2\right]$ qui vérifie dans le cas :

– du tirage aléatoire suivant une loi uniforme d'un site s, i.e. $q(x,y) = \frac{1}{N}$

$$p_{T_n}(x,y) = \begin{cases} 0 & \text{si } y \notin V_s(x) \cup \{x\} \\ q(x,y) \cdot \dfrac{\exp\left(-\frac{E(y)}{T_n}\right)}{\sum_{z \in V_s(x) \cup \{x\}} \exp\left(-\frac{E(z)}{T_n}\right)} & \text{si } y \in V_s(x) \cup \{x\} \end{cases} \tag{1.11}$$

où $V_s(x)$ correspond à l'ensemble des configurations différent de x au niveau du site s ;

– du parcours des sites suivant un ordre $\{s_1, ..., s_t, ..., s_N\}, s_t \in \{1, ..., N\}$ étant unique

$$p_{T_n}(x,y;t) = \begin{cases} 0 & \text{si } y \notin V_{s_t}(x) \cup \{x\} \\ \dfrac{\exp\left(-\frac{E(y)}{T_n}\right)}{\sum_{z \in V_{s_t}(x) \cup \{x\}} \exp\left(-\frac{E(z)}{T_n}\right)} & \text{si } y \in V_{s_t}(x) \cup \{x\} \end{cases} \tag{1.12}$$

où $V_{s_t}(x)$ est l'ensemble des configurations ayant une étiquette différente pour le site s_t par rapport à la configuration x.

Pour que la convergence du recuit simulé soit assurée, i.e. $\lim_{n \to \infty} P\left(X_n \in \Omega_{min}\right) = 1$ où Ω_{min} est l'ensemble des configurations d'énergie minimale, il a été montré [43] que $(T_n)_{n \in \mathbb{N}}$ doit vérifier :

$$\lim_{n \to \infty} T_n \geq \frac{R}{\ln n}, R = |\Omega| \cdot \left(\max\left\{E(x) \mid x \in \Omega\right\} - \min\left\{E(x) \mid x \in \Omega\right\}\right), \tag{1.13}$$

$(T_n)_{n \in \mathbb{N}}$ est appelé un schéma de température logarithmique. On a en particulier :

$$\begin{cases} \lim_{T \to 0} P_T(X_n = x) = \frac{1}{|\Omega_{min}|} \text{ si } x \in \Omega_{min} \\ \lim_{T \to 0} P_T(X_n = x) = 0 \text{ sinon} \end{cases} \tag{1.14}$$

1.2.2 Propriétés asymptotiques

L'étude des propriétés asymptotiques de la chaîne de Markov modélisant le recuit simulé a donné lieu à une multitude de travaux [2, 7, 43, 24, 25, 62, 132].

Condition suffisante de convergence

Un schéma de température $(T_n)_{n \in \mathbb{N}}$ sera dit asymptotiquement bon s'il garantit la convergence de l'algorithme. D'autre part, une valeur plus précise de la constante R introduite dans (1.13) a été obtenue [7] dans le cadre de la décomposition en cycles.

La notion de cycle fut introduite par Freidlin et Wentzell [42] dans le cadre de l'étude par l'approche des grandes déviations des systèmes dynamiques, puis dans l'étude théorique du recuit

simulé par Azencott [7]. En effet la chaîne de Markov modélisant le recuit simulé correspond à une exploration hiérarchique de l'espace des configurations Ω, très bien décrite par la décomposition en cycles de Ω. D'autre part la décomposition en cycles fournit un schéma adéquat pour l'étude de certaines propriétés asymptotiques [7], car les cycles permettent d'exprimer les propriétés de convergence du recuit simulé avec un nombre restreint de paramètres très simples et liés à la géométrie du paysage énergétique. Les travaux les plus aboutis dans ce domaine sont dus à Catoni [24, 25], travaux qui servirent de base à Trouvé [131, 132, 133] pour définir, lors de son étude théorique de la parallélisation du recuit simulé, la notion d'algorithme généralisé du recuit simulé ébauchée par Hwang et Sheu [62].

Vitesse de convergence

La vitesse de convergence du recuit simulé, c'est-à-dire le temps de calcul nécessaire pour trouver une configuration d'énergie minimale est fortement lié au paysage énergétique. Différents travaux [7, 25] ont abouti au théorème qui suit.

Théorème 1

Pour tout schéma de température asymptotiquement bon, un paysage énergétique et une matrice d'exploration, la vitesse de convergence est donnée par :

$$P(X_n \notin \Omega_{min}) \approx \left(\frac{K}{n}\right)^{\alpha} \tag{1.15}$$

pour n assez grand, $K > 0$ et $\alpha > 0$ des constantes bien choisies.

Catoni a exprimé la valeur optimale de l'exposant α en utilisant la notion de cycle dans [25].

1.2.3 Algorithme du recuit simulé

En général la température n'évolue pas de manière continue mais plutôt par paliers, ce qui représente une approximation par une suite constante par morceaux. L'algorithme du recuit simulé ne se modélise alors plus comme une chaîne inhomogène unique mais comme une suite de chaînes homogènes engendrées chacune à température constante. Cela ne pose cependant pas de problème particulier car la suite de chaînes homogènes et la chaîne inhomogène admettent la distribution de Gibbs (1.7) comme distribution stationnaire [43]. Les critères de convergence sont fixés pour permettre d'obtenir une solution proche d'une solution optimale en un temps fini.

Plusieurs remarques sont à faire :

- La température initiale T_0 doit être choisie de manière à ce que pratiquement toutes les configurations proposées soient acceptées.
- Il faut que les paliers soient suffisamment longs pour que la chaîne de Markov engendrée à température constante puisse atteindre la distribution stationnaire correspondante. Le critère de fin de palier devra donc être fixé en conséquence.
- La baisse de température entre deux paliers successifs doit être relativement faible pour permettre à la chaîne de Markov d'atteindre rapidement la distribution à l'équilibre.
- Le critère d'arrêt considéré est en général de l'un de ces trois types : le pourcentage de configurations acceptées qui passe en dessous d'un seuil fixé, la variance de l'énergie ou une température minimale.

L'algorithme de la figure 1.1 correspond à la version utilisant l'échantillonneur de Gibbs, avec parcours des sites suivant un ordre $\{s_1, ..., s_t, ..., s_N\}$ tiré aléatoirement à chaque itération.

$T \leftarrow T_0$ | *- Température initiale*
$X \leftarrow X_0$ | *- Configuration initiale*
Répéter
  Répéter
    $\{s_1, ..., s_t, ..., s_N\} \leftarrow$ **Ordre de parcours des sites tiré aléatoirement**
    Pour t **de** 1 **à** N **faire**
      Construire un ensemble de configurations $X' \in V_{s_t}(X) \cup \{X\}$
      Calculer les probabilités $p_i = P(x'_{s_t} = \lambda_i \mid x_r; r \neq s_t)$ **pour tout** λ_i **possible pour** x'_{s_t}
      $X \leftarrow X' \mid x'_{s_t} = \lambda_j$, **avec** j **le plus petit nombre vérifiant** $\sum_{i=1}^{j} p_i = \mu, \mu \in [0;1]$ **tiré selon une loi uniforme**
    Fin pour
  Jusqu'à fin de palier
  $T \leftarrow g(T)$ | *- g est strictement décroissante*
Jusqu'à *critère d'arrêt vérifié*

FIG. 1.1 – Algorithme du recuit simulé basé sur l'échantillonneur de Gibbs.

1.2.4 Accélération de l'algorithme

Différentes possibilités sont offertes pour accélérer l'algorithme du recuit simulé [7], et en particulier pour pallier la convergence trop lente du schéma de température logarithmique.

Schéma de température exponentiel

Le schéma de décroissance logarithmique convergeant relativement lentement on a introduit une décroissance exponentielle qui est de la forme $T_n = T_0 \cdot \alpha^n$, avec $n \in \mathbb{N}^+$ et $0 < \alpha < 1$. Ce schéma de température ne garantit pas la convergence, mais Catoni [25] a démontré que dans le cas d'un recuit en temps borné $L \in \mathbb{N}$, si on définit un schéma de température vérifiant $T_n = T_0 \cdot a_L^n, n \in \mathbb{N}^+, T_0$ étant indépendant de L et $a_L = (c \cdot \log L)^{\frac{1}{L}}$ alors :

$$P\left(X_L \notin \Omega_{min}\right) \approx \left(\frac{K'}{L}\right)^{\alpha} \tag{1.16}$$

α étant l'exposant le plus élevé possible que l'on puisse obtenir par un schéma de température logarithmique optimal.

Modification du voisinage

- À haute température pratiquement toutes les configurations sont acceptées. On définit donc un voisinage relativement étendu pour permettre à l'algorithme d'évoluer rapidement dans différentes parties de l'espace de recherche.
- À basse température le voisinage sera restreint aux configurations ne variant que très peu par rapport à la configuration courante.

Accélération par distorsion de la fonction d'énergie

Cette technique consiste à accroître la vitesse de convergence de l'algorithme, en améliorant la valeur optimale de l'exposant α dans (1.15) par l'utilisation d'un schéma de décroissance de

température qui soit optimal. Azencott [7] montre qu'en remplaçant la fonction d'énergie $E(x)$ par la fonction $h(E(x))$ où $h : \mathbb{R} \to \mathbb{R}$ est une fonction croissante, strictement concave et dérivable, et en utilisant un schéma de température adapté à $h(E)$ on obtient un exposant $\alpha_h > \alpha$ pour la vitesse de convergence. Il faut cependant remarquer que ce gain au niveau de la valeur de l'exposant est contrebalancé par une constante K plus grande.

Accélération par recuits multiples indépendants

On considère un recuit simulé devant s'exécuter en un temps borné $L \in \mathbb{N}$, et que l'on a modélisé par la suite de configurations $X_0, X_1, ..., X_L$ ayant pour vitesse de convergence $\left(\frac{K}{L}\right)^{\alpha}$. En répétant $s_L \approx \frac{L}{eK}$ recuits indépendants, chacun de longueur $l \approx eK$, et en gardant la meilleure solution parmi les solutions obtenues, on aboutit à une vitesse de convergence vérifiant $P\left(X_L \notin \Omega_{min}\right) \leq e^{-\rho L}$, avec $\rho = \frac{\alpha}{2eK}$.

1.3 Cas continu

Dans cette section nous allons nous focaliser sur l'équation de la diffusion [4, 44] et le recuit simulé adaptatif [64]. Le choix de ces deux méthodes s'explique par le fait qu'elles permettent dans le cas de configurations correspondant à des champs (ou vecteurs de variables aléatoires) (1.8) une remise à jour simultanée de tous les sites. Les expressions (1.8) et (1.9) se caractérisent dans le cas continu par $\Omega = \mathbb{R}^N, N \in \mathbb{N}^+$ et $\Omega_s = \mathbb{R}$ pour tout $s \in \{1, ..., N\}$. Le *Classical Simulated Annealing (CSA)*, le *Fast Simulated Annealing* (FSA) [124, 125] et la méthode dite *Stochastic Approximation with Smoothing* [122] ne seront que brièvement présentés.

1.3.1 Les algorithmes CSA, FSA et SAS

Classical Simulated Annealing et Fast Simulated Annealing

Ces deux algorithmes s'inspirent directement du recuit simulé, ne se différenciant de ce dernier qu'à deux niveaux [124, 125] :

① La manière dont est engendrée une nouvelle configuration. En effet, dans le cas du CSA la configuration Y_n s'obtient en additionnant à la configuration X_n un vecteur aléatoire $Z = (z_s)_{s \in \{1,...,N\}}$ dont chaque composante est tirée suivant une loi normale de moyenne nulle et d'écart type variant avec la température. Le *Fast Simulated Annealing* se caractérise quand à lui par un vecteur aléatoire dont les composantes suivent une distribution de Cauchy de moyenne également nulle et d'écart type également fonction de la température.

② Le schéma de température. Comme l'ont montré Geman et Geman [43], une condition suffisante pour que le *Classical Simulated Annealing* converge est que le schéma de température soit à décroissance logarithmique (voir (1.13)), soit en prenant une température initiale T_0 suffisamment grande :

$$T_n = \frac{T_0}{\ln n}, n > 0 \tag{1.17}$$

En comparaison, le schéma de température du FSA est plus rapide car il correspond à une descente linéaire :

$$T_n = \frac{T_0}{n}, n > 0 \tag{1.18}$$

Néanmoins, en pratique les schémas de température (1.17) et (1.18) ne sont pas utilisés. En fait comme pour le recuit simulé discret, on utilise le schéma de température exponentiel présenté à la section 1.2.4.

Stochastic Approximation with Smoothing

La philosophie de la méthode dite *Stochastic Approximation with Smoothing* proposée par Styblinski *et al.* [122] est très différente des deux algorithmes que nous avons présentés précédemment. De fait, la méthode SAS ne va pas chercher à optimiser directement la fonction d'énergie, mais la fonction $\overline{E}$ obtenue par convolution de E avec une densité de probabilité dont la variance est donnée par la température T_n. L'optimisation de $\overline{E}$ se fait durant un certain nombre d'itérations, ces itérations définissant un palier k auquel est associée la température $T_k, k \in \mathbb{N}^+$. Après avoir trouvé le minimum de $\overline{E}$ on passe au palier suivant, i.e. on fait décroître la température T_k suivant un schéma de décroissance exponentielle, puis on calcule la nouvelle fonction d'énergie « convoluée ». Pour minimiser la fonction $\overline{E}$ sur un palier de température on tient compte de deux informations :

- la direction opposée au gradient du point courant X_n ;
- la direction de recherche « moyenne » donnée par les points précédents sur le même palier de température.

En combinant les deux directions évoquées ci-dessus on obtient la direction dans laquelle chercher le minimum. Cependant les deux directions n'ont pas la même pondération car la direction donnée localement par X_n est prépondérante.

Comme on l'a vu la méthode SAS s'appuie sur deux idées. D'une part la convolution de la fonction E avec une densité de probabilité va lisser celle-ci, aboutissant à une fonction $\overline{E}$ plus régulière, ne comportant éventuellement qu'un minimum. D'autre part le paramétrage de la variance de la densité de probabilité par une température permet de contrôler la taille de la zone que l'on explore autour de la configuration courante X_n. En effet, au départ lorsque la température est élevée l'algorithme explore encore une région étendue autour de X_n. Ensuite à mesure que la température baisse, on se focalise de plus en plus sur le comportement local de la fonction.

1.3.2 Équation de la diffusion

Nous avons vu dans la section 1.2 deux procédures permettant d'échantillonner la distribution de Gibbs (1.7) à température constante. Or, il a été montré [4, 44] que l'équation différentielle stochastique donnant la loi conditionnelle d'un processus de diffusion non-homogène permet également d'échantillonner la distribution de Gibbs. Cette équation, dite équation de la diffusion, est de la forme :

$$dx_t = -\nabla E(x_t) \cdot dt + \sqrt{2 \cdot T_t} \cdot dw_t \tag{1.19}$$

où E est la fonction à minimiser, elle doit être lipschitzienne et vérifier $\lim_{\|x\| \to \infty} E(x) = +\infty$; T_t est la température à l'instant $t \in \mathbb{R}^+$; w_t est un mouvement brownien dans $\mathbb{R}^N, N \in \mathbb{N}^+$; $X_t = x_t(= (x_{t,1}, x_{t,2}, ..., x_{t,N})) \in \mathbb{R}^N$ est la configuration courante.

Du fait que l'équation de la diffusion admet à température constante la distribution de Gibbs (1.7) comme distribution limite de X_t, sa solution est proche du minimum global de E lorsque T est proche de zéro. D'où l'idée de combiner l'équation (1.19) avec un schéma de température

décroissant (bien choisi) pour aboutir au minimum global de E, et retrouver ainsi le résultat (1.14). Pour assurer la convergence de X_t vers le minimum global, lorsque t tend vers l'infini, diverses conditions ont été données sur E (voir ci-dessus) et T_t [44], ainsi que sur dt [4]. Geman *et al.* [44] ont en particulier mis en évidence qu'à l'image de T_n dans le cas discret (1.13), $(T_t)_{t\in\mathbb{R}^+}$ doit vérifier :

$$\lim_{t\to\infty} T_t \geq \frac{c}{\ln(2+t)} \tag{1.20}$$

avec c suffisamment grand. De même Aluffi *et al.* [4] ont montré que dt doit être suffisamment petit et tendre vers zéro.

En mettant l'équation (1.19) sous la forme d'une somme de deux termes $\delta_1(t) = -\nabla E(x_t)\cdot dt$ et $\delta_2(t) = \sqrt{2\cdot T_t}\cdot dw_t$ on voit aisément le principe sous-jacent au processus d'optimisation. En effet, le terme $\delta_1(t)$ correspond à une descente du gradient, tandis que le second terme $(\delta_2(t))$ induit une perturbation aléatoire dont l'amplitude est modulée par la température. Ainsi, une température élevée va se traduire par une perturbation importante, qui, on l'espère sera suffisante pour sortir d'un minimum local. À l'inverse, ultérieurement, lorsque la température baissera, les perturbations deviendront au fur et à mesure négligeables aboutissant à une recherche uniquement guidée par la descente du gradient. Il est donc à souligner que l'équation de la diffusion ne permet d'optimiser que des fonctions pour lesquelles le gradient est défini, contrairement aux méthodes précédentes (recuit simulé discret, CSA et FSA) qui s'appuient sur le calcul de l'énergie associée à une configuration pour guider la recherche.

$t = 0$	- *Initialisation du temps*
$T_t \leftarrow T_0$	- *Température initiale*
$\Delta P_t \leftarrow \Delta P_0$	- *Pas de temps initial*
$X_t \leftarrow X_0$	- *Configuration de départ*
Répéter	
$\Delta w_{t,i} \leftarrow \mathcal{N}(0, \Delta P_t)$	- *Mouvement brownien*
$x_{t+\Delta P_t,i} \leftarrow x_{t,i} - \frac{\partial E(x_t)}{\partial x_i}\cdot\Delta P_t + \sqrt{2\cdot T_t}\cdot\Delta w_{t,i}, i\in\{1,...,N\}, N\in\mathbb{N}^+$	
$T_{t+\Delta P_t} \leftarrow g(T_t)$	- *g et h sont deux fonctions*
$\Delta P_{t+\Delta P_t} \leftarrow h(\Delta P_t)$	*strictement décroissantes*
$t \leftarrow t + \Delta P_t$	
Jusqu'à *critère d'arrêt vérifié*	

FIG. 1.2 – Algorithme d'optimisation basé sur l'équation de la diffusion.

L'algorithme que nous présentons à la figure 1.2 s'obtient par intégration numérique en discrétisant l'équation de la diffusion par la méthode d'Euler-Cauchy [4]. On voit qu'il y a un paramètre en plus à fixer par rapport au recuit simulé discret et aux équivalents continus (CSA et FSA). En pratique, les fonctions g et h qui font décroître T_t et ΔP_t suivent un schéma de décroissance exponentielle (à l'image de la température dans le recuit simulé discret). On a ainsi $T_{t+\Delta P_t} = \alpha_T\cdot T_t$, avec $t\in\mathbb{R}^+$ et $0<\alpha_T<1$; $\Delta P_{t+\Delta P_t} = \alpha_P\cdot\Delta P_t$, où $0<\alpha_P<1$.

1.3.3 Recuit simulé adaptatif (Adaptive Simulated Annealing)

Une autre variante du recuit simulé a été introduite par Lester Ingber sous le nom *Very Fast Simulated Re-annealing* [63], appelé actuellement *Adaptive Simulated Annealing* (ASA) [64, 65], i.e. littéralement le « recuit simulé adaptatif ». En effet, l'ASA se différencie du recuit simulé et des autres méthodes par l'ajout d'un schéma de température spécifique à chaque composante d'une configuration $X_t = x_t \in \Omega$, où $t \in \mathbb{R}^+$ désigne le temps de recuit. L'idée est de ne pas générer la nouvelle configuration en utilisant une distribution identique pour toutes ses composantes (comme c'est le cas pour le CSA et le FSA), mais de paramétrer la taille du support de la distribution associée à chaque composante $x_{t,s}, s \in \{1, ..., N\}$ et $N \in \mathbb{N}^+$, par une température T_{s,t_s}, de temps de recuit $t_s \in \mathbb{R}^+$. Ces températures devraient donc évoluer de sorte que pour une composante ayant un impact « important » sur la fonction E, la distribution ait un support plus réduit que celle d'une composante affectant E de façon « limitée ».

Pour cela, Ingber a ajouté une phase dite de *reannealing* qui rééchelonne les températures T_{s,t_s} ainsi que la température T_t de la phase d'acceptation. Cette phase a lieu à intervalles réguliers (en fait toutes les N_{acc} configurations acceptées). Comme nous le verrons plus loin elle consiste à modifier les températures courantes (des remontées sont possibles) et les temps de recuit qui leur sont respectivement associés. En dehors du *reannealing* les températures décroissent régulièrement toutes les N_{gen} configurations engendrées (décroissance par palier ; $N_{gen} \geq N_{acc}$), alors que simultanément les temps de recuit augmentent.

Nous allons maintenant décrire plus précisément les différentes étapes du recuit simulé adaptatif. Nous supposerons que l'espace de recherche Ω est de la forme :

$$\Omega = \Omega_1 \times ... \times \Omega_N, \text{ où } x_{n,s} \in \Omega_s = [inf_s; sup_s] \subset \mathbb{R} \tag{1.21}$$

avec $N \in \mathbb{N}^+, s \in \{1, ..., N\}$.

① *Phase d'exploration*
On construit une nouvelle configuration $Y_t = (y_{t,1}, ..., y_{t,s}, ..., y_{t,N}) = y_t \in \Omega$ comme suit :

$$y_{t,s} = x_{t,s} + \lambda_s \cdot (sup_s - inf_s), y_{t,s} \in \Omega_s \tag{1.22}$$

avec $s \in \{1, ..., N\}, N \in \mathbb{N}^+$ et λ_s qui est une variable aléatoire dans $[-1; 1]$ vérifiant :

$$\lambda_s = \text{ sgn } \left(\chi_s - \frac{1}{2}\right) \cdot T_{s,t_s} \cdot \left(\left(1 + \frac{1}{T_{s,t_s}}\right)^{|2 \cdot \chi_s - 1|} - 1\right) \tag{1.23}$$

où χ_s est tiré aléatoirement dans $[0; 1]$.

② *Phase d'acceptation*
La nouvelle configuration Y_t est acceptée avec la probabilité

$$P(X_{t+1} = Y_t \mid X_0, ..., X_t) = \min \left[1, \frac{1}{1 + \exp\left(\frac{E(Y_t) - E(X_t)}{T_t}\right)}\right] \tag{1.24}$$

dans le cas contraire, Y_t est rejetée et $X_{t+1} = X_t$.

③ *Phase de reannealing*
On commence par calculer les dérivées partielles de la fonction E au point X_{min} correspondant à la configuration ayant induit la plus petite énergie jusqu'à présent :

$$\frac{\partial E(x_{min})}{\partial x_s} = \left| \frac{E(x_{min} + \delta \cdot \mathbf{1}_s) - E(x_{min})}{\delta} \right| \tag{1.25}$$

où δ est proche de zéro et $\mathbf{1}_i$ est un vecteur à N dimensions, avec la $i^{\text{ème}}$ position égale à un et des zéros ailleurs. Inger appelle ces dérivées partielles les « sensibilités » de E par rapport aux différentes composantes d'une configuration. Par souci de clarté nous noterons les dérivés partielles de la manière suivante : $d_s = \frac{\partial E(x_{min})}{\partial x_s}, s \in \{1, ..., N\}$.
Les différentes températures et temps de recuit sont ensuite rééchelonnés comme suit :
– pour ce qui concerne les composantes on a

$$T_{s,t_s} = \frac{\max\{d_1, ..., d_N\}}{d_s} \cdot T_{s,t_s} \text{ et } t_s = \left(-\frac{1}{c} \ln \left(\frac{T_{s,t_s}}{T_{s,0}} \right) \right)^N \tag{1.26}$$

où $c \in \mathbb{R}$ est un paramètre de contrôle ;
– de même, T_0 est réinitialisée avec l'énergie associée à la dernière configuration acceptée, T_t avec celle de X_{min} et le temps de recuit par

$$t = \left(-\frac{1}{c} \ln \left(\frac{T_t}{T_0} \right) \right)^N \tag{1.27}$$

④ *Phase de refroidissement*
Le schéma de décroissance des températures et l'évolution des temps de recuit vérifient :

$$t_s = t_s + 1 \quad , \quad T_{s,t_s} = T_{s,0} \cdot \exp\left(-c \cdot t_s{}^{\frac{1}{N}} \right) \tag{1.28}$$

$$t = t + 1 \quad , \quad T_t = T_0 \cdot \exp\left(-c \cdot t^{\frac{1}{N}} \right) \tag{1.29}$$

où c est le même paramètre qu'en (1.26). Il est à noter que dans [65], un paramètre supplémentaire (appelé *quenching factor*) permet de contrôler le schéma de décroissance des températures.

La figure 1.3 donne le schéma de l'ASA tel que décrit ci-dessus. Naturellement l'algorithme peut être « modulé » car on peut très bien se passer de la phase de *reannealing*.

Remarques :
– T_0 est fixée comme étant égale à l'énergie d'une configuration engendrée aléatoirement dans l'espace de recherche ; les températures $T_{s,0}$ sont initialisées à $1,0$; les temps de recuit t et $t_s, s \in \{1, ..., N\}$ sont mis à zéro. Les seuls paramètres que l'utilisateur doit donc spécifier sont N_{acc}, N_{gen} et c. La valeur optimale de chacun de ces trois paramètres est évidemment dépendant de la fonction à optimiser, mais leur choix n'est pas critique (l'« auto-adaptation » de l'ASA permet *a priori* de réduire l'impact de mauvaises valeurs). Ainsi dans le cas de problèmes en traitement du signal [30] les auteurs ont trouvé comme valeurs acceptables pour c l'intervalle $[1; 10]$. On peut aussi calculer la valeur de c à partir de (1.29) en ayant fixé T_0, la température finale et le temps de recuit maximum.

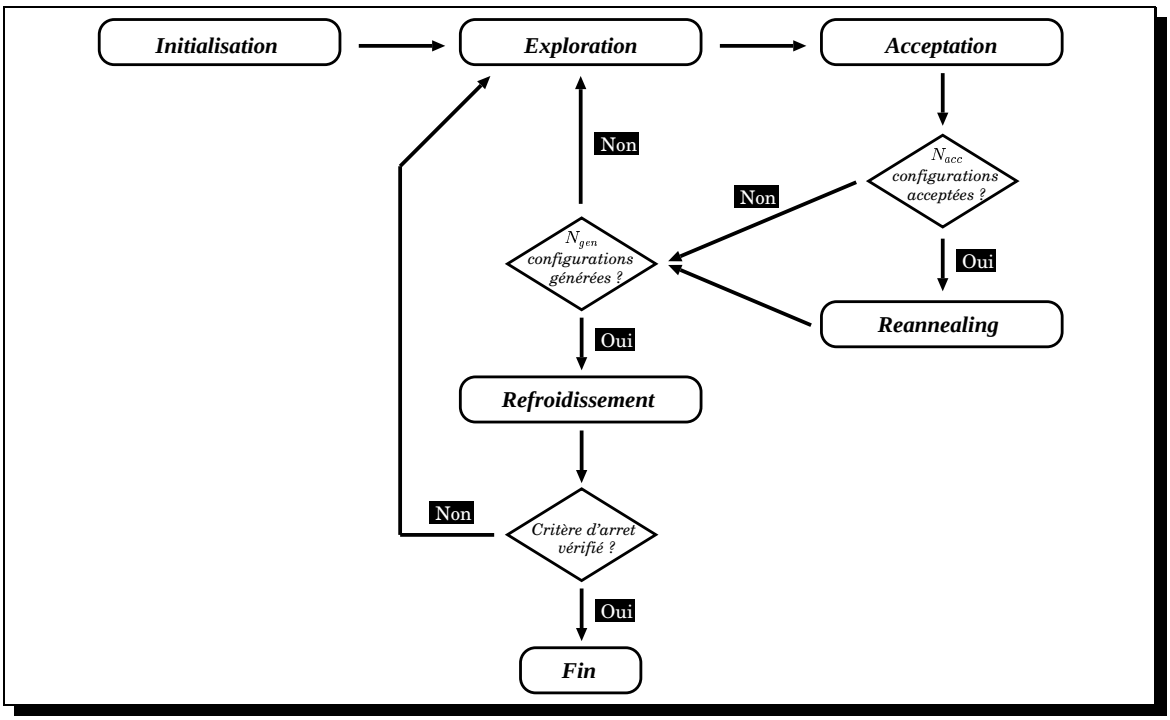


Fig. 1.3 – Schéma du recuit simulé adaptatif - *Adaptive Simulated Annealing* (ASA).

– Le critère d'arrêt est du même type que celui du recuit simulé discret (voir la section 1.2.3). Une autre possibilité est de fixer un nombre maximum de configurations engendrées.

Du point vue de la condition assurant la convergence il a été montré que tout schéma de température ne décroissant pas plus rapidement que dans (1.29) (ou (1.28)) est asymptotiquement bon (voir section 1.2.2).

1.4 Parallélisations

Les différents algorithmes que nous avons présentés précédemment font clairement apparaître la principale difficulté de la parallélisation : par définition un algorithme du type du recuit simulé est séquentiel. En effet, à l'exception de l'algorithme basé sur l'équation de la diffusion, les autres algorithmes (recuit discret, CSA, etc) acceptent toute nouvelle configuration engendrée conditionnellement à la configuration courante. Aussi, les parallélisations qui peuvent être envisagées consistent essentiellement à évaluer en parallèle plusieurs configurations ou séquences de configurations. Pour ce qui est de la parallélisation massive (remise à jour simultanée des variables), comme nous le verrons, elle induit des contraintes particulières qui réduisent son domaine d'application. Cependant pour l'algorithme basé sur l'équation de la diffusion, c'est ce dernier mode de parallélisation qui est le plus adéquat.

1.4.1 Recuits multiples parallèles

Recuits multiples indépendants

Supposons que l'on dispose de p processeurs, chaque processeur exécute alors un recuit avec un schéma de température commun à tous les processeurs et une configuration initiale choisie arbitrairement. Dans le cadre d'un recuit en temps borné, au bout de $L \in \mathbb{N}$ itérations, on choisit comme solution finale la configuration qui est d'énergie minimale, parmi toutes les configurations obtenues par les différents processeurs. Azencott met en évidence dans [6] que si le schéma de décroissance des températures est optimal, alors on obtient en $K^{\frac{p-1}{p}} L^{\frac{1}{p}}$ itérations parallèles une solution de qualité identique à celle obtenue en L itérations séquentielles, ce qui correspond à une accélération $A = \left(\frac{L}{K}\right)^{\frac{p-1}{p}}$: on remarque que A croît avec L et p, mais K étant très grand l'accélération reste limitée.

On a vu dans ce chapitre qu'il est possible de remplacer un recuit long, par plusieurs recuits optimaux séquentiels courts (section 1.2.4). On peut donc faire de même pour chaque processeur si le nombre d'itérations vérifie $L \geq 2eK$. À partir de la vitesse de convergence d'un recuit multiple optimal séquentiel ($P(X_L \notin \Omega_{min}) \leq e^{-\rho L}$ avec $\rho = \frac{\alpha}{2eK}$) on déduit la vitesse de convergence de p recuits multiples optimaux : $P(X_L \notin \Omega_{min}) \leq e^{-p\rho L}$. On a donc une accélération p, avec p processeurs, mais là encore ce résultat n'a qu'un intérêt théorique car pour définir la longueur du recuit il faut avoir une idée précise de la valeur de K. Or il n'existe aucune méthode générale permettant d'avoir une estimation précise de K à partir du problème considéré.

Recuits multiples avec interaction périodique

L'une des questions qui se pose est de savoir si l'on ne pourrait pas accélérer les schémas précédents en introduisant des interactions régulières entre les processeurs. Admettons que l'on ait p processeurs $P_0, P_1, ..., P_{p-1}$, un schéma de température commun à tous les processeurs et p configurations initiales choisies arbitrairement. On suppose également que les interactions ont lieu toutes les l itérations : pendant les itérations comprises entre $k \cdot l$ et $(k+1) \cdot l, k \in \mathbb{N}$ chaque processeur évolue indépendamment des autres ; lorsque les processeurs communiquent ils se transmettent les configurations de telle sorte que le processeur P_j reçoit les configurations des processeurs P_0 à P_{j-1}. Le processeur P_j choisit alors la nouvelle configuration $Y_{(k+1)\cdot l,j}$ de façon à ce que l'on ait :

$$E\left(Y_{(k+1)\cdot l,j}\right) = \min\left\{E\left(X_{(k+1)\cdot l,0}\right), ..., E\left(\left(X_{(k+1)\cdot l,j}\right)\right\}\right. \tag{1.30}$$

À l'issue des L itérations allouées, la configuration finale est celle retenue par le dernier processeur P_{p-1}.

La convergence de cet algorithme parallèle a été établie pour un schéma de température garantissant la convergence du recuit séquentiel. En revanche, il a été conjecturé dans [6, 8, 50] que ce schéma est asymptotiquement moins efficace que p recuits indépendants. Cette conjecture a été vérifiée expérimentalement sur différents problèmes par Graffigne [50]. D'autre part elle a également validé la conjecture disant que le schéma de recuits parallèles avec une unique interaction à la fin des recuits permet d'obtenir une vitesse de convergence supérieure à celle du recuit simulé séquentiel.

Recuits multi-températures

L'idée sous-jacente à cette approche est de faire évoluer en parallèle des recuits simulés à une température différente pour chaque processeur. Les processeurs interagissent périodiquement toutes les l itérations de manière à propager les configurations. Deux modes de propagation sont possibles : déterministe (propagation d'une température élevée vers une température plus basse) et probabiliste.

➔ *Propagation déterministe des configurations*

Graffigne [50] définit un schéma de ce type à partir du schéma des recuits multiples avec interactions (figure 1.4). À chaque processeur est associée une température fixe, et périodiquement les configurations sont propagées vers les températures plus basses. Un processeur recevant une configuration choisit alors comme configuration de départ de la prochaine sous-chaîne de Markov homogène qu'il va générer, la meilleure configuration entre celle qu'il a reçu et sa configuration courante.

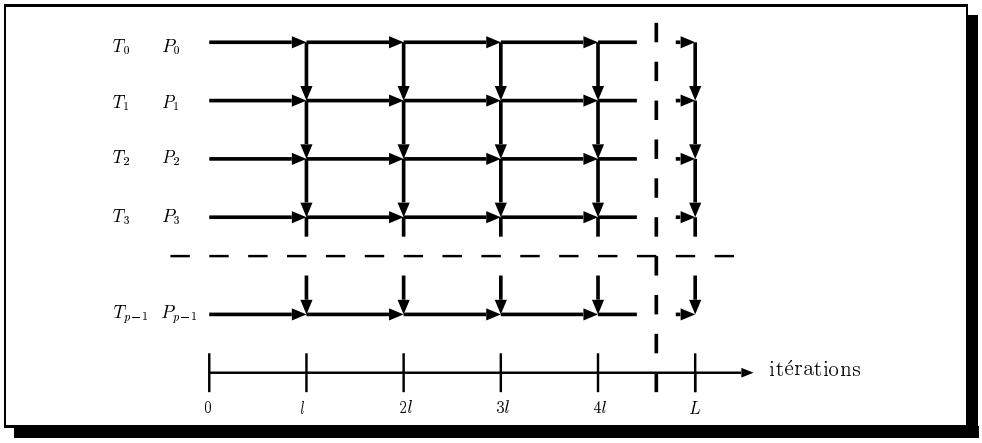


FIG. 1.4 – Recuit multi-températures déterministe.

Il se pose cependant un problème au niveau de la synchronisation des processeurs. En effet les processeurs à hautes températures acceptent pratiquement tous les mouvements, alors qu'à basse température, très peu sont acceptés. C'est pourquoi les processeurs à haute température mettent plus de temps pour réaliser les l transitions entre deux interactions. Par conséquent un schéma de communications par rendez-vous entre deux processeurs est plus adapté qu'un schéma bloquant attendant que tous les processeurs aient fait les l transitions. La vague de synchronisation part des processeurs à hautes températures et se propage vers les basses températures.

➔ *Propagation probabiliste des configurations*

Aarts et van Laarhoven [3] ont défini un algorithme dit « systolique », dans lequel une chaîne homogène (engendrée à température constante) est divisée en p sous-chaînes de longueur $\frac{L}{p}$ (L étant le nombre total d'itérations et p le nombre de processeurs). Comme le montre la figure 1.5, le schéma de l'algorithme est tel que le processeur $P_j, j \in \{1, ..., p-1\}$ ne démarre sa sous-chaîne à la température $T_n, n \in \{0, ..., p-1\}$ que lorsque le processeur P_{j-1} a terminé de générer la sienne à même température. D'autre part à chaque interaction, le processeur P_j doit choisir comme configuration de départ pour la sous-chaîne de température T_n entre la configuration

$(X_{j,n-1})$ qu'il a obtenu à la température T_{n-1} et la configuration de la sous-chaîne engendrée par le processeur P_{j-1} à la température T_n $(X_{j-1,n})$. Ce choix se fait de façon probabiliste :

- la probabilité de choisir $X_{j-1,n}$ vaut $\frac{p_n}{p_{n-1}+p_n}$
- la probabilité de choisir $X_{j,n-1}$ est égale à $\frac{p_{n-1}}{p_{n-1}+p_n}$

où p_n et p_{n-1} vérifient :

$$p_n = \frac{1}{Z_{T_n}} \exp\left(-\frac{E(X_{j-1,n})}{T_n}\right) \qquad p_{n-1} = \frac{1}{Z_{T_{n-1}}} \exp\left(-\frac{E(X_{j,n-1})}{T_{n-1}}\right) \tag{1.31}$$

$$Z_T = \sum_{x\in\Omega} \exp\left(-\frac{E(x)}{T}\right) \tag{1.32}$$

p_n et p_{n-1} correspondent aux probabilités dans la distribution stationnaire des configurations $X_{j-1,n}$ et $X_{j,n-1}$ respectivement. La constante de normalisation Z_T n'étant pas calculable en pratique on l'approche par la distribution normale.

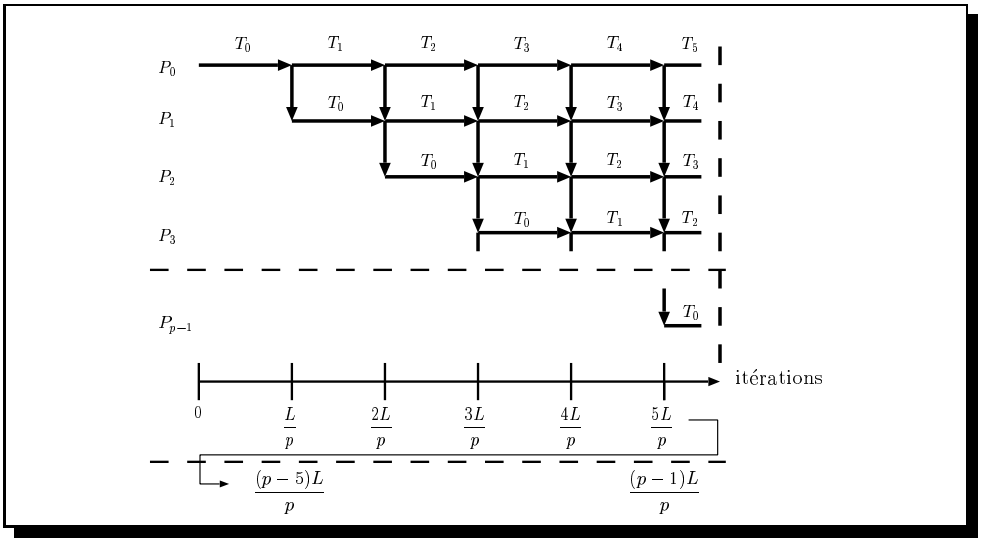


Fig. 1.5 – Recuit multi-températures probabiliste.

Cet algorithme est bien adapté à un réseau de processeurs dont la topologie est un anneau, néanmoins il faut faire attention au nombre de processeurs. En effet, quand le nombre de processeurs augmente la longueur des sous-chaînes diminue, or il faut que les sous-chaînes puissent converger vers la distribution de Gibbs.

1.4.2 Recuits parallèles par essais multiples

Le principe de la parallélisation par essais multiples est de faire des recherches simultanées en générant des sous-chaînes concurrentes indépendantes. C'est-à-dire que les p processeurs partent d'un même état initial puis chacun calcule une sous-chaîne dont la longueur est contrôlée par des paramètres fixés. Lorsque toutes les sous-chaînes sont calculées, les processeurs sont synchronisés et on choisit comme nouvelle configuration de départ des prochaines sous-chaînes une des

configurations finales. On attend une accélération surtout à basse température lorsque le taux d'acceptation $\chi(T)$ des configurations (nombre de configurations acceptées par rapport au nombre de configurations essayées) est très faible, i.e. qu'il faut tester beaucoup de configurations avant d'en trouver une qui soit acceptée. Les algorithmes reposant sur une parallélisation par essais multiples sont classés suivant deux critères :

① Les paramètres contrôlant la longueur des sous-chaînes. En général on en utilise un seul : le nombre de configurations acceptées [102] ou le nombre de configurations essayées.

② Le nombre de processeurs utilisés, car comme l'a montré Virot [135], il y a un nombre optimal de processeurs associé au taux d'acceptation χ, qui lui-même dépend de la température du recuit. On distinguera donc les algorithmes avec un nombre fixe de processeurs des algorithmes dont le nombre de processeurs évolue dynamiquement [1].

Nombre de processeurs fixe, longueur des sous-chaînes contrôlée par le nombre de configurations acceptées

➔ *Synchronisation à la première configuration acceptée*

Roussel-Ragot et Dreyfus [102] ont développé un algorithme parallèle par essais multiples où les processeurs sont synchronisés dès qu'une configuration a été acceptée. Leur algorithme suit un schéma maître-esclaves dans lequel le maître dirige la recherche en choisissant la configuration qui sera acceptée, met à jour les p esclaves et garde également un historique de l'exploration. Ils introduisent deux modes de synchronisation des processeurs, suivant que l'on est à haute ou à basse température :

- À haute température, lorsque $\chi(T) > \frac{1}{p}$ tous les processeurs sont synchronisés après avoir évalué une configuration. Ce modèle synchrone n'est pas intéressant à basse température, car lorsque $\chi(T)$ est très bas les processeurs sont synchronisés alors qu'aucune configuration n'a été acceptée.
- À basse température quand $\chi(T) < \frac{1}{p}$ les processeurs évaluent de manière asynchrone des configurations et ne sont synchronisés que lorsque l'un des processeurs a trouvé une configuration acceptable. Ce mode de fonctionnement à basse température ne peut pas être retenu pour des hautes températures car cela favoriserait les configurations dont le temps d'évaluation est court puisque l'on garde la première configuration acceptée. On peut résoudre ce problème en synchronisant après avoir accepté plusieurs configurations, ce qui permettrait d'avoir un modèle de communications unique pour les modes à basse et haute température.

Cet algorithme garantit les mêmes conditions de convergence que l'algorithme séquentiel et aboutit à basse température à une accélération p (le nombre de processeurs esclaves).

➔ *Synchronisation au bout de plusieurs configurations acceptées*

Contrairement au schéma précédent les processeurs ne sont synchronisés que lorsque l'un d'entre eux a accepté M configurations avec $M \geq 1$. Chaque processeur produit une sous-chaîne formée de M configurations en utilisant l'algorithme décrit par la figure 1.6.

L'analyse de ce schéma de parallélisation par Virot [135] montre que M est un paramètre essentiel à cause des différents régimes de l'algorithme. À basse température comme peu de configurations sont acceptées on peut prendre M proche ou égal à un puisque l'on a peu de communications. De plus cela permet d'avoir une accélération maximale. Inversement à haute

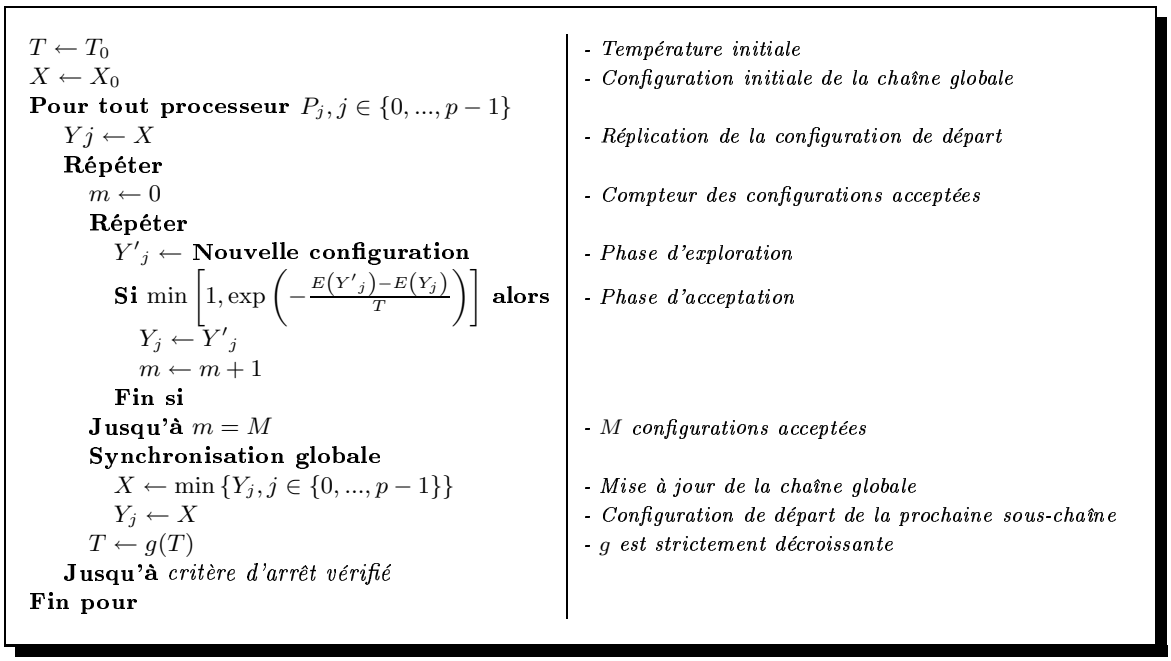


FIG. 1.6 – Algorithme exécuté par chaque processeur lors d'un recuit par essais multiples avec synchronisation au bout de M configurations acceptées.

température presque toutes les configurations sont acceptées, aussi pour ne pas avoir trop de communications et ne pas favoriser certaines configurations on prend M relativement grand. D'autre part Virot démontre qu'à basse température, avec $M = 1$, il existe un nombre optimal de processeurs, nombre qui croît lorsque le taux d'acceptation $\chi(T)$ décroît et qui dépend également des performances de la machine parallèle.

À partir du taux d'acceptation $\chi(T)$ on peut exprimer le temps moyen mis pour obtenir une configuration acceptable sur un processeur : $t = \frac{c}{\chi(T)}$, où c qui est une constante correspondant au temps mis pour choisir, évaluer et tester une configuration. On suppose de plus que le temps de synchronisation de p processeurs est de la forme $t_{sync} = a \cdot p$. Alors le nombre optimal de processeurs à la température T vaut :

$$P_{opt}(T) = \sqrt{\frac{t}{a}} = \sqrt{\frac{c}{a}} \cdot \sqrt{\frac{1}{\chi(T)}} \tag{1.33}$$

Nombre de processeurs évoluant dynamiquement

L'un des algorithmes parallèles dont le nombre de processeurs évolue est l'algorithme de « division » [1]. Dans cet algorithme p processeurs engendrent à haute température p sous-chaînes de longueur $\frac{L}{p}$, $L \in \mathbb{N}$ étant la longueur de la chaîne à température constante (descente par paliers). Lorsque les sous-chaînes sont toutes calculées, on choisit comme configuration de départ la configuration finale de l'une des p sous-chaînes. La configuration retenue est choisie de manière probabiliste comme dans l'algorithme « systolique » (voir 1.4.1), avec p configurations cette fois au lieu de deux. À chaque baisse de température, il faut que les sous-chaînes soient allongées, afin

de pouvoir atteindre le quasi-équilibre. C'est pourquoi on regroupe les processeurs au cours de l'algorithme, les processeurs d'un groupe engendrent alors une sous-chaîne en utilisant un schéma par essais multiples asynchrones. Les processeurs sont regroupés dès que le taux d'acceptation des configurations passe en dessous de 50%, et ceci jusqu'à obtenir une seule sous-chaîne qui soit de taille L.

1.4.3 Ferme de processeurs

Le principe de la parallélisation par ferme de processeurs est très simple. Un processeur maître engendre des configurations à partir de la configuration courante, soumettant chacune des configurations obtenues à un processeur esclave chargé de l'évaluer et de tester son acceptation. Lorsque l'un des esclaves accepte une configuration, il avertit le maître qui synchronise alors l'ensemble des esclaves et met à jour sa configuration courante, puis la recherche reprend. Ce type de parallélisation est aisé à mettre en œuvre et produit une chaîne de Markov identique à celle que produirait la version séquentielle.

Cette parallélisation n'est malheureusement pas toujours très efficace. En effet, à haute température pratiquement toutes les configurations testées sont acceptées ce qui se traduit par de nombreuses synchronisations, d'où un manque d'efficacité. Ce schéma n'est en fait intéressant qu'à basse température, et encore, à condition que le temps de génération d'une nouvelle configuration soit nettement inférieur au temps mis pour l'évaluer et la tester. Dans le cas contraire la parallélisation par ferme de processeurs n'a absolument aucun intérêt en raison du goulot d'étranglement que représente la génération des configurations.

1.4.4 Calculs spéculatifs

White *et al.* [137] ont conçu une version parallèle du recuit simulé en remarquant que l'on pouvait obtenir un recouvrement du temps de calcul d'une configuration et de celle qui la suit. Ce recouvrement est obtenu en calculant de manière anticipée la configuration suivante sans savoir si la configuration courante sera acceptée ou refusée. C'est-à-dire que pendant qu'un processeur évalue la configuration courante, deux autres processeurs évaluent chacun une configuration qui potentiellement la suivrait. L'un faisant l'hypothèse que la configuration courante sera acceptée alors que l'autre suppose qu'elle sera refusée. On peut donc schématiser cette parallélisation par un arbre binaire d'acceptation ou de refus des configurations. Lorsque la suite de décisions aboutit à une feuille de l'arbre, la configuration obtenue par ce processeur est envoyée à la racine de l'arbre puis le traitement reprend.

La principale question que pose cette parallélisation est de savoir quelle forme doit avoir l'arbre. Un arbre binaire équilibré formé de p processeurs permettra d'obtenir une accélération correspondant à la hauteur de l'arbre soit $log_2(p+1)$, mais cet arbre n'est pas adapté au recuit simulé. En effet, on sait que le taux d'acceptation des configurations varie avec la température, il est donc clair que l'arbre doit être déséquilibré en fonction de ce taux. Idéalement :

- Initialement à haute température, quand le taux $\chi(T)$ est très élevé, la branche correspondant à l'acceptation doit être très développée.
- Puis un rééquilibrage régulier a lieu, de sorte que l'arbre soit équilibré lorsque le taux d'acceptation $\chi(T)$ vaut 50%.

– Ensuite l'arbre se déséquilibre à nouveau pour obtenir à basse température un arbre dont la branche de refus est très développée.

Il faudrait donc faire évoluer la forme de l'arbre en fonction du taux d'acceptation associé à un palier de température. D'autre part, au cours de l'algorithme, le cheminement dans l'arbre se traduit par des branches qui n'ont plus aucune utilité. On peut également récupérer les processeurs du chemin de décision antérieur au processeur évaluant la configuration courante. Il serait donc intéressant de définir un algorithme permettant de récupérer dynamiquement les processeurs devenus inutiles afin de prolonger les sous-arbres vers lesquels le calcul peut évoluer.

Il faut remarquer que cette parallélisation permet d'obtenir une chaîne de Markov identique à celle que produirait le recuit séquentiel. De plus l'accélération que le calcul spéculatif peut apporter dépend fortement de l'importance du temps mis pour évaluer une configuration par rapport au temps mis pour la générer.

1.4.5 Parallélisme massif

Nous évoquons la parallélisation massive bien qu'elle ne soit pas aussi générique que les autres techniques de parallélisation. En effet, cette technique n'est applicable que pour des problèmes ayant certaines caractéristiques au niveau de l'espace de recherche et de la fonction d'énergie. Nous allons illustrer ce propos en présentant le cadre dans lequel le recuit simulé discret peut être massivement parallélisé. Ensuite nous parlerons de la parallélisation de l'algorithme basé sur l'équation de la diffusion, qui, nous le rappelons, requiert une fonction d'énergie dont le gradient est défini.

Recuit simulé discret

En analyse d'images les approches markoviennes associées avec les techniques statistiques de l'estimation bayésienne conduisent à résoudre des problèmes tels que la restauration d'images [43] ou la segmentation [82] en optimisant une fonction d'énergie. Dans ce cadre, ainsi que dans certains problèmes de physique, on a un espace de recherche qui est de la forme $\Omega = \Lambda^{|S|}$, avec $S \in \mathbb{N}^+$ qui est l'ensemble des sites (les pixels en imagerie) et Λ qui est l'ensemble des étiquettes possibles pour chaque site (les niveaux de gris). C'est-à-dire que Ω représente toutes les configurations possibles d'un champ de variables aléatoires $X = (X_s)_{s \in S}$ où chaque variable X_s prend sa valeur dans Λ. Le champ X est supposé markovien relativement à un système de voisinage $\mathcal{V} = \{\mathcal{V}_s \subseteq S, s \in S\}$, i.e. :

① $\forall x \in \Omega, P(X = x) > 0$

② $\forall s \in S$ et $\forall x \in \Omega, P(X_s = x_s \mid X_r = x_r, r \neq s) = P(X_s = x_s \mid X_r = x_r, r \in \mathcal{V}_s)$

La distribution de ce champ X suit alors une distribution de Gibbs avec une fonction d'énergie se décomposant en une somme de fonctions de potentiel d'interactions locales V_c définies sur les cliques $c \in \mathcal{C}$ associées à $\mathcal{V}$ (une clique est un ensemble de sites voisins deux à deux, singletons compris) : $\forall x \in \Omega, E(x) = \sum_{c \in \mathcal{C}} V_c(x)$.

De plus l'expression des probabilités conditionnelles locales [43] est alors de la forme :

$$P(X_s = x_s \mid X_r = x_r, r \neq s) = \frac{1}{Z_s} \exp\left(- \sum_{c \in \mathcal{C}_s} V_c(x) \right) \tag{1.34}$$

où $Z_s = \sum_{x' \in \Omega} \exp\left(-\sum_{c \in \mathcal{C}_s} V_c(x')\right)$, $\mathcal{C}_s$ est l'ensemble des cliques contenant le site s, et x' représente toute configuration qui diffère de x uniquement au niveau du site s.

Comme deux configurations x et y seront dites voisines si elles ne diffèrent qu'en un seul site s, le calcul de la variation d'énergie correspondant au passage de x à y est local et ne dépend que du site où elles diffèrent et de son voisinage. Ces caractéristiques locales conduisent à une parallélisation massive en remettant à jour simultanément plusieurs sites de l'image, chaque site étant associé à un processeur chargé de calculer localement la variation d'énergie. L'unique contrainte est que dans le cas de variables discrètes on ne peut remettre à jour simultanément que des sites non voisins. La simplicité de ce schéma fait que parmi toutes les parallélisations possibles du recuit simulé en imagerie [82] la parallélisation massive fut étudiée en premier.

Équation de la diffusion

Par définition l'algorithme d'optimisation basé sur l'équation de la diffusion (voir figure 1.2) est massivement parallèle. En effet, si on peut calculer localement la dérivée partielle de la fonction d'énergie par rapport à chaque composante du vecteur de variables à optimiser, alors on pourra remettre à jour en parallèle chacune des composantes. Aussi, en distribuant le vecteur de variables à raison d'une variable par processeur, on obtient une implantation massivement parallèle (cf. la figure 1.7). Les seules communications consistent à transmettre les données nécessaires à chaque processeur pour calculer la dérivée partielle. Ce mode de parallélisation a ainsi été utilisé par Roysam *et al.* [103, 104] pour résoudre un problème de reconstruction d'images à partir de données bruitées (images médicales (TEP) et images radar).

$t = 0$	- *Initialisation du temps*
$T_t \leftarrow T_0$	- *Température initiale*
$\Delta P_t \leftarrow \Delta P_0$	- *Pas de temps initial*
$X_t \leftarrow X_0$	- *Configuration de départ*
Répéter	
Pour tout processeur $P_i, i \in \{1, ..., N\}, N \in \mathbb{N}^+$	- *L'entier N est le nombre de variables à optimiser*
$\Delta w_{t,i} \leftarrow \mathcal{N}(0, \Delta P_t)$	- *Mouvement brownien*
$x_{t+\Delta P_t,i} \leftarrow x_{t,i} - \frac{\partial E(x_t)}{\partial x_i} \cdot \Delta P_t + \sqrt{2 \cdot T_t} \cdot \Delta w_{t,i}$	
Fin pour	
$T_{t+\Delta P_t} \leftarrow g(T_t)$	- *g et h sont deux fonctions*
$\Delta P_{t+\Delta P_t} \leftarrow h(\Delta P_t)$	*strictement décroissantes*
$t \leftarrow t + \Delta P_t$	
Jusqu'à *critère d'arrêt vérifié*	

FIG. 1.7 – Algorithme massivement parallèle de l'équation de la diffusion.

Chapitre 2

Recherche tabou

2.1 Introduction

La recherche tabou [45, 46] est une heuristique de haut niveau, ou méta-heuristique, introduite pour guider des heuristiques d'optimisation locale, afin de pouvoir explorer l'espace de recherche sans être piégé par les minima locaux. Cette technique d'optimisation est très proche du recuit simulé dans le sens où elle recherche également la configuration suivante dans un voisinage autour de la configuration courante. La recherche tabou a été appliquée à de nombreux problèmes standards tels que le coloriage de graphe ou encore le problème du sac à dos multidimensionnel [90]. Comme problème réel on peut citer l'assignation de fréquences dans un réseau de radiocommunications mobiles [55].

2.2 Description de l'algorithme de la recherche tabou

On a vu que pour ne pas rester bloqué par un minimum local, le recuit simulé accepte de façon probabiliste des configurations accroissant temporairement l'énergie totale du système. La recherche tabou procède de même, mais de manière déterministe en prenant comme nouvelle configuration celle qui a le coût le plus faible dans le voisinage de la configuration courante, celle-ci pouvant éventuellement être moins bonne que la configuration courante. Cette stratégie présente cependant l'inconvénient de pouvoir se bloquer dans des cycles, i.e. revenir sur des configurations déjà rencontrées. C'est pourquoi pour ne pas boucler, et permettre de sortir des pièges formés par les minima locaux, on contraint la recherche en classant certains mouvements (le passage d'une configuration à une autre) comme tabous.

En résumé la méthode tabou est une procédure itérative guidant le choix des mouvements en utilisant les heuristiques suivantes :

- assurer la décroissance de la fonction de coût que l'on cherche à minimiser, en sélectionnant un mouvement faisant décroître l'énergie ;
- si aucun mouvement ne permet de faire diminuer le coût, on choisit le mouvement entraînant la remontée en énergie la plus faible ;
- on maintient une liste de mouvements ou configurations tabous durant une certaine période afin d'interdire le retour vers une configuration déjà rencontrée.

On cherche à minimiser une fonction de coût $C(x)$ (équivalent à la fonction d'énergie $E(x)$ du recuit simulé) définie sur Ω l'espace des configurations admissibles du problème. On définit implicitement une notion de voisinage, avec $M(x)$ l'ensemble des mouvements applicables en x. En effet une configuration y appartient au voisinage $V(x)$ de x à condition que l'on puisse passer de x à y en un seul mouvement $m \in M(x)$ $(y = m(x))$.

Soit LT la liste des mouvements tabous et $Cand(x)$ l'ensemble des mouvements candidats pouvant être appliqués à x pour obtenir la configuration suivante, i.e. l'ensemble des voisins de x qui ne sont pas tabous $(V(x) \setminus \{m(x) \mid m(x) \in LT \text{ et } m \in M(x)\})$. On peut alors formaliser la recherche tabou simple par l'algorithme de la figure 2.1.

$X \leftarrow X_0$ | *- Configuration initiale*
$X_{min} \leftarrow X$ | *- Configuration de coût minimal*
$C_{min} \leftarrow C(X_{min})$ | *- Coût minimal*
$LT \leftarrow \emptyset, k \leftarrow 0$ | *- Liste tabou, compteur*

Tant que $k < MAX\_ITERATIONS$ **faire**
  $k \leftarrow k + 1$
  $Cand(X) \leftarrow V(X) \setminus \{m(X) \mid m(X) \in LT$ **et** $m \in M(X)\}$
  $X \leftarrow X' \in Cand(X) \mid C(X') = \min_{Y \in Cand(X)} (C(Y))$
  Si $C(X) < C_{min}$ **alors** $X_{min} \leftarrow X, C_{min} \leftarrow C(X)$ **Fin si**
$LT \leftarrow LT \cup \{X\}$
Fin tant que

FIG. 2.1 – Algorithme de la recherche tabou.

La liste tabou LT implantant les contraintes guidant la recherche, les configurations engendrées sont fortement dépendantes de la définition d'un mouvement tabou, de la structure de LT, et de la manière dont la liste est mise à jour. Il est donc clair que le choix de la liste tabou et de la procédure de mise à jour est essentiel.

2.3 Liste tabou

L'objectif de la liste tabou est d'interdire le retour vers des configurations déjà rencontrées. Glover définit dans [45] deux heuristiques d'implantation de la liste tabou :

① interdire de retourner en arrière en définissant une liste de longueur fixe T : le dernier mouvement est alors mémorisé en tête de liste tandis que le mouvement fait il y a T itérations disparaît ;

② permettre le retour vers une configuration déjà rencontrée à condition que le meilleur mouvement qui n'est pas tabou et permettant de quitter cette configuration ne soit pas le même que précédemment, c'est-à-dire qu'à ce moment là, on ne choisira pas une deuxième fois le même mouvement.

Ces implantations sont néanmoins peu utilisées : en pratique on ne choisit de stocker dans la liste tabou qu'un certain nombre d'informations correspondant à des attributs permettant de

caractériser le mouvement considéré. Dans le cadre du problème du voyageur de commerce cela reviendrait, en n'utilisant qu'un attribut, à ne mémoriser que l'arc ayant été ajouté ou supprimé. L'utilisation d'attributs présente cependant l'inconvénient de rendre tabous tous les mouvements dont les attributs se trouvent dans la liste tabou. Pour réduire le nombre de mouvements tabous il existe deux possibilités : définir d'autres attributs ou utiliser une fonction d'aspiration comme nous le verrons ci-dessous.

Une liste de taille non bornée permet d'être sûr de ne pas reconsidérer certaines configurations. Cependant conserver en mémoire toutes les configurations engendrées n'est bien entendu pas possible vu la taille mémoire que cela nécessiterait, mais également à cause du coût des procédures de recherche pour savoir si un mouvement est tabou. C'est pourquoi un des éléments importants lors de la définition de la liste est sa taille T. En effet, comme le fait remarquer Glover, il faut faire un compromis entre une grande taille évitant les cycles et une taille réduite permettant d'avoir un plus grand choix pour guider la recherche des configurations. De nombreux travaux [126] ont étudié la définition d'heuristiques permettant de fixer la taille de la liste. Ceux-ci mettent clairement en évidence qu'une taille dynamique est préférable à une taille statique. Battiti *et al.* [15] définissent ainsi un schéma de recherche tabou, le *Reactive Tabu Search*, dont l'une des caractéristiques est d'adapter automatiquement la taille de la liste tabou. Leur schéma réagit (d'où le terme « *Reactive* ») aux configurations qui sont répétées durant la recherche : T croît si une configuration est reconsidérée et décroît si aucune répétition n'a lieu durant un certain intervalle de temps. Il faut noter que la liste tabou est parfois également appelée mémoire à court terme du fait qu'elle ne conserve que les configurations correspondant aux T dernières itérations.

2.4 Raffinements

Plusieurs raffinements de la recherche tabou simple ont été proposés [45, 46]. Tout d'abord l'oubli du caractère tabou d'un mouvement si celui-ci est « favorable », c'est-à-dire s'il permet d'aboutir à la meilleure configuration obtenue jusqu'à présent. Le second raffinement consiste à exploiter l'historique de la recherche par l'intermédiaire de mémoires à moyen et long terme permettant respectivement d'intensifier et diversifier la recherche. La dernière possibilité qui est présentée est l'utilisation d'un schéma de choix de mouvements basé sur des probabilités.

2.4.1 Recherche tabou évoluée

Fonction d'aspiration

Lorsque les mouvements tabous sont caractérisés par l'intermédiaire d'attributs, tous les mouvements ayant les mêmes attributs qu'un mouvement tabou sont également classés tabous. On peut donc avoir un mouvement « favorable » ou « attractif » qui est tabou. Pour mettre fin à cette situation on oublie son statut tabou en le retirant de la liste des mouvements tabous. Dans cette optique on définit la fonction d'aspiration $A(m, x)$: on dira que le mouvement m a passé le niveau d'aspiration en x si $C(m(x)) \leq A(m, x)$, auquel cas on supprimera x de la liste tabou. Lors de la définition de $A(m, x)$ il faut faire attention à éviter de produire des cycles : en général $A(m, x)$ est défini comme étant le coût le plus faible rencontré jusqu'à présent.

Utilisation de l'historique de recherche

Pour exploiter l'information acquise au cours de l'évolution de l'algorithme, on définit des mémoires supplémentaires ayant une vision plus globale de l'espace des configurations : les mémoires à moyen et long terme. La mémoire à moyen terme sert pour intensifier la recherche vers des zones « favorables » de l'espace de recherche tandis que la mémoire à long terme diversifie la recherche en favorisant les zones encore inexplorées.

L'intensification

L'intensification vient du constat que les meilleures configurations ont des caractéristiques communes qui sont également utilisées dans le cadre des algorithmes génétiques. Dans cette optique, l'intensification recherche dans un premier temps, les caractéristiques communes aux « bonnes » solutions, puis oriente la recherche vers des zones de l'espace de recherche dont les configurations présentent les caractéristiques mises en évidence. Ce processus est répété régulièrement en utilisant une mémoire (la mémoire à moyen terme) permettant d'enregistrer et d'analyser les meilleures configurations rencontrées durant un certain laps de temps. La détermination des caractéristiques communes se fait en général à l'aide d'une matrice des bonnes solutions ou de techniques avancées d'analyse discriminante définissant une collection d'inégalités ou de conditions logiques. Comme méthodes d'intensification on peut citer :

- la pénalisation des mouvements conduisant à des configurations ne présentant pas les caractéristiques désirées en modifiant provisoirement la fonction de coût ;
- la restriction temporaire de la recherche en contraignant certaines variables des configurations ($x = (x_1, x_2, ..., x_N), N \in \mathbb{N}^+$ étant le nombre de variables) à avoir leur valeur dans un intervalle de taille restreinte.

La diversification

Le but de la diversification est l'inverse de celui de l'intensification : il est de guider l'exploration vers de nouvelles zones de l'espace de recherche contenant des configurations qui sont très différentes de celles rencontrées jusqu'à présent. Pour cela on mémorise des informations sur les configurations visitées tout au long de la recherche, d'où d'ailleurs le qualificatif de mémoire à long terme, afin de trouver les caractéristiques (des configurations) qui ne sont pas souvent rencontrées. Pour implanter la diversification, on utilise notamment les principes suivants :

- pénaliser les mouvements aboutissant à des configurations fréquemment rencontrées ;
- interdire les mouvements dont la fréquence d'occurrence dépasse un certain seuil ;
- utiliser une liste tabou longue activée périodiquement, implantant des conditions très strictes.

L'objectif est d'obtenir un point de départ dans une zone non explorée en utilisant l'historique de la recherche, contrairement à une méthode aléatoire.

Algorithme

L'introduction dans l'algorithme de recherche tabou simple (figure 2.1) d'un critère d'aspiration et de l'historique de recherche se traduit par la version évoluée de la recherche tabou (figure 2.2).

$X \leftarrow X_0, X_{min} \leftarrow X$ | *- Configuration initiale ; configuration de coût minimal*
$C_{min} \leftarrow C(X_{min})$ | *- Coût minimal*
$LT \leftarrow \emptyset, k \leftarrow 0$ | *- Liste tabou ; compteur*

Pour i **de** 1 **à** $MAX\_DIVERSIFICATIONS$ **faire**
  Pour j **de** 1 **à** $MAX\_INTENSIFICATIONS$ **faire**
    Tant que $k < MAX\_ITERATIONS$ **faire**
      $k \leftarrow k+1$, **soit** $X' \in V(X) \mid X' = m(X), m \in M(X)$ **et** $C(X') = \min_{Y \in V(X)}(C(Y))$
      Si $X' \in LT$ **alors**
        Si $C(X') \leq A(m, X)$ **alors** $X \leftarrow X'$
        sinon soit $X'' \in V(X)$ **la configuration la plus ancienne de** LT, $X \leftarrow X''$ **Fin si**
      sinon $X \leftarrow X'$ **Fin si**
      Si $C(X) < C_{min}$ **alors** $X_{min} \leftarrow X, C_{min} \leftarrow C(X)$ **Fin si**
      $LT \leftarrow LT \cup \{X\}$, **Mise à jour des mémoires**
    Fin tant que
    Intensification(X_{min}**,Mémoire à moyen terme)**
  Fin pour
  Diversification(Mémoire à long terme)
Fin pour

Fig. 2.2 – Algorithme de la recherche tabou évoluée.

2.4.2 Tabou probabiliste

L'incorporation d'un aspect probabiliste au niveau du choix des mouvements permet de diversifier la recherche de manière efficace en évitant les trop nombreuses opérations au niveau mémoire et évaluation d'une diversification plus systématique. Ce gain est néanmoins compensé par le fait que dans le cadre probabiliste des errements sont possibles, ce qui n'est pas le cas dans un cadre plus systématique. Pour calculer les probabilités, on utilise les connaissances acquises sur le problème, initialement ou au cours de l'évolution de la recherche, en associant à chaque mouvement un poids. Le poids de chaque mouvement peut évoluer durant la recherche suivant les principes suivants :

① Les mouvements « attractifs », i.e. menant à une configuration qui est de coût plus faible que la meilleure configuration rencontrée jusqu'à présent, doivent avoir une probabilité plus élevée ;

② Un mouvement tabou doit voir sa probabilité évoluer en fonction du temps écoulé depuis qu'il est entré dans la liste tabou, i.e. plus le caractère tabou est récent plus la probabilité est faible ;

③ Le niveau d'aspiration traduisant les performances de la recherche, tout mouvement passant le seuil d'aspiration doit voir sa probabilité augmenter.

Deux techniques de choix de mouvement sont données dans [45] :

① Utiliser un ensemble de mouvements candidats $Cand(x)$ défini à partir de l'ensemble des mouvements applicables $M(x)$, et calculer pour chaque mouvement $m \in Cand(x)$ son

poids, puis sa probabilité comme suit :

$$P(m) = \frac{w(m)}{\sum_{m' \in Cand(x)} w(m')}, \text{ où } w(m) \text{ est le poids de } m.$$

② Modéliser le choix des mouvements comme le résultat de tournois successifs entre les mouvements. Cette technique présente l'inconvénient de dépendre de l'ordre dans lequel les mouvements sont testés. Pour ne pas favoriser certains mouvements on fait croître la probabilité des mouvements entrant en compétition tardivement.

2.5 Comparaison entre la recherche tabou et le recuit simulé

Les études comparatives entre la recherche tabou et le recuit simulé semblent montrer que l'intérêt d'une méthode par rapport à l'autre varie suivant le point de vue selon lequel on se place. Ainsi pour de nombreux travaux le recuit simulé est meilleur en nombre d'évaluations de fonctions, tandis que dans [16] les auteurs considèrent que la recherche tabou donne de meilleurs résultats dans le cas de problèmes complexes ou si l'on désire une solution très proche de la configuration optimale.

Parmi les différences entre les deux méthodes, on peut noter les points suivants :

- la recherche tabou est une méthode utilisant intensivement la mémoire en effectuant un apprentissage, de telle sorte que l'historique de la recherche conditionne l'évolution de l'algorithme. À l'opposé, le recuit simulé ne tient aucun compte du passé lors du choix de la nouvelle configuration ;
- le recuit simulé peut accepter un mouvement après test d'un seul voisin, alors que la recherche tabou n'accepte un mouvement qu'après avoir évalué toutes les configurations du voisinage.

2.6 Parallélisations

Parmi les études de la parallélisation de la recherche tabou l'une des plus intéressante est la thèse de Taillard [127]. En effet il y développe une classification des différentes approches possibles pour paralléliser la recherche tabou :

① *Parallélisation de l'évaluation de la fonction à optimiser*, cette technique fait partie des parallélisations liées au problème ;

② *Parallélisation par décomposition du problème* (un partitionnement des données) ;

③ *Parallélisation de l'examen du voisinage*, i.e. de l'évaluation des mouvements candidats parmi lesquels sera sélectionné le prochain mouvement ;

④ *Parallélisation par recherches tabou multiples.*

Nous allons approfondir les deux dernières approches en raison de leur généricité permettant de les appliquer à des problèmes très variés.

2.6.1 Exploration distribuée du voisinage

On a vu lors de la présentation de la recherche tabou qu'à chaque itération de l'algorithme, le prochain mouvement est choisi dans un ensemble de mouvements candidats. On peut donc distribuer sur un ensemble de processeurs l'exploration de l'ensemble des mouvements candidats afin de déterminer le mouvement optimal. Le problème qui reste à résoudre est alors de bien équilibrer la distribution des mouvements candidats de sorte que la charge de calcul soit répartie équitablement entre tous les processeurs. Si on fait l'hypothèse que le temps mis pour évaluer une configuration est approximativement constant, on peut distribuer l'exploration de l'ensemble des mouvements applicables de différentes manières :

① *Distribution par énumération de l'ensemble des mouvements candidats*
Une première stratégie est d'utiliser un processeur maître chargé d'énumérer l'ensemble des mouvements candidats, le maître envoyant, à la demande, les mouvements aux processeurs esclaves. Puis lorsque l'énumération est terminée, chaque processeur calcule l'optimum de ses mouvements. On garde alors le meilleur de tous les optima locaux. Une seconde technique, reposant sur un ordonnancement implicite des mouvements de l'ensemble des candidats, consiste pour chaque processeur $P_k, 0 \leq k \leq p-1$ à évaluer les mouvements se trouvant aux positions $p \cdot n + k, n \geq 0$. C'est-à-dire que le processeur P_k effectue une énumération des mouvements dont la position est k modulo p : l'optimum global est alors obtenu de la même manière que précédemment. Ces techniques sont intéressantes car elles permettent d'obtenir une accélération quasi-linéaire, mais également d'explorer précisément le voisinage dans des problèmes que nous n'aurions pu explorer que par échantillonnage en séquentiel.

② *Échantillonnage distribué du voisinage*
Lorsque l'ensemble des mouvements candidats est relativement grand, il peut être intéressant que chaque processeur effectue un échantillonnage en tirant aléatoirement un nombre fixé de mouvements. Ensuite chaque processeur détermine le meilleur mouvement parmi les échantillons qu'il a tirés, l'optimum global est alors le meilleur mouvement parmi l'ensemble des optima obtenus localement. Cette technique d'évaluation du voisinage permet d'éviter l'énumération des mouvements qui peut s'avérer coûteuse, mais il est possible que l'on ait des configurations redondantes sur plusieurs processeurs.

2.6.2 Recherches tabou multiples

Cette approche est identique à celle utilisée dans le cadre de recuits simulés multiples, c'est-à-dire que l'on lance en parallèle des recherches tabou sur différents processeurs. On peut faire évoluer les différentes recherches indépendamment les unes des autres ou les faire interagir. L'idée à la base de ce schéma est que chaque processeur effectue sa recherche dans une zone de l'espace des configurations différente de celles des autres processeurs.

Recherches tabou indépendantes

Pour faire plusieurs recherches tabou, il y a deux possibilités : prendre des points de départ différents pour chaque processeur et/ou prendre des paramètres différents pour chaque instance de l'algorithme.

- Taillard a mis en évidence expérimentalement dans [126] que dans le cadre d'une recherche tabou simple séquentielle, on a une vitesse de convergence au bout de L itérations qui est de la forme :

$$P(X_L \notin \Omega_{min}) \approx e^{-\beta L}, \beta > 0 \tag{2.1}$$

  On peut donc espérer obtenir une vitesse de convergence identique en p recherches tabou parallèles de $\frac{L}{p}$ itérations. Il est également intéressant de noter la similitude avec la vitesse de convergence des recuits multiples indépendants (section 1.4.1).
- Parmi les paramètres que l'on peut modifier d'un processeur à un autre on a la longueur de la liste tabou, la liste des mouvements candidats, ou le nombre d'itérations avant qu'une phase d'intensification ou de diversification ne soit lancée, etc. Néanmoins il n'existe aucune méthode précise pour choisir ces paramètres, en général ceux-ci sont fixés à l'issue de tests.

Recherches tabou avec interactions

L'une des améliorations que l'on peut apporter au schéma précédent est d'introduire des communications entre les processeurs durant le processus de recherche. Plusieurs algorithmes parallèles sont possibles dans ce type de parallélisation :

- utiliser un schéma maître/esclave dans lequel le maître centralise les informations ; il peut éventuellement maintenir la liste tabou, les mémoires à moyen et à long terme. Les solutions obtenues par les esclaves sont alors transmises au maître par l'intermédiaire de communications synchrones, puis à partir des résultats collectés, le maître réinitialise les esclaves qui effectuent alors une nouvelle recherche. C'est par exemple, un schéma de ce type qui est utilisé par Niar *et al.* [90] pour résoudre le problème du sac à dos multidimensionnel en variables 0-1. Dans leur algorithme le maître contrôle les esclaves en modifiant dynamiquement les paramètres spécifiant la recherche tabou ;
- on peut également imaginer un schéma où tous les processeurs lancent une recherche tabou avec les mêmes paramètres, et échangent régulièrement des informations afin qu'ils explorent des zones distinctes de l'espace de recherche.

La ligne directrice de ce type de parallélisation est que la diversification des recherches est un aspect important de la recherche tabou, car elle conditionne apparemment fortement la qualité des différentes solutions obtenues.

Ces parallélisations sont très élémentaires. C'est pourquoi il ne nous a pas semblé intéressant de les retenir dans notre étude.

Chapitre 3

Algorithmes évolutionnaires

3.1 Introduction

Le terme « algorithmes évolutionnaires » désigne l'ensemble des algorithmes modélisant le processus de l'évolution et de la sélection naturelle, basé sur le néodarwinisme, théorie résultant de l'extension par la description génétique des mécanismes de l'hérédité, de la théorie de l'évolution des espèces développée par Darwin au cours du $19^{\text{ème}}$ siècle. L'intérêt porté à cette modélisation vient du fait que la vision de l'évolution donnée par le néodarwinisme, correspond à un problème d'optimisation. En effet, l'idée d'auto-adaptation sous-jacente à cette théorie est que, sous l'influence de conditions extérieures, les caractéristiques des êtres vivants se modifient progressivement lors de la phase de reproduction. Ceci permet d'aboutir à des générations mieux adaptées aux conditions complexes de leur environnement, et maximisant donc leur probabilité de survie. Les transformations au niveau chromosomique se traduisent, sur une certaine période, par l'émergence des espèces qui ont survécu en transmettant leur patrimoine génétique aux générations futures.

Concrètement, un algorithme évolutionnaire repose sur le concept de population d'individus, chaque individu représentant une solution potentielle du problème. L'algorithme fait évoluer la population au moyen de règles très précises, afin de simuler le processus de l'évolution naturelle. L'évolution de la population se traduit par l'émergence d'individus optimisant une fonction d'évaluation (en anglais, fonction de « fitness »), évaluant l'adaptation d'un individu à « l'environnement ».

Le principal attrait des algorithmes évolutionnaires est que ce sont des méthodes d'optimisation particulièrement efficaces pour des problèmes très irréguliers, définis par de nombreuses variables et un espace de recherche très étendu avec une structure mal définie. D'autre part, ces algorithmes présentent l'avantage d'effectuer une recherche à partir d'une population, soit une recherche multipoint, alors que d'autres algorithmes tels que le recuit simulé ou la recherche tabou font une recherche point par point.

Parmi ces algorithmes, les plus connus sont sans conteste les algorithmes génétiques introduits par Holland [60], et plus précisément, pour l'optimisation de variables, par De Jong [37]. Les deux autres grands types d'algorithmes évolutionnaires que l'on distingue en général sont la programmation évolutionnaire [41, 39] et les stratégies d'évolution développées en Allemagne

par Rechenberg et Schwefel [94, 95, 113, 116]. Le lecteur intéressé trouvera dans l'ouvrage de Bäck [13] une revue complète des différents aspects de ces trois classes d'algorithmes.

Plus récemment, Storn et Price [120, 121], ont utilisé des concepts des algorithmes génétiques et des stratégies d'évolution pour définir l'évolution différentielle, un algorithme qui a montré d'excellentes performances en termes de robustesse et d'efficacité lors de la minimisation de la suite de fonctions réelles multivariables du concours ICEC'96 [121].

Dans la suite nous allons nous intéresser en particulier à cet algorithme, ainsi qu'aux algorithmes génétiques et aux stratégies d'évolution. Dans un premier temps nous présenterons les grandes lignes d'un algorithme évolutionnaire, puis nous verrons les particularités propres à chaque type d'algorithme considéré.

3.2 Fondements et schéma d'un algorithme d'évolutionnaire

Dans la nature, les caractéristiques d'un individu sont codées par une chaîne de caractères qui forme un chromosome. Chaque information contenue dans le chromosome est codée par un gène correspondant à un ou plusieurs caractères. L'ensemble des gènes d'un individu est appelé son génotype et l'ensemble des caractéristiques qui sont codées son phénotype. Par analogie avec la nature deux niveaux de représentation sont possibles pour un individu :

- le niveau « génotypique », dans lequel un individu correspond à la représentation chromosomique d'une solution, en général dans ce cas on utilise l'alphabet binaire $\{0, 1\}$ pour représenter les gènes ;
- le niveau « phénotypique », où un individu représente directement une solution du problème.

On peut illustrer ce propos avec l'exemple suivant : considérons comme espace de recherche l'ensemble des triplets

$$(x_1, x_2, x_3) \in \{0, ..., 100\} \times \{0, ..., 200\} \times \{0, ..., 13\}$$

Le niveau « phénotypique » consiste alors à prendre pour individu le triplet (x_1, x_2, x_3), alors que le niveau « génotypique » avec un code binaire classique de 19 bits, code par exemple le triplet $(93, 171, 9)$ par la suite de bits 1011101|10101011|1001.

Parmi les algorithmes que nous allons étudier, on peut citer les algorithmes génétiques comme algorithmes opérant sur le génotype et les stratégies d'évolution pour le niveau « phénotypique ».

L'évaluation d'un individu donne une information qualitative sur l'individu par rapport à l'environnement considéré, i.e. le problème d'optimisation considéré. Aussi la fonction d'évaluation est-elle définie à partir du coût associé à l'individu.

Pour simuler l'évolution naturelle on applique en général successivement trois opérateurs à la population : le croisement (ou recombinaison) et la mutation, qui sont des opérateurs génétiques assurant la reproduction, calqués tous deux sur leur pendant biologique, puis la sélection (basée sur l'évaluation) pour former la nouvelle population. Chaque opérateur a un rôle différent : ainsi la sélection doit guider la recherche en favorisant la reproduction des meilleurs individus (ayant la meilleure évaluation), tandis que la mutation doit introduire de nouvelles informations en perturbant les individus et que le croisement doit échanger des informations entre plusieurs individus, ce qui doit permettre d'explorer l'espace de recherche.

Voyons maintenant les différentes étapes d'un algorithme évolutionnaire (figure 3.1). Soit $P(t)$ la population à la génération t et supposons qu'elle soit formée par μ individus $a_j(t), j \in \{1, ..., \mu\}$. La première opération consiste à générer aléatoirement la population de départ $P(0)$, puis à calculer la valeur de la fonction d'évaluation Φ pour chaque individu. Ensuite, l'algorithme engendre une suite de populations jusqu'à ce qu'un critère d'arrêt prédéfini soit vérifié.

$t = 0$ | - *Compteur des générations*
Initialisation$(P(t))$ | - $P(t) = \{a_1(t), ..., a_\mu(t)\}$
Évaluation$(P(t))$ | - $\{\Phi(a_1(t)), ..., \Phi(a_\mu(t))\}$
Tant que *critère d'arrêt non vérifié* **faire**
  $t = t + 1$
  $P'(t) =$ **Croisement**$(P(t-1))$ **(ou Recombinaison)**
  Mutation$(P'(t))$
  Évaluation$(P'(t))$
  $P(t) =$ **Sélection**$(P'(t) \cup Q(t))$, $Q(t) \in \{\emptyset, P(t-1)\}$
Fin tant que

FIG. 3.1 – Schéma d'un algorithme évolutionnaire.

Suivant l'algorithme évolutionnaire considéré, la sélection peut opérer uniquement sur la population des descendants ($Q(t) = \emptyset$), soit en tenant également compte de la population des parents ($Q(t) = P(t-1)$). Cette dernière alternative qui donne un caractère « élitiste » à la sélection est en général privilégiée, puisque la conservation du meilleur individu rencontré est alors garantie. D'ailleurs, dans le cas de l'algorithme génétique canonique [60], Rudolph [107] a montré que l'on n'est pas toujours assuré de la convergence vers l'optimum global, mais qu'en revanche un algorithme génétique élitiste converge toujours vers le meilleur individu possible.

Par souci de clarté, nous supprimerons dans les sections qui suivent le compteur de génération dans la notation des populations et des individus, dès lors que cette information ne sera pas utile (par exemple $P(t)$ sera simplement noté P).

3.3 Algorithmes génétiques (AGs)

Les algorithmes génétiques sont sans doute les algorithmes évolutionnaires les plus connus et les plus couramment utilisés. Leur formulation originelle a été donnée par J.H. Holland dans son ouvrage *Adaptation in Natural and Artificial Systems* [60], paru en 1975. Bien que Holland évoquât déjà les nombreux domaines d'application potentiels des algorithmes génétiques, ce fut la thèse de De Jong [37], soutenue la même année, qui mit plus particulièrement en lumière l'adéquation des algorithmes génétiques à la résolution de problèmes d'optimisation de variables. Dans son travail, De Jong introduisit également de nombreux raffinements de l'algorithme génétique canonique défini par Holland. Ces raffinements sont devenus par la suite des principes de base, à considérer lors de toute application des algorithmes génétiques à un problème réel.

Nous allons présenter les algorithmes génétiques dans leur formulation classique, utilisée notamment pour l'optimisation de variables continues. Pour certaines applications, il est cependant nécessaire d'adapter les algorithmes classiques à la structure particulière de l'espace de recherche, ce qui se traduit par la définition d'algorithmes génétiques spécialisés [84].

3.3.1 Représentation et fonction d'évaluation

Dans leur forme classique, les algorithmes génétiques utilisent une représentation binaire pour coder les individus, i.e. $a_j = (a_{j,1}, ..., a_{j,l})$ avec $a_j \in \{0,1\}^l, l \in \mathbb{N}$ et $j \in \{1, ..., \mu\}$. On peut néanmoins utiliser ce cadre booléen pour optimiser des variables réelles, par l'intermédiaire d'une fonction de décodage, assurant la transition de l'espace de recherche binaire vers l'espace de recherche continu $\Omega = \mathbb{R}^n$ (pour n variables réelles).

Supposons que le problème d'optimisation considéré ait une fonction de coût $C : \Omega \to \mathbb{R}$, où $\Omega = \prod_{i=1}^{n} [inf_i; sup_i] \subset \mathbb{R}$ et $inf_i < sup_i$. L'utilisation du cadre booléen pour optimiser la fonction se fait alors en découpant un individu a_j en autant de segments que de variables à optimiser, chacun de longueur l_s telle que $l = n \cdot l_s$. Chaque segment $(a_{j,i1}, ..., a_{j,il_s})$ code alors une variable x_i, et on passe du code binaire à une valeur réelle dans $[inf_i; sup_i]$ en échantillonnant régulièrement autant de valeurs que d'entiers que permet de coder le segment (i.e. 2^{l_s}). Dans cette optique on utilise une fonction de décodage injective $\Gamma^i : \{0,1\}^{l_s} \to [inf_i; sup_i]$ [37, 84] de la forme

$$\Gamma^i(a_{j,i1}, ..., a_{j,il_s}) = inf_i + \frac{sup_i - inf_i}{2^{l_s} - 1} \cdot \left(\sum_{k=0}^{l_s - 1} a_{j,i(l_s - k)} \cdot 2^k \right) \tag{3.1}$$

On combine ensuite les différentes fonctions de décodage Γ^i pour définir la fonction de décodage d'un individu, $\Gamma : \{0,1\}^l \to \Omega = \prod_{i=1}^{n} [inf_i; sup_i], \Gamma = \Gamma^1 \times ... \times \Gamma^n$, d'où le réel $x_j = \Gamma(a_j)$ représenté par a_j.

L'algorithme génétique travaille donc sur une discrétisation de l'espace de recherche continu : ceci explique qu'au mieux, dans sa formulation classique, il ne permet de trouver qu'une valeur approchée de l'optimum global. Cette valeur est d'autant plus proche de l'optimum global que le nombre de points échantillonnant l'espace de recherche est important, et donc que le nombre de bits utilisés pour coder un segment est grand.

Le code binaire classique présente pourtant un inconvénient : une petite modification du chromosome (changement d'un bit du code) peut se traduire par une grande différence au niveau phénotypique, c'est-à-dire lorsque l'on considère les réels. Pour remédier à ce problème on peut utiliser le code de Gray. L'avantage de ce dernier vient du fait que les codes de deux entiers consécutifs ont une distance de Hamming de 1, et ne diffèrent donc que d'une position au niveau de leur code binaire, comme l'illustre le tableau 3.1.

Entier	Binaire	Gray		Entier	Binaire	Gray
0	000	000		4	100	110
1	001	001		5	101	111
2	010	011		6	110	101
3	011	010		7	111	100

Tab. 3.1 – Equivalence entre code binaire et code de Gray.

Le code de Gray ne gomme cependant que très partiellement l'inconvénient du code binaire classique, puisqu'il est toujours possible de faire une transition entre deux entiers très distants en ne modifiant qu'un seul bit (par exemple passer de 6 à 1, et inversement). Il n'existe en fait

pas de code binaire qui permette de passer de deux codes binaires différant de un bit, à deux entiers ayant une différence de valeur absolue égale à un.

Pour définir la fonction d'évaluation, il est nécessaire de se rappeler les contraintes posées : le meilleur individu doit avoir l'évaluation la plus élevée et le mécanisme de base de sélection, la sélection proportionnelle (3.6), nécessite une évaluation positive. Pour cette raison, on ne peut prendre $\Phi(a_j) = C(\Gamma(a_j)), a_j \in P, j \in \{1, ..., \mu\}$ que dans le cas d'une maximisation, et encore faut-il que C ait un domaine de définition positif. En général, pour remplir les contraintes sur Φ, lors de minimisation ou lorsque C peut avoir des valeurs négatives, on définit Φ en ajustant la fonction de coût. De nombreuses techniques ont été définies [48, 52] ; parmi celles-ci, la plus courante ajuste le coût linéairement :

$$\Phi(a_j) = \alpha \cdot C(\Gamma(a_j)) + \beta, j \in \{1, ..., \mu\}, \alpha \in \mathbb{R}^* \text{ et } \beta \in \mathbb{R} \tag{3.2}$$

On peut par exemple, ajuster le coût linéairement suivant le plus mauvais individu, ce qui donne :
- $\Phi(a_j) = \max\{C(\Gamma(a_k)) \mid \forall a_k \in P, k \in \{1, ..., \mu\}\} - C(\Gamma(a_j))$ pour une minimisation ;
- $\Phi(a_j) = C(\Gamma(a_j)) - \min\{C(\Gamma(a_k)) \mid \forall a_k \in P, k \in \{1, ..., \mu\}\}$ pour une maximisation.

3.3.2 Opérateurs de reproduction et de sélection

Croisement

Le croisement ou « *crossover* » correspond à la recombinaison intervenant lors du processus de division cellulaire, il engendre deux nouveaux individus à partir de deux individus pouvant se reproduire. Les chromosomes des individus de départ (les parents) sont scindés à une ou plusieurs positions quelconques, puis les différents morceaux sont recollés ensemble en les croisant, ce qui permet de mélanger le patrimoine génétique des deux parents. L'idée à l'origine de cet opérateur est qu'en combinant des segments de chromosomes de « bons » parents, i.e. ayant une évaluation élevée, on devrait voir apparaître des individus ayant une meilleure évaluation [60].

Le croisement n'est pas appliqué systématiquement : en général on a une probabilité de croisement p_c comprise entre $0,6$ et 1. Tel est le résultat des valeurs proposées dans de nombreux travaux [37, 51]. Si le croisement n'est pas appliqué, les deux individus choisis pour se reproduire sont simplement copiés dans la nouvelle population.

L'opérateur de croisement originel introduit par Holland [60], dit croisement à un point, échange entre deux individus leur segment de chromosome situé à droite d'une position tirée aléatoirement dans le génotype. L'inconvénient de cet opérateur est que la probabilité d'échange de chaque bit est liée à sa position. En effet, plus la position du bit approche de l plus sa probabilité d'échange tend vers un.

Le croisement à un point a été ensuite généralisé à un nombre z de points de croisement quelconque, donnant lieu à quantité de travaux sur le choix de z : par exemple De Jong a proposé $z \in \{1, 2, 3, 4, 8\}$ [37]. Néanmoins tous ces résultats convergent vers le choix de petites valeurs paires. En prenant pour z le nombre maximum de points de croisement on obtient le croisement uniforme [123]. Ce dernier engendre les nouveaux individus en intervertissant aléatoirement les bits des individus parents.

Soient a'_S et a'_T les deux individus engendrés par croisement de a_S et a_T. On peut alors décrire les opérateurs de croisement évoqués ci-dessus, comme suit :

– Croisement multipoint (voir la figure 3.2)
z est fixe ou tiré aléatoirement ; $(\chi_1, ..., \chi_z \in \{1, ..., l-1\})$ sont les positions de croisement qui sont tirées aléatoirement, avec $\chi_k < \chi_{k+1}$ et $\chi_{z+1} = l$ si z impair, $k \in \{1, ..., l\}$.

$$a'_{S,i}; a'_{T,i} = \begin{cases} a_{T,i}; a_{S,i} & , \forall i \text{ vérifiant } (\chi_k < i \leq \chi_{k+1}), k \leq z, k \text{ impair} \\ a_{S,i}; a_{T,i} & , \text{sinon} \end{cases} \tag{3.3}$$

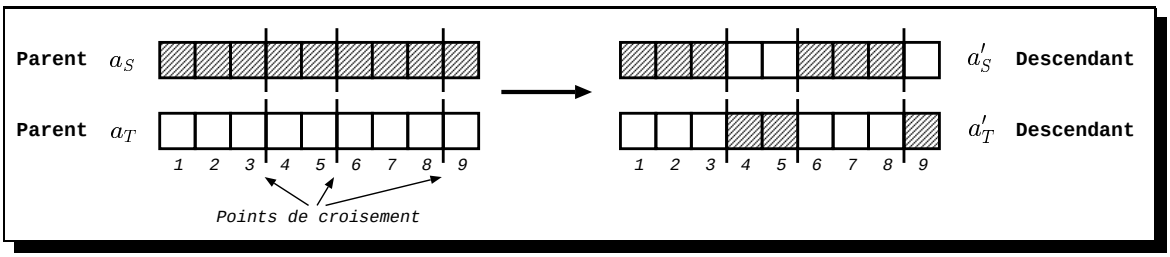


FIG. 3.2 – Croisement multipoint (z=3).

– Croisement uniforme

$$a'_{S,i}; a'_{T,i} = \begin{cases} a_{T,i}; a_{S,i} & , \chi_i > 1/2 \\ a_{S,i}; a_{T,i} & , \chi_i \leq 1/2 \end{cases} \text{ , avec } \chi_i \text{ tiré aléatoirement dans } [0;1] \tag{3.4}$$

Mutation

Cet opérateur permet de prévenir la perte de matériel génétique pouvant être utile, et garantit que l'on peut explorer l'espace de tous les chromosomes possibles. En effet, quand deux individus ont un gène identique en une même position, leurs descendants issus du croisement auront le même gène. La mutation met fin à cette situation en introduisant des altérations au niveau des gènes. Dans le cas du codage binaire, la mutation se traduit par la transformation du bit 0 en 1 et inversement ; la mutabilité de chaque gène est testée en comparant la probabilité de mutation p_m avec un nombre tiré aléatoirement. En général, p_m est très faible ($p_m \approx 0,001$ dans [37]), plus récemment il a été montré que $p_m = \frac{1}{l}$ est un choix adapté à de nombreux problèmes [12, 86]. Des travaux ont également montré que la modification dynamique de p_m au cours de l'algorithme, en prenant initialement une valeur élevée et en la réduisant au fur et à mesure, peut permettre d'améliorer l'efficacité et la vitesse de convergence de l'algorithme [112].

L'individu $a'_j = (a'_{j,1}, ..., a'_{j,l})$ correspondant à la mutation de $a_j = (a_{j,1}, ..., a_{j,l})$, s'obtient donc de la manière suivante, $\forall i \in \{1, ..., l\}, j \in \{1, ..., \mu\}$ on a :

$$a'_{j,i} = \begin{cases} a_{j,i} & , \text{si } \chi_i > p_m \\ 1 - a_{j,i} & , \text{si } \chi_i \leq p_m \end{cases} \text{ , avec } \chi_i \text{ tiré aléatoirement dans } [0;1] \tag{3.5}$$

Le fait de prendre une valeur très faible pour la probabilité de mutation se traduit par une faible différence au niveau chromosomique entre les deux individus a_j et a'_j. Néanmoins, l'utilisation du code binaire standard fait que de petites différences entre les génotypes, peuvent aboutir à de grandes disparités quand on considère les phénotypes. Ceci est également le cas pour le code de Gray, mais dans une moindre mesure.

Sélection

Le rôle de la sélection est de choisir un ensemble de descendants pour la phase de reproduction (soit $Q = \emptyset$). C'est pourquoi, à chaque descendant est associée une probabilité de sélection (ou de survie) p_s, modélisant la capacité du descendant à survivre jusqu'à se reproduire. Cette probabilité est basée sur l'évaluation, de sorte que, plus l'évaluation d'un individu est élevée, plus la probabilité qu'il procrée est grande.

La méthode de sélection introduite par Holland [60], appelée sélection proportionnelle, attribue aux individus une probabilité p_s proportionnellement à leur évaluation, soit :

$$p_s(a_j) = \frac{\Phi(a_j)}{\sum_{k=1}^{\mu} \Phi(a_k)}, \text{ où } j \in \{1, ..., \mu\} \text{ et } \mu \text{ est la taille de la population} \tag{3.6}$$

On tire ensuite aléatoirement μ individus en utilisant une roue de loterie divisée en autant de secteurs que d'individus [48], chaque secteur ayant une taille proportionnelle à la probabilité de sélection de l'individu qu'il représente. Comme nous l'avons déjà évoqué, le problème de cette méthode est qu'elle nécessite une évaluation positive et qui soit maximale pour le meilleur individu.

Une autre méthode de sélection intéressante est la sélection sur le rang [14, 52]. Cette dernière commence par établir un classement des individus suivant leur évaluation, puis associe à chaque individu une probabilité de sélection linéairement à son rang. Ceci, de façon à ce que le meilleur individu ait pour p_s la valeur la plus élevée. Le principal avantage de cette méthode par rapport à la sélection proportionnelle est la disparition des contraintes liées à l'évaluation (positivité notamment). Le tirage des individus se fait comme dans la sélection proportionnelle.

Nous évoquons aussi la sélection par tournois successifs [49], une méthode qui est notamment intéressante dans le cadre d'une parallélisation. Celle-ci n'attribue pas de probabilité de sélection aux individus, mais tire q individus afin de former une sous-population, puis choisit le meilleur individu dans cette sous-population. Cette procédure est ensuite répétée, jusqu'à obtenir une population d'individus apte à la reproduction identique à la population de départ. Lorsque $q = 2$ on parle de tournoi binaire.

Finalement il est à souligner que, du fait de la présence d'un aspect probabiliste dans les méthodes de sélection présentées ci-dessus, il est possible et même probable qu'un descendant soit sélectionné plusieurs fois. C'est pourquoi si un individu est nettement meilleur que la moyenne, il se peut qu'il soit dominant dans la nouvelle population, devenant alors un « super-individu ».

3.3.3 Remarques sur l'algorithme

Par rapport au schéma donné à la figure 3.1, il est à noter qu'en général dans un algorithme génétique la sélection ne se fait que sur la population des descendants. Le meilleur individu rencontré étant recopié en dehors de la population afin de le conserver. Pour ce qui est du choix des paramètres (p_c, p_m, μ), ceux-ci sont fixés sur la base de recommandations établies à la suite d'études, et sont ensuite ajustés empiriquement. En effet, les différents paramètres influent sur les performances de l'algorithme génétique, donc, en étudiant son comportement (efficacité, vitesse de convergence), on peut guider leur choix. Cette remarque est également valable pour le choix de l'opérateur de croisement et de la méthode de sélection.

3.3.4 Aspects théoriques

L'essentiel des analyses théoriques des algorithmes génétiques a porté sur l'étude de la convergence (preuve et conditions), l'influence des différents paramètres de l'algorithme et l'analyse de la « difficulté » des fonctions à optimiser. Différentes approches sont à la base de ces analyses théoriques : la modélisation markovienne [28, 36, 107, 108, 109], l'analyse de la déceptivité définie à partir de la théorie des schémas [47] et la modélisation sous forme d'un système dynamique [68]. Ce dernier modèle est particulièrement intéressant car il exhibe un lien entre le comportement de certains algorithmes génétiques et la géométrie fractale. Cette connexion permet notamment d'utiliser des propriétés de la géométrie fractale pour étudier les algorithmes génétiques [76, 75].

Dans la suite nous allons restreindre notre propos à la présentation de l'incontournable théorie des schémas développée par Holland [60], et à la question de la convergence des algorithmes génétiques. Nombre de travaux ont étudié la convergence des algorithmes génétiques, cependant les résultats obtenus ne le sont en général que par l'intermédiaire de modèles correspondant à des simplifications des algorithmes utilisés en pratique. On ne peut donc pas dire qu'une preuve de la convergence des algorithmes génétiques ait été établie dans le cas général. Pour établir la convergence de l'algorithme génétique simple (ou canonique) introduit par Holland, différentes approches ont été examinées. Certaines études ont utilisé le parallélisme implicite et le théorème des schémas (voir ci-dessous) mais les résultats ont été peu convaincants. En revanche le modèle markovien, comme pour le recuit simulé, a permis d'établir de nombreux résultats [109].

Théorie des schémas

Holland [60] fut le premier à proposer une interprétation (qui s'avéra très fructueuse) des algorithmes génétiques en introduisant la notion de *schéma*. Un schéma est un ensemble d'individus de l'espace de recherche, présentant un certain nombre (éventuellement zéro) de caractéristiques communes, i.e. de positions (bits) identiques au niveau du codage, et que l'on représente par un hyperplan. Clairement un individu appartient à plusieurs hyperplans, ainsi dans le cas d'un codage binaire, 10110101 appartient aux schémas $*0******$, $*0110*01$ ou encore $101101*1$, le symbole $*$ indiquant que la valeur du gène associé est indifférente. Pour un codage sur l caractères utilisant un alphabet de k caractères il existe donc $(k+1)^l$ schémas possibles. La notion de schéma permet d'interpréter le parcours de l'espace des solutions par l'algorithme génétique comme un échantillonnage de différents hyperplans. Dans ce cadre, les individus des différentes populations permettent d'évaluer l'adaptation de chaque schéma et donc d'affiner la recherche vers ceux qui sont le plus favorables. Cette interprétation se traduit par le théorème des schémas [60, 48].

Théorème 2 *Théorème des schémas*

Considérons un algorithme génétique sélectionnant les individus proportionnellement à leur évaluation, utilisant un croisement à un seul point et la mutation, avec une probabilité respective p_c et p_m. Alors pour tout schéma (ou hyperplan) H on a l'inégalité suivante :

$$m(H,t+1) \geq m(H,t)\,\frac{\Phi(H,t)}{\overline{\Phi}(t)}\left(1 - p_c \cdot \frac{\delta(H)}{l-1} - O(H)\cdot p_m\right),$$

- *$m(H,t)$ est la fréquence d'apparition des chromosomes du schéma H dans $P(t)$;*
- *$\Phi(H,t)$ est l'évaluation moyenne des chromosomes appartenant à l'hyperplan H ;*

- $\overline{\Phi}(t)$ *est l'évaluation moyenne de la population ;*
- $\delta(H)$ *est la distance entre le premier et le dernier bit qui sont fixes dans* H *(appelée également la longueur de définition du schéma) ;*
- l *est la longueur des chromosomes du schéma ;*
- $O(H)$ *est le nombre de bits qui sont fixes dans* H *(appelé également l'ordre du schéma).*

Une généralisation du théorème pour le croisement à plusieurs points a été également définie. D'autre part, dans le cas d'un codage binaire on remarque que l'évaluation d'un individu donne une information sur l'évaluation moyenne des 2^l schémas auxquels appartient l'individu. L'algorithme génétique étudie donc simultanément un nombre important de schémas, ce que Holland appelle le « parallélisme implicite ».

On peut donc comparer différentes représentations possibles en calculant le nombre de schémas traités simultanément par l'algorithme. Le meilleur choix est bien entendu le codage maximisant le parallélisme implicite, ce qui favorise tout naturellement les alphabets de cardinalité faible, et justifie le choix du code binaire. En ce qui concerne le choix de la sélection proportionnelle, Holland la justifie par l'étude du problème du bandit-manchot [48] ; en clair cette méthode de sélection est supposée allouer de façon optimale le nombre de fois qu'un individu est choisi pour se reproduire.

Pourtant, récemment, certains résultats de Holland ont été remis en cause [40]. En interprétant différemment le symbole $*$, il a tout d'abord été établi un résultat opposé à celui de Holland : le parallélisme implicite est maximal avec l'alphabet ayant le plus de symboles ; finalement il a été montré que la cardinalité de l'alphabet du code n'est pas déterminant, en fait il n'y a pas de bénéfice à maximiser le parallélisme implicite. De même, l'analyse du problème du bandit-manchot et les résultats qu'elle implique sur la sélection, ou l'utilisation du théorème des schémas en présence de bruit ont été discutées.

Concernant l'étude de l'influence du codage sur les performances d'un algorithme génétique, de nombreux résultats ont été obtenus en considérant les chromosomes comme une combinaison de « briques de base » (segments du chromosome ; appelées également *building blocks*) plus ou moins indépendantes. De plus, l'interaction de gènes voisins ou éloignés (*epistasis*) est un facteur non négligeable dans l'efficacité d'un algorithme génétique. En effet, si l'interaction entre deux gènes est faible, cela signifie qu'ils peuvent être optimisés indépendamment, or dans ce cas il y a des techniques plus efficace qu'un algorithme génétique. Réciproquement, si les différentes interactions sont trop fortes, les « briques de base » deviennent trop grandes, ce qui handicape un algorithme génétique.

Modélisation markovienne et convergence

Parmi les travaux ayant utilisé le cadre markovien il y en a deux qui nous semblent très intéressants : l'étude de la convergence de l'algorithme génétique canonique (AGC) faites par Rudolph [107] et la thèse de Cerf [27, 28]. Le travail de Rudolph est intéressant du fait qu'il utilise des propriétés des chaînes de Markov finies pour étudier la convergence, alors que Cerf introduit un modèle qui lui permet, grâce à la théorie de Freidlin-Wentzell [42], de définir des conditions suffisantes de convergence, liées à la taille de la population. Cette dernière étude est d'autant plus intéressante qu'elle utilise le même cadre mathématique que celui utilisé par Catoni [24, 25], puis étendu par Trouvé [131, 132, 133] pour établir les propriétés asymptotiques du recuit simulé. Nous avons choisi de présenter plus en détail le travail de Rudolph car il est plus simple à comprendre.

➔ *Rappels sur les chaînes de Markov finies*

Nous rappelons tout d'abord un certain nombre de définitions et de résultats qui seront utilisés pour analyser la convergence des algorithmes génétiques.

Une chaîne de Markov permet de décrire une trajectoire aléatoire sur un espace fini Υ de configurations, ayant pour cardinal m. Elle est entièrement définie par un vecteur de probabilités initiales $p^0 = (p_i^0)_{i\in\{1,...,m\}}$, et par les probabilités de transition entre deux configurations à l'instant t : $p_{ij}(t), i$ et $j \in \Upsilon$, que l'on regroupe dans une matrice de transition $P(t) = (p_{ij}(t))_{1\leq i,j\leq m}$ et $t \in \mathbb{N}$.

Dans le cadre de l'analyse de l'algorithme génétique simple, on considère une chaîne de Markov homogène, c'est-à-dire dont la matrice de transition ne dépend pas de l'instant t. On notera donc la matrice de transition et les probabilités de transition simplement P et p_{ij} respectivement. La distribution de la chaîne à l'instant t est alors donnée par $p^t = p^0 P^t$.

On introduit maintenant une définition permettant d'établir une classification utile des matrices de transition [118].

Définition 1

Soit P une matrice carrée de taille $m \times m$, elle est dite :

(1) *non négative* $(P \geq 0)$ *si* $p_{ij} \geq 0, \forall i, j \in \{1, ..., m\}$,

(2) *positive* $(P > 0)$ *si* $p_{ij} > 0, \forall i, j \in \{1, ..., m\}$.

Une matrice P non négative est dite :

(3) *primitive, si* $\exists k \in \mathbb{N}$ *tel que* P^k *soit positive,*

(4) *réductible, si P peut se mettre sous la forme (S et T sont des matrices carrées) :*

$$\begin{pmatrix} S & 0 \\ R & T \end{pmatrix}$$

en appliquant les mêmes permutations sur les lignes et les colonnes de P,

(5) *stochastique, si* $\sum_{j=1}^{m} p_{ij} = 1, \forall i \in \{1, ..., m\}$.

Une matrice P stochastique est dite :

(3) *stable si toutes les lignes sont identiques,*

(4) *positive par colonne si elle possède au moins un élément positif par colonne.*

Le lemme qui suit est à la base de l'étude de la convergence.

Lemme 1

Soient $CR = (c_{ij}), M = (m_{ij})$ et $S = (s_{ij})$ trois matrices stochastiques, avec M positive et S positive par colonne. Alors $CR \times M \times S$ est positive

Théorème 3 *[66, page 123]*

Soit P une matrice stochastique primitive, alors P^k (produit matriciel) converge quand $k \to \infty$ vers une matrice stochastique positive stable $P^\infty = (1, ..., 1)^T \bullet p^\infty$ (produit scalaire), avec

$$p^\infty = p^0 \lim_{k\to\infty} P^k = p^0 P^\infty$$

qui est sans élément nul et unique, indépendamment de la distribution initiale p^0.

Théorème 4 *[66, page 126]*
Soit P une matrice stochastique réductible vérifiant

$$P = \begin{pmatrix} S & 0 \\ R & T \end{pmatrix}$$

avec S de dimension $n \times n$ qui est une matrice stochastique primitive, et $R, T \neq 0$.
Alors

$$P^{\infty} = \lim_{k\to\infty} P^k = \lim_{k\to\infty} \begin{pmatrix} S^k & 0 \\ \sum_{i=0}^{k-1} T^i R S^{k-1-i} & T^k \end{pmatrix} = \begin{pmatrix} S^{\infty} & 0 \\ R^{\infty} & 0 \end{pmatrix}$$

est une matrice stochastique stable avec $P^{\infty} = (1, ..., 1)^T p^{\infty}$, où $p^{\infty} = p^0 P^{\infty}$ est unique indépendamment de la distribution initiale, et $p^{\infty} = (p_i^{\infty})_{i\in\{1,...,m\}}$ satisfait : $p_i^{\infty} > 0$ pour $1 \leq i \leq n$ et $p_i^{\infty} = 0$ pour $n < i \leq m$.

➔ *Analyse des algorithmes génétiques par chaîne de Markov*

L'algorithme génétique canonique maximisant la fonction d'évaluation $\Phi : \{0,1\}^l \to \mathbb{R}_+$ peut être décrit par une chaîne de Markov dont l'espace des configurations est $\Upsilon = \{0,1\}^{\mu \cdot l}$, où l est le nombre de gènes d'un chromosome et μ est la taille de la population. Chaque configuration est ainsi définie par une population unique. D'autre part, on introduit un opérateur $\pi_j(X_t)$ qui extrait le j-ième individu (chromosome) de la configuration X_t (induite par la population à la génération t), ce qui permet d'identifier tous les individus d'une population.

L'évolution de la population de l'algorithme génétique canonique, résultant de l'action des opérateurs génétiques, est ensuite modélisée par une matrice de transition P, qui peut se décomposer sous forme d'un produit de matrices stochastiques : $P = CR \times M \times S$, avec CR, M et S qui modélisent les transitions intermédiaires respectives du croisement, de la mutation et de la sélection. Ceci se traduit par le théorème suivant :

Théorème 5
La matrice de transition P d'un algorithme génétique canonique ayant une probabilité de croisement $p_c \in [0;1]$, une probabilité de mutation $p_m \in [0;1]$ et une sélection proportionnelle est primitive.

Corollaire 1
L'algorithme génétique canonique avec les paramètres tels que définis dans le théorème ci-dessus est une chaîne de Markov ergodique.

Preuve Conséquence directe des théorèmes 3 et 4.

On peut donc en déduire que la distribution initiale n'a pas d'effet sur l'AGC et que par conséquent l'initialisation de l'algorithme peut être faite de manière arbitraire. D'autre part, l'application initiale de l'opérateur de sélection peut en théorie être omise du fait que la distribution limite ne changera pas. En revanche le corollaire précédent influence grandement le comportement asymptotique de l'algorithme génétique canonique. Avant toute chose donnons une définition précise de ce que l'on entend par convergence d'un algorithme génétique (on se place dans le cas où on cherche le maximum d'une fonction).

Définition 2
Soit $\Phi_t = max\{\Phi(\pi_j(X_t)) \mid j \in \{1, ..., \mu\}\}$ *une séquence de variables aléatoires représentant la meilleure évaluation de la population à la génération* t. *Alors l'algorithme génétique canonique converge vers l'optimum global si et seulement si*

$$\lim_{t\to\infty} P(\Phi_t = \Phi_{max}) = 1$$

avec $\Phi_{max} = \max\{\Phi(a) \mid a \in \{0,1\}^l\}$.

Ceci mène au résultat majeur suivant :

Théorème 6 *[107]*
Tout algorithme génétique canonique ayant une probabilité de croisement $p_c \in [0;1]$, *une probabilité de mutation* $p_m \in [0;1]$ *et une sélection proportionnelle ne converge pas vers l'optimum global.*

Preuve Soit $X_t \in \Upsilon$ tel que $\max\{\Phi(\pi_j(X_t)) \mid j \in \{1, ..., \mu\}\} < \Phi_{max}$ et p_X^t la probabilité que l'AGC soit dans la configuration X à l'instant t. On a de manière évidente $P(\Phi_t = \Phi_{max}) \leq 1 - p_{X_t}^t$, or le théorème 3 établit que la probabilité que l'AGC soit dans la configuration X converge vers $p_X^\infty > 0$. D'où : $\lim_{t\to\infty} P(\Phi_t = \Phi_{max}) \leq 1 - p_{X_t}^\infty < 1$.
Ce résultat peut sembler surprenant. Cependant, en pratique il est courant de garder en dehors de la population le meilleur individu rencontré au cours de l'évolution de l'algorithme (élitisme). Ce qui du fait de l'ergodicité de la chaîne de Markov engendrée, assure que le meilleur optimum global sera rencontré après un nombre fini (élevé) de transitions. D'où le théorème :

Théorème 7 *[107]*
Tout algorithme génétique canonique défini comme dans le théorème 6, et conservant le meilleur individu rencontré au cours du temps converge vers l'optimum global.

Ce dernier théorème met clairement en évidence que la convergence de l'AGC n'est pas inhérente à l'algorithme, mais résulte de la conservation du meilleur individu rencontré. En conclusion, pour faire converger l'AGC il faut soit, modifier l'opérateur de sélection soit, définir des probabilités de mutation et de croisement évoluant au cours du temps, afin de rendre le processus markovien inhomogène. En tout état de cause, l'opérateur de sélection joue un rôle essentiel dans la convergence. Sur le plan pratique ce résultat n'a qu'un intérêt limité, un aspect plus intéressant serait le temps nécessaire à un algorithme génétique pour converger vers l'optimum global [10].
À noter que Rudolph a également étendu cette étude à des espaces de recherche arbitraires [107] ; plus récemment, il a présenté un « tour d'horizon » des résultats obtenus sur les algorithmes évolutionnaires grâce au modèle markovien [109].

3.4 Stratégies d'évolution (SEs)

Les stratégies d'évolution sont apparues en Allemagne au courant des années soixante. À l'époque, Rechenberg et Schwefel qui étaient étudiants à l'Université Technique de Berlin, cherchaient à résoudre des problèmes d'optimisation tels que la forme optimale d'un corps dans un flux [94]. Comme méthode de résolution, ils essayèrent tout d'abord une méthode de descente du

gradient, puis, suite à l'échec de celle-ci, ils optèrent pour la définition d'une méthode modifiant aléatoirement les paramètres spécifiant le corps, à l'image des mutations naturelles. Les premières implantations de cette stratégie dite à deux membres, et les résultats théoriques de la thèse de Schwefel [113], aboutirent au remplacement de la mutation discrète considérée initialement, par une mutation opérant des modifications suivant une distribution gaussienne de moyenne nulle et d'écart type σ donné. (Plus récemment des travaux ont étudié le remplacement de la gaussienne par une distribution de Cauchy, aboutissant à des résultats prometteurs. Néanmoins, nous n'allons considérer que la version « gaussienne » des stratégies d'évolution). Schématiquement la stratégie à deux membres utilise une représentation réelle et ne fait évoluer qu'un seul individu au seul moyen de la mutation. Pour ce qui est de la sélection, celle-ci met en compétition les deux individus (d'où le nom de stratégie à deux membres) et ne conserve que le meilleur. De plus elle sert de base à la notation des stratégies, d'où la notation SE$(1+1)$ pour la stratégie à deux membres. Ce qui est intéressant d'autre part concernant cette stratégie, c'est qu'elle présente des similitudes avec le recuit simulé continu (*Classical Simulated Annealing* et *Fast Simulated Annealing*, suivant la distribution utilisée) [105].

Sur la base d'une étude théorique de cette stratégie sur deux modèles de fonction, Rechenberg [95] proposa une amélioration sous la forme d'une règle de modification de l'écart type des mutations qu'il appelle la *Règle de succès* $\frac{1}{5}$:

Le ratio ϕ de mutations favorables, i.e. améliorant l'évaluation, par rapport au nombre total de mutations appliquées devrait être de $\frac{1}{5}$. S'il est supérieur à cette valeur, il faut augmenter la valeur de l'écart type ; inversement si ϕ est inférieure à cette valeur, l'écart type doit être diminué.

Cette règle modifie la valeur de l'écart type toutes les l générations et peut se formuler ainsi :

$$\sigma(t+1) = \begin{cases} c_d \cdot \sigma(t) & \text{si } \phi < \frac{1}{5} \\ c_i \cdot \sigma(t) & \text{si } \phi > \frac{1}{5} \\ \sigma(t) & \text{sinon} \end{cases}$$

où ϕ est le ratio de mutations favorables, $c_d < 1$ et $c_i > 1$ sont les coefficients contrôlant respectivement la décroissance et la croissance de l'écart type en passant de la génération t à la génération $t+1$. Dans [115] Schwefel fixe des valeurs précises pour ces coefficients ainsi que pour la fréquence de mesure du ratio sur la base de résultats théoriques. L'idée à l'origine de cette règle est de faire des pas plus grands dans l'espace de recherche lorsque l'on progresse et de les réduire dans le cas contraire.

Le raffinement suivant, également introduit par Rechenberg, consista à intégrer le concept de population, et donc un opérateur de recombinaison (équivalent du croisement), pour donner naissance à la première stratégie multi-membre notée SE$(\mu+1)$, où $\mu > 1$ est la taille de la population. Le principe de cette dernière est d'engendrer un individu à partir de μ individus au moyen des opérateurs de recombinaison et de mutation. Le nouvel individu remplace ensuite le plus mauvais individu parmi les μ individus de départ à la condition qu'il ait une meilleure évaluation. Cette stratégie n'eut cependant pas un grand succès, mais constitue plutôt une étape intermédiaire avant d'aboutir aux stratégies SE$(\mu+\lambda)$ et SE(μ,λ) avec $1 \leq \mu < \lambda$, introduites par Schwefel [115, 116] et qui sont actuellement la référence.

Ces deux dernières stratégies se différencient au niveau de la sélection. Ainsi $(\mu+\lambda)$ indique que cette stratégie engendre λ individus à partir de μ individus par recombinaison puis mutation et que la nouvelle population est construite en sélectionnant de manière déterministe les

μ meilleurs individus (au sens de l'évaluation) parmi les $\mu + \lambda$ individus. La stratégie (μ, λ) utilise le même protocole pour engendrer de nouveaux individus, en revanche la sélection est restreinte aux nouveaux individus. En limitant la durée de vie des individus à une seule génération on permet notamment à l'algorithme de sortir du domaine d'attraction d'un optimum local, contrairement à la stratégie $(\mu + \lambda)$, où le coût du meilleur individu suit une décroissance monotone en raison de l'élitisme. Bien que la stratégie SE(μ, λ) semble être un meilleur choix que la stratégie élitiste, en pratique les différences au niveau performances sont souvent limitées.

3.4.1 Représentation et fonction d'évaluation

Les stratégies d'évolution travaillent directement sur les variables réelles $x_i, i \in \{1, ..., n\}$, que l'on cherche à optimiser dans le cadre d'un problème ayant pour espace de recherche $\Omega = \mathbb{R}^n$. En plus de ces variables, on trouve un certain nombre de paramètres propres aux stratégies d'évolution, i.e. les variances (en fait les écarts types) et les covariances d'une distribution multinormale de dimension n utilisée pour la mutation. Le nombre de paramètres supplémentaires a évolué de manière croissante avec les différents raffinements de la stratégie d'évolution initiale :

- la stratégie SE$(1+1)$ n'utilisait qu'un seul écart type qui n'était pas inclus dans l'individu ;
- ensuite, des écarts types différents pour chaque composante à optimiser ont été introduits et intégrés dans les individus, afin de faire des mutations de pas différents suivant chaque axe de coordonnées ;
- finalement furent ajoutés des paramètres afin de corréler les mutations. C'est-à-dire que l'on fait muter le vecteur de variables que l'on cherche à optimiser en ajoutant un vecteur aléatoire suivant une distribution multinormale de dimension n. La distribution peut donc s'orienter dans l'espace de recherche suivant la direction la plus favorable. On parlera dans ce cas de mutations corrélées.

L'intégration des paramètres spécifiant la mutation, et le fait qu'ils subissent également le processus de l'évolution, doit permettre à l'algorithme de « s'auto-adapter », de façon à orienter la recherche favorablement et d'assurer une certaine diversité au niveau des individus engendrés. Le nombre variable de paramètres supplémentaires permet de choisir le nombre de degrés de liberté que l'on autorise pour la mutation des individus, mais implique un surcoût au niveau du temps de calcul non négligeable, et ceci d'autant que leur nombre est important.

La forme générale d'un individu dans une stratégie d'évolution $(\mu + \lambda)$ ou (μ, λ) est donc : $a_j = (x_j, \sigma_j, \alpha_j) \in \mathbb{R}^n \times \mathbb{R}_+^{n_\sigma} \times [-\pi; \pi]^{n_\alpha}$, avec $j \in \{1, ..., \mu\}$ ou $j \in \{1, ..., \lambda\}$, $n_\sigma \in \{1, ..., n\}$ et $n_\alpha \in \left\{0, \frac{(2n-n_\sigma)(n_\sigma-1)}{2}\right\}$. À côté du vecteur de variables $(x_{j,1}, ..., x_{j,n})$ on a donc de 1 à n écarts types $\sigma_{j,i'}$ et un maximum de $\frac{n(n-1)}{2}$ angles de rotation $\alpha_{kl} \in [-\pi; \pi]$, où $k \in \{1, ..., n-1\}$, $l \in \{k+1, ..., n\}$. Les vecteurs σ_j et α_j définissent la matrice de covariance de la distribution multinormale de moyenne nulle :

$$p(z) = \frac{\exp\left(-\frac{1}{2} z^T \mathbf{Cov}(\sigma, \alpha)^{-1} z\right)}{\sqrt{(2\pi)^n \cdot det(\mathbf{Cov}(\sigma, \alpha))}} \tag{3.7}$$

$\mathbf{Cov}(\sigma, \alpha) = (c_{kl})$ est la matrice de covariance. Un paramétrage directe par les covariances n'a pas été retenu car en faisant évoluer les covariances il est difficile de garantir l'orthogonalité du système de coordonnées, ou de manière équivalente que la matrice reste définie positive.

C'est pourquoi on utilise des angles de rotation pour modifier l'orientation de la distribution multinormale. Les angles sont liés aux covariances et aux variances par la relation suivante :

$$tan(2\alpha_{kl}) = \frac{2 \cdot c_{kl}}{{\sigma_k}^2 - {\sigma_l}^2} \tag{3.8}$$

Les angles α_{kl} décrivent en fait les rotations qui sont nécessaires pour passer d'un vecteur de mutation non corrélé à sa version corrélée. Ce mécanisme est décrit dans [13] et plus précisément par Rudolph dans [106]. Rudolph donne également une borne supérieure sur la somme $\mu + \lambda$ des tailles de population afin de pouvoir construire la matrice de corrélation en une seule génération.

Pour ce qui est de la définition de la fonction d'évaluation, cette dernière est identique à la fonction de coût, soit $\Phi(a_j) = C(x_j), j \in \{1, ..., \mu\}$ ou $j \in \{1, ..., \lambda\}$: en effet, contrairement aux algorithmes génétiques dont les opérateurs de sélection sont en général probabilistes, les stratégies d'évolution font une sélection déterministe. Dans le cadre d'une minimisation, l'individu ayant la meilleure évaluation est celui de coût minimal.

3.4.2 Opérateurs de reproduction et de sélection

Recombinaison

La recombinaison, équivalent du croisement des algorithmes génétiques, engendre une population intermédiaire formée de λ individus. Pour engendrer un individu ${a'}_j, j \in \{1, ..., \lambda\}$, la recombinaison peut faire appel à un nombre variable d'individus, allant de 2 à n individus de la population (rappel : n est le nombre de composantes d'un individu), on parle alors dans ce dernier cas de recombinaison multi-partenaire. Divers opérateurs de recombinaison ont été définis, les plus couramment considérés sont la recombinaison *discrète* et la recombinaison *intermédiaire*.

La recombinaison discrète reprend le principe du croisement uniforme : les composantes du nouvel individu sont choisies aléatoirement parmi les composantes correspondantes d'un couple d'individus. Quant à l'opérateur de recombinaison intermédiaire, il engendre un individu égal à la moyenne arithmétique de deux individus. Cet opérateur fut ensuite étendu en prenant une pondération de la différence entre deux composantes non plus égale à $\frac{1}{2}$, mais à un facteur tiré dans $[0; 1]$. On distinguera cette extension de la recombinaison intermédiaire, en disant qu'elle est généralisée. À chaque opérateur est d'autre part associée une version dite multi-partenaire ou globale, ces versions sont caractérisées par le fait que l'un des deux individus du couple est retiré pour chaque composante. Dans la version généralisée multi-partenaire, en plus du changement d'un individu, le facteur de pondération est également retiré. Ces opérateurs sont décrits formellement ci-dessous pour les variables $x_{j,i}$, avec $j \in \{1, ..., \lambda\}$ et $i \in \{1, ..., n\}$, que l'on cherche à optimiser :

$${x'}_{j,i} = \begin{cases} x_{S,i} \text{ ou } x_{T,i} & \textit{Recombinaison discrète} \\ x_{S,i} \text{ ou } x_{T_i,i} & \textit{Multi-partenaire discrète} \\ x_{S,i} + (x_{T,i} - x_{S,i})/2 & \textit{Intermédiaire} \\ x_{S,i} + (x_{T_i,i} - x_{S,i})/2 & \textit{Multi-partenaire intermédiaire} \\ x_{S,i} + \chi.(x_{T,i} - x_{S,i}) & \textit{Généralisée intermédiaire} \\ x_{S,i} + \chi_i.(x_{T_i,i} - x_{S,i}) & \textit{Multi-partenaire généralisée intermédiaire} \end{cases} \tag{3.9}$$

- S et $T \in \{1, ..., \mu\}$ sont les indices de deux individus tirés aléatoirement dans la population de départ.
- Le facteur χ est tiré aléatoirement dans l'intervalle $[0; 1]$.
- T_i signifie que l'on tire l'indice T à nouveau pour chaque composante $x'_{j,i}$, de même pour le facteur χ_i.

Il n'existe aucun résultat permettant d'estimer le comportement d'un opérateur de recombinaison. Aussi n'a-t-il pas été possible jusqu'à présent d'établir de règle permettant de guider le choix d'un opérateur pour l'optimisation d'une fonction particulière. Il a été aussi mis en évidence expérimentalement, que les opérateurs de recombinaison pouvaient avoir des comportements différents sur une même fonction suivant le nombre de variables à optimiser. Le choix d'un opérateur ne dépendrait donc pas uniquement de la topologie de la fonction à optimiser, mais également de sa dimension. De manière générale, parmi les opérateurs de recombinaison (3.9) la recombinaison discrète donne de bons résultats pour les variables à optimiser, et la recombinaison intermédiaire semble la plus appropriée pour les paramètres de la stratégie.

Mutation

L'opérateur de mutation n'a aucun caractère sexuel, il engendre un individu $a'_j = (x'_j, \sigma'_j, \alpha'_j)$ où $j \in \{1, ..., \lambda\}$, par perturbation des composantes de l'individu $a_j = (x_j, \sigma_j, \alpha_j)$ suivant la procédure décrite ci-dessous $\left(\text{soit } n_\sigma = n, \text{ d'où } \forall i \in \{1, ..., n\}, \forall i' \in \left\{1, \ldots, \frac{n(n-1)}{2}\right\}\right)$:

$$\begin{aligned} \sigma'_{j,i} &= \sigma_{j,i} \cdot \exp\left(\tau' \cdot \mathcal{N}(0,1) + \tau \cdot \mathcal{N}_i(0,1)\right) \\ \alpha'_{j,i'} &= \alpha_{j,i'} + \beta \cdot \mathcal{N}_j(0,1) \\ x'_j &= x_j + \mathcal{N}\left(0, \mathbf{Cov}\left(\sigma'_j, \alpha'_j\right)\right) \end{aligned} \tag{3.10}$$

Dans un premier temps chaque écart type est muté par multiplication avec des nombres aléatoires suivant deux distributions log-normales. Une distribution est spécifiée par un facteur commun à tous les écarts types, induisant un changement global, tandis qu'un facteur propre à chaque $\sigma_{j,i}$ définit la seconde distribution, permettant une modification propre à chaque axe de coordonnées. Ensuite, les différents angles de rotation sont perturbés par addition d'un nombre aléatoire suivant une distribution normale propre à chaque angle. La mutation x'_j du vecteur de de variables est finalement obtenue par addition d'un vecteur aléatoire, ce dernier étant défini par les vecteurs d'écarts types et d'angles de rotation mutés. Dans le cas où $n_\sigma = n$ et $n_\alpha = 0$, les composantes de x'_j vérifient : $x'_{j,i} = x_{j,i} + \sigma'_{j,i} \cdot \mathcal{N}_i(0,1), i \in \{1, ..., n\}$. On peut également avoir un nombre d'écarts types inférieur au nombre de variables à optimiser : la mutation des variables $x_{j,i}$ auxquelles ne sont pas associés d'écarts types (i.e. pour $i \in \{n_\sigma + 1, ..., n\}$) se fait alors en prenant $\sigma'_{j,i} = \sigma'_{j,n_\sigma}$.

Schwefel [114] justifie le choix de distributions log-normales pour pertuber σ et de lois normales pour α par plusieurs raisons et notamment : de petites modifications devraient être plus probables que de grandes ; en l'absence de sélection, aucune direction de recherche ne devrait être privilégiée. Il suggère également des valeurs pour les différents facteurs :

$$\tau' = \frac{1}{\sqrt{2n}}, \tau = \frac{1}{\sqrt{2\sqrt{n}}} \text{ et } \beta \approx 0,0873 \ (5^\circ \text{ en radians}) \tag{3.11}$$

Néanmoins suivant le paysage de la fonction que l'on cherche à optimiser ces valeurs ne sont pas optimales.

D'autre part, il est possible que lors de l'exécution de l'algorithme, certains écarts types atteignent une valeur nulle : dans ce cas la mutation n'aura plus aucune influence sur la variable correspondante. L'optimisation ne se fera donc plus que sur un sous-espace de Ω. Dans [117] les auteurs proposent des bornes inférieures pour les écarts types, notamment pour garantir que tous les écarts types ont une valeur suffisante pour perturber le vecteur de variable. De même, les angles de rotation peuvent sortir de leur domaine de définition. Aussi pour mettre fin à ce problème on supposera que le domaine est bouclé sur lui-même.

Sélection

Comme nous l'avons déjà précisé la sélection est déterministe dans les stratégies d'évolution : le choix des individus formant la population de la génération suivante se fait sur la base du rang de l'évaluation. Différents opérateurs de sélection peuvent être définis : nous présentons ci-dessous la sélection déterministe classique.

La sélection déterministe est l'opérateur de sélection originel. Deux versions différenciant les stratégies SE$(\mu+\lambda)$ et SE(μ,λ) existent. Dans le premier cas la sélection se fait sur la population des parents et de leurs descendants ($Q(t) = P(t-1)$ dans l'algorithme de la figure 3.1) et on dira que la sélection est élitiste ou préservative ; dans le second cas seule la population des nouveaux individus est considérée ($Q(t) = \emptyset$). Si on considère une minimisation, on a donc (en notant les populations comme dans la figure 3.1) :

$$P = \text{Sélection}\left(P' \cup Q\right) = \left\{a_j \in P' \cup Q, j \in \{1,...,\mu\} \mid \forall a_k \in (P' \cup Q) \setminus P, k \in \{1,...,\lambda\} \text{ on a } \Phi(a_k) \geq \Phi(a_j)\right\} \quad (3.12)$$

avec $|Q| = \mu$ si SE$(\mu+\lambda)$; $|Q| = \emptyset$ si SE(μ,λ).

La stratégie non élitiste est en général préférée car elle permet :
- dans le cas d'une fonction dont la topologie évolue, de suivre un optimum ;
- de sortir d'un optimum local (à condition que son bassin d'attraction ne soit pas trop étendu), ce qui est intéressant pour l'optimisation de fonctions très irrégulières ;
- en limitant la durée de vie d'un individu à une génération, d'assurer une plus grande diversité au niveau des paramètres de la stratégie.

L'opérateur de sélection est chargé de guider la recherche. Aussi le choix des tailles de population influence-t-il grandement le comportement de l'algorithme : plus μ est petit, plus la recherche est guidée et la vitesse de convergence augmentée. À l'inverse une valeur élevée se traduit par une exploration plus importante de l'espace de recherche. D'ailleurs comme l'a montré Schwefel [114], la vitesse de convergence induite par la sélection non élitiste (μ,λ) dépend fortement du rapport $\frac{\lambda}{\mu}$. D'autre part, des raffinements au niveau de la durée de vie des individus sont également possibles [71, 117].

Dans le cadre d'une parallélisation, la sélection la plus adaptée est *a priori* la sélection par tournoi. Cette dernière est définie dans les stratégies d'évolution de la même manière que dans les algorithmes génétiques.

3.4.3 Remarques sur l'algorithme

Nous précisons quelques points sur les stratégies d'évolution par rapport au schéma général d'un algorithme évolutionnaire (figure 3.1).

Initialisation de la population de départ

Pour obtenir $P(0)$ on distingue deux étapes d'initialisation : l'une pour les vecteurs $x_j(0)$, $j \in \{1, ..., \mu\}$ de variables à optimiser, l'autre pour les vecteurs de paramètres de la stratégie. Pour initialiser les vecteurs $x_j(0)$ deux méthodes sont considérées [117] : un échantillonnage aléatoire de l'espace de recherche Ω ; la perturbation aléatoire d'un individu de départ (celui-ci est fixé par l'utilisateur ou tiré aléatoirement) pour engendrer les $\mu - 1$ individus restants. Les écarts types sont généralement fixés à l'issue de tests, néanmoins, il n'est pas conseillé de prendre de valeurs trop grandes, car celles-ci suivant la taille de la population $P(t)$ peuvent faire diverger l'algorithme. Pour ce qui est des angles de rotation, aucune indication n'a été formulée suite à une non utilisation de cette stratégie en pratique. La raison sous-jacente à ce désintérêt nous semble être un coût calculatoire important, non compensé par un gain pertinent au niveau des performances, et une difficulté à trouver les bons paramètres pour une fonction donnée.

Critère d'arrêt

Schwefel propose de baser le critère d'arrêt sur la comparaison du coût entre le meilleur et le plus mauvais individu [114, 117]. Il définit deux mesures possibles (nous les présentons dans le cas d'une minimisation) :

$$\max\{\Phi(a_j(t)), j \in \{1, ..., \mu\}\} - \min\{\Phi(a_j(t)), j \in \{1, ..., \mu\}\} \leq \epsilon_1 \tag{3.13}$$

$$\frac{\mu}{\epsilon_2} \cdot (\max\{\Phi(a_j(t)), j \in \{1, , ..., \mu\}\} - \min\{\Phi(a_j(t)), j \in \{1, , ..., \mu\}\}) \leq \sum_{j=1}^{\mu} |\Phi(a_j(t))| \tag{3.14}$$

avec $\epsilon_1 > 0$, $\epsilon_2 > 0$ et t qui est le compteur de génération. Ces mesures seront dites respectivement absolue (3.13) et relative (3.14). Elles peuvent être modifiées de façon à mesurer la convergence de la population sur un certain nombre de générations. Ainsi la mesure absolue aurait la forme :

$$\min\{\Phi(a_j(t - \delta t)), j \in \{1, ..., \mu\}\} - \min\{\Phi(a_j(t)), j \in \{1, ..., \mu\}\} \leq \epsilon_1 \tag{3.15}$$

Pourtant en pratique ces mesures sont rarement utilisées. Elles sont remplacées par un nombre maximum de générations engendrées, soit $t \leq t_{max}$.

3.4.4 Aspects théoriques

Les bases des études théoriques des différentes stratégies d'évolution résultent des premiers travaux de Rechenberg [95] sur le calcul de la vitesse de convergence de la stratégie $(1 + 1)$. Rechenberg dériva tout d'abord une expression de la vitesse de convergence pour des fonctions de coût convexe, puis à partir de ce premier résultat il put exprimer :

- la valeur optimale pour l'écart type σ ;
- la vitesse de convergence maximale ;
- la probabilité de succès d'une mutation, i.e. engendrant un meilleur individu.

Les valeurs obtenues pour les probabilités de succès lui servirent en particulier pour formuler la *Règle de succès* $\frac{1}{5}$ modifiant l'écart type σ, afin d'obtenir une « bonne » vitesse de convergence.

Dans un premier temps nous allons focaliser la présentation sur la stratégie $(1 + 1)$. Nous présenterons d'abord en détail comment sont obtenus les différents résultats évoqués précédemment dans le cadre de l'optimisation de la fonction $C(x) = \|x - x_{opt}\| = \sum_{i=0}^{n} {x_i}^2$ où $x_{opt} = 0$ est l'optimum [18, 95], puis nous aborderons la question de la convergence globale.

Suivra une partie donnant succinctement des résultats analogues à ceux évoqués précédemment en considérant des stratégies plus évoluées (SE$(1,\lambda)$; SE(μ,λ), sans et avec recombinaison) pour optimiser $C(x)$. Cependant, les stratégies considérées ne sont en général que des simplifications des stratégies utilisées en pratique. Le nombre de paramètres de la stratégie est notamment réduit à un écart type unique, qui de plus, ne « s'auto-adapte » pas. D'ailleurs, bien qu'il soit admis que plusieurs paramètres de contrôle de la stratégie (n écarts types par exemple ; avec ou sans angles de rotation) « s'auto-adaptant » améliorent la convergence, aucun résultat théorique ne permet de justifier cette constatation. Néanmoins, il est à noter que récemment, Rudolph [110] a montré qu'une stratégie SE$(1+1)$ permettant à l'écart type de s'ajuster, se traduit par une probabilité de convergence vers l'optimum global qui est en général inférieure à un.

Résultats théoriques sur la stratégie d'évolution $(1+1)$

Soit $a(t) = x(t)$ l'individu à la génération t, la stratégie $(1+1)$ engendre alors un individu $a'(t+1) = x'(t+1) = x(t)+z$, où z est un vecteur aléatoire dont les composantes $z_i, i \in \{1,...,n\}$ sont tirées suivant une distribution $\mathcal{N}(0,\sigma)$:

$$p(z_i) = \frac{1}{\sqrt{2\pi}\sigma} exp\left(-\frac{1}{2}\left(\frac{z_i}{\sigma}\right)^2\right) \tag{3.16}$$

Ensuite, l'opérateur de sélection choisit le nouvel individu entre celui de départ et son descendant, ce qui se formalise par :

$$\begin{aligned} \Phi(a'(t+1))(= C(x(t)+z)) \leq \Phi(a(t))(= C(x(t)) \quad &\Rightarrow \quad a(t+1) = x(t+1) = x(t)+z \\ \Phi(a'(t+1))(= C(x(t)+z)) > \Phi(a(t))(= C(x(t)) \quad &\Rightarrow \quad a(t+1) = x(t+1) = x(t) \end{aligned}$$

On définit la vitesse de convergence φ comme l'espérance mathématique de la variation de la distance euclidienne entre l'individu $a(t) = x(t)$ et l'optimum $a_{opt} = x_{opt}$ lorsque l'on passe de la génération t à la génération $t+1$:

$$\varphi = E[\|x_{opt} - x(t)\| - \|x_{opt} - x(t+1)\|] \tag{3.17}$$

Il est évident que la fonction $C(x)$ admet x_{opt} comme centre de symétrie, et que l'ensemble des individus de même coût définissent une hypersphère de dimension $n-1$. Considérons la surface correspondant à l'intersection de l'hypersphère avec le plan défini par le vecteur z et le segment $[x(t); x_{opt}]$ (voir la figure 3.3). Si z est dans le cercle, la mutation est acceptée, puisque la distance r de $x(t)+z$ à l'optimum est inférieure à R. Dans le cas contraire elle est rejetée.

La vitesse de convergence (3.17) est donc définie par l'espérance de la variable aléatoire $R-r$. Or R étant constant, la densité de probabilité de la variable $R-r$ correspond à la densité de probabilité de r. Considérons la projection de z sur $[x(t); x_{opt}]$; celle-ci définit une variable aléatoire e dont dépend r $(r^2 = (R-e)^2 + l^2 - e^2)$, donc pour $n \gg 1$ on a :

$$\varphi = \int_{e_i}^{e_s} (R-r)\, p(e)\, de \tag{3.18}$$

avec $e_i = \frac{l^2}{2R}$ ($x(t)+z$ est sur le cercle), $e_s = l = \sigma\sqrt{n}$ ($x(t)+Z$ est sur $[x(t); x_{opt}]$).

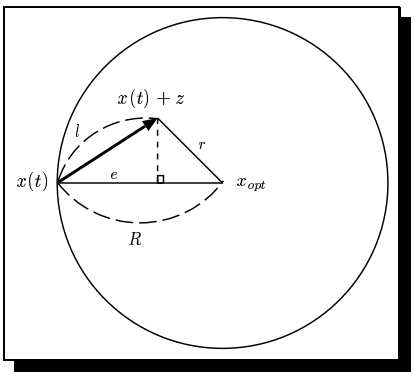


FIG. 3.3 – Surface correspondant à l'intersection de l'hypersphère avec le plan défini par le vecteur z et le segment $[x(t); x_{opt}]$.

En introduisant les valeurs normalisées

$$\varphi^* = E^*[R - r] = \frac{n}{R}\varphi \text{ et } \sigma^* = \frac{n}{R}\sigma \tag{3.19}$$

on obtient pour $n \gg 1$ à la limite :

$$\begin{aligned} \varphi^* &= \frac{\sigma^*}{\sqrt{2\pi}} \int_{\frac{\sigma^*}{2}}^{\sqrt{n}} \left(t - \frac{\sigma^*}{2}\right) exp\left(-\frac{1}{2}t^2\right)\, dt \\ &= \frac{\sigma^*}{\sqrt{2\pi}} exp\left(-\frac{(\sigma^*)^2}{8}\right) - \frac{(\sigma^*)^2}{2}\left[\frac{1}{2} - \Phi_0\left(\frac{\sigma^*}{2}\right)\right] \end{aligned} \tag{3.20}$$

où $\Phi_0(x) = \frac{1}{\sqrt{2\pi}} \int_0^x exp\left(-\frac{1}{2}t^2\right)\, dt$ (intégrale de Gauss).

La valeur optimale σ^*_{opt} pour l'écart type normalisé s'obtient en calculant la valeur σ^* pour laquelle on a $\frac{d\varphi^*}{d\sigma^*} = 0$, puis on dérive la vitesse de convergence normalisée optimale φ^*_{opt} en remplaçant σ^* par σ^*_{opt} dans (3.20).

La probabilité de succès p_s de la mutation se calcule facilement à partir de $p(e)$, puisque les mutations qui induisent une amélioration au niveau du coût doivent vérifier $e_i \leq e \leq l$ $(n \gg 1)$. Par conséquent :

$$p_s = \int_{e_i}^{l} p(e)\, de = \left(\frac{1}{2} - \Phi_0\left(\frac{\sigma^*}{2}\right)\right) \tag{3.21}$$

Il est alors aisé de trouver la probabilité optimale en faisant le calcul pour σ^*_{opt}.

En ce qui concerne la convergence globale de la stratégie SE$(1+1)$, le théorème suivant a été établi :

Théorème 8 *[59]*

Soit $\sigma > 0$ et considérons un problème d'optimisation régulier vérifiant $C_{opt} > -\infty$ (minimisation) ou $C_{opt} < +\infty$ (maximisation), C_{opt} étant le coût optimal, alors

$$\lim_{t \to \infty} P\left(C(x(t)) = C_{opt}\right) = 1$$

Ce théorème signifie que l'optimum global est trouvé avec une probabilité égale à un, à condition que le temps d'exécution soit « suffisamment » grand.

Résultats théoriques sur des stratégies plus évoluées

Les premiers travaux sur des stratégies plus évoluées sont dus à Rechenberg [95] et Schwefel [114]. L'approche de Schwefel était de plus suffisamment générale pour être appliquée aux stratégies $(\mu + \lambda)$ et (μ, λ). Les résultats que nous donnons ci-dessous sont extraits de travaux plus récents.

Pour la stratégie (μ, λ) sans recombinaison, Rechenberg [97] a obtenu pour la vitesse de convergence normalisée, l'expression suivante :

$$\varphi^*_{\mu,\lambda} = -\frac{(\sigma^*)^2}{2} + c_{\mu,\lambda}\sigma^*, \tag{3.22}$$

où $c_{\mu,\lambda}$ appelé le coefficient de progression, est une constante dépendant notamment de μ et λ.

De l'étude de $\varphi^*_{\mu,\lambda}$ on déduit que $c_{\mu,\lambda}$ est la valeur optimale pour l'écart type normalisé, induisant une vitesse de convergence normalisée optimale égale à $\frac{1}{2}(c_{\mu,\lambda})^2$. Néanmoins, suite à une étude expérimentale, il devait s'avérer que cette expression n'était valable que pour de petites valeurs de $\sigma^*(n \gg 1)$, et que d'autre part $\varphi^*_{\mu,\lambda}$ dépendait également de la dimension n du problème considéré. Une expression de $\varphi^*_{\mu,\lambda}$ dépendant de n fut donnée par Beyer [20] :

$$\varphi^*_{\mu,\lambda} = n\left(1 - \sqrt{1 + \frac{(\sigma^*)^2}{n}}\right) + c_{\mu,\lambda}\sigma^* \sqrt{\frac{1 + \frac{(\sigma^*)^2}{2n}}{1 + \frac{(\sigma^*)^2}{n}}} \tag{3.23}$$

En prenant $\mu = 1$ on obtient les expressions établies lors de travaux antérieurs [18, 96] sur la stratégie $(1, \lambda)$, avec $c_{1,\lambda} = \frac{\lambda}{\sqrt{2\pi}} \int_{-\infty}^{+\infty} t\ exp\left(-\frac{1}{2}t^2\right) \left[\frac{1}{2} + \Phi_0(t)\right]^{\lambda-1}\ dt \approx \sqrt{2\ln\lambda}$.

Beyer [19] a également étudié la vitesse de convergence de la stratégie (μ, λ) avec recombinaison. Les opérateurs qu'il a considérés sont la recombinaison discrète et la recombinaison intermédiaire (voir (3.9)), mais toutes deux étendues de façon à considérer les μ individus de la population comme parents d'un descendant (la version intermédiaire modifiée se traduit par exemple comme suit : $x'_{j,i} = \frac{1}{\mu}\sum_{S=1}^{\mu} x_{S,i} + \mathcal{N}(0, \sigma^2)$, avec $j \in \{1, ..., \lambda\}$ et $i \in \{1, ..., n\}$). Les vitesses de convergence normalisées obtenues sont :

- pour la recombinaison discrète (notée SE$(\mu/\mu_D, \lambda)$))

$$\varphi^*_{\mu/\mu_D,\lambda} = n\left(1 - \sqrt{1 + \frac{(\sigma^*)^2}{n}}\right) + \frac{\sqrt{\mu}c_{\mu/\mu,\lambda}\sigma^*}{\sqrt{1 + \frac{(\sigma^*)^2}{n}}} \tag{3.24}$$

  approchée par $\varphi^*_{\mu/\mu_D,\lambda} = -\frac{(\sigma^*)^2}{2} + \sqrt{\mu}c_{\mu/\mu,\lambda}\sigma^*$ [97] pour de petites valeurs de σ^* (3.24.a) ;
- pour la recombinaison intermédiaire avec μ parents (notée SE$(\mu/\mu_I, \lambda)$)

$$\varphi^*_{\mu/\mu_I,\lambda} = n\left(1 - \sqrt{1 + \frac{(\sigma^*)^2}{\mu n}}\right) + c_{\mu/\mu,\lambda}\sigma^* \frac{1 + \frac{(\sigma^*)^2}{2\mu n}}{\sqrt{1 + \frac{(\sigma^*)^2}{2n}}\sqrt{1 + \frac{(\sigma^*)^2}{\mu n}}} \tag{3.25}$$

  approchée par $\varphi^*_{\mu/\mu_I,\lambda} = -\frac{(\sigma^*)^2}{2\mu} + c_{\mu/\mu,\lambda}\sigma^*$ (3.25.a)

où $c_{\mu/\mu,\lambda} = \frac{\lambda-\mu}{2\pi}\begin{pmatrix}\lambda\\ \mu\end{pmatrix}\int_{-\infty}^{+\infty} exp(-t^2)\left(\frac{1}{2}+\Phi_0(t)\right)^{\lambda-\mu-1}\left(\frac{1}{2}-\Phi_0(t)\right)^{\mu-1}dt$, pour $\lambda \to \infty$ et $0 < \frac{\mu}{\lambda} < 1$ on a approximativement $c_{\mu/\mu,\lambda} = \frac{\lambda}{\mu}\frac{1}{\sqrt{2\pi}}exp\left(-\frac{1}{2}\left[\Phi_0^{-1}\left(\frac{1}{2}-\frac{\mu}{\lambda}\right)\right]^2\right)$.

Ces différents résultats montrent clairement que la vitesse de convergence normalisée dépend du rapport $\frac{\lambda}{\mu}$. De l'étude des approximations de (3.24.a) et (3.25.a), valables pour $n \gg 1$, λ assez grand, μ pas trop petit, il a été montré que $\varphi^* \approx \mu \ln \frac{\lambda}{\mu}$ [21]. Dans cette dernière expression le facteur μ est dû à la recombinaison, tandis que $\frac{\lambda}{\mu}$ reflète l'influence de la sélection.

3.5 Évolution différentielle (ED)

L'idée à l'origine de l'évolution différentielle [120, 121] est d'utiliser la différence entre deux individus choisis aléatoirement dans la population pour perturber un autre individu. De même que les stratégies d'évolution, l'évolution différentielle est orientée vers l'optimisation de variables réelles. Cependant, l'évolution différentielle est plus simple à mettre en œuvre que ces dernières car elle ne nécessite pas de distribution normale, voir multinormale, et s'abstrait donc de la difficulté d'adaptation des paramètres spécifiant ces distributions. Au niveau des opérateurs, les auteurs distinguent un opérateur de « perturbation » et un opérateur de croisement ; néanmoins comme nous le verrons ces deux opérateurs peuvent être fusionnés en un seul.

3.5.1 Représentation et fonction d'évaluation

Tout comme les stratégies d'évolution, l'algorithme de l'évolution différentielle opère au niveau phénotypique en travaillant directement sur un vecteur de réels correspondant aux variables que l'on cherche. En revanche les individus n'intègrent pas de paramètres de contrôle comme c'est le cas dans les stratégies d'évolution (section 3.4.1). Concernant l'évaluation de chaque individu l'évolution différentielle ne nécessite pas d'adaptation de la fonction de coût. Dans la suite on remplacera donc la fonction d'évaluation par la fonction de coût. En résumé, pour une fonction de coût $C : \mathbb{R}^n \to \mathbb{R}$ on a : $a = x \in \mathbb{R}^n$ et $\Phi(a) = C(x)$.

3.5.2 Opérateurs de reproduction et de sélection

Reproduction

Comme le montre la figure 3.4, l'évolution différentielle suit les grandes lignes du schéma des algorithmes évolutionnaires (figure 3.1). La seule particularité notable se situe au niveau de la phase de reproduction. Storn *et al.* [120, 121] décrivent la reproduction par l'application successive de deux opérateurs : tout d'abord un opérateur de « perturbation » engendrant une population intermédiaire (Etape 1), puis un croisement de chaque parent avec l'individu intermédiaire qui lui correspond au moyen d'une adaptation du croisement à deux points des algorithmes génétiques (Etape 2). En étudiant les différents opérateurs de perturbation qui ont été définis, on constate que les opérateurs présentés dans [121] et étendus dans [119] sont en fait de nouvelles variantes des opérateurs de recombinaison des stratégies d'évolution [13]. D'où l'idée de fusionner les deux opérateurs en un seul que l'on identifiera par opérateur de reproduction.

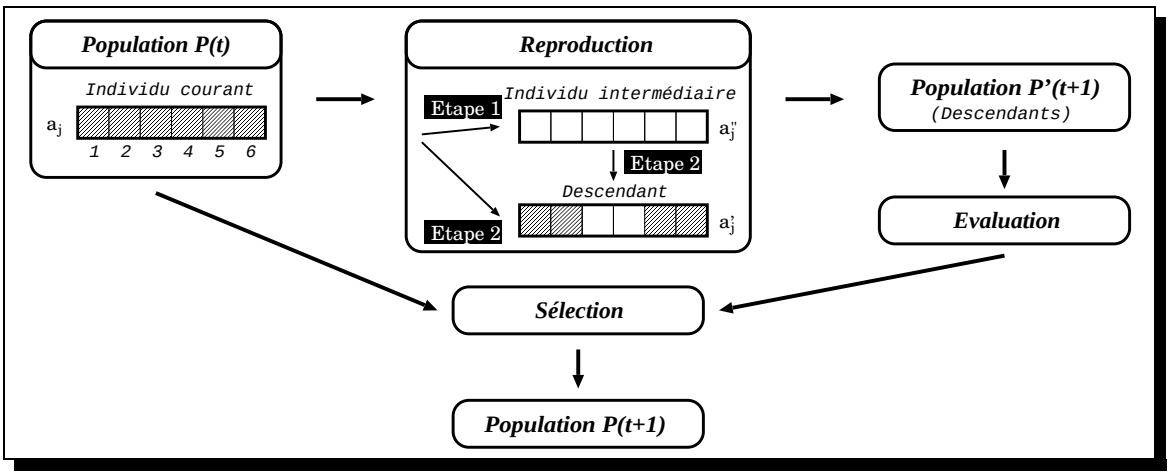


Fig. 3.4 – Génération d'une nouvelle population dans l'évolution différentielle.

On suppose que la population contient μ individus, chaque individu étant formé de n composantes, d'où $a_j = (x_{j,1}, x_{j,2}, ..., x_{j,n}), j \in \{1, ..., \mu\}$. Soit a'_j l'individu obtenu à partir de a_j, on peut alors décrire formellement l'opérateur de reproduction comme suit.

Soient χ_1 et $\chi_2 \in \{1, ..., n\}$ les deux points de croisement :

$$\chi_1 = (\chi \bmod n) + 1 \text{ et } \chi_2 = ((\chi + L - 1) \bmod n) + 1 \tag{3.26}$$

où χ et L sont deux entiers tirés aléatoirement pour chaque individu dans $\{1, ..., n\}$. χ définit la première position de croisement, alors que L est le nombre de variables qui sont modifiées. De plus, l'entier L est tiré de sorte que $P(L \geq \nu) = (p_c)^{\nu-1}$, où p_c est la probabilité de croisement donnée par l'utilisateur. Le nouvel individu $a'_j = (x'_{j,i})_{i \in \{1,...,n\}}$ est alors engendré de la manière suivante (plusieurs méthodes (1)-(4) sont possibles) :

$$x'_{j,i} = \begin{cases} x_{S,i} + F \cdot (x_{T,i} - x_{U,i}) & (1) \\ x_{opt,i} + F \cdot (x_{S,i} - x_{T,i}) & (2) \\ x_{opt,i} + F \cdot (x_{S,i} + x_{T,i} - x_{U,i} - x_{V,i}) & (3) \\ x_{j,i} + \gamma \cdot (x_{opt,i} - x_{j,i}) + F \cdot (x_{S,i} - x_{T,i}) & (4) \\ x_{j,i} & \end{cases} \begin{array}{ll} & \\ & \\ & \\ & \\ \text{si} & (\chi_1 \leq \chi_2) \wedge (\chi_1 \leq i \leq \chi_2) \text{ ou} \\ & (\chi_1 > \chi_2) \wedge (1 \leq i \leq \chi_2 \vee \chi_1 \leq i \leq n) \\ \text{sinon} & \end{array} \tag{3.27}$$

L'individu $a_{opt} = x_{opt}$ correspond à l'optimum courant. Les indices $S, T, U, V \in \{1, ..., \mu\}$ sont choisis aléatoirement de façon à être mutuellement différents ($S \neq T \wedge S \neq U \wedge S \neq V$ et $T \neq U \wedge T \neq V \wedge U \neq V$), mais également différents de l'indice j de l'individu courant ($S \neq j \wedge T \neq j \wedge U \neq j \wedge V \neq j$). $F \in [0, 1; 1]$ et $\gamma \in [0, 1; 1]$ sont des facteurs constants contrôlant l'amplification des vecteurs différence. Habituellement on donne à γ la même valeur que celle de F afin de réduire le nombre de paramètres de l'algorithme. À noter qu'il est possible d'avoir $\chi_2 < \chi_1$: c'est pourquoi nous avons adapté le croisement à deux points en considérant que les individus sont « circulaires ».

Sélection

La sélection est entièrement déterministe. Elle consiste à faire un tournoi entre chaque individu a_j de la population courante et son descendant a'_j. Cela signifie qu'entre a_j et a'_j on garde celui qui a la meilleure évaluation ; en cas d'égalité on conserve a_j (honneur à l'ancien). Formellement dans le cas d'une minimisation :

$$\text{Sélection}\left(P' \cup P\right) = \left\{s(a_j, a'_j), j \in \{1, ..., \mu\}\right\} \tag{3.28}$$

avec $s(a_j, a'_j) = (1 - p_s(a'_j)) \cdot a_j + p_s(a'_j) \cdot a'_j$ et $p_s(a'_j) = 1_{\mathbb{R}_+^*}(\Phi(a_j) - \Phi(a'_j))$. Ce type de sélection est dit μ-élitiste.

3.5.3 Choix des différents paramètres de l'algorithme

Plusieurs règles pour fixer les paramètres sont données [119] :

- Choisir une population initiale dispersée autant que possible sur l'ensemble de l'espace de recherche : en général la population de départ est construite par échantillonnage aléatoire dans l'espace de recherche. Une autre solution possible est de tirer aléatoirement un individu dans l'espace de recherche, puis d'engendrer la population en ajoutant aux composantes de cet individu des valeurs tirées suivant une distribution normale ;
- Prendre pour p_c une valeur faible ($\approx 0,30$), sauf si l'algorithme converge vers un optimum local, auquel cas on peut essayer une valeur plus proche de 1. Plus la probabilité de croisement est proche de zéro, plus le nombre de composantes de l'individu de départ qui sont « perturbées » est faible, et inversement ;
- Prendre la même valeur pour F et γ avec $F \in \left[\frac{1}{2}, 1\right]$;
- Plus la population est grande, plus F devrait être faible.

3.6 Comparaison des différents algorithmes évolutionnaires

Parmi les algorithmes évolutionnaires vus précédemment, les algorithmes génétiques (AGs) et les stratégies d'évolution (SEs) sont ceux ayant atteint le plus grand degré de maturité. En effet, depuis leur apparition dans les années soixante, ces techniques ont donné lieu à de nombreux travaux comme en témoigne la littérature. Aussi va-t-on surtout donner des éléments de comparaison portant sur ces deux algorithmes.

La première chose que l'on constate lorsque l'on étudie les algorithmes génétiques et les stratégies d'évolution, c'est que ces techniques ont été définies dans des buts différents. Ainsi à l'origine du développement des stratégies d'évolution on trouve des problèmes d'optimisation de variables, alors que les algorithmes génétiques, de leur côté, peuvent être vus comme des techniques de recherche plus générales dont l'idée était d'allouer un nombre croissant exponentiellement d'essais aux schémas d'évaluation au-dessus de la moyenne.

Les différences entre algorithmes génétiques et stratégies d'évolution apparaissent lorsque l'on considère les éléments induits par un algorithme évolutionnaire. Par exemple :

- Les algorithmes génétiques utilisent en général une représentation binaire alors que les stratégies d'évolution voient un individu comme un vecteur de variables réelles. En fait la première méthode travaille au niveau génotypique alors que la seconde considère le niveau

phénotypique. Cette différence peut cependant disparaître en considérant un AG à code réel (i.e. $a = x$) : dans ce cas il faut également définir des opérateurs appropriés ;
- Un algorithme génétique à code binaire de longueur fixe induit une borne inférieure sur la précision avec laquelle on peut obtenir l'optimum ;
- L'opérateur de sélection est probabiliste dans les algorithmes génétiques alors qu'il est déterministe dans les stratégies d'évolution ;
- Lors de la sélection dans une stratégie d'évolution les plus mauvais individus parmi les $\mu + \lambda$ individus (ou λ dans le cas de la stratégie non élitiste) n'ont aucune chance de se reproduire. À l'inverse dans un algorithme génétique, la sélection probabiliste permet en théorie à tous les individus de se reproduire.
- Au niveau des paramètres on remarque que les paramètres des algorithmes génétiques sont en général fixes alors que dans les stratégies d'évolution les paramètres contrôlant la distribution normale utilisée pour la mutation font partie intégrante des individus et subissent également le processus d'évolution. En conséquence les paramètres σ et éventuellement α d'une stratégie d'évolution « s'auto-adaptent », mais cela ne doit pas faire prendre à la légère le choix de leur valeur initiale.

Néanmoins lorsque l'on regarde le développement des deux méthodes dans sa globalité, il apparaît clairement que les différences qui étaient au départ très marquées se sont réduites par la suite. Les premières stratégies d'évolution on ainsi vu apparaître un opérateur de recombinaison avant la mutation, tandis que la communauté des algorithmes génétiques a réévalué sa position vis-à-vis de l'opérateur de mutation qu'elle considérait plutôt comme un opérateur secondaire. Des études ont aussi été menées pour introduire l'idée d'auto-adaptation des paramètres dans les algorithmes génétiques, en particulier pour la probabilité de mutation [11, 12] ; inversement, l'intégration de concepts des AGs dans les stratégies d'évolution fut proposée [117].

En ce qui concerne l'évolution différentielle du fait de son développement plus récent, cet algorithme n'a pas un cadre aussi bien établi que les deux autres. Cependant, par rapport aux stratégies d'évolution on peut faire les remarques suivantes :
- sa mise en œuvre est plus simple, notamment en raison de l'absence des paramètres de contrôle de la densité de probabilité gaussienne qu'utilisent ces dernières lors de la mutation ;
- elle nécessite un nombre plus restreint de paramètres de contrôle puisque qu'elle ne requiert au maximum que trois paramètres : les deux facteurs d'amplification F et γ, encore qu'en général on prend $\gamma = F$, et une probabilité de croisement.

3.7 Parallélisations

Contrairement au recuit simulé dont la parallélisation n'est pas aisée du fait de la nature séquentielle de l'algorithme [3, 7], les modes de parallélisation des algorithmes évolutionnaires [130] sont plus facilement identifiés, puisque ces algorithmes sont intrinsèquement parallèles. Le parallélisme peut être introduit dans un algorithme évolutionnaire à trois niveaux : au niveau de la population, des individus ou de la fonction d'évaluation. Ce dernier niveau ne sera pas considéré car étant lié au problème d'optimisation considéré, alors que les deux autres niveaux induisent des parallélisations génériques. C'est sur celles-ci que nous nous concentrons dans la suite.

3.7.1 Parallélisme à gros grain

La parallélisation au niveau de la population consiste à diviser la population initiale en autant de sous-populations (appelées « îlots ») que de processeurs, chaque processeur exécutant ensuite l'algorithme évolutionnaire considéré sur sa sous-population L'intérêt de ce type de parallélisation à gros grain vient du fait que plusieurs instances de l'algorithme traitant des sous-populations différentes permettent d'accroître les chances de trouver de bonnes solutions. Les sous-populations peuvent être connectées suivant un voisinage défini afin de permettre la migration de certains individus d'une sous-population à une autre. Une alternative consiste à laisser évoluer les sous-populations indépendamment et à ne les faire communiquer qu'à la fin de la recherche. On peut donc imaginer deux modèles de communication :

Le modèle des sous-populations isolées

Dans ce modèle chaque processeur travaille indépendamment des autres et communique uniquement à la fin du processus de recherche afin de choisir la meilleure solution parmi celles trouvées. L'accélération que l'on obtient est linéaire. L'inconvénient majeur de ce modèle est qu'une convergence prématurée vers un optimum local d'une ou plusieurs sous-populations est possible. Ce problème est surtout induit par deux facteurs : la taille des sous-populations qui n'est pas assez grande et une mauvaise initialisation de celles-ci, i.e. les individus ne sont pas suffisamment dispersés sur l'espace de recherche, d'où un manque de diversité génétique. Il est donc fort possible qu'un processeur soit bloqué par une sous-population « pauvre », sans savoir que d'autres processeurs travaillent dans des régions plus prometteuses de l'espace de recherche.

Le modèle des sous-populations liées ou avec migration

Cette approche revient à échanger régulièrement des informations entre les différents processeurs sur le(s) meilleur(s) individu(s) que chacun a trouvé(s). En règle générale, c'est le meilleur individu qui migre d'un processeur vers un autre processeur, il remplace alors un autre individu de la sous-population du processeur cible de la migration, celui-ci étant soit le plus mauvais ou tiré aléatoirement [23]. Il est également usuel de diffuser le meilleur individu à l'ensemble des processeurs afin de guider la recherche dans la direction la plus favorable [73]. En conséquence il est nécessaire de spécifier de nouveaux paramètres dans l'algorithme : la fréquence des migrations, le nombre d'individus qui migrent et la topologie du schéma de migration entre processeurs (anneau [23], grille, etc).

Différents choix sont possibles pour ces paramètres. Kwok *et al.* [73] ont par exemple proposé une évolution dynamique de la période entre les migrations : en notant t_{max} le nombre de générations engendrées, la période initiale vaut $\lceil \frac{t_{max}}{2} \rceil$, puis successivement $\lceil \frac{t_{max}}{4} \rceil$, $\lceil \frac{t_{max}}{8} \rceil$, ... $\lceil \frac{t_{max}}{2i} \rceil$ avec $i \geq 1$. L'idée est qu'au départ la diversité de la population étant élevée, il vaut mieux laisser les processeurs travailler afin de favoriser l'exploration de l'espace de recherche. Puis, lorsque la population commence à converger, il est nécessaire d'exploiter plus rapidement l'information. En effet, certains processeurs risquent d'optimiser inutilement le meilleur individu de leur sous-population alors qu'il a peut-être une évaluation inférieure à celles des meilleurs individus lorsque l'on considère l'ensemble de la population.

D'autre part, comme le font remarquer Calégari *et al.* [23], le nombre de sous-populations est un aspect dont il faut tenir compte car celui-ci influence fortement l'accélération obtenue. C'est pourquoi pour eux, associer le modèle par sous-populations à la parallélisation à gros grain n'est

pas adéquat car on peut très bien avoir un nombre très élevé de sous-populations, chacune de taille très réduite, puisque la population a toujours la même taille (des sous-populations réduites à deux individus par exemple).

Pour ce qui est des communications entre processeurs, celles-ci peuvent être synchrones ou asynchrones. L'avantage du schéma synchrone est la simplicité de sa mise en œuvre, néanmoins si on considère le coût des communications, l'asynchrone est plus intéressant. Ce propos est toutefois à nuancer suivant l'architecture cible. De fait, beaucoup de paramètres entrent en jeu pour apprécier les bénéfices et les inconvénients respectifs de chaque schéma de communication : rapidité des communications inter-processeurs, topologie du réseau d'interconnexion, processeurs hétérogènes ou non. En effet, il est évident que sur un cluster de PC dont les processeurs sont de puissance variable, les communications asynchrones seront plus compétitives (le temps de calcul d'une nouvelle sous-population sur un processeur étant variable, il est intéressant que les processeurs les plus rapides puissent continuer leur calcul aussitôt que les données dont ils ont besoin sont disponibles). À l'opposé, si on considère une machine massivement parallèle dont les processeurs sont tous de même puissance, les communications synchrones seront moins pénalisantes (d'autant plus que le schéma de communication est régulier).

Les différents travaux évoqués précédemment montrent que le parallélisme à gros grain permet d'obtenir de bonnes performances sur différentes architectures :

- L'algorithme génétique parallèle de Kwok *et al.* [73], utilisé pour résoudre un problème d'ordonnancement de tâches sur machine multiprocesseurs, exhibe une accélération quasi-linéaire sur Intel Paragon (communications synchrones). De plus, ses performances ne subissent aucune dégradation avec l'augmentation de la taille des graphes (scalabilité) ;
- De même, l'algorithme génétique parallèle défini dans [23] afin de traiter un problème de placement de radio-émetteurs qui soit optimal en nombre d'émetteurs et du point de vue de la surface couverte, a également une accélération quasi-linéaire. Leur algorithme repose sur un réseau de stations connectées sous la forme virtuelle d'un anneau orienté le long duquel les individus vont migrer d'une sous-population à une autre. Les auteurs mettent également en évidence l'apport de la division en sous-populations et l'influence de la taille des sous-populations : 40 sous-populations de 4 individus permettent d'obtenir une meilleure solution que 4 sous-populations de 40 individus, ce dernier partitionnement aboutissant déjà à une solution dont la qualité est supérieure à celle résultant d'une seule population de 160 individus.

3.7.2 Parallélisme à grain fin

La distribution de la population à raison d'un individu par processeur se traduit par un algorithme parallèle à grain plus fin, dit data-parallèle. Dans ce schéma, l'élément majeur est la topologie du réseau de communications car elle conditionne la diffusion des individus. En effet, celle-ci se fait par l'intermédiaire des opérateurs de sélection et de reproduction qui se font localement en considérant un voisinage. Ceci détermine implicitement le degré d'« isolation » et, par conséquent, la diversité des individus de la population. Du fait des interactions locales il n'y a pas de convergence prématurée car les caractéristiques des individus sont lentes à se propager sur le réseau de processeurs. Ce n'est qu'après un certain nombre de générations, que l'on voit apparaître des groupes d'individus ayant les mêmes caractéristiques.

Manderick *et al.* [81] ont défini ainsi un algorithme parallèle sur ce principe en utilisant un réseau de processeurs élémentaires organisés sous la forme d'une grille torique. En particulier, ils montrèrent qu'une sélection locale permet d'avoir une meilleure exploration de l'espace de recherche et de contrecarrer une éventuelle domination précoce de la population par les meilleurs individus. On peut également citer les travaux de Talbi *et al.* [128], consacrés à la définition d'un algorithme génétique massivement parallèle pour résoudre le problème du placement de tâches sur une machine parallèle de façon à optimiser sa charge. Cet algorithme parallèle montre qu'à l'image du modèle à gros grain vu ci-dessus, le parallélisme à grain fin permet d'obtenir une accélération quasi-linéaire.

Dans l'étude que nous présentons, nous allons utiliser une approche identique à celle de Manderick *et al.*, mais appliquée aux stratégies d'évolution et à l'évolution différentielle. En effet, bien que les algorithmes évolutionnaires comportent différentes familles d'algorithmes, paradoxalement l'essentiel des travaux sur leur parallélisation n'a eu pour cadre que les algorithmes génétiques. Par ailleurs, notre étude se fonde sur le modèle de programmation data-parallèle, tout en étant mis en œuvre sur une architecture parallèle de type MIMD (*Multiple Instruction Multiple Data*).

Chapitre 4

Algorithmes parallèles hybrides recuit simulé/algorithme génétique

4.1 Motivations

Un algorithme hybride consiste à combiner différentes techniques d'optimisation, l'objectif étant de définir un algorithme possédant les avantages des techniques d'optimisation mises en jeu tout en restreignant leurs inconvénients de façon à obtenir de bonnes caractéristiques. Les algorithmes hybrides les plus fréquemment considérés sont une combinaison du recuit simulé et d'un algorithme génétique. En effet, ces deux méthodes d'optimisation ont des avantages et des inconvénients qui sont complémentaires.

Ainsi le recuit simulé est intéressant du fait qu'une preuve formelle de sa convergence vers l'optimum global existe, ce qui n'est pas aussi clairement établi pour les algorithmes génétiques. D'un autre coté les algorithmes génétiques effectuent une recherche multipoint en travaillant sur une population d'individus alors que le recuit simulé n'itère que sur un seul individu (équivalent à une configuration).

Au niveau de la parallélisation, les algorithmes génétiques sont facilement parallélisables [130] ce qui est plus problématique pour le recuit simulé en raison de la nature séquentielle de l'algorithme [3, 7]. D'autre part les accélérations obtenues sont souvent limitées pour le recuit simulé alors que certaines versions parallèles des algorithmes génétiques conduisent à une accélération quasi-linéaire [23, 73, 128]. C'est donc au niveau de la parallélisation que la mise en œuvre d'algorithmes hybrides peut se révéler particulièrement intéressante. C'est sur ce point que nous allons focaliser notre attention.

Le principe des méthodes hybrides est de combiner l'opérateur de croisement des algorithmes génétiques avec la sélection probabiliste locale du recuit simulé, par l'intermédiaire du schéma de température. En fait, on peut considérer ce schéma comme un recuit parallèle dans lequel a été introduit l'aspect « population » des algorithmes génétiques, puisque l'algorithme itère sur une population plutôt que sur un individu grâce à l'opérateur de croisement. D'ailleurs certains chercheurs [105] estiment que le recuit simulé et les algorithmes génétiques sont des algorithmes « relativement proches ». En effet, ils considèrent le recuit simulé comme un algorithme évolutionnaire dont la population est réduite à un seul individu et dont les descendants sont obtenus

exclusivement par mutation. De nombreux algorithmes hybrides ont été proposés, parmi ceux-ci les plus intéressants sont le *Parallel Recombinative Simulated Annealing* proposé par Mahfoud et Goldberg [78], ainsi que le *Genetic Simulated Annealing* de Chen *et al.* [29]. Le fait que ces algorithmes intègrent un schéma de température leur permet notamment de conserver les propriétés de convergence du recuit simulé, et donc d'établir une preuve de convergence.

4.2 Parallel Recombinative Simulated Annealing (PRSA)

Le PRSA [78] peut être vu comme plusieurs recuits simulés évoluant en parallèle, utilisant la mutation comme opérateur de voisinage et dont les solutions indépendantes sont recombinées par l'intermédiaire du croisement. En fait le croisement étend l'opérateur de voisinage du recuit simulé. Comme le montre la figure 4.1, l'algorithme PRSA a la même structure qu'un algorithme évolutionnaire, mais avec en plus une température qui est utilisée lors de la sélection. Chaque étape de l'algorithme est bien entendu exécutée en parallèle sur chaque processeur.

$T \leftarrow T_0$ | *- Température initiale (une valeur élevée)*
Initialisation de la population initiale
Évaluation de la population
Répéter
  Répéter $\lambda/2$ **fois** | *-* λ *est la taille des sous-populations sur un processeur*
    (a) Sélection de deux individus dans la sous-population courante
    (b) Reproduction engendrant deux descendants
    (c) Évaluation des descendants
    (d) Faire un ou deux tests de Boltzmann entre descendants et parents, afin de sélectionner les individus formant la nouvelle sous-population
  Abaisser périodiquement T
Jusqu'à *critère d'arrêt vérifié*

FIG. 4.1 – Schéma du *Parallel Recombinative Simulated Annealing.*

Un test de Boltzmann consiste à mettre en compétition deux individus a et a' d'évaluation respective $\Phi(a)$ et $\Phi(a')$ (en fait, $\Phi(a)$ correspond au coût (ou énergie) de l'individu a). Dans le cas d'une minimisation, l'individu a gagne avec la probabilité :

$$\frac{1}{1+\exp\left(\frac{\Phi(a)-\Phi(a')}{T}\right)} \tag{4.1}$$

La probabilité (4.1) est appelée la probabilité logistique. On peut éventuellement la remplacer par le critère de Metropolis (1.5). Différentes compétitions entre descendants et parents sont possibles :

① *Double acceptation/rejet* : les deux parents et les deux descendants sont acceptés ou rejetés simultanément. Pour cela on ne fait qu'un test de Boltzmann mais en additionnant respectivement l'évaluation des deux parents et des deux descendants ;

② *Simple acceptation/rejet* : chaque parent concourt contre un descendant ; on fait donc deux tests.

Comme le font remarquer les auteurs, au niveau de l'implantation sur machine parallèle, deux schémas de communications sont possibles (on note μ la taille de la population) :

① *Synchrone*
Ce mode repose sur un schéma maître/esclave. Les $p-1$ processeurs esclaves engendrent chacun une sous-population initiale de taille $\frac{\mu}{p-1}$, puis engendrent une nouvelle sous-population suivant l'algorithme PRSA. Lorsqu'une sous-population a été engendrée celle-ci est envoyée au processeur maître, qui après avoir reconstitué l'ensemble de la population, redistribue aléatoirement tous les individus aux processeurs esclaves qui reprennent alors le calcul.

② *Asynchrone*
Dans ce mode les p processeurs lancent l'algorithme PRSA après avoir engendré chacun une sous-population initiale de taille $\frac{\mu}{p}$. Quand une (ou plusieurs) nouvelle sous-population a été engendrée, un certain nombre d'individus choisis aléatoirement sont transmis et remplacés par des individus provenant des autres processeurs. Un aspect important est donc le réseau d'interconnexion entre processeurs et le mode de transmission des individus (global ou local (voisins)).

Mahfoud *et al.* [78] ont implanté les deux schémas de communications sur une *Connection Machine* CM-5 ayant une partition de 64 nœuds. Toutefois, seuls des résultats de la version asynchrone sont présentés. De plus, les accélérations présentées ne sont pas obtenues en considérant les temps de calculs, mais le nombre d'évaluations de la fonction de coût. De fait, lors des expérimentations, le système et la librairie de passage de messages étaient encore en cours de développement. Par conséquent, on peut dire que, d'une part la difficulté du problème considéré influence grandement les résultats présentés (le nombre d'évaluations nécessaires pour converger), d'autre part on n'a aucune idée de l'impact des communications sur les performances.

Cet algorithme présente *a priori* un certain nombre d'avantages :

① Parallélisme implicite (voir la théorie des schémas présentée dans la section 3.3.4).

② Preuves de la convergence de l'algorithme vers l'optimum global (pour différentes stratégies de sélection et d'acceptation ; étapes (a) et (d) de l'algorithme), obtenues en s'appuyant sur les résultats théoriques du recuit simulé (voir la section 1.2).

③ Permet de préserver la diversité globale de la population.

④ Conservation des bonnes solutions.

Il est clair que le parallélisme n'est limité que par la taille de la population. D'autre part les tests effectués par les auteurs ont mis en évidence deux modes de convergence : un mode rapide basé surtout sur le croisement et un mode lent basé sur la mutation. Néanmoins les tests n'ayant pas été effectués sur des problèmes très complexes et de grande taille (optimisation d'une combinaison de 8 fonctions, chacune définie sur 3 bits, soit des individus de 24 bits ; deux découpages des individus sont considérés : soit les bits d'une même fonction sont contiguës, soit ils sont entrelacés), et en raison des conditions d'expérimentations évoquées ci-dessus, il faut rester prudent quand à l'efficacité réelle de cette parallélisation.

4.3 Genetic Simulated Annealing (GSA)

Cet algorithme parallèle [29] est à grain fin, i.e. chaque individu est distribué sur un processeur. Le point important dans le cas d'un algorithme parallèle de ce type est la structure du voisinage car elle spécifie la manière dont les individus seront diffusés. Les auteurs du GSA ont défini le voisinage d'un individu (configuration) comme étant l'ensemble des individus se trouvant à ses huit points cardinaux, à une distance d'au plus une constante fixée initialement. Pour une distance égale à un on a donc un 8-voisinage.

On fera l'hypothèse que la topologie du réseau de processeurs est une grille 2D dont chaque processeur peut communiquer avec ses huit voisins. Les transferts entre processeurs sont définis implicitement dans le code en référençant un processeur dans l'une des huit directions possibles par appel à la fonction **Voisin**. Ainsi par exemple **Voisin**(1,3).v référence la variable v sur le troisième processeur dans la direction nord-est (0 = Nord, 1 = Nord-est, ..., 7 = Nord-ouest).

Chaque processeur exécute l'algorithme suivant :

① le schéma de température est initialisé dans un premier temps en fixant la température initiale et le coefficient de décroissance (ces paramètres peuvent varier d'un processeur à un autre) ;

② l'individu initial de chaque processeur est amélioré itérativement par reproduction avec un individu voisin jusqu'à atteindre un nombre maximal d'itérations et en faisant décroître la température à chaque itération.

Finalement le résultat de l'algorithme GSA est l'individu ayant la meilleure évaluation (le coût ou l'énergie minimal(e) dans le cas d'une minimisation) parmi tous les individus de la population finale. L'algorithme associé à chaque processeur est décrit formellement par la figure 4.2, avec :

- γ et δ sont des nombres entier tirés aléatoirement, représentant respectivement une direction et une distance, ce qui permet de déterminer un individu « voisin » pour la reproduction. Comme la graine utilisée pour initialiser le générateur de nombres aléatoires est une variable globale, la direction et la distance sont identiques sur l'ensemble des processeurs.
- $MAX\_ITERATIONS$ qui correspond au nombre de modifications possibles d'un individu sur un processeur.
- $MAX\_DISTANCE$ qui détermine la distance maximale entre deux individus voisins.

Remarques :

- le fait que la graine utilisée soit commune à l'ensemble des processeurs assure leur coordination globale, garantissant ainsi qu'aucun processeur n'est considéré simultanément comme voisin pour la reproduction par plusieurs processeurs. On a donc un schéma régulier de communications ;
- l'opérateur de reproduction engendre deux nouveaux individus a'_1 et a'_2 à partir de l'individu courant a_1 et du voisin a_2, par croisement puis mutation comme dans un algorithme génétique classique ;
- la phase de sélection consiste à choisir comme nouvel individu l'un des deux descendants ou l'individu courant. Pour cela un tournoi est organisé entre les trois individus de sorte que chaque individu ait été en compétition avec les deux autres. La compétition entre individus est simulée par une adaptation du critère stochastique (1.5) du recuit simulé :

$T \leftarrow T_0$ | *- Température initiale*
$a_1 \leftarrow a_0$ | *- Individu de départ*
Pour i **de** 1 **à** $MAX\_ITERATIONS$ **faire**
  $DIRECTION \leftarrow \gamma, \gamma \in \{0,1,2,3,4,5,6,7\}$
  $DISTANCE \leftarrow \delta, \delta \in \{1, ..., MAX\_DISTANCE\}$
  $a_2 \leftarrow$ **Voisin**$(DIRECTION, DISTANCE).a_1$
  $\{a'_1, a'_2\} \leftarrow$ **Reproduction**(a_1, a_2)
  $a_1 \leftarrow$ **Sélection**(a_1, a'_1, a'_2, T)
  $T \leftarrow g(T)(= \alpha \cdot T, \alpha \in]0;1[)$ | *- g est strictement décroissante*
Fin pour

FIG. 4.2 – Schéma du *Genetic Simulated Annealing* sur un processeur.

– pour une minimisation on a

$$Duel(a, a', T) = \text{ si } \big(\Phi(a) - \Phi(a')\big) \leq T \text{ alors } a \text{ sinon } a' \tag{4.2}$$

– et pour une maximisation

$$Duel(a, a', T) = \text{ si } \big(\Phi(a) - \Phi(a')\big) \leq T \text{ alors } a' \text{ sinon } a \tag{4.3}$$

Chen *et al.* ont utilisé ce critère déterministe car il a des performances équivalentes au critère classique (1.5) tout en étant plus efficace au niveau du temps de calcul. À l'issue des trois matches entre individus, le gagnant est déterminé par transitivité, c'est-à-dire que si un individu gagne contre les deux autres il est obligatoirement le gagnant. Néanmoins à haute température, il se peut que l'on ne puisse pas désigner de gagnant : dans ce cas on conserve l'individu courant (honneur à l'ancien). Le résultat de la sélection peut être résumé par le tableau 4.1 (celui-ci peut également être vu comme une table de vérité).

	Sélection(a_1, a'_1, a'_2, T)			
	Duel(a_1, a'_1, T)	**Duel**(a_1, a'_2, T)	**Duel**(a'_1, a'_2, T)	**Individu choisi**
1	a_1	a_1	a'_1	a_1
2	a_1	a_1	a'_2	a_1
3	a_1	a'_2	a'_1	a_1
4	a_1	a'_2	a'_2	a'_2
5	a'_1	a_1	a'_1	a'_1
6	a'_1	a_1	a'_2	a_1
7	a'_1	a'_2	a'_1	a'_1
8	a'_1	a'_2	a'_2	a'_2

TAB. 4.1 – Résultat de la sélection dans le *Genetic Simulated Annealing*.

– pour terminer, la température est abaissée par réduction géométrique de raison $\alpha \in]0;1[$ proche de 1 (fonction g), calculée à partir de la température initiale T_0 et de la température

finale T_f, avec $0 < T_f < T_0$, de façon à avoir au bout de $MAX\_ITERATIONS, T = T_f$. En général les paramètres T_0 et T_f sont fixés à l'issue de tests. Il est à noter que ce schéma correspond au schéma de température exponentiel classique du recuit simulé.

Le *Genetic Simulated Annealing* a été étudié par ses auteurs sur deux plans :

1. l'influence du choix du schéma de température sur l'algorithme, en comparant deux schémas de température sur différentes tailles de population (nombre de processeurs) :
   - *Schéma de température uniforme* : T_0 et T_f sont fixés après une phase de tests, puis répliqués sur tous les processeurs.
   - *Schéma de température aléatoire* : T_0 et T_f sont fixés de manière probabiliste pour chaque processeur.
2. l'impact de la croissance de la taille de la population sur les performances de l'algorithme.

Deux problèmes d'optimisation sont considérés lors de l'étude : le problème du voyageur de commerce pour 100 villes et la définition d'un code correcteur d'erreurs de 24 mots de 12 bits chacun. Ces deux problèmes sont relativement complexes notamment par rapport à ceux retenus par Mahfoud *et al.* [78] pour étudier leur algorithme hybride, les résultats au niveau performances du GSA sont donc plus probants. L'expérimentation sur les deux problèmes d'optimisation précédents permet de faire les observations suivantes :

- le schéma de température aléatoire donne de meilleurs résultats, ou au pire identiques, que le schéma uniforme, et ceci sans aucune influence de la taille de la population. En effet, dans le cas du schéma aléatoire on a une palette de schémas de température ce qui permet dans le cadre de l'algorithme GSA de conserver un équilibre entre diversité de la population et perte de matériel génétique. En fait les processeurs ayant une température élevée assurent la diversité de la population alors que les processeurs qui ont une température faible protègent les individus favorables (bonnes solutions) ;
- les performances croissent avec la taille de la population ce qui est intéressant puisque ce n'est pas le cas des algorithmes génétiques et d'autres méthodes hybrides à partir de certaines tailles de population. D'ailleurs dans le cas du PRSA, l'étude expérimentale montre que les performances de l'algorithme diminuent dès que la taille de la population dépasse une centaine d'individus. Deux éléments semblent expliquer ce phénomène : le croisement et la sélection par schéma de température. Lorsque la taille de la population croît le nombre de combinaisons possibles au niveau génétique croît exponentiellement, ce qui revient à pouvoir explorer une plus grande partie de l'espace de recherche. D'autre part la sélection par schéma de température permet de maintenir la diversité de la population.

Le *Genetic Simulated Annealing* est donc un algorithme parallèle qui permet d'éviter l'optimisation préalable des paramètres en utilisant un schéma de température aléatoire. De plus son efficacité croît *a priori* avec la taille de la population (le nombre de processeurs) ce qui est très intéressant, surtout si l'implantation se fait sur une machine massivement parallèle.

Ces bonnes propriétés nous ont poussé à retenir cet algorithme comme représentant des méthodes hybrides parallèles. Nous allons l'étudier dans la suite.

Conclusion

Nous avons vu dans cette partie différents algorithmes d'optimisation globale. Cette diversité traduit notamment l'inexistence d'un algorithme d'optimisation « universel » permettant de résoudre n'importe quel problème d'optimisation. D'ailleurs à la question « Existe-t-il un algorithme d'optimisation donnant les meilleures performances quel que soit le problème considéré ? », Holpert et MacReady [61] ont répondu par le *No Free Lunch Theorem*, établissant qu'en « moyenne » sur l'ensemble de toutes les fonctions de coût possibles (noté $\mathcal{F}$), tous les algorithmes d'optimisation (sans exception) ont des performances équivalentes. Par exemple, si le recuit simulé donne de meilleurs résultats qu'un algorithme génétique sur un sous-ensemble $\phi \subset \mathcal{F}$ de fonctions de coût, alors inversement l'algorithme génétique sera plus performant sur $\mathcal{F} \backslash \phi$. En d'autres mots, il n'y a pas de « miracles » : pour obtenir une optimisation efficace il faut que l'algorithme d'optimisation appréhende les caractéristiques de l'espace de recherche et de la fonction de coût (ou d'énergie). Cela signifie aussi que si on considère un problème d'optimisation particulier, il existe bien un meilleur algorithme d'optimisation pour le résoudre. Cette partie fait également apparaître que la modélisation de phénomènes physiques ou biologiques a permis de définir de nombreux algorithmes d'optimisation.

Parmi les différents algorithmes abordés, le recuit simulé est sans aucun doute celui qui a fait l'objet des études les plus approfondies. De nombreux travaux ont établis des résultats théoriques concernant divers aspects (preuve de convergence, vitesse de convergence). De même, sur le plan du parallélisme, les différents modes de parallélisation possibles ont été intensément étudiés. En particulier des parallélisations induisant une chaîne de Markov différente de la version séquentielle, mais sans dégradation des performances, ont été introduites dans plusieurs travaux (algorithmes multi-températures). Enfin, certains chercheurs ont également proposés des algorithmes qui sont liés au recuit simulé (notamment le recuit simulé adaptatif et l'équation de la diffusion). On remarque en particulier qu'ils sont appuyés par une base théorique, d'ailleurs, certains tirent parti des résultats du recuit simulé. Bien entendu, étudier tous les algorithmes du type recuit simulé que nous avons vus n'est pas envisageable. Dans la suite nous nous focalisons sur deux algorithmes : le recuit simulé adaptatif et l'équation de la diffusion. En effet, ces deux algorithmes n'ont pas été autant étudiés que le recuit simulé « classique ». Par exemple, dans le cas du recuit simulé adaptatif, on ne sait pas si les algorithmes multi-températures restent valides. Pour l'équation de la diffusion, c'est la question des performances réelles de la parallélisation massive qui nous semble intéressante à étudier. Notamment dans la perspective d'une mise en œuvre sur machine MIMD. Finalement, il est à noter que ces deux algorithmes permettent de traiter directement des problèmes dont l'espace de recherche est continu, ce qui, comme nous allons le voir dans la partie qui suit, est le cas des problèmes que nous considérons.

La recherche tabou se caractérise par des parallélisations plus élémentaires, consistant en l'évolution parallèle de plusieurs instances de l'algorithme séquentiel (une par processeur) pouvant éventuellement échanger des informations (avec ou sans interactions). C'est pourquoi nous n'avons pas jugé intéressant de retenir cet algorithme. Un point à noter est l'absence d'un modèle théorique pouvant permettre de préciser le rôle des différents paramètres dans l'algorithme.

Cependant, de tous les algorithmes d'optimisation globale, c'est sans aucun doute le domaine des algorithmes évolutionnaires qui est le plus actif. Nous avons vu que plusieurs classes d'algorithmes reposant sur les mêmes concepts ont été développées ; initialement dans des perspectives assez différentes. L'attrait de ces algorithmes est, de plus, renforcé par leur parallélisme intrinsèque. De tous les algorithmes évolutionnaires, les plus connus sont les algorithmes génétiques, car leur définition permet de les appliquer à une grande variété de problèmes. Plus spécifiquement pour l'optimisation de variables réelles, les stratégies d'évolution et l'évolution différentielle se révèlent être des alternatives intéressantes aux algorithmes génétiques. Du point de vue de la parallélisation, celle-ci consiste à distribuer la population sur les différents processeurs. On distingue ainsi deux modèles de parallélisation : le modèle à gros grain (plusieurs individus par processeur) et le modèle à grain fin (un individu par processeur). Dans le premier cas, on peut voir globalement l'algorithme parallèle comme plusieurs instances de l'algorithme séquentiel évoluant en parallèle sur des sous-populations de taille réduite, avec ou sans interactions. En revanche, dans le second cas ce sont l'initialisation, l'évaluation et les différents opérateurs (croisement ou recombinaison, mutation et sélection) qui sont parallélisés. Naturellement au niveau des communications, le modèle à gros grain est beaucoup moins coûteux et, *a priori*, plus facile à mettre en œuvre que le modèle à grain fin. D'autre part, il faut remarquer que la très grande majorité des travaux ont portés sur les algorithmes génétiques. Aussi notre étude va-t-elle plus particulièrement s'intéresser aux stratégies d'évolution et à l'évolution différentielle. D'ailleurs, à notre connaissance aucune étude de la parallélisation de l'évolution différentielle n'a été menée à ce jour.

La définition d'algorithmes parallèles, hybrides de certains des algorithmes évoqués précédemment est une voie qui a également donné lieu à beaucoup de travaux. Les hybrides qui sont le plus communément considérés mettent en jeu des concepts du recuit simulé et des algorithmes génétiques. L'intérêt potentiel de cette association résulte en grande partie des caractéristiques complémentaires de ces deux algorithmes. Toutefois les bénéfices et les inconvénients réels sont difficiles à cerner. Par conséquent l'étude d'un hybride parallèle s'impose. Parmi les hybrides présentés, nous avons choisi le *Genetic Simulated Annealing* du fait des bonnes propriétés qu'il semble posséder.

Néanmoins si on considère un problème pratique particulier, il s'avère que le choix d'une méthode n'est généralement pas guidé par des règles précises, mais qu'il est fait de façon déterministe puis est validé empiriquement. L'objet du travail présenté est précisément de répondre aux questions suivantes :

- Quel est le comportement des algorithmes d'optimisation globale dans le cadre d'un problème particulier ?
- Quel est le mode de parallélisation le plus adéquat pour ces algorithmes, mis à part l'hybride parallèle, et quels sont les effets induits par la parallélisation ?

Avant d'aborder ces questions, nous allons tout d'abord faire dans la partie suivante une présentation succincte de la problématique du recalage en imagerie médicale. Le recalage rigide et le recalage déformable qui constituent les deux problèmes d'application sont présentés plus en détails.

Deuxième partie

Domaine d'application de l'étude : le recalage d'images médicales

Introduction

Une approche couramment utilisée pour comparer des algorithmes d'optimisation est l'étude de leur comportement sur des fonctions de coût (ou d'énergie) « synthétiques », i.e. ne modélisant pas un problème réel. Une suite de fonctions très connue est la suite introduite par De Jong dans sa thèse [37], mais il existe nombre d'autres fonctions (fonction de Ackley, de Griewank, etc) [139]. Plus généralement, le choix d'une fonction de coût ou sa définition ne sont pas aisés car il faut faire attention à éviter de nombreux pièges (symétrie, séparabilité, etc) [138] qui peuvent favoriser un algorithme par rapport à un autre. En fait cette approche n'est intéressante que si la fonction de coût synthétique approche « convenablement » la fonction de coût du problème que l'on cherche à résoudre, ou si elles ont des caractéristiques communes. D'ailleurs si on considère les performances de quelques algorithmes d'optimisation sur différentes fonctions de coût synthétiques on se rend rapidement compte que les résultats obtenus ne sont pas pertinents. Sur telle fonction on aura l'algorithme A qui sera meilleur que l'algorithme B, et inversement sur telle autre fonction. Il est donc préférable d'étudier directement les algorithmes d'optimisation sur le problème d'optimisation considéré.

Parmi les problèmes qui s'expriment sous la forme d'un problème d'optimisation on trouve de nombreux problèmes d'imagerie. De manière générale, ceux-ci consistent à déterminer le minimum global d'une fonction de coût décrivant les interactions entre les différentes variables modélisant les caractéristiques des images à traiter [56]. Ainsi, nous allons considérer la problématique du recalage en imagerie médicale, et plus particulièrement le recalage rigide et le recalage déformable. Ce domaine d'application est très intéressant car comme nous le verrons les fonctions de coût qui doivent être optimisées sont non linéaires, irrégulières et présentent de nombreux minima locaux. D'autre part, suivant la nature de la transformation recherchée pour effectuer le recalage, le nombre de variables à optimiser peut aller de moins d'une dizaine pour le recalage rigide à quelques centaines de milliers, voire plusieurs millions en recalage déformable (suivant la méthode considérée). L'intérêt pratique est de faciliter l'utilisation en routine clinique de certaines méthodes de recalage, i.e. de réduire leur temps de calcul, améliorer la qualité des images obtenues, permettre de traiter des images ayant une résolution plus élevée. De plus, bien que l'imagerie médicale constitue sans aucun doute le domaine d'application privilégié en raison du grand nombre d'applications médicales [53], d'autres domaines ont recours au recalage d'images (l'imagerie satellitaire, la robotique, l'imagerie militaire, etc).

Cette partie est composée de deux chapitres : dans un premier chapitre nous faisons un bref état de l'art du recalage en imagerie médicale. Nous présentons les divers critères qui définissent une méthode de recalage (nature des images, transformation recherchée, etc). Le second chapitre aborde plus en détail les méthodes de recalage choisies comme problèmes d'application des algorithmes d'optimisation étudiés (nombre de variables à optimiser, fonctions de coût).

Chapitre 5

État de l'art du recalage en imagerie médicale

L'objet de ce chapitre est de donner au lecteur une idée de la difficulté que pose le problème du recalage en imagerie médicale et de proposer un bref état de l'art structuré du domaine. Un grand nombre de méthodes de recalage ont été proposées. Une synthèse de techniques classiques de recalage a été faite par Brown [22] et une classification de ces méthodes dans le contexte de l'imagerie médicale a été donnée dans [134].

5.1 Objectif et définition du problème

Le recalage d'images consiste souvent à minimiser une fonction de coût ou fonction de similarité exprimant la similitude au niveau des voxels (des pixels en 2D) des images que l'on cherche à recaler. Plus précisément, l'objectif du recalage de deux ou plusieurs images est d'aligner géométriquement la ou les images à recaler sur une image de référence de façon à ce que les voxels (ou pixels) représentant les mêmes structures sous-jacentes soient superposés.

Pour définir une méthode de recalage, différents aspects doivent être pris en compte :

① La nature des images que l'on cherche à recaler (même modalité ou non) et les informations (primitives) qui seront utilisées pour effectuer le recalage.

② Le type de la transformation recherchée.

③ La mesure de la « ressemblance » (similarité entre l'image de référence et l'image à recaler).

④ L'algorithme de recherche de la transformation optimale.

La complexité d'une méthode de recalage est essentiellement liée à sa supervision ou non par l'utilisateur, au traitement d'images monomodales (provenant de la même modalité, i.e. source d'acquisition) ou multimodales, et au nombre de variables définissant la transformation.

5.2 Modalités et primitives utilisées

L'imagerie médicale se caractérise par une grande variété de modalités. Celles-ci sont liées à des mesures d'ordre physique ou chimique et sont classées suivant la nature de l'information qu'elles produisent.

On distingue :
- les modalités anatomiques (os et organes) : radiographie, tomodensitométrie (CT) et imagerie par résonance magnétique (IRM), échographie, images vidéo (laryngoscope, ...) ;
- les modalités fonctionnelles (activité du cerveau, débits) : scintigraphie, tomographie par émission de positons (TEP), tomographie par émission monophotonique (TEMP), imagerie fonctionnelle par résonance magnétique (IRMf), etc.

Certaines modalités ont un « effet biologique » significatif sur le patient, en particulier celles nécessitant l'absorption de substances ou l'introduction d'appareillage : elles sont dites invasives. C'est notamment le cas de la tomodensitométrie et des modalités de médecine nucléaire (TEP, TEMP) qui requièrent l'injection de substances radioactives. La figure 5.1 illustre cette diversité de modalités. Des coupes transversales d'images du crâne et du cerveau d'un même patient obtenues par quatre systèmes d'acquisition différents sont présentées.

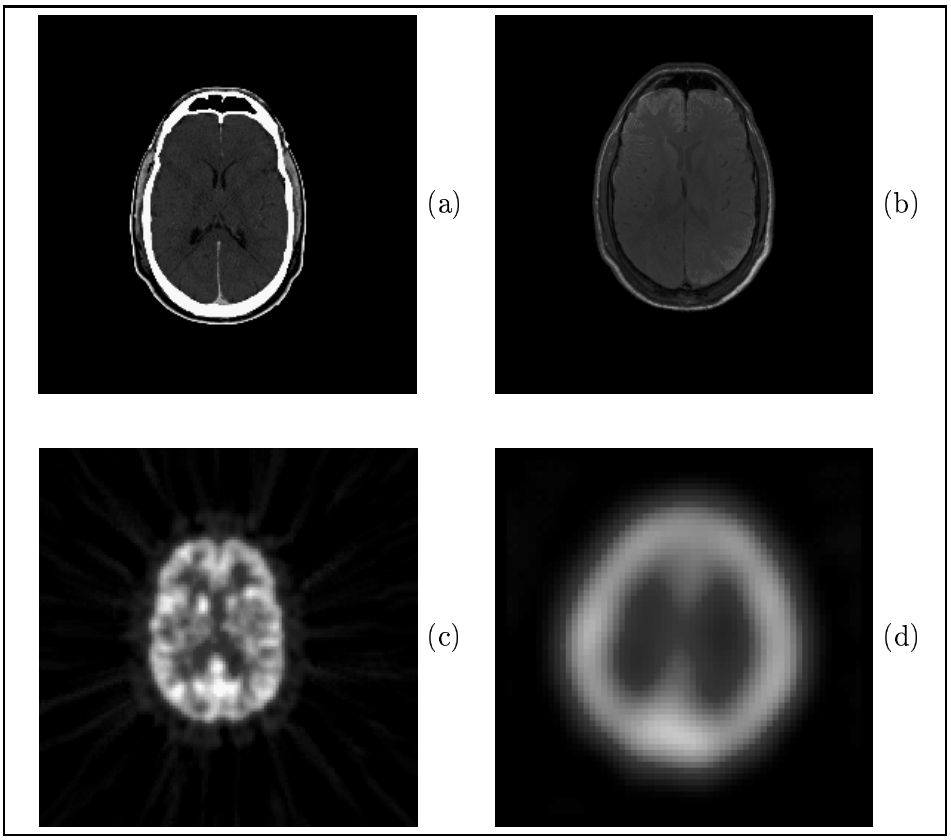


Fig. 5.1 – Images du crâne et du cerveau d'un même patient obtenues par : (a) tomodensitométrie, (b) imagerie par résonance magnétique, (c) tomograhie par émission de positons et (d) tomographie par émission monophotonique.

En ce qui concerne les primitives utilisées, les méthodes de recalage en imagerie médicale s'appuient soit sur des informations produites par des capteurs externes au patient, typiquement des marqueurs ou un cadre stéréotaxique fixés sur le crâne du patient, soit directement sur le contenu des images à recaler : contours, volumes, voxels, etc.

5.2.1 Recalage d'images monomodales

Le recalage monomodal est le plus simple. Il est essentiellement orienté vers le traitement d'images provenant du même patient afin de suivre l'évolution d'une pathologie, on parle alors de recalage intra-patient. Cependant le recalage inter-patient fait également appel au recalage monomodal dans le cadre d'études morphométriques (calcul de formes moyennes et variations autour de ces formes), notamment pour la construction d'atlas anatomiques probabilistes [54, 129]. La « simplicité » relative de ce type de recalage vient du fait que les images présentent des informations identiques. En conséquence il est possible de travailler directement sur les niveaux de gris. D'autre part dans le cas du recalage intra-patient il n'y a pas de variation anatomique entre les images. De fait l'objectif du recalage dans ce cas est de compenser le changement de position du patient lors des différentes acquisitions. C'est pourquoi le recalage intra-patient consiste souvent en un recalage rigide, ou à déterminer des transformations simples permettant d'adapter les géométries des différents systèmes d'acquisition.

5.2.2 Recalage d'images multimodales

Plus complexe à mettre en œuvre que le recalage monomodal, le recalage multimodal a pour objet la fusion d'informations, en combinant des données de nature différente. Par exemple, le recalage IRM/TEMP permet de bénéficier de la bonne localisation spatiale des IRM, une propriété que ne possèdent pas les images fonctionnelles en raison d'une mauvaise résolution. Finalement, il est à remarquer que le recalage monomodal peut être considéré comme un cas particulier du recalage multimodal.

5.3 Transformations géométriques

Différentes transformations peuvent être considérées, leur action sur l'image à recaler pouvant être soit globale, soit locale (variable d'une région à une autre de l'image). Une transformation peut être rigide, affine, projective ou déformable [79]. La transformation rigide n'autorise que des translations et des rotations, alors que la transformation affine conserve l'alignement et le parallélisme. Pour ce qui est des transformations projectives, elles permettent la mise en correspondance de lignes non parallèles entre elles et ne conserve l'alignement que pour les droites horizontales ou verticales. Enfin, une transformation déformable permet de transformer des lignes droites en courbes. La figure 5.2 illustre les diverses transformations en dimension deux (inspirée d'une figure de [79]). À noter que l'on trouve d'autres classifications des transformations [5, 74], notamment en deux classes : transformations rigides ; transformations non rigides ou élastiques.

Naturellement, déterminer une transformation rigide définie par 6 variables (en 3D) ou une transformation affine qui a 12 degrés de liberté est plus aisé que de trouver une transformation déformable [79, 74] qui peut être définie par plusieurs dizaines de milliers, voire plusieurs millions de variables. Les modèles de déformation physiques, issus de la mécanique des fluides, développés par Christensen *et al.* [32], et utilisés en recalage déformable constituent un exemple typique de la complexité qui peut être atteinte dans ce contexte. Ainsi pour des images 3D de taille $128 \times 128 \times 100$ voxels, le modèle de la mécanique des fluides se traduit par l'estimation d'un vecteur de $9,8 \times 10^6$ variables, ce qui sur machine monoprocesseur nécessite

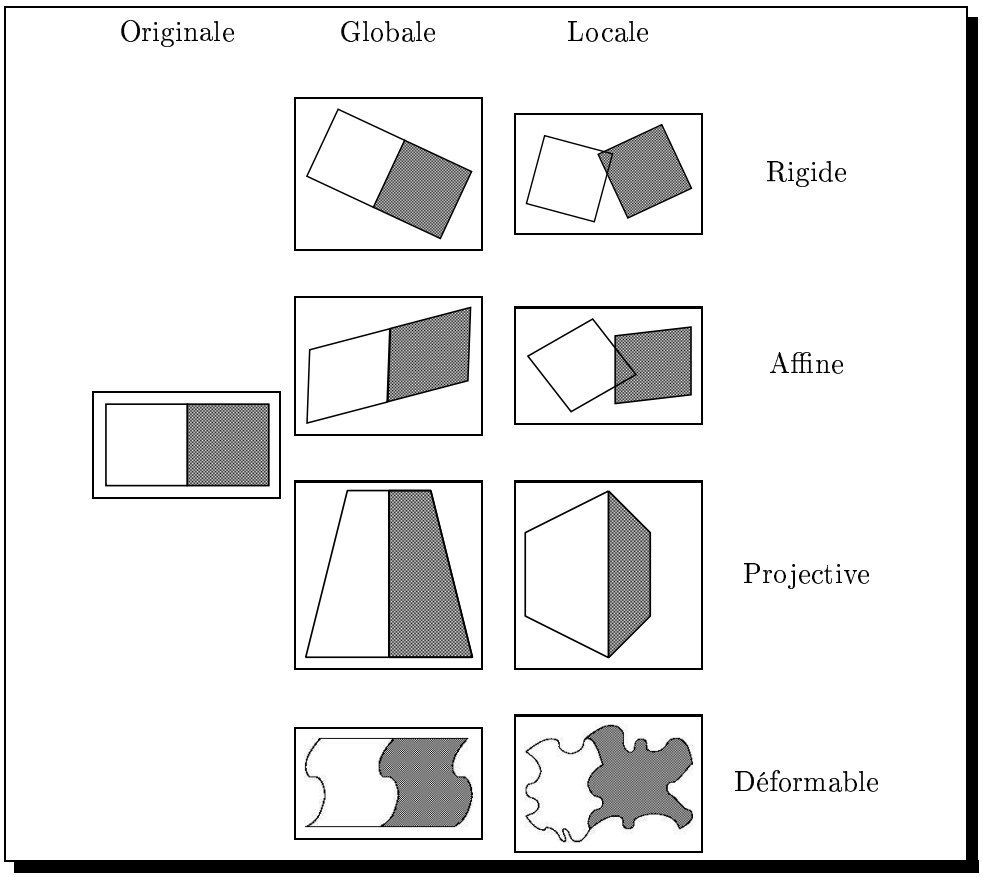
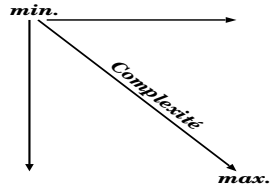


FIG. 5.2 – Exemples de transformations en 2D.

plusieurs jours. Sa parallélisation sur une MasPar MP2 a permis de réduire le temps de calcul à moins de deux heures [32]. Plus récemment, Christensen [31] a montré que dans le cas du modèle élastique, le temps de calcul obtenu sur la MasPar (128×128 processeurs) peut être divisé par $4,3$ en passant sur une machine SGI Power Challenge (16 processeurs). Il est donc clair que définir un algorithme de recalage déformable compatible avec une utilisation en routine clinique est difficile. Néanmoins, récemment Musse *et al.* [88] ont défini une méthode de recalage déformable s'appuyant sur une analyse multirésolution permettant d'obtenir d'excellents résultats en approximativement une demi-heure en calcul séquentiel. Cette méthode de recalage induit l'optimisation d'un nombre croissant de variables, mais sans commune mesure avec le modèle fluide de Christensen : 3 variables, puis 81, 1029, 10125 et 89373 pour des images de 128^3 voxels. À noter que certaines méthodes de recalage élastique ont également un temps de calcul qui n'est que de quelques minutes [101].

En définitive on peut résumer la complexité du recalage en imagerie médicale suivant le type de la transformation recherchée et la nature des images à recaler par le tableau ci-contre.

Complexité du recalage	Recalage d'images (monomodales multimodales)
Transformation : Globale { - Rigide, - Affine } ; Locale - Déformable	

TAB. 5.1 – Complexité du recalage suivant la transformation et la nature des images.

5.4 Critère de similarité

La façon de quantifier la similarité de deux images dépend des primitives utilisées par la méthode recalage. Certaines techniques utilisent des critères évaluant la correspondance de points extrémaux, lignes de crêtes, surfaces ou volumes, ce qui nécessite un pré-traitement des images (segmentation) [38, 101]. L'autre possibilité est l'utilisation directe des données (niveaux de gris des voxels) des différentes images [92], on parle alors de recalage dense (*voxel/pixel based*).

Parmi les approches ayant donné des résultats satisfaisants dans le cas d'images très dissemblables, beaucoup font appel à une fonction de similarité. Ces méthodes consistent à minimiser une fonction de coût exprimant la similarité au niveau des voxels (ou pixels) entre les différentes images, ce qui implique l'utilisation d'une technique d'optimisation. Des métriques évaluant la similarité des images ont été proposées pour le recalage monomodal et multimodal [77, 92, 136, 140]. Les fonctions de similarité classiques sont généralement associées à une formulation au sens des moindres carrés ou à la maximisation de la corrélation entre les images [100]. Certaines fonctions utilisent également des informations statistiques telles que la moyenne, la variance, l'entropie ou l'information mutuelle [57, 77].

Nous donnons ci-après quelques exemples de fonctions de similarité « classiques ». Elles sont supposées quantifier la similarité entre une image de référence $I(.)$ et une image à recaler $J(.)$ suivant une transformation T (cette dernière est quelconque).

Fonction de similarité quadratique

Dans le cadre du recalage monomodal, une fonction qui est communément utilisée est la fonction de similarité quadratique. Cette dernière suppose que les deux images à recaler ne diffèrent que par un bruit gaussien additif. Soit s le vecteur 3D des coordonnées spatiales, la fonction de coût à optimiser s'écrit alors :

$$C\left(I\left(.\right),J\left(T\left(.\right)\right)\right)=\sum_{s}\left[I\left(s\right)-J\left(T\left(s\right)\right)\right]^{2} \tag{5.1}$$

Comme le souligne Brown [22], la fonction de similarité quadratique comporte de nombreux minima locaux et une forte non linéarité.

Uniformité inter-images

Cette fonction de similarité, proposée par Woods *et al.* [140], est très populaire pour le recalage d'images multimodales. Elle repose sur l'hypothèse d'uniformité inter-image pour une

paire d'images multimodales : à toute région homogène dans l'image de référence correspond, après recalage, une région également uniforme dans l'image recalée. Pour cela, $I(.)$ est dans un premier temps partitionnée en G classes de niveaux de gris, G étant égal au nombre de niveaux de gris présents dans $I(.)$. La partition obtenue est alors projetée sur l'image à recaler. Puis on calcule la valeur moyenne μ_g et la variance σ_g^2 de chacune des régions dans $J(T(.))$, $g \in \{1, ..., G\}$. L'hypothèse d'uniformité inter-image peut alors se traduire de la manière suivante : si les deux images sont correctement recalées, la variance normalisée de chacune des G régions est minimale dans l'image à recaler. La fonction de coût à minimiser est :

$$C\left(I\left(.\right), J\left(T\left(.\right)\right)\right) = \sum_{g=1}^{G} \frac{N_g \sigma_g(T(.))}{N \mu_g(T(.))} \tag{5.2}$$

où

$$\sigma_g\left(T\left(.\right)\right) = \sqrt{\sum_{s|I(s)=g} \left[J\left(T\left(s\right)\right) - \mu_g\left(T\left(.\right)\right)\right]^2} \tag{5.3}$$

et

$$\mu_g\left(T\left(.\right)\right) = \frac{1}{N_g} \sum_{s|I(s)=g} J\left(T\left(s\right)\right) \tag{5.4}$$

N et N_g représentent respectivement le nombre total de voxels dans une image et le nombre de voxels ayant le niveau de gris g dans $I(.)$. Bien entendu, dans le cas d'images très dissemblables, l'hypothèse d'uniformité inter-images sur laquelle s'appuie cette fonction n'est plus totalement vérifiée.

Information mutuelle

L'information mutuelle [77, 136] fait partie des mesures de similarité reposant sur l'entropie et exploitant l'histogramme conjoint des images. L'hypothèse est que l'information mutuelle $\mathcal{I}$ est maximale lorsque les images sont correctement recalées. Formellement la fonction devant être minimisée est :

$$\begin{aligned} C\left(I\left(.\right), J\left(T\left(.\right)\right)\right) &= -\mathcal{I}\left(I\left(.\right) - J\left(T\left(.\right)\right)\right), & (5.5) \\ &= -\sum_{g=1}^{G}\sum_{k=1}^{K} p(g,k) \log\left(\frac{p(g,k)}{p(g)p(k)}\right), & (5.6) \end{aligned}$$

où G et K représentent respectivement le nombre de niveaux de gris de $I\left(.\right)$ et $J\left(.\right)$; $p(g)$ est la probabilité qu'un voxel de l'image de référence a le niveau de gris g, $p(k)$ est la probabilité qu'un voxel de l'image à recaler a le niveau de gris k ; $p(g,k)$ est la probabilité qu'un voxel de l'image de référence a le niveau de gris g alors que le même voxel dans l'image à recaler a le niveau de gris k ($p(g,k)$ correspond aux éléments de l'histogramme conjoint de $I\left(.\right)$ et de $J\left(T\left(.\right)\right)$). Au niveau des performances, il est « admis » que l'information mutuelle donne de meilleurs résultats que l'uniformité inter-images en recalage d'images multimodales.

Ainsi que nous l'avons vu, chaque mesure de similarité repose sur une hypothèse au niveau des images. La fonction de similarité quadratique n'admet par exemple qu'un bruit gaussien comme

différence entre les deux images après recalage. Lorsque les images « violent » l'hypothèse, le recalage que l'on obtient peut s'avérer imprécis, voire erroné. Aussi, Nikou *et al.* [92, 91] ont proposé d'introduire des estimateurs robustes dans les fonctions de similarité, afin de tolérer la présence de voxels qui ne sont pas conformes à l'hypothèse sous-jacente à la fonction de coût considérée. En effet, un estimateur robuste permet de tolérer la présence d'une proportion plus ou moins importante (cette caractéristique s'appelle le point de rupture ; elle dépend de la fonction d'influence de l'estimateur) de données aberrantes dans les images sans influence sur le résultat du recalage.

La figure ci-dessous, tirée de [91], illustre le recalage rigide multimodal IRM/TEMP.

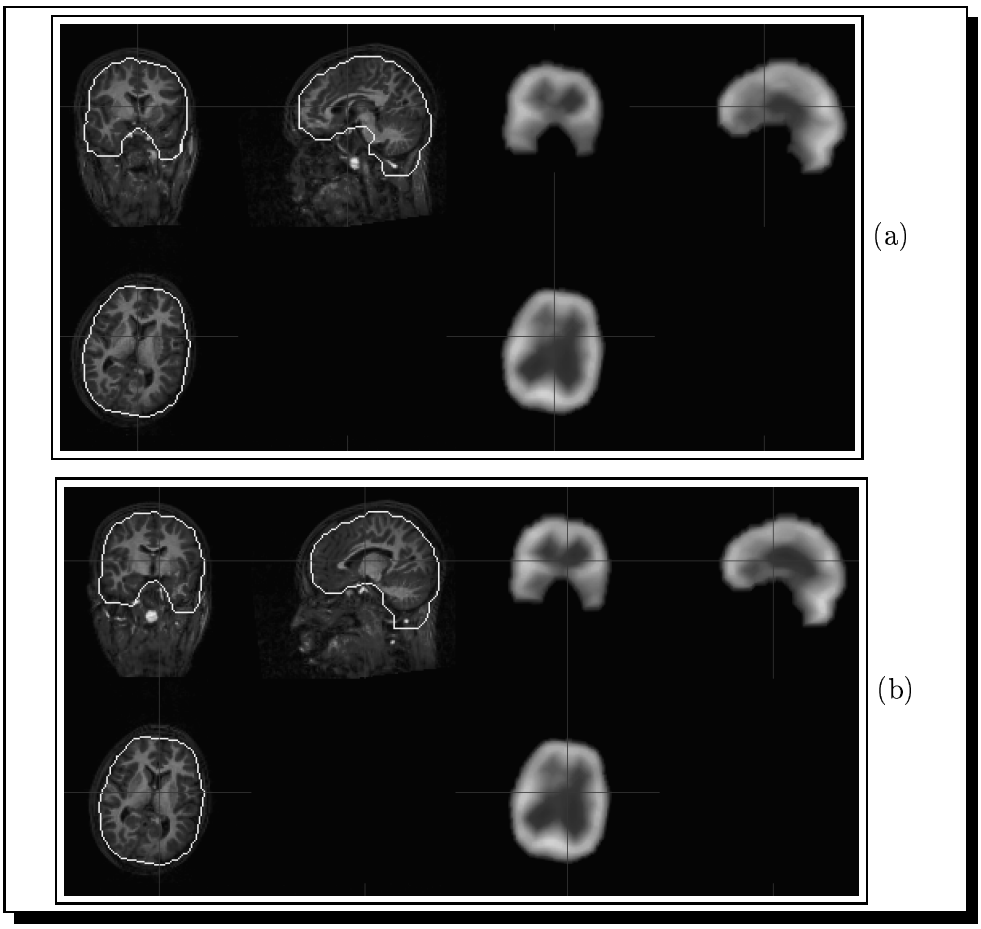


Fig. 5.3 – Exemples de recalage rigide d'images multimodales (IRM/TEMP) : (a) recalage obtenu par l'information mutuelle, (b) recalage obtenu par la version robuste de l'uniformité inter-images proposée par Nikou *et al.* [91] (visualisation multiplanaire ; les contours du cerveau dans l'image TEMP sont superposés sur l'IRM).

5.5 Recherche de la transformation optimale

Il existe une grande variété de techniques pour rechercher les paramètres décrivant la transformation optimale. Celles-ci ne sont d'ailleurs pas uniquement utilisées pour les approches basées sur la minimisation d'une fonction de coût, puisque les méthodes de recalage s'exprimant au moyen d'équations aux dérivées partielles ou dans le cadre de l'estimation bayésienne y font aussi appel.

En général, en recalage monomodal, les fonctions de coût sont moins complexes que celles nécessaires au recalage multimodal. Dans la plupart des méthodes de recalage, les auteurs s'appuient sur une méthode déterministe (simplex, Newton-Raphson ou descente du gradient par exemple). L'inconvénient majeur de ces algorithmes est leur sensibilité aux minima locaux, aussi pour éviter ce problème, une solution possible est une initialisation proche de l'optimum global. Nikou *et al.* [92] utilisent ainsi une optimisation stochastique rapide par recuit simulé pour guider la recherche déterministe. Une autre alternative est d'utiliser un algorithme d'optimisation globale pour l'ensemble du processus d'optimisation. On trouve ainsi des approches s'appuyant sur le recuit simulé [92], les algorithmes génétiques [101] (remarquons cependant que les auteurs estiment qu'il faut raffiner le résultat obtenu par une optimisation locale), les stratégies d'évolution [38], etc (voir l'article de synthèse [79]). Bien que le bénéfice en recalage multimodal d'une technique globale est évident, ces dernières souffrent de leur coût calculatoire, ceci d'autant plus que le nombre de variables à optimiser est important. C'est pourquoi l'utilisation des méthodes globales est généralement cantonné à des transformations qui sont décrites par très peu de variables [92]. Leur emploi en recalage déformable est d'ailleurs quasi inexistant.

La prépondérance des méthodes déterministes s'explique également par la possibilité de réduire leur sensibilité aux minima locaux et d'accélérer leur traitement au moyen d'approches hiérarchiques. Ainsi, dans leur article de synthèse sur le recalage non rigide hiérarchique, Lester *et al.* [74] distinguent trois niveaux auxquels on peut introduire une hiérarchie :

- Tout d'abord on peut considérer une hiérarchie au sein des algorithmes de recalage. Cela signifie que les images sont alors recalées en appliquant successivement différentes techniques de recalage estimant des transformations de plus en plus complexes (le nombre de degrés de liberté croît au fur et à mesure des itérations). Typiquement on commence par effectuer un recalage rigide, puis rigide avec zoom, ensuite affine, pour terminer avec l'estimation de déformations qui sont initialement globales puis raffinées pour se limiter à des zones de plus en plus réduites des images.
- On peut également choisir une approche hiérarchique sur les données images. Le processus d'optimisation n'est alors plus mené à résolution constante, mais suivant une approche consistant à traiter les images à des résolutions croissantes. Pour obtenir les images aux différentes résolutions on a généralement recours à des transformations pyramidales [9] (pyramide gaussienne, laplacienne, voire ondelettes).
- Enfin il est possible de hiérarchiser les déformations elles-mêmes. Dans ce cas deux types d'approches paramétriques peuvent être envisagées : une paramétrisation du champ de déformation en le décomposant sur une base de fonctions dont le nombre croît progressivement (Fourier, bases d'ondelettes, etc), ou une augmentation du nombre de nœuds utilisés pour interpoler la déformation.

Chapitre 6

Méthodes de recalages considérées

Dans ce chapitre nous allons présenter plus en détails les deux problèmes de recalage que nous allons considérer dans la suite. Ceux-ci sont représentatifs des problèmes à traiter : transformation rigide d'une part, transformation déformable de complexité moyenne d'autre part (plusieurs dizaines de milliers, voire centaines de milliers de paramètres). Ces deux transformations sont complémentaires car le recalage rigide est l'étape préalable à la recherche de toute autre transformation. Enfin, il est à noter que dans le cadre rigide, nous nous plaçons dans le contexte du recalage monomodal et multimodal, tandis que le recalage non rigide est restreint, dans notre étude, au traitement d'images monomodales.

6.1 Recalage rigide

Le recalage rigide par fonction de similarité consiste à estimer les paramètres Θ d'une transformation rigide T_Θ minimisant une fonction de coût $C\left(I\left(.\right), J\left(T_\Theta\left(.\right)\right)\right)$, évaluant quantitativement la « ressemblance » entres les images I et J :

$$\Theta_{\min} = \arg\ \min\left[C\left(I\left(.\right), J\left(T_\Theta\left(.\right)\right)\right)\right], \tag{6.1}$$

où Θ est le vecteur des paramètres de translation et de rotation :

$$\Theta = (t_x, t_y, t_z, \theta_x, \theta_y, \theta_z)^T \ \text{(pour des images 3D)}, \tag{6.2}$$

où $(t_x, t_y, t_z)^T$ est le vecteur décrivant la translation suivant les axes X, Y et Z ; $(\theta_x, \theta_y, \theta_z)^T$ spécifie les angles de rotation autour de ces axes ; $I\left(.\right)$ est l'image de référence et $J\left(.\right)$ est l'image devant être recalée.

Formulation mathématique de la transformation

On peut décrire plus précisément la transformation rigide T_Θ de centre $g = (x_g, y_g, z_g)^T$ (dans le cas des IRM que nous traitons il s'agit du centre de gravité du volume d'intérêt, i.e. le cerveau) au moyen d'une expression matricielle. En notant $s = (x, y, z)^T$ le vecteur 3D des coordonnées

spatiales dans I et $s' = (x', y', z')^T$ son image par la transformation rigide on a :

$$\begin{pmatrix} x' \\ y' \\ z' \\ 1 \end{pmatrix} = \left(\begin{array}{ccc|c} & R & & \begin{matrix} t_x \\ t_y \\ t_z \end{matrix} \\ \hline 0 & 0 & 0 & 1 \end{array}\right) \times \begin{pmatrix} x - x_g \\ y - y_g \\ z - z_g \\ 1 \end{pmatrix} \tag{6.3}$$

où R est une matrice 3×3 vérifiant $R = R_x \times R_y \times R_z$,

$$R_x = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\theta_x & -\sin\theta_x \\ 0 & \sin\theta_x & \cos\theta_x \end{pmatrix}, R_y = \begin{pmatrix} \cos\theta_y & 0 & \sin\theta_y \\ 0 & 1 & \\ -\sin\theta_y & 0 & \cos\theta_y \end{pmatrix}, R_z = \begin{pmatrix} \cos\theta_z & -\sin\theta_z & 0 \\ \sin\theta_z & \cos\theta_z & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Fonctions de similarité retenues

Pour le recalage rigide, nous allons considérer la fonction de similarité quadratique (5.1) et l'information mutuelle (5.5). La première est communément utilisée pour le recalage monomodal, tandis que la seconde est une fonction représentative de celles traitant le cas multimodal (bien que nous ne traitions ici que le recalage monomodal IRM/IRM). De plus, comme les algorithmes d'optimisation présentés dans la première partie sont généralement indépendants de la fonction de coût, le passage d'une fonction de similarité à une autre est aisé.

6.2 Recalage déformable

Le recalage déformable, ou non rigide, est un domaine de recherche très actif. On distingue deux catégories d'approches : d'une part les approches qui reposent sur des primitives géométriques (points, surfaces, lignes de crêtes, etc) et d'autre part les approches denses. Il est à noter que les méthodes utilisant des primitives géométriques semblent moins performantes que les méthodes denses du fait de la perte d'information. Pour ce qui est des approches denses, celles-ci s'appuient souvent sur la modélisation de propriétés physiques de matériaux ou de fluides. Les travaux qui font référence dans ce domaine sont ceux de Christensen *et al.* [33]. Dans leur approche le cerveau du patient est successivement « vu » comme un matériau élastique puis un fluide, les déformations sont alors régies respectivement par l'équation de Navier et celle de Navier-Stokes (viscosité constante). L'avantage du modèle fluide est la possibilité d'estimer de grandes déformations tout en préservant la topologie (point important en recalage déformable), alors que l'élasticité linéaire n'autorise que de « petites » déformations. Plus récemment le modèle élastique fut utilisé avec succès pour le recalage d'images non cérébrales, en l'occurrence le recalage non rigide de mammographies 2D d'une même patiente [99] (vues des deux seins ou d'un même sein, mais séparées par un laps de temps). L'inconvénient majeur des approches basées sur un modèle de déformation fluide est leur coût calculatoire : plusieurs jours de calcul en séquentiel pour des images 3D, qui peut être ramené à quelques heures au moyen d'une implantation parallèle [32], ce qui reste toutefois rédhibitoire pour des applications en routine clinique. Pour pallier les inconvénients des différentes approches, la tendance actuelle est de considérer une approche hiérarchique (cf. le dernier paragraphe de la section 5.5). Cela permet d'une part de réduire la complexité calculatoire du recalage, d'autre part l'algorithme est dans ce cas beaucoup moins sensible aux minima locaux [56].

Dans ce travail nous avons retenu la méthode de recalage non rigide développée par Musse *et al.* [87, 88, 89]. Cette dernière fait partie de la famille des approches paramétriques qui décomposent le champ de déformation sur des bases hiérarchiques de fonctions. De fait, le champ de déformation est décomposé sur une base de fonctions à support compact en se plaçant dans le cadre théorique de l'analyse multirésolution des signaux d'énergie finie. Le modèle du champ de déformation est alors un modèle continu, la transformation correspondant initialement à une représentation globale qui est progressivement raffinée pour devenir locale.

Comme beaucoup d'autres méthodes de recalage, l'approche considérée repose sur la minimisation d'un critère de similarité évaluant qualitativement la « ressemblance » des images. En recalage non rigide une fonction de similarité qui est communément utilisée est l'erreur quadratique [33], le problème posé consiste ainsi à estimer le champ de vecteurs de déplacement d qui minimise la fonction de coût (ou d'énergie) suivante :

$$C(d) = \int_{\Omega} |I(s) - J(s + d(s))|^2 ds, \tag{6.4}$$

où Ω est le support des images ; $I(.)$ est l'image de référence et $J(.)$ est l'image devant être recalée ; s définit la position d'un voxel dans le domaine Ω. Par conséquent on a :

$$d_{\min} = \arg\ \min\left[C(d)\right], \tag{6.5}$$

d est supposé appartenir à un espace hilbertien.

La transformation T_d qui permet de recaler deux images est ainsi définie par :

$$\begin{aligned} T_d : \mathbb{R}^3 &\rightarrow \mathbb{R}^3 \\ s = (x, y, z)^T &\mapsto (x + d_x(s), y + d_y(s), z + d_z(s))^T = (x', y', z')^T = s' \end{aligned} \tag{6.6}$$

T_d appartient à l'ensemble des transformations de coût (ou énergie) fini(e) de $\mathbb{R}^3$ dans $\mathbb{R}^3$.

Avant de présenter le modèle de déformation paramétrique, nous décrivons brièvement l'analyse multirésolution sur laquelle le modèle s'appuie.

6.2.1 Analyse multirésolution

L'analyse multirésolution est décrite dans beaucoup de travaux [9, 67], en particulier ceux ayant trait aux ondelettes et notamment l'article de Mallat [80] qui est une référence dans ce domaine. Nous allons limiter notre présentation aux signaux à une dimension. En effet, l'extension aux signaux de dimension deux et trois se fait ici simplement par produit tensoriel d'une analyse multirésolution de signaux monodimensionnels.

On se place dans $L^2(\mathbb{R})$, l'espace vectoriel des fonctions à une variable $f(x)$, mesurables et de carré intégrable. Cet espace hilbertien est muni d'une norme. La transformée de Fourier d'une fonction, de même que l'opération de convolution y sont définies. Enfin, si une famille de vecteurs $\{f_n\}$ est un cadre de référence (*frame*) de $L^2(\mathbb{R})$, elle définit une base de Riesz si toute fonction $f(x)$ de l'espace peut s'écrire de façon unique sous la forme d'une combinaison linéaire des vecteurs $\{f_n\}$.

Considérons $\mathcal{A}^l$ l'opérateur d'approximation du signal $f(x) \in L^2(\mathbb{R})$ à la résolution 2^l (l est appelé le niveau de résolution), on note $\mathcal{A}^l f(x)$ le signal résultant de l'application de cet opérateur

sur $f(x)$. On commence par caractériser cet opérateur par l'intermédiaire des propriétés que l'on en attend :

1. $\mathcal{A}^l$ est un opérateur de projection orthogonale. En effet c'est un opérateur de projection sur un sous-espace vectoriel particulier $V_l \in L^2(\mathbb{R})$ qui peut être vu comme l'espace de toutes les approximations possibles à la résolution 2^l de fonctions de $L^2(\mathbb{R})$, i.e.

$$\mathcal{A}^l \circ \mathcal{A}^l = \mathcal{A}^l \tag{6.7}$$

De plus $\mathcal{A}^l f(x)$ est l'approximation la plus proche de $f(x)$

$$\forall g(x) \in V_l, \|g(x) - f(x)\| \geq \|\mathcal{A}^l f(x) - f(x)\| \tag{6.8}$$

2. $\mathcal{A}^{l+1}$ contient toute l'information nécessaire au calcul du même signal à la résolution inférieure $\mathcal{A}^l$, c'est le principe de causalité

$$\forall l \in \mathbb{Z}, V_l \subset V_{l+1} \tag{6.9}$$

On en déduit également qu'il est alors possible de construire V_{l+1} en dilatant d'un facteur deux tous les éléments de V_l, soit

$$\forall l \in \mathbb{Z}, f(x) \in V_l \Leftrightarrow f(2x) \in V_{l+1} \tag{6.10}$$

3. La translation de la fonction $f(x)$ se traduit par une translation identique à la résolution 2^l si et seulement si la valeur de la translation est un multiple de 2^{-l}, d'où

$$f(x) \in V_0 \Leftrightarrow f(x+1) \in V_0 \tag{6.11}$$

Soit aussi

$$\forall k, l \in \mathbb{Z}, \mathcal{A}^l \left(f\left(x - k2^{-l}\right)\right) = \left(\mathcal{A}^l f\right)\left(x - k2^{-l}\right) \tag{6.12}$$

4. Lorsque le niveau de résolution converge vers $+\infty$ l'approximation du signal converge vers le signal original, inversement quand le niveau de résolution converge vers $-\infty$ l'approximation du signal converge vers zéro. Ce principe s'écrit

$$\lim_{l \to +\infty} V_l = \bigcup_{l=-\infty}^{+\infty} V_l \text{ est dense dans } L^2(\mathbb{R}) \text{ et } \lim_{l \to -\infty} V_l = \bigcap_{l=-\infty}^{+\infty} V_l = 0 \tag{6.13}$$

5. Les bases de fonctions des espaces V_l doivent pouvoir être construites par translation et contraction/dilatation d'une seule fonction appelée fonction d'échelle. De fait s'il existe une fonction d'échelle $\phi \in V_0$, d'intégrale non nulle et telle que l'ensemble des fonctions $\{\phi(x-k) \mid k \in \mathbb{Z}\}$ est une base de Riesz de V_0, on obtient en utilisant (6.10) que la suite de fonctions

$$\left(\phi_i^l(x) = 2^{\frac{l}{2}} \phi(2^l x - i)\right)_{i \in \mathbb{Z}} \tag{6.14}$$

est une base de Riesz de V_l.

En conclusion :
- La collection d'opérateurs $\mathcal{A}^l$ associée, qui satisfait les propriétés données précédemment, permet d'obtenir l'approximation de toute fonction de $L^2(\mathbb{R})$ à la résolution 2^l.
- Une collection d'espaces vectoriels $(V_l)_{l\in\mathbb{Z}}$ satisfaisant les propriétés (6.9) à (6.14) définit une analyse multirésolution de $L^2(\mathbb{R})$.

Il est à noter que lorsque $f(x)$ a un domaine de définition fini $\mathcal{D}_f$ (par exemple $[0;1]$), seules les fonctions de base $\left(\phi_i^l(x)\right)_{i\in\mathbb{Z}}$ de V_l dont le support est entièrement inclus dans $\mathcal{D}_f$ seront considérées. De même, si la fonction d'échelle ϕ est à support compact, chaque espace V_l sera de dimension finie m_l.

Dès lors qu'une analyse multirésolution est définie, on peut obtenir la décomposition hiérarchique de tout signal $f(x)$ en le projetant sur les différents sous-espaces vectoriels $V_l, l \in \mathbb{Z}$. Le signal approché au niveau de résolution l est alors caractérisé numériquement par les coefficients résultant de la projection (on calcule le produit scalaire), soit à la résolution 2^l :

$$\mathcal{A}^l f = A^l \bullet \Phi^{l^T} \tag{6.15}$$

où $\Phi^l = \left(\phi_0^l, \phi_1^l, ..., \phi_{m_l-1}^l\right)$ définit une base de V_l et $A^l = \left(a_0^l, a_1^l, ..., a_{m_l-1}^l\right)$ est le vecteur contenant les coefficients correspondants.

Comme le montre Mallat dans son article [80], le calcul du vecteur de coefficients à la résolution 2^{l+1} à partir du vecteur à la résolution 2^l est obtenu directement par convolution du vecteur A^l avec un filtre H après avoir préalablement inséré des zéros entre chaque coefficient ($\uparrow_2$). Inversement, pour déterminer la décomposition du signal $f(x)$ à la résolution 2^l on convolue tout d'abord A^{l+1} avec le filtre $\hat{H}$ qui est le miroir de H, puis on fait un sous-échantillonnage à la cadence deux ($\downarrow_2$). Cette procédure de transition entre deux résolutions successives est directement induite par la propriété de causalité. Formellement :

$$A^{l+1} = (\uparrow_2 (A^l)) * H, A^l = \downarrow_2 ((A^{l+1}) * \hat{H}) \tag{6.16}$$

Les filtres $H = (h_k)_{k\in\mathbb{Z}}$ et $\hat{H} = (h_{-k})_{k\in\mathbb{Z}}$ sont dérivés de l'équation de dilatation ou raffinement :

$$\phi(x) = \sum_k h_k \phi(2x - k) \tag{6.17}$$

Les filtres H et $\hat{H}$ sont de réponse impulsionnelle finie lorsque ϕ est à support compact. Il est à remarquer qu'en précisant les propriétés que doit respecter la séquence $(h_k)_{k\in\mathbb{Z}}$ (appelée filtre miroir conjugué) on induit des contraintes sur la fonction d'échelle, et inversement puisque H permet de déterminer la transformée de Fourier de ϕ (voir [80]).

6.2.2 Modélisation hiérarchique du champ de vecteurs de déplacement en 3D

Pour représenter hiérarchiquement le champ continu de vecteurs de déplacement $d : \Omega = [0;1]^3 \rightarrow \mathbb{R}^3$ on va considérer une analyse multirésolution de $L^2(\mathbb{R}^3)$. Comme nous l'avons déjà souligné on obtient l'analyse multirésolution désirée par simple produit tensoriel d'une analyse multirésolution de $L^2(\mathbb{R})$. La fonction d'échelle $\phi_{3D}(x, y, z)$ est donc définie sous la forme du produit suivant :

$$\phi_{3D}(x, y, z) = \phi(x)\phi(y)\phi(z) \tag{6.18}$$

où $\phi(x)$ est la fonction d'échelle de l'analyse multirésolution $L^2(\mathbb{R})$.

On en déduit que la suite de fonctions :

$$\Phi^l = \left(\phi^l_{i,j,k}(x,y,z) = 2^{\frac{3l}{2}}\phi(2^l x - i)\phi(2^l y - j)\phi(2^l z - k)\right)_{i,j,k\in\{0,\ldots,m_l-1\}} \tag{6.19}$$

définit une base du sous-espace vectoriel V_l.

Le traitement du vecteur de déplacement d se fait alors en considérant la décomposition de chacune de ses trois composantes. Au niveau de résolution l on a donc :

$$d^l(s) = d^l(x,y,z) = \begin{cases} d^l_x(x,y,z) = A^l_x \bullet {\Phi^l}^T \\ d^l_y(x,y,z) = A^l_y \bullet {\Phi^l}^T \\ d^l_z(x,y,z) = A^l_z \bullet {\Phi^l}^T \end{cases} \tag{6.20}$$

où A^l_x, A^l_y et A^l_z sont comme dans (6.15) les matrices de coefficients. Le passage d'une résolution à une autre est obtenu en étendant au cas 3D les relations (6.16).

6.2.3 Propriétés du modèle

Nous avons vu dans la section précédente que le champ de vecteurs de déplacement d est projeté sur une suite de sous-espaces vectoriels V_l. Or les fonctions de base de V_l sont toutes obtenues à partir de la fonction d'échelle ϕ (voir 6.19). En conséquence, les propriétés du modèle de déformation hiérarchique (le champ d et la transformation T_d) résulteront des propriétés de la fonction d'échelle. C'est pourquoi la fonction d'échelle est choisie de manière à avoir les caractéristiques suivantes : continuité, différentiabilité et support compact. La dernière propriété est importante pour représenter les déformations sur un domaine Ω borné. En effet les problèmes aux bords sont alors facilement résolus en ne considérant que les fonctions de base qui sont entièrement incluses dans Ω (les déformations sont supposées nulles sur le bord du volume 3D). De plus, les fonctions d'échelle à support compact induisent une bonne localisation spatiale.

Parmi les nombreuses fonctions d'échelles qui ont été proposées, particulièrement dans le cadre des ondelettes [35] (le lien entre l'analyse multirésolution et les ondelettes est décrit dans [80] ou [67]), Musse a retenu les fonctions $\beta-$splines ϕ_α de degré α. Celles-ci sont définies récursivement comme suit :

$$\phi_0(x) = \begin{cases} 1 \text{ si } x \in [0;1] \\ 0 \text{ sinon} \end{cases} \tag{6.21}$$

$$\phi_\alpha(x) = \phi_{\alpha-1}(x) * \phi_0(x) \text{ pour } \alpha > 0 \tag{6.22}$$

Ces fonctions sont à support compact et continûment dérivables jusqu'à l'ordre α.

6.2.4 Optimisation hiérarchique

Le recalage déformable de deux images consiste à estimer le champ de déformation d minimisant la fonction de coût (6.4). Le champ de déformation d étant inconnu, on s'appuie sur le modèle décrit ci-dessus pour estimer d suivant une approche ascendante, i.e. partant du niveau de résolution l_i on parcourt la suite de sous-espaces vectoriels $V_{l_i} \subset \ldots \subset V_l \subset \ldots \subset V_{l_f}$ jusqu'à atteindre la résolution la plus fine (niveau l_f). Pour cela les variables du modèle de déformation, c'est-à-dire les matrices de coefficients suivants :

$$A^l_x = \left(a^l_{x_{i,j,k}}\right)_{i,j,k\in\{0,\ldots,m_l-1\}}, A^l_y = \left(a^l_{y_{i,j,k}}\right)_{i,j,k\in\{0,\ldots,m_l-1\}} \text{ et } A^l_z = \left(a^l_{z_{i,j,k}}\right)_{i,j,k\in\{0,\ldots,m_l-1\}} \tag{6.23}$$

sont déterminées à chaque niveau de résolution l de façon à minimiser la fonction de coût $C(d^l)$. Le champ de vecteurs de déplacement est supposé initialement nul à la résolution de départ, soit $A_x^{l_i} = A_y^{l_i} = A_z^{l_i} = 0$. La transition d'un niveau de résolution à un autre se fait simplement en utilisant l'opérateur de reconstruction ad hoc (voir (6.16)). En résumé la procédure d'optimisation peut être décrite par l'algorithme de la figure 6.1.

$l \leftarrow l_i$ | *- Niveau de résolution initiale*
$A_x^l = A_y^l = A_z^l = 0$ | *- Initialisation des variables du modèle*
Répéter
| *- Détermination des variables minimisant la fonction de coût C*
$\left(\widetilde{A_x^l}, \widetilde{A_y^l}, \widetilde{A_z^l}\right) =$ **arg** min $\left[C\left(d^l\right)\right]$
$l \leftarrow l + 1$
| *- Transition entre le niveau $l-1$ et le niveau l*
$A_x^l = \left(\uparrow_2 \left(\widetilde{A_x^{(l-1)}}\right)\right) * H, A_y^l = \left(\uparrow_2 \left(\widetilde{A_y^{(l-1)}}\right)\right) * H$ **et** $A_z^l = \left(\uparrow_2 \left(\widetilde{A_z^{(l-1)}}\right)\right) * H$
Jusqu'à $l > l_f$

FIG. 6.1 – Recalage déformable par optimisation hiérarchique.

Dans le cas de la fonction d'échelle ϕ_0, la base que l'on obtient est la base de Haar. Dans celle-ci le champ de vecteurs de déplacement d est optimisé hiérarchiquement en le contraignant à être constant sur des zones dont la taille décroît inversement avec la résolution (illustration de la figure 6.2). Ceci n'est cependant plus vrai dans le cas d'une $\beta-$spline de degré supérieur, car la contrainte est bien plus complexe. Comme le montre la figure 6.3 pour ϕ_1, les fonctions de base ont notamment des supports qui ne sont plus disjoints.

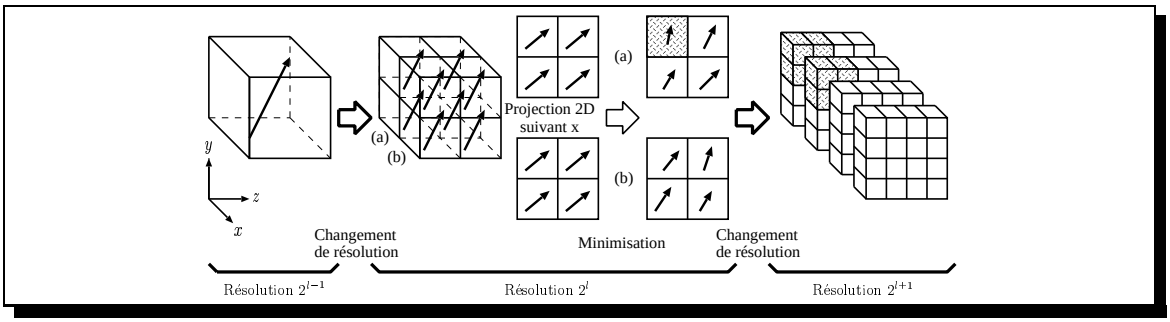


FIG. 6.2 – Optimisation hiérarchique du champ de déformation en 3D pour la base de fonctions induite par ϕ_0 (base de Haar).

6.2.5 Nombre de variables à optimiser

Voyons maintenant le nombre de variables à optimiser induit par le modèle de déformation décrit précédemment. Cela permettra au lecteur de se faire une idée de la difficulté du problème d'optimisation sous-jacent.

Considérons une fonction d'échelle ϕ de support $[0; sup_\phi[$. Dans ces conditions, au niveau de résolution l, chacune des fonctions de base $\phi^l_{i,j,k}$ de Φ^l, où $i,j,k \in \{0,...,m_l-1\}$ (équation (6.19)) a pour support :

$$\Omega_{\phi^l_{i,j,k}} = \left[\frac{i}{2^l}; \frac{i+sup_\phi}{2^l}\right[\times \left[\frac{j}{2^l}; \frac{j+sup_\phi}{2^l}\right[\times \left[\frac{k}{2^l}; \frac{k+sup_\phi}{2^l}\right[\tag{6.24}$$

Comme le support des images est supposé être $\Omega = [0;1] \times [0;1] \times [0;1]$, on déduit aisément les valeurs possibles pour les différents indices :

$$i,j,k \in \{0,...,m_l - 1 = 2^l - sup_\phi\} \tag{6.25}$$

Par conséquent les matrices A^l_x, A^l_y et A^l_z ont pour dimension $\left(m_l = 2^l - sup_\phi + 1\right)^3$ et le nombre total de variables devant être optimisées est (en 3D) :

$$var_\phi = \dim A^l_x + \dim A^l_y + \dim A^l_z = 3\left(2^l - sup_\phi + 1\right)^3 \tag{6.26}$$

Pour une dimension D quelconque, l'équation précédente devient $var_\phi = D\left(2^l - sup_\phi + 1\right)^D$.

À partir de l'équation (6.26) on peut fixer les bornes inférieure l_i et supérieure l_f des niveaux de résolution admissibles :

$$var_\phi > 0 \quad \Rightarrow \quad l_i \geq \ln_2 sup_\phi \tag{6.27}$$

$$var_\phi < N \quad \Rightarrow \quad l_f \leq \ln_2 \left(\left(\frac{N}{3}\right)^{\frac{1}{3}} + sup_\phi - 1\right) \tag{6.28}$$

où N est le nombre de voxels d'une image.

Pour les expérimentations que nous présentons dans la troisième partie, nous avons choisi pour fonction d'échelle la $\beta-$spline ϕ_1, pour laquelle $sup_\phi = 2$. La figure 6.3 montre les bases de fonction que ϕ_1 induit dans le cas monodimensionnel aux niveaux de résolution 1 et 2. Ainsi, suivant la taille des IRM 3D considérées on a d'après (6.27) et (6.28) les bornes suivantes pour le niveau de résolution :

$$N = (128)^3 \text{ voxels} \quad \Rightarrow \quad l_i = 1 \leq l \leq l_f = 6$$
$$N = (256)^3 \text{ voxels} \quad \Rightarrow \quad l_i = 1 \leq l \leq l_f = 7$$

Cependant, dans les IRM tous les voxels ne portent pas une information pertinente. En effet, une bonne partie des voxels correspond au fond. C'est pourquoi on peut réduire le nombre de variables à optimiser en supprimant toute fonction de base $\phi^l_{i,j,k}$ dont le support est une zone des images ne contenant pas d'information, on note $var_{\phi,r}$ le nombre de variables réduit. Cette réduction est opérée préalablement au traitement de chaque niveau de résolution, puisque l'image recalée obtenue à la résolution l détermine explicitement les fonctions de base qui doivent être conservées à la résolution $l+1$. Notons que pour les IRM que nous traitons une réduction n'est possible qu'à partir de $l = 3$. En définitive, le tableau suivant donne le nombre de paramètres à optimiser suivant la résolution pour des IRM de 128^3 voxels (attention : $var_{\phi,r}$ varie suivant le couple d'images que l'on recale entre elles).

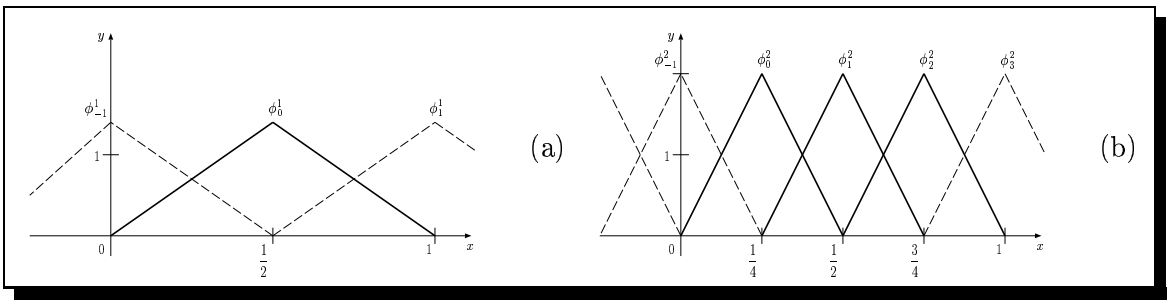


FIG. 6.3 – Bases de fonctions ϕ_1 dans le cas 1D pour les niveaux de résolution 1 (a) et 2 (b). Les fonctions en trait discontinu ne sont pas prises en compte car elles sont en dehors de $[0;1]$.

l	2^l	$\dim A_x^l$	var_ϕ	$var_{\phi,r}$	l	2^l	$\dim A_x^l$	var_ϕ	$var_{\phi,r}$
1	2	$(1)^3 = 1$	3	3	4	16	$(15)^3 = 3375$	10125	6057
2	4	$(3)^3 = 27$	81	81	5	32	$(31)^3 = 29791$	89373	38121
3	8	$(7)^3 = 343$	1029	960	6	64	$(63)^3 = 250047$	750141	263619

TAB. 6.1 – Nombre de variables suivant le niveau de résolution l pour la fonction d'échelle ϕ_1.

Dans sa thèse, Musse [87] optimise les matrices A_x^l, A_y^l et A_z^l au moyen d'approches déterministes. Le processus d'optimisation est mené en deux étapes : pour les résolutions 1 et 2, il utilise une méthode de quasi-Newton. Ensuite, il optimise localement chaque triplet de variables $\left(a^l_{x_{i,j,k}}, a^l_{y_{i,j,k}}, a^l_{z_{i,j,k}}\right)$, en parcourant les indices i, j et k de façon à ne pas traiter successivement des triplets associés à des fonctions d'échelle $\phi^l_{i,j,k}$ dont les supports ne sont pas disjoints (ce parcours peut être vu comme un coloriage de graphe).

La figure 6.4 tirée de [87], permet de se faire une idée des performances du modèle de déformation hiérarchique que nous venons de présenter. Cet exemple de recalage synthétique montre en particulier que le modèle est tout à fait capable d'appréhender de grandes déformations (plusieurs dizaines de voxels). Pour terminer, nous donnons dans le tableau 6.2 les temps de calcul moyens obtenus par Musse sur station de travail Hewlett-Packard 9000/C360 (360MHz) en fonction du niveau de la résolution jusqu'à laquelle est mené le recalage. Ces temps de calculs nous serviront d'éléments de comparaison dans la troisième partie.

Niveau l	1	2	3	4	5
Résolution 2^l	2	4	8	16	32
Temps en secondes	2	18	98	450	1322

TAB. 6.2 – Temps de calcul moyens obtenus par Musse [87] pour le recalage de deux IRM 128^3 voxels pour différentes résolutions finales.

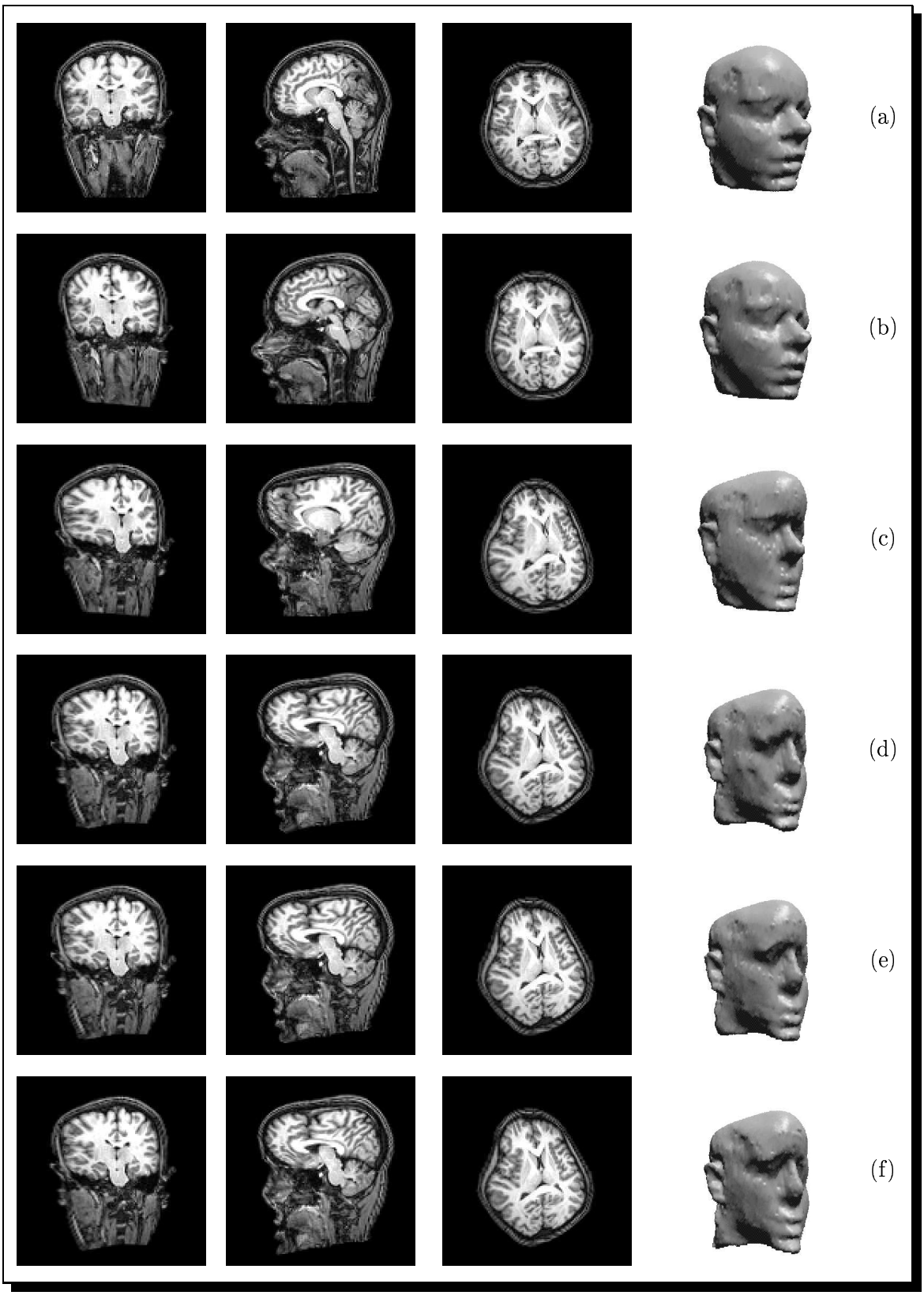


Fig. 6.4 – Recalage déformable hiérarchique d'une image source (a) sur une image synthétique (f) (obtenue par déformation synthétique de (a)). Résultat après recalage affine (b), après recalage non rigide au niveau de résolution 2 (c), 4 (d) et 5 (e).

6.3 Complexité du processus d'optimisation

La difficulté intrinsèque du processus d'optimisation induit par chacun des deux problèmes de recalage décrit dans les sections précédentes est fonction de deux facteurs :

① le nombre de variables à optimiser (les degrés de liberté de la transformation recherchée) ;

② le paysage de la fonction de coût mesurant la similarité de l'image recalée avec l'image de référence.

Dans le cas du recalage rigide, la transformation est définie par un nombre fixe de variables qui est de six (translation plus rotation en 3D). À l'opposé, la méthode de recalage non rigide met en jeu un nombre croissant de variables. Celui-ci croît d'ailleurs très rapidement (suivant la localisation des déformations recherchées : initialement globales, puis de plus en plus locales) pour atteindre plusieurs centaines de milliers de variables. Il faut également noter l'influence de la taille des données, car comme nous l'avons vu, elle conditionne le niveau de résolution final. Aussi, si on passe d'une paire d'images 128^3 voxels à des images 256^3 voxels, le nombre de variables à optimiser à la résolution la plus fine est également multiplié par un facteur huit. En pratique, cela signifie que l'on passe d'un peu plus de 750000 variables à plus de six millions (à comparer avec les 9,8 millions de variables du modèle fluide pour des images $128 \times 128 \times 100$ voxels, qui requiert presque deux heures de calcul en parallèle).

Cependant, le facteur le plus important à prendre en compte dans un processus d'optimisation est la fonction de coût, et plus précisément ses caractéristiques. En effet, hormis le nombre de variables de la fonction de coût, la complexité de l'optimisation est liée aux caractéristiques suivantes : convexité, séparabilité ($C(x,y) = C(x) \cdot C(y)$), symétrie ($C(x,y) = C(y,x)$), linéarité, dérivabilité de la fonction de coût. Par exemple, si une fonction est séparable, chacune de ses variables peut être optimisée indépendamment des autres. De même, l'inexistence de dérivées partielles rendra caduque l'utilisation de toute méthode s'appuyant sur celles-ci pour guider la recherche.

Les fonctions de similarité que nous avons considérées sont dérivables (bien que très difficilement dans le cas de l'information mutuelle), en revanche elles sont non séparables, non symétriques, fortement non linéaires et présentent de nombreux minima locaux. C'est pourquoi, si on ne considère pas d'approche hiérarchique, le recalage doit s'appuyer nécessairement sur un algorithme d'optimisation globale, soit totalement, soit partiellement comme phase d'initialisation d'un algorithme d'optimisation locale.

Conclusion

Le recalage d'images est une opération de base dans de nombreux domaines, notamment en imagerie médicale où les applications sont très nombreuses. Comme nous l'avons souligné dans le chapitre consacré à l'état de l'art, une approche fréquemment utilisée consiste à définir le recalage d'images comme un problème d'optimisation, i.e. minimiser une fonction de coût, dite fonction de similarité, évaluant qualitativement la « ressemblance » entre l'image de référence et l'image recalée. Nous avons décidé de nous intéresser plus particulièrement au recalage rigide et au recalage déformable (ou non rigide), en portant notre attention dans les deux cas sur une approche dense (c'est-à-dire basée sur les niveaux de gris des images).

Dans le contexte particulier de l'imagerie médicale nous avons vu que la difficulté posée par le recalage est essentiellement fonction de deux facteurs : la nature des images et le nombre de degrés de liberté de la transformation recherchée pour effectuer le recalage. De fait le recalage d'images de même nature est bien plus aisé que la mise en correspondance d'images multimodales mettant en évidence des informations différentes. De même, déterminer une transformation globale définie par moins d'une dizaine de variables (recalage rigide) est clairement plus facile que de trouver une transformation locale paramétrée par plusieurs milliers voire centaines de milliers de variables (recalage non rigide).

Plus précisément le recalage rigide consiste en une combinaison d'une rotation et d'une translation (3D). Pour mesurer la « ressemblance » de l'image recalée avec l'image de référence (ou cible), nous avons choisi deux fonctions de similarité : la fonction de similarité quadratique et l'information mutuelle. La première est bien adaptée au recalage d'images monomodales, alors que l'information mutuelle est supposée donner les meilleurs résultats en recalage multimodal (même si nous ne considérons dans nos expérimentations que des images de même modalité).

Le schéma de recalage non rigide considéré est celui qui a été défini par Musse dans sa thèse [87]. L'originalité de cette approche est qu'elle repose sur une analyse multirésolution du champ de vecteurs de déplacement à estimer. Ainsi le champ de vecteurs de déplacement n'est pas estimé à résolution constante, mais à des résolutions croissantes consistant à raffiner le champ de vecteurs de façon à progressivement prendre en compte des déformations de plus en plus locales. Il est à noter que c'est le champ de déformation qui évolue et non la résolution des images, celles-ci sont considérées dans leur globalité tout au long du processus de recalage. L'analyse multirésolution est définie par l'intermédiaire d'une fonction d'échelle, aussi le champ de vecteurs de déplacement et donc la transformation correspondante, héritent de ses propriétés de continuité et différentiabilité. Comme la plupart des techniques de recalage en imagerie médicale, l'estimation du champ de vecteurs de déplacement s'exprime sous la forme de la minimisation d'une fonction de coût, en l'occurrence l'erreur quadratique moyenne (le recalage non rigide est en effet quasiment cantonné

au recalage monomodal). De plus cette approche paramétrique hiérarchique permet de garantir une décroissance continue du coût tout en réduisant la sensibilité aux minima locaux.

Finalement, un point commun à la majorité des méthodes de recalage s'exprimant sous la forme d'un problème d'optimisation, est qu'elles s'appuient sur une méthode d'optimisation locale. En effet, bien que les fonctions de similarité soient non linéaires et comportent quantité de minima locaux, l'utilisation d'une stratégie multirésolution au niveau du parcours des images pour le calcul du coût ou d'une approche multiéchelle des déformations permet de lisser la fonction de coût. En fait, l'utilisation d'une technique d'optimisation globale est essentiellement vue comme une étape préalable nécessaire pour bien initialiser une méthode locale. Notre travail apporte donc une contribution à ce niveau, puisque comme nous le montrons dans la partie qui suit, certaines méthode globales sont bien adaptées au recalage d'images médicales, ne requérant aucun raffinement ultérieur de la solution par une méthode d'optimisation locale.

Le recours au parallélisme s'impose essentiellement pour deux raisons. D'une part il permet d'obtenir des algorithmes de recalage compétitifs, d'autre part il assure leur pérennité dans le futur. En effet, une méthode de recalage est d'autant plus utilisée en routine clinique que son temps de calcul est réduit. De plus, la tendance actuelle est à l'augmentation de la résolution des images, et par voie de conséquence à un accroissement du volume de données à traiter. Ainsi, en faisant passer les images de 128^3 voxels à 256^3 voxels, le temps de calcul des algorithmes de recalage séquentiels est *a priori* au minimum multiplié par huit. D'ailleurs si on regarde les temps de calcul moyens en séquentiel donnés par Musse (voir le tableau 6.2) on voit clairement l'apport potentiel du parallélisme. En effet, pour des IRM 128^3 voxels, le recalage jusqu'au cinquième niveau de résolution requiert un peu plus de 20 minutes (sur une station de travail HP 9000/C360), soit en extrapolant, presque trois heures pour des IRM 256^3 voxels. Enfin, comme on l'a vu, une taille de 256^3 voxels permettrait le traitement d'un niveau de résolution supplémentaire. Or celui-ci nécessiterait l'optimisation de plusieurs millions de variables, ce qui n'est pas envisageable actuellement en séquentiel.

Troisième partie

Étude de l'adéquation d'algorithmes d'optimisation globale à la problématique du recalage et à leur mise en œuvre data-parallèle

Introduction

L'état de l'art sur les algorithmes d'optimisation combinatoire et leurs parallélisations, présenté dans la première partie, pose clairement la question du choix d'un algorithme et d'une parallélisation dans le cadre d'un problème particulier. De fait, lorsque l'on est confronté à un problème d'optimisation, il est difficile de dire *a priori* que tel algorithme, avec telle parallélisation, sera meilleur que tel autre. C'est pourquoi l'étude de plusieurs algorithmes d'optimisation globale s'impose.

L'objet de cette partie est d'étudier cette problématique dans le contexte du recalage en imagerie médicale. Plus précisément, nous considérons d'une part le recalage rigide et d'autre part le recalage déformable (tous deux en 3D et denses). Le lecteur a d'ailleurs pu se faire une idée de la difficulté posée par ces deux problèmes dans la partie précédente.

Nous menons cette étude en deux temps.

D'abord nous mettons en œuvre en parallèle plusieurs algorithmes d'optimisation globale dans le cas du problème de recalage rigide. Après avoir préalablement déterminé dans le cadre séquentiel, pour chaque algorithme retenu :

- son adéquation à résoudre le problème d'optimisation considéré ;
- le modèle de programmation parallèle qui est le mieux adapté.

Chaque algorithme séquentiel et parallèle, est étudié d'une part sur le plan de ses performances, i.e. la qualité du recalage induit et les temps de calcul[1], et d'autre part au niveau de l'impact de ses paramètres afin d'estimer sa robustesse (la sensibilité au choix des paramètres). En effet, un algorithme d'optimisation robuste est bien plus intéressant qu'un algorithme nécessitant un ajustement fastidieux afin de trouver le bon jeu de paramètres.

Nous tirons ensuite parti des résultats obtenus dans le cas rigide, pour poursuivre l'étude dans le cas déformable sur un champ d'investigation plus réduit. De fait, il est peu probable qu'un algorithme d'optimisation, parallèle ou non, qui ne donne pas de performances satisfaisantes en rigide, soit plus performant pour le recalage déformable.

Rappelons également que nous avons retenu les algorithmes suivants (voir la première partie) : l'équation de la diffusion et le recuit simulé adaptatif (deux algorithmes du type recuit simulé) ; les stratégies d'évolution et l'évolution différentielle (de la famille des algorithmes évolutionnaires) ; enfin le *Genetic Simulated Annealing*, un hybride parallèle recuit simulé/algorithme génétique.

Pour terminer, il est à noter que les expérimentations étant réalisées sur une machine parallèle MIMD, la mise en œuvre des algorithmes suivant une approche data-parallèle a également pour objectif d'apprécier les performances de ce modèle de programmation lorsqu'il n'est pas associé

[1] Bien entendu, ce sont des temps de calcul moyens que nous donnons.

avec le modèle d'exécution adéquat (une machine SIMD). En clair, on va tester un modèle de programmation, plutôt qu'un modèle d'exécution.

Trois chapitres composent cette partie : le premier est consacré à la présentation du cadre expérimental. Après un rappel sur la programmation parallèle et les mesures de performances, divers aspects liés à l'implantation sont présentés. Le chapitre qui suit concerne la mise en œuvre parallèle des différents algorithmes d'optimisation choisis et leur étude dans le cadre du problème du recalage rigide. En écho aux résultats obtenus pour le cas rigide, dans le dernier chapitre, qui traite le recalage déformable, nous focalisons notre étude sur deux algorithmes : l'évolution différentielle et l'équation de la diffusion.

Chapitre 7

Présentation du cadre expérimental

Dans ce chapitre nous présentons les différents modèles de programmation parallèle qui peuvent être envisagés pour paralléliser les algorithmes retenus, les critères de performance des algorithmes parallèles, ainsi que les conditions expérimentales. Enfin les données utilisées pour valider les algorithmes dans l'application de recalage d'images médicales sont précisées, de même que les éventuels pré-traitements.

7.1 Programmation parallèle et mesures de performances

En matière de programmation parallèle, deux modèles de programmation se sont progressivement dégagés : le parallélisme de contrôle et le parallélisme de données ou data-parallélisme [93]. Le parallélisme de contrôle était, à l'origine défini comme le modèle de programmation associé aux machines parallèles MIMD (*Multiple Instruction Multiple Data*), alors que le modèle de programmation dominant pour les machines SIMD (*Single Instruction Multiple Data*) était le parallélisme de données. Cette adéquation modèle de programmation/modèle d'exécution résultait en grande partie de l'architecture des machines parallèles. Cependant, au cours du temps, ce lien s'est réduit jusqu'à disparaître plus ou moins et ces deux modèles de programmation sont devenus des paradigmes en eux-mêmes. On a vu ainsi se développer des machines MIMD permettant également de faire du data-parallélisme, ce qui a peut-être contribué à la prépondérance actuelle de ce type de machines parallèles, outre leurs qualités et potentialités intrinsèques.

7.1.1 Le parallélisme de contrôle

Dans ce modèle de programmation, on définit un programme comme un ensemble de processus séquentiels exécutés concomitamment. Ceux-ci peuvent communiquer par l'intermédiaire de messages ou de variables partagées. Un modèle théorique de cette approche est défini notamment à travers les *Communicating Sequential Processes* (CSP) de Hoare [58] et les nombreuses études qui leur sont associées. Le gain que l'on peut espérer grâce à ce modèle de programmation est naturellement fonction de la taille des blocs d'instructions séquentielles pouvant être exécutés indépendamment les uns des autres, et des performances et de la densité des communications ou des accès partagés à la mémoire. On parle volontiers, alors, de parallélisme « à gros grain ».

7.1.2 Le parallélisme de données

L'idée sous-jacente au data-parallélisme est de voir la programmation parallèle comme une mise en œuvre séquentielle de traitements simultanés sur des données à accès parallèle. En pratique, il y a un seul *thread* contrôlant le flot d'instructions et chaque instruction est exécutée en parallèle sur un ensemble de données distribuées sur des processeurs virtuels organisés suivant une topologie fixée. Ce modèle de programmation est donc particulièrement adapté aux problèmes réguliers et traitant un imposant volume de données. Un point délicat dans le data-parallélisme est la distribution des données : le nombre de processeurs physiques est, en effet, généralement inférieur au nombre de processeurs virtuels. Par conséquent le programmeur ou le compilateur doivent préciser l'allocation réelle des données et il est important de bien définir cette distribution afin de réduire les communications autant que faire se peut. La programmation data-parallèle peut être facilitée par le fait que cette distribution des données et les communications inter-processeurs peuvent être engendrées directement par le compilateur utilisé. Le compilateur est alors l'artisan essentiel du passage du modèle de programmation au modèle d'exécution, en tenant compte le mieux possible du schéma de communication induit par l'algorithme et de l'architecture de la machine cible. Toutefois, les langages de programmation data-parallèle permettent généralement à l'utilisateur de préciser le mode de distribution désiré au moyen de directives. Dans ce mode de programmation, le gain que l'on peut espérer est, évidemment, fonction du nombre de données pouvant être traitées simultanément et des contraintes imposées par une synchronisation à chaque instruction. On parle dans ce cas de parallélisme « à grain fin ».

7.1.3 Mesure des performances des programmes parallèles

Pour évaluer un algorithme parallèle plusieurs critères de mesures ont été définis [26]. Dans le travail que nous présentons, l'étude des performances des différents algorithmes parallèles s'appuiera sur les indicateurs suivants :

① *Accélération*
L'accélération correspond au rapport du temps de calcul d'une solution avec le programme séquentiel le plus rapide possible sur le temps de calcul de l'algorithme parallèle s'exécutant sur plusieurs processeurs, supposés tous identiques.

② *Accélération relative*
L'accélération relative est définie comme le rapport du temps de calcul d'une solution avec le programme parallèle s'exécutant sur un processeur sur le temps de calcul du même algorithme mais exécuté sur plusieurs processeurs.

③ *Efficacité*
Cette mesure est égale au rapport de l'accélération relative sur le nombre de processeurs.

Nous rappelons également la loi d'Amdahl qui donne une borne supérieure sur l'accélération relative que l'on peut obtenir avec P processeurs :

$$\textit{Accélération relative} \leq \frac{1}{s + p/P} \tag{7.1}$$

où s est la fraction du temps d'exécution qui demeure obligatoirement séquentielle, alors que p est le dual, i.e. la fraction du temps d'exécution du programme qui peut s'exécuter en parallèle ($s + p = 1$).

7.2 La machine parallèle SGI Origin2000

Les différents algorithmes séquentiels et parallèles, mis en œuvre dans cette étude, ont été implantés sur l'Origin2000 de l'Université Louis Pasteur. Ce calculateur parallèle comporte un total de 52 processeurs divisé en 32 processeurs MIPS R10000 à 195MHz et 20 processeurs MIPS R12000 à 300MHz. La mémoire est, quand à elle, de 20Go et l'espace disque de 336Go. L'Origin2000 est une machine parallèle MIMD à mémoire partagée, physiquement distribuée, avec un hypercube pour réseau d'interconnexion (voir la figure 7.1 pour la machine de l'ULP). Puisque la mémoire est « virtuellement » partagée, les communications réelles, induites par des opérations telles que la diffusion d'une donnée, voire des communications virtuellement régulières et « locales », pourront se révéler irrégulières. Par ailleurs, l'architecture CC-NUMA (*Cache Coherent - Non-Uniform Access Memory*) de l'Origin2000 fait que le temps d'accès à une donnée dépend de sa position dans l'espace mémoire global. Le placement des données n'est donc pas à négliger, même si en pratique l'usage de bibliothèques de communication optimisées pour cette architecture peut prévenir de performances trop médiocres.

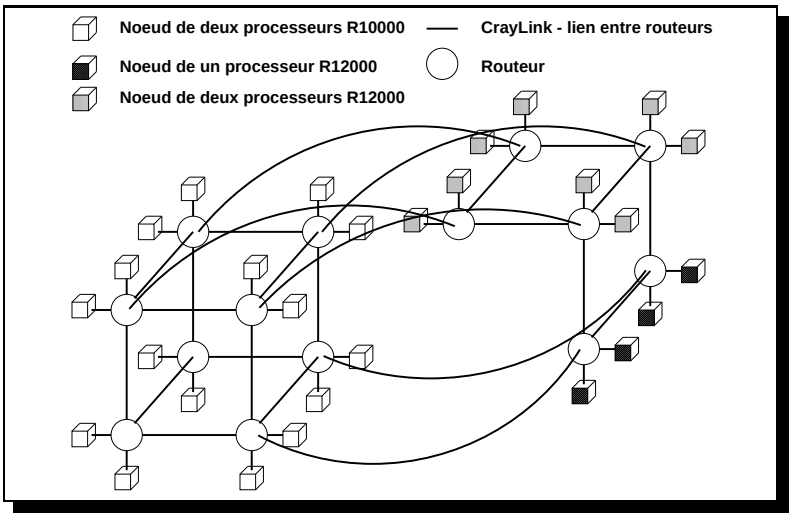


Fig. 7.1 – Topologie du calculateur SGI Origin2000 de l'Université Louis Pasteur.

7.2.1 Langages de programmation

Pour implanter les algorithmes étudiés nous avons suivi l'approche suivante : le chargement des données (images) et les algorithmes d'optimisation ont été écrits en Fortran 90 ; la partie concernant le recalage, i.e. le calcul des images recalées et de la fonction de coût sont eux écrits en C. L'interfaçage entre les deux langages n'a pas posé de problème particulier, si ce n'est que lorsque l'on échange des tableaux, l'ordre des indices de boucles est inversé (la case (i, j, k) d'un tableau en Fortran a pour position (k, j, i) en C). Pour ce qui est des parallélisations, celles-ci ont été réalisées au moyen de directives *High Performance Fortran* [70] (pour le recalage rigide) et *OpenMP* [34] (pour le recalage déformable). Il nous semblait en effet intéressant de voir si des paradigmes de programmation parallèle dans lesquels le compilateur produit le détail des communications, permettent de traiter efficacement le passage du modèle de programmation au modèle d'exécution.

✐ High Performance Fortran (HPF)

Dans HPF [111], les objets « parallèles » sont des tableaux dont on peut spécifier le placement au moyen de directives précédées par `!HPF$` :

- la directive `DISTRIBUTE` définit complètement la distribution des données ;
- la directive `ALIGN` permet d'indiquer l'alignement relatif entre plusieurs tableaux ;
- les deux directives précédentes qui sont statiques, i.e. que le placement est fixe, ont également des équivalents dynamiques (`REDISTRIBUTE` et `REALIGN`). Elles sont utilisables à la condition que les tableaux soient déclarés comme dynamiques.

Le placement doit de préférence suivre un schéma régulier. Outre le placement sur des processeurs virtuels (l'adéquation processeur physique/processeur virtuel dépend de leur nombre), de nombreuses opérations data-parallèles permettent de manipuler les tableaux :

① Expressions et assignations sur un tableau.

② Manipulations intrinsèques du Fortran 90 sur les tableaux.

③ Instruction `FORALL`.

④ Assertion `INDEPENDENT` pour les instructions `DO` et `FORALL`.

⑤ Librairie HPF de fonctions sur les tableaux.

✐ OpenMP

La philosophie de OpenMP est différente de celle de HPF puisqu'il n'y a pas de distribution de données. En effet, le parallélisme repose plus sur un partage du « travail » sur un espace mémoire supposé global. OpenMP étend le langage de programmation séquentiel de façon à obtenir un langage de programmation SPMD (*Single Program Multiple Data*). Pour cela, on a recours à des directives (sentinelle `!$OMP`) permettant :

- le partage du travail (région parallèle définit par les directives `PARALLEL` et `END PARALLEL`, structure de contrôle `DO - END DO`, etc) ;
- le partage ou la privatisation des données (attributs `SHARED` et `PRIVATE`) ;
- la synchronisation (barrière, section critique, etc).

OpenMP est donc *a priori* particulièrement bien adapté aux machines parallèles à mémoire partagée et en l'occurrence à l'Origin2000 qui est le calculateur servant de cadre d'expérimentation.

7.3 Validation du recalage - données utilisées

Pour évaluer la qualité du recalage rigide nous avons appliqué chaque algorithme sur un jeu de 20 recalages 3D IRM/IRM. Les vingt images à recaler ont été obtenues en appliquant des transformations rigides engendrées aléatoirement à une image IRM 3D (128^3 voxels) de référence provenant de l'Institut de Physique Biologique des Hôpitaux Universitaires de Strasbourg (image 9.5(i), mais restreinte au cerveau). Les différentes transformations ont des valeurs de translation comprises entre -20 et $+20$ voxels et des angles de rotation entre -20 et $+20$ degrés.

Dans le cas non rigide nous utilisons des images IRM provenant de plusieurs patients (images 9.5(a) et 9.5(i)). Elles sont préalablement normalisées en intensité car les conditions d'acquisition ne sont pas strictement identiques. En effet, les voxels représentant la même information dans chacune des images n'ont pas la même intensité et la fonction de coût (6.4) ne prend en compte que des variations liées à un bruit gaussien.

Chapitre 8

Cas du recalage rigide

8.1 Introduction

Tout d'abord, il est à noter que nous avons décidé d'inscrire le processus d'optimisation dans une approche hiérarchique des données, dès lors que l'algorithme d'optimisation ne guide la recherche qu'au seul moyen du coût associé à une solution potentielle. Ainsi, à l'exception de l'équation de la diffusion, tous les algorithmes d'optimisation considérés sont appliqués sur une séquence de grilles 3D de résolution croissante. En pratique, cela consiste à calculer les fonctions de coût (5.1) et (5.5) sur un sous-échantillonnage croissant des voxels des images 3D ($128 \times 128 \times 128$ voxels) : initialement on parcourt les images en ne considérant qu'un voxel sur 81, puis 1 sur $27, 9, 3$ et finalement tous les voxels. L'introduction de ce sous-échantillonnage se traduit par une remontée du coût lors du changement de résolution. Cette remontée est particulièrement importante dans le cas de la fonction de similarité quadratique. Aussi, pour avoir une décroissance relativement continue, on « normalise » la fonction de coût en la divisant par le nombre de voxels du sous-échantillonnage. La fonction de coût considérée devient alors l'erreur quadratique moyenne (à noter que les coûts présentés correspondent en fait à la racine carrée de cette dernière).

Cette approche multirésolution permet de réduire la complexité calculatoire, mais surtout, les algorithmes multirésolutions sont beaucoup moins sensibles aux minima locaux des fonctions de coût qu'un algorithme monorésolution [56]. En fait, il a été conjecturé [56] que le traitement multirésolution régularise les fonctions de coût, ce qui accélère la convergence vers les bonnes solutions. D'autre part, nous accentuons encore la convergence en centrant l'espace de recherche des variables de la translation sur le vecteur de translation résultant du recalage des barycentres des deux images 3D que l'on veut recaler entre elles. Nous avons fait ce choix plutôt qu'un pré-centrage de tous les paramètres par recalage primaire sur les axes principaux, afin de montrer que les algorithmes convergent sans aucune connaissance particulière sur les angles de rotation (mis à part les bornes de l'intervalle de recherche).

Un aspect qui est également important est le choix de la méthode d'interpolation utilisée pour calculer l'image recalée. En effet, la précision du calcul induit par une méthode d'interpolation a un impact direct sur le coût et donc la qualité du recalage. Parmi les techniques d'interpolation habituellement considérées on trouve l'interpolation aux plus proches voisins, l'interpolation

trilinéaire ou l'interpolation par sinus-cardinal. Naturellement la précision d'une technique d'interpolation se traduit par un coût calculatoire qui croît proportionnellement. Un bon compromis précision/temps de calcul est obtenu avec l'interpolation trilinéaire, d'où son choix.

Afin d'évaluer précisément la qualité du recalage produit par les différents algorithmes, nous avons comparé les paramètres estimés par l'algorithme aux vrais paramètres de recalage (transformations inverses des transformations rigides utilisées pour créer le jeu d'images à recaler, voir section 7.3). Pour cela la moyenne et l'écart type de l'erreur de recalage ont été calculés sur les 20 recalages. La précision sous-voxel éventuelle du recalage est établie par l'intermédiaire de la mesure de l'écart quadratique moyen (EQM) entre chaque image recalée et la solution optimale :

$$EQM = \sqrt{\frac{1}{N}\sum_{s=1}^{N} \|p_s - \hat{p}_s\|^2} \tag{8.1}$$

où N est le nombre de voxels de l'échantillonnage ; p_s est la position estimée d'un voxel et $\hat{p}_s$ est sa position optimale.

8.2 Équation de la diffusion

Pour mettre en œuvre l'algorithme de l'équation de la diffusion décrit à la figure 1.2 (section 1.3.2), il faut d'une part pouvoir calculer le gradient de l'énergie et d'autre part préciser divers paramètres :

- la température initiale T_0 et le pas de temps initial ΔP_0 ;
- les coefficients α_T et α_P qui assurent la décroissance respective de la température T_t et du pas de temps ΔP_t ($t \in \mathbb{R}^+$ est le temps) ;
- le critère d'arrêt.

Dans le problème du recalage rigide qui nous intéresse, les configurations sont de la forme $X_t = (t_x(t), t_y(t), t_z(t), \theta_x(t), \theta_y(t), \theta_z(t))$ et la fonction d'énergie (E) est remplacée par la fonction de coût (C) choisie, en l'occurrence l'erreur quadratique moyenne (dérivée de 5.1) :

$$C\left(I\left(.\right), J\left(T_\Theta\left(.\right)\right)\right) = \frac{1}{|\Omega|}\sum_{s\in\Omega}\left[I\left(s\right) - J\left(T_\Theta\left(s\right)\right)\right]^2 \tag{8.2}$$

où $s = (x, y, z)^T$ est la position d'un voxel et Ω est l'ensemble des voxels de l'IRM. En effet, bien que l'équation de la diffusion ne sous-échantillonne pas les images, l'erreur quadratique moyenne permet de réduire les valeurs numériques du gradient sans aucune influence sur le processus d'optimisation. L'information mutuelle ne sera pas traitée car le calcul analytique de sa dérivée est difficile, ce qui rend d'autant plus complexe le calcul du gradient.

8.2.1 Gradient de l'erreur quadratique moyenne

Seul le calcul de la composante du gradient correspondant à θ_x est décrit (voir section 6.1), les autres composantes sont données directement.

Commençons par calculer la dérivée partielle par rapport à θ_x :

$$\frac{\partial C\left(I\left(.\right), J\left(T_\Theta(.)\right)\right)}{\partial \theta_x} = \frac{1}{|\Omega|}\sum_{s\in\Omega} 2\left(J\left(T_\Theta(s)\right) - I\left(s\right)\right)\frac{\partial\left(J\left(T_\Theta(s)\right)\right)}{\partial \theta_x} \tag{8.3}$$

Introduisons $s' = (x', y', z')^T = T_\Theta(s)$. On obtient :

$$\begin{aligned}\frac{\partial C\left(I\left(.\right), J\left(T_\Theta(.)\right)\right)}{\partial \theta_x} &= \frac{1}{|\Omega|}\sum_{s\in\Omega} 2(J(s') - I(s))\left(\left(\frac{\partial J(s')}{\partial x'}, \frac{\partial J(s')}{\partial y'}, \frac{\partial J(s')}{\partial z'}\right) \bullet \left(\frac{\partial x'}{\partial \theta_x}, \frac{\partial y'}{\partial \theta_x}, \frac{\partial z'}{\partial \theta_x}\right)^T\right) \\ &= \frac{1}{|\Omega|}\sum_{s\in\Omega} 2(J(s') - I(s))\left(\nabla_{s'} J(s') \bullet \left(\frac{\partial R}{\partial \theta_x} \times (s - g)\right)\right) \qquad (8.4)\end{aligned}$$

où $g = (x_g, y_g, z_g)^T$ est le centre de la transformation T_Θ.

Finalement en procédant de même pour les variables restantes, on obtient pour le gradient de l'énergie $(\nabla_\Theta C\left(I\left(.\right), J\left(T_\Theta(.)\right)\right))$, le vecteur suivant :

$$\left(\tfrac{1}{|\Omega|}\left(\textstyle\sum_{s\in\Omega} 2(J(s') - I(s))\left(\nabla_{s'} J(s') \bullet (1,0,0)^T\right)\right), \qquad (8.5)$$

$$\tfrac{1}{|\Omega|}\left(\textstyle\sum_{s\in\Omega} 2(J(s') - I(s))\left(\nabla_{s'} J(s') \bullet (0,1,0)^T\right)\right), \qquad (8.6)$$

$$\tfrac{1}{|\Omega|}\left(\textstyle\sum_{s\in\Omega} 2(J(s') - I(s))\left(\nabla_{s'} J(s') \bullet (0,0,1)^T\right)\right), \qquad (8.7)$$

$$\tfrac{1}{|\Omega|}\left(\textstyle\sum_{s\in\Omega} 2(J(s') - I(s))\left(\nabla_{s'} J(s') \bullet \left(\frac{\partial R}{\partial \theta_x} \times (s - g)\right)\right)\right), \qquad (8.8)$$

$$\tfrac{1}{|\Omega|}\left(\textstyle\sum_{s\in\Omega} 2(J(s') - I(s))\left(\nabla_{s'} J(s') \bullet \left(\frac{\partial R}{\partial \theta_y} \times (s - g)\right)\right)\right), \qquad (8.9)$$

$$\tfrac{1}{|\Omega|}\left(\textstyle\sum_{s\in\Omega} 2(J(s') - I(s))\left(\nabla_{s'} J(s') \bullet \left(\frac{\partial R}{\partial \theta_z} \times (s - g)\right)\right)\right)\right) \qquad (8.10)$$

8.2.2 Paramètres

Au niveau des paramètres nous avons fait les choix suivants :

- Le critère d'arrêt est un nombre maximum de configurations produites (Max_{conf}).
- En ce qui concerne le pas de temps ΔP_t, il est rapidement apparu qu'une valeur respectivement différente pour les paramètres de translation ($\Delta P_{t,T}$) et de rotation ($\Delta P_{t,R}$) s'imposait. En effet, pour produire la nouvelle configuration, on multiplie le gradient par le pas de temps (cf. algorithme 1.2). Or la valeur des dérivées partielles pour les paramètres de translation et de rotation ne sont pas du tout du même ordre (voir les figures 8.2(a) et 8.2(b) qui montrent l'évolution du gradient lors d'un recalage[1]). De fait, une translation de $0,1$ voxel suivant l'axe des x a une influence nettement plus réduite sur le coût qu'une rotation de $0,1$ radians autour de ce même axe.
  À l'issue de l'observation de la valeur initiale du gradient pour plusieurs recalages, nous avons décidé de procéder comme suit. Dans un premier temps, nous avons fixé deux pas de temps ΔP_T (translation) et ΔP_R (rotation) :

$$\Delta P_T = 10^{-5} \text{ et } \Delta P_R = 10^{-7} \qquad (8.11)$$

  Avec ces pas de temps on aurait pour une dérivée partielle $\frac{\partial C}{\partial t_x}$ égale à 100000, un déplacement $\delta_1(0)$ (définition au troisième paragraphe de la section 1.3.2) de un voxel, tandis que $\frac{\partial C}{\partial \theta_x} = -5000000$ induirait par $\delta_1(0)$ une rotation de $0,5$ radians (≈ 29 degrés). On

[1] Attention au changement d'échelle de l'ordonnée au niveau de la soixantième itération.

fixe ensuite les pas de temps utilisés pour engendrer la dernière configuration (itération Max_{conf}) à 10^{-7} et 10^{-9}, respectivement pour la translation et la rotation. Puis on calcul les coefficients assurant la décroissance des pas de temps et les pas initiaux :

$$\alpha_{P_T} = \left(\frac{10^{-7}}{\Delta P_T}\right)^{\frac{1}{Max_{conf}}} \Rightarrow \Delta P_{0,T} = \alpha_{P_T} \cdot \Delta P_T \tag{8.12}$$

$$\alpha_{P_R} = \left(\frac{10^{-9}}{\Delta P_R}\right)^{\frac{1}{Max_{conf}}} \Rightarrow \Delta P_{0,R} = \alpha_{P_R} \cdot \Delta P_R \tag{8.13}$$

Ces relations sont déduites directement de $\Delta P_{t+\Delta P_t} = \alpha_P \cdot \Delta P_t$ (section 1.3.2).

- Pour le schéma de température on suit la même approche que pour les pas de temps, c'est-à-dire que l'on fixe la valeur de la dernière température utilisée pour engendrer une configuration à 10^{-7}, puis, à partir de la température T précédant T_0 dans le schéma de température, on détermine α_T et T_0 :

$$\alpha_T = \left(\frac{10^{-7}}{T}\right)^{\frac{1}{Max_{conf}}} \Rightarrow T_0 = \alpha_T \cdot T \tag{8.14}$$

En conclusion, et comme nous l'avons fait remarqué lors de la présentation de l'équation de la diffusion dans la première partie, nous avons utilisé un schéma de décroissance exponentielle pour les pas et la température. Un dernier point à noter est que la distribution multinormale utilisée pour simuler le mouvement brownien n'a pas pour écart type le pas, mais un multiple de la température courante $(0,1 \cdot T_t)$. En effet, les pas étant très faibles, le mouvement brownien que l'on aurait obtenu en les utilisant se serait traduit par une perturbation quasi-nulle de la descente du gradient (terme $\delta_2(t)$). Les graphiques (c) et (d) de la figure 8.2 montrent la pertinence de notre approche sur un recalage particulier.

8.2.3 Performances et comportements de l'algorithme séquentiel

Convergence

Commençons par étudier l'impact de la température initiale et du nombre de configurations engendrées (Max_{conf}). Dans cette optique, nous avons échantillonné régulièrement des températures T à partir de 50. Les températures successives sont obtenues en divisant par deux la température précédente, ainsi après $T = 50$ on a $T = 25,\ 12,5$, etc. En ce qui concerne le nombre de configurations produites, nous avons choisi deux valeurs : $Max_{conf} = 200$ et $Max_{conf} = 100$. À partir des valeurs de T et de Max_{conf} on peut grâce aux équations (8.12), (8.13) et (8.14) déterminer les différents coefficients de réduction de la température et des pas de temps. Le tableau 8.1 résume les jeux de paramètres que l'on obtient suivant ce processus.

La figure 8.1 montre l'évolution moyenne du coût sur 20 recalages synthétiques pour les différents jeux de paramètres. On constate notamment que plus la température initiale T_0 est élevée, plus le nombre d'itérations nécessaires pour assurer une bonne convergence est important. En effet, il est évident au regard de la valeur du coût qu'une température élevée se traduit par une perturbation importante de la descente du gradient. L'algorithme effectue alors en quelque sorte de grands sauts dans l'espace de recherche. Or comme les pas de temps sont relativement

T	$Max_{conf} = 200$			$Max_{conf} = 100$		
	T_0	α_T	$\alpha_{P_T} = \alpha_{P_R}$	T_0	α_T	$\alpha_{P_T} = \alpha_{P_R}$
50	45,235	0,905				
25	22,696	0,908				
12,5	11,387	0,911				
6,25	5,713	0,914	0,977	5,223	0,836	
3,125	2,867	0,917		2,630	0,841	0,955
1,5625	1,438	0,921		1,324	0,847	

Tab. 8.1 – Jeux de paramètres pour étudier la convergence de l'équation de la diffusion.

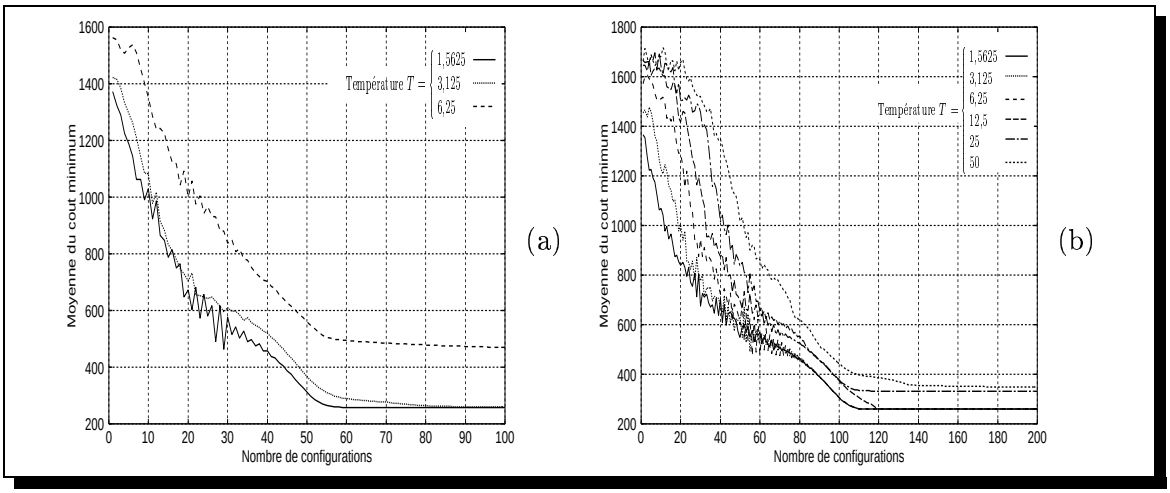


Fig. 8.1 – Évolution moyenne du coût minimum pour différentes températures, suivant le nombre de configurations engendrées : (a) $Max_{conf} = 100$ et (b) $Max_{conf} = 200$.

Δt_x	voxels	$0,19 \pm 0,15$	$\Delta\theta_x$	degrés	$0,001 \pm 0,001$
Δt_y		$0,14 \pm 0,10$	$\Delta\theta_y$		$0,001 \pm 0,001$
Δt_z		$0,16 \pm 0,09$	$\Delta\theta_z$		$0,002 \pm 0,002$

Tab. 8.2 – Moyennes et écarts types des erreurs de recalage par l'équation de la diffusion, dans le cas de la fonction de similarité quadratique, pour $T = 1,5625$ et $Max_{conf} = 100$.

faibles, le terme δ_2 décroît trop rapidement ce qui peut aboutir à un blocage dans un minimum local. En fait, les meilleurs résultats sont obtenus avec les températures les plus faibles, c'est-à-dire celles qui induisent une exploration à une échelle fine et un schéma de décroissance plus lent. Notons également que les meilleures courbes de convergence se confondent vers la fin, car à ce moment là, l'algorithme est dans un mode déterministe (seul le gradient guide la recherche).

Si on regarde plus précisément l'évolution des variables de X_t lors d'un recalage, présentée sur les graphiques 8.2(e) pour la translation et 8.2(f) pour la rotation (la configuration optimale est $X_t = (15, -7, -17, -11, 11, 3)$), on peut noter deux points :

- d'une part les paramètres de la translation sont rapidement trouvés. En effet, l'initialisation du vecteur de translation par recalage des barycentres des deux images 3D donne une solution très proche de la solution optimale ;
- d'autre part les angles de rotation évoluent énormément pendant le premier quart des 100 itérations, puis ayant apparemment trouvé le bassin d'attraction de l'optimum global, les oscillations décroissent lentement.

De plus, une soixantaine d'itérations semble suffisante pour obtenir une très bonne estimation de la configuration optimale. En effet, au-delà de l'itération 60 ce ne sont plus que des décimales des paramètres de translation et de rotation qui varient. En revanche comme le montre les graphiques 8.2(a) et 8.2(b), le gradient évolue lui tout au long des 100 itérations, se rapprochant de plus en plus du vecteur nul.

8.2.4 Qualité du recalage et temps d'exécution

Par définition l'algorithme de l'équation de la diffusion a deux modes de « fonctionnement » : initialement, c'est un algorithme stochastique, puis, à mesure que la perturbation aléatoire diminue, la descente du gradient prend le dessus et donne un comportement déterministe à l'algorithme. Il est donc naturel que deux exécutions de l'algorithme aboutissent au même résultat lorsque le bassin d'attraction final est identique (d'où d'ailleurs la confusion des courbes pour les températures $T = 1,5625$, $T = 3,125$ et $T = 6,25$ dans la figure 8.1(b). Cela signifie également que les erreurs de recalage sont identiques pour les différents jeux de paramètres qui permettent d'aboutir au bassin d'attraction du minimum global de chaque recalage, du moins si la bonne vallée n'est pas trouvée trop tardivement.

Le tableau 8.2 donne la moyenne et l'écart type de l'erreur de recalage pour chaque paramètre de la transformation rigide, dans le cas d'une température T égale à $1,5625$ et de 100 itérations. Elles sont identiques pour 200 itérations, ainsi que pour les températures ayant la même convergence finale (soit $T = 3,125$ et $6,25$ pour les deux nombres d'itérations, et $T = 12,5$ pour $Max_{conf} = 200$). Il est clair que la précision obtenue est excellente. D'ailleurs, l'écart quadratique moyen étant de $0,33$ voxels, on peut dire que le recalage est de précision sous-voxel. Sur le plan du temps de calcul, l'exécution sur un processeur R12000 nécessite un peu plus de 19 minutes, ce qui est tout à fait acceptable.

8.2.5 Parallélisation de l'équation de la diffusion

Nous avons vu dans la première partie (section 1.4.5) que l'équation de la diffusion permet une implantation massivement parallèle en distribuant le calcul de chaque dérivée partielle sur

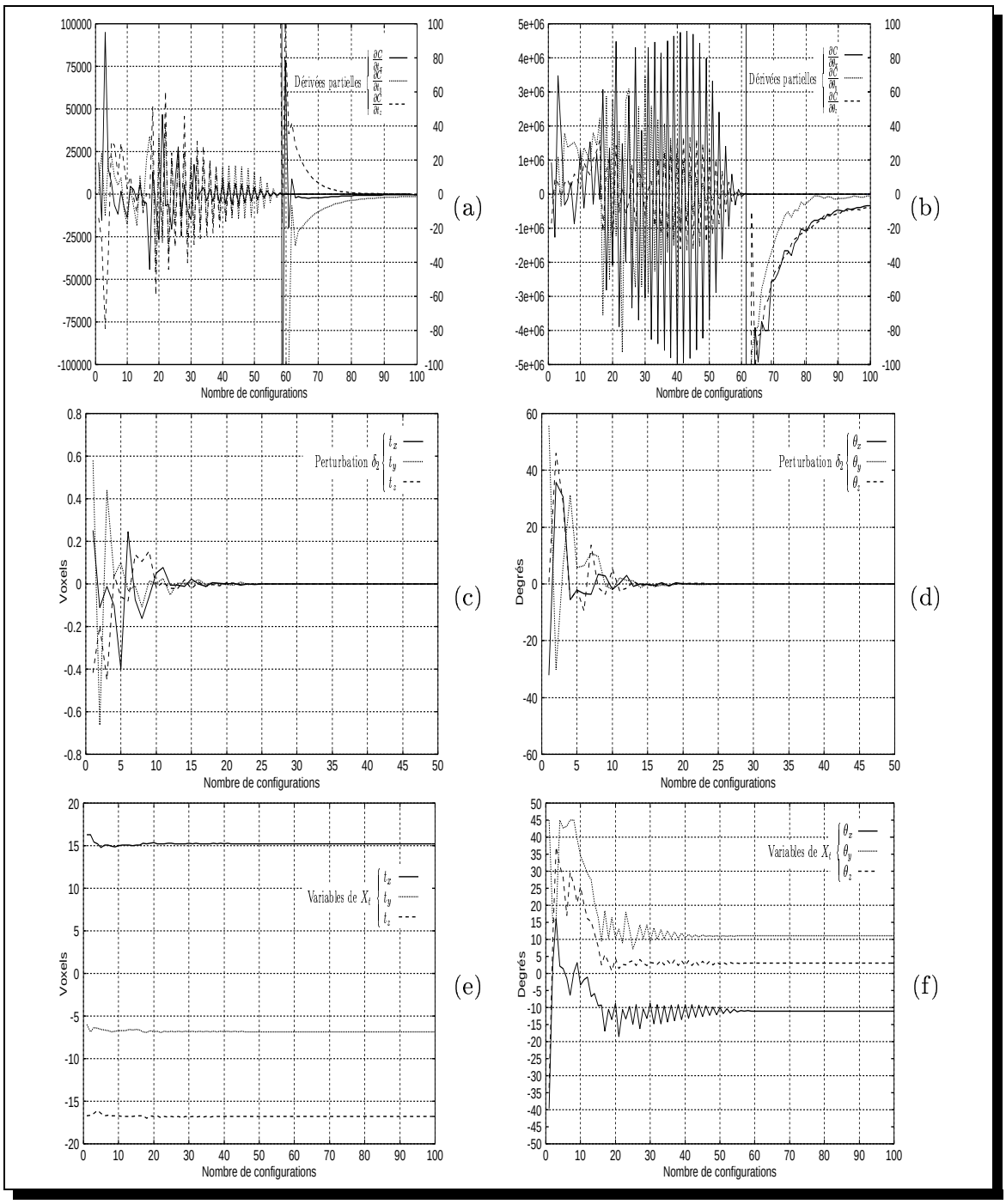


Fig. 8.2 – Illustration sur un recalage rigide par l'équation de la diffusion de l'évolution de : la valeur des dérivées partielles de la translation (a) et de la rotation (b) ; la perturbation δ_2 de la descente du gradient, également pour la translation (c) et la rotation (b) ; des variables de la configuration X_t, i.e. des paramètres de translation (e) et de rotation (f).

un processeur. Cela signifie que dans le cas du recalage rigide, le parallélisme potentiel est de six, i.e. six processeurs peuvent être utilisés simultanément. On en déduit donc que l'on peut espérer au mieux diviser le temps de calcul séquentiel par un facteur six, soit passer de 19 minutes à un peu plus de 3 minutes.

Cependant, dans le cas de la transformation rigide, le calcul du gradient de la fonction de coût (voir les dérivées partielles (8.5) à (8.10)) met en évidence une redondance importante des calculs. De fait, le calcul spécifique à chaque processeur est relativement limité : la seule variation se situe au niveau du vecteur que l'on multiplie avec le gradient associé à chaque voxel de l'image recalée suivant T_Θ (soit $\nabla_{s'} J(s')$). De plus, ce vecteur est constant pour les paramètres de translation ($(1,0,0)^T$ pour t_x, $(0,1,0)^T$ pour t_y et $(0,0,1)^T$ pour t_z), alors que pour les paramètres de rotation, un calcul préalable pour chaque voxel est nécessaire, par exemple $\frac{\partial R}{\partial \theta_z} \times (s-g)$ dans le cas de θ_x. En définitive :

- le temps de calcul d'une dérivée partielle sera différent suivant que l'on traite un paramètre du vecteur de translation ou un angle de rotation, ce qui devrait logiquement se traduire par un déséquilibre de la charge de calcul ;
- on peut estimer que l'apport de la parallélisation sera moindre par rapport à la version séquentielle qui ne calculerait le gradient de chaque voxel de l'image recalée qu'une seule fois, pour ensuite faire chaque multiplication propre à la dérivée partielle d'un paramètre de T_Θ.

En fait, ce type d'algorithme est peu adapté à une mise en œuvre data-parallèle dans le cas précis du recalage rigide, car les calculs locaux spécifiques à chaque processeur sont négligeables. C'est pourquoi, il ne nous a pas semblé intéressant de mettre en œuvre cette parallélisation.

8.3 Recuit simulé adaptatif (Adaptive simulated annealing)

Comme le fait apparaître sa définition (section 1.3.3), cette variante continue du recuit simulé « classique » est bien plus complexe à mettre en œuvre que ce dernier. En effet, alors que le schéma de température dans le recuit simulé n'est habituellement utilisé que pour contrôler l'acceptation des nouvelles configurations. Le recuit simulé adaptatif associe également une température à chacune des composantes/variables de la configuration, et utilise celles-ci pour diriger la recherche dans la phase d'exploration. De plus, il intègre une phase supplémentaire qui est la phase dite de *reannealing*, grâce à laquelle les températures peuvent éventuellement remonter. Cette phase est vraiment une nouveauté dans les algorithmes de type recuit simulé, puisque habituellement, une température n'est supposée que décroître.

En conséquence on pourrait s'attendre à devoir fixer un grand nombre de paramètres. Or, il faut constater que toutes les étapes de l'algorithme sont précisément définies par Lester [63, 64, 65]. Aussi, seuls quatre paramètres doivent-être spécifiés par l'utilisateur :

- le critère d'arrêt, qui dans notre cas sera un nombre maximum de configurations produites. Comme dans l'équation de la diffusion, ce nombre sera noté Max_{conf} ;
- N_{gen}, la longueur d'un palier de températures, i.e. le nombre de configurations qui sont engendrées à températures constantes ;
- N_{acc}, le nombre de configurations qui sont acceptées avant tout déclenchement de la phase de *reannealing* ($N_{acc} \leq N_{gen}$) ;

– enfin le coefficient c qui contrôle la vitesse à laquelle les températures seront abaissées.

Notons que nous utiliserons Max_{conf} pour déterminer à quel moment auront lieu les changements de résolution, en l'occurrence à chaque fois que $\frac{Max_{conf}}{4}$ configurations seront produites. À la résolution la plus fine aucune configuration n'est donc engendrée, l'énergie de la configuration courante est simplement recalculée. Le nombre total de vecteurs Θ à évaluer est égal à $Max_{conf}+5$ (il faut ajouter les quatre réévaluations de la configuration courante, ainsi que le calcul de l'énergie de la configuration initiale).

Voyons maintenant si cet algorithme est apte à résoudre le problème de recalage considéré, et quelle est l'influence respective de chacun des paramètres évoqués ci-dessus.

8.3.1 Performances et comportements de l'algorithme séquentiel

Les expérimentations préalables ont fait apparaître que pour le problème du recalage rigide, on peut se passer de la phase de *reannealing*. En effet, d'une part cette phase n'améliore pas la qualité des recalages et d'autre part les remontées de températures qu'elle induit affecte la vitesse de convergence. La conséquence directe de la non-utilisation du *reannealing* est une simplification de l'algorithme :

– les températures qui étaient associées à chacune des composantes de la configuration X_t, i.e. $T_{s,t}, s \in \{1, ..., N\}$ où N est le nombre de variables (soit six pour le recalage rigide), peuvent être fusionnées dans une température unique. De fait, ces températures ont toutes la même valeur initiale et le même schéma de refroidissement (temps de recuit également identique), seule la phase de *reannealing* aurait pu les modifier de façon à ce que chacune ait une valeur qui lui soit propre.

– un paramètre de moins à fixer, à savoir le nombre de configurations acceptées N_{acc}.

Toutefois, la suppression de cette étape du recuit simulé adaptatif lui enlève la caractéristique d'« auto-adaptation », et rend donc l'algorithme plus sensible au choix des paramètres N_{gen} et c. C'est ce que nous allons voir dans la suite. Finalement, on peut remarquer que la suppression du *reannealing* fait que l'ASA peut être vu en quelque sorte comme un recuit simulé « classique » avec une phase d'exploration spécifique et un schéma de refroidissement très rapide.

Convergence

Afin de mettre en évidence l'influence sur l'algorithme des différents paramètres, l'ASA a été exécuté en considérant plusieurs jeux de paramètres. Dans ces derniers, Max_{conf} est supposé avoir la même valeur, soit 900 configurations, alors que pour N_{gen} et c, tandis que l'un a une valeur fixe, l'autre évolue. En effet, le critère d'arrêt n'a à proprement parler aucune influence sur la convergence de l'algorithme. Idéalement, il doit juste être choisi de façon à ne pas stopper l'algorithme si celui-ci n'a pas fini de converger. Ainsi, N_{gen} prend successivement les valeurs 10, 20, 40 et 80 avec $c = 7,5$, puis, réciproquement N_{gen} est fixé à 20 et c est échantillonné régulièrement dans son intervalle de définition ($[1; 10]$). Au vu des courbes de convergence que l'on obtient pour l'erreur quadratique moyenne lors de la mise en œuvre des jeux de paramètres (graphiques 8.3(a) et 8.3(b)), il faut reconnaître que c'est le paramètre c qui influe essentiellement sur la convergence de l'algorithme. Le paramètre N_{gen} n'a lui qu'un impact limité sur l'algorithme : la vitesse de convergence est légèrement affectée par la longueur des paliers de température. En effet, comme le montrent les graphiques (c), (d), (e) et (f) de la figure 8.3[2], c'est c qui contrôle la vitesse à

[2] Attention l'ordonnée est représentée suivant une échelle logarithmique.

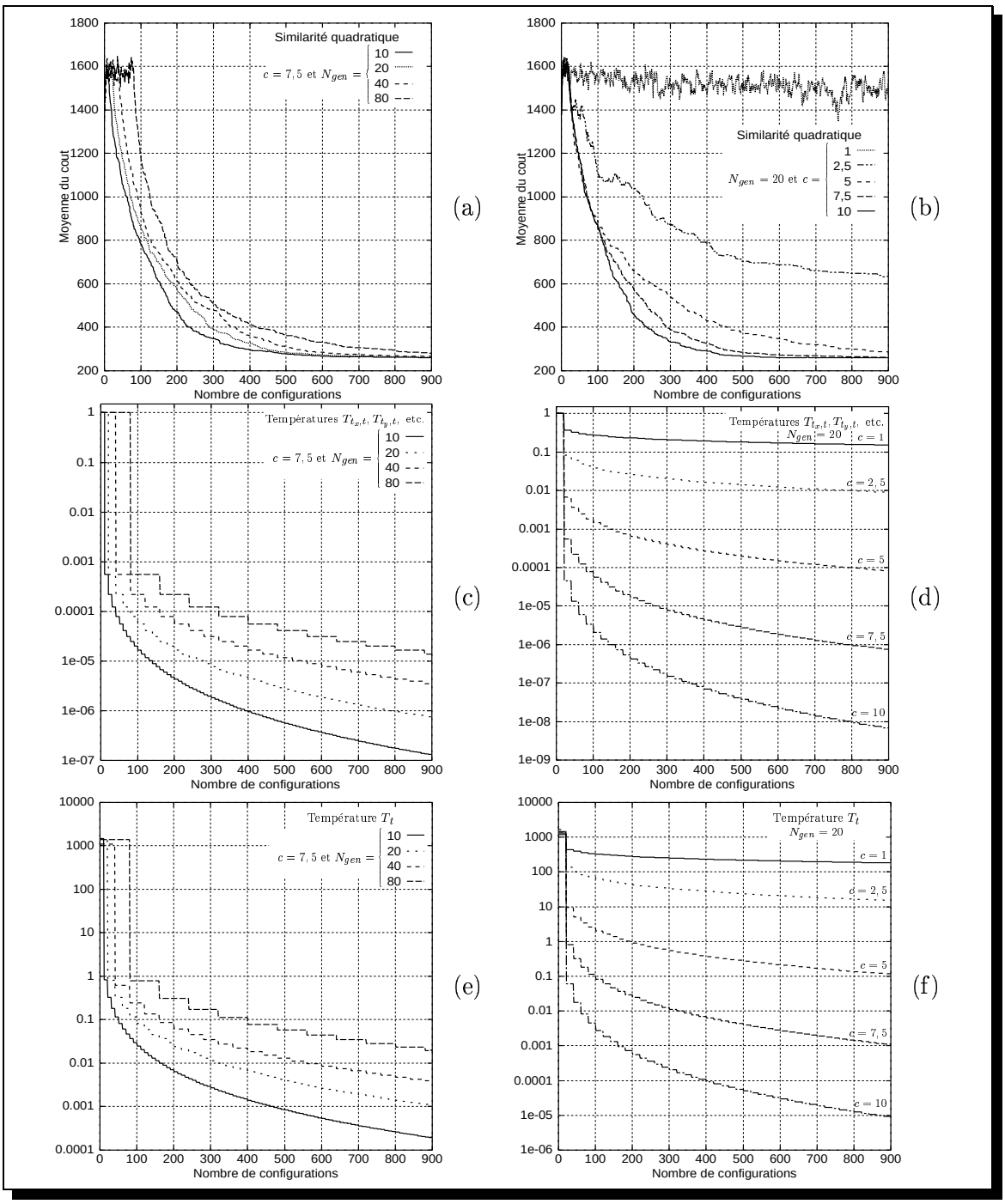


FIG. 8.3 – Recuit simulé adaptatif séquentiel - évolution moyenne du coût au cours de l'algorithme pour différentes valeurs de (a) N_{gen} et (b) c ; (c) et (d) présentent l'évolution associée respectivement à (a) et (b) de la température des variables de Θ ; (e) et (f) décrivent de la même manière la décroissance de la température d'acceptation.

laquelle les températures ($T_{s,t}$ est associée aux variables et T_t est la température d'acceptation) décroissent : plus c est proche de sa borne inférieure, plus la décroissance est lente, inversement, la décroissance la plus rapide s'obtient avec la borne supérieure. Or, plus la température d'acceptation baisse, moins on autorise de remontée en énergie, et donc plus rapidement l'algorithme converge vers un optimum (éventuellement local).

D'ailleurs, comparé au recuit simulé « classique », on voit que lorsque c est proche de sa borne supérieure, la décroissance des températures est extrêmement rapide. De plus, les coefficients de réduction utilisés pour abaisser les températures dans l'ASA évoluent, puisqu'il décroissent suivant une exponentielle inverse. Ceci est mis plus clairement en évidence à la figure 8.4(a) (l'ordonnée a une échelle logarithmique) qui montre l'évolution du coefficient de réduction α_t des températures pour $c = 1$ et $c = 10$, avec $N_{gen} = 20$, soit 45 paliers de températures. Rappelons qu'originellement il y a sept coefficients de réduction (un par variable de Θ, plus un pour la température d'acceptation), mais que ce nombre est réduit à un seul du fait de l'absence de la phase de *reannealing* et qu'il s'obtient à partir de (1.29) :

$$\alpha_t = \exp\left(-c \cdot t^{\frac{1}{6}}\right) \tag{8.15}$$

dans le cas du recalage rigide.

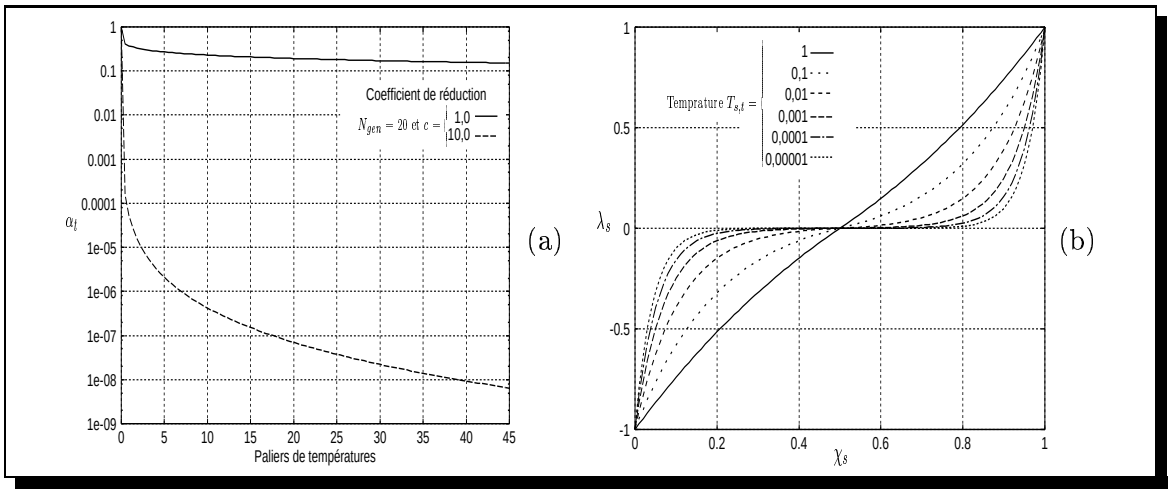


FIG. 8.4 – Recuit simulé adaptatif séquentiel - (a) évolution du coefficient de réduction des températures pour différentes valeurs du paramètre c; (b) évolution de la variable aléatoire λ_s utilisée dans la phase d'exploration, pour plusieurs températures $T_{s,t}$.

Par conséquent, on comprend aisément, que lorsque c a une valeur proche de sa borne inférieure, la décroissance de la température d'acceptation est tellement lente, que les remontées d'énergie sont très nombreuses et la convergence à peine perceptible. Puis au fur et à mesure que c croît, les remontées en énergie sont de moins en moins apparentes, disparaissant très rapidement, i.e. à partir de $c = 5$. En fait, lorsque le paramètre c vaut 10, la décroissance de la température d'acceptation est telle que, dès sa deuxième valeur, une augmentation de l'énergie est quasi impossible. Aussi, on pourrait s'attendre à être bloqué dans un optimum local, puisque

l'algorithme ne fait qu'accepter des configurations aboutissant à une descente de l'énergie. Or la courbe de convergence moyenne (figure 8.3(b)) montre bien que ce n'est pas le cas.

Deux facteurs peuvent expliquer cette observation :

- d'une part le lissage de la fonction de coût résultant de l'approche multirésolution est tel que les minima locaux ont pratiquement disparu ;
- d'autre part, la manière dont est parcouru l'espace de recherche par l'algorithme (la phase d'exploration). C'est-à-dire que la méthode utilisée pour produire les nouvelles configurations prévient le blocage prématuré dans le bassin d'attraction d'un optimum local.

Le lissage de la fonction de coût réduit effectivement le nombre de minima locaux, mais ne les fait pas tous disparaître. Statistiquement, il est donc peu probable que la configuration de départ, qui est construite aléatoirement, soit toujours dans le voisinage du minimum global. Or il faut bien constater qu'aucune exécution du recuit simulé adaptatif avec c égal à 10 n'a convergé vers un minimum local. C'est donc la phase d'exploration qui semble *a priori* expliquer le bon comportement de l'algorithme.

De fait, l'étude de la phase d'exploration du recuit simulé adaptatif (section 1.3.3) montre que cette dernière ne consiste pas à considérer un petit voisinage autour de la configuration courante, mais est définie de façon à permettre de faire de grands sauts dans l'espace de recherche, sauts dont l'amplitude est contrôlée par les températures associées aux composantes des configurations. En effet, comme le montre l'équation (1.22), pour produire une nouvelle configuration Y_t, on additionne à chacune des composantes de X_t une pondération aléatoire de la taille de l'intervalle de recherche. Cette pondération $\lambda_s \in [-1;1]$ est différente pour chaque variable $x_{t,s}$ et évolue suivant la température associée à celle-ci. Ainsi, comme le montre la figure 8.4(b) dans le cas du recalage rigide (une seule température commune à toutes les composantes), initialement lorsque $T_{s,t}$ est élevée, Y_t est obtenue en perturbant fortement X_t. Ensuite, à mesure que la température associée aux composantes baisse, les perturbations sont moins importantes, mais toutefois suffisantes pour produire des configurations très différentes de la configuration courante (voir les graphiques de la figure 8.5). En tout état de cause, la phase d'exploration telle que l'a définie Ingber, fait que l'algorithme ne risque pas d'être prématurément bloqué dans un optimum local.

L'évolution du facteur de pondération dans le cas du recalage rigide (graphique 8.4(b)) permet aussi de mieux comprendre le principe sous-jacent à la phase dite de *reannealing*. En effet, cette phase a notamment pour rôle de faire remonter les températures des composantes affectant peu l'énergie. Ce qui se traduit implicitement par la modification des facteurs de pondération associés de manière à produire des perturbations plus importantes dans la phase d'exploration.

Afin de vérifier que le comportement que nous avons observé et donc que les explications proposées sont indépendantes de la fonction de coût, l'information mutuelle a été optimisée en considérant les trois meilleurs jeux de paramètres. Les courbes de convergence que l'on obtient, présentées dans la figure ci-après, mettent en évidence un comportement analogue à celui exhibé par l'algorithme lors du traitement de l'erreur quadratique moyenne. Cela semble donc signifier que les explications avancées restent valables pour l'information mutuelle. Remarquons que les courbes présentent les quatre pics attendus (voir l'introduction de ce chapitre, section 8.1), chacun résultant d'un changement de résolution.

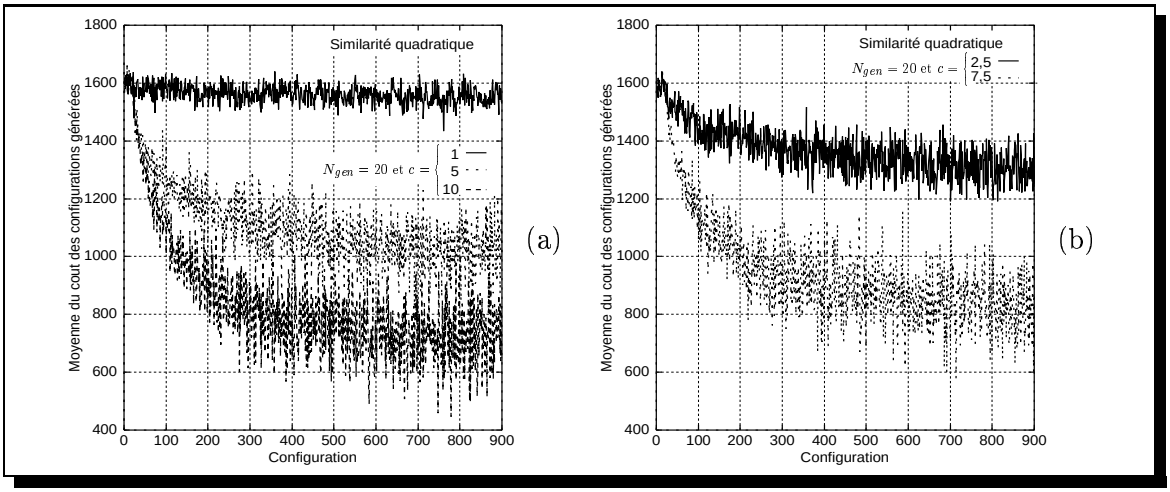


FIG. 8.5 – Recuit simulé adaptatif séquentiel - évolution moyenne sur les 20 recalages, dans le cas de l'erreur quadratique moyenne, du coût des configurations produites par la phase d'exploration

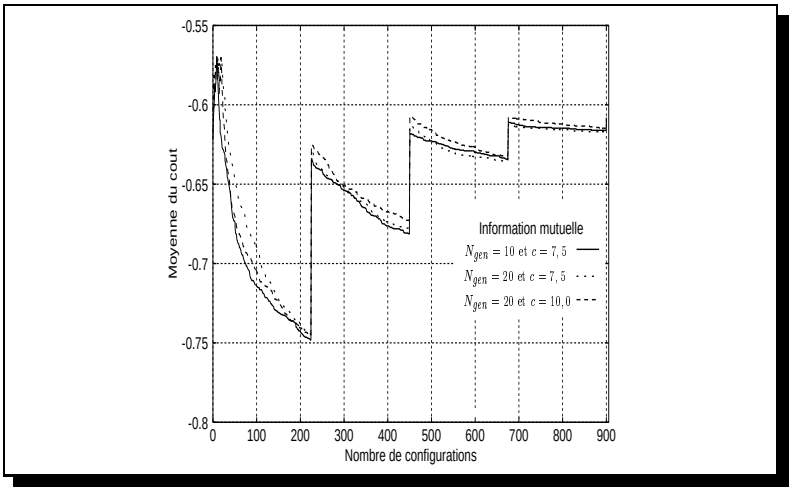


FIG. 8.6 – Recuit simulé adaptatif séquentiel - évolution moyenne du coût dans le cas de l'information mutuelle lorsque l'on considère les jeux de paramètres ayant donné les meilleurs résultats pour l'erreur quadratique moyenne.

Qualité des recalages et temps d'exécution

Une meilleure appréciation de la qualité des recalages que l'*Adaptive Simulated Annealing* « simplifié » (i.e. dans notre cas, sans « auto-adaptation ») permet d'obtenir, peut être donnée au moyen de statistiques sur les erreurs de recalage. Le calcul de ces valeurs statistiques sur 20 recalages (tableau 8.3), pour les deux fonctions de coût, en considérant à chaque fois les trois jeux de paramètres optimaux, montre que le recalage est précis.

Il faut particulièrement noter la faiblesse des erreurs au niveau des paramètres de rotation θ_x, θ_y et θ_z, qui sont pourtant les plus difficiles à trouver. D'autant plus que contrairement aux paramètres de translation, ils sont initialisés aléatoirement et ont un espace de recherche étendu (l'intervalle de recherche des angles de rotation est en effet $[-45°; +45°]$). Remarquons également la relative constance des erreurs de translation, alors que dans la cas de la rotation les variations sont plus importantes. Enfin, il est à noter que, comparé à l'erreur quadratique moyenne, l'information mutuelle se traduit globalement par des recalages de légèrement moindre qualité. Néanmoins, le calcul de l'écart quadratique moyen montre que dans les deux cas la précision sous-voxel désirée est atteinte : pour l'erreur quadratique moyenne, l'écart le moins bon est de $0,37 \pm 0,13$ voxels, contre $0,40 \pm 0,16$ voxels pour l'information mutuelle.

	Erreur quadratique moyenne			Information mutuelle		
$N_{gen}; c$	$10; 7,5$	$20; 7,5$	$20; 10$	$10; 7,5$	$20; 7,5$	$20; 10$
Δt_x	$0,21 \pm 0,15$	$0,20 \pm 0,15$	$0,20 \pm 0,15$	$0,21 \pm 0,20$	$0,21 \pm 0,15$	$0,21 \pm 0,15$
Δt_y	$0,14 \pm 0,11$	$0,14 \pm 0,09$	$0,14 \pm 0,10$	$0,14 \pm 0,10$	$0,13 \pm 0,09$	$0,15 \pm 0,10$
Δt_z	$0,17 \pm 0,10$	$0,16 \pm 0,09$	$0,17 \pm 0,10$	$0,16 \pm 0,11$	$0,18 \pm 0,09$	$0,16 \pm 0,09$
$\Delta \theta_x$	$0,05 \pm 0,04$	$0,08 \pm 0,06$	$0,04 \pm 0,03$	$0,04 \pm 0,04$	$0,10 \pm 0,09$	$0,06 \pm 0,05$
$\Delta \theta_y$	$0,07 \pm 0,06$	$0,08 \pm 0,07$	$0,02 \pm 0,02$	$0,06 \pm 0,05$	$0,07 \pm 0,06$	$0,05 \pm 0,03$
$\Delta \theta_z$	$0,06 \pm 0,05$	$0,07 \pm 0,04$	$0,03 \pm 0,02$	$0,07 \pm 0,07$	$0,14 \pm 0,16$	$0,07 \pm 0,05$

Note. Les erreurs de translation sont données en voxels et les erreurs de rotation en degrés.

Tab. 8.3 – Statistiques sur les erreurs de recalage du recuit simulé adaptatif séquentiel.

Concernant les temps de calcul, ceux-ci sont respectivement pour l'erreur quadratique moyenne et l'information mutuelle, de l'ordre de 10 minutes 15 secondes et d'un peu plus de 14 minutes. Le surcoût non négligeable de l'information mutuelle s'explique par un plus grand nombre de calculs à effectuer. En effet, pour calculer les probabilités sur lesquelles s'appuie l'information mutuelle, l'histogramme de l'image de référence et celui de l'image recalée, ainsi que leur histogramme conjoint doivent être préalablement calculés.

8.3.2 Parallélisation de l'Adaptive Simulated Annealing

Parmi toutes les parallélisations des algorithmes de type recuit simulé que nous avons présentées dans le premier chapitre (section 1.4), une grande partie peut être considérée sans problème pour paralléliser le recuit simulé adaptatif. En fait, on peut directement utiliser tous les schémas de parallélisation dès lors qu'ils se traduisent par un algorithme parallèle ayant un comportement identique à celui de la version séquentielle, à savoir :

① La mise en œuvre de recuits multiples parallèles, avec ou sans interactions.

② La parallélisation par essais multiples avec une granularité fixe ou variable, par exemple l'algorithme de « division » de Aarts et Bont [1].

③ L'utilisation d'une ferme de processeurs en considérant un schéma maître/esclaves.

Il nous a donc semblé plus intéressant d'étudier les parallélisations, qui en plus de réduire le temps de calcul, se distinguent de l'algorithme séquentiel au niveau de la chaîne de Markov qu'elles produisent. C'est pourquoi, dans la suite nous allons nous focaliser sur la parallélisation par recuits multi-températures.

En outre, l'étude de cette approche data-parallèle est attrayante dans le cas de l'ASA en raison de l'idée sur laquelle elle repose : faire évoluer en parallèle des recuits simulés à une température d'acceptation différente pour chaque processeur (cf. figures 1.4 et 1.5). Or, comme le recuit simulé adaptatif intègre aussi des températures qui sont associées aux variables des configurations, il est intéressant de voir quel est le comportement de l'algorithme lorsque l'on utilise la même approche pour ces températures que pour la température d'acceptation. D'autre part se pose le problème de la prise en compte ou non de la phase de *reannealing*. En effet, comme cette phase de l'algorithme peut faire éventuellement remonter la température d'acceptation au cours de l'algorithme, il faut veiller à ce que la propagation des configurations se fasse toujours comme il faut : dans le cas déterministe, du processeur ayant la température d'acceptation la plus élevée vers la plus basse. Une solution simple consisterait à réordonnancer préalablement à toute propagation les températures d'acceptation, puis en utilisant le même schéma de réordonnancement, de redistribuer également les températures associées aux variables. En filigrane apparaît la question de la stratégie à adopter vis-à-vis de ces remontées, i.e. s'il ne faut pas envisager d'ajouter un schéma de refroidissement afin d'abaisser à nouveau les températures qui ont augmenté.

Notons que le problème que nous venons d'évoquer ne se présente pas dans le cas du recalage rigide, puisque nous n'avons pas utilisé de phase de *reannealing*. Les courbes de convergence des versions parallèles seront obtenues en considérant le jeu de paramètres suivant : $N_{gen} = 20$ et $c = 7,5$, c'est-à-dire un des jeux ayant donné les meilleurs résultats en séquentiel. Enfin, le critère d'arrêt des algorithmes parallèles sera défini de façon à ce que le nombre total de configurations engendrées soit approximativement égal à celui de la version séquentielle (Max_{conf}).

Propagation déterministe des configurations

Le premier recuit simulé adaptatif parallèle multi-températures que nous avons implanté est une adaptation directe de l'algorithme proposé par Graffigne pour le recuit simulé « classique » [50] (celui-ci a été décrit dans la section 1.4.1). En effet, comme nous l'avons déjà fait remarqué, sans *reannealing* on peut estimer que l'ASA est un recuit simulé « classique » qui se distingue par un opérateur de voisinage particulier et un protocole de refroidissement extrêmement rapide. Aussi, la seule particularité notable de la version multi-températures « adaptative » pour le recalage rigide se situe au niveau du traitement de la température $T_{s,t}$ associée aux composantes des configurations. Dans la continuité de la température d'acceptation, $T_{s,t}$ sera supposée fixe sur chaque processeur.

Pour choisir les températures T_t et $T_{s,t}$ à affecter à un processeur, l'approche suivante a été retenue :

① On calcule à l'avance toutes les températures grâce au coefficient de réduction α_t (8.15) (les températures initiales T_0 et $T_{s,0}$ sont pré-définies), sachant que le nombre de paliers

de températures est défini à partir de Max_{conf} et de N_{gen}. On obtient ainsi deux ensembles de $\frac{Max_{conf}}{N_{gen}}$ températures, l'un pour l'acceptation, l'autre pour produire les nouvelles configurations :

$$T_{\frac{Max_{conf}}{N_{gen}}-1} \leq T_{\frac{Max_{conf}}{N_{gen}}-2} \leq ... \leq T_1 \leq T_0 \tag{8.16}$$

$$T_{s,\frac{Max_{conf}}{N_{gen}}-1} \leq T_{s,\frac{Max_{conf}}{N_{gen}}-2} \leq ... \leq T_{s,1} \leq T_{s,0} \tag{8.17}$$

② Supposons que l'on dispose de p processeurs, le processeur d'indice j reçoit alors les températures

$$T_{\frac{(j+1)\cdot Max_{conf}}{N_{gen}\cdot(p+1)}} \text{ et } T_{s,\frac{(j+1)\cdot Max_{conf}}{N_{gen}\cdot(p+1)}} \tag{8.18}$$

Ceci revient à échantillonner régulièrement p températures en commençant par celle se trouvant à la position $\frac{Max_{conf}}{N_{gen}\cdot(p+1)}$. Un inconvénient peut se poser du fait que pour un nombre de processeurs très restreint, les valeurs les plus hautes pour T_t et $T_{s,t}$ peuvent déjà être très basses. Dans ce cas, l'alternative suivante est envisageable : la température la plus faible et la plus élevée sont obligatoirement choisies, ensuite les températures manquantes sont échantillonnées régulièrement comme précédemment, mais en considérant que l'on ne dispose plus que de $p-2$ processeurs.

Quelques mots pour finir sur le nombre total de phases de propagations de configurations, Max_{prop}, que nous utilisons comme critère d'arrêt de l'algorithme data-parallèle. Celui-ci se calcule à partir de Max_{conf}, du nombre s de configurations engendrées sur un processeur entre deux propagations (la période) et du nombre de processeurs :

$$Max_{prop} = \frac{Max_{conf}}{p \cdot s} \tag{8.19}$$

En effet, le nombre de configurations générées sur un processeur est égal à Max_{conf} sur p, en divisant par s on détermine donc le nombre de phases de communications entre les processeurs. Dans le cadre du traitement multirésolution des images, ce nombre est aussi utilisé pour contrôler à quel moment s'opère un changement de résolution, soit toutes les $\frac{Max_{prop}}{4}$ propagations parallèles.

Prenons pour fonction d'énergie l'erreur quadratique moyenne. Les expérimentations que nous avons menées sur 4, 8 et 12 processeurs, avec à chaque fois des périodes entre deux propagations de 1, 5, 10 et 20 configurations, montrent que l'approche multi-températures déterministe reste *a priori* valable pour le recuit simulé adaptatif « simplifié » (sans *reannealing*). En effet, comme l'illustre les courbes d'évolution moyenne du coût de la configuration courante sur le processeur ayant la température d'acceptation la plus basse, figure 8.7, l'algorithme semble apparemment toujours converger vers le bassin d'attraction de l'optimum global.

On constate d'autre part que l'augmentation du nombre de processeurs se traduit par une baisse de la vitesse de convergence, ce qui est normal, puisque le nombre de propagations nécessaires pour transmettre éventuellement une configuration de la température d'acceptation la plus élevée vers la plus basse croît. Ceci est d'ailleurs très clair lorsque l'on étudie l'écart entre les courbes obtenues avec les différents nombres de processeurs, mais un même nombre de propagations : plus s est grand, donc les propagations moindres, plus les courbes sont éloignées les unes des autres.

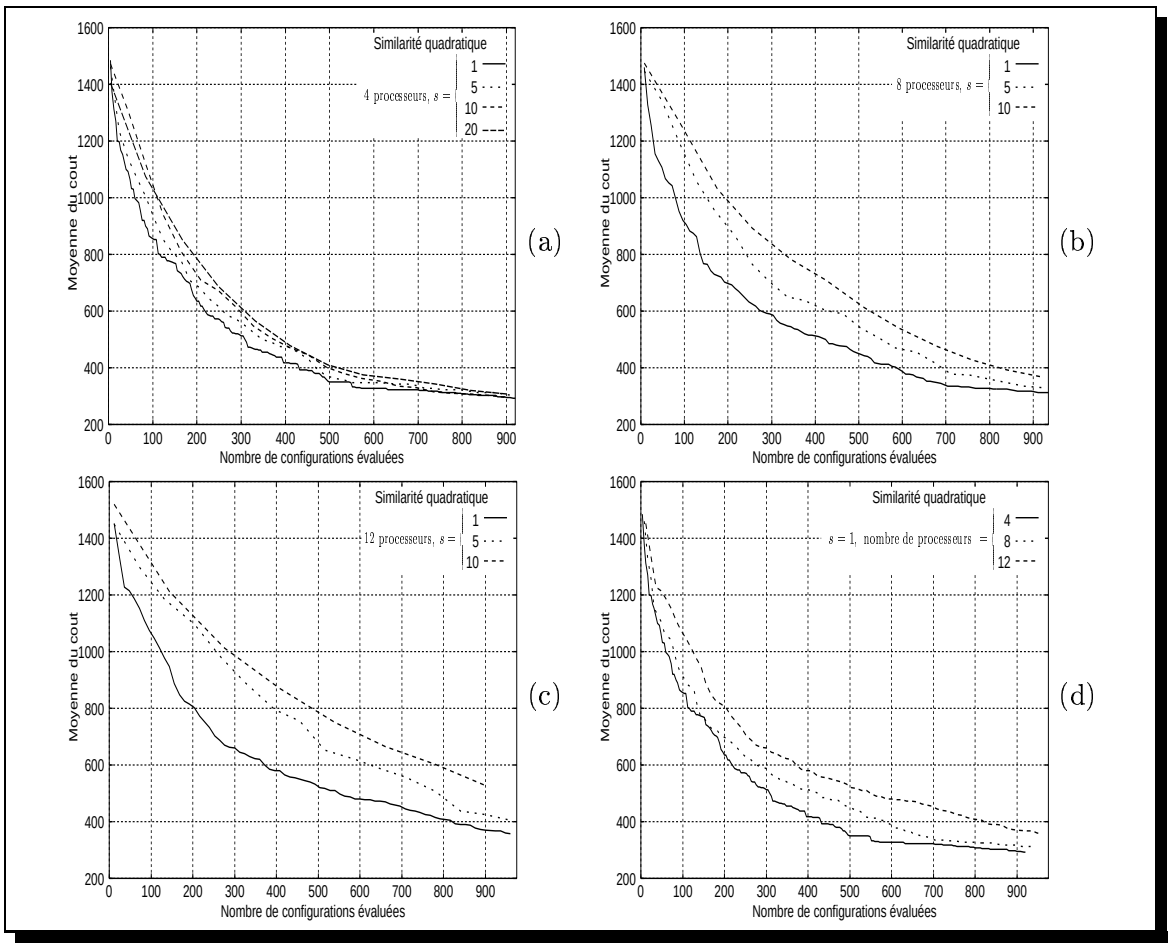


FIG. 8.7 – Recuit simulé adaptatif multi-températures déterministe - évolution moyenne sur 20 recalages du coût de la configuration courante du processeur ayant les températures les plus basses, pour différents nombres de processeurs et périodes s.

Les courbes de la version quadriprocesseur (graphique 8.7(a)) montrent la corrélation existante entre la vitesse de convergence et le nombre de propagations. En effet, plus les propagations sont nombreuses, plus l'exploration de l'espace de recherche est « guidé ». Alors que les processeurs ayant une température associée aux variables qui est élevée font un parcours à grande échelle de l'espace de recherche, à mesure que $T_{s,t}$ baisse, l'amplitude des perturbations de la configuration courante diminue, ce qui se traduit par une exploration plus « fine ». Aussi, un plus grand nombre de propagations fournira plus rapidement aux processeurs à basse température des configurations se trouvant dans différentes vallées de la fonction d'énergie. Au final, la configuration trouvée par les différentes instances sur 4 processeurs de l'algorithme parallèle sont cependant d'énergie assez proche.

Propagation probabiliste des configurations

Nous avons ensuite implanté une seconde version multi-températures pour l'ASA en considérant cette fois le schéma de propagation probabiliste introduit par Aarts et van Laarhoven [3] (voir section 1.4.1). Par rapport à l'algorithme multi-températures original, dit « systolique », notre version se caractérise au niveau du nombre de températures qui sont traitées successivement par chaque processeur. En effet, à l'image de la version déterministe, l'algorithme systolique suppose qu'il y a autant de températures d'acceptation que de processeurs (rappelons que pour l'*Adaptive Simulated Annealing*, ce sont des couples de températures : $(T_t, T_{s,t}), t \in \{0, ..., p-1\}$). Or dans notre implantation, nous avons décidé de traiter le même nombre de températures que dans la version séquentielle, i.e. pour $Max_{conf} = 900$ et $N_{gen} = 20$ (longueur d'un palier de températures), 45 couples de températures. Ainsi, tout processeur génère pour chaque couple de températures une sous-chaîne homogène de longueur $\frac{N_{gen}}{p}$, où p est le nombre de processeurs. Mises bout à bout, ces sous-chaînes aboutissent à une chaîne homogène de longueur identique à celle que l'on obtient dans la version séquentielle. Notre adaptation multi-températures probabiliste est donc relativement proche de l'algorithme séquentiel. En revanche au niveau du schéma de propagation des configurations, aucune modification n'a été apportée.

Le contrôle des changements de résolution se fait de la même manière que dans la version multi-températures déterministe présentée ci-dessus. La seule différence se situe au niveau du calcul du nombre de phases de propagations :

$$Max_{prop} = \frac{Max_{conf}}{N_{gen}} + p - 1 \tag{8.20}$$

C'est-à-dire qu'il y autant de phases de propagations que de températures, auxquelles il faut ajouter les $p-1$ propagations parallèles dues au démarrage décalé des processeurs.

La figure 8.8 montre le comportement de cet algorithme multi-températures probabiliste, dit « original », lors de l'optimisation de l'erreur quadratique moyenne avec 4 et 8 processeurs. À noter que ces courbes correspondent à l'évolution moyenne du coût de la configuration courante sur le processeur ayant démarré le plus récemment.

On remarque que la dégradation de la convergence croît avec le nombre de processeurs. Ceci s'explique par le fait que plus on a de processeurs, plus les sous-chaînes homogènes sont courtes, ce qui influe directement sur la convergence de l'algorithme. Enfin, comme attendu, le schéma de propagation probabiliste se traduit par des remontées de l'énergie (du coût). Pour conclure on peut estimer que cette implantation n'est pas intéressante, car elle ne permettra pas d'obtenir de bons résultats avec un nombre important de processeurs sans accroître la longueur des paliers de températures, et donc le nombre total de configurations engendrées.

Dans un second temps, nous avons modifié la façon dont est choisie, après chaque interaction entre les processeurs, la configuration de départ de la prochaine sous-chaîne. L'approche que nous avons privilégiée est la suivante : le processeur $P_j, j \in \{1, ..., p-1\}$ accepte pour configuration de départ la configuration reçue du processeur P_{j-1} ($X_{j-1,t}$) avec la probabilité

$$\min\left[1, \frac{1}{1 + \exp\left(\frac{E(X_{j-1,t}) - E(X_{j,t-1})}{T_{j,t-1}}\right)}\right] \tag{8.21}$$

Cela revient à utiliser la probabilité de la phase d'acceptation de l'ASA (1.24).

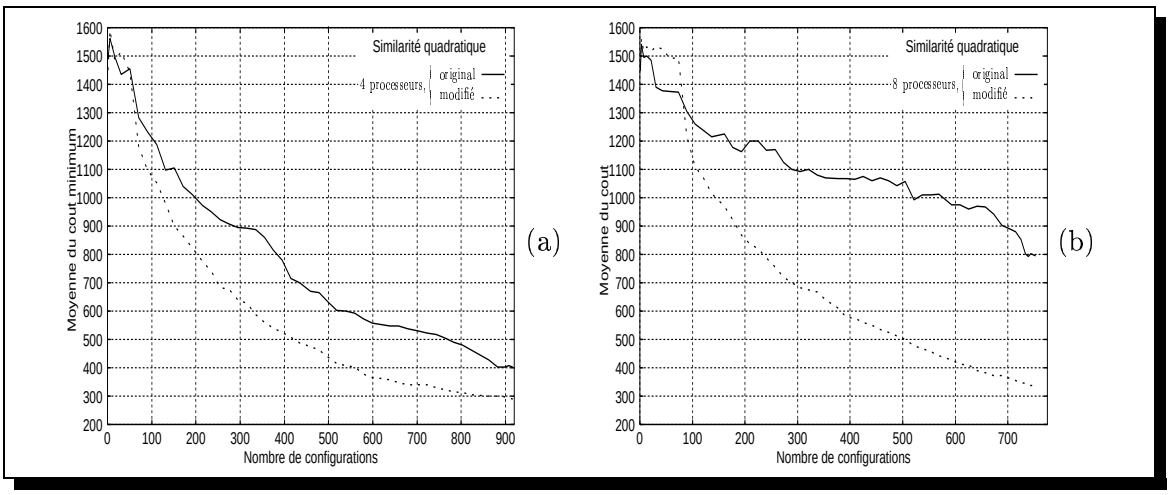


FIG. 8.8 – Recuit simulé adaptatif multi-températures probabiliste - évolution moyenne du coût de la configuration courante sur le dernier processeur qui a été activé.

Les courbes de convergence de cette deuxième version de l'algorithme systolique, que nous dirons « modifié », sont présentées dans la même figure que la version dite « originale » (conditions expérimentales identiques). Elles mettent en lumière un comportement beaucoup plus intéressant : une bonne vitesse de convergence et une dégradation nettement moindre des performances lorsque le nombre de processeurs augmente. En effet, la probabilité d'acceptation (8.21) se traduit par des remontées d'énergie uniquement lorsque les températures d'acceptation sont relativement hautes, à basse température on n'accepte quasiment que les configurations d'énergie plus faible.

Nous terminons par deux graphiques donnant une vue synthétique de la convergence de différents algorithmes multi-températures, les uns par rapport aux autres, mais également en comparaison de la version séquentielle. Le premier ensemble de courbes est relatif à l'erreur quadratique moyenne (figure 8.9(a)), alors que le second fut obtenu en prenant pour fonction d'énergie l'information mutuelle (figure 8.9(b)).

Le classement des courbes de convergence met au jour un comportement analogue pour les deux fonctions de coût. D'une part, la version déterministe quadriprocesseur avec une période s égale à une configuration et l'algorithme systolique « modifié » aboutissent à des configurations de qualité équivalente. D'autre part, une implantation directe de l'algorithme multi-températures probabiliste proposé par Aarts *et al.* [3] est vraiment inadéquate au problème du recalage rigide. En dernier lieu, l'algorithme multi-températures déterministe présente dans les deux cas une vitesse de convergence initiale nettement plus rapide que la version séquentielle, ce qui était attendu en raison du choix déterministe des configurations et du faible nombre de températures (la propagation des configurations vers les basses températures est très rapide).

Comparé à la version séquentielle, la configuration finale produite par les meilleurs algorithmes multi-températures est évidemment de moins bonne qualité. Par voie de conséquence, si on veut un recalage ayant une précision du même ordre que l'algorithme séquentiel, le recours à une méthode d'optimisation locale s'impose.

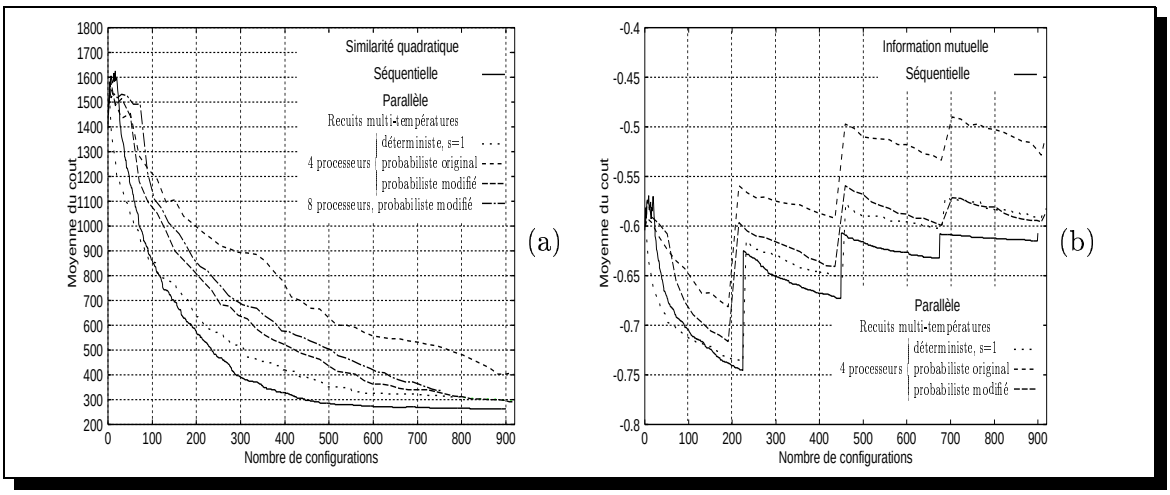


FIG. 8.9 – Recuit simulé adaptatif - Courbes de convergence des algorithmes multi-températures et de la version séquentielle lors de la minimisation (a) de l'erreur quadratique moyenne ; (b) de l'information mutuelle.

✐ Qualité des recalages produits par les parallélisations multi-températures et temps de calcul

Le tableau 8.4 qui regroupe les statistiques sur les erreurs de recalage des versions multi-températures quadriprocesseur, avec une période de une configuration pour la version déterministe, est en adéquation complète avec le classement évoqué plus haut. En particulier le mauvais comportement de l'algorithme systolique « original » est très clair : l'écart quadratique moyen entre chaque image recalée et la solution optimale est en moyenne dans le cas le plus favorable (similarité quadratique) de $1,68 \pm 1,40$ voxels. Les configurations finales ne sont donc pas vraiment dans la voisinage immédiat de l'optimum recherché. La version dite « modifiée » et l'approche déterministe donnent de meilleurs résultats. Ainsi, les configurations finales obtenues lors de la minimisation de l'erreur quadratique moyenne ne sont pas très loin du minimum global. Néanmoins, pour l'information mutuelle, ces deux recuits multi-températures parallèles donnent comme l'algorithme systolique « original » des recalages de précision supérieure au voxel (l'EQM est égal à $0,76 \pm 0,26$ voxels pour l'algorithme déterministe et $0,72 \pm 0,26$ voxels pour le systolique « modifié »). Cela montre manifestement la pertinence de la dernière observation que nous avons faite dans le paragraphe précédent.

Lorsqu'on regarde les temps de calcul des différentes adaptations multi-températures de l'ASA (tableau 8.5), on constate que si le nombre de propagations parallèles, i.e. Max_{prop}, est comme Max_{conf} un multiple de quatre, l'accélération est excellente. Par exemple, les versions systoliques sur 4 et 8 processeurs qui vérifient cette propriété, ont respectivement une accélération légèrement inférieure ou égale à 3,5 et de l'ordre de 7. L'algorithme multi-températures déterministe remplit moins facilement cette condition. La version quadriprocesseur exhibe notamment une dégradation de l'accélération qui croît avec s. En effet, dès lors que Max_{prop} n'est pas divisible par 4, le nombre de configurations engendrées à la résolution la plus fine, qui est la plus coûteuse, n'est plus nul, mais est fonction du reste de la division. Il est donc logique que la comparaison par rapport à la version séquentielle, qui ne produit aucune configuration lorsque l'on considère tous

	Erreur quadratique moyenne			Information mutuelle		
$N_{gen}; c$	Déterministe	Probabiliste		Déterministe	Probabiliste	
$20; 7, 5$	$s = 1$	Original	Modifié	$s = 1$	Original	Modifié
Δt_x	$0,22 \pm 0,12$	$0,31 \pm 0,14$	$0,18 \pm 0,13$	$0,28 \pm 0,18$	$0,41 \pm 0,35$	$0,20 \pm 0,15$
Δt_y	$0,17 \pm 0,10$	$0,22 \pm 0,21$	$0,17 \pm 0,10$	$0,17 \pm 0,16$	$0,49 \pm 0,54$	$0,20 \pm 0,16$
Δt_z	$0,18 \pm 0,11$	$0,33 \pm 0,32$	$0,20 \pm 0,15$	$0,18 \pm 0,16$	$0,45 \pm 0,25$	$0,17 \pm 0,17$
$\Delta \theta_x$	$0,32 \pm 0,23$	$0,85 \pm 0,66$	$0,30 \pm 0,15$	$0,39 \pm 0,25$	$2,52 \pm 6,88$	$0,24 \pm 0,18$
$\Delta \theta_y$	$0,30 \pm 0,26$	$0,74 \pm 0,86$	$0,28 \pm 0,19$	$0,22 \pm 0,19$	$1,92 \pm 4,95$	$0,31 \pm 0,17$
$\Delta \theta_z$	$0,19 \pm 0,18$	$0,92 \pm 1,38$	$0,35 \pm 0,31$	$0,34 \pm 0,26$	$4,18 \pm 13,40$	$0,38 \pm 0,28$

Note. Les erreurs de translation sont données en voxels et les erreurs de rotation en degrés.

Tab. 8.4 – Statistiques sur les erreurs de recalage de quelques versions multi-températures quadriprocesseur du recuit simulé adaptatif.

les voxels des images, soit dans ce cas biaisée. Notons que par rapport à la version déterministe quadriprocesseur de période une configuration, les versions systoliques présentent le léger surcoût attendu qui résulte du démarrage non simultané de tous les processeurs.

En dernière analyse, on peut résumer les résultats de cette étude en disant que le recuit simulé adaptatif « simplifié » est une méthode d'optimisation globale apte à résoudre le problème du recalage rigide. Sa parallélisation suivant l'approche multi-températures a permis d'obtenir deux algorithmes data-parallèles qui fournissent rapidement un vecteur Θ dans le bassin d'attraction de l'optimum global, vecteur qui doit être ensuite raffiné par une méthode d'optimisation globale si on veut systématiquement une précision sous-voxel.

	Max_{conf} ou Max_{prop}	Processeurs R12000	Erreur quadratique moyenne		Information mutuelle	
Séquentiel	900	1	608,60		841,10	
Multi-temp. déterministe				Accélération		Accélération
	225	4	172,75	3,52	246,10	3,41
s = 1	112	8	97,30	6,25	/	/
	75	12	80,90	7,53	/	/
s = 5	45	4	188,45	3,23	/	/
s = 10	22	4	250,25	2,43	/	/
s = 20	11	4	423,25	1,44	/	/
Multi-temp. probabiliste				Accélération		Accélération
Original	48	4	174,00	3,50	241,45	3,48
	52	8	86,83	7,00	/	/
Modifié	48	4	176,55	3,45	244,55	3,43
	52	8	87,50	6,96	/	/

Tab. 8.5 – Temps de calcul en secondes du recuit simulé adaptatif séquentiel, ainsi que de différentes versions parallèles multi-températures.

8.4 Évolution différentielle

Dans cette section, nous commençons par préciser la définition retenue pour l'opérateur de reproduction. L'étude de l'algorithme séquentiel met en évidence de bonnes performances et nous permet ensuite de sélectionner le modèle de programmation parallèle adéquat, en l'occurrence le modèle data-parallèle. Finalement, après avoir décrit la parallélisation que nous proposons, nous montrons sa validité.

8.4.1 L'opérateur de reproduction choisi

Par rapport à la présentation que nous avons faite de l'évolution différentielle dans la section 3.5, seul le choix de la définition de l'opérateur de reproduction reste à préciser. Parmi les quatre définitions possibles (voir l'équation (3.27)), nous avons retenu, suite à des tests préalables, la quatrième. Ainsi, dans le cas du recalage rigide, un nouvel individu $a'_j = (x'_{j,i})_{i\in\{1,...,6\}} = \left(t_{xj}, t_{yj}, t_{zj}, \theta_{xj}, \theta_{yj}, \theta_{zj}\right) = \Theta_j$ est obtenu comme suit :

$$x'_{j,i} = \begin{cases} x_{j,i} + \gamma \cdot (x_{opt,i} - x_{j,i}) + F \cdot (x_{S,i} - x_{T,i}) & \text{si } (\chi_1 \leq \chi_2) \wedge (\chi_1 \leq i \leq \chi_2) \text{ ou } (\chi_1 > \chi_2) \wedge (1 \leq i \leq \chi_2 \vee \chi_1 \leq i \leq n) \\ x_{j,i} & \text{sinon} \end{cases} \tag{8.22}$$

Rappelons que $a_{opt} = x_{opt}$ correspond à l'optimum courant, que les indices S et $T \in \{1, ..., \mu\}$ sont tirés aléatoirement de façon à être mutuellement différents et vérifient $S \neq j, T \neq j$, et que χ_1, χ_2 sont les deux points de croisement. Enfin γ et F auront la même valeur dans $[0, 1; 1]$.

La figure 8.10 illustre le principe sous-jacent à cette méthode de reproduction dans le cas d'une minimisation (pour des individus comportant deux composantes, soit $a_j = (x_{j,1}, x_{j,2})$; a''_j correspond à l'individu intermédiaire, i.e. sans considérer de croisement) :

- le vecteur différentiel $(x_{min,i} - x_{j,i})$ guide la recherche vers le minimum courant a_{min}. La vitesse à laquelle on se rapproche de ce dernier est contrôlée par l'intermédiaire de γ ;
- tandis que le vecteur $(x_{S,i} - x_{T,i})$ perturbe en quelque sorte le mouvement vers le minimum courant. Cette perturbation est d'autant plus importante que F a une valeur proche de la borne supérieure de son intervalle de définition.

8.4.2 Étude de l'algorithme séquentiel

Une étude du comportement de l'algorithme séquentiel s'impose, d'une part pour vérifier son adéquation à la résolution du problème posé par le recalage rigide, d'autre part pour déterminer la parallélisation la plus adaptée dans ce cas. Suite aux expérimentations, l'algorithme va engendrer le même nombre de populations pour les quatre premiers niveaux de résolution, contre aucune à la résolution la plus fine (la population courante est simplement réévaluée, comme c'est le cas lors du passage d'une résolution à une autre). Si on note t_{max} le nombre maximum de populations engendrées, on a $\frac{t_{max}}{4}$ populations qui sont engendrées lors du traitement de 1 voxel sur 81, puis 1 sur 27, 9 et 3, et un total de $t_{max} + 5$ populations qui sont évaluées (il faut ajouter l'évaluation de $P(0)$ et la réévaluation nécessaire lors de chaque changement de résolution).

Dans un premier temps nous allons étudier l'influence de la taille de la population lors de l'optimisation de l'erreur quadratique moyenne sur le jeu de 20 recalages. Nous montrons ainsi

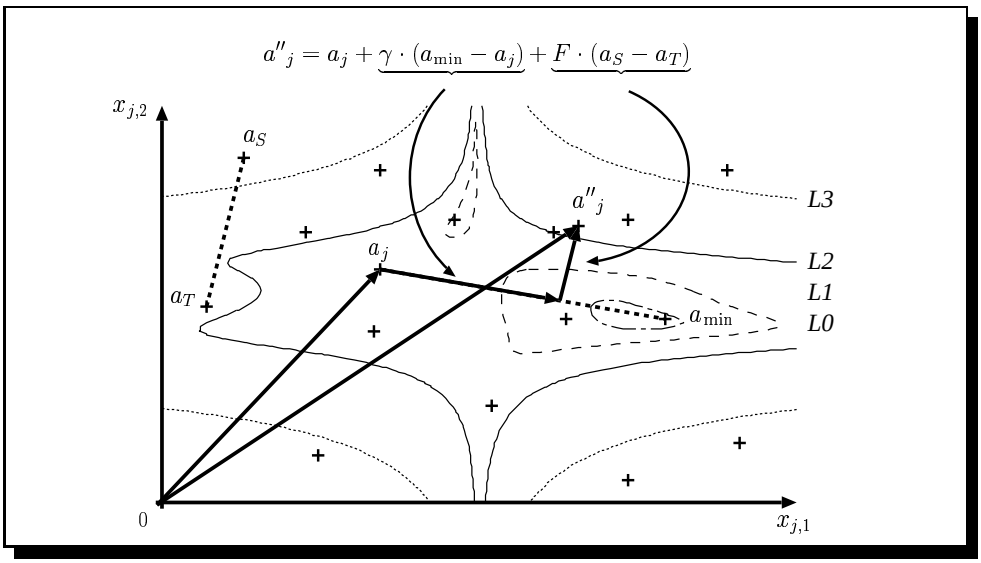


FIG. 8.10 – Illustration de la reproduction choisie dans l'évolution différentielle (les courbes de niveaux du coût vérifient : $L0 < L1 < L2 < L3$).

à la figure 8.11 l'évolution moyenne du coût minimum suivant le nombre d'individus évalués, pour plusieurs tailles de population (4, 8, 16, 32, 64 et 128 individus) et différentes valeurs de la probabilité de croisement ((a) $p_c = 0,30$ et (b) $p_c = 0,80$). Avec, dans les deux cas γ et F qui valent $0,525$ (l'impact de la valeur de ces deux paramètres de l'algorithme sera mis en évidence plus loin). Il apparaît clairement, que dans le contexte du recalage rigide, une population de taille réduite permet à l'évolution différentielle de converger rapidement en obtenant d'excellents résultats. En effet, dans le cas (a) une taille de 8 individus donne une convergence optimale, tandis que pour (b) c'est une taille de 16 individus. Remarquons que ces tailles de populations aboutissent à des courbes de convergence quasi-identiques (figure 8.11(c)), et que ceci est également le cas pour l'information mutuelle (voir le graphique 8.11(d)). De plus, comme l'atteste le tableau 8.6, la qualité du recalage induit par les deux tailles de population est très bonne. Les erreurs de rotation sont en particulier très faibles. D'ailleurs, pour les deux fonctions de coût on a une précision sous-voxel, puisque l'EQM est en moyenne de l'ordre de 0,33 voxels. Un autre aspect à noter est l'impact de la probabilité de croisement sur la convergence pour une même taille de population. De fait, pour une population très petite (4 et 8 individus) une valeur trop grande pour p_c se traduit par une convergence prématurée vers un optimum local. En effet, dans ce cas précis, une probabilité de croisement proche de un se traduit par la perte trop rapide des caractéristiques des parents lors de la reproduction. Enfin, il est évident que les populations contenant un nombre important d'individus nécessitent d'engendrer de nombreuses populations pour converger.

En conclusion, les tailles de population qui induisent une convergence optimale ne sont pas suffisamment importantes pour justifier une approche par division en « îlots ». C'est pourquoi nous avons décidé de nous orienter vers le data-parallélisme.

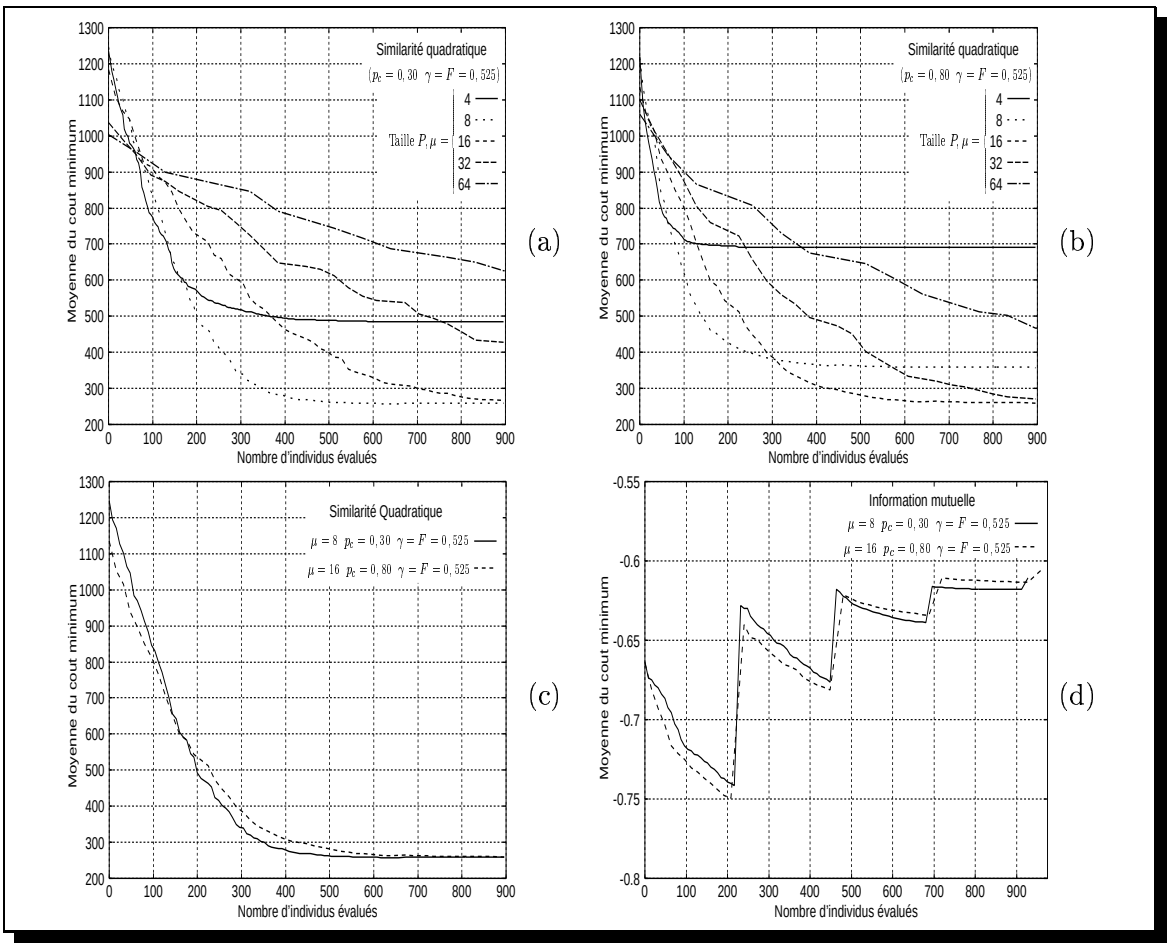


FIG. 8.11 – Évolution différentielle séquentielle - évolution moyenne du coût minimum : dans le cas de l'erreur quadratique moyenne pour différentes tailles de population ((a) $p_c = 0,30$ et (b) $p_c = 0,80$; $\gamma = F = 0,525$) ; courbes de convergence optimale pour l'erreur quadratique moyenne (c) et l'information mutuelle (d).

	Erreur quadratique moyenne		Information mutuelle	
$\mu; p_c; t_{max}$	$8; 0,30; 108$	$16; 0,80; 52$	$8; 0,30; 112$	$16; 0,80; 56$
Δt_x	$0,19 \pm 0,15$	$0,20 \pm 0,15$	$0,19 \pm 0,15$	$0,23 \pm 0,17$
Δt_y	$0,14 \pm 0,10$	$0,15 \pm 0,10$	$0,14 \pm 0,10$	$0,17 \pm 0,14$
Δt_z	$0,16 \pm 0,09$	$0,16 \pm 0,09$	$0,17 \pm 0,09$	$0,16 \pm 0,09$
$\Delta\theta_x$	$0,005 \pm 0,004$	$0,030 \pm 0,040$	$0,020 \pm 0,020$	$0,040 \pm 0,050$
$\Delta\theta_y$	$0,004 \pm 0,003$	$0,020 \pm 0,030$	$0,020 \pm 0,020$	$0,040 \pm 0,040$
$\Delta\theta_z$	$0,009 \pm 0,006$	$0,040 \pm 0,080$	$0,030 \pm 0,030$	$0,070 \pm 0,070$

Note. Les erreurs de translation sont données en voxels et les erreurs de rotation en degrés.

TAB. 8.6 – Statistiques sur les erreurs de recalage de l'évolution différentielle séquentielle.

8.4.3 Parallélisation de l'évolution différentielle

Considérons une population de 16 individus et une grille de processeurs virtuels de taille 4×4, que l'on supposera bouclée sur elle-même. On distribue les différentes populations de l'algorithme (population des parents et population des descendants) de manière à n'avoir qu'un seul individu d'une population par processeur virtuel (on a donc un parent et un descendant par processeur). Cette approche data-parallèle induit la parallélisation de la phase d'initialisation, de l'évaluation d'une population et des opérateurs de reproduction et de sélection.

L'initialisation de la population peut se faire aisément en parallèle, puisque celle-ci se fait aléatoirement en tirant les individus dans l'espace de recherche dont les bornes sont données. De même, la phase d'évaluation est directement parallélisable sans problème notable car elle revient à calculer le coût associé à un individu. La sélection telle qu'elle est définie (équation (3.28)) est également bien adaptée à la parallélisation, car elle consiste à faire une comparaison entre un individu et son descendant. Or, chaque descendant $a'_j, j \in \{1, ..., \mu\}$ avec $\mu = 4 \times 4$ est localisé sur le même processeur que son parent a_j, la comparaison se fait donc localement sur un processeur sans aucune communication. L'opérateur de reproduction décrit par (8.22) implique en revanche des communications. En effet, pour engendrer un nouvel individu a'_j il est nécessaire de connaître l'individu a_{min} de coût minimum dans la population courante, ainsi que de tirer aléatoirement deux individus (a_S et a_T) mutuellement différents, mais également différents de l'individu a_j considéré.

Naturellement, la diffusion de a_{min} sur la grille de processeurs virtuels se traduit *a priori* par des communications irrégulières dont le coût exact dépend de la machine parallèle. De plus, lorsqu'un processeur physique implémente plusieurs processeurs virtuels, les communications entre ceux-ci disparaissent, mais au prix d'une séquentialisation des tâches exécutées par les processeurs virtuels. D'autre part, la diffusion étant une opération de base en data-parallélisme, on peut penser que le compilateur devrait pouvoir optimiser cette opération suivant l'architecture de la machine cible.

Afin de pouvoir choisir en parallèle les individus a_S et a_T, nous associons à chaque processeur virtuel un voisinage défini comme étant formé par ses huit-voisins immédiats sur la grille : a_S et a_T correspondent alors aux individus courants de deux processeurs distincts tirés dans ce voisinage (figure 8.12). La position de ces deux processeurs dans le voisinage est tirée pour l'ensemble de la population, lors de chaque phase de reproduction, ce qui permet d'avoir des communications

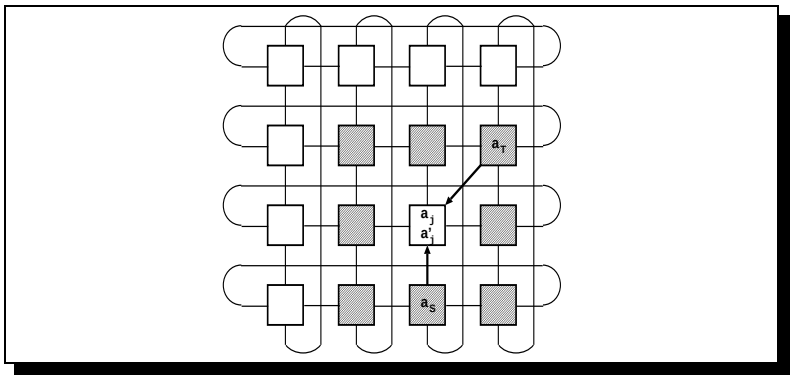


Fig. 8.12 – Exemple de transfert de a_S et a_T (le voisinage est en gris).

régulières. Néanmoins, dans les architectures actuelles, la définition d'un voisinage virtuel ne semble plus aussi important que dans le passé. Il est évident que l'introduction d'un voisinage pour tirer les individus a_S et a_T fait que l'algorithme parallèle n'est pas sémantiquement équivalent à la version séquentielle de l'évolution différentielle. Dans un cas le choix des deux individus se fait parmi huit individus (version parallèle), tandis que dans l'autre cas quinze individus sont potentiellement éligibles. On pourrait donc s'attendre à obtenir des résultats différents, mais comme nous le verrons dans le paragraphe suivant, ce n'est pas le cas. Notons que par nature, l'algorithme parallèle est fortement synchronisé.

En résumé, l'algorithme data-parallèle que nous venons de présenter peut être schématisé par la figure ci-dessous.

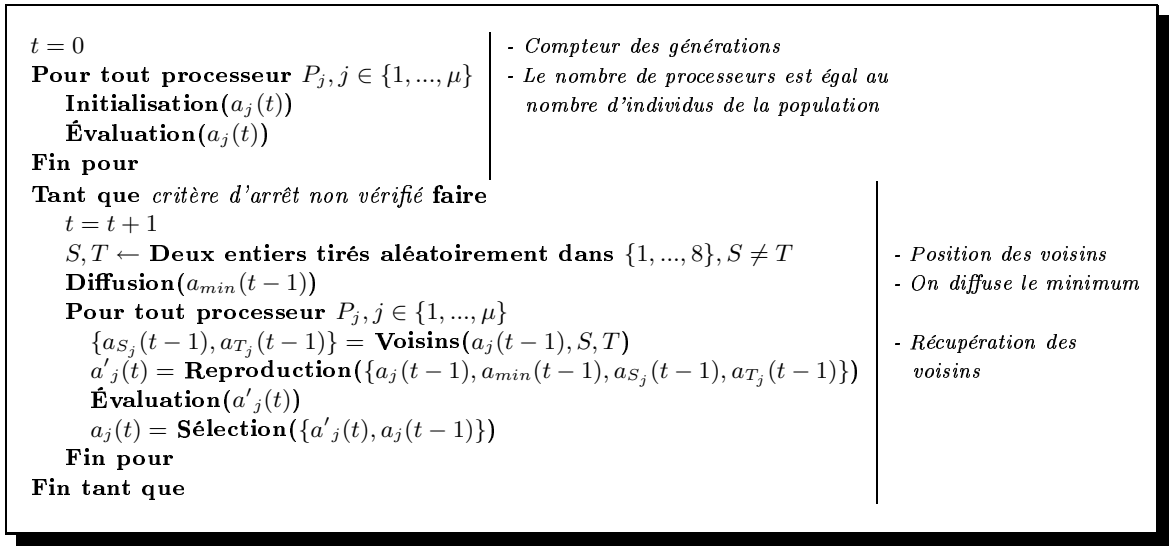


FIG. 8.13 – Schéma de l'évolution différentielle parallèle.

8.4.4 Performances et analyse de l'algorithme parallèle

Le choix des différents paramètres de l'évolution différentielle s'est fait d'une part sur la base des règles établies par Storn [119], et s'appuie d'autre part sur une étude empirique de différents jeux de paramètres. Les valeurs retenues pour γ, F et p_c sont : $\gamma = F = 0,525$; $p_c = 0,80$. Pour ce qui est de la population initiale, celle-ci est générée aléatoirement en tirant les paramètres de T_Θ suivant une distribution uniforme dans l'espace de recherche. Enfin, le nombre maximum de populations engendrées (t_{max}), qui constitue le critère d'arrêt de l'algorithme, a été fixé de façon à évaluer un peu plus de 900 individus. La population sera supposée comporter dans un premier temps 16 individus. Nous fixons t_{max} à 52 pour l'erreur quadratique moyenne, et à 56 pour l'information mutuelle.

Qualité des recalages

Les figures 8.14(a) et 8.14(c) présentent, chacune pour une fonction de coût, l'évolution moyenne du coût minimum sur les 20 recalages en fonction du nombre d'individus évalués. Les

courbes en trait continu correspondent à la version parallèle exécutée sur 8 processeurs physiques, tandis que les courbes discontinues décrivent le comportement de la version séquentielle. La comparaison des courbes met clairement en évidence la pertinence de notre approche. En effet, la parallélisation n'a pas d'impact significatif sur le résultat du recalage puisqu'elle permet d'aboutir à un vecteur de paramètres Θ de coût équivalent à celui de la version séquentielle. En plus de la variation du coût minimum, les figures 8.14(b) et 8.14(d) montrent la variation du coût réel (i.e. moyen et maximum) de la version parallèle. On voit ainsi que l'ensemble de la population converge très régulièrement. Une conséquence directe de la similitude des versions séquentielle et parallèle au niveau de la convergence est la conservation de la précision sous-voxel du recalage. En effet, si on compare les tableaux 8.6 et 8.7, on voit aisément que les résultats sont équivalents. L'écart quadratique moyen pour chacune des fonctions de coût dans le cas parallèle est systématiquement inférieur à $0,50$ voxels.

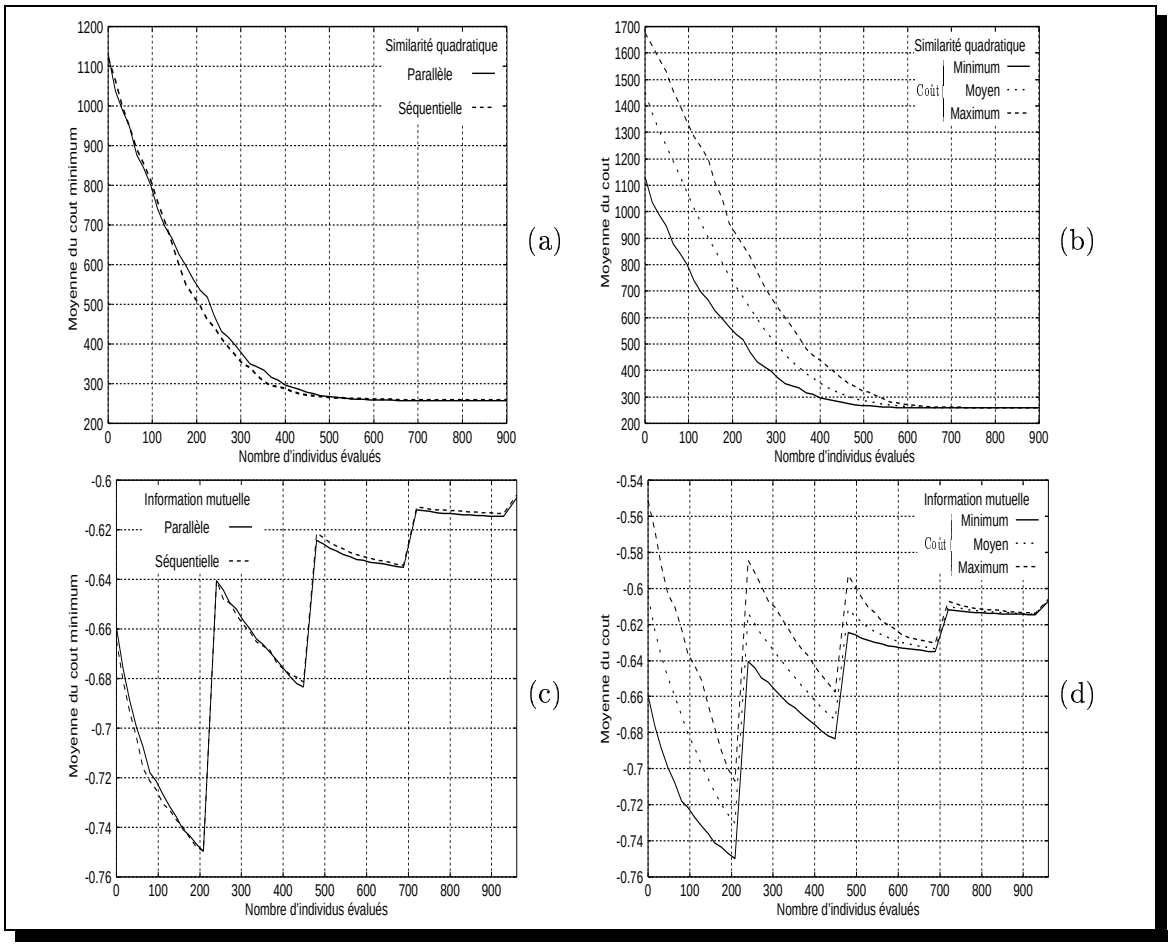


Fig. 8.14 – Évolution différentielle - évolution moyenne sur les 20 recalages du coût (a) minimum, (b) minimum, moyen et maximum, dans le cas de l'erreur quadratique moyenne ; (c) et (d) sont les courbes respectives pour l'information mutuelle.

	Erreur quadratique moyenne	Information mutuelle
$\mu; p_c; t_{max}$	$16; 0,80; 52$	$16; 0,80; 56$
Δt_x	$0,20 \pm 0,14$	$0,20 \pm 0,15$
Δt_y	$0,14 \pm 0,10$	$0,14 \pm 0,11$
Δt_z	$0,16 \pm 0,09$	$0,16 \pm 0,10$
$\Delta\theta_x$	$0,02 \pm 0,02$	$0,02 \pm 0,02$
$\Delta\theta_y$	$0,03 \pm 0,04$	$0,02 \pm 0,02$
$\Delta\theta_z$	$0,04 \pm 0,03$	$0,05 \pm 0,04$

Note. Les erreurs de translation sont données en voxels et les erreurs de rotation en degrés.

TAB. 8.7 – Statistiques sur les erreurs de recalage de l'évolution différentielle parallèle exécutée sur 8 processeurs.

L'excellent comportement de notre algorithme data-parallèle résulte en grande partie du voisinage choisi pour l'opérateur de reproduction, puisque c'est au niveau de cet opérateur que se différencie la version parallèle de la version séquentielle. Dans le cas de l'évolution différentielle séquentielle une taille de population de 9 individus est suffisante pour converger vers un vecteur de paramètres Θ proche de l'optimal (d'après l'étude de l'influence de la taille de la population, 8 individus suffisent). Or, en définissant un huit-voisinage autour de chaque processeur virtuel lors de la phase de reproduction, on a justement une sous-population de 9 individus qui est considérée localement pour la reproduction.

Robustesse de l'algorithme

Différents jeux de paramètres ont été testés : p_c a été successivement fixée à $0,30$ et $0,80$, et dans chaque cas la valeur de γ et F a été échantillonnée régulièrement dans $[0,1;1]$. Les courbes de l'évolution moyenne du coût minimum présentées à la figure 8.15 pour chaque jeu de paramètres montrent que seules les valeurs pour γ et F proches des bornes de l'intervalle donnent des résultats défavorables. Il apparaît également que la valeur optimale pour ces deux paramètres est identique pour les deux probabilités de croisement. On remarque aussi que les deux valeurs pour p_c permettent d'aboutir à un recalage de précision sous-voxel ($p_c = 0,80$ a cependant une meilleure vitesse de convergence et une meilleure précision). On peut donc conclure que contrairement à d'autres algorithmes d'optimisation, l'évolution différentielle est un algorithme robuste.

Temps de calcul

L'algorithme parallèle a été exécuté sur 2, 4, 8, 16 et 32 processeurs avec différentes tailles de population. En effet, comme le montre l'extrait de programme HPF ci-dessous, nous avons utilisé une distribution par bloc des différents tableaux de données, sans spécifier de taille.

```
      DOUBLE PRECISION, DIMENSION(4,4,6) : : POP ! Population de taille 4 × 4
      DOUBLE PRECISION, DIMENSION(4,4) : : EVALUATION ! Evaluations associées à pop
      INTEGER, DIMENSION(2) : : POS_AMIN ! Position du minimum dans pop
      DOUBLE PRECISION, DIMENSION(6) : : AMIN ! L'individu a_min
      ...
!HPF$ DISTRIBUTE POP(BLOCK,BLOCK,*) ! Placement des données
!HPF$ DISTRIBUTE EVALUATION(BLOCK,BLOCK)
      ...
      POS_AMIN=MINLOC(EVALUATION) ; AMIN=POP(POS_AMIN(1),POS_AMIN(2), :)
```

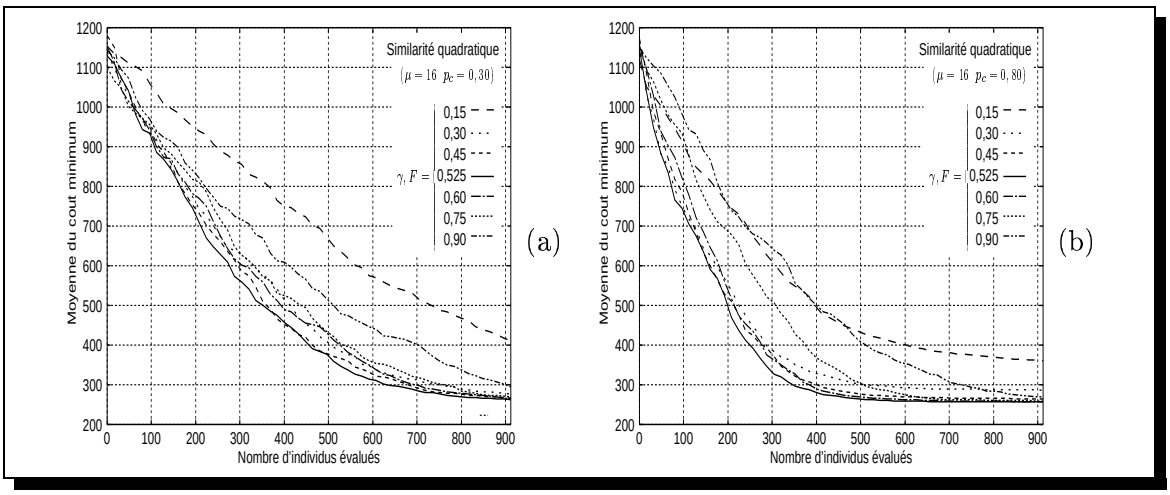


FIG. 8.15 – Évolution différentielle parallèle - évolution moyenne du coût minimum en fonction de γ et F ($\gamma = F$) : (a) $p_c = 0,30$; (b) $p_c = 0,80$.

Par conséquent, le nombre d'individus et donc le nombre de processeurs virtuels projetés sur un processeur physique est calculé par le compilateur. Supposons que l'on ait une grille de $x \times y$ processeurs virtuels et une grille de $p_x \times p_y$ processeurs physiques. Dans ce cas, le compilateur va placer $\lceil \frac{x}{p_x} \rceil \times \lceil \frac{y}{p_y} \rceil$ processeurs virtuels sur un même processeur physique. Ce placement est effectué jusqu'à ce que tous les processeurs virtuels soient associés à un processeur physique. En pratique pour une grille de 4×4 individus, si on utilise une grille de 4×3 processeurs physiques, on aura des blocs de 2×2 individus par processeur, soit le même placement que pour 4×2 processeurs. Il faut donc impérativement choisir un nombre de processeurs physiques qui soit un diviseur du nombre de processeurs virtuels se traduisant par un reste nul. Soit pour 4×4 individus, $2, 4, 8$ et 16 processeurs physiques. Cette contrainte reste valide pour toutes les versions data-parallèles d'un algorithme évolutionnaires écrites en HPF.

Le tableau 8.8 montre pour l'erreur quadratique moyenne la décroissance des temps d'exécution lors du passage de la version séquentielle aux différentes versions parallèles. On constate qu'en passant de l'évolution différentielle séquentielle à l'algorithme parallèle et qu'en accroissant le nombre de processeurs physiques on obtient une réduction régulière du temps d'exécution. Pour une de population de 16 individus et des images de 128^3 voxels, la version séquentielle converge en 11 minutes 40 secondes alors que la parallélisation sur 16 processeurs a un temps de calcul d'approximativement 52 secondes. Dans le cas d'images 3D de taille 256^3 voxels ($t_{max} = 40$), ces temps de calcul sont respectivement de 1 heure 51 minutes et 8 minutes 32 secondes. Notons que l'on observe une réduction identique du temps de calcul pour l'information mutuelle, puisqu'en moyenne l'algorithme séquentiel requiert 16 minutes 30 secondes contre un peu plus de 2 minutes 15 secondes sur 8 processeurs (IRM 128^3 voxels). Pour des tailles de population nettement plus grandes, le temps d'exécution séquentiel explose. En effet, plusieurs milliers d'individus doivent être engendrés pour assurer la convergence. Pour 128 individus on a ainsi un temps de calcul de presque 2 heures et demi, ce qui aboutit sur 8 processeurs à un temps d'exécution pratiquement équivalent à celui de la version séquentielle pour une population de 32 individus.

Processeurs R12000		1	2	4	8	16	
Individus	t_{max}						
16	52	700,80	349,41	184,47	95,54	52,16	
Accélération			2,01	3,80	7,34	13,44	
Processeurs R10000		1	2	4	8	16	32
32	24	1207,57	609,27	324,91	166,06	88,47	49,02
Accélération			1,98	3,72	7,27	13,65	24,63
128	52	8585,35	4318,15	2281,88	1158,84	/	/
Accélération			1,99	3,76	7,41		

Tab. 8.8 – Temps de calcul (en secondes) de l'évolution différentielle dans le cas de l'erreur quadratique moyenne, suivant la taille de la population, le nombre maximum de populations engendrées (t_{max}) et le nombre de processeurs.

En étudiant la courbe des facteurs d'accélération pour des populations de 16 individus et 32 individus (figures 8.16(a) et 8.16(b)) on remarque une dégradation des performances en augmentant le nombre de processeurs : sur 2 processeurs l'accélération est quasi-linéaire, puis la courbe s'écarte de plus en plus de la courbe idéale de l'accélération linéaire. En dépit de cette dégradation, on a cependant une accélération excellente sur 16 processeurs, puisqu'elle est de 13,44 pour 16 individus et de 13,65 pour 32 individus. À souligner que la taille des images 3D n'a pas d'impact significatif sur les performances (une accélération de 13 sur 16 processeurs, pour une population de 16 individus).

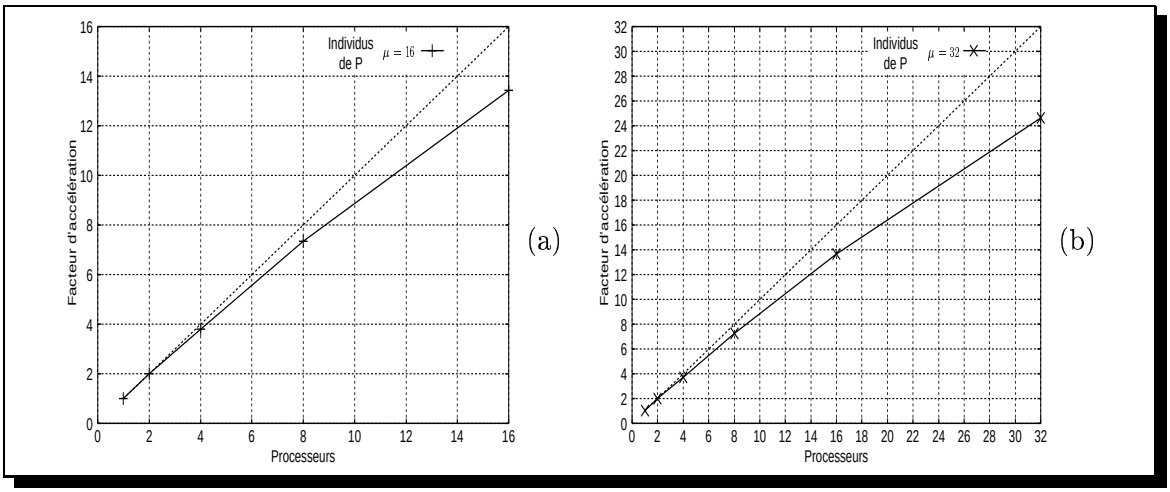


Fig. 8.16 – Courbe des facteurs d'accélération de l'évolution différentielle parallèle, dans le cas de l'erreur quadratique moyenne, pour une taille de population de (a) 16 individus (processeurs R12000) et (b) 32 individus (processeurs R10000).

La perte de performance que l'on observe résulte en grande partie de l'accroissement des communications irrégulières entre processeurs physiques. Ces communications, nécessaires pour

diffuser a_{min}, sont coûteuses même si cette opération est en « théorie » optimisée par le compilateur. Ce problème peut s'avérer gênant dans le cas d'applications requérant une population de grande taille. C'est pourquoi, il y a sans doute une borne sur la taille de la population au-delà de laquelle une approche par division en sous-populations est préférable, et inversement.

Le second point qui peut poser problème est la longueur de croisement L qui est variable d'un processeur à un autre. En conséquence, l'opérateur de reproduction n'a pas le même coût sur l'ensemble de la grille de processeurs. Ceci peut se traduire dans le cas d'individus ayant beaucoup de composantes par un déséquilibre de la charge. Pour l'application de recalage rigide que nous considérons dans ce chapitre, ce problème ne s'est cependant pas manifesté.

8.5 Stratégies d'évolution

À l'image de l'évolution différentielle, la mise en œuvre séquentielle des stratégies d'évolution va nous permettre d'apprécier le comportement de ce type d'algorithmes évolutionnaires et de déterminer le modèle de programmation parallèle le plus approprié dans le cas du recalage rigide. Nous verrons en particulier que l'approche data-parallèle proposée pour l'évolution différentielle reste relativement valable pour les stratégies d'évolution élitistes, i.e. celles faisant la sélection en tenant compte des parents et des descendants, notées SE($\mu + \lambda$) (voir le troisième paragraphe de la section 3.4.2). Mais avant toute chose, quelques mots sur la représentation retenue pour les individus et les opérateurs de reproduction.

8.5.1 Représentation et opérateurs de reproduction

Lors de l'état de l'art (section 3.4), nous avons vu que les stratégies d'évolution manipulent directement les variables continues que l'on cherche à optimiser et, qu'à côté de celles-ci, chaque individu intègre des paramètres supplémentaires utilisés dans la phase de mutation : les variances (en fait les écarts types) et les covariances d'une distribution multinormale. Le choix du nombre de paramètres doit se faire en gardant en tête deux remarques :

- le nombre de degrés de liberté de la mutation croît avec le nombre de paramètres ;
- plus on a de paramètres, plus la phase reproduction est coûteuse, puisque ces derniers subissent également le processus d'évolution.

Dans le cas du recalage rigide, un bon compromis consiste à choisir autant d'écarts types que de variables à optimiser et aucune covariance, i.e. de ne pas corréler les mutations. En effet, cela permet d'avoir des mutations propres à chacune des variables et ce n'est pas très coûteux. De toute manière un écart type pour les variables de translation et un autre pour les angles de rotation est le minimum requis, car pour une même valeur numérique, la translation et la rotation n'ont pas le même impact sur la fonction de coût. Quand aux covariances, il faut avouer que le surcoût calculatoire qu'elles impliquent, fait qu'en pratique elles sont très peu utilisées. Les individus sont donc en l'espèce de la forme :

$$a_j = (x_{j,i}, \sigma_{j,i})_{i \in \{1,...,6\}} = \left(t_{xj}, t_{yj}, t_{zj}, \theta_{xj}, \theta_{yj}, \theta_{zj}, \sigma_{t_x,j}, \sigma_{t_y,j}, \sigma_{t_z,j}, \sigma_{\theta_x,j}, \sigma_{\theta_y,j}, \sigma_{\theta_z,j}\right) = (\Theta_j, \Sigma_j) \tag{8.23}$$

avec $j \in \{1, ..., \mu\}$ ou $j \in \{1, ..., \lambda\}$, suivant que l'on considère la population des parents ou des descendants.

Pour la recombinaison, nous utilisons une approche différente pour les variables à optimiser et les paramètres : une recombinaison intermédiaire pour le vecteur Θ_j ; une recombinaison multi-partenaire généralisée intermédiaire pour le vecteur Σ_j. À partir de (3.9), l'obtention d'un descendant $a'_j = (x'_{j,i}, \sigma'_{j,i})_{i \in \{1,...,6\}}$ peut se formaliser de la manière suivante :

$$x'_{j,i} = x_{S,i} + (x_{T,i} - x_{S,i})/2 \text{ et } \sigma'_{j,i} = \sigma_{S,i} + \chi_i \cdot (\sigma_{T_i,i} - \sigma_{S,i}) \tag{8.24}$$

où les indices S, T et $T_i \in \{1, ..., \mu\}$, qui sont tirés aléatoirement, définissent les individus qui sont recombinés pour engendrer le nouvel individu ; χ_i est un réel tiré suivant une distribution uniforme dans $[0; 1]$. Le vecteur de variables Θ_j correspond donc à la moyenne arithmétique de deux individus, tandis que Σ_j est engendré en combinant des couples d'individus dont l'un des partenaires évolue, de même que le facteur de pondération χ_i.

Finalement, la mutation telle qu'elle est définie dans (3.10), nécessite en l'absence de covariances, uniquement la donnée d'une valeur pour chacun des facteurs τ' et τ. Ceux-ci sont fixés en utilisant les valeurs suggérées par Schwefel [114].

8.5.2 Étude de l'algorithme séquentiel

Le principal objectif de cette étude est d'estimer la capacité des stratégies d'évolution à résoudre le problème d'optimisation posé par le recalage rigide, et en cas de bonnes performances, de trouver la parallélisation qui soit la plus appropriée. Naturellement, des réponses à des questions telles que le choix d'une stratégie élitiste ou non sont aussi recherchées.

Précisons tout d'abord que le traitement multirésolution des images se fera à un détail près, de la même manière que dans l'évolution différentielle. De fait, pour un maximum de t_{max} populations engendrées, les stratégies d'évolution produiront lors du traitement de chacun des quatre premiers niveaux de résolution $\frac{t_{max}}{4}$ populations. En revanche à la résolution finale, et contrairement à l'évolution différentielle qui ne fait que réévaluer la population, les stratégies d'évolution pourront éventuellement, suivant la valeur du reste de la division de t_{max} par quatre, engendrer un certain nombre de populations. Dans l'étude que nous présentons, t_{max} a été choisi de façon à ce que chaque instance d'une stratégie d'évolution évalue un peu plus de 900 individus.

De même que pour l'évolution différentielle, l'étude de l'influence de la taille de la population des parents (μ) et de celle des descendants (λ) est nécessaire pour identifier le modèle de programmation parallèle le plus adéquat. Simultanément, nous étudions l'impact de la valeur initiale pour les écarts types des angles de rotation sur la convergence de l'algorithme en considérant quatre valeurs (en radians) qui correspondent approximativement à $1°, 2°, 5°$ et $10°$. Les écarts types associés à la translation sont quand à eux systématiquement fixés à un, puisque de petites mutations sont largement suffisantes en raison du pré-centrage de l'espace de recherche de la translation par recalage des barycentres des images.

Le premier groupe de graphiques que nous présentons, figure 8.17, montre le comportement des stratégies d'évolution élitistes et non élitistes pour différents couples de taille de population (μ, λ) vérifiant $\frac{\lambda}{\mu} = 5$ (cette contrainte est totalement arbitraire). Les graphiques 8.17(a) à 8.17(d) présentent ainsi dans le cas de l'optimisation de l'erreur quadratique moyenne sur 20 recalages rigides, l'évolution moyenne du coût minimum suivant le nombre d'individus évalués, induit par les stratégies SE$(2 + 10)$, SE$(2, 10)$, SE$(4 + 20)$, SE$(4, 20)$, SE$(8 + 40)$ et SE$(8, 40)$. Chaque graphique est associé à une valeur initiale différente pour les $\sigma_{\theta_x,j}, \sigma_{\theta_y,j}$ et $\sigma_{\theta_z,j}$.

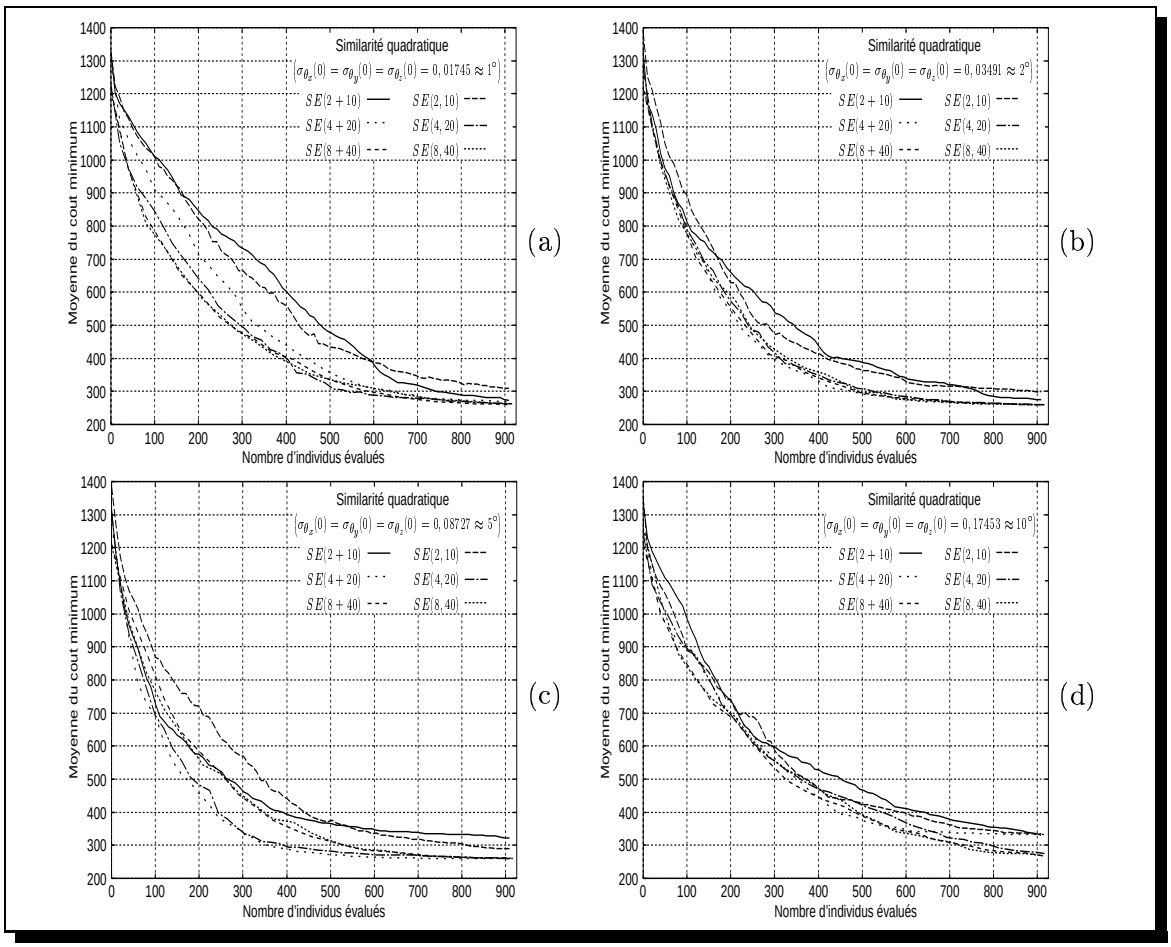


FIG. 8.17 – Stratégies d'évolution séquentielles, élitistes et non élitistes - évolution moyenne du coût minimum dans le cas de l'erreur quadratique moyenne pour différentes tailles de population (μ et λ vérifient $\frac{\lambda}{\mu} = 5$) et valeur initiale pour les écarts types des angles de rotation.

Au vu des courbes que l'on obtient, on peut dire qu'une taille réduite pour la population des parents permet à l'algorithme de converger en donnant de très bons résultats. En effet, les meilleures courbes de convergence sont le fait des couples $(4, 20)$ et $(8, 40)$. Concernant la valeur initiale des écarts types des angles de rotation, les deux extrêmes donnent les moins bons résultats : tantôt elle est trop faible, tantôt elle est trop grande. Comme on peut le voir avec les courbes des couples $(8, 40)$, la valeur initiale de $\sigma_{\theta_x,j}, \sigma_{\theta_y,j}$ et $\sigma_{\theta_z,j}$ affecte essentiellement la vitesse de convergence de l'algorithme. D'autre part, on constate qu'aucun des deux types de stratégies (élitiste et non élitiste) ne prend le pas sur l'autre. Pour conclure, les meilleurs résultats sont obtenus par les stratégies SE$(4 + 20)$, SE$(8 + 40)$ et SE$(8, 40)$. D'ailleurs la bonne précision des recalages est claire si on regarde le tableau 8.9. Notons que les deux dernières stratégies sont également les moins sensibles à la valeur initiale des écarts types, ce qui semble signifier que pour obtenir une stratégie d'évolution la plus robuste possible vis-à-vis des écarts types initiaux, il faudrait augmenter les tailles des populations.

$t_{max} = 22$	Erreur quadratique moyenne			
Écart type rotation	0,01745	0,03491	0,08727	0,17453
Δt_x	$0,19 \pm 0,14$	$0,19 \pm 0,14$	$0,21 \pm 0,16$	$0,19 \pm 0,13$
Δt_y	$0,14 \pm 0,10$	$0,15 \pm 0,10$	$0,14 \pm 0,10$	$0,16 \pm 0,11$
Δt_z	$0,16 \pm 0,08$	$0,16 \pm 0,09$	$0,16 \pm 0,09$	$0,15 \pm 0,09$
$\Delta\theta_x$	$0,05 \pm 0,04$	$0,06 \pm 0,06$	$0,05 \pm 0,03$	$0,08 \pm 0,10$
$\Delta\theta_y$	$0,09 \pm 0,08$	$0,04 \pm 0,04$	$0,04 \pm 0,03$	$0,07 \pm 0,06$
$\Delta\theta_z$	$0,06 \pm 0,06$	$0,05 \pm 0,06$	$0,08 \pm 0,10$	$0,18 \pm 0,20$

Note. Les erreurs de translation sont données en voxels et les erreurs de rotation en degrés.

Tab. 8.9 – Statistiques sur les erreurs de recalage de la SE$(8 + 40)$ séquentielle dans le cas de l'erreur quadratique moyenne.

Nous avons ensuite utilisé le même cadre expérimental qu'auparavant, mais en considérant cette fois des stratégies d'évolution élitistes dont les tailles de population sont identiques, i.e. $\mu = \lambda$. De plus, les valeurs retenues pour μ et λ sont égales à celles employées lors de l'étude de la version séquentielle de l'évolution différentielle, à savoir de 4, 8, 16 et 32 individus, de même pour les différentes valeurs pour t_{max} (par exemple pour $\mu = \lambda = 16$ on a $t_{max} = 52$). Cette seconde vague d'expérimentations est évidemment réalisée dans la perspective d'une implantation parallèle des stratégies d'évolution analogue à celle que nous avons proposée pour l'évolution différentielle, c'est-à-dire un schéma de parallélisation consistant à distribuer les populations des parents et des descendants à raison d'un individu par processeur virtuel.

Les figures 8.18(a) à 8.18(d), qui montrent l'évolution moyenne du coût minimum que l'on obtient pour les différentes stratégies et valeurs initiales pour les écarts types, permettent d'envisager positivement une approche pour la parallélisation, similaire à celle de l'évolution différentielle. En particulier, on remarque que sur l'ensemble des quatre graphiques, les meilleures courbes de convergence sont induites par des populations de 8 et 16 individus, soit les mêmes tailles que pour l'évolution différentielle. Néanmoins, pour une taille de population de 4 individus, les deux algorithmes évolutionnaires ont un comportement différent : à l'opposé de l'évolution différentielle, une stratégie d'évolution élitiste ne converge pas toujours prématurément vers un optimum local. En effet, l'opérateur de mutation utilisé dans les stratégies d'évolution permet d'assurer une meilleure diversification des individus. D'autre part, ainsi que nous l'avions fait remarquer à l'issue des premières expérimentations, plus μ et λ sont grands (dans notre cas 32 individus), moins la valeur initiale pour les écarts types des angles de rotation influe sur la vitesse de convergence. Sur le plan de la précision, le tableau 8.10 met en évidence que les images recalées ont une précision satisfaisante. Comparé à l'évolution différentielle (tableau 8.6), on observe que la SE$(16 + 16)$ est tout à fait compétitive.

Lorsque l'on fait le bilan des expérimentations que nous venons d'exposer, on peut légitimement dire que les stratégies d'évolution forment une famille d'algorithmes évolutionnaires adaptée au problème d'optimisation posé par le recalage rigide. De plus, à l'instar de l'évolution différentielle, le data-parallélisme est *a priori* le modèle de programmation parallèle idoine. Par conséquent, la parallélisation que nous allons mettre en avant pour les stratégies d'évolution reprendra les grandes lignes de celle que nous avons proposée pour l'évolution différentielle.

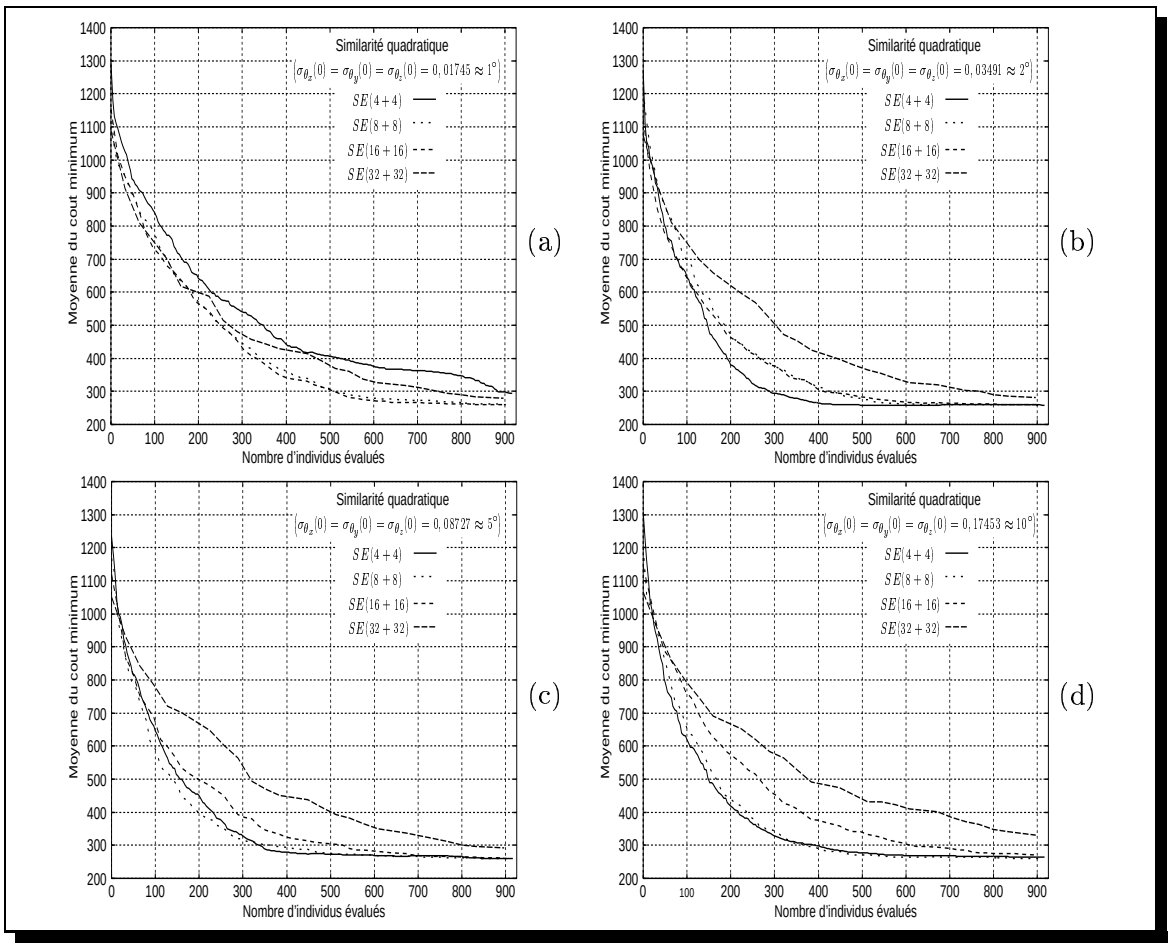


FIG. 8.18 – Stratégies d'évolution élitistes séquentielles - évolution moyenne du coût minimum dans le cas de l'erreur quadratique moyenne pour différentes tailles de populations (μ et λ vérifient $\frac{\lambda}{\mu} = 1$) et valeur initiale pour les écarts types des angles de rotation.

$t_{max} = 52$	Erreur quadratique moyenne			
Écart type rotation	0,01745	0,03491	0,08727	0,17453
Δt_x	$0,19 \pm 0,14$	$0,21 \pm 0,15$	$0,19 \pm 0,15$	$0,18 \pm 0,15$
Δt_y	$0,16 \pm 0,09$	$0,15 \pm 0,10$	$0,15 \pm 0,10$	$0,18 \pm 0,10$
Δt_z	$0,17 \pm 0,10$	$0,16 \pm 0,10$	$0,17 \pm 0,10$	$0,17 \pm 0,09$
$\Delta\theta_x$	$0,03 \pm 0,02$	$0,06 \pm 0,05$	$0,06 \pm 0,04$	$0,08 \pm 0,08$
$\Delta\theta_y$	$0,05 \pm 0,04$	$0,04 \pm 0,03$	$0,07 \pm 0,08$	$0,13 \pm 0,16$
$\Delta\theta_z$	$0,04 \pm 0,03$	$0,05 \pm 0,05$	$0,06 \pm 0,07$	$0,19 \pm 0,25$

Note. Les erreurs de translation sont données en voxels et les erreurs de rotation en degrés.

TAB. 8.10 – Statistiques sur les erreurs de recalage de la SE(16 + 16) séquentielle dans le cas de l'erreur quadratique moyenne.

8.5.3 Parallélisation des stratégies d'évolution

Paralléliser suivant une approche data-parallèle les stratégies d'évolution consiste, comme pour l'évolution différentielle, à distribuer les différentes populations de l'algorithme. Nous distribuons ainsi la population des parents de façon à avoir un individu par processeur virtuel, processeurs supposés être organisés sous la forme d'une grille torique, alors que les descendants sont projetés à raison de $\frac{\lambda}{\mu}$ individus par processeur (on contraint λ à être divisible par μ). Il ne nous reste alors plus qu'à définir des « équivalents » parallèles des différentes étapes de l'algorithme et entre autres des opérateurs de recombinaison, mutation et sélection parallèles. Dans la suite, nous nous plaçons dans le même contexte que lors de la présentation de l'évolution différentielle parallèle : une population de parents de 16 individus et une grille de 4×4 processeurs virtuels.

Parmi les différentes étapes d'une stratégie d'évolution (voir le schéma d'un algorithme évolutionnaire, figure 3.1, et la section 3.4 relative aux SEs), l'initialisation de la population des parents, les différentes phases d'évaluation (au départ des parents, puis des descendants), ainsi que la mutation, sont parallélisables sans difficultés. En effet, aucune communication entre processeurs virtuels n'est requise par ces trois opérations lorsqu'on les applique sur les populations distribuées. Seuls les opérateurs de sélection et de recombinaison impliquent des communications, il s'agit donc de définir des versions parallèles pour ces deux opérateurs qui soient efficaces, tout en ne modifiant pas trop fondamentalement le comportement originel de l'algorithme.

Comme dans l'évolution différentielle, la sélection est déterministe dans les stratégies d'évolution, mais deux définitions sont possibles suivant que l'on considère la version élitiste ou non. Malgré tout, dans les deux cas si on veut un opérateur de sélection qui soit sémantiquement équivalent à celui d'une version séquentielle, une synchronisation globale avec de nombreuses communications irrégulières est nécessaire. En effet, les nouveaux parents correspondent aux μ meilleurs individus parmi, soit les parents courants et les descendants dans le cas élitiste, soit uniquement les descendants. En pratique, un processeur « maître » devra donc recenser au pire tous les individus, sélectionner les μ meilleurs, puis redistribuer ces derniers. Au final, un opérateur de sélection parallèle similaire à la version séquentielle s'avèrera particulièrement inefficace au niveau des communications. Par conséquent, pour obtenir une implantation data-parallèle performante, la définition d'un opérateur de sélection sémantiquement différent s'impose.

L'approche que nous avons choisie consiste à définir une sélection parallèle n'induisant pas de communications. Pour ce faire, le nouveau parent sur un processeur virtuel sera le meilleur individu parmi le parent courant et les $\frac{\lambda}{\mu}$ descendants engendrés sur ce processeur virtuel. En quelque sorte on a μ sélections élitistes du même type que celle d'une $\mathrm{SE}\left(1 + \frac{\lambda}{\mu}\right)$ qui sont exécutées simultanément. Le fait que la sélection sur un processeur virtuel soit élitiste permet de préserver localement le meilleur individu, en revanche globalement on ne conserve pas nécessairement les μ meilleurs individus. De fait, il se peut très bien que sur un processeur, le parent courant et les descendants qui viennent d'être engendrés ne soient nullement dans les μ meilleurs individus lorsque l'on considère les populations de parents et de descendants dans leur globalité. Cette sélection parallèle n'est donc pas vraiment élitiste ou non élitiste, mais plutôt un hybride. Bien entendu, la question qui se pose est sa validité, c'est-à-dire si elle permet d'obtenir une stratégie d'évolution data-parallèle ayant une bonne convergence. À remarquer que le type de la sélection locale servira pour noter les algorithmes data-parallèles. Par exemple $\mathrm{SE}(16 + 16)$ signifiera que l'on a un parent et un descendant par processeur virtuel, avec une sélection locale élitiste.

En ce qui concerne l'opérateur de recombinaison, la distribution des parents à raison d'un individu par processeur virtuel va naturellement induire des communications, puisque comme le montre la définition retenue pour engendrer un nouvel individu $a'_j, j \in \{1, ..., \lambda\}$ (8.24), 8 individus parents sont au maximum nécessaires. Ce dernier chiffre est un maximum, car contrairement à l'évolution différentielle les individus a_S, a_T et a_{T_i} où $i \in \{1, ..., 6\}$, ne doivent pas être obligatoirement différents les uns des autres. Afin de pouvoir engendrer en parallèle un descendant sur chaque processeur virtuel, nous allons utiliser la même approche que dans l'évolution différentielle pour choisir les individus participant à la recombinaison avec un individu parent. On associe donc à chaque processeur virtuel un huit-voisinage (voir la figure 8.12), et on choisit aléatoirement les individus a_S, a_T et a_{T_i} parmi les parents de ce voisinage. Le choix des individus est identique à chaque fois qu'on engendre μ descendants en parallèle, de sorte que les communications seront obligatoirement régulières. Logiquement, le fossé sémantique entre la version séquentielle et la version parallèle des stratégies d'évolution va encore se creuser dès lors que $\mu \geq 10$.

Dans le cas d'une population de parents de taille inférieure à 9 individus, par exemple si on considère la SE$(4 + 20)$ parallèle, un huit-voisinage est toujours défini puisque la grille de processeurs virtuels est torique. Cependant, il y a un problème : certains individus seront plusieurs fois dans le huit-voisinage. Par conséquent, pour ne pas introduire un biais dans le choix des voisins nous allons procéder comme suit. Supposons que les processeurs sont organisés sous forme d'une grille de taille $L \times l$. Le voisinage de chaque parent sera alors de taille $L \times l - 1$, avec le premier voisin qui se trouve sur le processeur à l'est du processeur courant, le second sur celui qui est au nord-est, etc, jusqu'à avoir le nombre de voisins requis. Dès lors, la position possible pour un voisin sera comprise entre 1 (référence l'individu sur le processeur à l'est) et $L \times l - 1$.

La figure 8.19 récapitule l'implantation data-parallèle que nous venons de décrire.

$t = 0$ | *- Compteur des générations*
Pour tout processeur $P_j, j \in \{1, ..., \mu\}$ | *- Le nombre de processeurs est égal au nombre d'individus de la population des parents ; les processeurs sont supposés former une grille torique de taille* $L \times l$
  Initialisation$(a_j(t))$
  Évaluation$(a_j(t))$
Fin pour
Tant que *critère d'arrêt non vérifié* **faire**
  $t = t + 1$
  $S, T, (T_i)_{i \in \{1,...,6\}} \leftarrow$ **Huit entiers tirés aléatoirement dans** $\{1, ..., \min(L + l - 1, 8)\}$
  Pour tout processeur $P_j, j \in \{1, ..., \mu\}$
    Pour k **de** 1 **à** $\frac{\lambda}{\mu}$ **faire**
      $\{a_{S_j}(t-1), a_{T_j}(t-1), a_{T_{1j}}(t-1), ..., a_{T_{6j}}(t-1), \} =$ **Voisins**$\left(a_j(t-1), S, T, (T_i)_{i \in \{1,...,6\}}\right)$
      $a'_{j,k}(t) =$ **Recombinaison**$\left(\left\{a_{S_j}(t-1), a_{T_j}(t-1), \left(a_{T_{ij}}(t-1)\right)_{i \in \{1,...,6\}}\right\}\right)$
      Mutation$(a'_{j,k}(t))$ | *- $a'_{j,k}$ est le k-ième descendant sur le processeur P_j*
      Évaluation$(a'_{j,k}(t))$
    Fin pour
    $a_j(t) =$ **Sélection**$\left(\left\{a_j(t-1), a'_{j,1}(t), ..., a'_{j,\frac{\lambda}{\mu}}(t), \right\}\right)$
  Fin pour
Fin tant que

Fig. 8.19 – Schéma d'une stratégie d'évolution parallèle.

8.5.4 Performances et analyse de l'algorithme parallèle

Pour valider l'algorithme parallèle, nous allons comparer chacune des stratégies d'évolution parallèles suivantes : SE(4+20), SE(8+40) et SE(16+16), à sa version élitiste séquentielle correspondante. Les comparaisons sont faites sur les deux fonctions de coût, en prenant à chaque fois les mêmes quatre valeurs initiales pour les écarts types des angles de rotation. En ce qui concerne le nombre total de populations engendrées, t_{max}, celui-ci est respectivement fixé à 45, 22 et 52 dans le cas l'erreur quadratique moyenne ; il est égal à 49, 24 et 56 lorsque l'on recale les images en utilisant l'information mutuelle. Toutes les versions parallèles ont été exécutées sur autant de processeurs physiques que de processeurs virtuels requis, soit 4, 8 et 16 processeurs physiques.

Qualité des recalages

Les graphiques 8.20(a) à 8.20(d) présentent l'évolution moyenne du coût minimum sur le jeu de 20 recalages suivant le nombre d'individus évalués, lorsque le recalage est fait en prenant comme fonction de coût l'erreur quadratique moyenne. Globalement, on constate que les parallélisations donnent en général des courbes de convergence moins bonnes et surtout moins rapides que les versions séquentielles. Ceci s'explique aisément par le fait que les algorithmes parallèles ne sont que localement élitistes. Néanmoins, pour une population de parents de 4 individus on voit que lorsque les écarts types sont fixés initialement à 10°, la version parallèle se comporte mieux. En effet, comme cette dernière n'est pas élitiste, elle assure une meilleure diversité de la population en préservant parfois localement de « mauvais » individus. Cela permet d'éviter la convergence prématurée vers un optimum local. D'autre part, bien que les algorithmes parallèles convergent moins rapidement, ils aboutissent néanmoins à des vecteurs Θ qui sont proches de l'optimum. Ceci est très clair lorsque l'on compare par exemple les statistiques sur les erreurs de recalage de la SE(16 + 16) parallèle (tableau 8.11), avec celles de la SE(16 + 16) séquentielle (tableau 8.10). Pour certaines valeurs initiales de $\sigma_{\theta_x,j}, \sigma_{\theta_y,j}$ et $\sigma_{\theta_z,j}, j \in \{1, ..., \mu\}$, les stratégies d'évolution parallèles SE(4 + 20) et SE(16 + 16) produisent un recalage de précision sous-voxel. En fait, pour obtenir un recalage de bonne qualité indépendamment du choix de la valeur initiale pour les écarts types des angles de rotation et des tailles de population, il faudrait raffiner le résultat avec une méthode d'optimisation locale. Une alternative est d'accroître la valeur de t_{max}, car ainsi qu'on peut le voir sur la figure 8.22, l'algorithme data-parallèle nécessite plus de générations que le séquentiel pour converger, toujours en raison de son non élitisme « global ».

Lorsque l'on étudie les courbes que l'on obtient pour l'information mutuelle, figure 8.21, les remarques précédentes apparaissent plus clairement. En effet, à tailles de populations identiques, la version séquentielle est plus compétitive que la parallélisation. De plus, les différences de qualité entre les recalages obtenus en séquentiel et ceux en parallèle sont du même ordre que pour l'erreur quadratique moyenne (tableau 8.12). Et comme pour l'erreur quadratique moyenne, certaines stratégies d'évolution parallèles donnent suivant l'initialisation des paramètres un recalage de précision sous-voxel. Notons que c'est les cas de la SE(16 + 16) parallèle : pour des écarts types initiaux de 1° pour la rotation, l'écart quadratique moyen est égal à $0,43 \pm 0,16$ voxels.

À l'issue des expérimentations, on peut dire que la parallélisation que nous avons décrite dans la section précédente est relativement valable pour le recalage rigide. En effet, il est tout à fait possible d'obtenir des recalages de précision sous-voxel. Cependant, comme dans le cas séquentiel, le choix des tailles de population, de même que des valeurs initiales pour les écarts types n'est pas aisé. Aussi, si on veut éviter l'ajustement fastidieux des divers paramètres de l'algorithme parallèle, le raffinement de la solution finale ou l'augmentation du nombre de populations engendrées par rapport à la version séquentielle doivent être envisagées.

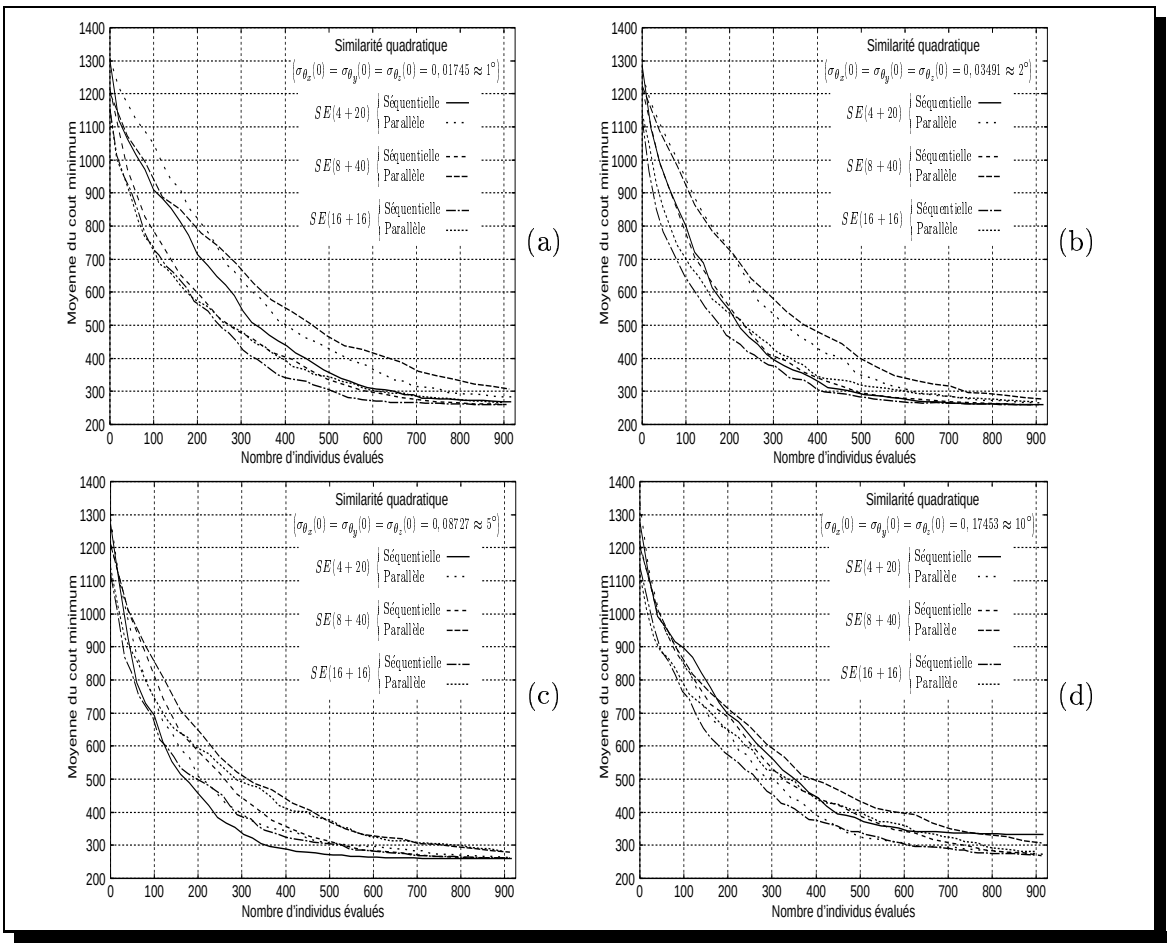


FIG. 8.20 – Stratégies d'évolution séquentielles et parallèles - évolution moyenne du coût minimum dans le cas de l'erreur quadratique moyenne pour différentes tailles de populations et valeur initiale pour les écarts types des angles de rotation.

$t_{max} = 52$	Erreur quadratique moyenne			
Écart type rotation	0,01745	0,03491	0,08727	0,17453
Δt_x	$0,19 \pm 0,14$	$0,23 \pm 0,16$	$0,21 \pm 0,17$	$0,20 \pm 0,17$
Δt_y	$0,14 \pm 0,10$	$0,14 \pm 0,08$	$0,21 \pm 0,12$	$0,12 \pm 0,10$
Δt_z	$0,18 \pm 0,10$	$0,16 \pm 0,08$	$0,21 \pm 0,12$	$0,18 \pm 0,11$
$\Delta\theta_x$	$0,08 \pm 0,06$	$0,10 \pm 0,07$	$0,11 \pm 0,10$	$0,18 \pm 0,19$
$\Delta\theta_y$	$0,14 \pm 0,23$	$0,11 \pm 0,12$	$0,23 \pm 0,21$	$0,25 \pm 0,13$
$\Delta\theta_z$	$0,11 \pm 0,09$	$0,18 \pm 0,14$	$0,25 \pm 0,21$	$0,22 \pm 0,15$

Note. Les erreurs de translation sont données en voxels et les erreurs de rotation en degrés.

TAB. 8.11 – Statistiques sur les erreurs de recalage de la SE(16 + 16) parallèle dans le cas de l'erreur quadratique moyenne.

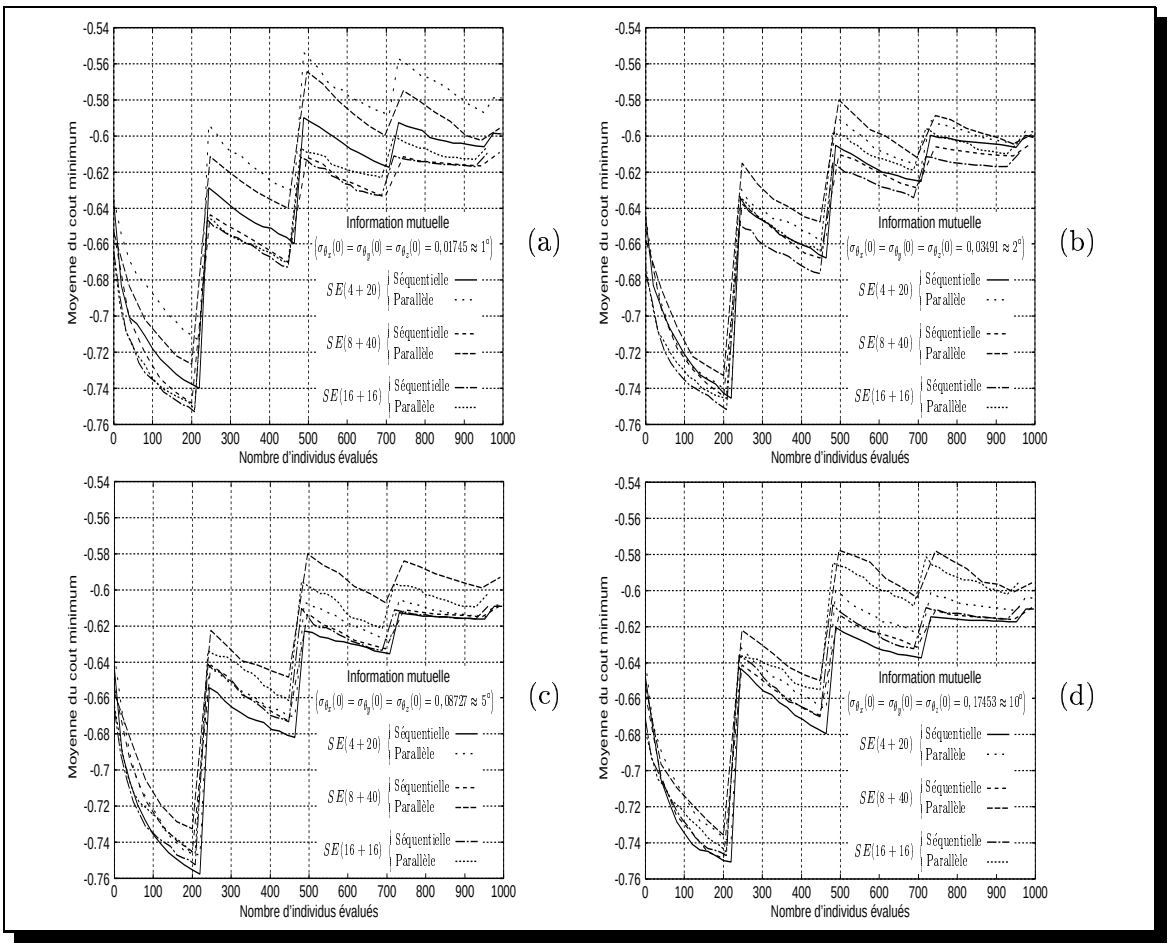


Fig. 8.21 – Stratégies d'évolution séquentielles et parallèles - évolution moyenne du coût minimum dans le cas de l'information mutuelle pour différentes tailles de populations et valeur initiale pour les écarts types des angles de rotation.

$t_{max} = 56$	Information mutuelle			
Écart type rotation	0,01745	0,03491	0,08727	0,17453
Δt_x	$0,20 \pm 0,14$	$0,22 \pm 0,16$	$0,21 \pm 0,15$	$0,18 \pm 0,14$
Δt_y	$0,17 \pm 0,10$	$0,16 \pm 0,14$	$0,15 \pm 0,11$	$0,17 \pm 0,16$
Δt_z	$0,18 \pm 0,10$	$0,17 \pm 0,10$	$0,16 \pm 0,11$	$0,15 \pm 0,12$
$\Delta\theta_x$	$0,13 \pm 0,10$	$0,14 \pm 0,14$	$0,13 \pm 0,11$	$0,17 \pm 0,21$
$\Delta\theta_y$	$0,09 \pm 0,07$	$0,13 \pm 0,12$	$0,16 \pm 0,11$	$0,23 \pm 0,20$
$\Delta\theta_z$	$0,14 \pm 0,14$	$0,17 \pm 0,12$	$0,20 \pm 0,18$	$0,28 \pm 0,35$

Note. Les erreurs de translation sont données en voxels et les erreurs de rotation en degrés.

Tab. 8.12 – Statistiques sur les erreurs de recalage de la SE(16 + 16) parallèle dans le cas de l'information mutuelle.

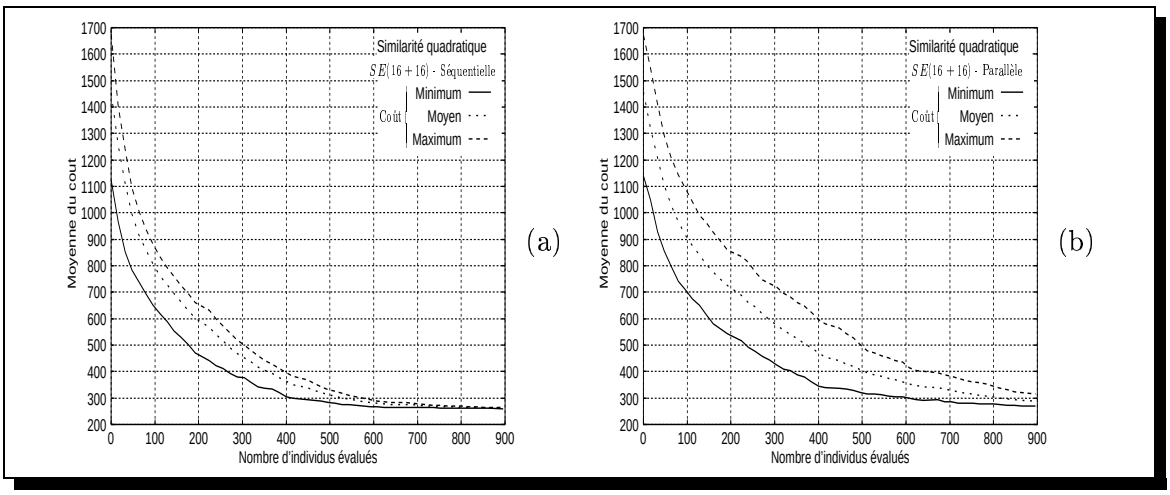


FIG. 8.22 – Stratégie d'évolution SE(16 + 16) - évolution moyenne sur les 20 recalages, dans le cas de l'erreur quadratique moyenne, du coût minimum, moyen et maximum pour (a) la version séquentielle ; (b) la version parallèle.

Temps de calcul

L'exécution de la SE(16 + 16) parallèle sur 2, 4, 8 et 16 processeurs met en évidence d'excellentes performances pour ce qui est des temps de calcul. En effet, alors que la version séquentielle s'exécute en un peu plus de 12 minutes 45 secondes, la version parallèle sur 16 processeurs demande moins d'une minute (cf. tableau 8.13). Le calcul des facteurs d'accélération et la représentation graphique de la courbe qu'ils définissent (figure 8.23) montre que l'on est pas très loin de la courbe idéale. L'origine de cette très bonne réduction du temps de calcul résulte de la régularité des communications. Comparé à l'évolution différentielle parallèle (figure 8.16), qui induit des communications irrégulières, la dégradation des performances est logiquement moindre lorsque le nombre de processeurs croît. À noter que la stratégie d'évolution SE(16+16) a un temps de calcul plus important que l'évolution différentielle pour un même nombre de populations engendrées, à cause d'une part du plus grand nombre d'individus qui sont impliqués dans la reproduction, et d'autre part de l'absence de mutation dans l'ED. Le recalage en utilisant l'information mutuelle n'altère bien entendu en rien les performances de l'algorithme parallèle : en séquentiel il faut approximativement 19 minutes contre 2 minutes 35 secondes sur 8 processeurs R12000.

Les stratégies d'évolution peuvent donc être considérées pour résoudre le problème du recalage rigide. Leur mise en œuvre n'est cependant pas aisée du fait des nombreux choix de paramètres qu'il faut faire. L'implantation data-parallèle que nous avons décrite, et qui suit le même schéma que celle de l'évolution différentielle, excelle au niveau de la réduction des temps de calcul. En revanche, pour toujours obtenir une bonne précision, un ajustement des paramètres initiaux (écarts types) et éventuellement de la valeur de t_{max} est indispensable.

Processeurs R12000		1	2	4	8	16
Individus	t_{max}					
16	52	766,93	382,53	196,24	103,75	54,27
Accélération			2,00	3,91	7,39	14,13
Processeurs R10000		1	2	4	8	16
16	52	1178,51	578,51	299,68	154,84	82,60
Accélération			2,04	3,93	7,61	14,27

Tab. 8.13 – Temps de calcul (en secondes) de la stratégie d'évolution SE$(16+16)$ dans le cas de l'erreur quadratique moyenne suivant le nombre de processeurs.

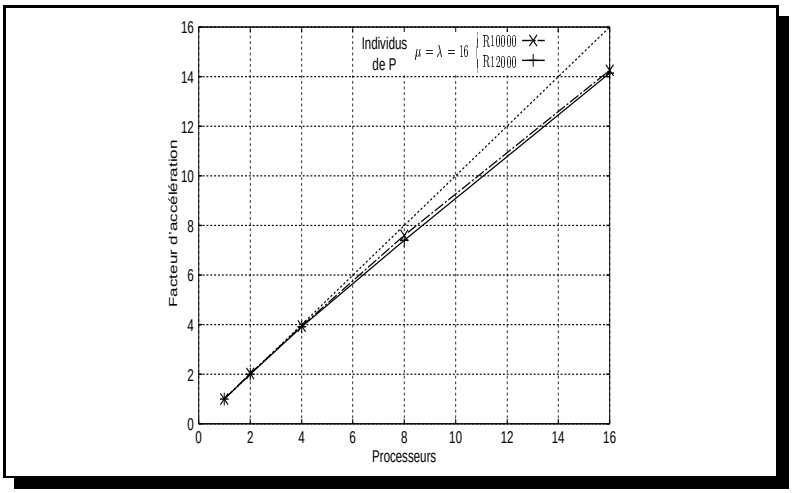


Fig. 8.23 – Courbe des facteurs d'accélération de la stratégie d'évolution SE$(16+16)$ parallèle dans le cas de l'erreur quadratique moyenne.

8.6 Genetic Simulated Annealing

L'objectif premier de l'étude de cet algorithme data-parallèle est de déterminer la pertinence d'un hybride parallèle recuit simulé/algorithme génétique pour résoudre le problème du recalage rigide. Cependant, comme pour valider le GSA, ses auteurs [29] l'avaient implanté sur une machine SIMD MasPar MP1 (128×128 processeurs) en considérant des problèmes d'optimisation discrets, notamment le voyageur de commerce, notre étude va aussi permettre de se faire une idée sur le maintien ou non des bonnes performances sur une machine non massivement parallèle.

Avant d'étudier le comportement du *Genetic Simulated Annealing*, décrit à la section 4.3, nous allons donner quelques précisions sur la représentation retenue et les opérateurs de reproduction considérés. Nous abordons également la question de l'initialisation du schéma de température associé à chaque individu/processeur virtuel.

8.6.1 Représentation et opérateurs de reproduction

Contrairement aux algorithmes évolutionnaires précédents (évolution différentielle et stratégies d'évolution), et comme tout algorithme génétique, le *Genetic Simulated Annealing* n'utilise pas de représentation précise pour les individus (voir les problèmes traités par Chen *et al.* [29]). On pourrait donc très bien envisager une représentation binaire (classique ou de Gray). Dans le cas du recalage rigide, cela se traduirait par une discrétisation plus ou moins fine de l'espace de recherche suivant le nombre de bits utilisés pour coder chacune des variables de Θ. Or ce paramètre n'est pas anodin, puisqu'il conditionne notamment la précision que l'on peut obtenir et n'est pas sans influence sur le temps de calcul (nécessité de décoder les individus avant de les évaluer). Cependant, au regard des résultats induits par l'évolution différentielle et les stratégies d'évolution, nous allons conserver la représentation réelle (voir le premier paragraphe de la section 8.4).

En ce qui concerne les opérateurs de reproduction, comme le GSA est par définition un hybride recuit simulé/algorithme génétique, les auteurs ont tout naturellement considéré des opérateurs de cette famille d'algorithmes évolutionnaires. Ainsi, le croisement est supposé être multipoint ou uniforme (section 3.3.2), et consiste donc sur chaque processeur à engendrer deux nouveaux individus en échangeant le matériel génétique des deux individus donnés en entrée. Or, dans le cas d'un codage réel, un croisement qui ne fait qu'échanger le matériel génétique des individus n'est pas satisfaisant. En effet, un croisement modifiant les composantes des individus, du type de ceux des stratégies d'évolution (voir (3.9)), permet un meilleur parcours de l'espace de recherche. C'est pourquoi, nous avons décidé de créer un croisement hybride qui en plus de mélanger le matériel génétique des deux parents, introduit de nouvelles informations dans la population. Plus précisément, étant donné un couple d'individus parents (a_1, a_2) :

- a'_1, le premier descendant, est obtenu par croisement uniforme des deux parents ;
- tandis que le second descendant, a'_2, correspond à une moyenne pondérée de a_1 et a_2, i.e. :

$$a'_2 = (1-p) \cdot a_1 + p \cdot a_2 \tag{8.25}$$

où $p \in [0;1]$ est le poids. Cela revient donc à engendrer un individu sur le segment $[a_1 a_2]$ en contrôlant sa position au moyen de p. Pour le problème du recalage rigide qui nous intéresse, nous avons défini p de façon à ce que a'_2 soit proche du parent dont le coût est le plus faible, d'où :

$$p = \frac{\Phi(a_1)}{\Phi(a_1) + \Phi(a_2)} \tag{8.26}$$

Comme opérateur de mutation, diverses adaptations de l'opérateur de mutation des algorithmes génétiques (section 3.3.2) sont possibles : changement de signe de la variable qui mute ; perturbation d'une variable en ajoutant un nombre aléatoire ; etc. Dans notre implantation, nous avons choisi de faire muter une variable en remplaçant sa valeur par un nombre tiré au hasard sur son intervalle de définition. Il faut aussi rappeler que cet opérateur requiert la définition d'une probabilité de mutation p_m.

Pour finir, précisons que le critère d'arrêt de l'algorithme, i.e. $MAX\_ITERATIONS$ populations engendrées, est identique à ceux de l'évolution différentielle et des stratégies d'évolution. Aussi, afin d'avoir une notation qui soit uniforme, dans la suite nous le noterons plutôt t_{max}.

8.6.2 Schéma de température

Suite aux résultats obtenus par Chen *et al.* [29], nous avons privilégié l'approche par schéma de température aléatoire. Dans cette optique, chaque processeur virtuel initialise le schéma de température qui lui est associé de la manière suivante :

① pour fixer la température initiale T_0, on calcule tout d'abord une remontée d'énergie moyenne (i.e. du coût) que l'on veut éventuellement accepter à la température T_0. Cette valeur moyenne, notée R_{moy}, est obtenue en considérant 100 valeurs de remontée tirées dans $[0; R]$, où R est la remontée maximale pré-définie. Dans un deuxième temps, la probabilité d'acceptation p_{acc} de la remontée moyenne calculée précédemment est tirée aléatoirement dans l'intervalle $[p_{acc,inf}; p_{acc,sup}]$. Finalement, on peut calculer T_0 de façon à accepter avec une probabilité p_{acc}, une remontée R_{moy} :

$$p_{acc} = \exp\left(-\frac{R_{moy}}{T_0}\right) \Rightarrow T_0 = -\frac{R_{moy}}{\ln p_{acc}} \tag{8.27}$$

② la température finale est obtenue par tirage aléatoire dans l'intervalle $[T_{f,inf}; T_{f,sup}]$;

③ on termine la définition du schéma de température par le calcul du coefficient de réduction $\alpha \in]0; 1[$. Ce dernier s'obtient facilement à partir de T_0, T_f et du nombre maximum (t_{max}) de populations engendrées :

$$T_f = \alpha^{t_{max}} \cdot T_0 \Rightarrow \alpha = \exp\left(\frac{1}{t_{max}} \cdot \ln\left(\frac{T_f}{T_0}\right)\right) \tag{8.28}$$

En soit, cette procédure n'est pas très compliquée, elle nécessite cependant la définition d'un nombre relativement important de paramètres.

Ces deux premiers paragraphes mettent clairement en évidence l'inconvénient majeur d'un algorithme hybride recuit simulé/algorithme génétique, et plus généralement de toute hybridation : de multiples choix à faire et un accroissement non négligeable du nombre de paramètres à fixer. La question qui se pose est donc de savoir si le jeu en vaut la chandelle. C'est-à-dire dans le cas du GSA, si l'introduction d'un schéma de température dans la sélection est bénéfique sur le plan des performances.

8.6.3 Performances et analyse de l'algorithme Genetic Simulated Annealing

Avant de voir plus en détail le comportement de l'algorithme, soulignons que tout comme dans les stratégies d'évolution étudiées précédemment, le nombre de populations engendrées au dernier niveau de résolution ne sera pas nécessairement nul. En effet, comme nous le verrons t_{max} ne sera pas obligatoirement un multiple de 4. Par conséquent, il arrivera parfois que $t_{max} - 4 \cdot \lfloor \frac{t_{max}}{4} \rfloor$ ne soit pas nul, ce qui se traduira par des itérations à la résolution la plus fine. D'autre part, en raison de tailles de populations plutôt réduites, nous avons fixé $MAX\_DISTANCE$ à 1. On a donc sur un processeur, l'individu voisin nécessaire pour la reproduction qui est choisi dans un voisinage du même type que celui de l'évolution différentielle. Enfin, il est à noter que nous avons restreint l'étude du GSA à l'erreur quadratique moyenne, car, comme nous allons le voir, il requiert un grand nombre de paramètres, d'où une certaine difficulté à trouver de bonnes valeurs pour ces derniers.

✐ Impact du schéma de température et de la probabilité de mutation

Pour mettre en évidence l'impact du schéma de température, nous considérons d'une part différentes valeurs pour la remontée maximale autorisée et d'autre part plusieurs nombres maximaux de populations engendrées. Ce dernier nombre est fonction de la taille de la population (μ). Il est fixé de façon à ce que de l'ordre de 900 individus soient évalués (mis à part l'évaluation de $P(0)$), afin de pouvoir apprécier la convergence du *Genetic Simulated Annealing* par rapport à l'évolution différentielle et aux stratégies d'évolution. Pour ce qui est des bornes des intervalles de définition de p_{acc} et T_f, nous avons choisi des valeurs qui permettent d'obtenir des schémas de température variés d'un processeur virtuel à un autre :

- $[p_{acc,inf}; p_{acc;sup}] = [0,25; 1[$
- $[T_{f,inf}; T_{f,sup}] = [0,001; 0,25]$

Le dernier paramètre devant être spécifié, la probabilité de mutation p_m, est dans un premier temps définie de sorte qu'une variable sur six testées mute. Nous verrons plus loin, l'influence de ce paramètre sur la convergence de l'algorithme.

Les quatre graphiques présentés à la figure 8.24 montrent l'évolution moyenne sur 20 recalages du coût minimum pour des populations de 8, 16 et 32 individus (t_{max} vaut respectivement 54, 26 et 12), et des valeurs de remontée maximale décroissant régulièrement de 500 à 62,5. Plusieurs remarques peuvent être faites :

- plus la remontée est importante, plus la vitesse de convergence au début de l'algorithme est faible. En effet, plus R est élevée, plus les « mauvais » individus sont préservés, ralentissant d'autant la convergence ;
- à nombre d'individus évalués équivalent, une petite population se traduira par des schémas de décroissance plus lents sur ses processeurs qu'une population plus grande. En prenant une grande population, il y a donc un risque d'obtenir une décroissance trop rapide des températures, ce qui peut favoriser le blocage dans un optimum local ;
- c'est toujours la même taille de population qui donne les meilleurs résultats. Par conséquent on pourrait penser que l'introduction d'un schéma de température dans la phase de sélection n'est absolument pas bénéfique, puisqu'il ne ferait que modifier la vitesse de convergence. De plus, cela signifierait également qu'un schéma de température nul donnerait les meilleurs résultats. Or, si on regarde précisément les courbes pour $R = 125$ et $R = 62,5$, on constate que la valeur la plus élevée donne « légèrement » de meilleurs résultats. En conclusion, l'introduction d'un schéma de température apporte bien un léger plus, puisque cela permet d'améliorer la qualité des recalages.

Par rapport à l'évolution différentielle et aux stratégies d'évolution, on constate que l'algorithme parallèle *Genetic Simulated Annealing* se comporte honorablement. De fait, pour une taille de population de 16 individus, la vitesse de convergence initiale est du même ordre que celle de l'évolution différentielle parallèle (figure 8.14(a)), voire plus rapide. En revanche elle décroît fortement par la suite, ce qui aboutit au final, après l'évaluation de plus de 900 individus, à des recalages qui sont de moins bonne qualité.

Afin de vérifier que les performances du GSA augmentent bien avec la taille de la population, une propriété qui a été exhibée par Chen *et al.* [29] lors de leurs expérimentations, nous avons ensuite doublé le nombre maximum de populations engendrées. En prenant pour R la valeur ayant donnée les meilleurs résultats, soit $R = 125$. On a ainsi pour une taille de population de 8, 16 et

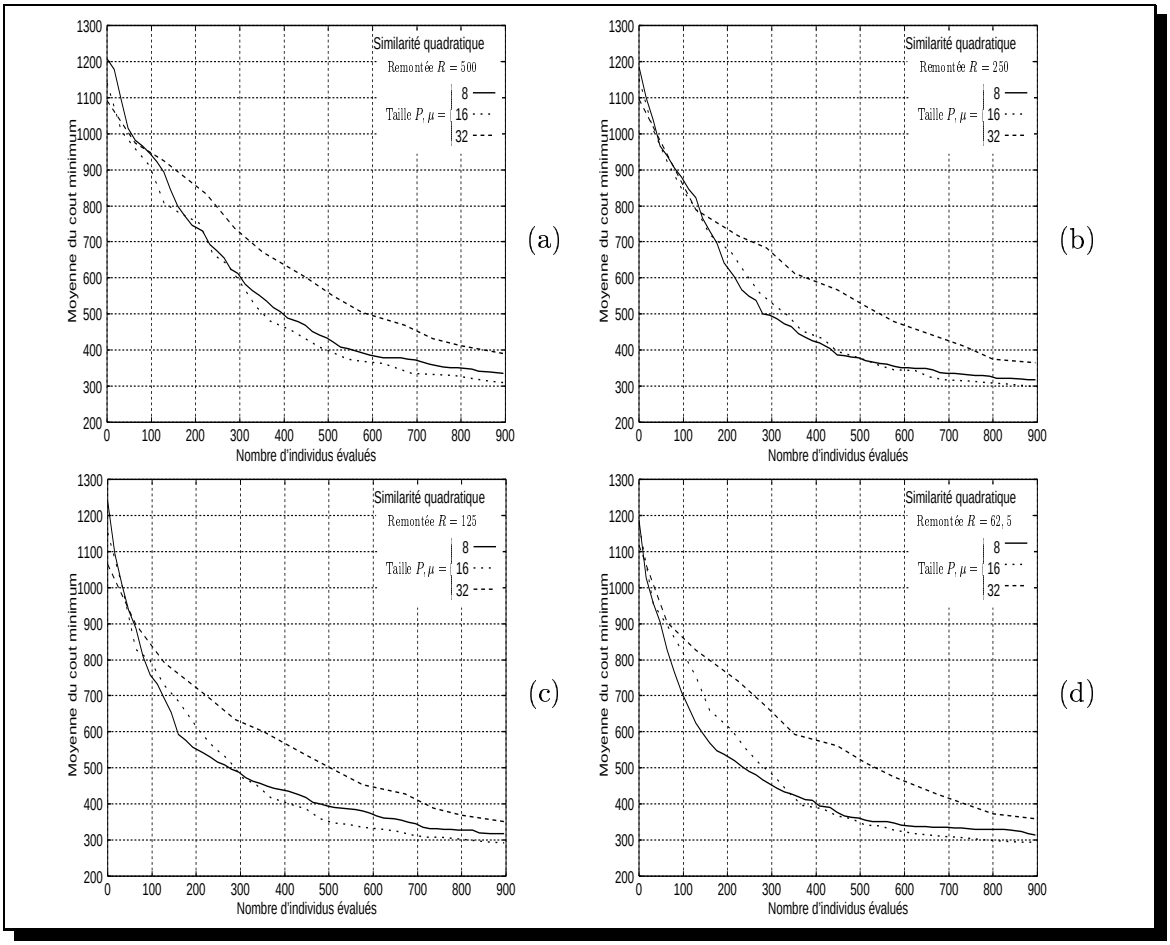


FIG. 8.24 – *Genetic Simulated Annealing* - convergence de l'algorithme suivant la taille de la population (soit également le nombre maximum de populations engendrées) et la remontée que l'on permet : (a) $R = 500$, (b) $R = 250$, (c) $R = 125$ et (d) $R = 62,5$.

32 individus, t_{max} qui vaut respectivement 108, 52 et 24, ce qui se traduit notamment sur chaque processeur virtuel (où se trouve un individu de la population) par un schéma de température de décroissance moins rapide que précédemment. Au vu des courbes que l'on obtient, présentées par le graphique 8.25(a), la propriété semble se vérifier.

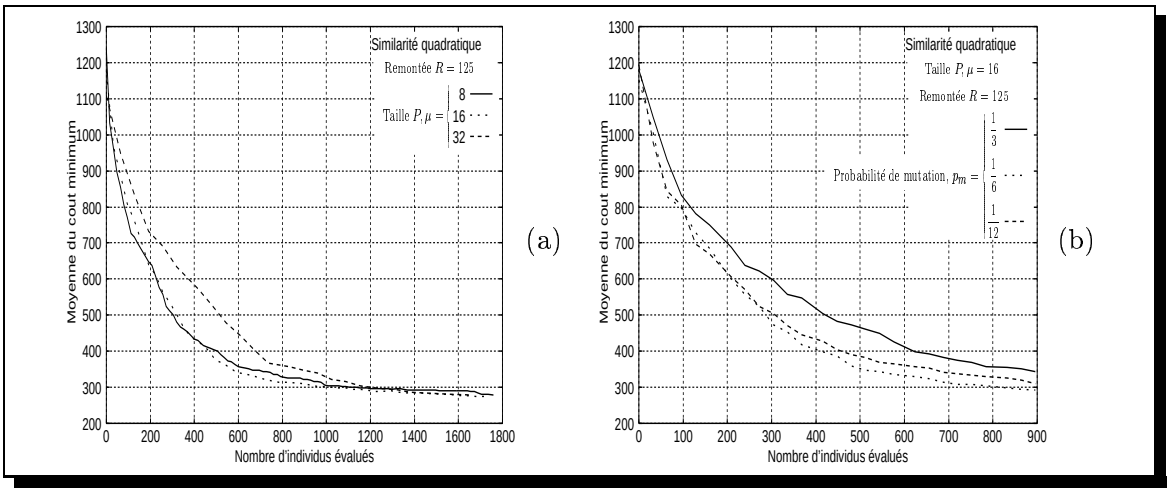


Fig. 8.25 – *Genetic Simulated Annealing* - convergence de l'algorithme suivant la taille de la population, $R = 125$: (a) lorsque l'on augmente le nombre de population engendrées ; (b) pour différentes valeurs de la probabilité de mutation.

Finalement, il reste à étudier l'influence de la probabilité de mutation (p_m) sur la convergence. Aussi, nous avons exécuté l'algorithme avec une taille de population de 16 individus, une remontée de 125, et p_m qui est successivement égale à : $\frac{1}{3}$, $\frac{1}{6}$ et $\frac{1}{12}$, soit un nombre de mutations décroissant régulièrement, passant de une variable sur trois mutées, à une sur douze. Ce que l'on remarque (figure 8.25(b)), c'est qu'un trop grand nombre de mutations altère la convergence de l'algorithme. En effet, les mutations perturbent aléatoirement les individus de la population, aussi, si elles sont trop nombreuses, elles aboutissent à la perte d'informations reflétant l'historique de la recherche. Cependant, le rôle de la mutation semble insignifiant dans la génération du premier tiers de populations, i.e. pour $t \leq \frac{t_{max}}{3}$. De fait, on voit que les courbes sont quasi-identiques pour $p_m = \frac{1}{6}$ et $p_m = \frac{1}{12}$, ce qui montre bien que c'est le croisement qui est prépondérant pour guider la recherche. Toutefois, les deux courbes divergent ensuite, avec la probabilité de mutation la plus faible, qui est la moins performante. Il apparaît donc que dans un deuxième temps, l'opérateur de mutation permet d'introduire des informations pertinentes, aboutissant à une amélioration de la qualité des résultats.

Qualité des recalages

Comme nous l'avons déjà souligné, à l'issue de 900 individus évalués, les résultats induits par le *Genetic Simulated Annealing* sont de moins bonnes qualité que ceux de l'évolution différentielle ou des stratégies d'évolution (parallèles). En effet, comme on peut le voir sur la figure 8.26(b) qui présente l'évolution moyenne du coût minimum, moyen et maximum pour une population de 16 individus et $R = 125$, c'est-à-dire pour le jeu de paramètres qui a donné les meilleurs résultats,

le coût minimum final n'est que légèrement inférieur à 300. D'ailleurs, les erreurs moyennes sur les 20 recalages, données dans le tableau 8.14, mettent en évidence la moindre qualité des recalages. Le calcul de l'écart quadratique moyen montre en particulier que certains recalages n'ont pas la précision sous-voxel souhaitée ($EQM = 0,74 \pm 0,34$ voxels).

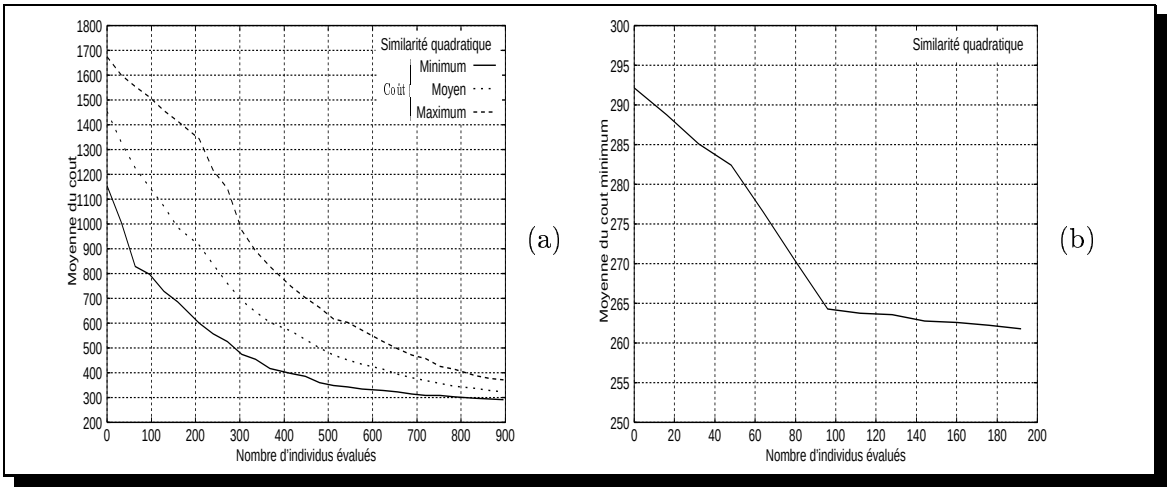


FIG. 8.26 – (a) *Genetic Simulated Annealing* - évolution moyenne sur les 20 recalages du coût minimum, moyen et maximum (16 individus et $R = 125$) ; (b) amélioration moyenne du coût minimum obtenu en (a) par l'algorithme *Iterated Conditional Modes*.

Pour résoudre ce problème, deux solutions peuvent être envisagées : d'une part engendrer un plus grand nombre de populations en augmentant t_{max}, d'autre part le raffinement du vecteur Θ résultant de l'exécution du GSA par l'intermédiaire d'une méthode d'optimisation locale.

La première solution présente l'avantage d'être facilement mise en œuvre. De plus, elle semble se justifier au vu des courbes moyennes du coût moyen et maximum : par rapport aux mêmes courbes pour l'évolution différentielle parallèle (figure 8.14(b)) on remarque que les courbes ne sont pas confondues, ce qui signifie que la convergence de l'algorithme n'est pas terminée. Néanmoins, pour obtenir une amélioration significative de la qualité des recalages, il faut apparemment plus que doubler le nombre de populations engendrées. En effet, le graphique 8.25(a), qui montre la convergence du GSA lors du doublement de la valeur de t_{max}, permet d'affirmer que l'algorithme converge de plus en plus lentement. Si on veut obtenir une amélioration significative des résultats, il faudra donc générer un grand nombre de populations supplémentaires, avec pour conséquence la plus immédiate, un accroissement fort important du temps de calcul, i.e. la multiplication par un facteur supérieur à deux.

L'alternative à l'augmentation du nombre de populations engendrées est de raffiner le résultat produit par le *Genetic Simulated Annealing* par une méthode d'optimisation locale, par exemple l'algorithme *Iterated Conditional Modes* (ICM). Cet algorithme déterministe, introduit par Besag [17] en 1986, est décrit dans le contexte du recalage rigide à la figure 8.27. Comme l'ICM n'accepte que des vecteurs Θ induisant une décroissance de l'énergie (du coût), il doit-être impérativement initialisé dans le voisinage de l'optimum global. Ceci est *a priori* le cas en considérant comme vecteur de départ le résultat du GSA.

$\Theta \leftarrow \Theta_0$ - *Configuration initiale*
$\Theta_{min} \leftarrow \Theta$
$C_{min} \leftarrow C(\Theta)$ - *Coût minimal*
$Indicateurs \leftarrow \{faux, faux, faux, faux, faux, faux\}$ - *Vecteur permettant de distinguer les paramètres de Θ à optimiser (faux), de ceux qui le sont déjà*
$k \leftarrow 0$ - *Compteur*
Tant que $k < MAX\_ITERATIONS$ **faire**
  $k \leftarrow k + 1$
  Répéter
    $i \leftarrow$ **Un entier tiré selon un loi uniforme dans** $\{1, 2, 3, 4, 5, 6\}$**, où** $1 \Leftrightarrow t_x, 2 \Leftrightarrow t_y, ...$
    Construire un ensemble D_Θ **de configurations qui différent de** Θ **au niveau de la** i**-ème variable**
    $\Theta \leftarrow \Theta' \in D_\Theta \mid C(\Theta') = \min_{\Theta'' \in D_\Theta} (C(\Theta''))$
    Si $C(\Theta) < C_{min}$ **alors** $\Theta_{min} \leftarrow \Theta, C_{min} \leftarrow C(\Theta)$ **Fin si**
    Mettre la i**-ème variable de** $Indicateurs$ **à** $vrai$
  Jusqu'à $Indicateurs = \{vrai, vrai, vrai, vrai, vrai, vrai\}$
  Réduire l'espace de recherche du vecteur Θ
  $Indicateurs \leftarrow \{faux, faux, faux, faux, faux, faux\}$
Fin tant que

Fig. 8.27 – Algorithme *Iterated Conditional Modes* (ICM) dans le cas du recalage rigide.

L'inconvénient majeur de l'ICM est que l'évaluation des différents vecteurs de paramètres se fait à la résolution la plus fine. Le coût en temps de calcul est donc très important, surtout si le nombre de configurations construites dans l'ensemble D_Θ lors d'une itération est grand (le nombre de configurations évaluées est égal à : $MAX\_ITERATIONS \cdot (6 \cdot |D_\Theta|)$). Pour réduire la complexité calculatoire de l'algorithme, nous prolongeons le schéma d'évaluation parallèle des individus (les vecteurs Θ) du *Genetic Simulated Annealing*. Ainsi, le nombre de configurations construites lors de l'optimisation d'une variable est tel que le cardinal de D_Θ soit égal à la taille de la population du GSA. En quelque sorte, le schéma parallèle est continu tout au long du processus d'optimisation. Cela permet notamment de tirer parti de manière optimale des processeurs physiques qui sont utilisés par l'algorithme d'optimisation globale.

La pertinence de l'approche par optimisation locale de la solution obtenue par le *Genetic Simulated Annealing* est clairement mise en évidence par le graphique 8.26. On voit notamment que deux itérations suffisent (192 vecteurs de paramètres sont évalués) pour améliorer sensiblement la qualité des recalages. Le résultat du calcul de statistiques sur les erreurs de recalage, présenté dans le tableau 8.14, montre que c'est essentiellement la précision des angles de rotation qui est meilleure. Au final, l'ICM permet d'obtenir systématiquement un recalage de précision sous-voxel : l'écart quadratique moyen entre le recalage obtenu et la solution optimale est de $0,40 \pm 0,12$ voxels, ce qui correspond pratiquement à une amélioration qualitative d'un facteur deux du résultat de l'optimisation par l'agorithme GSA.

Genetic Simulated Annealing					
Δt_x	voxels	$0,24 \pm 0,17$	$\Delta\theta_x$	degrés	$0,36 \pm 0,26$
Δt_y		$0,21 \pm 0,15$	$\Delta\theta_y$		$0,24 \pm 0,17$
Δt_z		$0,22 \pm 0,14$	$\Delta\theta_z$		$0,41 \pm 0,39$
Iterated Conditional Modes					
Δt_x	voxels	$0,19 \pm 0,15$	$\Delta\theta_x$	degrés	$0,12 \pm 0,08$
Δt_y		$0,15 \pm 0,10$	$\Delta\theta_y$		$0,16 \pm 0,08$
Δt_z		$0,16 \pm 0,09$	$\Delta\theta_z$		$0,05 \pm 0,03$

Tab. 8.14 – Moyennes et écarts types des erreurs de recalage après *Genetic Simulated Annealing*, puis raffinement par l'algorithme déterministe *Iterated Conditional Modes*.

Temps de calcul

Passons maintenant à l'étude des performances au niveau des temps de calcul. Pour ce faire, le *Genetic Simulated Annealing* a été exécuté en considérant le jeu de paramètres ayant donné les meilleurs recalages, sur 1, 4, 8 et 16 processeurs R12000. Au vu des temps que l'on obtient, et qui sont présentés dans la première ligne du tableau 8.15, on peut faire la remarque suivante : pour un nombre d'individus évalués équivalent, le GSA est nettement plus coûteux que l'évolution différentielle qui présente un schéma de parallélisation relativement proche. En effet, t_{max} n'étant pas un multiple de 4, deux itérations du *Genetic Simulated Annealing* ont lieu à la résolution la plus fine (ce qui correspond à l'évaluation de 64 individus), contre aucune dans l'évolution différentielle. La phase de reproduction est également plus gourmande dans le GSA, car elle est bien plus complexe à mettre en œuvre. Enfin, le schéma de température requiert des cycles de calcul qui ne sont pas négligeables.

Cependant, le calcul des facteurs d'accélération relative (la courbe que l'on obtient est donnée dans le graphique 8.28) montre que l'algorithme hybride est un peu plus efficace que la version parallèle de l'évolution différentielle. En effet, l'efficacité est respectivement de 96,75%, 93,13% et 85,53% sur 4, 8 et 16 processeurs dans le cas du *Genetic Simulated Annealing* contre 95,10%, 91,81% et 84,08% pour l'évolution différentielle parallèle. Pour ce dernier algorithme, le calcul des facteurs d'accélération relative s'est fait à partir des temps d'exécution du tableau 8.8, sachant que sur un processeur R12000, l'algorithme requiert 701,70 secondes. L'explication de la meilleure efficacité du GSA se situe au niveau des communications : l'hybride parallèle nécessite beaucoup moins de communications. En effet, lors de la reproduction le nombre d'individus parents requis est différent dans les deux algorithmes parallèles. De fait, dans le GSA, un seul individu voisin est nécessaire en plus de l'individu du processeur (virtuel) considéré, soit une communication. Tandis que dans l'implantation parallèle proposée pour l'évolution différentielle, la définition retenue pour l'opérateur de reproduction (8.22) se traduit par trois communications : il faut transmettre les voisins a_S et a_T, ainsi que l'optimum courant a_{min}. De plus, l'étape de reproduction (sur un processeur virtuel) du *Genetic Simulated Annealing* produit deux nouveaux individus, alors que ce n'est qu'un seul dans l'évolution différentielle parallèle.

La seconde série de temps de calcul du tableau 8.15 permet d'apprécier le surcoût induit par l'utilisation de l'algorithme d'optimisation locale *Iterated Conditional Modes*. La première chose que l'on constate si on regarde le temps obtenu sur un processeur, c'est que l'évaluation de moins de 200 vecteurs de paramètres dans l'ICM est aussi coûteuse que l'évaluation des 900 individus

Processeurs R12000				1	4	8	16
GSA sans	Individus	t_{max}	R				
raffinement par ICM	16	26	125	935,84	241,82	125,62	68,39
Accélération relative					3,87	7,45	13,68
GSA avec	$MAX\_ITERATIONS$						
raffinement par ICM	2			1864,82	487,26	251,14	131,66
Accélération relative					3,83	7,43	14,16

Tab. 8.15 – Temps de calcul (en secondes) du *Genetic Simulated Annealing*, sans et avec raffinement par l'algorithme *Iterated Conditional Modes*, pour différents nombres de processeurs.

dans le *Genetic Simulated Annealing*. L'origine de ce phénomène est la résolution à laquelle se font les évaluations : à la résolution la plus fine dans l'algorithme d'optimisation locale ; suivant une approche multirésolution dans l'hybride parallèle. Par conséquent, si on veut obtenir un recalage de qualité dans un temps de calcul raisonnable, la parallélisation de l'ICM s'impose. La deuxième remarque concerne justement l'efficacité de notre implantation parallèle : on constate que celle-ci décroît moins rapidement suivant le nombre de processeurs que dans le cas du GSA (la courbe des facteurs d'accélération relative s'écarte moins rapidement de la courbe d'accélération idéale). En effet, bien que la parallélisation de l'ICM se réduise uniquement à distribuer l'évaluation des vecteurs de paramètres de D_Θ, et donc implique de nombreuses communications irrégulières, la proportion de calcul est nettement plus importante dans l'ICM, d'où une décroissance plus lente de l'efficacité. Enfin, au vu des efficacités sur 4 processeurs, on voit que les communications irrégulières sont les plus coûteuses, puisque l'efficacité est respectivement de 96,75% et 95,68% sans et avec raffinement par l'ICM. En définitive, l'utilisation de l'ICM est d'autant moins pénalisante que le nombre de processeurs physiques utilisés est important.

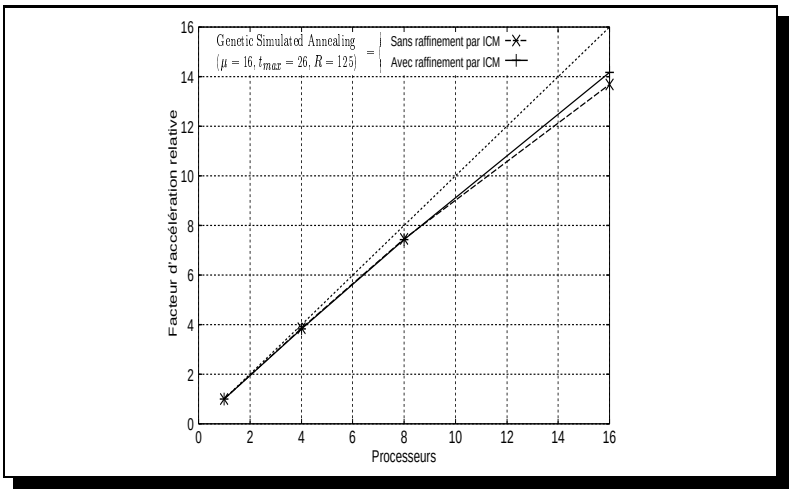


Fig. 8.28 – Courbe des facteurs d'accélération relative du *Genetic Simulated Annealing*, sans et avec raffinement par la méthode d'optimisation locale ICM.

8.7 Conclusion partielle

Ce chapitre a été consacré à l'étude de l'adéquation de différents algorithmes d'optimisation globale au problème du recalage rigide, ainsi qu'à leur mise en œuvre data-parallèle. Parmi les algorithmes étudiés, il y a d'une part deux algorithmes qui sont liés au recuit simulé classique : l'*Adaptive Simulated Annealing* ou recuit simulé adaptatif et l'équation de la diffusion, qui opèrent tout deux dans le domaine continu. D'autre part, l'évolution différentielle et les stratégies d'évolution, des algorithmes évolutionnaires qui utilisent une représentation réelle. Enfin, en dernier lieu, nous avons également considéré un algorithme hybride data-parallèle recuit simulé/algorithme génétique, en l'occurrence le *Genetic Simulated Annealing*.

Le premier algorithme étudié, l'équation de la diffusion, a pour principe la perturbation aléatoire de la descente du gradient. Comme cette perturbation est contrôlée par une température, l'algorithme est initialement stochastique, puis, au fur et à mesure des itérations, il change de mode pour devenir déterministe. Le recours au gradient fait que la fonction de similarité doit être dérivable, d'où la restriction de l'algorithme à l'erreur quadratique moyenne et donc au recalage monomodal. L'équation de la diffusion requiert peu de paramètres : une température, un pas de temps et un critère d'arrêt, et comme nous l'avons vu, il est relativement robuste. Aussi, si le nombre de configurations engendrées est suffisant pour identifier le bassin d'attraction de l'optimum global, alors l'équation de la diffusion converge toujours vers le même vecteur de paramètres de la transformation rigide, avec au final un recalage de précision sous-voxel. Notons que le choix du pas de temps doit être fait en fonction de la valeur des dérivées partielles, ce qui explique pourquoi nous avons associé un pas différent aux paramètres de translation et aux paramètres de rotation. En raison du nombre limité de variables à optimiser, la parallélisation massive de l'équation de la diffusion (remise à jour en parallèle de chacune des variables) ne permet d'escompter qu'une réduction limitée du temps de calcul. De plus, elle s'avère inintéressante, car l'expression du gradient de l'erreur quadratique moyenne par rapport aux variables à optimiser montre que les calculs locaux propres à chaque processeur sont négligeables.

Le recuit simulé adaptatif se caractérise par une température associée à chacune des variables à optimiser, en plus de la température d'acceptation. Ces températures sont utilisées par l'algorithme dans la phase d'exploration pour affiner ou élargir la recherche. En outre, cet algorithme comporte une phase supplémentaire, dite phase de *reannealing*, dont l'objectif est la remontée de toutes les températures, à l'exception de celle associée à la variable qui influence le plus grandement l'évolution du coût. Toutefois, à la suite d'expérimentations préliminaires nous avons vu que la phase de *reannealing* pouvait être supprimée dans le cas du recalage rigide, auquel cas on peut prendre une température unique pour toutes les variables. D'autre part, déterminer un bon jeu de paramètres pour l'algorithme n'est pas trop difficile. Nous avons en particulier mis en évidence l'influence sur la vitesse de convergence de l'un de ces paramètres. Le résultat de l'étude de la version séquentielle est que l'*Adaptive Simulated Annealing* est clairement apte à résoudre le problème posé, avec des erreurs de recalage relativement faibles. Nous nous sommes alors orienté vers une mise en œuvre data-parallèle multi-températures déterministe et probabiliste. Du fait de l'absence de la phase de *reannealing* la parallélisation suivant une approche multi-températures de l'*Adaptive Simulated Annealing* ne pose pas de problème particulier : par exemple, dans le cas de la propagation déterministe, la température associée aux variables est à l'image de la température d'acceptation, fixe sur un processeur. Néanmoins, nous avons discuté du problème

qu'induirait le *reannealing* au niveau du schéma de propagation des configurations. Dans le cas d'un schéma de propagation déterministe, le recuit simulé adaptatif multi-températures met au jour une baisse de la vitesse de convergence avec l'augmentation du nombre de processeurs. De même, plus la période entre deux propagations est réduite, plus la convergence est rapide (la recherche est mieux guidée). Le schéma de propagation probabiliste a, lui, été mis en œuvre d'une manière particulière : nous n'avons pas considéré le même nombre de températures que de processeurs, mais exactement le même nombre qu'en séquentiel. Cet algorithme systolique modifié n'a pas donné de bons résultats. En revanche, si on remplace la probabilité d'acceptation d'une configuration provenant d'un autre processeur par celle de la phase d'acceptation de l'ASA on obtient de meilleurs résultats : bonne vitesse de convergence et dégradation moindre des performances avec l'augmentation du nombre de processeurs. Sur le plan des temps de calcul, nous avons vu qu'il est possible d'avoir d'excellentes accélérations. Le recuit simulé adaptatif multi-températures déterministe et la version systolique modifiées permettent ainsi d'obtenir rapidement les paramètres d'une transformation rigide qui n'est pas très loin de la transformation optimale. Cependant, leur raffinement est nécessaire pour garantir la précision sous-voxel, notamment lorsqu'on prend comme fonction de similarité l'information mutuelle.

L'étude de l'évolution différentielle a montré que c'est un algorithme d'optimisation globale robuste et flexible. Le fait qu'une taille de population réduite permet à l'algorithme de converger rapidement et d'aboutir à un recalage final de très bonne qualité, nous a conduit à proposer une parallélisation à grain fin. Celle-ci consiste à distribuer la population à raison d'un individu par processeur et à paralléliser les différentes étapes de l'algorithme. La seule distinction entre les versions séquentielle et parallèle se situe au niveau de la phase de reproduction, puisque dans l'approche data-parallèle, nous considérons un voisinage pour choisir les individus qui participent à la reproduction localement sur un processeur (en plus du meilleur individu courant). Bien que notre algorithme parallèle soit sémantiquement différent de l'évolution différentielle séquentielle, il donne lieu à un recalage de précision sous-voxel, avec une accélération quasi-linéaire suivant le nombre de processeurs. De fait, il est apparu que les deux versions de l'algorithme ont un comportement identique grâce à la taille du voisinage choisi. La dégradation des performances calculatoires que l'on observe résulte des communications irrégulières requises pour la diffusion du minimum courant.

Pour les stratégies d'évolution, la mise en œuvre séquentielle s'est traduit par des résultats du même ordre que pour l'évolution différentielle. Cependant, il faut remarquer que du fait de l'intégration de paramètres supplémentaires dans les individus en plus des variables à optimiser, cette mise en œuvre est plus compliquée. Un des résultats que l'on a mis en évidence est, qu'aussi bien les stratégies d'évolution élitistes que les non élitistes permettent d'obtenir d'excellents recalages. L'étude de l'influence de la taille de la population des parents et des descendants nous a poussé à suivre la même approche pour la parallélisation que pour l'évolution différentielle. Sémantiquement, la différence entre les versions séquentielle et parallèle est plus marquée. Puisqu'en plus de la phase de reproduction, la sélection parallèle que nous avons choisie localement élitiste, ne l'est pas globalement, notre parallélisation n'est pas à proprement parler élitiste ou non élitiste. Le résultat est que son comportement est très différent des stratégies d'évolution séquentielles : une convergence plus lente et une précision moindre. Pour garantir une bonne précision, un ajustement des paramètres de l'algorithme s'impose et, éventuellement, un plus grand nombre d'individus engendrés qu'en séquentiel. Si on veut un algorithme parallèle robuste, on peut envis-

ager un raffinement par une méthode d'optimisation locale. D'autre part, bien que les temps de calcul soient légèrement supérieurs à ceux de l'évolution différentielle, comme les communications sont régulières, on a de meilleurs facteurs d'accélération.

En dernier lieu, le *Genetic Simulated Annealing* a montré qu'il pouvait être intéressant d'avoir recours à l'hybridation. De fait, l'introduction de schémas de température dans la phase de sélection a permis d'améliorer la qualité des recalages. Cependant l'accroissement du nombre de paramètres de l'algorithme n'est pas à négliger. En étudiant l'impact de la probabilité de croisement, nous avons vu que, dans un premier temps, c'est surtout le croisement qui guide la recherche, puis à mesure que la population converge, un opérateur de mutation avec un taux raisonnable (une variable mutée sur six testées) apporte des informations bénéfiques. Néanmoins, qualitativement les recalages obtenus sont moins bons que ceux produits par l'évolution différentielle parallèle ou la parallélisation des stratégies d'évolution (en considérant les meilleurs jeux de paramètres). Nous avons vu qu'il est possible de garantir la précision sous-voxel en utilisant un algorithme d'optimisation locale tel que l'*Iterated Conditional Modes*, mais au prix d'un surcoût calculatoire important. Par ailleurs, en dépit d'un schéma relativement proche de l'évolution différentielle parallèle, le GSA s'avère plus coûteux en temps de calcul, mais en requérant moins de communications, d'où une meilleure efficacité. De plus, en parallélisant l'ICM, il est possible de limiter le surcoût, ce qui permet *in fine* d'obtenir sur un grand nombre de processeurs un temps de calcul global, i.e. optimisation globale et locale ensemble, tout à fait acceptable.

D'une manière générale, cette étude a mis au jour la bonne adéquation du modèle data-parallèle pour la mise en œuvre parallèle de nombreux algorithmes d'optimisation globale. D'autre part, l'implantation des algorithmes parallèles au moyen de directives HPF sur une machine MIMD établit clairement l'indépendance entre les modèles de programmation et d'exécution.

Chapitre 9

Cas du recalage déformable

9.1 Introduction

À la suite de l'étude des algorithmes d'optimisation globale sur le problème du recalage rigide, menée dans le chapitre précédent, nous avons décidé de focaliser notre attention dans le cadre du recalage déformable sur deux algorithmes :

① l'évolution différentielle ;

② l'équation de la diffusion.

D'une part l'évolution différentielle s'est avérée être un algorithme évolutionnaire performant, robuste et nécessitant peu de paramètres. Sa parallélisation suivant une approche data-parallèle s'est, de plus, traduit par un algorithme parallèle efficace, permettant d'obtenir un recalage rigide de qualité très proche de celui produit par la version séquentielle. Au demeurant, les comportements de l'évolution différentielle parallèle et de la version séquentielle sont quasiment équivalents.

D'autre part l'équation de la diffusion est l'algorithme d'optimisation globale qui a donné les meilleurs recalages rigides en séquentiel. De surcroît, par définition, l'équation de la diffusion est sur le plan du parallélisme un algorithme massivement parallèle. Or cette approche est *a priori* la plus adéquate pour la méthode de recalage déformable considérée qui requiert l'optimisation de plusieurs centaines de milliers, voire de millions de variables (cf. section 6.2.5). Néanmoins, ainsi que nous l'avons vu à la section 8.2 pour le problème du recalage rigide, la pertinence de cette parallélisation et le gain que l'on peut obtenir sont fonction de l'expression des dérivées partielles. Le fait que l'équation de la diffusion ne soit adaptée qu'à l'optimisation de fonctions de coût dérivables n'est en outre pas problématique, puisque la fonction de similarité qui est en général utilisée en recalage déformable est l'erreur quadratique (6.4) et celle-ci se dérive sans difficultés. En effet, à l'heure actuelle ce ne sont presque exclusivement que des images 3D monomodales qui sont traitées, l'extension au cas multimodal n'en est qu'à ses débuts.

Les autres algorithmes d'optimisation globale (recuit simulé adaptatif, stratégies d'évolution, etc) ont été exclus de l'étude soit en raison de leurs performances moindres sur le problème du recalage rigide, notamment en parallèle, soit du fait de leur manque de robustesse ou des nombreux choix nécessaires pour les mettre en œuvre. En particulier, l'approche multi-températures de l'*Adaptive Simulated Annealing* et le *Genetic Simulated Annealing* obligent à recourir à une

méthode d'optimisation locale dans la perspective d'une précision sous-voxel systématique. Ceci est également le cas des stratégies d'évolution, dès lors que l'ajustement fastidieux des paramètres veut être évité. Par ailleurs, cette dernière famille d'algorithmes évolutionnaires ne peut être raisonnablement envisagée pour optimiser un grand nombre de variables (comme c'est le cas en recalage déformable). En effet, l'intégration des paramètres dans les individus (écarts types) se traduit par des individus comportant un très grand nombre de valeurs, puisqu'un paramètre commun à toutes les variables n'est pas possible en recalage déformable à cause des déformations qui deviennent de plus en plus locales lorsque la résolution croît. Comme les paramètres subissent en outre aussi le processus de l'évolution, une stratégie d'évolution sera très coûteuse en temps de calcul. Pour terminer, se pose la question du choix de la valeur initiale de chacun de ces paramètres.

Rappelons que bien que la fonction de similarité considérée soit l'erreur quadratique, la fonction de coût qui est effectivement optimisée est l'erreur quadratique moyenne. Le fait de diviser par le nombre de voxels des images permet de réduire la valeur des dérivées partielles, mais aussi de la fonction de coût. De plus, comme vu dans l'état de l'art du recalage en imagerie médicale (section 5.5), le recalage déformable de deux images n'est pas réalisé directement, mais s'inscrit dans une approche hiérarchique consistant à recaler les images en considérant progressivement des transformations de plus en plus complexes. En clair, le recalage déformable est la dernière étape d'une séquence de recalages où on a successivement recherché une transformation rigide, rigide avec zoom et affine. Comme nous le verrons ci-dessous, pour effectuer les différents recalages préalables au recalage déformable, nous avons utilisé l'évolution différentielle parallèle. Notons pour finir que les deux algorithmes d'optimisation considérés ne seront pas étudiés en séquentiel, mais immédiatement en parallèle ; qu'aucune transformation multirésolution des données images n'est utilisée (la fonction de coût est calculée en tenant compte de tous les voxels) et que pour calculer l'image recalée nous avons conservé l'interpolation trilinéaire.

9.2 Évolution différentielle

Relativement à l'implantation data-parallèle proposée pour résoudre le problème du recalage rigide, nous n'avons apporté que très peu de modifications pour traiter le cas affine et déformable : les seuls changement se situent au niveau de la longueur de croisement L et de la probabilité de croisement p_c. En effet, tel qu'il est défini, l'opérateur de reproduction de l'évolution différentielle (section 3.5.2), et donc de la version parallèle, suppose que le nombre L de composantes (variables) qui sont échangées entre un individu de départ $a_j, j \in \{1, ..., \mu\}$ et l'individu intermédiaire correspondant a''_j pour produire un descendant a'_j, varie d'un processeur virtuel à un autre. Aussi est-il fort probable que lors de l'optimisation de plusieurs milliers (ou millions) de variables, comme c'est le cas en déformable, apparaisse le problème de déséquilibre de la charge que nous avons évoqué (cf. le paragraphe avant la section 8.5). Pour pallier ce problème potentiel, la solution la plus simple est que L soit commun à tous les processeurs virtuels lorsque l'on engendre une population de descendants. Naturellement, la procédure que nous utilisons pour déterminer L reste inchangée : c'est un entier qui est tiré de façon à vérifier $P(L \geq \nu) = (p_c)^{\nu-1}$. La probabilité de croisement n'est d'autre part plus fixe tout au long de l'exécution de l'algorithme, mais est choisie pour chaque phase de reproduction dans l'intervalle $[p_{c,min}; p_{c,max}] \subset [0; 1]$. Cela permettra de limiter l'influence de ce paramètre dans le processus d'optimisation.

9.2.1 Recalage rigide, rigide avec zoom et affine

Les recalages préalables au recalage déformable sont effectués en séquence : la transformation trouvée pour un recalage sert pour initialiser la population de l'évolution différentielle parallèle devant effectuer le recalage suivant. Ainsi, le meilleur individu trouvé à l'issue de l'optimisation dans le cas du recalage rigide est conservé dans le population initiale pour le recalage rigide avec zoom, de même lors du passage du rigide avec zoom à l'affine. Les individus restants de $P(0)$ (la population initiale) sont eux engendrés aléatoirement dans un intervalle dont la taille est spécifiée par l'utilisateur et qui est centré sur le meilleur individu trouvé à la suite du recalage précédent. Dans le cas du recalage rigide il n'y a pas de meilleur individu au départ, on centre donc l'intervalle sur le vecteur nul.

Pour réaliser les recalages en pratique, les paramètres de l'évolution différentielle data-parallèle ont été fixés à partir des paramètres optimaux identifiés dans le chapitre précédent :

- une population de 16 individus ;
- γ et F ont pour valeur $0,525$;
- la probabilité de croisement vérifie $p_c \in [0,30; 0,80]$;
- le nombre t_{max} de populations engendrées est égal à 10 pour chaque recalage.

La valeur relativement faible de t_{max} vient du fait que dans le cas des transformations rigides, la translation et la rotation ne sont pas très importantes : l'objectif du recalage est de corriger la légère variation de la position spatiale des patients lors des prises de vue. Quand à la transformation affine, elle doit fournir rapidement une première estimation des déformations globales.

L'appréciation de la qualité du recalage affine final est uniquement visuelle car on ne connaît pas la transformation optimale que l'on recherche. La figure 9.3(b) présente l'image résultat que l'on obtient lors du recalage de l'image 9.3(a) sur l'image 9.5(i), tandis que 9.6(b) correspond au recalage affine sur la même image de référence (ou cible) de l'image 9.6(a) d'un autre patient (toutes ces images sont des IRM 128^3 voxels). Comme on peut le voir, le repositionnement de la tête semble correct, ce qui est également le cas des déformations globales si on compare le cervelet sur les différentes images. Une évaluation plus précise de la qualité pourrait être obtenue en moyennant l'intensité d'un grand nombre d'images recalées sur la même image cible, auquel cas la qualité du recalage pourrait être quantifiée suivant le flou de l'image moyenne. L'inconvénient c'est que cela nécessite d'avoir à disposition une base de données de plusieurs dizaines d'images. Notons que comme pour le recalage rigide dans le chapitre précédent, on peut également considérer des recalages synthétiques.

Toutefois, afin de mieux cerner le comportement de l'évolution différentielle parallèle, nous allons comparer le coût des transformations finales trouvées par cet algorithme lors de chaque recalage (rigide, rigide avec zoom, ...), avec le coût des transformations auxquelles aboutit une méthode d'optimisation locale. Cette dernière est en l'occurrence une méthode de quasi-Newton implantée dans MEDIMAX, le logiciel de traitement d'images médicales développé à l'Institut de Physique Biologique des Hôpitaux Universitaires de Strasbourg. C'est d'ailleurs grâce à ce logiciel que nous visualisons les images. Le tableau 9.1 donne le coût des diverses transformations trouvées par chaque algorithme, le constat est que l'évolution différentielle parallèle est relativement compétitive, avec un recalage affine nettement meilleur

Au niveau du temps de calcul, l'exécution sur 8 et 16 processeurs R10000 requiert en moyenne respectivement 5 minutes 34 secondes et 3 minutes 33 secondes. Bien entendu, lorsque l'on

Transformation recherchée	Evolution différentielle parallèle	Méthode d'optimisation locale (quasi-Newton)
Rigide	2069,21	2069,05
Rigide avec zoom	2060,54	2069,05
Affine	2037,32	2063,05

Tab. 9.1 – Illustration du coût de la transformation trouvée lors de chaque recalage par l'évolution différentielle parallèle, de même pour la méthode d'optimisation locale utilisée dans le logiciel MEDIMAX.

compare ces temps avec ceux que nous avions lors du recalage rigide multirésolution, pour un plus grand nombre de populations engendrées (tableau 8.8), il est évident que la suppression de l'approche multirésolution se traduit par un accroissement important du coût calculatoire.

9.2.2 Recalage déformable

De même que pour le recalage affine, aucune mesure ne permet de quantifier la qualité du recalage déformable. C'est pourquoi, à côté de la visualisation des images que nous obtenons aux différents niveaux de résolution et en l'absence d'images moyennes, nous prendrons comme référence pour évaluer la qualité du recalage, le coût obtenu par la procédure d'optimisation utilisée par Musse dans sa thèse [87], décrit à la section 6.2.5 (voir le tableau ci-dessous). Celle-ci a été validée sur une base de données de plusieurs dizaines d'images, ainsi que sur des déformations synthétiques. À remarquer que cette procédure d'optimisation étant déterministe, pour un même couple d'images, on obtient toujours le même résultat. Le temps de calcul sur une station de travail Hewlett-Packard 9000/C360 est de 22 minutes pour une optimisation menée jusqu'au dernier niveau de résolution.

Niveau de résolution	Procédure d'optimisation déterministe
1	2011,61
2	1843,76
3	1624,41
4	1415,57
5	1214,57

Tab. 9.2 – Erreurs quadratiques moyennes du recalage déformable de l'IRM 128^3 voxels 9.3(b) sur l'IRM de référence 9.5(i) obtenues par la procédure d'optimisation utilisée par Musse [87].

Lorsque l'on traite le recalage déformable avec l'évolution différentielle parallèle, il faut déterminer la stratégie que l'on va adopter pour la population vis-à-vis du changement de résolution, c'est-à-dire la façon dont la population sera initialisée à chaque niveau de résolution l. Évidemment, pour le premier niveau de résolution la question ne se pose pas : $P(0)$ est engendrée de façon totalement aléatoire, seules les bornes de l'espace de recherche sont spécifiées (soit des déformations de plus ou moins 10 voxels). Pour initialiser la population aux niveaux de résolution $l \geq 2$ plusieurs voies sont possibles :

- dans la continuité des recalages rigide, rigide avec zoom et affine, on peut ne conserver d'un niveau à un autre que le meilleur individu a_{min}. Les individus restants sont alors produits aléatoirement dans le voisinage de ce dernier ;
- une solution plus directe est de simplement propager toute la population finale trouvée à la résolution 2^l au niveau de résolution $l+1$.

Dans un premier temps, nous avons mis en œuvre la seconde stratégie. Cependant à la suite d'expérimentations préliminaires, il est rapidement apparu que celle-ci n'était pas du tout adaptée. En effet, la population ayant convergé au premier niveau de résolution, lorsqu'on la propage, la population initiale que l'on obtient pour le second niveau se trouve concentrée dans la même zone de l'espace de recherche. Or du fait de l'analyse multirésolution de la méthode de recalage déformable, à chaque niveau de résolution le paysage de la fonction de coût évolue et le meilleur individu au niveau de résolution précédent peut alors être relativement « éloigné » de la zone où se trouve l'optimum recherché à la résolution courante. C'est pourquoi, lors des premières générations l'algorithme va itérer afin de sortir de la zone où est concentrée la population. Le problème est alors que plus les individus sont rassemblés, plus la vitesse à laquelle on peut s'éloigner de cette partie de l'espace de recherche est lente. Par conséquent on s'est orienté vers la première stratégie : son inconvénient est cependant que, suivant la taille de l'intervalle de recherche autour de l'optimum courant, il faudra plus ou moins de temps à l'algorithme pour commencer à converger.

Nous avons exécuté l'algorithme parallèle en conservant globalement les mêmes paramètres que pour les recalages préalables. Les changements se situent au niveau des bornes de l'intervalle de la probabilité de croisement et de la valeur de t_{max}, ce dernier paramètre varie d'un niveau de résolution à un autre. Le tableau 9.3 regroupe les coûts les plus faibles auxquels on est parvenu avec les différents jeux de paramètres sur les deux premiers niveaux de résolution. Le couple d'images considéré est le même que celui pour lequel on a donné les coûts obtenus par la procédure d'optimisation déterministe.

Niveau de résolution	Nombre de générations t_{max}	Probabilité de croisement $p_c \in [0,30;0,55]$	Probabilité de croisement $p_c \in [0,55;0,80]$
1	15	2005,17	2006,13
	20	2004,73	
2	30	1898,55	1919,21
	40	1866,79	
	60	1840,42	

TAB. 9.3 – Meilleures erreurs quadratiques moyennes du recalage déformable de l'IRM 128^3 voxels 9.3(b) sur l'IRM de référence 9.5(i) obtenues par l'évolution différentielle parallèle.

La comparaison des valeurs des tableaux 9.2 et 9.3 montre que l'évolution différentielle permet d'aboutir à de meilleurs résultats que l'approche déterministe de Musse [87]. Le bémol vient du nombre de populations qui doivent être engendrées pour atteindre ces résultats. En effet, au premier niveau de résolution la quinzaine de populations qui est requise est acceptable, mais dès le second niveau on passe à une soixantaine de populations, soit une multiplication par un facteur quatre. On peut donc raisonnablement estimer que cette augmentation va se poursuivre,

voire s'amplifier lors du passage aux niveaux de résolutions suivants. Or si on regarde les temps de calcul sur 8 processeurs R10000 : de l'ordre de 21 minutes pour $l = 2$ et $t_{max} = 60$ (au niveau de résolution 1, $t_{max} = 15$), il est très clair que pour obtenir au cinquième niveau de résolution une image recalée de même qualité que celle produite par l'optimisation déterministe (voire meilleure), plusieurs heures de calculs seront nécessaires. D'ailleurs, au regard des 22 minutes de calcul sur la station Hewlett-Packard de la procédure d'optimisation déterministe pour réaliser l'optimisation jusqu'au niveau de résolution 5, l'évolution différentielle parallèle n'est assurément pas une alternative intéressante. En effet, même si les performances brutes du processeur HP PA-RISC 8500 de la station de travail (*benchmarks* : $SPECfp95 = 28,1$ et $SPECint95 = 26,0$) sont nettement meilleures que celles du MIPS R10000 ($SPECfp95 = 19,1$ et $SPECint95 = 11,4$) de l'Origin2000, le fait que l'évolution différentielle parallèle a besoin, pour mener à bon terme l'optimisation sur les deux premiers niveaux de résolution, du même temps que la procédure déterministe qui traite les 5 niveaux de résolution, laisse présager d'un temps d'exécution très mauvais pour $l = 5$. Remarquons que pour la probabilité de croisement, une plage de valeurs induisant la modification d'un trop grand nombre de variables ralentit également la convergence.

Ainsi, l'approche multirésolution du calcul des déformations permet à une approche déterministe d'être plutôt compétitive. Tandis que l'évolution différentielle souffre du grand nombre de variables à optimiser et surtout du problème de l'initialisation de la population lors du changement de résolution, qui comme on l'a vu affecte la vitesse de convergence. Bien sûr, nous avons examiné diverses solutions pour résoudre ce problème et notamment :

- l'ajout de quelques itérations d'une méthode d'optimisation locale afin de raffiner chaque descendant avant la sélection ;
- le raffinement de la population initiale, toujours par une méthode d'optimisation locale ;
- la réduction de la taille de l'intervalle de recherche lorsque la résolution croît, puisqu'au fur et à mesure que l augmente, les déformations sont de plus en plus proches de la valeur optimale ;

tout cela sans grand succès.

En fin de compte, le résultat est que l'évolution différentielle ne peut pas être retenue pour résoudre le problème du recalage déformable dans le cadre de la méthode considérée. Notamment en raison des temps d'exécution pour les résolutions les plus fines qui sont sans aucun doute rédhibitoires. De fait, comme on n'arrive pas à tirer parti de l'analyse multirésolution pour guider la recherche en passant d'un niveau de résolution à un autre, i.e. bien initialiser les différentes populations de départ, l'algorithme est long à converger. Du reste, on peut légitimement se demander, dans ce cadre, si l'optimisation directe du dernier niveau de résolution, sans passer par les résolutions précédentes, ne conduirait pas à des résultats comparables.

9.3 Équation de la diffusion

La mise en œuvre de l'approche massivement parallèle de l'équation de la diffusion est conditionnée par l'expression du gradient de la fonction d'énergie (de coût). Aussi, à l'instar du recalage rigide, nous allons tout d'abord calculer le gradient de l'erreur quadratique moyenne afin de vérifier la pertinence de cette approche. De fait, la parallélisation massive repose essentiellement sur le calcul en parallèle des dérivées partielles. Elle est donc d'autant plus intéressante que les calculs sont peu redondants d'un processeur à un autre.

9.3.1 Gradient de l'erreur quadratique moyenne

Voyons le détail du calcul pour la composante correspondant à la variable $a^l_{x_{i,j,k}}$, où i,j,k sont dans $\{0,...,m_l-1\}$ (voir (6.23)).

On rappelle qu'à partir de la fonction de coût correspondant à l'erreur quadratique (6.4), donnée dans la section 6.2, on détermine l'expression de l'erreur quadratique moyenne :

$$C(d) = \frac{1}{|\Omega|}\int_\Omega |I(s) - J(s+d(s))|^2 ds, \tag{9.1}$$

où Ω est le support des images ; $I(.)$ est l'image de référence et $J(.)$ est l'image devant être recalée ; s définit la position d'un voxel dans le domaine Ω.

La dérivée partielle de C au niveau de résolution l par rapport à la variable $a^l_{x_{i,j,k}}$ vaut alors :

$$\frac{\partial C\left(d^l(s)\right)}{\partial a^l_{x_{i,j,k}}} = \frac{1}{|\Omega|}\int_\Omega 2\left(J(s+d^l(s)) - I(s)\right)\frac{\partial\left(J\left(s+d^l(s)\right)\right)}{\partial a^l_{x_{i,j,k}}} ds \tag{9.2}$$

où $s=(x,y,z)^T$ et $d^l(s) = (d^l_x(s), d^l_y(s), d^l_z(s))^T$ (cf. (6.20)).

On introduit ensuite $s' = (x',y',z')^T$, l'image de s par la transformation T_{d^l} (6.6), pour aboutir à l'expression suivante :

$$\frac{\partial C\left(d^l(s)\right)}{\partial a^l_{x_{i,j,k}}} = \frac{1}{|\Omega|}\int_\Omega 2(J(s')-I(s))\left(\nabla_{s'}J(s')\bullet\left(\frac{\partial\left(x+d^l_x(s)\right)}{\partial a^l_{x_{i,j,k}}}, \frac{\partial\left(y+d^l_y(s)\right)}{\partial a^l_{x_{i,j,k}}}, \frac{\partial\left(y+d^l_y(s)\right)}{\partial a^l_{x_{i,j,k}}}\right)^T\right) ds \tag{9.3}$$

En utilisant (6.20) on obtient une simplification :

$$\begin{aligned}\frac{\partial C\left(d^l(s)\right)}{\partial a^l_{x_{i,j,k}}} &= \frac{1}{|\Omega|}\int_\Omega 2(J(s')-I(s))\left(\nabla_{s'}J(s')\bullet\left(\phi^l_{i,j,k}(s),0,0\right)^T\right) ds &(9.4)\\ &= \frac{1}{|\Omega|}\int_\Omega 2(J(s')-I(s))\frac{\partial J(s')}{\partial x'}\phi^l_{i,j,k}(s) ds &(9.5)\end{aligned}$$

Or on a vu dans la section 6.2.5 que chacune des fonctions de base $\phi^l_{i,j,k}$, où i,j,k appartiennent à l'ensemble $\{0,...,m_l-1\}$ a pour support $\Omega_{\phi^l_{i,j,k}} \subset \Omega$. En conséquence la composante du gradient qui correspond à la variable $a^l_{x_{i,j,k}}$ est en définitive égale à :

$$\frac{\partial C\left(d^l(s)\right)}{\partial a^l_{x_{i,j,k}}} = \frac{1}{|\Omega|}\int_{\Omega_{\phi^l_{i,j,k}}} 2(J(s')-I(s))\frac{\partial J(s')}{\partial x'}\phi^l_{i,j,k}(s) ds \tag{9.6}$$

Finalement, on en déduit les matrices des dérivées partielles de l'erreur quadratique moyenne

par rapport aux variables des matrices A_x^l, A_y^l et A_z^l :

$$\left(\frac{\partial C\left(d^l(s)\right)}{\partial a^l_{x_{i,j,k}}}\right)_{i,j,k\in\{0,\ldots,m_l-1\}} = \left(\frac{1}{|\Omega|}\int_{\Omega_{\phi^l_{i,j,k}}} 2(J(s')-I(s))\frac{\partial J(s')}{\partial x'}\phi^l_{i,j,k}(s)ds\right)_{i,j,k\in\{0,\ldots,m_l-1\}} \quad (9.7)$$

$$\left(\frac{\partial C\left(d^l(s)\right)}{\partial a^l_{y_{i,j,k}}}\right)_{i,j,k\in\{0,\ldots,m_l-1\}} = \left(\frac{1}{|\Omega|}\int_{\Omega_{\phi^l_{i,j,k}}} 2(J(s')-I(s))\frac{\partial J(s')}{\partial y'}\phi^l_{i,j,k}(s)ds\right)_{i,j,k\in\{0,\ldots,m_l-1\}} \quad (9.8)$$

$$\left(\frac{\partial C\left(d^l(s)\right)}{\partial a^l_{z_{i,j,k}}}\right)_{i,j,k\in\{0,\ldots,m_l-1\}} = \left(\frac{1}{|\Omega|}\int_{\Omega_{\phi^l_{i,j,k}}} 2(J(s')-I(s))\frac{\partial J(s')}{\partial z'}\phi^l_{i,j,k}(s)ds\right)_{i,j,k\in\{0,\ldots,m_l-1\}} \quad (9.9)$$

9.3.2 Parallélisation massive

Le calcul des dérivées partielles de l'erreur quadratique moyenne montre de manière évidente la pertinence de la parallélisation massive de l'équation de la diffusion (décrite à la section 1.3.2). En effet, comme on peut le voir à partir des équations (9.7), (9.8) et (9.9), l'idée est de distribuer le calcul des dérivées suivant les domaines $\Omega_{\phi^l_{i,j,k}}$. Ainsi chaque triplet de dérivées partielles $\left(\frac{\partial C\left(d^l(s)\right)}{\partial a^l_{x_{i,j,k}}}, \frac{\partial C\left(d^l(s)\right)}{\partial a^l_{y_{i,j,k}}}, \frac{\partial C\left(d^l(s)\right)}{\partial a^l_{z_{i,j,k}}}\right)$ où $i,j,k \in \{0,\ldots,m_l-1\}$ pourra être calculé sur le même processeur.

L'intérêt de cette parallélisation vient surtout du fait que le calcul de la différence entre l'image recalée et l'image cible (terme $2(J(s')-I(s))$), de même que le calcul du gradient de l'image recalée (terme $\nabla_{s'}J(s')$), qui sont les opérations les plus coûteuses, peuvent être effectuées en parallèle. Comme le montre la figure 9.1 pour le second niveau de résolution, le gain escompté est intéressant, puisque lorsque l'on ne dispose que d'un processeur, ces calculs sont faits de façon optimale en considérant tout le domaine Ω (qui correspond au volume d'une image), alors qu'en disposant de plusieurs processeurs on peut distribuer ces calculs par l'intermédiaire du partitionnement de Ω en sous-domaines $\Omega_{\phi^l_{i,j,k}}$.

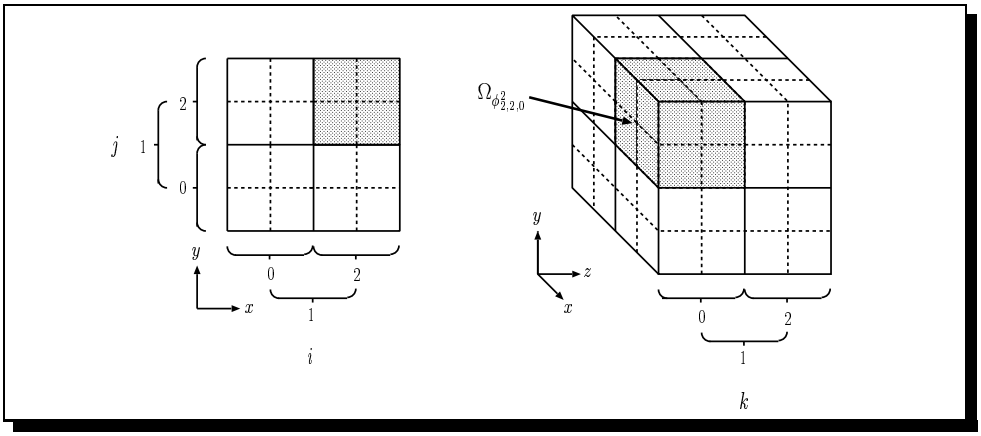


FIG. 9.1 – Découpage de Ω en domaines $\Omega_{\phi^2_{i,j,k}}$ où $i,j,k \in \{0,1,2\}$, obtenu en prenant pour fonction d'échelle la $\beta-$spline ϕ_1.

Notons cependant que les sous-domaines ne sont pas tous disjoints car les fonctions de base $\phi^l_{i,j,k}$ se chevauchent (cf. figure 6.3), par exemple $\Omega_{\phi^2_{2,2,0}}$ et les sous-domaines $\Omega_{\phi^2_{1,1,0}}, \Omega_{\phi^2_{1,1,1}}, \Omega_{\phi^2_{1,2,0}}$, $\Omega_{\phi^2_{1,2,1}}, \Omega_{\phi^2_{2,1,0}}, \Omega_{\phi^2_{2,1,1}}, \Omega_{\phi^2_{2,2,1}}$ ont une intersection non nulle. Par conséquent il y a des calculs redondants sur les images (différence et gradient), mais comme au fur et à mesure que l croît le nombre de sous-domaines devient de plus en plus important, en dépit de la redondance de certains calculs la parallélisation massive est d'autant plus intéressante. Ceci est indéniable au regard du tableau 9.4 qui donne pour des IRM 128^3 et 256^3 voxels, le nombre de processeurs pouvant en théorie évoluer en parallèle à chaque niveau de résolution et le nombre effectif en pratique lors d'un recalage déformable. En fait, le nombre potentiel de processeurs qui peuvent être actifs à la résolution 2^l correspond à la dimension des matrices A^l_x, A^l_y et A^l_z, soit ${m_l}^3$. En pratique, on réduit le nombre total de variables à optimiser aux résolutions élevées en supprimant les sous-domaines $\Omega_{\phi^l_{i,j,k}}$ qui ne contiennent que du fond dans les deux images à recaler, d'où un nombre réel de processeurs égal à la division du nombre de variables réduit par 3. En effet, l'acquisition d'une IRM 3D de la tête d'un patient se traduit par un volume dont seul 20 à 30% des voxels portent de l'information. Cela signifie également que le nombre réel de processeurs dépend du couple d'images que l'on recale entre elles et varie légèrement d'une exécution à une autre car les déformations que l'on trouve ne sont pas toujours les mêmes.

	Théorie Sans réduction		Pratique			
			IRM 128^3		IRM 256^3	
l	Variables	Processeurs	Variables	Processeurs	Variables	Processeurs
1	3	1	3	1	3	1
2	81	27	81	27	81	27
3	1029	343	960	320	930	310
4	10125	3375	6057	2019	5058	1686
5	89373	29791	38121	12707	29346	9782
6	750141	250047	263619	87873	198015	66005
7	6145149	2048383			1436451	478817

Tab. 9.4 – Nombre de processeurs pouvant être actifs simultanément suivant le niveau de résolution l en théorie, i.e. sans réduction, et en pratique lors du recalage d'un couple d'images : 9.3(b) sur 9.5(i) dans le cas 128^3 ; 9.9(b) sur 9.9(a) dans le cas 256^3.

Idéalement, l'architecture parallèle la plus adéquate pour implanter l'équation de la diffusion dans le cadre du problème de recalage déformable est une machine SIMD comportant un nombre important de processeurs. Mais la tendance actuelle étant aux machines MIMD, comme l'Origin2000, nous allons en fait étudier le comportement d'un algorithme massivement parallèle sur ce type de machine.

Pour conclure, on donne à la figure 9.2, l'algorithme parallèle, sans réduction des paramètres, exécuté à chaque niveau de résolution. La fonction d'énergie E représente l'erreur quadratique moyenne ($E(x_t) = C\left(d^l(s)\right)$) ; les variables du modèle déformable, i.e. les composantes des matrices A^l_x, A^l_y et A^l_z sont « rangées » dans la configuration $X_t = x_t \left(= \left(x_{t,1}, x_{t,2}, ..., x_{t,3\cdot {m_l}^3}\right)\right)$

de la manière suivante :

$$x_{t,3\cdot(k\cdot {m_l}^2+j\cdot m_l+i)+1} = a^l_{x_{i,j,k}} \quad (9.10)$$

$$x_{t,3\cdot(k\cdot {m_l}^2+j\cdot m_l+i)+2} = a^l_{y_{i,j,k}} \quad (9.11)$$

$$x_{t,3\cdot(k\cdot {m_l}^2+j\cdot m_l+i)+3} = a^l_{z_{i,j,k}} \quad (9.12)$$

où $i, j, k \in \{0, ..., m_l - 1\}$.

$t = 0$ — *- Initialisation du temps*
$T_t \leftarrow T_0$ — *- Température initiale*
$\Delta P_t \leftarrow \Delta P_0$ — *- Pas de temps initial*
$X_t \leftarrow X_0$ — *- Configuration de départ, générée aléatoirement ou bien résultat de l'optimisation au niveau de résolution précédent*
Répéter
  Pour tout processeur $P_u, u \in \{1, ..., {m_l}^3\}$
    $\Delta w_{t,3\cdot(u-1)+1} \leftarrow \mathcal{N}(0, \Delta P_t)$
    $x_{t+\Delta P_t,3\cdot(u-1)+1} \leftarrow x_{t,3\cdot(u-1)+1} - \frac{\partial E(x_t)}{\partial x_{t,3\cdot(u-1)+1}} \cdot \Delta P_t + \sqrt{2 \cdot T_t} \cdot \Delta w_{t,3\cdot(u-1)+1}$ — *- Mouvement brownien*
    $\Delta w_{t,3\cdot(u-1)+2} \leftarrow \mathcal{N}(0, \Delta P_t)$
    $x_{t+\Delta P_t,3\cdot(u-1)+2} \leftarrow x_{t,3\cdot(u-1)+2} - \frac{\partial E(x_t)}{\partial x_{t,3\cdot(u-1)+2}} \cdot \Delta P_t + \sqrt{2 \cdot T_t} \cdot \Delta w_{t,3\cdot(u-1)+2}$ — *- Mouvement brownien*
    $\Delta w_{t,3\cdot(u-1)+3} \leftarrow \mathcal{N}(0, \Delta P_t)$
    $x_{t+\Delta P_t,3\cdot(u-1)+3} \leftarrow x_{t,3\cdot(u-1)+3} - \frac{\partial E(x_t)}{\partial x_{t,3\cdot(u-1)+3}} \cdot \Delta P_t + \sqrt{2 \cdot T_t} \cdot \Delta w_{t,3\cdot(u-1)+3}$ — *- Mouvement brownien*
  Fin pour
  $T_{t+\Delta P_t} \leftarrow g(T_t)$ — *- g, h : deux fonctions dont la décroissance est stricte*
  $\Delta P_{t+\Delta P_t} \leftarrow h(\Delta P_t)$
  $t \leftarrow t + \Delta P_t$
Jusqu'à *critère d'arrêt vérifié*

Fig. 9.2 – Algorithme massivement parallèle de l'équation de la diffusion pour le recalage déformable au niveau de résolution l.

9.3.3 Implantation en OpenMP

Le choix d'OpenMP comme paradigme de programmation parallèle nous permet de nous abstraire du problème du placement des données sur les différents processeurs, notamment pour ce qui est des images. Plus généralement, toutes les communications induites sont implicitement gérées par le compilateur en tirant parti de la mémoire partagée de l'Origin2000. L'algorithme de la figure 9.2 s'implante aisément par l'intermédiaire d'une région parallèle et d'une boucle partagée. En effet, comme on ne dispose pas toujours de ${m_l}^3$ processeurs, entre autre pour les niveaux de résolution $l \geq 3$ (${m_l}^3$ va de quelques centaines à plusieurs milliers, voire millions ; cf. tableaux 9.4), l'exécution simultanée des processeurs sera modélisée par une boucle `DO - END DO` dont les itérations seront partagées. Chaque processeur physique n'exécute alors qu'un sous-ensemble des ${m_l}^3$ itérations au sein de la région parallèle, la répartition de la boucle peut être contrôlée par un type : `SCHEDULE(STATIC<,`*chunk*`>)`, `SCHEDULE(DYNAMIC<,`*chunk*`>)` et `SCHEDULE(GUIDED<,`*chunk*`>)`. Ainsi, ce sont des paquets d'itérations successives qui sont assignées à chacun des processeurs, paquets dont la taille est soit fixe (l'entier *chunk* donné par

l'utilisateur ; la taille peut également être prédéfinie ou calculée automatiquement à partir du nombre de processeurs en l'absence d'indication), soit décroît exponentiellement au fil des paquets (dans ce cas le *chunk* est la borne inférieure de la taille des paquets, le dernier est éventuellement plus petit). D'autre part, l'assignation peut être statique ou dynamique (y compris pour la répartition `GUIDED`), i.e. tous les paquets sont immédiatement affectés suivant un algorithme de type *round-robin* (affectation cyclique) ou bien chaque processeur prend le premier paquet disponible.

Pour le recalage déformable, nous avons opté pour une assignation statique des paquets, sans leur donner de taille puisque le nombre total d'itérations ${m_l}^3$ varie avec le niveau de résolution et les images. En effet, une assignation dynamique n'est intéressante que lorsqu'il y a un déséquilibre de la charge au niveau des paquets. Le déséquilibre éventuel de la charge du calculateur n'est pas pris en compte car l'algorithme n'est exécuté que si on dispose du nombre de processeurs physiques demandé, de plus ceux-ci ne sont pas partagés avec un autre algorithme. L'assignation dynamique est *a priori* intéressante dans le cas d'une exécution sur des processeurs hétérogènes.

9.3.4 Paramètres

En ce qui concerne les paramètres de l'équation de la diffusion lors de l'optimisation d'un niveau de résolution l, à savoir le critère d'arrêt, le pas de temps initial et la température initiale, nous avons suivi la même approche que pour le recalage rigide (voir la section 8.2.2) :

- L'algorithme s'arrête lorsqu'un total de Max_{conf} configurations a été engendré.
- Nous fixons un pas de temps ΔP à partir duquel, en décidant que le dernier pas de temps utilisé pour engendrer une configuration est égal à 10^{-5}, on calcule à la fois le coefficient α_P assurant la décroissance du pas et le pas initial. Toutefois, comme la valeur des dérivées partielles va diminuer à mesure que la résolution augmente, nous allons faire croître la valeur de ΔP avec l, afin que l'on ait toujours un terme $\delta_1(t) = -\nabla E(x_t) \cdot dt$ (section 1.3.2) suffisant pour perturber la configuration courante. C'est pourquoi, pour compenser la baisse du coefficient de réduction du pas qui en résulte, les calculs seront faits comme si un total de $l \cdot Max_{conf}$ configurations était engendré à chaque niveau de résolution :

$$\alpha_P = \left(\frac{10^{-5}}{\Delta P}\right)^{\frac{1}{l \cdot Max_{conf}}} \Rightarrow \Delta P_0 = \alpha_P \cdot \Delta P \tag{9.13}$$

- Pour le schéma de température, la démarche est la même que pour le pas de temps, c'est-à-dire que l'on considère que la dernière température utilisée pour engendrer une configuration est 10^{-7}, puis en fixant la température T que l'on suppose précéder T_0 dans le schéma de température, on détermine α_T et T_0 :

$$\alpha_T = \left(\frac{10^{-7}}{T}\right)^{\frac{1}{Max_{conf}}} \Rightarrow T_0 = \alpha_T \cdot T \tag{9.14}$$

Le remplacement de l'écart type de la distribution multinormale (simulation du mouvement brownien), originellement le pas, par un multiple de la température courante a aussi été conservé.

Le tableau ci-après donne suivant le niveau de résolution, les valeurs que nous avons considérées pour la température T et le pas de temps, Max_{conf} est lui toujours égal à 10. Comme nous le verrons, ce jeu de paramètres qui est issu d'expérimentations intensives a permis d'obtenir

de bons résultats pour différents couples d'images à recaler entre elles. Néanmoins, il n'est sans doute pas adapté à toutes les paires d'images, en particulier lorsque les déformations sont très importantes (par exemple, la valeur initiale du pas et son évolution doivent être définis à partir de la valeur des dérivées partielles).

Niveau de résolution	T	α_T	ΔP	α_P
1	100	0,126	0,0001	0,794
2	100	0,126	0,0005	0,822
3	10	0,158	0,001	0,858
4	10	0,158	0,005	0,856
5	1	0,200	0,01	0,871
6	1	0,200	0,05	0,868
7	0,1	0,251	0,1	0,877

TAB. 9.5 – Jeu de paramètres utilisé pour le recalage déformable avec l'algorithme parallèle de l'équation de la diffusion.

9.3.5 Performances et analyse de l'algorithme parallèle

Qualité des recalages

Pour évaluer la qualité des recalages déformables que produit la version parallèle de l'équation de la diffusion, nous avons suivi la même méthodologie que dans le cas de l'évolution différentielle parallèle (section 9.2.2) : validation visuelle de la qualité des images recalées et comparaison avec la procédure d'optimisation déterministe introduite par Musse dans sa thèse [87] (uniquement pour des IRM 128^3 voxels car il n'a pas traité d'images de taille 256^3 voxels). Les résultats que nous présentons résultent de l'exécution de l'algorithme parallèle sur 16 processeurs R12000.

➔ *IRM* 128^3 *voxels*

Dans le cas des IRM 128^3 voxels nous avons recalé successivement deux couples d'images.

Les images recalées obtenues aux différents niveaux de résolution lors du recalage de l'image 9.3(b) (résultat du recalage affine de 9.3(a)) sur l'image 9.5(i) sont visibles pour les niveaux de résolution : 1 et 2 à la figure 9.3 ; 3,4 et 5 en 9.4 ; 6 à la figure 9.5. L'évolution du visage, en particulier du nez et du menton, montre très clairement que la localité des déformations croît avec le niveau de résolution. La figure 9.5 permet également de comparer l'image recalée à la résolution la plus fine avec l'image de référence et l'image de départ, i.e. sans les recalages préalables (rigide, rigide plus zoom et affine). L'excellente qualité du recalage est manifeste, en particulier au niveau des ventricules, du tronc cérébral et du cervelet. De plus, le calcul sur une vingtaine d'exécution de l'algorithme, de la moyenne de l'erreur quadratique moyenne à chaque niveau de résolution, donnée dans le tableau 9.6, permet de dire que la parallélisation massive de l'équation de la diffusion est très compétitive par rapport à la procédure d'optimisation utilisée par Musse (voir le tableau 9.2). L'image recalée à laquelle on arrive avec l'équation de la diffusion est suivant les niveaux de résolution sur le plan de la qualité : du même ordre (niveaux 1 et 4), légèrement meilleure (niveaux 2 et 5), voire très nettement supérieure (niveau 3).

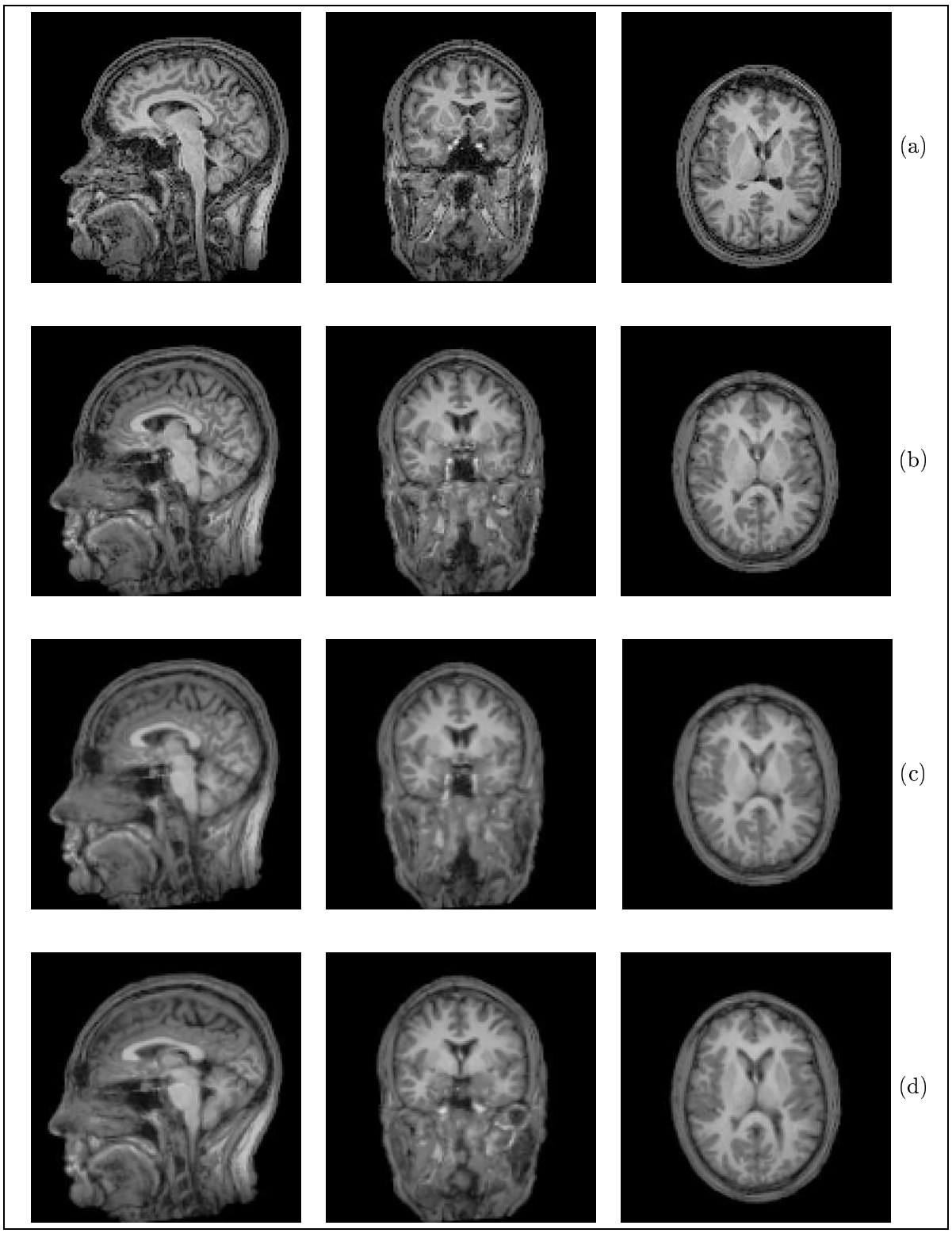


Fig. 9.3 – Recalage déformable d'une image source (a) sur l'image de référence 9.5(i). Les résultats sont présentés après recalage affine (b), recalage non rigide à la résolution 1 (c) et 2 (d).

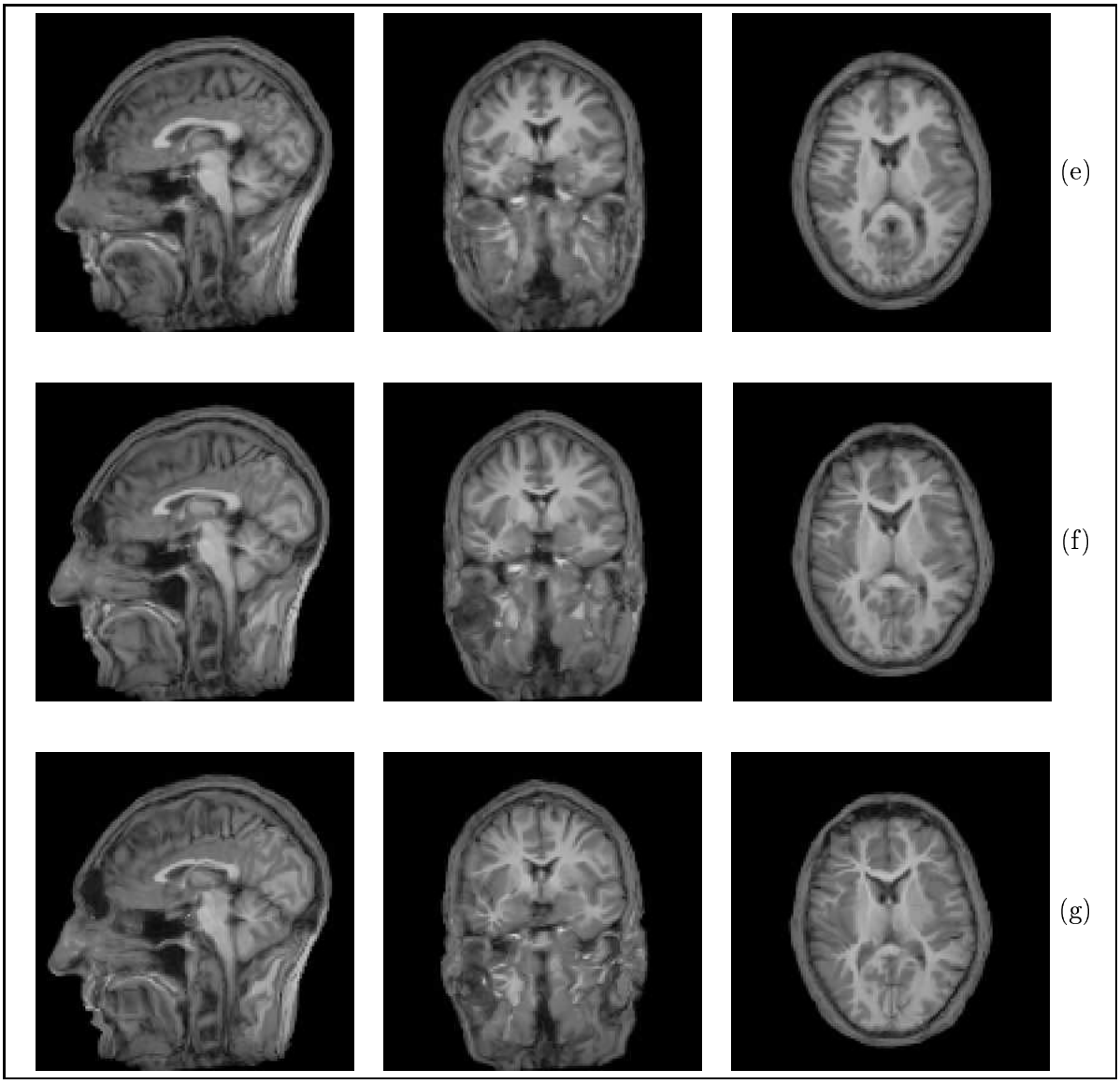


Fig. 9.4 – Suite de la figure 9.3 - résultat après recalage non rigide à la résolution 3 (e), 4 (f) et 5 (g).

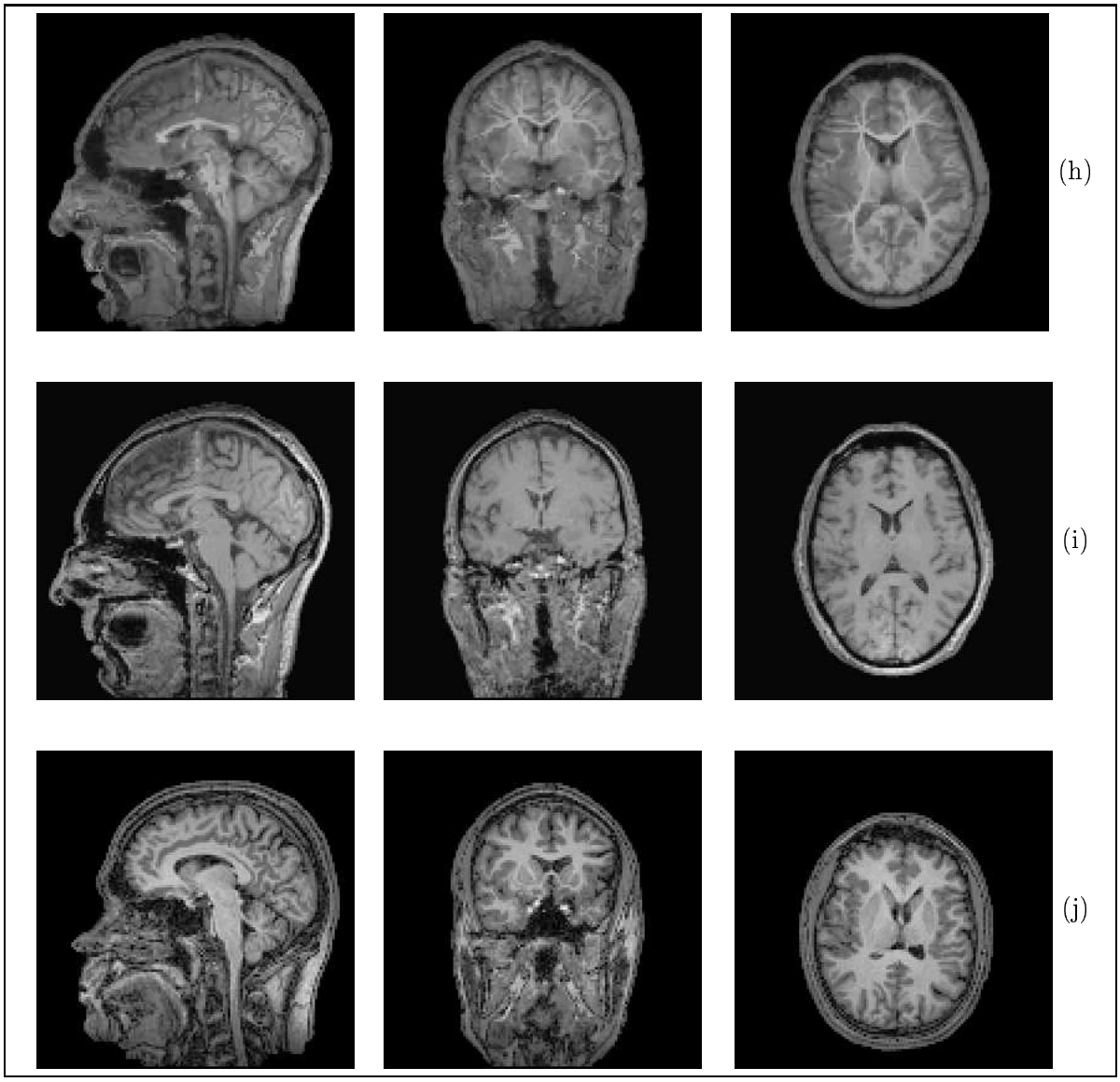


Fig. 9.5 – Suite de la figure 9.4 - recalage déformable hiérarchique d'une image source (j) sur l'image de référence (i). Résultat après recalage non rigide à la résolution 6 (h).

Niveau de résolution	IRM 128^3	IRM 256^3
1	2011,82	2163,41
2	1832,44	1613,71
3	1595,51	1273,40
4	1418,80	1089,28
5	1206,44	876,37
6	1038,03	709,46
7		580,74

TAB. 9.6 – Moyenne sur 20 exécutions de l'erreur quadratique moyenne du recalage déformable d'un couple d'images : 9.3(b) sur 9.5(i) dans le cas 128^3 ; 9.9(b) sur 9.9(a) dans le cas 256^3.

Le second couple d'images que nous avons traité, présenté figures 9.6, 9.7 et 9.8, est un peu particulier. En effet, comme on peut le voir, l'image source est « incomplète » dans le sens où il manque une partie du bas de la tête du patient. Aussi, lors du recalage déformable de l'image 9.6(b) sur l'image de référence 9.8(i) on observe un comportement singulier de la méthode de recalage qui est en quelque sorte poussée à ses limites : par exemple le cervelet subit des déformations importantes qui aboutissent à sa scission en différentes parties. L'image recalée que l'on obtient au dernier niveau de résolution (figure 9.8), bien que très proche de l'image de référence, montre que la transformation finale estimée ne conserve pas la topologie des structures anatomiques ce qui est un problème. À souligner que ce problème est commun à toutes les méthodes de recalage déformable et que l'intérêt qui lui est porté est assez récent. Notons que du fait de l'absence de certaines parties de l'image source, les deux images n'étaient de toute façon pas « topologiquement » équivalentes. En plus de la scission du cervelet, il est évident que d'autres parties du système nerveux central n'ont pas conservé leur topologie. La coupe frontale (chaque triplet d'images présente de gauche à droite une coupe sagitale, axiale et frontale) met ainsi au jour la fusion d'une partie du télencéphale avec le crâne.

➔ *IRM* 256^3 *voxels*

Pour évaluer l'équation de la diffusion parallèle dans le cas d'images 256^3 voxels, nous avons disposé d'une seule paire d'images. La figure 9.9 présente suivant une vision tridimensionnelle l'image de référence (9.9(a)), l'image à recaler (9.9(b)) et les images recalées résultant de l'optimisation lorsque celle-ci est menée jusqu'aux niveaux de résolution 6 (9.9(c)) et 7 (9.9(d)). Mentionnons que l'image source 9.9(b) ne subit pas de recalages préalables et que le point de vue des images est variable. De plus, aucune vision multiplanaire ne permet d'apprécier les déformations internes, néanmoins il ne fait aucun doute que celles-ci sont du même type que dans le cas de la seconde paire d'images 128^3 voxels (la tête dans l'image 9.9(b) n'est pas complète). Extérieurement, on peut dire que le traitement du septième niveau de résolution aboutit à une image recalée où la tête a un volume et une apparence quasiment similaires à l'image de référence. Le bon comportement de l'algorithme parallèle de l'équation de la diffusion est d'autre part confirmé par l'évolution moyenne sur 20 exécutions de l'erreur quadratique moyenne (cf. tableau 9.6).

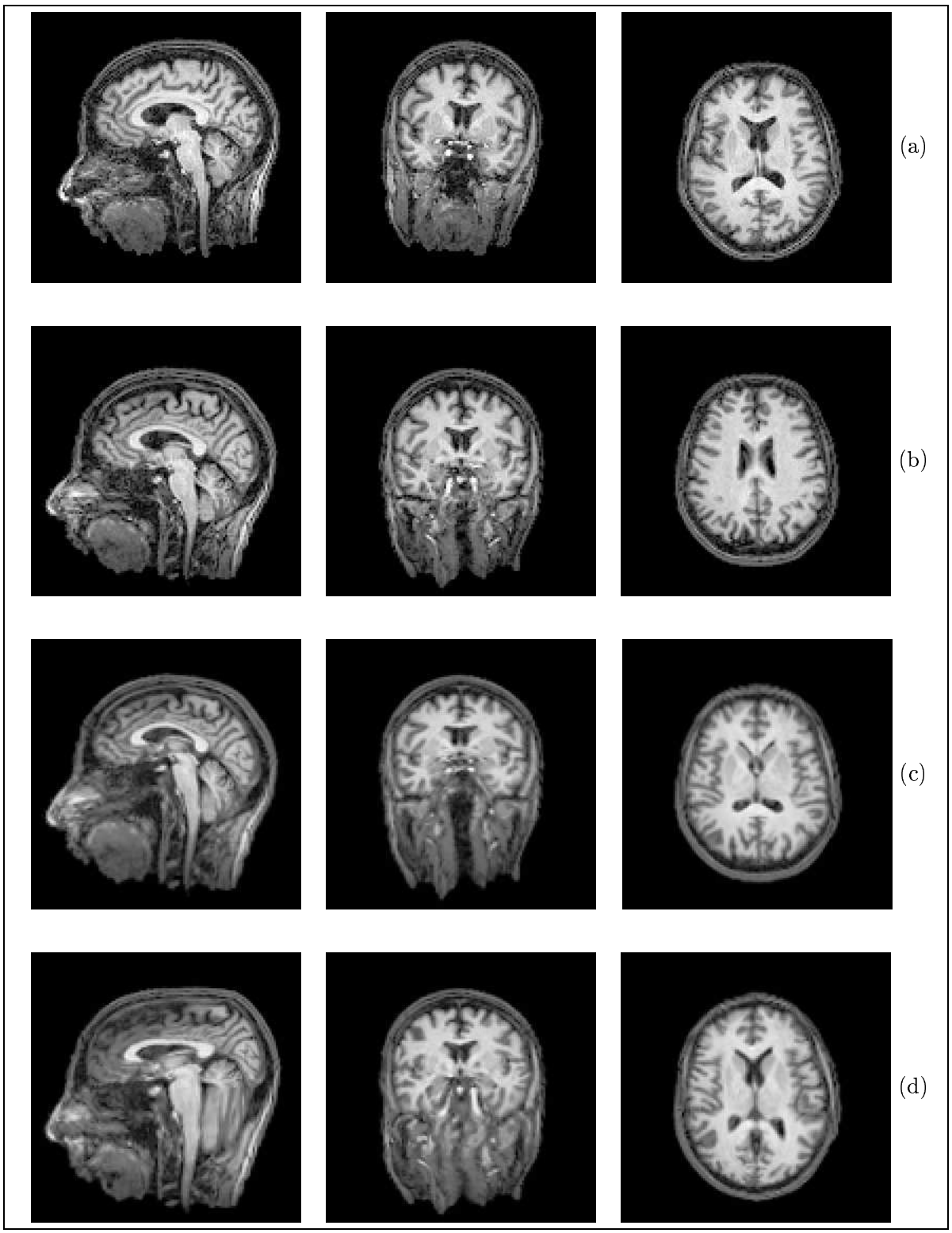


Fig. 9.6 – Recalage déformable d'une image source (a) sur l'image de référence 9.8(i). Les résultats sont présentés après recalage affine (b), recalage non rigide à la résolution 1 (c) et 2 (d).

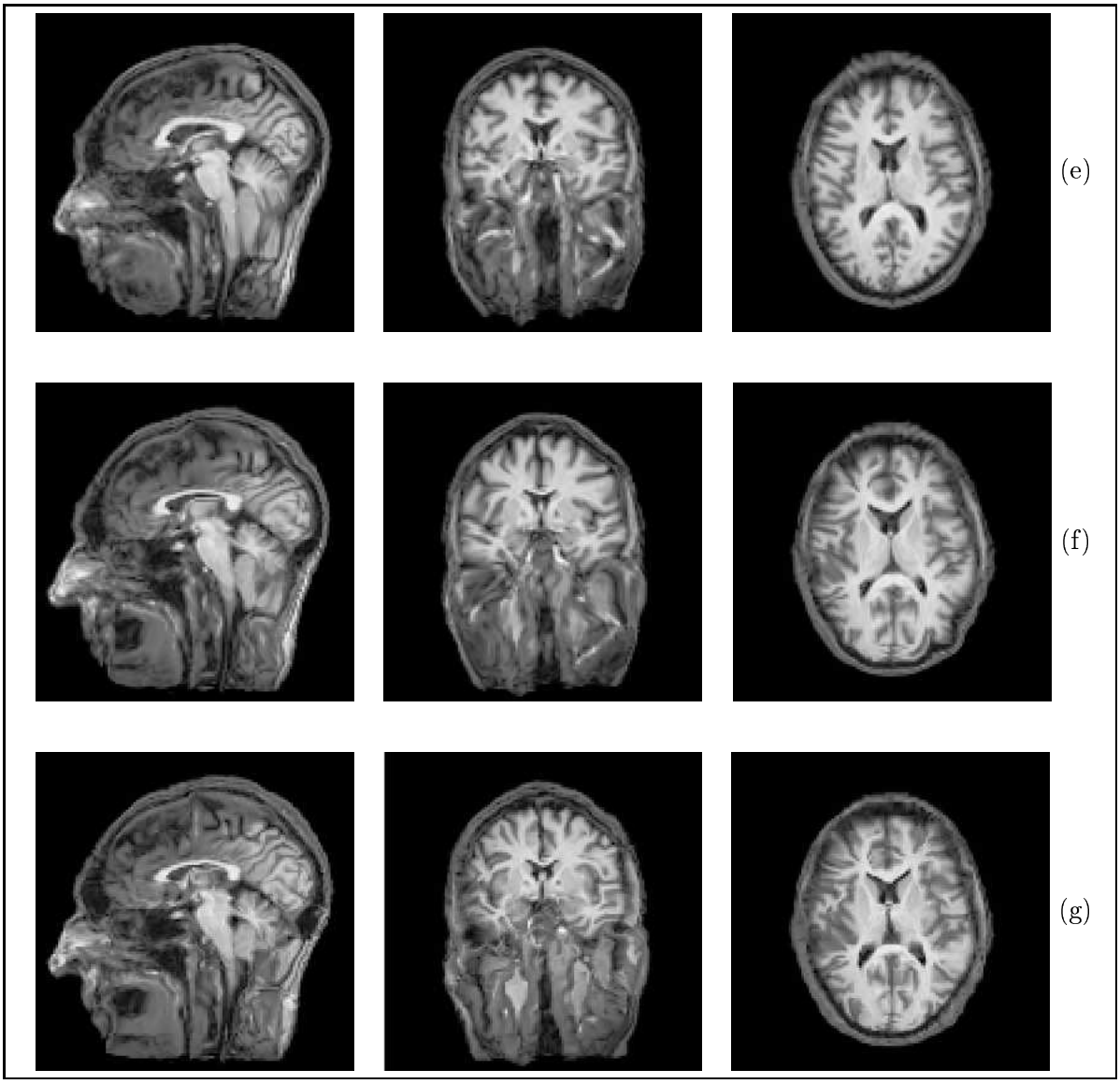


Fig. 9.7 – Suite de la figure 9.6 - résultat après recalage non rigide à la résolution 3 (e), 4 (f) et 5 (g).

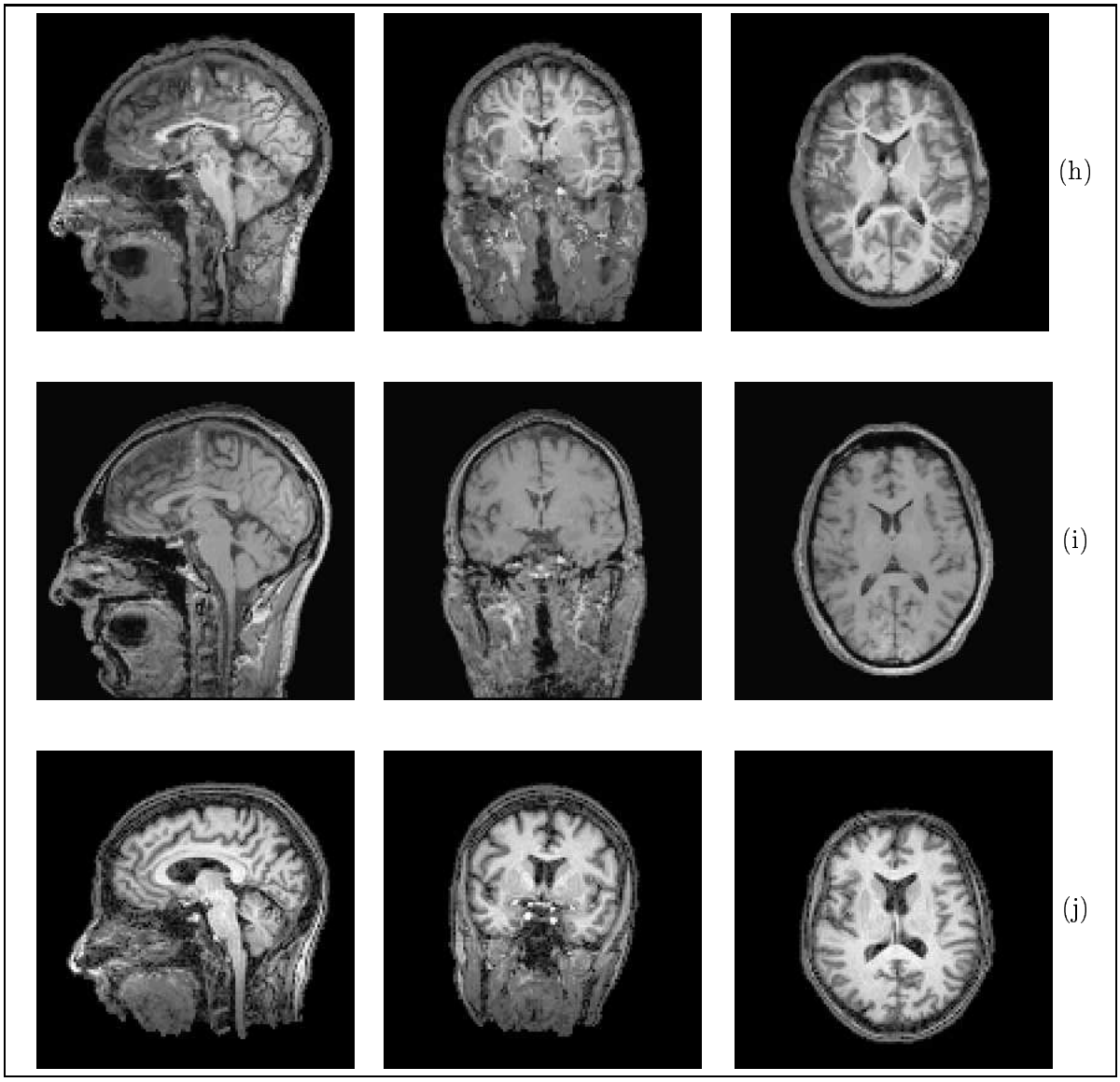


Fig. 9.8 – Suite de la figure 9.7 - recalage déformable hiérarchique d'une image source (j) sur l'image de référence (i). Résultat après recalage non rigide à la résolution 6 (h).

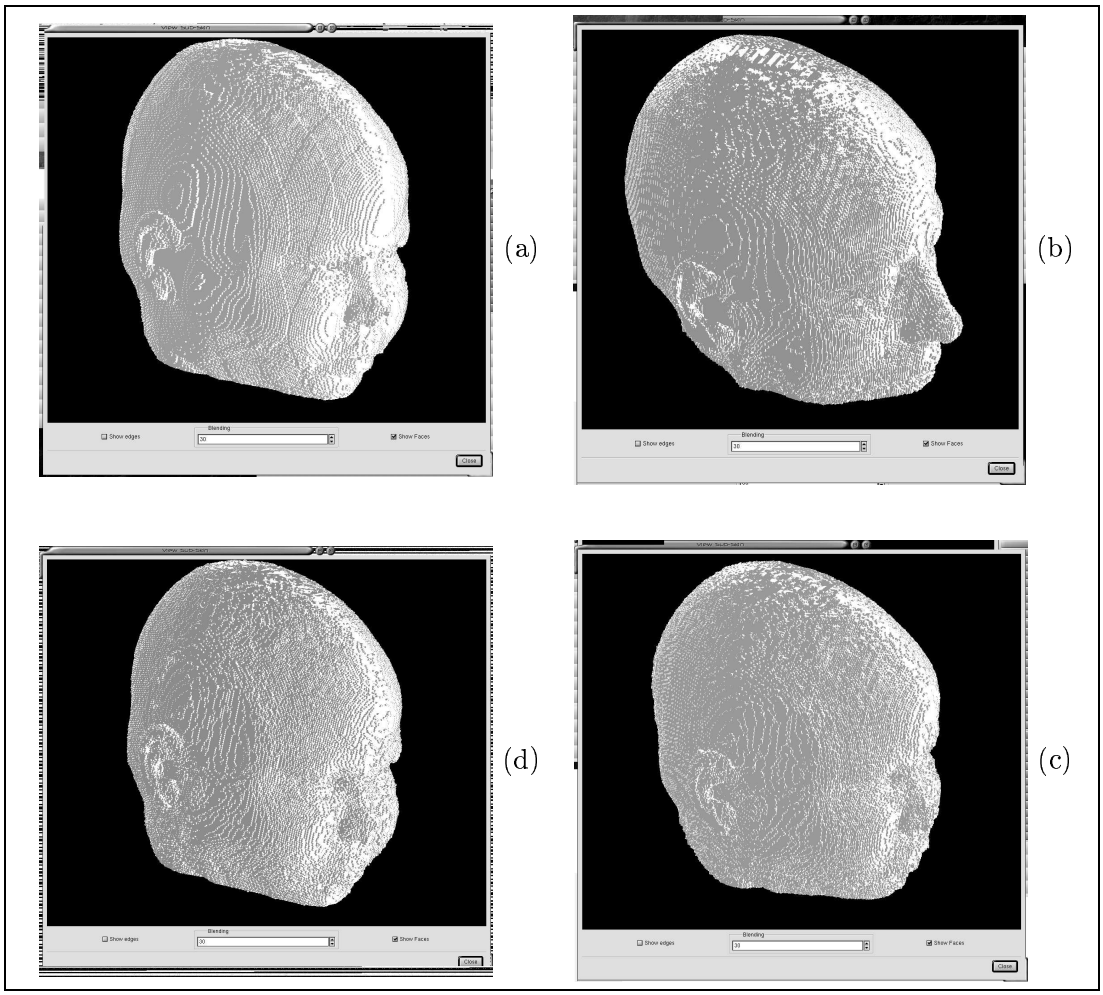


Fig. 9.9 – Recalage déformable d'une image source (b) sur l'image de référence (a), les images (c) et (d) correspondent respectivement au résultat après recalage non rigide aux niveaux de résolution 6 et 7 (vision tridimensionnelle).

Temps de calcul

L'algorithme de recalage déformable (figure 6.1), avec l'algorithme parallèle de l'équation de la diffusion comme méthode d'optimisation à chaque niveau de résolution, a été exécuté pour les deux tailles d'images sur différents nombres de processeurs. Le tableau 9.7 présente ainsi les temps de calcul que l'on obtient dans le cas d'une paire d'IRM 128^3 voxels lorsque le recalage est mené jusqu'au cinquième niveau de résolution, tandis que pour des images de taille 256^3 voxels, les temps sont donnés pour un niveau de résolution finale égal à 5, 6 et 7. Le nombre de processeurs que l'on a considéré fut successivement de 1, 8 et 16 pour les deux types de processeurs, plus pour les IRM 128^3 une exécution sur 24 et 32 processeurs R10000.

On constate que le recalage de deux IRM 128^3 nécessite sur un processeur R10000 environ 35 minutes contre 24 minutes sur R12000, ce qui est compétitif comparé aux 22 minutes de la procédure d'optimisation déterministe sur une station de travail dont le processeur est bien plus puissant que ceux de l'Origin2000. Le traitement des images 256^3, comme on s'y attendait, requiert plusieurs heures de calcul. Le temps d'exécution est dans le cas le plus favorable, i.e. pour un processeur R12000, respectivement de l'ordre de 3 heures, 3 heures 30 minutes et un peu moins de 4 heures pour les niveaux de résolution 5, 6 et 7. Quand à utiliser un R10000, cela se traduit au minimum par un surcoût d'1 heure 20 minutes, de plus ce dernier croît à mesure que l'on augmente la résolution.

L'exécution sur 8 et 16 processeurs (voire plus) met en évidence une réduction non négligeable du temps de calcul. Par exemple, pour les images les plus petites on s'approche des 5 minutes de calcul sur R10000, à noter que 32 processeurs de ce type ne permettent pas d'atteindre le temps que l'on obtient avec 16 processeurs R12000. La réduction est très appréciable lors du recalage d'IRM 256^3, puisque pour les processeurs les plus rapides les temps sont inférieurs à l'heure de calcul dès que l'on met en œuvre 8 processeurs. Pour avoir le même résultat (moins d'une heure) sur R10000, il faut le double de processeurs. Remarquons aussi que le recalage jusqu'à $l = 5$ de deux images 256^3 demande sur 16 R12000 un temps de calcul qui n'est supérieur que de 6 minutes à celui nécessaire sur un processeur lors du traitement des IRM 128^3.

Les facteurs d'accélération relative reflètent plus précisément le gain effectif qu'induit la parallélisation massive de l'équation de la diffusion. Dans le cas d'une mise en œuvre sur 8 processeurs, on a une accélération pour le type de processeur le moins puissant qui est de 4,30 ou 4,35 (approximativement) suivant la taille des images que l'on recale, alors qu'en considérant des R12000 celle-ci est de 4,30 ou 4,50. Quand on augmente le nombre de processeurs, l'accélération croît bien entendu, mais pas de façon importante. Par ailleurs, l'évolution du facteur d'accélération relative suivant le nombre de R10000 lors du recalage de deux IRM 128^3 voxels (1, 8, 16, 24 et 32 processeurs) montre que l'efficacité, qui n'est déjà que de 53,75% sur 8 processeurs, baisse très rapidement. On peut même considérer qu'elle « s'effondre ».

La première impression que l'on a des performances calculatoires de la parallélisation massive n'est donc pas positive, puisque malgré une réduction significative des temps de calcul (surtout pour les images 256^3), l'algorithme exhibe un manque flagrant d'efficacité. Cependant, ce sentiment doit être tempéré, car à y regarder de plus près, les facteurs d'accélération relative ne sont pas mauvais du tout comparé à ce que l'on pouvait espérer. De fait, une accélération quasi-linéaire, du type de celle observée pour les algorithmes parallèles sur le problème du recalage rigide, ne peut pas être obtenue avec l'algorithme de recalage déformable. En effet, une étape

Images	IRM 128^3 voxels		IRM 256^3 voxels					
Résolution finale	5		5		6		7	
l	Temps	Acc.	Temps	Acc.	Temps	Acc.	Temps	Acc.
Processeurs R12000								
1	1468,26		10954,40		12434,70		13897,44	
8	341,58	4,30	2437,95	4,49	2740,55	4,54	3114,60	4,46
16	257,89	5,70	1798,28	6,09	1998,15	6,22	2293,80	6,06
Processeurs R10000								
1	2095,35		15791,75		17924,20		20014,00	
8	487,75	4,30	3649,06	4,33	4074,09	4,40	4619,83	4,33
16	366,30	5,72	2735,14	5,77	2973,58	6,03	3446,88	5,81
24	329,25	6,36						
32	307,10	6,82						

TAB. 9.7 – Temps de calcul (en secondes) et accélération relative de l'algorithme de recalage déformable utilisant la version parallèle de l'équation de la diffusion comme méthode d'optimisation (IRM de différentes tailles et différents niveaux de résolution finale).

importante de ce dernier est purement séquentielle, avec un temps de calcul qui n'est pas négligeable : la phase de transition d'un niveau de résolution à un autre (voir la figure 6.1). On rappelle qu'elle consiste à recalculer les matrices de coefficients A_x^l, A_y^l et A_z^l, puis à supprimer tous les coefficients associés aux fonctions de base $\Omega_{\phi^l_{i,j,k}}$ où $i,j,k \in \{0,...,m_l-1\}$ qui ont pour support une partie des images ne portant pas d'informations. De plus, dans la section 9.3.2, nous avons vu que la version massivement parallèle de l'équation de la diffusion ne requiert qu'un processeur pour traiter le premier niveau de résolution. Or plus la fraction du temps d'exécution qui est séquentielle est importante, plus l'accélération relative est limitée. En conséquence, pour mieux juger la parallélisation massive, nous allons comparer chacune des accélérations relatives que l'on a obtenues à sa borne supérieure calculée grâce à la loi d'Amdahl (cf. section 7.1.3).

Pour calculer la borne supérieure de l'accélération relative sur un nombre de processeurs, il a fallu déterminer le temps d'exécution de la partie parallèle de l'algorithme de recalage déformable, c'est-à-dire l'addition des temps de calcul correspondants à chaque fois à l'optimisation d'un niveau de résolution l vérifiant $2 \leq l \leq l_f$, où l_f est le dernier niveau. À partir de ces temps de calcul, qui sont regroupés dans le tableau 9.8, et de ceux du tableau 9.7, il est aisé de calculer les différentes bornes supérieures, comme l'illustre l'exemple qui suit pour des processeurs R10000 et des images 128^3 voxels.

La partie de l'algorithme de recalage déformable (utilisant l'algorithme parallèle de l'équation de la diffusion) qui peut être exécutée en parallèle requiert pour cette taille d'image sur un R10000 1877,15 secondes, alors que l'exécution complète nécessite 2095,35 secondes. On en déduit que :

- la fraction du temps d'exécution pouvant être parallèle vaut $p = \frac{1877,15}{2095,35} \approx 0,896$;
- la fraction du temps d'exécution obligatoirement séquentielle est $s = 1 - p \approx 0,104$.

D'où, de l'équation 7.1, la borne supérieure pour l'accélération relative sur 8 processeurs :

$$\textit{Accélération relative} \leq \frac{1}{s + p/8} \Leftrightarrow \textit{Accélération relative} \leq 4,63$$

Images	IRM 128^3 voxels		IRM 256^3 voxels					
Résolution finale	5		5		6		7	
l	Temps	Acc.	Temps	Acc.	Temps	Acc.	Temps	Acc.
Processeurs R12000								
1	1320,00		9895,00		11261,90		12554,44	
8	180,00	7,33	1371,10	7,22	1551,15	7,26	1762,30	7,12
16	99,83	13,22	730,06	13,55	821,15	13,71	940,90	13,34
Processeurs R10000								
1	1877,15		14125,00		16108,80		17941,75	
8	259,80	7,23	1966,56	7,18	2235,82	7,20	2521,83	7,11
16	150,00	12,51	1048,38	13,47	1175,79	13,70	1357,18	13,22
24	100,50	18,68						
32	79,10	23,73						

Tab. 9.8 – Temps de calcul (en secondes) et accélération relative de la partie de l'algorithme de recalage pouvant être exécutée ou s'exécutant en parallèle, pour des IRM de différentes tailles et différents niveaux de résolution finale.

En procédant de même, on arrive respectivement aux bornes supérieures suivantes sur 16, 24 et 32 processeurs : 6,25, 7,07 et 7,57. La limite vers laquelle elle tend en augmentant le nombre de processeurs est $1/s$, soit dans ce cas précis 9,62. Lorsque l'on compare la courbe des facteurs d'accélération relative de l'algorithme de recalage déformable avec celle des bornes supérieures correspondantes calculées précédemment (graphique 9.10(a)), on remarque que les deux courbes ont la même allure et sont plutôt proches. Cela signifie donc qu'il faut revoir notre jugement sur la parallélisation massive de l'équation de la diffusion : ses performances sont très proches de celles que l'on pouvait espérer atteindre au mieux. Ce constat est aussi valable pour le modèle de processeur R12000 et les images 256^3 voxels.

D'autre part, les facteurs d'accélération relative que l'on obtient à partir des temps de calcul du tableau 9.8, c'est-à-dire de la partie exécutable en parallèle et qui sont donnés dans ce même tableau, mettent au jour de très bonnes performances. En effet, sur 8 processeurs la valeur de l'accélération relative est comprise entre 7,11 et 7,33 ce qui est excellent ; sur 16 processeurs la plage de valeur est comprise entre 12,51 et 13,71, cela reste bon mais traduit une légère dégradation des performances. Cette dégradation trouve son origine dans les communications sous-jacentes, car bien que l'Origin2000 ait une mémoire qui soit adressée globalement, physiquement elle est distribuée, d'où d'inévitables communications. Il est intéressant de remarquer qu'à l'image de la courbe 9.10(b), obtenue à partir des temps utilisés pour illustrer le calcul de la borne supérieure de l'accélération relative (tableau 9.8), on retrouve une courbe qui a la même allure que celle de l'hybride parallèle dans le cas du recalage rigide. En tout état de cause, l'efficacité de la partie parallèle est très correcte, de plus elle ne « s'effondre » pas : si on reprend l'exemple des R10000 avec des images de taille 128^3 voxels on a sur 8, 16, 24 et 32 processeurs une efficacité respective de 90,38%, 78,19%, 77,83% et 74,16%.

En résumé, la parallélisation massive de l'équation de la diffusion permet d'obtenir en peu d'itérations un recalage déformable de qualité, avec une réduction substantielle du temps de cal-

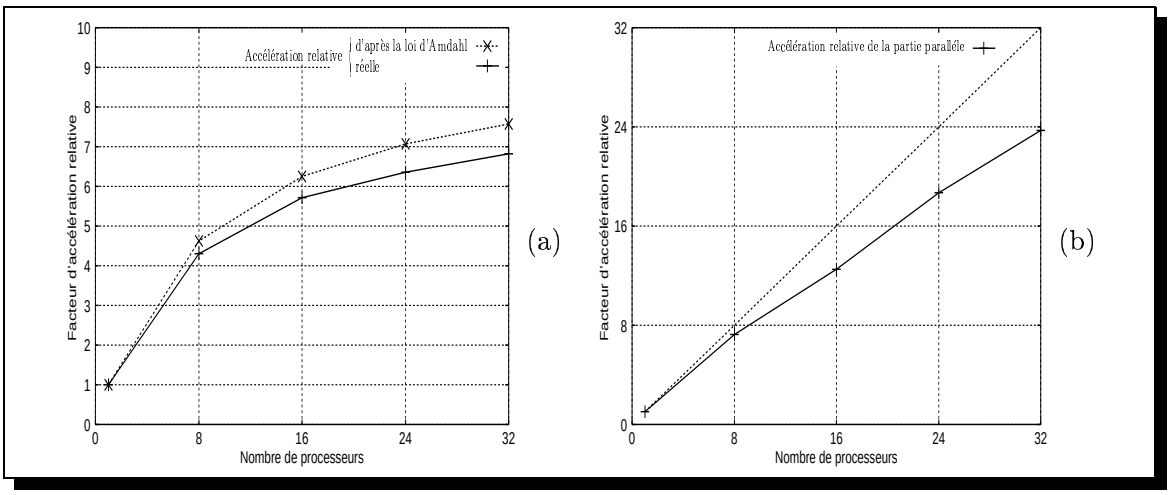


FIG. 9.10 – Courbe des facteurs d'accélération relative sur R10000 pour des IRM 128^3 voxels, en considérant les temps (a) de l'exécution complète, (b) uniquement de la partie qui est parallèle.

cul sur un processeur (clairement rédhibitoire pour les images 256^3). D'autre part, au vu du coût calculatoire des parties purement séquentielles, i.e. la transition entre deux niveaux de résolution et le traitement du premier niveau de résolution, l'algorithme parallèle fait preuve d'une bonne efficacité. Le seul point vraiment négatif est le choix des paramètres qui n'est pas facile. Cependant, cette remarque est à nuancer car l'influence de certains paramètres sur le comportement de l'algorithme est apparemment du même ordre que pour le recalage rigide, en particulier pour ce qui de la température T et du nombre de configurations Max_{conf}. Cela reste néanmoins à vérifier plus précisément.

9.4 Conclusion partielle

Dans ce chapitre, nous avons étudié deux algorithmes d'optimisation globale parallèles pour la méthode de recalage déformable considérée : l'évolution différentielle data-parallèle, que nous avons proposé dans le cadre du recalage rigide, et la version massivement parallèle de l'équation de la diffusion. Ces choix résultent pour le premier algorithme de ses bonnes performances en rigide, du peu de paramètres qu'il requiert et de sa robustesse, alors que l'équation de la diffusion semble *a priori* l'algorithme d'optimisation le plus adéquat pour optimiser le nombre croissant et important de variables qu'induit la modélisation multirésolution des déformations.

Nous avons montré que l'évolution différentielle data-parallèle est apte à traiter efficacement les recalages préalables au recalage déformable, c'est-à-dire les recalages rigide, rigide avec zoom et affine. Car, rappelons-le, le recalage déformable s'inscrit dans une suite de recalages hiérarchisés consistant à successivement rechercher des transformations dont le nombre de degrés de liberté va croissant. En revanche, lorsqu'il s'agit de passer au recalage déformable, il s'avère que l'évolution différentielle parallèle ne peut pas tirer correctement parti de l'approche hiérarchique de la méthode pour guider la recherche. En effet, comme à chaque niveau de résolution

la population converge, on ne peut pas la conserver comme population de départ à la résolution suivante car les individus sont trop « similaires ». Parmi les solutions envisagées pour résoudre ce problème, celle ayant donné les « meilleurs » résultats consiste à garder l'individu de coût minimum trouvé à la résolution qui vient d'être traitée et à engendrer les individus restants dans le voisinage de ce dernier. L'inconvénient de cette procédure d'initialisation est que la convergence de la population est longue à se dessiner, d'autant plus que le nombre de variables à optimiser est important. Néanmoins, l'évolution différentielle parallèle peut trouver un bon recalage à un niveau de résolution (cet algorithme donne d'ailleurs de très bons résultats au premier niveau de résolution), mais au prix d'un nombre de populations engendrées qui augmente de façon importante avec la résolution, avec pour conséquence directe un temps d'exécution qui est sans aucun doute rédhibitoire aux résolutions les plus fines.

Dans le cas de l'équation de la diffusion, l'expression des dérivées partielles de l'erreur quadratique moyenne par rapport à chacune des variables à optimiser à un niveau de résolution met en évidence des calculs réguliers et locaux. Dès lors, vu la croissance du nombre de variables à optimiser, la parallélisation massive sur machine SIMD autoriserait à haute résolution l'utilisation simultanée d'un très grand nombre de processeurs. Toutefois, ce type d'architecture ayant été supplanté par les machines MIMD, nous avons implanté l'algorithme data-parallèle sur une Origin2000 en utilisant comme paradigme de programmation parallèle OpenMP. Le résultat est une implantation qui est efficace relativement à la partie parallèle de l'algorithme de recalage, avec des images recalées qui sont d'excellente qualité, bien que le choix des paramètres ne soit pas aisé. Par exemple, le recalage de deux IRM 256^3 voxels mené jusqu'à la résolution la plus fine requiert sur un processeur R12000 environ 4 heures contre moins de 40 minutes sur 16 processeurs de ce type. Cela montre qu'il est tout à fait possible d'obtenir de bonnes performances en exécutant un algorithme massivement parallèle sur une machine dont le modèle d'exécution n'est pas adéquat. Enfin, lors des expérimentations, il est apparu qu'il serait intéressant que la méthode de recalage non rigide assure la conservation de la topologie.

Deux perspectives sont envisageables. D'une part, un raffinement de l'implantation de l'algorithme parallèle de l'équation de la diffusion, car comme il n'est pas possible de disposer d'autant de processeurs que ne le permettrait l'algorithme, il serait intéressant d'éliminer les calculs redondants sur les images au niveau des dérivées partielles. Cela consisterait à ne plus optimiser simultanément au niveau de résolution l tous les triplets de variables $\left(a^l_{x_{i,j,k}}, a^l_{y_{i,j,k}}, a^l_{z_{i,j,k}}\right)$, où $i,j,k \in \{0,...,m_l-1\}$, mais à les traiter en deux phases :

- dans un premier temps on optimise les triplets associés à des sous-domaines $\Omega_{\phi^l_{i,j,k}}$ qui sont disjoints et qui correspondent à un pavage complet du domaine Ω ;
- puis optimiser les triplets restants en utilisant les calculs sur les images qui auront été faits dans la phase précédente.

La seconde perspective est l'extension multimodale, c'est-à-dire de remplacer l'erreur quadratique moyenne par une fonction de similarité permettant de traiter des images multimodales. À première vue cela ne semble pas poser de problème particulier, mais à y regarder de plus près, il y a une difficulté importante : les fonctions de similarité dans le cadre multimodal sont difficilement dérivables analytiquement.

Conclusions générales et perspectives

Dans cette thèse, nous avons étudié la parallélisation d'algorithmes d'optimisation combinatoire dans le cadre de la problématique du recalage d'images, une opération de traitement d'images usuelle dans de nombreux domaines, en particulier en imagerie médicale. De manière générale, les techniques de recalage d'images s'expriment sous la forme d'un problème de minimisation d'une fonction de coût, ou fonction de similarité, évaluant la similitude des images à recaler. Ce problème nécessite un algorithme d'optimisation pouvant s'affranchir des nombreux minima locaux et de la non linéarité de la fonction de coût. D'autre part, le temps de calcul requis pour recaler les images doit être compatible avec une utilisation clinique et la pérennité des méthodes de recalages par rapport à l'augmentation de la taille des données assurée. Rappelons que nous considérons des méthodes de recalage denses, c'est-à-dire basées sur les niveaux de gris des voxels (des pixels en 2D). C'est pourquoi la parallélisation de l'algorithme d'optimisation globale, sur lequel s'appuie la méthode de recalage, s'impose. Après un état de l'art non exhaustif des algorithmes d'optimisation combinatoire et de leurs parallélisations dans la première partie de la thèse, puis la présentation du domaine d'application qu'est le recalage d'images médicales dans la seconde, la troisième partie expose l'étude que nous avons mené en deux temps.

Le premier volet de notre étude avait pour objet le recalage rigide : celui-ci consiste à rechercher une transformation correspondant à la combinaison d'une translation et d'une rotation. Les deux fonctions de similarité choisies, à savoir l'erreur quadratique moyenne et l'information mutuelle permettent de traiter respectivement des images monomodales et multimodales.

Nous avons tout d'abord étudié deux algorithmes d'optimisation globale liés au recuit simulé et qui s'avèrent en séquentiel tout à fait adéquat pour résoudre le problème posé. Ainsi, l'équation de la diffusion est un algorithme relativement robuste qui produit les recalages ayant la meilleure précision, mais il a l'inconvénient de contraindre la fonction de coût à être dérivable, d'où sa restriction à l'erreur quadratique moyenne[1]. Sa parallélisation massive n'a toutefois pas été mise en œuvre, car l'expression des dérivées partielles, sur lesquelles elle s'appuie pour remettre à jour en parallèle les variables à optimiser, a montré que les calculs locaux spécifiques à chaque processeur sont négligeables. Le second algorithme, l'*Adaptive Simulated Annealing* ou recuit simulé adaptatif, qui se distingue par l'ajout de températures utilisées dans une phase d'exploration originale, est également convaincant en séquentiel en lui ôtant sa capacité d'« auto-adaptation » (les températures ne remontent plus), avec des erreurs de recalage relativement faibles. L'étude de sa parallélisation suivant l'approche par recuits multi-températures, qui induit un comportement

[1] D'autres fonctions de coût, telles que l'information mutuelle, ont été dérivées récemment.

différent de l'algorithme séquentiel, permet de dire que le schéma de propagation déterministe des configurations est approprié pour obtenir rapidement, avec de bonnes accélérations, une transformation rigide proche de l'optimum. Pour garantir la précision sous-voxel désirée, les paramètres de la transformation doivent cependant être raffinés. En revanche, l'algorithme systolique qui repose sur un schéma de propagation probabiliste des configurations, lorsque l'on considère toutes les températures, doit être modifié au niveau de la probabilité d'acceptation, ce que nous avons fait avec succès en prenant comme probabilité d'acceptation celle de l'ASA.

Nous avons ensuite considéré deux algorithmes évolutionnaires utilisant une représentation réelle, i.e. l'évolution différentielle et les stratégies d'évolution. L'algorithme de l'évolution différentielle est notamment intéressant en raison du faible nombre de paramètres requis et de sa robustesse, contrairement aux stratégies d'évolution qui intègrent des paramètres au niveau des individus en plus des variables à optimiser. L'étude de l'influence de la taille des populations nous a conduit à choisir le parallélisme de données comme mode de parallélisation. Nous avons en particulier proposé un algorithme data-parallèle pour l'évolution différentielle, qui, bien que sémantiquement différent de la version séquentielle a un comportement équivalent. La pertinence de notre approche apparaît clairement lors des expérimentations, puisque l'on obtient un recalage de précision sous-voxel avec une accélération quasi-linéaire. En suivant une approche identique pour la parallélisation des stratégies d'évolution, nous avons défini un algorithme parallèle qui est le meilleur pour ce qui est de la réduction des temps de calcul, mais qui, du fait d'une différence sémantique plus prononcée par rapport à l'algorithme séquentiel, converge plus lentement et avec une moins bonne précision. Aussi, un ajustement précis des paramètres, éventuellement conjugué avec un plus grand nombre de populations engendrées qu'en séquentiel, est nécessaire pour être sûr d'avoir un recalage de qualité. Le raffinement de la transformation trouvée est également commode. Concernant l'hybride data-parallèle *Genetic Simulated Annealing*, son étude a mis en évidence qu'il faut plutôt le voir comme une première étape donnant une bonne initialisation pour une méthode d'optimisation locale. Nous avons ainsi vu qu'en faisant appel à l'algorithme *Iterated Conditional Modes*, qui n'accepte que des transformations qui font baisser le coût, il est possible d'obtenir systématiquement une précision sous-voxel et, qu'en le parallélisant, on peut réduire significativement le surcoût qu'induit son utilisation.

Dans un second temps, nous avons abordé le problème du recalage non rigide ou déformable, en nous restreignant au recalage d'images monomodales. La méthode hiérarchique considérée repose sur une modélisation paramétrique multirésolution des déformations. La transformation déformable ainsi recherchée est de complexité moyenne, avec un nombre de variables à optimiser qui croît régulièrement avec le niveau de résolution, tout en étant lié à la taille des images à traiter. Par exemple, le recalage d'une paire d'images 256^3 débute par l'optimisation de 3 variables, pour aboutir s'il est mené jusqu'à la résolution la plus fine à l'optimisation d'environ 1,5 millions de variables. À la suite des résultats obtenus sur le problème du recalage rigide, nous avons porté notre attention sur deux algorithmes d'optimisation parallèles : l'évolution différentielle data-parallèle et la parallélisation massive de l'équation de la diffusion

L'algorithme évolutionnaire parallèle s'est révélé apte à résoudre les recalages préalables au déformable. De fait, on ne recherche pas immédiatement la transformation déformable, car celle-ci s'inscrit dans une suite hiérarchisée de recalages consistant à déterminer des transformations de plus en plus complexes. En revanche, la parallélisation de l'équation de la diffusion n'est pas intéressante pour les recalages préalables, car comme nous l'avons constaté pour le recalage

rigide, toute transformation globale se traduit par une redondance importante des calculs sur les différents processeurs. Ainsi, pour les recalages rigide, rigide avec zoom et affine, qui précèdent le déformable, on peut résoudre le problème d'optimisation avec l'évolution différentielle parallèle. Par contre, pour le recalage déformable, nous avons vu que cet algorithme donne les meilleurs résultats au premier niveau de résolution, mais qu'ensuite ses performances se dégradent très rapidement, car il est difficile de tirer parti de la méthode hiérarchique pour guider la recherche. Dans le cas de la parallélisation massive de l'équation de la diffusion, le calcul du gradient de l'erreur quadratique moyenne par rapport aux variables à optimiser montre la pertinence de cette approche. L'implantation data-parallèle que nous avons effectuée permet d'accroître, avec la résolution, le nombre de processeurs pouvant être activés, ce qui, à haute résolution permettrait d'utiliser plusieurs milliers, voire dizaines de milliers et même centaines de milliers suivant la taille des images, de processeurs. Naturellement, l'architecture la plus adéquate est une machine SIMD, néanmoins nous avons vu que sur une machine MIMD il est tout à fait possible d'obtenir d'excellentes performances. L'algorithme data-parallèle a été validé sur des images 128^3 voxels et 256^3 voxels, apportant ainsi une solution au temps de calcul rédhibitoire en séquentiel (notamment pour les images 256^3) et assurant la pérennité future de la méthode de recalage non rigide.

Plus globalement, le travail que nous avons présenté a mis en évidence l'apport non négligeable du parallélisme en recalage d'images médicales, et en particulier la bonne adéquation du modèle de programmation data-parallèle. Les performances excellentes que l'on a mis au jour sur l'Origin2000, une machine MIMD, en utilisant HPF et OpenMP, montrent qu'en dépit de la disparition des architectures SIMD, le parallélisme de données reste d'actualité et compétitif.

Plusieurs perspectives peuvent être envisagées. D'une part en ce qui concerne les algorithmes d'optimisation globale :

- ☞ À l'heure actuelle, lorsque l'on cherche à résoudre un problème d'optimisation on choisit *a priori* un algorithme d'optimisation, puis on le valide empiriquement. Quitte à passer ensuite à une autre méthode d'optimisation si celle que l'on vient d'essayer ne s'avère pas adéquate. Il serait donc intéressant de définir des heuristiques permettant de guider le choix d'une méthode d'optimisation : ces règles devraient s'appuyer sur une analyse approfondie de la fonction de coût qui est à optimiser.
- ☞ Le problème de la détermination du meilleur jeu de paramètres d'un algorithme d'optimisation globale constitue en lui-même un problème d'optimisation. Par conséquent, on peut songer à utiliser un autre algorithme d'optimisation globale pour identifier un bon jeu de paramètres. Le coût associé aux paramètres d'un algorithme serait défini en fonction de l'objectif que l'on se fixe : par exemple, trouver le jeu de paramètres donnant le plus rapidement un bon recalage nécessiterait de faire une pondération entre le coût du recalage et son temps de calcul.

D'autre part, plus spécifiquement pour le domaine d'application considéré, i.e. le recalage d'images médicales, il serait intéressant de poursuivre notre travail dans les directions suivantes :

- ☞ Les algorithmes d'optimisation globale data-parallèles devraient être validés sur une base de données de plusieurs dizaines d'images. En effet, dans le cas du recalage rigide on a

uniquement traité des recalages synthétiques. Tandis que pour le recalage non rigide, il est primordial de faire cette validation afin de déterminer plus précisément la généralité du jeu de paramètres. De plus, du fait de la collaboration entre le LSIIT et l'Institut de Physique Biologique, en cas de validation sur une base de données d'images de la méthode de recalage déformable utilisant la parallélisation massive de l'équation de la diffusion comme algorithme d'optimisation, une utilisation en routine clinique sera vraisemblablement examinée.

- ☞ L'extension multimodale de la méthode de recalage non rigide et son impact éventuel sur la parallélisation de l'équation de la diffusion constituent également un sujet de recherche important.
- ☞ Enfin, ainsi que nous l'avons souligné lors des expérimentations, la propriété de conservation de la topologie serait clairement bénéfique pour le recalage déformable. La méthode considérée permet de garantir cette propriété en contraignant le jacobien de la transformation déformable à être positif. Une perspective importante est donc d'étudier la manière d'intégrer la contrainte sur le jacobien dans la version parallèle de l'équation de la diffusion.

Bibliographie

[1] E.H.L. Aarts, F. de Bont, E. Habers, and P.J.M. van Laarhoven. Parallel implementation of the statistical cooling algorithm. *Integration VLSI Journal*, 4 :209–238, 1986.

[2] E.H.L. Aarts and P.J.M. van Laarhoven. Statistical cooling : a general approach to combinatorial optimization problems. *Philips Journal of Research*, 40(4) :193–226, 1985.

[3] E.H.L. Aarts and P.J.M. van Laarhoven. A parallel statistical cooling algorithm. In *Proc. of the Symposium on The Theoritical Aspects of Computer Science*, pages 87–97. Springer Lecture Notes in Computer Science, 1986.

[4] F. Aluffi-Pentini, V. Parisi, and F. Zirilli. Global optimization and stochastic differential equations. *Journal of Optimization Theory and Applications*, 47(1) :1–16, September 1985.

[5] N. Ayache. L'analyse automatique des images médicales : état de l'art et perspectives. Technical Report 3364, INRIA, Rocquencourt, 1998.

[6] R. Azencott. *Parallelizing Simulated Annealing : An Overview of Basic Techniques*, chapter 4, pages 37–46. In [7], 1992.

[7] R. Azencott, editor. *Simulated annealing : parallelization techniques.* Wiley-Interscience, 1992.

[8] R. Azencott and C. Graffigne. *Parallel annealing by periodically interacting multiple searches : acceleration rates*, chapter 6, pages 81–90. In Azencott [7], 1992.

[9] N. Baaziz and C. Labit. Transformations pyramidales d'images numériques. Technical Report 526, IRISA, Rennes, 1990.

[10] T. Bäck. The interaction of mutation rate, selection and self-adaptation. In *Parallel Problem Solving from Nature 2*, Amsterdam, 1992. R. Männer and B. Manderick (editors), Elsevier.

[11] T. Bäck. Self adaptation in genetic algorithms. In *Proc of the First European Conf. Artificial Life*, pages 263–271, Amsterdam, 1992. MIT Press.

[12] T. Bäck. Optimal mutation rates in genetic search. In *Proc. of the Fifth Int. Conf. on Genetic Algorithms*, pages 2–8, San Mateo, California, 1993. Morgan Kaufmann Publishers.

[13] T. Bäck. *Evolutionary algorithms in theory and practice.* Oxford University Press, 1996.

[14] J.E. Baker. Adaptive selection methods for genetic algorithms. In *Proc. of the First Int. Conf. on Genetic Algorithms and their Applications*, pages 101–111, Hillsdale, New Jersey, 1985. J.J. Grefenstette (editor), Lawrence Erlbaum Associates.

[15] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2) :126–140, 1994.

[16] R. Battiti and G. Tecchiolli. Simulated annealing and tabu search in the long run : a comparison on QAP tasks. *Computer an Mathematics with Applications*, 28(6) :1–8, 1994.

[17] J. Besag. On the statistical analysis of dirty pictures. *Journal of the Royal Statistical Society*, 48 Serie B(3) :259–302, 1997.

[18] H.-G. Beyer. Toward a theory of evolution strategies : some asymptotical results from the $(1, +\lambda)$-theory. *Evolutionary Computation*, 1(2) :165–188, 1993.

[19] H.-G. Beyer. Toward a theory of evolution strategies : on the benefits of sex - the $(\mu/\mu, \lambda)$-theory. *Evolutionary Computation*, 3(1) :81–111, 1995.

[20] H.-G. Beyer. Toward a theory of evolution strategies : the (μ, λ)-theory. *Evolutionary Computation*, 2(4) :381–407, 1995.

[21] H.-G. Beyer. On the asymptotic behavior of multirecombinant evolution strategies. In *Parallel Problem Solving from Nature 4*, Heidelberg, 1996. H.-M. Voigt, W. Ebeling, I. Rechenberg and H.-P. Schwefel (editors), Springer.

[22] L.G. Brown. A survey of image registration techniques. *ACM Computing Survey*, 24(4) :325–376, 1992.

[23] P. Calégari, F. Guidec, P. Kuonen, and D. Kobler. Parallel island-based genetic algorithm for radio network design. *Journal of Parallel and Distributed Computing*, 47 :86–90, 1997.

[24] O. Catoni. *Etude asymptotique des algorithmes de recuit simulé.* PhD thesis, Université Paris XI (Orsay), 1990.

[25] O. Catoni. *Rates of convergence for sequential annealing : a large deviation approach*, chapter 3, pages 25–35. In Azencott [7], 1992.

[26] Y. Censor and S. A. Zenios. *Parallel optimization : Theory, Algorithms and Application.* Oxford University Press, 1997.

[27] R. Cerf. Asymptotic convergence of genetic algorithms. Preprint submitted, February 1993.

[28] R. Cerf. *Une théorie asymptotique des algorithmes génétiques.* PhD thesis, Université Montpellier II, 1994.

[29] H. Chen, N.S. Flan, and D.W. Watson. Parallel genetic simulated annealing : a massively parallel SIMD algorithm. *IEEE Trans. Parallel Distrib. Systems*, 9(2) :126–136, February 1998.

[30] S. Chen and B.L. Luk. Adaptive simulative annealing for optimization in signal processing applications. *Signal Processing*, 79 :117–128, 1999.

[31] G.E. Christensen. MIMD vs. SIMD parallel processing : A case study in 3D medical image registration. *Parallel Computing*, 24 :1369–1383, 1998.

[32] G.E. Christensen, M.I. Miller, M.W. Vannier, and U. Grenander. Individualizing neuro-anatomical atlases using a massively parallel computer. *IEEE Computer*, January :32–38, 1996.

[33] G.E. Christensen, R.D. Rabbitt, and M.I. Miller. Deformable templates using large deformation kinematics. *IEEE Trans. on Image Processing*, 5(10) :1435–1447, 1996.

[34] L. Dagum and R. Menon. OpenMP : an industry-standard API for shared memory programming. *IEEE Computational Science & Engineering*, 5(1) :46–55, 1998.

[35] I. Daubechies. Orthonormal bases of compactly supported wavelet. *Commun. Pure Appl. Math.*, 41 :909–996, November 1988.

[36] T.E. Davis and J.C. Principe. A Markov framework for the simple genetic algorithm. *Evolutionary Computation*, 1(3) :269–298, 1993.

[37] K. De Jong. *An analysis of the behavior of a class of genetic adaptative systems.* PhD thesis, University of Michigan, 1975.

[38] D. Fischer, P. Kohlhepp, and F. Bulling. An evolutionary algorithm for the registration of 3D surface representations. *Pattern Recognition*, 32 :53–69, 1999.

[39] D.B. Fogel. *Evolving artificial intelligence.* PhD thesis, University of California, San Diego, 1992.

[40] D.B. Fogel. *Some recent important foundational results in evolutionary computation*, chapter 4, pages 55–70. In Miettinen et al. [85], 1999.

[41] L.J. Fogel. Toward inductive inference automata. In *Proc. of the Int. Federation for Information Processing Congress Conf.*, pages 395–399, Munich, 1962.

[42] M.I. Freidlin and A.D. Wentzell. *Random perturbations of dynamical systems.* Springer-Verlag, 1984.

[43] D. Geman and S. Geman. Stochastic relaxation, Gibbs distribution, bayesian restoration of images. *IEEE Trans. Pattern Analysis Machine Intelligence*, 6 :721–741, November 1984.

[44] S. Geman and C.R. Hwang. Diffusions for global optimization. *SIAM Journal on Control and Optimization*, 24 :1031–1087, 1986.

[45] F. Glover. Tabu search - Part I. *ORSA Journal on Computing*, 1(3) :190–206, Summer 1989.

[46] F. Glover. Tabu search - Part II. *ORSA Journal on Computing*, 2(1) :4–32, Winter 1990.

[47] D.E. Goldberg. Genetic algorithms and walsh functions : I. A gentle introduction, II. Deception and its analysis. *Complex Systems*, 3(2) :129–171, April 1989.

[48] D.E. Goldberg. *Genetic algorithms in search, optimization and machine learning.* Addison-Wesley, 1989.

[49] D.E. Goldberg and K. Deb. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of Genetic Algorithms*, San Mateo, California, 1991. G.J.E. Rawlins (editor), Morgan Kaufmann Publishers.

[50] C. Graffigne. *Parallel annealing by periodically interaction : an experimental study*, chapter 5, pages 47–79. In Azencott [7], 1992.

[51] J.J. Grefenstette. Optimization of control parameters for genetic algorithms. *IEEE Trans. on Systems, Man and Cybernetics*, 16(1) :122–128, 1986.

[52] J.J. Grefenstette. How genetic algorithms works : a critical look at implicit parallelism. In *Proc. of the Third Int. Conf. Genetic Algorithms and Their Applications*, pages 20–27, San Mateo, California, 1989. J.D. Schaffer (editor), Morgan Kaufmann Publishers.

[53] W. Grimson, G. Ettinger, S. White, T. Lozano-Pérez, W. Wells, and R. Kikinis. An automatic registration method for frameless stereotaxy, image guided surgery, and enhanced reality visualization. *IEEE Trans. on Medical Imaging*, 15 :129–140, 1996.

[54] A. Guimond, G. Subsol, and J.-P. Thirion. Automatic MRI database exploration and average brain building. *Int. Journal of Pattern Recognition and Artificial Intelligence*, 11(8), December 1997.

[55] J.K. Hao, R. Dorne, and P. Galinier. Tabu search for frequency assignement in mobile radio networks. *Journal of Heuristics*, 4(1) :47–62, 1998.

[56] F. Heitz, P. Perez, and P. Bouthemy. Multiscale minimization of global energy functions in some visual recovery problems. *Comput. Vis. Graphics Image Proc. Image Understanding*, 59(1) :125–134, 1994.

[57] D. Hill, C. Studholme, and D. Hawkes. Voxel similarity measures for automated image registration. In *Proc. of the Third Conf. on Visualization in Biomedical Computing (VBC'94).* in R.A. Robb (editor), October 1994.

[58] C.A.R. Hoare. Communicating sequential processes. *Comm. ACM*, 21(8) :666–677, 1978.

[59] F. Hoffmeister, T. Bäck, and H.-P. Schwefel. A survey of evolution strategies. In *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, pages 2–9, University of California, Los Altos, 1991. R. Belew and L. Booker (editors), Morgan Kaufmann Publishers.

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

[61] W.D. Holpert and W. Macready. No free lunch theorems for optimization. *IEEE Trans. on Evolutionary Computation*, 1(1) :67–82, 1997.

[62] C.R. Hwang and S.J. Sheu. Singular perturbed Markov chains and exact behaviors of simulated annealing process. *Journal of Theoritical Probability*, 5(2) :223–249, 1992.

[63] L. Ingber. Very fast simulated re-annealing. *Mathematical and computing Modelling*, 12 :967–993, 1989.

[64] L. Ingber. Simulated annealing : practice versus theory. *Mathematical and Computing Modelling*, 18(11) :29–57, 1993.

[65] L. Ingber. Adaptive simulated annealing (ASA) : lessons learned. *Control and Cybernetics*, 25(1) :33–54, 1996.

[66] M. Iosifescu. *Finite Markov processes and their applications.* Wiley, Chichester, 1980.

[67] B. Jawerth and W. Sweldens. An overview of wavelet based multiresolution analyses. *SIAM Review*, 36 :377–412, 1994.

[68] J. Juliany and M.D. Jose. The genetic algorithm fractal. *Evolutionary Computation*, 2(2) :165–180, 1994.

[69] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220 :671–680, May 1983.

[70] C.H. Koelbel, D.B. Loveman, R.S. Schreiber, G.L Steel Jr., and M.E. Zosel. *The High Performance Fortran Handbook.* The MIT Press, Cambridge, Massachusetts, 1994.

[71] J.J. Korczak and P. Lipinski. Evolution strategies : principles and prototypes. Technical Report 05, Laboratoire des sciences de l'image, de l'informatique et de la télédétection (LSIIT), Université Louis Pasteur, Strasbourg, 2001.

[72] P. Krolak, W. Felts, and G. Marble. A man-machine approach to solving the traveling salesman problem. *Comm. ACM*, 14(4) :327–334, 1971.

[73] Y.K. Kwok and I. Ahmad. Efficient scheduling of arbitrary tasks graphs to multiprocessors using a parallel genetic algorithm. *Journal of Parallel and Distributed Computing*, 47 :58–77, 1997.

[74] H. Lester and S.R. Arridge. A survey of hierarchical non-linear medical image registration. *Pattern Recognition*, 32 :129–149, 1999.

[75] E. Lutton. *Genetic algorithms and fractals*, chapter 15, pages 327–350. In Miettinen et al. [85], 1999.

[76] E. Lutton and J.L. Lévy Véhel. Hölder functions and deception of genetic algorithms. *IEEE Trans. on Evolutionary Computation*, 2(2) :56–72, July 1998.

[77] F. Maes, A. Collignon, D. Vandermeulen, G. Marchal, and P. Suetens. Multimodality image registration by maximisation of mutual information. *IEEE Trans. on Medical Imaging*, 16(2) :187–198, 1997.

[78] S.W. Mahfoud and D.E. Goldberg. Parallel recombinative simulated annealing : a genetic algorithm. *Parallel Computing*, 21 :1–28, 1995.

[79] J.B.A. Maintz and M.A. Viergever. A survey of medical image registration. *Medical Image Analysis*, 2(1) :1–36, 1998.

[80] S. Mallat. A theory for multiresolution signal decomposition : the wavelet representation. *IEEE Trans. Pattern Analysis Machine Intelligence*, 11(7) :674–693, 1989.

[81] B. Manderick and P. Spiessens. A massively parallel genetic algorithm : implementation and first analysis. In *Proc. of the Fourth Int. Conf. on Genetic Algorithms*, University of California, San Diego, 1991. R. Belew and L. Booker (editors), Morgan Kaufmann Publishers.

[82] E. Memin. *Algorithmes et architectures parallèles pour les approches markoviennes en analyse d'images.* PhD thesis, Université Rennes I, Juin 1993.

[83] N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller. Equation of state calculation by fast computer machines. *Journal of Chemical Physics*, 21(6) :1087–1092, June 1953.

[84] Z. Michalewicz. *Genetic algorithms + data structures = evolution programs.* Springer, 1994.

[85] K. Miettinen, M.M. Mäkelä, P. Neittaanmaki, and J. Périaux, editors. *Evolutionary algorithms in engineering and computer science.* Wiley and Sons, 1999.

[86] H. Mühlenheim. How genetic algorithms really work : I. Mutation and hillclimbing. In *Parallel Problem Solving from Nature 2*, Amsterdam, 1992. R. Männer and B. Manderick (editors), Elsevier.

[87] O. Musse. *Contribution à la mise en correspondance non rigide d'images médicales : une approche paramétrique hiérarchique sous contraintes topologiques. Application au recalage déformable du cerveau en imagerie IRM.* PhD thesis, Université Strasbourg I, Décembre 2000.

[88] O. Musse, F. Heitz, and J.-P. Armspach. 3D deformable image matching using multiscale minimization of global energy functions. In *IEEE Conf. on Computer Vision and Pattern Recognition*, Fort Collins, 1999.

[89] O. Musse, F. Heitz, and J.-P. Armspach. Hierarchical deformable matching of 3D MR images over multiscale nested subspaces. Preprint submitted to IEEE Transactions on Medical Imaging, 1999.

[90] S. Niar and A. Freville. A parallel tabu search algorithm for the 0-1 multidimensional knapsack problem. In *International Parallel Processing Symposium IEEE-ACM*, Genève, April 1997.

[91] C. Nikou, F. Heitz, and J.-P. Armspach. Robust voxel similarity metrics for the registration of dissimilar single and multimodal images. *Pattern Recognition*, 32 :1351–1368, 1999.

[92] C. Nikou, F. Heitz, J.-P. Armspach, I.-J. Namer, and D. Grucker. Registration of MR/MR and MR/SPECT brain images by fast stochastic optimization of robust voxel similarity measures. *Neuroimage*, 8 :30–43, 1998.

[93] G.-R. Perrin and A. Darte, editors. *The data parallel programming model*, volume 1132 of *Lecture Notes in Computer Science*. Springer-Verlag, 1996.

[94] I. Rechenberg. Cybernetic solution path of an experimental problem. Technical Report 1122, Royal Aircraft Establishment, Farnborough, 1965.

[95] I. Rechenberg. *Evolutionsstrategie : optimierung technischer systeme nach prinzipien des biologischen evolution.* Frommann-Holzboog Verlag, Stuttgart, 1973.

[96] I. Rechenberg. *Artificial evolution and artificial intelligence*, pages 83–103. R. Forsyth (editor), Chapman and al., London, 1989.

[97] I. Rechenberg. *Evolutionstrategie'94.* Frommann-Holzboog Verlag, Stuttgart, 1994.

[98] E.M. Reingold, J. Nievergelt, and N. Deo. *Combinatorial algorithms : theory and practice.* Prentice-Hall, 1977.

[99] F. Richard and C. Graffigne. An image matching model for the registration of time sequence or bilateral mammogram pairs. In *Proc. of the Fifth Int. Workshop on Digital Mammography*, Toronto, Canada, June 2000.

[100] A. Roche, G. Malandain, X. Pennec, and N. Ayache. The correlation ratio as a new similarity measure for multimodal image registration. In *Proc. of MICCAI'98*, pages 1115–1124, Cambridge, Massachusetts, October 1998.

[101] J.-M. Rouet, J.-J. Jacq, and C. Roux. Genetic algorithms for a robust 3D MR-CT registration. *IEEE Trans. on Information Technology in Biomedicine*, 4(2) :126–136, 2000.

[102] P. Roussel-Ragot and G. Dreyfus. *Parallel annealing by multiple trials : an experimental study on a transputer network*, chapter 7, pages 91–108. In Azencott [7], 1992.

[103] B. Roysam and M.I. Miller. Joint symbolic and stochastic inference via massively parallel architectures : application to medical imaging systems. Technical report, Rensselaer Polytechnic Institute, Troy, NY 12180, 1990.

[104] B. Roysam, M.I. Miller, K.R. Smith, and J.A. O'Sullivan. Representing and computing regular languages on massively paralle networks. *IEEE Trans. on Neural Networks*, 2(1) :56–72, January 1991.

[105] G. Rudolph. Massively parallel simulated annealing and its relation to evolutionary algorithms. Manuscript.

[106] G. Rudolph. On correlated mutations in evolution strategies. In *Parallel Problem Solving from Nature 2*, Amsterdam, 1992. R. Männer and B. Manderick (editors), Elsevier.

[107] G. Rudolph. Convergence analysis of canonical genetic algorithms. *IEEE Trans. on Neural Networks*, 5(1) :96–101, January 1994.

[108] G. Rudolph. Convergence of evolutionary algorithms in general search spaces. In *Proc. the Third IEEE Conf. on Evolutionary Computation*, pages 50–54, Piscataway, 1996. IEEE Press.

[109] G. Rudolph. Finite Markov chains results in evolutionary computation : a tour d'horizon. *Fundamenta Informaticae, ISO Press*, 1998.

[110] G. Rudolph. Self-adapation and global convergence : a counter-example. Preprint, January 1999.

[111] R.S. Schreiber. *An Introduction to HPF*, chapter 2, pages 27–44. Volume 1132 of Perrin and Darte [93], 1996.

[112] M. Schütz and T. Bäck. Intelligent mutation rate control in canonical algorithms. In *Proc. of the Nineth Int. Symp. on Methodologies for Intelligent Systems, Lectures Notes in Computer Science*, volume 929, pages 893–907, Berlin, 1995. Z. Ras (editor), Springer.

[113] H.-P. Schwefel. *Kybernetische evolution als strategie des experimentellen forschung in der strömungstechnik*. PhD thesis, Universität Berlin, 1965. Diplomarbeit.

[114] H.-P. Schwefel. *Numerische optimierung von computer-modellen mittels der evolutionsstrategies*, volume 26 of *Interdisciplinary Systems Research*. Birkhäuser, Basel, 1977.

[115] H.-P. Schwefel. *Numerical optimization of computer models*. Wiley, Chichester, 1981.

[116] H.-P. Schwefel. *Evolution and optimum seeking*. Sixth-Generation Technology Series. Wiley, New York, 1995.

[117] H.-P. Schwefel and G. Rudolph. Contemporary evolution strategies. In *Advances in Artificial Life, Proc. of the Third Int. Conf. on Artificial Life, Lecture Notes in Artificial Life*, volume 929, pages 893–907, Berlin, 1995. F. Morán, A. Moreno, J.J. Merelo and P. Chacón (editors), Springer.

[118] E. Seneta. *Non-negative matrices and Markov chains*. Springer, New York, 1981. 2nd edition.

[119] R. Storn. On the usage of differential evolution for function optimization. In *NAFIPS*, pages 519–523, Berkeley, 1996.

[120] R. Storn and K. Price. Differential evolution - A simple and efficient adaptative scheme for global optimization over continous spaces. Technical Report TR-95-012, ICSI, March 1995.

[121] R. Storn and K. Price. Minimizing the real functions of the ICEC'96 contest by differential evolution. In *Int. Conf. on Evolutionary Computation*, Nagoya, Japan, 1996.

[122] M.A. Styblinski and T.-S. Tang. Experiments in nonconvex optimization : stochastic approximation with function smoothing and simulated annealing. *Neural Networks*, 3 :467–483, 1990.

[123] G. Syswerda. Uniform crossover in genetic algorithms. In *Proc. of the Third Int. Conf. on Genetic Algorithms and Their Applications*, pages 2–9, San Mateo, California, 1989. J.D. Schaffer (editor), Morgan Kaufmann Publishers.

[124] H. Szu. Fast simulated annealing. In *Proc. of the American Institute of Physics, "Neural Networks for Computing"*, pages 420–425, Snowbird, Utah, 1986. J.S. Denker (editor).

[125] H. Szu and R. Hartley. Nonconvex optimization by fast simulated annealing. In *Proc. of the IEEE*, volume 75, pages 1538–1540, 1987.

[126] E. Taillard. Robust tabu search for the quadratic assignment problem. *Parallel Computing*, 17 :443–455, 1991.

[127] E. Taillard. *Recherches itératives dirigées parallèles.* PhD thesis, Ecole Polytechnique Fédérale de Lausanne, 1993.

[128] E.-G. Talbi and T. Muntean. A new approach for the mapping problem : a parallel genetic algorithm. Technical report, 1991.

[129] P. Thompson and G. Calmon. Visualization and mapping of anatomic abnormalities using a probabilistic brain atlas based on random fluid transformations. *Medical Image Analysis*, 1(4), 1997.

[130] M. Tomassini. *Parallel and distributed evolutionary algorithms*, chapter 7, pages 113–133. In Miettinen et al. [85], 1999.

[131] A. Trouvé. *Parallélisation massive du recuit simulé.* PhD thesis, Université Paris XI (Orsay), 1993.

[132] A. Trouvé. Rough large deviation estimates for the optimal convergence speed exponent of generalized simulated annealing algorithms. Technical Report LMENS-94-8, Ecole Normale Supérieure, April 1994.

[133] A. Trouvé. Cycle decompositions and simulated annealing. *SIAM Journal on Control and Optimization*, 34(3) :966–986, May 1996.

[134] P.A. van den Elsen, E.J.D. Paul, and M.A. Viergever. Medical image matching - a review with classification. *IEEE Engineering in Medicine and Biology*, 12(4) :26–39, 1993.

[135] B. Virot. *Parallel annealing by multiple trials : experimental study of a chip placement problem using a sequent machine*, chapter 8, pages 109–127. In Azencott [7], 1992.

[136] W. Wells, P. Viola, H. Atsumi, S. Nakajima, and R. Kikinis. Multimodal volume registration by maximization of mutual information. *Medical Image Analysis*, 1(1) :33–51, 1996.

[137] E.E. White, R.D. Chamberlain, and M.A. Franklin. Parallel simulated annealing using speculative computation. *IEEE Trans. Parallel Distrib. Systems*, 2(4) :484–494, October 1991.

[138] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Building better test functions. In *Proc. of the Sixth Int. Conf. on Genetic Algorithms*, pages 239–246, San Francisco, California, 1995. Morgan Kaufmann Publishers.

[139] D. Whitley, K. Mathias, S. Rana, and J. Dzubera. Evaluating evolutionary algorithms. *Artificial Intelligence*, 85 :245–276, 1996.

[140] R.P. Woods, J.C. Maziotta, and S.R. Cherry. MRI-PET Registration with automated algorithm. *Journal of Comput. Assist. Tomography*, 17(4) :536–546, 1993.

Publications

① M. Salomon. Parallélisation de l'évolution différentielle pour le recalage rigide d'images médicales volumiques. In *RenPar'2000, 12èmes Rencontres Francophones du Parallélisme*, pages 53-58, 19-22 Juin 2000, Besançon, France.

② M. Salomon. Parallélisation de l'évolution différentielle pour le recalage rigide d'images médicales volumiques. *Technique et science informatiques*, 20(5):605-627, 2001.

③ M. Salomon, G.-R. Perrin and F. Heitz. Differential evolution for medical image registration. In *2001 International Conference on Artificial Intelligence (IC-AI'2001)*, Regular Research Report (7 pages), June 25-28, Las Vegas, U.S.A.