RenPar'2000
Besançon, 19 juin - 22 juin 2000

Parallélisation de l'évolution différentielle pour le recalage rigide d'images médicales volumiques

Michel Salomon

LSIIT-ICPS
UPRES-A CNRS 7005, Université Louis Pasteur, Strasbourg
Pôle API, Boulevard Sébastien Brant
67400 Illkirch

Résumé

Dans le domaine de l'imagerie médicale, de nombreuses méthodes de recalage d'images s'expriment sous la forme de la minimisation d'une fonction de coût (ou d'énergie) définie à partir de l'image de référence et de l'image à recaler. Cette minimisation est difficile à mener en raison de la présence de nombreux minimums locaux ; d'autre part ces méthodes sont très coûteuses en temps de calcul. Pour résoudre ce problème d'optimisation nous considérons un algorithme d'évolution : l'évolution différentielle. Afin de réduire le temps de calcul, nous tirons parti du parallélisme intrinsèque des algorithmes d'évolution pour proposer une implantation data-parallèle.

Mots-clés : recalage d'images médicales 3D, évolution différentielle, data-parallélisme.

1. Introduction

Au cours de la décennie passée le recalage d'images est devenu une opération de base dans de nombreuses applications telles que l'analyse du mouvement, l'analyse de scènes, etc. De tous les domaines ayant recours au recalage d'images, l'imagerie médicale est sans aucun doute un domaine d'application privilégié. En effet, le recalage d'images intervient dans nombre d'applications médicales, allant de l'assistance lors d'interventions chirurgicales [5] à l'analyse fonctionnelle d'images permettant de suivre l'évolution de lésions ou d'évaluer l'efficacité d'un traitement. Le recalage d'images consiste à mettre en correspondance deux (ou plusieurs) images différentes de telle façon que les coordonnées d'un point physique soient identiques dans les différentes images. Lorsque le recalage traite des images provenant du même système d'acquisition (Imagerie par Résonance Magnétique, Tomographie par Emission de Positrons, etc) il est dit monomodal, dans le cas contraire on parle de recalage multimodal.

Différentes méthodes de recalage ont été développées. Néanmoins une grande partie de ces techniques ne permettent pas de traiter des images très dissemblables. Parmi les approches ayant donné des résultats satisfaisants en présence de grandes dissimilarités beaucoup font appel à une fonction de similarité. Ces méthodes consistent à minimiser une fonction de coût exprimant la similitude au niveau des voxels (ou pixels) entre les différentes images, ce qui requiert la définition d'une technique d'optimisation. Dans le cadre du recalage monomodal IRM/IRM que nous allons considérer dans cette étude, les approches par fonctions de similarité sont généralement associées à une formulation au sens des moindres carrés.

La résolution de problèmes d'optimisation intervient dans une variété de domaines. Une conséquence de cette diversité est le développement d'une multitude de méthodes de résolution que l'on peut diviser en deux catégories : les méthodes de résolution locales telles que la descente du gradient et les méthodes globales comme le recuit simulé [7] par exemple. Du fait de la difficulté et de l'irrégularité de la fonction de coût induite par le recalage d'images, nous avons choisi comme technique d'optimisation une méthode globale, en l'occurrence un algorithme d'évolution : l'évolution différentielle [12]. La parallélisation de cet algorithme s'impose pour réduire les temps de calcul et dans l'optique de pouvoir traiter des problèmes de taille plus grande.

Cet article présente une implantation data-parallèle de l'évolution différentielle. Les performances obtenues montrent la pertinence de l'approche proposée. Avant d'aborder la parallélisation, nous présentons la fonction de coût retenue et l'algorithme de l'évolution différentielle dans le cadre évolutionnaire.

2. Recalage d'images médicales monomodales par fonctions de similarité

Comme nous l'avons déjà précisé, un grand nombre de méthodes de recalage d'images ont été proposées. Une synthèse des techniques classiques a été faite par Brown dans [2] et une classification de ces méthodes a été donnée par van den Elsen et al. [14]. Les approches par fonction de similarité sont des méthodes de recalage guidées par les données, l'objectif étant de déterminer la transformation minimisant un critère évaluant la similarité des niveaux de gris des voxels se correspondant dans les différentes images. Des métriques évaluant la similarité des images ont été proposées pour le recalage monomodal et le recalage multimodal.

Recaler des images par fonction de similarité consiste à estimer les paramètres Θ d'une transformation rigide T_Θ minimisant une fonction de coût $C(I(.), J(T_\Theta(.)))$, évaluant qualitativement la « ressemblance » entre les images I et J :

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

où $\Theta_{\min}$ est le vecteur des paramètres de translation et de rotation :

$$\Theta_{\min} = (t_x, t_y, t_z, \theta_x, \theta_y, \theta_z)^T \quad \text{(pour des images 3D)},$$

$I(.)$ est l'image référence et $J(.)$ est l'image devant être recalée.

Dans le cadre du recalage monomodal, une fonction largement utilisée est la fonction de similarité quadratique. Soit x le vecteur 3D des coordonnées spatiales, la fonction de coût à optimiser est alors :

$$C\left(I(x), J(T_\Theta(x))\right) = \sum_x \left[I(x) - J(T_\Theta(x))\right]^2$$

Cette fonction de coût, tout comme celles induites par d'autres mesures de similitude classiques, se caractérise par de nombreux minima locaux et une forte non linéarité. Il est donc nécessaire pour optimiser la fonction de coût d'avoir recours à des algorithmes pouvant sortir du bassin d'attraction d'un minimum local. Ceci a conduit au choix de l'évolution différentielle, une méthode d'optimisation globale de la famille des algorithmes d'évolution, qui se prêtent aisément à la parallélisation.

3. Optimisation par l'algorithme de l'évolution différentielle

3.1. Une vue générale des algorithmes d'évolution

Dans le courant des années soixante de nombreux chercheurs ont eu l'idée de simuler le processus de l'évolution naturelle, aboutissant ainsi à des algorithmes d'optimisation efficaces et robustes. Parmi ces algorithmes, les plus connus sont sans conteste les algorithmes génétiques introduits par Holland [6], et plus précisément, pour l'optimisation de paramètres, par DeJong [3]. Deux autres grands types d'algorithmes d'évolution sont la programmation évolutionnaire résultant des travaux de Fogel [4] et les stratégies d'évolution développées en Allemagne par Rechenberg et Schwefel [10,11]. Une synthèse complète des différents aspects de ces trois types d'algorithmes d'évolution peut être trouvée dans le livre de Bäck [1]. L'évolution différentielle définie par Storn et Price [12] est un algorithme plus récent, dont les premières applications mettent en évidence sa robustesse et son efficacité.

L'idée à la base des algorithmes d'évolution est de faire évoluer les individus d'une population par l'intermédiaire d'opérateurs stochastiques de manière à favoriser l'émergence d'individus minimisant (ou maximisant) une fonction d'évaluation, chaque individu de la population représentant une solution potentielle au problème que l'on cherche à résoudre. Par analogie avec la nature, deux niveaux de représentation sont possibles :

- le niveau « génotypique », dans lequel un individu correspond à la représentation chromosomique d'une solution. La solution est donc codée : en général on utilise l'alphabet binaire $\{0, 1\}$ pour représenter les gènes. Un individu est alors une suite de bits ;
- le niveau « phénotypique », dans lequel un individu représente directement une solution du problème.

L'évaluation d'un individu donne une information qualitative sur l'individu par rapport à l'environnement considéré, i.e. la fonction à optimiser. Aussi la fonction d'évaluation est-elle définie à partir de la fonction de coût. Pour simuler l'évolution naturelle on applique successivement trois opérateurs à la population : le croisement (ou recombinaison) et la mutation, qui sont deux opérateurs génétiques assurant la reproduction, puis la sélection (basée sur l'évaluation) pour former la nouvelle population. Chaque opérateur a un rôle bien précis ; ainsi la sélection doit guider la recherche en favorisant la reproduction des meilleurs individus (ayant la meilleure évaluation), la mutation doit introduire de nouvelles informations et le croisement échanger des informations entre plusieurs individus, ce qui doit permettre d'explorer l'espace de recherche. L'algorithme engendre une suite de populations jusqu'à ce qu'un critère d'arrêt prédéfini par l'utilisateur soit vérifié. On peut donc décrire un algorithme d'évolution par le schéma suivant :

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

La population $P(t)$ est formée par μ individus $a_j, j \in \{1, ..., \mu\}$, et Φ est la fonction d'évaluation. Suivant l'algorithme d'évolution considéré, la sélection peut opérer soit uniquement sur la population des descendants, soit en tenant également compte de la population des parents. Le critère d'arrêt est en général un nombre maximum de générations engendrées, mais une mesure évaluant la convergence de l'algorithme peut aussi être introduite.

3.2. Evolution différentielle

L'idée à l'origine de l'évolution différentielle est d'utiliser la différence entre deux individus choisis aléatoirement dans la population pour perturber un troisième individu. L'évolution différentielle opère au niveau phénotypique, travaillant directement sur les paramètres du vecteur que l'on cherche à optimiser. Les individus sont donc de la forme $a = x \in \mathbb{R}^n$, et comme fonction d'évaluation on peut prendre la fonction de coût $f : \mathbb{R}^n \rightarrow \mathbb{R}$, soit $\Phi(a) = f(x)$ (dans le cas du recalage $a = \Theta$ et $\Phi(a) = C\left(I(.), J(T_\Theta(.))\right)$).

Les auteurs [12] décrivent la reproduction par l'application successive de deux opérateurs : tout d'abord une mutation engendrant une nouvelle population, puis un croisement de chaque parent avec son descendant respectif par l'intermédiaire d'un croisement à deux points. Dans la suite nous allons fusionner ces deux opérateurs en un seul que nous nommerons mutation par croisement. On suppose que la population contient μ individus, chaque individu étant formé de n composantes, d'où $a_j = x_j = (x_{j,1}, x_{j,2}, ..., x_{j,n})$, $j \in \{1, ..., \mu\}$ (donc $P(t) = \{a_j(t) = (t_{x\,j}(t), t_{y\,j}(t), t_{z\,j}(t), \theta_{x\,j}(t), \theta_{y\,j}(t), \theta_{z\,j}(t))^T, j \in \{1, ..., \mu\}\}$ pour le recalage rigide d'images). Soit $a'_j = x'_j$ l'individu obtenu à l'issue de la mutation par croisement de l'individu a_j, on peut alors décrire formellement cet opérateur par :

$$\chi_1 \text{ et } \chi_2 \in \{1, ..., n\} \text{ sont les points de croisement obtenus par } \begin{cases} \chi_1 = (\chi \bmod n) + 1 \\ \chi_2 = ((\chi + L - 1) \bmod n) + 1 \end{cases}$$

où χ et L sont deux entiers tirés aléatoirement pour chaque individu dans $\{1, ..., n\}$. χ représente le premier point de croisement, alors que L est le nombre de paramètres qui sont modifiés. De plus l'entier L est tiré de sorte que $P\{L \geq \nu\} = (p_c)^{\nu - 1}$, où $\nu > 0$ et p_c est la probabilité de croisement donnée par l'utilisateur. Le nouveau vecteur $x'_j = \left(x'_{j,i}\right), i \in \{1, ..., n\}$ est alors engendré de la manière suivante :

$$x'_{j,i} = \begin{cases} x_{j,i} + \lambda \cdot (x_{min,i} - x_{j,i}) + F \cdot (x_{S,i} - x_{T,i}) \text{ pour } & \begin{array}{ll} \chi_1 \leq i \leq \chi_2 & \text{si } \chi_1 \leq \chi_2 \\ 1 \leq i \leq \chi_2 \text{ et } \chi_1 \leq i \leq n & \text{si } \chi_1 > \chi_2 \end{array} \\ x_{j,i} \quad \text{sinon} \end{cases} \tag{1}$$

L'individu $a_{min} = x_{min}$ correspond au minimum courant. Les indices $S, T \in \{1, ..., \mu\}$ doivent être mutuellement différents, mais également différents de l'indice j de l'individu courant. F et $\lambda \in [0, 1]$ sont des facteurs constants contrôlant l'amplification des vecteurs différence ; habituellement on donne à λ la même valeur que pour F afin de réduire le nombre de paramètres de l'algorithme. Storn [13] appelle le schéma de mutation (1) « *Rand to best* », ce qui signifie que l'individu a est perturbé en additionnant la valuation d'un vecteur différence choisi aléatoirement à la valuation du vecteur différence attirant a vers le minimum courant. Après reproduction, la nouvelle population est obtenue par l'intermédiaire d'une sélection déterministe correspondant à un tournoi binaire entre chaque individu a_j de la population de

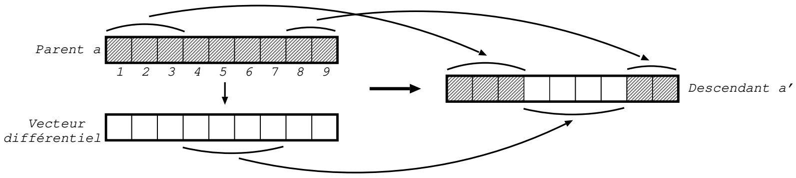


FIG. 1 – *Reproduction pour* $n = 9, \chi = 4$ *et* $L = 4$

départ et son descendant respectif $a'_j, j \in \{1, ..., \mu\}$. Formellement dans le cas d'une minimisation :

$$\text{Sélection}(P \cup P') = \{s(a_j, a'_j), j \in \{1, ..., \mu\}\}, \text{ avec } s(a_j, {a'}_j) = (1 - p_s({a'}_j)) \cdot a_j + p_s({a'}_j) \cdot {a'}_j \quad (2)$$
$$\text{et } p_s({a'}_j) = 1_{\mathbb{R}^+}(f(x_j) - f({x'}_j))$$

Ce type de sélection est dit μ-élitiste. Le choix des différents paramètres (F, λ, p_c, initialisation de $P(0)$) est en général guidé par plusieurs règles établies à l'issue d'études empiriques [13].

4. Parallélisation

La parallélisation d'un algorithme d'optimisation peut être motivée par plusieurs raisons, notamment : la réduction du temps de calcul en distribuant équitablement la charge sur les processeurs ; la possibilité d'implanter de nouvelles stratégies d'optimisation différant des méthodes d'optimisation séquentielles.

4.1. Comment paralléliser un algorithme d'évolution

Les algorithmes d'évolution sont intrinsèquement parallèles, leur parallélisation semble donc plus aisée que celle d'autres algorithmes tel que le recuit simulé [7]. Le parallélisme peut être introduit dans un algorithme d'évolution à trois niveaux : au niveau de la population, des individus ou de la fonction d'évaluation. Ce dernier niveau ne sera pas évoqué car étant lié au problème d'optimisation considéré, alors que les deux autres niveaux induisent des parallélisations génériques.

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 d'évolution considéré sur sa sous-population. L'intérêt de ce type de parallélisation à gros grain vient du fait que plusieurs instances de l'algorithme d'évolution traitant des sous-populations différentes permet 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 vers une autre : en général c'est le meilleur individu d'une sous-population qui migre. Une alternative consiste à laisser évoluer les sous-populations indépendamment et à ne les faire communiquer qu'à la fin de la recherche, afin de choisir la meilleure solution parmi celles trouvées par les différents processeurs.

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, puisque celle-ci se fait par l'intermédiaire des opérateurs de sélection et reproduction qui se font localement en considérant un voisinage. Ceci détermine implicitement le degré d' « isolation » et ainsi la diversité des individus de la population.

4.2. Approche data-parallèle de l'évolution différentielle

Une étude de l'algorithme séquentiel dans le contexte du recalage d'images a montré qu'une population de taille réduite permettait de converger en obtenant d'excellents résultats. Les tailles de population n'étant pas suffisamment importantes pour justifier une approche par division en « îlots », nous avons décidé de nous orienter vers le data-parallélisme.

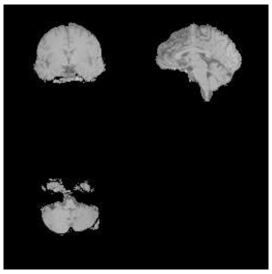
Les différentes populations de l'algorithme (population des parents a, des descendants a' et les populations des individus a_S et des individus a_T), sont distribuées sur une grille de processeurs que l'on supposera bouclée sur elle-même, de manière à n'avoir qu'un seul individu par processeur. Cette approche data-parallèle induit la parallélisation des deux opérateurs de l'évolution différentielle, i.e. la sélection et la mutation par croisement. La sélection telle qu'elle est définie (2) est particulièrement adaptée à cette parallélisation puisqu'elle consiste à faire un tournoi entre un individu et son descendant, or ces deux individus se trouvant sur le même processeur aucune communication n'est nécessaire. La phase de reproduction décrite par (1) implique en revanche des communications. En effet, pour engendrer un nouvel individu il est nécessaire de connaître l'individu a_{min} de coût minimum dans la population courante, ainsi que de tirer aléatoirement deux individus mutuellement différents (a_S et a_T) mais également différents de l'individu courant. Dans cette optique, est associé à chaque processeur, un voisinage défini comme étant formé par ses huit-voisins immédiats sur la grille : a_S et a_T correspondent alors à l'individu courant de deux processeurs distincts tirés dans ce voisinage. 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 se traduit par des communications régulières. La détermination de l'individu a_{min} ne pose pas de problème particulier, de fait le calcul du coût se faisant en parallèle, nous stockons le coût associé à un individu sur le processeur où ce dernier est localisé. Il est évident que l'algorithme parallèle n'a pas le même comportement que la version séquentielle, ce qui pourrait se traduire par des résultats différents, mais comme nous le verrons ci-dessous ce n'est pas le cas. Notons que par nature, l'algorithme parallèle est fortement synchronisé.

5. Résultats expérimentaux

5.1. Implantation et cadre expérimental

Dans le cadre du recalage d'images, l'algorithme parallèle s'inscrit dans une stratégie d'optimisation multirésolution consistant à partir d'une résolution grossière vers une résolution plus fine (1 voxel sur 81, 27, 9, 3 et 1), l'évolution différentielle parallèle faisant le même nombre d'itérations pour les quatre premiers niveau de résolution et une seule itération à la résolution complète. D'autre part, afin d'accélérer la convergence, l'espace de recherche des paramètres de la translation est centré sur le vecteur de translation résultant du recalage des barycentres des deux images volumiques que l'on veut recaler.

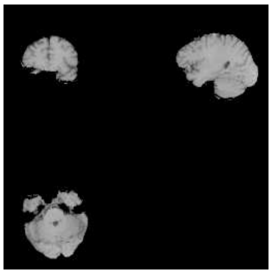
La parallélisation décrite ci-dessus a été implantée sur l'Origin 2000 de l'ULP (32 processeurs, 8 Go de mémoire) dans le langage data-parallèle HPF. Cette machine parallèle est à mémoire partagée, physiquement distribuée, et a pour réseau d'interconnexion un hypercube. Pour évaluer la qualité du recalage nous avons appliqué une vingtaine de transformations prédéfinies à une image IRM 3D de référence, afin d'obtenir 20 images à recaler. Les différentes transformations ont des valeurs de translation comprises entre -20 et +20 voxels et des angles entre -20 et +20 degrés. L'image de référence provient de l'Institut de Physique Biologique des Hôpitaux Universitaires de Strasbourg.



a

Paramètres		Erreur moyenne et écart-type
Δt_x	voxels	$0,197 \pm 0,142$
Δt_y	voxels	$0,168 \pm 0,106$
Δt_z	voxels	$0,175 \pm 0,107$
$\Delta \theta_x$	degrés	$0,094 \pm 0,066$
$\Delta \theta_y$	degrés	$0,119 \pm 0,102$
$\Delta \theta_z$	degrés	$0,118 \pm 0,134$

b



c

FIG. 2 – *(a) Image de référence, (b) erreur moyenne sur 20 recalages, (c) une des 20 images à recaler.*

5.2. Performances et analyse des résultats

Les courbes de l'évolution moyenne du coût minimum sur 20 recalages (figure 3 à gauche), obtenues pour la parallélisation sur 8 processeurs et pour la version séquentielle, montrent la validité de notre approche. En effet, la parallélisation n'a pas d'impact visible sur le résultat du recalage puisqu'elle permet d'aboutir à un vecteur Θ de paramètres de coût équivalent à celui de la version séquentielle. D'autre part, le tableau de la figure 2 montre que la version parallèle permet d'obtenir un recalage de qualité, puisque celui-ci est de précision sous-voxel. Nous pensons que le bon comportement de la parallélisation est dû essentiellement à la définition du voisinage. De fait des tests en séquentiel ont montré qu'une population de 9 individus se traduisait par de bons résultats, or le voisinage défini dans la version parallèle induit une sous-population pour la reproduction de cette taille. Cette hypothèse reste cependant à valider plus précisément. La courbe des facteurs d'accélération par rapport à la version séquentielle montre que l'on obtient une accélération croissante avec le nombre de processeurs. La dégradation que l'on observe en augmentant le nombre de processeurs résulte en grande partie de l'augmentation des communications irrégulières nécessaires pour diffuser le minimum sur l'ensemble des processeurs. En fait, à basse résolution, les communications sont plus coûteuses que le calcul de la fonction Φ. D'autre part, les positions et les longueurs de croisement varient d'un processeur à un autre, ce qui induit un léger déséquilibre de la charge.

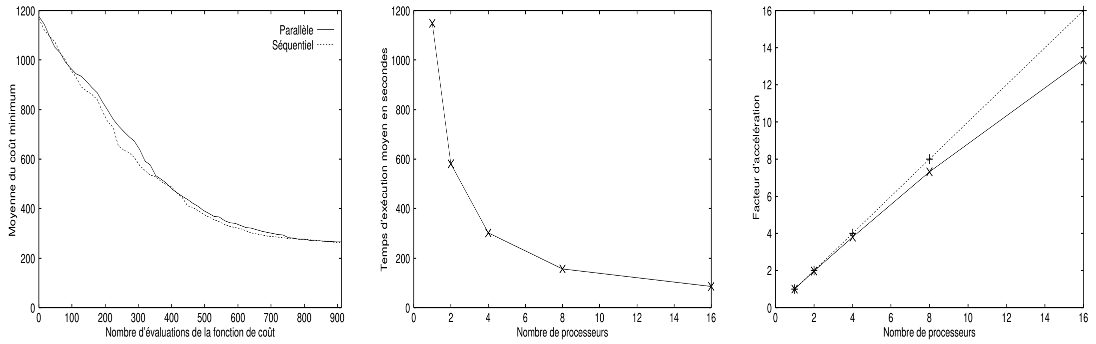


FIG. 3 – *Evolution moyenne du coût minimum, temps d'exécution moyen et facteurs d'accélération.*

6. Conclusion

La parallélisation que nous proposons pour l'évolution différentielle permet d'obtenir un recalage de qualité sous-voxel, ceci avec de bonnes accélérations par rapport à la version séquentielle. L'approche data-parallèle proposée a consisté à distribuer les populations, et à paralléliser les différents opérateurs, ainsi que l'évaluation des individus. Les développements futurs de ce travail sont l'étude de l'influence de la taille et de la forme du voisinage sur l'algorithme, ainsi que la détermination de règles pour le choix des paramètres de l'évolution différentielle parallèle. L'application à d'autres méthodes de recalages telles que les méthodes robustes de recalage rigide [9] et le recalage déformable [8] est également envisagée.

Bibliographie

1. Bäck (T.) – Evolutionary Algorithms in Theory and Practice. – Oxford University Press, 1996.
2. Brown (L.G.) – A survey of image registration techniques. – ACM Computing Survey, vol. 24, $N°4$, p. 325-376, 1992.
3. DeJong (K.) – An analysis of the behaviour of a class of genetic adaptive systems – PhD Thesis, University of Michigan, 1975.
4. Fogel (L.J.) – Toward inductive inference automata. – Proceeding of the Int. Federation for Information Processing Congress Conf., p. 395-399, 1962.
5. Grimson (W.), Ettinger (G.), White (S.), Lozano-Pérez (T.), Wells (W.) et Kikinis (R.) – An automatic registration method for frameless stereotaxy, image guided surgery, and enhanced reality visualization. – IEEE Transactions on Medical Imaging, vol. 15, p. 129-140, 1996.
6. Holland (J.H.) – Adaptation in Natural and Artificial Systems. – University of Michigan Press, 1975.
7. Kirkpatrick (S.) et al. – Optimization by simulated annealing. – Science, vol. 220, p. 671-680, 1983.
8. Musse (O.), Heitz (F.) et Armspach (J.-P.) – 3D Deformable Image Matching Using Multiscale Minimization of Global Energy Functions, IEEE Conference on Computer Vision and Pattern Recognition 1999.
9. Nikou (C.), Heitz (F.), Armspach (J.-P.), Namer (I.-J.) et Grucker (D.) – Registration of MR/MR and MR/SPECT Brain Images by Fast Stochastic Optimization of Robust Voxel Similarity Measures. – Neuroimage, vol. 8, p. 30-43, 1998.
10. Rechenberg (I.) – Evolutionsstrategie : Optimierung Technischer Systeme nach Prinzipien des Biologischen Evolution. – Fromman-Holzwoog, 1973.
11. Schwefel (H.-P.) – Numerische Optimierung von Computer-Modellen mittels der Evolutionsstrategie. – Interdisciplinary Systems Research, vol. 26, Birkhäuser, 1977.
12. Storn (S.) et Price (K.) – Minimizing the real functions of the ICEC'96 contest by Differential Evolution. – Int. Conf. on Evolutionary Computation, 1996.
13. Storn (S.) – On the Usage of Differential Evolution for Function Optimization. – NAFIPS p. 519-523, 1996
14. Van den Elsen (P.A.), Paul (E.J.D.) et Viergever (M.A.) – Medical image matching - a review with classification. – IEEE Engineering in Medicine and Biology, vol. 12, $N°4$, p. 26-39, 1993.