1/00

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

المدرسة الوطنية المتعددة التقنيات
Ecole Nationale Polytechnique

ECOLE NOTIONALE POLYTECHNIQUE

DEPARTEMENT DU GENIE INDUSTRIEL

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Projet de fin d'études pour l'obtention du diplôme d'ingénieur en Génie Industriel

Le problème du voyageur de commerce par les réseaux neuronaux : Approche de résolution

Proposé et encadré par

Dr N.ABOUN

Réalisé par

M.AIT ALI SLIMANE

Promotion 2000

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

ملخص :

لقد جلبت مسألة التاجر المسافر منذ عدة سنوات اهتمام كبير و ولدت عدة أعمال. ظهرت في السنوات الأخيرة عدة طرق لحل هذه المسألة. من بين هذه الطرق، ركزنا اهتمامنا على الطرق العقلية أي الطرق التي أساسها الشبكات العصبونية الإصطناعية.

الهدف من هذا العمل هو التعريف بكل هذه الطرق الجديدة ثم برمجة احداها لاختبار جودتها واقتراح بعض التحسينات.

كلمات مفاتيح: مسألة التاجر المسافر ، الشبكات العصبونية الاصطناعية .

Résumé :

Depuis de nombreuses années, le problème du voyageur de commerce suscite un grand intérêt et a donné lieu à de nombreux travaux. Ces dernières années ont vu l'apparition de nouvelles approches de résolution du problème. Parmi ces méthodes, nous nous sommes intéressés aux méthodes neuronales i.e. celles basées sur les réseaux de neurones artificiels.

L'objet de ce travail est de faire un état de l'art de ces méthodes. Nous poursuivrons par l'implémentation de la méthode de compétition harmonique afin d'en évaluer les performances et proposer certaines améliorations.

Mots clés: Le problème du voyageur de commerce, Optimisation combinatoire, Réseaux de neurones artificiels, Cartes auto-organisées, Réseaux élastiques.

Abstract :

Since many years, the traveling salesman problem has been of great interest and gave place to many works. These last years saw the apparition of news approaches of resolution of the problem. Among these methods, we interested ourselves to neural methods i.e. those based on the artificial neural networks.

The object of this work is to make a state of the art of these methods. We will pursue by the implementation of the harmonic competition method in order to evaluate its performances and to propose some improvements.

Key words: The travelling salesman problem, Combinatorial optimisation, Artificial neural networks, Self organizing maps, Elastic net.

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

REMERCIEMENTS

Mes remerciements iront tout d'abord à Melle ABOUNE, ma promotrice, pour m'avoir proposé un sujet aussi passionnant, pour son aide et la patience dont elle a fait preuve. Ses orientations et ses conseils ont été décisifs pour la réalisation de ce travail.

Je remercie M.Ouabdeslam pour avoir accepté d'évaluer ce travail en siégeant dans le jury.

Je remercie également M.Lamraoui pour ses conseils et sa participation au jury.

Je voudrais remercier également Mmes Belmokhtar et Bencherif pour avoir accepté de siéger dans le jury.

Merci à ma famille et particulièrement à mon frère pour son aide pratique.

Merci également à Adel pour son aide et son soutien ainsi qu'à Rabie pour ses cours de Delphi.

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

SOMMAIRE

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Chapitre 4 : Le TSP et l'auto-organisation

Chapitre 5 : Etude des différentes méthodes

Chapitre 6 : Implémentation de la méthode de compétition harmonique

المدرسة الوطنية المتعددة التقنيات
BIBLIOTHEQUE — المكتبة
Ecole Nationale Polytechnique

Introduction générale

Le problème du voyageur de commerce (the Traveling Salesman Problem) est peut être le problème le plus populaire de l'optimisation combinatoire. L'une de ses variantes est le problème du voyageur de commerce euclidien : Connaissant les coordonnées de N villes dans le plan euclidien, il s'agit de déterminer le circuit de longueur minimale passant une seule fois par chaque ville. C'est à cette variante particulière du TSP que ce travail fera référence.

Le problème à été posé sous cette appellation et dans les termes qu'on lui connaît aujourd'hui au plus tard dans les années 1940. c'est un chercheur américain de la RAND corporation, Merill Flood, qui a popularisé le terme au sein de la communauté scientifique afin de créer un challenge intellectuel sur des modèles hors théorie des jeux. Un prix a même été offert pour tout théorème constituant un apport conséquent à la résolution du problème. Une des raisons de la popularité du TSP alors était son lien étroit avec des sujets éminents de la toute récente Programmation Linéaire (Dantzig1940) tels que le problème de transport et le problème d'affectation.

L'approche de résolution la plus remarquée à cette époque fut celle de Dantzig, Fulkerson et Johnson (1954) où furent évoqués pour la première fois les concepts de plans sécants et de Branch et Bound. Depuis, les recherches se sont poursuivies et ont donné naissance à des méthodes parfois étonnantes.

Le TSP euclidien étant NP-complet, les méthodes exactes ne peuvent déterminer une solution optimale en un temps polynomial. Nombre d'heuristiques ont alors été développées afin d'arriver à une solution approximative en un temps raisonnable.

Parmi ces heuristiques, nous pouvons compter les approches neuronales. Ces approches, basées sur les réseaux de neurones artificiels, sont récentes et suscitent l'intérêt de la communauté scientifique.

Hopfield et Tank (1985) furent les premiers à tenter une approche neuronale de résolution du TSP. Leur approche faisait appel au réseau de Hopfield et à sa fonction d'énergie qu'ils modifièrent afin que sa minimisation par le réseau de Hopfield coïncide avec la minimisation de la longueur d'un tour. La méthode de Hopfield et Tank a donné naissance à la famille des méthodes neuronales répondant à une formulation du problème en programmation linéaire en nombres entiers et à ouvert la voie à d'autres travaux dont l'apport était de mêler l'utilisation du réseau de Hopfield à d'autres techniques destinées à en améliorer les performances.

Un autre fait marquant fut celui de la naissance des méthodes neuronales géométriques pour la résolution du TSP. Durbin et Willshaw ont, en 1987, imaginé un réseau élastique inspiré des cartes auto-organisatrices de Kohonen capable de former un tour en s'adaptant à la distribution des villes dans le plan. Dans la lignée des approches géométriques, Angéniol et al adaptent, en 1988, une carte auto-organisatrice de Kohonen au TSP. Cette seconde famille de méthodes possède, elle aussi, son lot de travaux dont le but était d'améliorer les performances des méthodes neuronales géométriques. La représentation des réseaux de neurones sous forme

de graphe et leur plasticité sont pour beaucoup dans le développement de ce type d'approches.

Notre travail consistera à faire un état de l'art des approches neuronales de la seconde famille de méthodes, c'est à dire celles utilisant une représentation géométrique du TSP. Nous présenterons quelques méthodes appartenant aux deux classes que sont les réseaux élastiques et les cartes auto-organisatrices dans l'ordre chronologique afin de mettre en relief leur évolution. Il ne s'agit aucunement de faire une recherche exhaustive sur les méthodes neuronales d'optimisation, mais uniquement de présenter quelques-unes des méthodes les plus marquantes du point de vue de la méthodologie et des résultats. Nous ferons une discussion autour méthodes présentées sur la base des résultats obtenus par les auteurs, nous donnerons les avantages et inconvénients de chacune. Nous aurons recours à l'implémentation lorsque les résultats de la littérature seront insuffisants pour permettre une comparaison. Lors de l'implémentation, nous ferons l'étude de certains paramètres de l'algorithme et ferons quelques propositions destinées à accélérer sa convergence.

Pour ce faire, ce document s'articulera autour de cinq chapitres :

- Dans le premier nous ferons une présentation du problème du voyageur de commerce et dresserons une typologie de ses heuristiques afin de situer les méthodes neuronales au sein de cette typologie. Nous terminerons en donnant quelques éléments permettant l'évaluation des heuristiques.
- Le second chapitre servira à introduire les concepts de base des réseaux de neurones artificiels après un rappel de leurs fondements biologiques. Le domaine des réseaux de neurones étant très vaste, nous ferons en sorte que cette partie théorique soit tournée vers le concept d'auto-organisation.
- Les fondements des réseaux élastiques et les méthodes s'y rattachant seront présentées dans le troisième chapitre. L'approche pionnière de Durbin et Willshaw ainsi que trois améliorations récentes feront l'objet de cette partie.
- Le quatrième chapitre accueillera la théorie des cartes auto-organisatrices de Kohonen, l'approche de B.Angéniol et al ainsi que deux méthodes constituant des améliorations de cette dernière.
- Dans le cinquième chapitre, nous ferons un bilan des méthodes présentées en terme de qualité des solutions obtenues, de temps de calcul et de facilité d'implémentation. Nous retiendrons à la fin de cette partie la méthode à implémenter.
- Ce dernier chapitre décrira l'implémentation de la méthode désignée dans le chapitre précédent c'est à dire la méthode de compétition harmonique. Nous présenterons au cours de cette partie certaines propositions pouvant améliorer les performances de la méthode. Les résultats de l'implémentation seront commentés et analysés.

Nous conclurons notre travail par certaines remarques et suggestions.

1.1. Présentation du problème

Le problème du voyageur de commerce (TSP) est un problème classique de la recherche opérationnelle et plus précisément de l'optimisation combinatoire.

1.1.1. Enoncé du problème [19]

Son énoncé est simple : « un voyageur de commerce doit, partant de chez lui, parcourir N villes en passant une seule fois par chacune et revenir chez lui, en minimisant le coût (distance, temps ou dépense) de son trajet. »
Il consiste donc à déterminer la permutation π des villes {1,2,...,N} qui minimise la quantité : $\sum_{i} d(C_{\pi(i)}, C_{\pi(i+1)}) + d(C_{\pi(1)}, C_{\pi(N)})$

Où : C_k symbolise la $k^{ième}$ ville visitée.

Le TSP peut être modélisé sous forme de programme linéaire en nombres entiers comme suit : : Minimiser $\sum_{i=1}^{N}\sum_{j=i+1}^{N} d(Ci,Cj)\, x_{ij}$

Sous les contraintes: $\sum_{i=1}^{N} xij=1$

$$\sum_{j=1}^{N} xij=1$$

$$\sum_{ij} xij=1$$

$$x_{ij} \subset \{0,1\}$$

Le TSP doit son importance à ses applications pratiques et à son implication dans la théorie de la complexité. En effet, il a été prouvé que résoudre la TSP revenait à résoudre les autres problèmes NP-complets car les problèmes de cette classe sont polynomialement réductibles les uns aux autres. C'est pour cela que le TSP sert de banc d'essai à beaucoup de nouvelles approches d'optimisation combinatoire. **[19]**

Pour l'instant, nous ne connaissons pas d'algorithme donnant la solution exacte du TSP en un temps polynomial en N (nombre de villes). C'est pour cela que les recherche se sont orientées vers des approches donnant une solution proche de l'optimalité et dont le temps de calcul est raisonnable. **[19]**

Le TSP euclidien

La variante du TSP que nous allons étudier est le TSP euclidien (E-TSP) où les distances entre villes sont symétriques : $C_{ij}=C_{ji}$, et où l'inégalité triangulaire est satisfaite : $C_{ij}+C_{jk} \leq C_{ik}$

Les villes sont placées dans le plan et données par leurs coordonnées x_i et y_i.
Les distances entre villes sont euclidiennes : $dij=\sqrt{(xi-xj)^2+(yi-yj)^2}$

1.1.2. Complexité algorithmique

Le TSP appartient à la classe des problèmes NP-complets (Non deterministic Polynomial). C'est à dire qu'il n'existe pas d'algorithme procurant la solution exacte du TSP en un temps polynomial. Par temps polynomial nous entendons ici, un temps de calcul borné par un polynôme en N (nombre de villes)**[19]**

Le nombre total de combinaisons possibles sur une instance à N villes est (N-1)!,/2 dans le cas du TSP symétrique. Par exemple, sur une instance du TSP à 42 villes, le nombre de combinaisons possibles est de $1.67^* 10^{49}$, ce nombre est astronomique sachant que le nombre de particules estimé dans l'univers est de 10^{48} . **[28]**
Durant les 20 dernières années, les efforts de recherche et les progrès rapides fait dans le domaine de l'informatique ont aboutit à la résolution exacte d'instances de plus en plus grandes du TSP : de 318 villes (Crowder et Padberg, 1980) à 2392 villes (Padberg et Rinaldi, 1987) puis à 7397 villes (Appelgate, Bixby, Chvatàl et Cook, 1994). Cette dernière instance a été résolue en 4 ans sur une SPARC station2. De nos jours, résoudre une instance à 100 villes ne prend que quelques heures et une instance à 1000 villes quelques jours sur une station de travail (work station) **[14]**

1.1.3. Quelques applications du TSP

Le TSP euclidien a de nombreuses applications, parmi elles nous pouvons citer :

- L'optimisation de tournées d'une flotte de véhicules**[14]**, Le routage des appels téléphoniques **[27]** et plus généralement, toutes les tâches logistiques devant être optimisées.
- L'allocation de ressources et la planification des budgets. **[27]**
- L'ordonnancement des tâches d'un processeur ou de toute autre machine de sorte à minimiser le temps de complétion. Ici le temps de passage d'une tâche à une autre est assimilé à une distance entre deux villes.
- La conception des puces VLSI où il est nécessaire d'optimiser les circuits reliant les différentes portes logiques de la puce.
- L'élaboration du parcours d'un pinceau traceur de cartes géographiques ou tout autre traçage. **[4]**
- La cristallographie à rayon X où le rayon doit balayer la surface d'un corps. **[14]**

1.2. Typologie des heuristiques pour le TSP [32]

L'alternative aux méthodes exactes et énumératives est l'utilisation d'heuristiques. Ces heuristiques peuvent procurer des solutions proches de l'optimum en un temps 'raisonnable'. De nombreuses heuristiques ont été développées ces dernières années. Nombre d'entre elles s'inspirent de mécanismes et de comportements issus de la nature. La recherche de nouvelles voies dans ce domaine se poursuit et les chercheurs n'hésitent pas à recourir à la nature. La dernière inspiration biologique nous vient du singe vert pour construire une tournée, raisonne sur les trois sommets suivant. **[27]**

Nous introduisons ici certaines heuristiques afin de positionner notre travail au sein de la typologie actuelle.

1.2.1. Heuristiques de recherche locale [19]

Ces heuristiques sont de trois types :

Les procédures de construction de tournées

Les procédures les plus connues sont :

Nearest neighbor algorithm : il consiste, à partir d'une ville de départ, à insérer au fur et à mesure la ville la plus proche du dernier sommet inséré jusqu'à épuisement des villes.

Procédure d'insertion : à la $k^{ième}$ itération, la procédure considère un circuit à k sommets et détermine le sommet à insérer suivant l'un des critères suivant :
Nearest insertion : trouver le sommet s le plus proche du circuit puis déterminer les sommets i et j qui minimisent $C_{ik}+C_{kj}-C_{ij}$ et insérer s entre i et j.
Cheapest insertion : identique à la précédente, elle recherche cependant un sommet s à l'extérieur du circuit et des couples (i, j) à l'intérieur tels que : $C_{ik}+C_{kj}-C_{ij}$ soit minimisé.
Il existe d'autres procédures d'insertion telles que l'Insertion arbitraire et farthest insertion. Ces procédures procurent des solutions de l'ordre de 10 à 15% en dessous de la solution optimale (en terme de qualité). **[14]**

Les procédures d'affinage de tournées **[19]**

Procédures r-Optimale :
Ces procédures consistent à échanger les arcs d'un tour pour en réduire le coût. Plus précisément, à partir d'un tour initial aléatoire, il faut échanger un ensemble de r-arcs par un ensemble de la même taille si celui-ci réduit le coût de la tournée. Lorsque plus aucun r-changement n'est possible, la solution est dite r-Opt.

Lin (1965) a introduit et étudié les procédures 2-Opt et 3-Opt et a montré que l'on pouvait obtenir des tours de bonne qualité assez rapidement. Afin d'atteindre la solution optimale, Lin a suggéré d'appliquer la procédure à différents tours initiaux générés aléatoirement.

Plus tard (1973), Lin et Kernighan ont réalisé qu'il valait mieux que r soit variable. Leur algorithme (L-K) commence par une recherche étendue par 3-Opt qui est suivie par une recherche plus locale par 2-Opt. Cet algorithme constitue une référence par rapport à laquelle tous les autres algorithmes sont testés. **[20]**

Figure 1.3.1. Procédure 2-Opt sur une instance à 5 villes.

Le temps nécessaire à la vérification de la r-optimalité est $O(N^r)$ car il existe C_N^r possibilités d'échanger r arcs. Pour chaque possibilité, il y a un nombre constant de candidats dont le calcul du coût est constant aussi.

Du fait de la qualité de leurs solutions (2% en dessous de la solution optimale pour L-K), les heuristiques r-Opt font l'objet de nombreuses recherches : elles sont souvent combinées aux nouvelles approches de résolution du TSP afin d'améliorer la qualité de leurs solutions. Elles ont également fait l'objet d'un effort de parallélisation (implémentation sur des machines à processeurs parallèles) ce qui a accéléré leur temps d'exécution à $N^r/\log(N)$ **[16]**.

Procédure or-Optimal :
Développées par Or (1976), cette procédure combine une recherche r-Opt et une procédure d'insertion d'une chaîne de moins de 3 sommets.

Les procédures mixtes

Elles consistent à obtenir un tour à partir d'une procédure de construction de tournée. Puis à affiner cette tournée en lui appliquant une procédure d'affinage.

1.2.2. Recherche tabou [14]

Cette approche introduite par Glover (1986) et par Rosier, Troyon et Liebling (1986) est basée sur l'observation suivante : Les optimums locaux ne constituent pas toujours de bonnes solutions.

Il serait alors intéressant de modifier une procédure de recherche locale (L-K par exemple) afin d'éviter les optimums locaux et de faire une recherche plus poussée.

Le principe de la recherche tabou est d'exécuter une procédure de recherche locale à partir de tours générés aléatoirement. Des changements sont effectués sur le tour de départ et à chaque itération les voisins du tour en cours sont identifiés et la

meilleure amélioration est retenue afin de fournir un tour de départ. Un problème peut alors se poser : un changement sur un tour peut conduire à un minimum local déjà visité et ainsi annuler les efforts précédents pour trouver la meilleure solution. C'est pour cela que l'heuristique Tabou stock dans des 'listes tabou' les mouvements effectués sur l'heuristique et évite ainsi à la procédure de recherche de retomber dans un minimum local déjà visité.

Il existe d'autres règles de la recherche tabou : les conditions de niveau d'aspiration qui autorisent certains mouvements interdits par la liste tabou en ayant certaines garanties. Les règles d'intensification quant à elles, permettent d'intensifier les recherches dans un voisinage d'une solution considérée comme bonne.

1.2.3. Le recuit simulé [26]

Introduit par Kirkpatrick, Gelatt et Vecchi (1983), cette heuristique est inspirée du comportement thermodynamique des atomes dans le recuit d'un matériau. Le recuit est le procédé de refroidissement lent d'un liquide, qui donne le temps aux atomes de se redistribuer dans le matériau au fur et à mesure qu'ils perdent leur mobilité en refroidissant. Cette redistribution a pour objectif la minimisation de l'énergie du système. Dans le cas du TSP, l'énergie E du système est la longueur du tour. Ce tour est modifié tout au long de l'algorithme afin d'en réduire la longueur.

Algorithme

Choisir une configuration initiale x_1 du tour.

Choisir une température initiale T et une constante k.

Répéter : déterminer une nouvelle configuration x_2 du tour dans l'espace d'états

en effectuant des changements sur x_1 et calculer $\Delta E = E(x_2)-E(x_1)$

Décision : Si $\Delta E<0$ Alors passer de l'état x_1 à l'état x_2

Sinon générer un nombre aléatoire r, $0\leq r \leq 1$

calculer $P=e^{\left(\frac{-\Delta E}{KT}\right)}$

Si $r< P$ Alors passer de l'état x_1 à l'état x_2

Sinon rester à l'état x_1

Si nombre d'itérations > max. itérations ou $x_{nouveau} = x_{ancien}$

Pour M itérations consécutives Alors réduire la température.

Jusqu'à ce que : E reste inchangée pendant N itérations consécutives■

Le recuit simulé autorise, avec une certaine probabilité, des transformations du tour qui augmentent sa longueur. Cela permet d'éviter les minima locaux en prenant le risque d'aller dans des zones qui, à première vue, ne réduisent pas la longueur du tour.

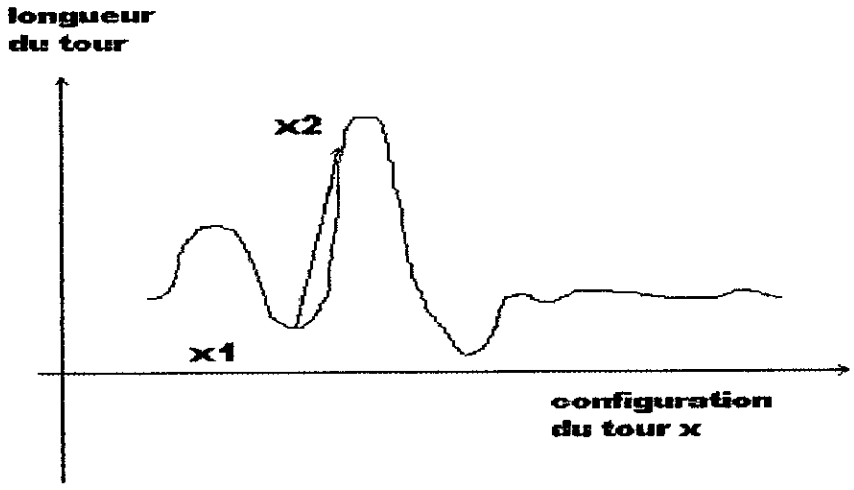


Figure 1.3.4. Déplacement autorisé de l'état x_1 à l'état x_2 avec une probabilité P.

1.2.4. Space filing curve [4]

Cette approche que l'on peut traduire par 'courbe couvrant l'espace' a été introduite par J. Bartholdi et L. Platzman en 1982. Spacefillling curve est une correspondance entre un espace de petite dimension vers un espace de plus grande dimension. Une spacefillling curve connue est celle de Sierpinski, elle est formée de motifs simples qui sont recopiés et rétrécis plusieurs fois. Une propriété intéressante de ce type de courbes est qu'elles ont tendance à visiter tout les points d'une région, une fois qu'elles y pénètrent. Par conséquent, deux points proches dans le plan auront tendance à être proches le long de la courbe.

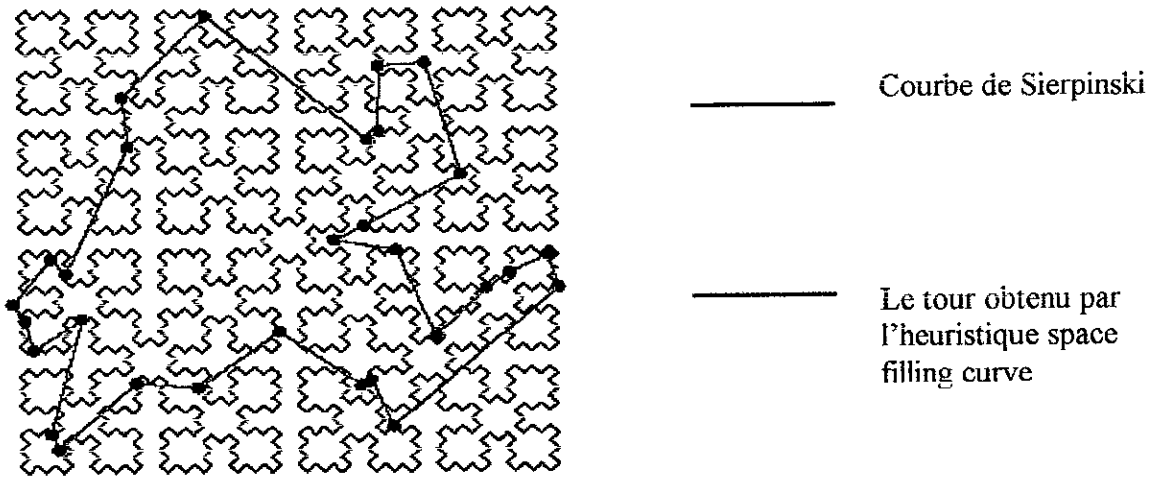


Figure 1.3.2. Solution heuristique du TSP où les villes sont visitées dans le même ordre que celui défini par la courbe de Sierpinski. [4]

Ces courbes sont à la base de l'heuristique. En effet, afin d'obtenir un tour raisonnablement petit, il suffit de visiter les villes dans l'ordre dans lequel la courbe les visites. Les tours obtenus sont, en moyenne, sont plus long que la solution optimale de 25%, en moyenne.

Le temps de calcul d'un tel algorithme est O(N logN), quant au temps de mise à jour de la solution, lorsque de nouvelles villes sont ajoutées, il est O(N logN). Ce type d'heuristiques n'a pas recours à la matrice des distances inter villes, il suffit de connaître leurs positions dans le plan. De plus, ces algorithmes sont facilement parallèlisables.

Cette heuristique fut utilisée par la croix rouge américaine pour concevoir ses tournées pour la distribution du sang dans les hôpitaux ainsi que pour l'optimisation d'un pinceau traceur de routes sur des cartes géographiques, le temps de traçage fut réduit de 10 heures à une demi-heure (université de Tokyo) **[4]**

1.2.5. Optimisation par Colonies de fourmis [11]

Le premier système s'inspirant des colonies de fourmis a été introduit par Marco Dorigo dans sa thèse de Ph.D. (1992), il lui donna le nom de système fourmi. C'est le résultat de la recherche en intelligence informatique appliquées à l'optimisation combinatoire menée à Polytechnico di Milano en collaboration avec Alberto Colorni et Vittorio Maniezzo. Ce système à été appliqué au TSP et au problème d'affectation quadratique. **[29]**

Plus récemment (1996), Dorigo et Gambardella ont travaillé sur de nouvelles versions du système fourmi. L'optimisation par colonie de fourmi est l'une d'entre elles. Appliquée au TSP, cette méthode donne de très bons résultats. Une version hybride combinant les colonies de fourmis et une recherche locale surpasse tous les algorithmes connus concernant le problème d'affectation quadratique et le problème d'ordonnancement sur un ensemble vaste de problèmes test.

Le système fourmi s'inspire du comportement des vraies fourmis. Ces dernières sont capables de trouver le plus court chemin entre leur nid et la source d'aliments. Une fourmi dépose une certaine quantité de phéromone sur son chemin. Les fourmis suivantes auront une probabilité d'emprunter ce chemin qui sera proportionnelle à la quantité de phéromones déposée.

$$P_K(r,s)=\frac{T(r,s)[E(r,s)]^b}{\sum T(r,s)[E(r,s)]^b} \quad \text{si} \quad S \notin M_K$$

Où M_K est la mémoire de la fourmi k.

T(r, s) est la quantité de phéromones déposée entre les villes r et s.

E(r, s) est l'inverse de la distance entre r et s, appelé aussi la visibilité.

La fourmi choisit son prochain arrêt en fonction de la probabilité affectée à chaque ville non encore visitée. Une fois le tour d'une fourmi terminé, elle aura déposé une quantité Q de phéromones sur son trajet. Plus le trajet est court et plus la quantité de phéromones déposée est grande, les fourmis suivantes favoriseront alors les trajets les plus courts. Il faut noter qu'après chaque cycle -passage d'une fourmi- une certaine quantité de phéromones se sera évaporée. **[29]**

1.2.6. Algorithmes Génétiques

Les algorithmes génétiques sont des algorithmes informatiques inspirés de la théorie néo- darwinienne de l'évolution des espèces, qui veut que les individus les plus aptes à survivre dans leur milieu soient sélectionnés en se reproduisant plus vite que les autres.

Dans ce cadre, les solutions réalisables du TSP (les trajets) sont des ***individus***, elles sont représentées sous formes d'une séquence de ***chromosomes*** : les villes à visiter. Ces individus appartiennent à une ***population***.

L'algorithme présenté ci-dessus est une amélioration de celui introduit par Holland (1975) dans le sens où il comprend une procédure de recherche locale destinée à améliorer la qualité des individus à chaque étape. **[14]**

Algorithme

1) Générer une population initiale S composée de k individus.
2) Appliquer un algorithme de recherche locale à chaque solution afin de l'améliorer. Les solutions améliorées remplaceront les solutions initiales.
3) Tant qu'il n'y a pas convergence, faire :

3.1) **Sélectionner** k' couples distincts de S. les couples sont deux trajets parents, tous les individus doivent avoir la même chance d'être sélectionnés afin d'assurer la diversité de la population. Cependant, pour une meilleure convergence de l'algorithme, il est préférable que les trajets les plus courts aient plus de chances d'être sélectionnés.

3.2) Les parents sont combinés pour donner naissance à un nouvel individu via un opérateur de **crossover** (croisement). Un facteur de ***mutation*** (modification aléatoire du code génétique d'un individu) est introduit afin de préserver la diversité de la population et d'éviter les minimums locaux.
3.3) Appliquer un algorithme de recherche locale aux k' trajets fils qui constitueront les éléments de S'.
3.4) Sélectionner k survivants parmi S U S' et les placer dans S. Aller en 2).

5) Placer la meilleure solution dans l'ensemble S. **[14]**

Exemple : Soit un TSP à 10 villes dont les deux solutions réalisables **a** et **b** sont données par la séquence des villes à visiter :

a=[**1**-**2**-**3**-**4**-5-6-7-8-9-10] + b=[2-**7**-1-4-**10**-**6**-**9**-**8**-3-**5**]=> f = [1-2-3-4-10-6-9-8-5-7]

1.2.7. Les réseaux de neurones artificiels

Les approches heuristiques neuronales se divisent en deux classes principales :

Dans la première, le réseau répond à une formulation du TSP en programmation linéaire en nombres entiers. Dans cette classe, nous retrouvons la première application des réseaux de neurones au TSP que nous devons à Hopfield et Tank (1985).

La seconde classe concerne les instances géométriques du TSP : les neurones sont ici considérés comme des points de l'espace cherchant à s'identifier aux villes. Les deux variantes basiques de cette classe sont : les réseaux élastiques de Durbin et Willshaw (1987) et les cartes auto organisatrices (de Kohonen) développées par Angéniol et al (1988).

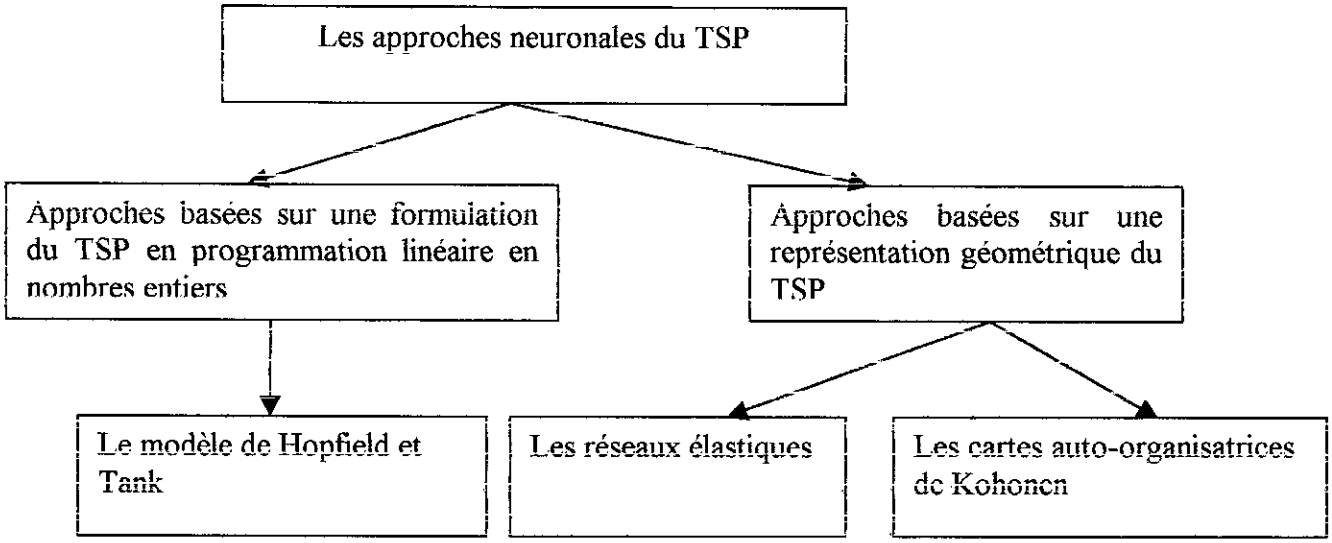


Figure 1.5.7. Typologie des approches neuronales pour le TSP.

Dans la mesure où nous n'aborderons que les approches basées sur une représentation géométrique du TSP, nous présentons ici brièvement l'approche de Hopfield et Tank qui a ouvert la voie aux méthodes neuronales basées sur une formulation en programmation linéaire en nombres entiers du TSP.

Pour la première fois en 1982, J.Hopfield introduit une fonction d'énergie dans la théorie des réseaux de neurones artificiels. Ce réseau dont le but est de minimiser cette fonction d'énergie est issu du modèle des verres de Spin de Ising (cf. Annexe I)

Ce réseau est composé de N^2 neurones (N étant le nombre de villes) tous interconnectés et disposés comme des éléments d'une matrice N*N. Les lignes de cette matrice représentent les villes x et les colonnes leur position i dans le tour obtenu **[10]**. Les états des neurones sont 0 ou 1, l'état du neurone V_{xi} sera égal à 1 si la ville x est visitée à la $i^{ème}$ étape. La fonction d'énergie à minimiser est définie par :

$$E=\frac{A}{2}\left(\sum_{x}\sum_{i}\sum_{\substack{j\\ j\neq i}}V_{xi}V_{xj}\right)+\frac{B}{2}\left(\sum_{y}\sum_{x}\sum_{\substack{y\\ y\neq x}}V_{xi}V_{yi}\right)+\frac{C}{2}\left(\sum_{x}\sum_{i}V_{xi}-N\right)^2+\frac{D}{2}\sum_{y}\sum_{\substack{y\\ y\neq x}}\sum_{i}D_{xy}V_{xy}\left(V_{yi+1}+V_{yi-1}\right)$$

Où V_{xj}= 1 si la ville x est visitée à la $j^{ème}$ étape.

- Le premier terme contraint le réseau à n'activer qu'un seul neurone par ligne.
- Le second terme contraint le réseau à n'activer qu'un seul neurone par colonne.
- Le troisième terme contraint le réseau à n'activer que N neurones dans tout le réseau.
- Quant au quatrième terme, il fait en sorte de favoriser les trajets les plus courts.

Les poids des connexions entre neurones sont alors les suivants :

$$T_{xi,yj}=\ ^{A}\delta_{xy}(1-\delta_{ij})\ ^{B}\delta_{ij}(1-\delta_{xy})\ ^{C}\ ^{D}D_{xy}(\delta_{ji+1}+\delta_{ji-1})$$

Où δ_{ij} est le symbole de Kronecker : δ_{ij}=1 si (i=j) et 0 sinon.

Une matrice initiale booléenne (NxN) représentant l'état des neurones est introduite dans le réseau. A partir de cet état initial, le réseau calculera sa matrice de sortie, constituée des états des neurones, qui sera réinjectée au réseau. Dans un processus récurrent, les sorties sont calculées et réinjectées en entrée jusqu'à ce que le réseau converge vers un état stable.

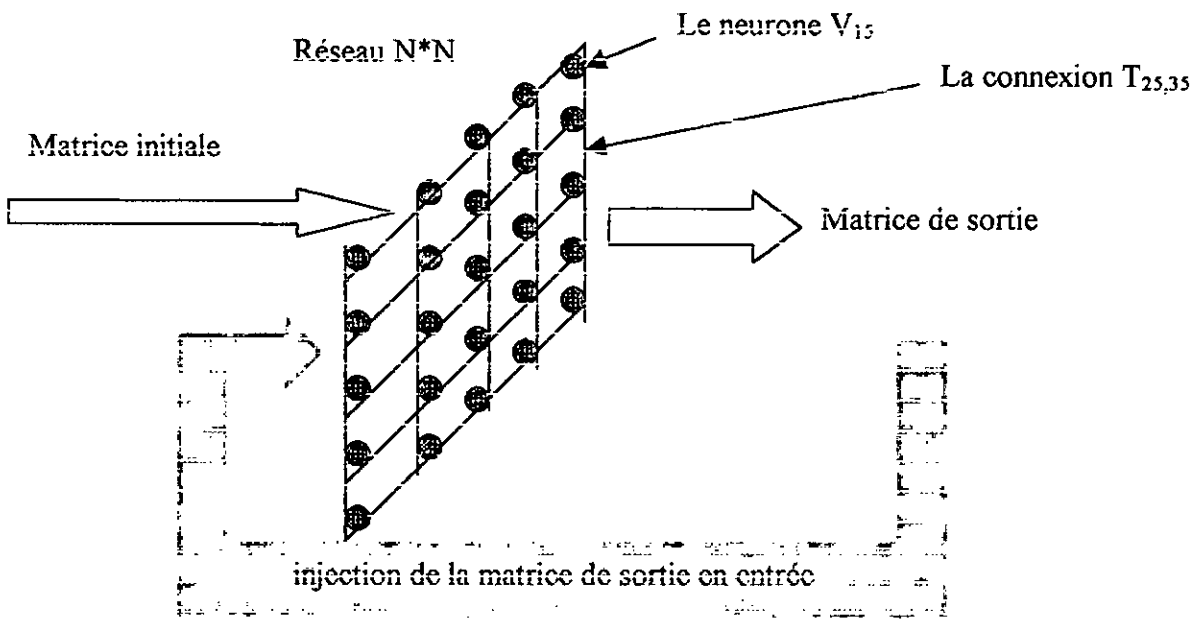


Figure 2.2.1. Schéma de fonctionnement du réseau de Hopfield.

Résultats et limitations

Sur 20 instances à 10 villes, l'algorithme trouve 16 solutions réalisables dont la moitié sont des solutions optimales. Par contre, pour des instances à 30 villes, l'algorithme rencontre des problèmes de convergence vers une solution réalisable et les seules solutions réalisables obtenues ont un écart de 17% de la solution optimale. **[31]**

De plus, ce type de formulation est tributaire de l'ajustement de 4 paramètres (A,B,C et D) à chaque taille du problème. C'est pour cela que réitérer l'expérience de Hopfield et Tank est fastidieux. Wilson et Pawley (1988) s'y sont essayés et n'ont pas pu aboutir à des solutions valides. Sur 100 essais, seules 15 solutions réalisables ont été trouvées et leur qualité ne dépassait pas celle de tours générés aléatoirement. **[13]**

Le réseau nécessitant N^2 neurones tous interconnectés, il faut mettre à jour N^4 connexions ce qui est considérable en terme de temps de calcul et de capacité de stockage.

Des travaux tentant de déterminer les paramètres de l'algorithme ont vu le jour, parmi ces travaux nous pouvons citer ceux de Bhide et Kabuka **[5]** et ceux de Cavalieri et Russo **[8]** faisant appel à la logique floue.

1.3. Comment évaluer les performances des heuristiques ?

1.3.1. Estimation de la borne inférieure

Lors de l'évaluation des performances empiriques d'une heuristique, il n'est pas toujours possible de comparer les solutions obtenues à la solution optimale. C'est pour cela que, pour de grandes instances du TSP euclidien, où les villes sont uniformément distribuées dans le carré [0, 1], la pratique est de comparer les résultats de l'heuristique à une valeur que nous pouvons calculer : la borne inférieure du tour optimal.

Depuis que Beardwood, Halton et Hammersley (1959) ont démontré que le ratio de la solution optimale à $\sqrt{N}$ approchait une constante C_{opt} lorsque $N \to \infty$, plusieurs estimations de C_{opt} ont été faites. Une estimation faite par Bearwood et al donne $C_{opt} \sim 0.749$. **[14]**

Stein, quant à lui, estime C_{opt} à 0.765 et situe la solution optimale dans l'intervalle :
$[0.765\sqrt{N}, (0.765+\frac{N}{4})\sqrt{N}]$ **[14]**

Des expériences récentes faites par Percus et Martin (1996) suggèrent que :

$C_{opt} \sim 0.7124$. **[14]**

La borne H_K est la solution du problème linéaire, modélisant le TSP, relaxé de ses contraintes. Elle a été calculée empiriquement **[15]** par Johnson, McGeoch et Rothberg (1996) :

Pour N>100, $C_{HK}(N) \cong 0.70805 + \frac{0.52229}{\sqrt{N}} + \frac{1.31572}{N} - \frac{3.07474}{N\sqrt{N}}$

La borne de Held et Karp semble donner une bonne estimation de la solution optimale OPT(I), bien qu'elle soit, théoriquement, supérieure à 2*OPT(I)/3.

Il a été démontré **[15]** que la borne H-K, pour des problèmes test réels, approximait la solution optimale avec une précision inférieure à 2%. Concernant les instances générées aléatoirement, la précision est en deçà de 0.8%, de plus la borne H-K fournit une réduction de la variance lorsque des instances aléatoires sont utilisées.

Il existe d'autres méthodes d'estimation de la borne inférieure de la solution optimale telles que la méthode de Golden qui suppose que les longueurs de circuits (population) suivent une loi de Weibull et plus récemment, la méthode des moments qui estime la borne inférieure à partir des caractéristiques de la population. **[2]**

Analyse des performances

Lorsqu'une nouvelle heuristique est testée, il est nécessaire de connaître ses performances. Dans ce qui suit nous parlerons de la déviation par rapport à l'optimum qui est l'erreur relative produite par l'heuristique par rapport à l'optimum ou à sa borne inférieure :

R= (solution heuristique-optimum)/optimum. Exprimée en pourcentage.

1.3.1. Analyse empirique [19]

Ce type d'analyse fut l'unique méthode d'évaluation des algorithmes pendant des années. Il s'agit de comparer les résultats d'un algorithme sur un ensemble de problèmes test à la solution optimale ou alors à la borne inférieure si la solution optimale est inconnue ou encore aux résultats obtenus avec le meilleur algorithme. Les temps de calculs de ces différents algorithmes sont eux aussi comparés entre eux. Ce type d'analyse est le plus répandu mais il se heurte à la difficulté suivante :

Il faut que les problèmes test générés soient représentatifs sinon l'interprétation des résultats dépendra de la distribution statistique des instances du problème.

1.3.2. Analyse du plus grand écart par rapport à l'optimum [19]

Il s'agit de calculer la pire déviation ou éloignement par rapport à l'optimum que l'algorithme peut produire. Ce type d'analyse procure une garantie de performance à l'heuristique considérée. En effet, quelle que soit l'instance du TSP considérée, nous savons dans quel intervalle se trouvera la solution.

Sahni et Gonzalez (1976) ont montré, pour le TSP, que garantir une erreur maximale à un algorithme était aussi difficile que déterminer la solution optimale.

Le problème avec ce type d'analyse est que l'heuristique est jugée sur ses plus médiocres performances. L'étude empirique des heuristiques suggère qu'un tel comportement est extrêmement rare, c'est à dire que les résultats les plus médiocres sont obtenues sur des instances particulières ('pathologiques ') du TSP et ne reflètent pas le comportement de l'algorithme.

1.3.3. Analyse probabiliste [19]

Il s'agit d'une approche théorique consistant à prouver que certaines méthodes procurent des solutions proches de l'optimum lorsque le nombre de villes devient grand. Cela suppose, d'une part, que les instances du problème sont issues d'une certaine loi de probabilité et, d'autre part, que le coût de la solution optimale est estimé. Dans le cas du TSP euclidien, où les villes sont uniformément et indépendamment distribuée dans le carré [0, 1], le coût de la solution optimale tend vers $C_{opt}\sqrt{N}$, N étant le nombre de villes et C_{opt} une constante.

Les résultats de l'analyse probabiliste sont asymptotiques et ne sont valables que pour un nombre de villes suffisamment grand. Alors que nous ignorons tout sur les propriétés de convergence de ces heuristiques "asymptotiquement efficaces". Il serait bon de connaître le comportement de l'heuristique pour de vraies valeurs de n (nombre de villes).

L'analyse probabiliste doit être vue comme une explication du comportement observé de l'heuristique étudiée plutôt que comme une garantie de performance.

2.1. Introduction au connexionisme

Qu'est ce que le connexionisme ? [18]

Le connexionisme, les réseaux de neurones ou encore les sciences neuromimétiques sont autant de noms qui désignent une technique découlant du **cognitivisme**. Ce dernier, issu du latin cognito qui signifie connaissance, désigne un ensemble de théories, issues de l'intelligence artificielle et de la cybernétique, portant sur les processus d'acquisition des connaissances.

Les sciences cognitives font appel à la biologie, à la psychologie, à la linguistique, à la logique et à l'informatique. Elles ont pour objet la connaissance, la mémoire, la perception et le raisonnement. Elles ont recours à l'ordinateur et à l'intelligence artificielle pour modéliser le comportement de l'esprit humain.

" Le connexionisme est cette nouvelle branche de l'informatique où les calculs sont effectués par des réseaux de neurones artificiels. Les informaticiens simulent les neurones et leurs connexions sur des ordinateurs (tout comme les ingénieurs créent des modèles virtuels d'ailes d'avion ou de gratte-ciel). Un algorithme d'apprentissage ajuste les connexions entre neurones, pour que le réseau, dédié à une tâche précise, l'accomplisse de mieux en mieux" [inter-4]

Les connexionnistes modernes considère Frank Rosenblatt, qui publia le premier article sur le sujet en 1957, comme le fondateur de la discipline; or, dès 1948, Alan Mathison Turing avait étudié les réseaux de neurones dans un article intitulé *Intelligent Machinery*.

Chronologie succincte de l'évolution du connexionisme [3], [18]

1943: naissance des réseaux de neurones artificiels, en tant que discipline, lors de la parution des travaux de MacCulloch et Pitts, deux biophysiciens dont l'ambition était de modéliser le fonctionnement du système nerveux.
1948: Turing invente sa fameuse machine de type B, précurseur des réseaux de neurones artificiels.
1949: vinrent les travaux de Hebb qui énonça une règle qualitative expliquant le mécanisme d'apprentissage par le biais de modifications au niveau synaptique.
1959: Rosenblatt présente le premier réseau destiné au traitement de l'information: le perceptron. Celui-ci est conçu à partir des informations disponibles sur le système visuel biologique et est exploité pour la reconnaissance de formes.
1960: Widrow et Hoff proposent l'Adaline qui deviendra le paradigme des réseaux multicouches.
1969: Minsky et Papert démontrent qu'avec un réseau mono-couche on ne pouvait séparer que des données linéairement séparables et qu'il fallait aller vers des réseaux multicouches. A la suite de ses révélations, la recherche connaîtra une certaine stagnation.
1970: Hérault propose un modèle électronique de la transmission synaptique et Aleksander développe un micro-circuit modélisant une cellule nerveuse.
1972: Kohonen travaille sur les mémoires associatives et les associateurs linéaires ainsi que leur application à la reconnaissance de formes.

1977: Amari réalise une étude statistique d'une population de neurones pour tenter de comprendre le comportement collectif des neurones.
1980: Wilkie, Stonham et Aleksander créent le système WISARD qui résout le problème de la séparation linéaire en incorporant plus d'une couche dans le réseau. A partir de là la recherche en matière de réseaux neuronaux connaît un véritable essor.
1981: McClelland et Rumelhart proposent un modèle basé sur le comportement compétitif des populations de neurones.
1982: Hopfield analyse la dynamique des réseaux complètement rebouclés et établie une analogie avec le modèle physique des verres de Spin.
Kohonen présente un modèle d'auto-organisation dans lequel le réseau est capable de développer une organisation à partir de stimulations.
Ce dernier modèle sera largement exploité et développé par la suite.
1986: Le Cun, Rumelhart et Hinton proposent une règle de calcul des connexions pour les réseaux multicouches: la règle de rétro-propagation du gradient.

En dépit de leur apparente simplicité, les réseaux de neurones artificiels ont des comportements spécifiques au vivant: faculté d'apprendre par l'exemple, c'est-à-dire de s'auto-organiser en vue de l'accomplissement d'une tâche sans disposer de règles connues a priori, capacité à traiter des données incomplètes, des informations bruitées ou floues, avec une rapidité d'exécution certaine. Nous allons introduire les éléments naturels dont s'inspirent les réseaux de neurones artificiels et qui font leur particularité.

2.2. Fondements biologiques

Nés du désir de comprendre le fonctionnement du cerveau, les premiers modèles de réseaux de neurones artificiels ont été conçus de façon à reproduire les mécanismes biologiques tels qu'ils étaient connus au début des années 40. Certes ces modèles - qui à de minimes différences près sont encore ceux utilisés aujourd'hui - apparaissent désormais, au regard des progrès accomplis par la neurobiologie au cours des 50 dernières années, comme des représentations extrêmement simplifiées des processus électriques et chimiques complexes (et encore partiellement incompris) mis en jeu dans le vivant.

2.2.1. Le neurone naturel [1]

Le neurone est une entité cellulaire qui constitue l'élément de base du système nerveux. Le nombre estimé de neurones dans le système nerveux est de plus de 1000 milliards. Leurs formes et certaines caractéristiques permettent de les répartir en quelques grandes classes. En effet, les neurones se spécialisent dans telle ou telle tâche suivant leur position dans le cerveau.

Malgré leur diversité, les neurones possèdent la même structure de base, à savoir:

Un corps cellulaire (soma): Il contient le noyau du neurone ainsi que la machine biochimique nécessaire à la synthèse des enzymes.

Des dendrites : qui servent à capter les signaux envoyés au neurone.

Un axone: C'est le long de l'axone que les signaux partent du neurone pour atteindre un autre neurone.

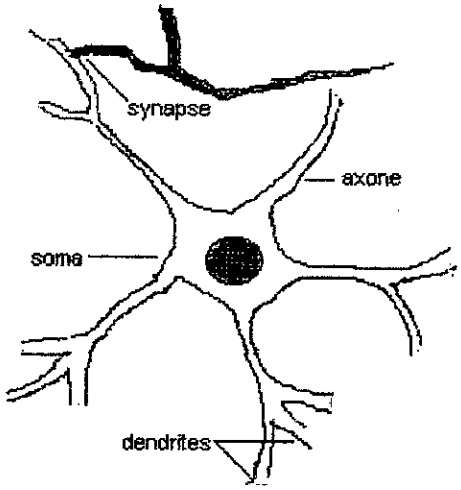


Figure 2.1. Le neurone naturel

La synapse : La synapse est la zone d'accolement de la terminaison d'un axone avec la dendrite d'un autre neurone, avec un autre axone ou bien avec un corps cellulaire.

Cette richesse de possibilités en terme de connexions donne une idée de la complexité des structures pouvant se constituer dans le cerveau au gré de l'évolution et de l'adaptation.

2.2.2. Le cerveau et l'auto-organisation

Les biologistes estiment que le cerveau contient des milliards de neurones, qu'il fonctionne à une fréquence de 1000 impulsions par seconde et qu'il reçoit des signaux d'entrée et produit des réponses en effectuant certains apprentissages et en s'auto-organisant. Cette capacité d'apprentissage tient de la plasticité synaptique et donc de l'aptitude du cerveau à modifier ses efficacités synaptiques. En effet, les synapses servent à limiter les signaux qui passent d'un neurone à l'autre, elles favorisent, donc ou défavorisent les signaux qui les traversent. La règle de Hebb qui renforce les corrélations les plus importantes entre les neurones est l'une des premières à avoir été formalisées, et elle semble d'ailleurs être utilisée dans le cerveau. **[18]**

La notion de population de neurones définie par Hebb a permis de décrire une sorte de coopération entre neurones. Chacun des neurones de cette population est nécessaire pour maintenir l'activation des autres. On retrouve l'opposé de cette notion pour décrire des neurones ayant un fonctionnement concurrentiel. Dans un ensemble de neurones concurrentiels, lorsque seul un neurone reste actif, on dit qu'il y a eu un phénomène de type "winner takes all". **[3]**

2.3. Les réseaux de neurones artificiels

Tout comme dans le cas des réseaux de neurones naturels, l'unité de base des réseaux de neurones artificiels est le neurone.

2.3.1. Le neurone formel [1]

Le neurone est une unité de calcul ou de traitement qui procède en quatre étapes:

1- il reçoit, en entrée, des données (e_i) provenant d'autres neurones à travers des synapses (connexions inter neurones)

2- il applique à ces données une fonction d'entrée, le plus souvent une somme des entrées pondérée par les poids synaptiques W_{ij}

3- sa fonction d'activation permet d'évaluer son état interne en comparant le résultat de la fonction d'entrée avec son seuil d'excitation ou en lui appliquant sa fonction d'activation.
Cette fonction d'activation peut être la fonction signe, la fonction de Heaviside ou encore une fonction sigmoïde.

4- en fonction de cet état, le neurone produira une sortie qui à son tour sera transmise à d'autres neurones ou alors constituera la sortie du réseau.

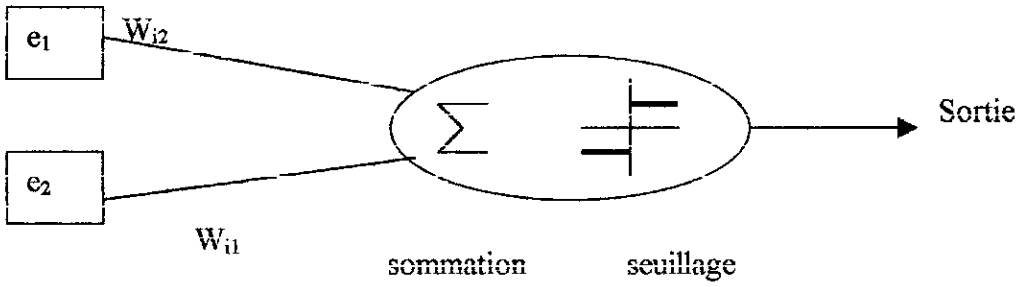


Figure 2.2. Schéma de fonctionnement d'un neurone formel

Dans la figure 2.2, nous avons représenté le fonctionnement du neurone de McCulloch et Pitts dont la fonction d'entrée est la somme pondérée des composantes du vecteur d'entrée. La fonction d'activation est la fonction signe, ses valeurs possibles sont 1 et –1, c'est à dire que si le résultat de l'entrée est supérieur au seuil 0 alors le neurone est activé, sinon il ne l'est pas.

2.3.2. Un ensemble de neurones : le réseau

Le neurone formel ne pouvant exécuter qu'une sommation et un seuillage, il est intéressant de combiner plusieurs neurones afin d'obtenir un système capable d'opérations plus complexes. On imite ainsi la structure du cerveau en faisant travailler ensemble plusieurs neurones interconnectés.

Qu'est ce qu'un réseau de neurones artificiels?

Un réseau neuronal est un système informatique inspiré de l'organisation des cellules du cerveau humain. Ce réseau est à l'origine une tentative de modélisation mathématique du cerveau.

L'idée principale des réseaux de neurones est la suivante:

On se donne une unité simple, le neurone, qui est capable de réaliser quelques calculs élémentaires à partir de données numériques uniquement (en aucun cas symboliques). On relie ensuite entre elles un nombre important de ces unités, sous forme de couches interconnectées, afin d'obtenir un système d'une grande puissance de calcul. Ces neurones sont des unités abstraites, au départ, que l'on simule sur un ordinateur classique dit séquentiel. Ces unités ont 3 fonctions de base: entrée d'un signal, traitement des données et sortie des résultats.

En dépit de leur ``apparente" simplicité, les réseaux de neurones artificiels exhibent des comportements spécifiques au vivant : faculté d'apprendre par l'exemple, c'est-à-dire de s'auto-organiser en vue de l'accomplissement d'une tâche sans disposer de règles connues a priori, capacité à traiter des données incomplètes, des informations bruitées ou floues, le tout avec une rapidité d'exécution. **[18]**

De nos jours, les réseaux de neurones artificiels ne se contentent plus uniquement de modéliser les réseaux cérébraux (afin de mieux les comprendre) **[17].** Les réseaux artificiels représentent plutôt des unités fonctionnelles du cerveau telles que

- La vision,
- La perception,
- La prise de décision (raisonnement, estimation, résolution de problèmes)
- La reconnaissance de la voix et la compréhension de discours,
- Les fonctions motrices (manipulation et mouvement).

La modélisation de ces unités fonctionnelles artificielles aurait pour but la conception de robots autonomes.

Bien souvent, et notamment en physique des particules, il est possible d'envisager les réseaux de neurones artificiels d'un point de vue purement mathématique, comme de simples algorithmes statistiques, particulièrement adaptés à la résolution de problèmes de combinatoire (reconstruction de traces, clusters...) ou de discrimination multi-variable (analyse discriminante), sans jamais avoir recours à l'analogie biologique.

Les trois catégories de réseaux de neurones artificiels : **[17]**

Les réseaux à transfert de signal :
Dans ce type de réseaux, le signal de sortie dépend uniquement du signal d'entrée. Les réseaux appartenant à cette catégorie sont : les réseaux à couche utilisant la propagation-avant tels que le Perceptron et l'Adaline, les réseaux à rétro-propagation du gradient ou encore les réseaux à fonction de base radiale.

Les réseaux à transfert d'état :
La présence de *feedback* dans ces réseaux fait converger l'état d'activité des neurones vers des états stables, des *attracteurs*. Un représentant typique de cette catégorie est le réseau de Hopfield.

Les réseaux compétitifs ou auto-organisés :
Les neurones de ce réseau, en recevant les mêmes entrées, réagissent différemment c'est à dire qu'ils produiront différents niveaux d'activations. Un mécanisme de compétition détermine le neurone vainqueur. Ce vainqueur apprendra à reconnaître le type d'entrée pour lequel il a plus de disposition. La modification des poids du vainqueur fera en sorte qu'il devienne sensible à l'entrée à laquelle il doit la victoire.

Relation entre les réseaux de neurones artificiels et naturels **[17]**

Nous listons ici quelques caractéristiques communes des réseaux de neurones naturels et artificiels,

- Une représentation et un traitement analogues des informations. La représentation des données est distribuée et les traitements sont simples et peuvent être asynchrones.
- Une tolérance aux données bruitées ou partiellement dégradées.
- La capacité de faire une approximation de la distribution des données (traitements statistiques)
- Une adaptation aux changements de l'environnement et l'émergence d'un traitement intelligent de l'information, à travers l'auto-organisation, en réponse aux données d'entrée.

2.4. Les champs d'application des réseaux de neurones artificiels

Les réseaux de neurones artificiels consistent en un ensemble d'outils et de méthodes de calcul, pouvant être appliqués **[25]** à des domaines aussi divers que :

- le traitement d'information, le traitement de signal (image, parole),
- la classification, la compression et le cryptage de données,
- la statistique,
- la prédiction de séries temporelles,
- la reconnaissance de formes,
- l'apprentissage par l'exemple,
- la mémorisation,
- la généralisation
- le contrôle de processus,

- et l'optimisation.

Les nombreuses applications industrielles des réseaux de neurones artificiels apparues depuis quelques années justifient l'intérêt grandissant porté à ce domaine, et font oublier le scepticisme qui était de mise il n'y a pas encore si longtemps.

3.1. Introduction

Après avoir introduit les mécanismes d'auto-organisation que le cerveau peut mettre en oeuvre, nous nous intéresserons dans ce chapitre à l'une des deux approches neuronales basées sur une représentation géométrique du TSP euclidien et s'inspirant du concept de cartes auto-organisées.

Le type de réseaux étudié dans ce chapitre est le réseau élastique (Elastic Net) introduit par Durbin et Willshaw en 1987. Ce type de réseaux s'inspire des cartes auto-organisatrices de Kohonen (1982) où les neurones sont capables de modifier leurs poids afin d'épouser la structure de données qui leur est présentée.

Nous commencerons par présenter les réseaux élastiques et leur approche de résolution du TSP, puis nous donnerons les principales améliorations effectuées sur ce type d'approches en présentant les résultats expérimentaux donnés par les auteurs de chacune. Nous conclurons en faisant une synthèse de ce type d'approche.

3.2. Présentation du réseau élastique

Ce réseau (Elastic Net) introduit par Durbin et Willshaw (1987) s'inspire des cartes auto-organisatrices de Kohonen (cf. chapitre 4). Il s'agit d'un réseau à une seule couche dont les neurones sont interconnectés et forment un anneau. Ces neurones sont identifiés par leurs coordonnées dans le plan, ces coordonnées représentent leurs poids et sont aptes à s'adapter aux formes qui leur sont présentées. En entrée les neurones reçoivent les coordonnées des villes et font en sorte de s'en rapprocher. Il faut pour les problèmes d'optimisation plus de neurones que de villes, Durbin et Willshaw ont fixé le rapport entre le nombre de neurones et le nombre de villes à 2.5. **[31]**

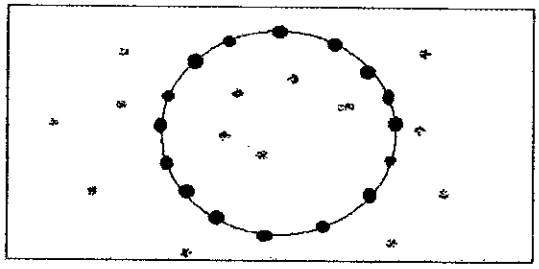


Figure 3.1. L'anneau de neurones (en rouge) et les villes (en bleu)

Outre l'optimisation géométrique, les réseaux élastiques interviennent dans plusieurs domaines : traitement de l'image (IRM, scanner), reconnaissance de formes (écriture manuscrite, expressions du visage, objets), détermination des trajectoires de particules et de robots, classification...etc.

3.3. Application au TSP [6]

Le réseau élastique ne s'applique qu'aux instances géométriques du TSP, c'est à dire au TSP euclidien. Durbin et Willshaw ont proposé l'approche suivante :

Nous disposons de n villes placées dans le carré unité et de m neurones dynamiques placés dans le même carré. Ces neurones sont disposés de sorte à former un anneau. Cet anneau à pour centre le centre de gravité des villes. Il sera étiré, c'est à dire que les neurones seront déplacés dans le plan, dans le but de minimiser une fonction d'énergie.

Ici les villes i sont représentées par leurs positions x_i dans le plan et les neurones par leurs positions y_j.

$$E = -\alpha K \sum_{i=1}^{N} \ln \sum_{j=1}^{M} \exp\left(\frac{-\|x_i - y_j\|}{2K^2}\right) + \beta \sum_{j=1}^{m} \|y_j - y_{j+1}\| \qquad (3.1)$$

Avec $K \to 0$

Où $\|x_i - y_j\|$ est le carré de la distance euclidienne entre la ville x_i et le neurone y_j dans le plan.

La fonction d'énergie est composée de deux termes :

Le premier représente l'attraction qu'exercent les villes sur les neurones. Minimiser le premier terme revient à minimiser la distance entre les neurones et les villes, en d'autres termes, rapprocher les neurones des villes afin que l'anneau épouse la configuration des villes dans le plan.

Le second terme favorise le plus court chemin en maintenant une certaine cohésion entre les neurones afin d'éviter qu'ils ne se dispersent dans le plan. En effet, le second terme représente la longueur de l'anneau de neurones, et le minimiser revient à minimiser la longueur du tour tracé par l'anneau de neurones après convergence de l'algorithme.

Nous pouvons faire une analogie avec le recuit simulé et imaginer K comme étant le facteur température, l'énergie est alors minimisée en utilisant un processus de refroidissement : $K := (1-\varepsilon)K$

Les positions des neurones sont mises à jour suivant une descente de gradient :

$\Delta Y = K \nabla_y E$ **[30]**

Soit : P l'ensemble des neurones de l'anneau et C l'ensemble des villes.

Soit : x_i la position de la ville i et y_j la position du neurone j, x_i et y_j étant des vecteurs à deux dimensions.

Le déplacement subit par le neurone j est le suivant :

$$\Delta y_j = \alpha \sum w_{ij}(x_i - y_j) + \beta K(y_{j-1} - 2y_j + y_{j+1}) \quad (3.2)$$

Ici j-1 et j+1 sont les voisins du neurone j sur l'anneau.

w_{ij} représente l'influence normalisée de la ville i sur le neurone j.

L'influence normalisée w_{ij} est un ratio mesurant l'influence de la ville x_i sur le neurone y_j. Il représente la part de l'influence de la ville x_i sur le neurone y_j sur l'influence qu'exerce cette même ville sur tous les neurones du réseau.

$$w_{ij} = \frac{\Phi(\|x_i - y_j\|, K)}{\sum_{k \in P} \Phi(\|x_i - y_k\|, K)} \quad (3.3)$$

Avec $\phi(d,K) = \exp(-\frac{d^2}{2K^2})$ (3.4)

ϕ étant une fonction d'influence qui croît lorsque la distance entre le ville et le neurone décroît.

Le déplacement Δy_j du neurone j correspond à une descente de gradient sur une fonction d'énergie dont l'optimum global résout le TSP lorsque $K \to 0$. **[30]**

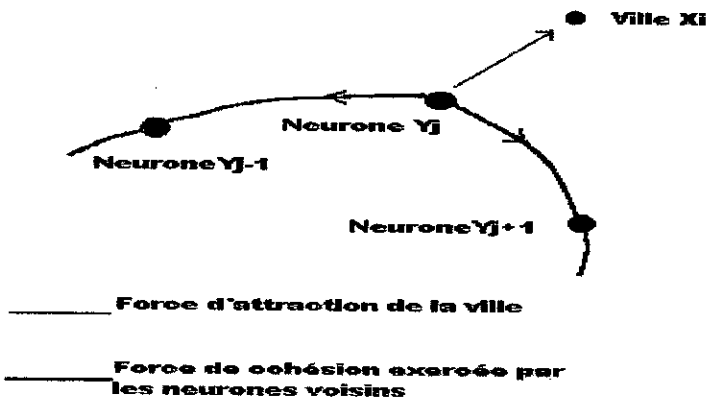


Figure 3.2. Schéma des forces agissant sur l'anneau de neurones.

Dans la formule (3.2) décrivant le déplacement subit par chaque neurone y_j, le premier terme signifie que y_j sera déplacé sous l'influence de toutes les villes xi moyennant un facteur de pondération w_{ij}. En d'autres termes, plus une ville est proche d'un neurone et plus elle contribuera à son déplacement, ce déplacement rapprochera le neurone de la ville. En effet, si le neurone y_j se situe à droite de la ville x_i alors la différence (x_i-y_j) sera négative et le neurone y_j sera déplacé vers la gauche.

Quant au second terme de la formule (3.2) il sert à éviter que le neurone y_j ne s'éloigne trop de ses voisins. En effet, ce second terme étant la somme de $(y_{j+1}-y_j)$ et de $(y_{j-1}-y_j)$ le type d'attraction décrit plus haut se produit.

Les valeurs de α et de β servent à pondérer les ampleurs relatives des deux forces agissant sur j, à savoir : la force d'attraction des villes et la force de cohésion entre les neurones.

Il a été prouvé (Durbin et al, 1989) que lorsqu'un minimum local est atteint, chaque ville est attachée à au moins un neurone, c'est à dire que chaque ville a pu attirer au moins un neurone au point qu'il se confonde avec elle. **[30]**

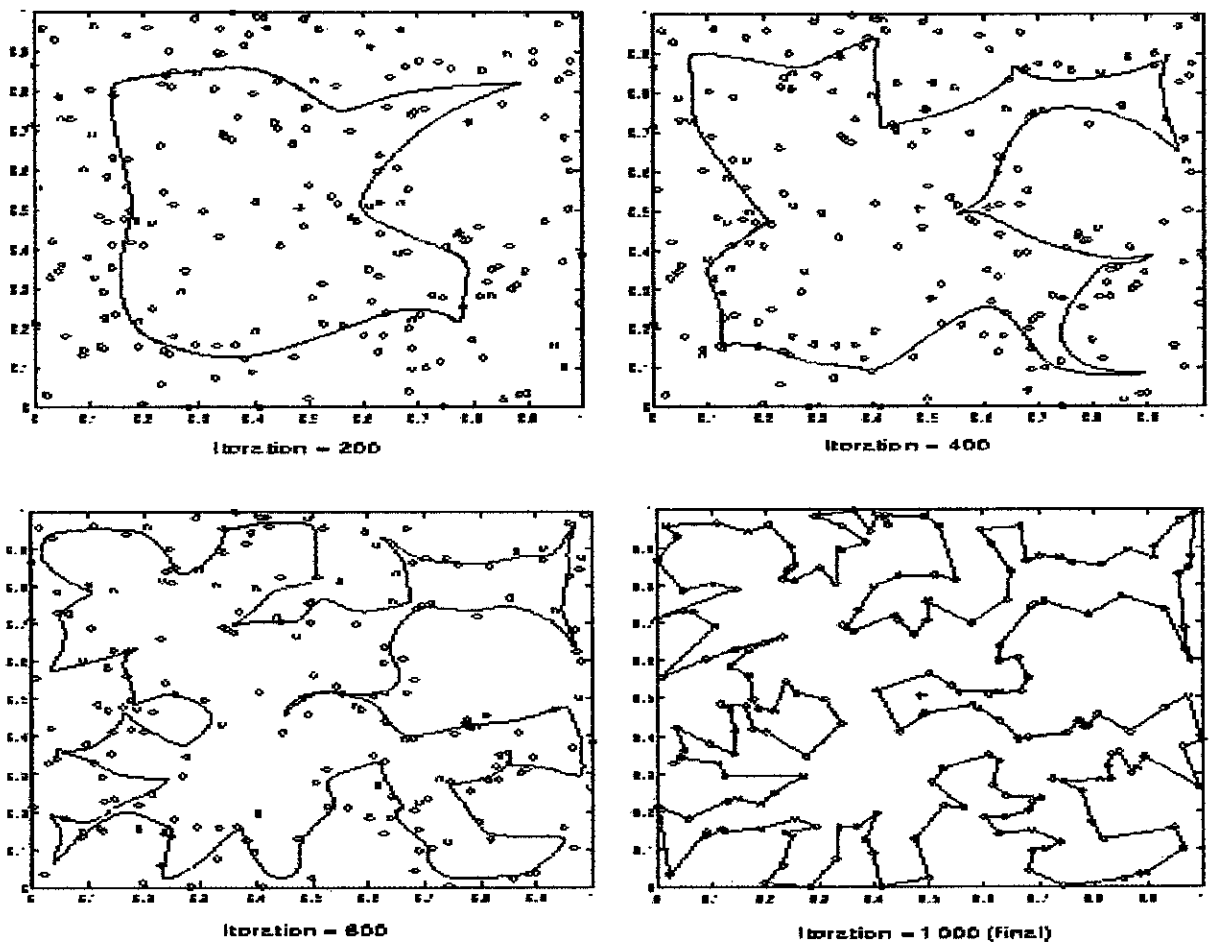


Figure 3.3: Evolution du réseau élastique sur une instance à 200 villes.**[30]**

Dans la mesure où :

- A chaque itération M.N calculs de wij et de $\phi(x_i,y_j)$ sont effectués, M étant un multiple de N.
- Le nombre total d'itérations qui est le produit de W (nombre d'itérations par tape) et de T (nombre d'étapes) est un nombre largement supérieur à N.

La complexité de l'algorithme de Durbin et Willshaw est par conséquent $O(N^3)$**[14]**

3.4. Améliorations récentes

A la suite de l'article de Durbin et Willshaw (1987), plusieurs travaux ont vu le jour et qui tentaient d'améliorer la version initiale de l'algorithme. La première tentative fût celle de Burr(1988) **[7]** qui proposa un système de forces symétriques et un rythme de refroidissement destinés à améliorer la convergence.

Nous décrirons ici trois voies d'améliorations du réseau élastique que sont : le réseau élastique avec filtre, le réseau élastique hiérarchique et le réseau élastique convexe.

3.4.1. Le réseau élastique avec filtre (Boeres, de Cavalho, Barbosa 1992)

Cet algorithme **[6]** reprend le principe du réseau élastique, décrit ci-dessus, et met en place un mécanisme de filtrage appelé γ-filtre ($\gamma \in [0, 1]$) Ce filtre ne permet les déplacements des neurones que sous une attraction significative des villes, c'est à dire que le niveau d'attraction des villes devra dépasser un certain seuil pour provoquer le déplacement des neurones.

Pour chaque valeur de K, plusieurs itérations sont effectuées. A chaque itération, tous les neurones sont déplacés. L'algorithme s'arrête lorsque au moins un neurone est affecté à chaque ville, c'est à dire qu'au moins un neurone est proche de chaque ville à une erreur près (à spécifier).

Pour des valeurs importantes de N (nombre de villes) et par conséquent de M (nombre de neurones), le réseau élastique classique devient très gourmand en temps de calcul. En effet, le déplacement d'un neurone dépend des positions de toutes les villes et à chaque itération w_{ij} doit être recalculé en fonction des nouvelles positions de tous les neurones.

Le γ-filtre à été introduit afin de limiter les déplacements des neurones aux cas ou l'influence de la ville représente 100γ % de l'influence de toutes les villes.

Description de l'algorithme

Mettre en place le filtre consiste à déterminer le sous-ensemble $C_\gamma(j, K)$ de C (ensemble des villes) dont les éléments x_i satisfont à :

$$\sum_{i\in C,(j,K)} \phi(|x_i - y_j|,K) = \gamma \sum_{i\in C} \phi(|x_i - y_j|,K) \qquad (3.5)$$

Déterminer un tel ensemble pour une disposition donnée des villes est difficile. C'est pour cela que les villes sont supposées être réparties uniformément dans le plan avec une densité ρ.

Sous cette approximation, les villes situées dans un rayon R du neurone j satisfont à :

$$\begin{aligned}\sum \phi(|x_i - y_j|,K) &\approx 2\pi p \int_{r=0}^{R} \phi(r,K)\, r dr \\ &= 2\pi p \int_{r=0}^{R} \exp\left(-\frac{r^2}{2K^2}\right) r dr \\ &= 2\pi p K^2 \left(\exp\left(-\frac{R^2}{2K^2}\right) - 1\right)\end{aligned} \qquad (3.6)$$

Une autre approximation stipule que les villes sont contenues dans un cercle de rayon $l/\sqrt{\pi}$, l étant le côté du carré contenant les villes.

L'ensemble $C_\gamma(j, K)$ est alors assimilé à un cercle centré en y_j de rayon R_γ, ce cercle est le même pour tous les neurones.

Connaissant l'expression de ϕ en fonction du rayon (3.6), nous pouvons réécrire (3.5) de la façon suivante :

$$\exp\left(\frac{-R_\gamma^2(K)}{2K^2}\right) - 1 = \gamma\left(\exp\left(-\frac{l^2}{2\pi K^2}\right) - 1\right) \qquad (3.7)$$

Le rayon R_γ est alors :

$$R_\gamma(K) = \sqrt{-2K^2 ln\left(\gamma\left(\exp\left(-\frac{l^2}{2\pi K^2}\right) - 1\right) + 1\right)} \qquad (3.8)$$

Désormais, les calculs dans les équations (3.3) et (3.4) sont effectués uniquement sur les villes x_i appartenant à $C_\gamma(j, K)$.

Résultats des simulations : **[6]**

L'algorithme a été implémenté pour différentes valeurs de γ et a été comparé au réseau élastique classique et à l'heuristique de Lin et Kernighan. Les problèmes test utilisés ont été générés aléatoirement dans le carré unité. Les tailles des instances sont les suivantes : 100, 200, 300, 400, 500, 750, 1000. l'exécution a été faite sur une SUNsparcstation.

Concernant les réseaux élastiques, l'anneau initial était un cercle de rayon 1 centré au milieu du carré unité. Le nombre initial de neurones m se situait entre N/2 et N/4 selon les valeurs de N et de β. La valeur initiale de la température K est de 0.24 et décroît de 5% à chaque étape (2% pour les instances à 750 et 1000

villes). Le nombre d'itération pour chaque valeur de K a été fixé à 2, un nombre plus grand n'ayant pas apporté d'améliorations.

Les valeurs expérimentales des paramètres α et β:

$\alpha \in [0.2, 0.8]$ et $\beta \in [0.1, 0.4]$ suivant la réaction de l'algorithme à chaque instance.

L'algorithme s'arrête dès que chaque ville possède au moins un neurone se trouvant à une distance ε d'elle. ε varie entre 0.05 pour les petites instances et 0.02 pour l'instance à 1000 villes.

Les valeurs présentées en histogrammes sont des valeurs moyennes, calculées par les auteurs, sur 10 instances de même taille générées aléatoirement.

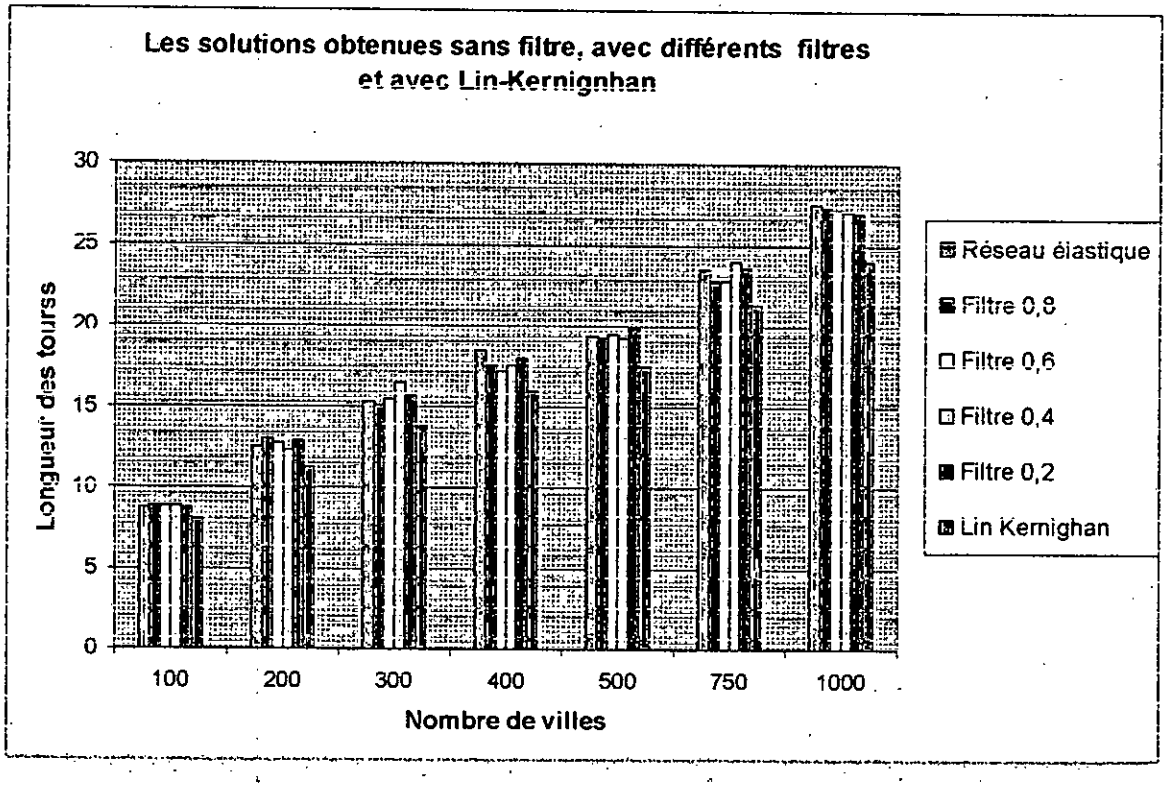


Figure 3.4. (a)

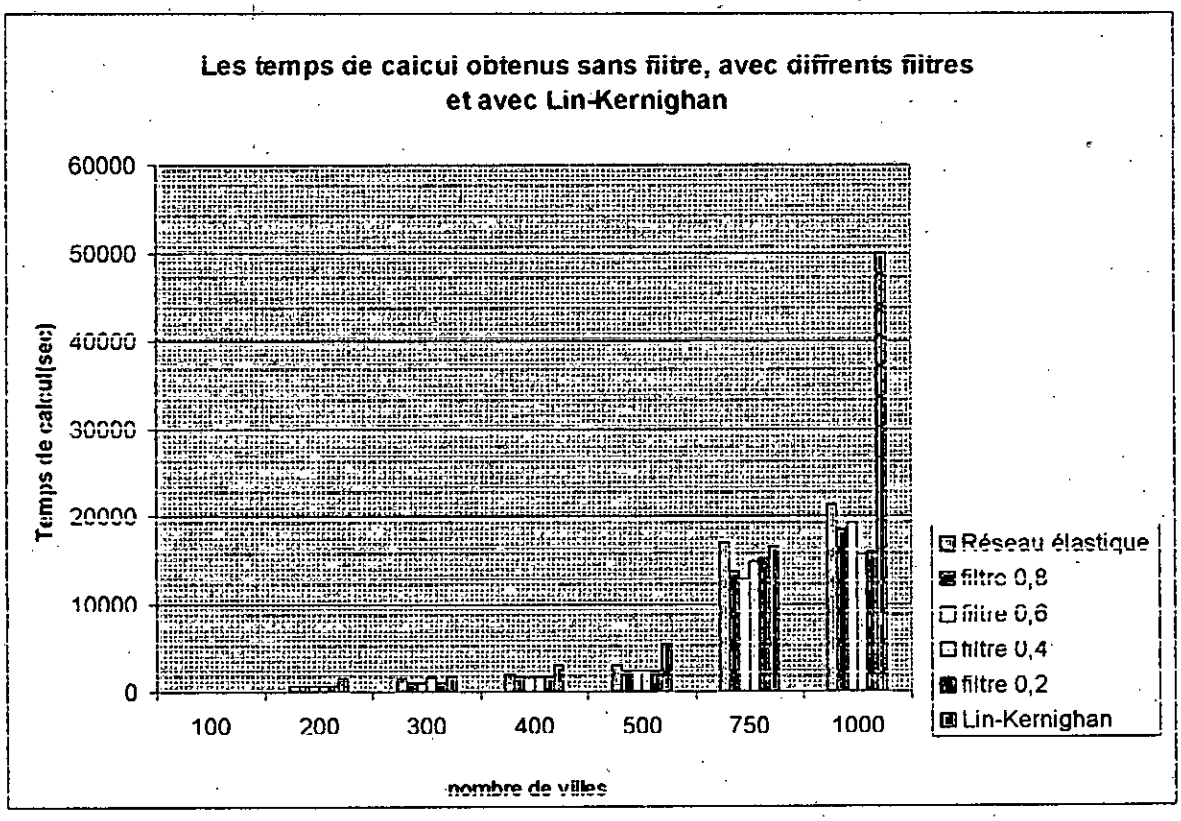


Figure 3.4. (b)

Figure 3.4. Comparaison des performances du réseau élastique classique, du γ-filtre et de l'algorithme de Lin et Kernighan. **[6]**

(a) En terme de longueur de tour (b) En terme de temps de calcul

Nous remarquons que l'algorithme de Lin et Kernighan procure des solutions de 5 à 10% meilleures que les réseaux élastiques. Parmi les réseaux élastiques, les longueurs de tour sont très proches les unes des autres, et sont même meilleurs pour le réseau avec filtre. Ce qui signifie que le filtrage n'altère pas la qualité des solutions des réseaux élastiques classiques mais contribue à l'améliorer. . Cette amélioration peut atteindre les 7% dans le cas du filtre 0.6 pour 400 villes.

Les temps de calculs sont sensiblement les mêmes pour des instances de petite taille. Dès que le nombre de villes dépasse les 400, l'écart se creuse entre les réseaux élastiques (avec ou sans filtre) et la méthode de Lin et Kernighan. Cet écart atteint 35000 secondes (soit 10 heures) pour N=1000 villes. Les réseaux élastiques sont alors 3 fois plus rapides que la procédure de Lin et Kernighan.

Au sein des réseaux élastiques, le filtrage réduit les temps de calcul. L'accélération se fait sentir à partir de 300 villes, elle peut atteindre 5000 secondes (en moyenne) pour des instances à 1000 villes soit un temps de calcul 1.3 fois inférieur à celui d'un réseau élastique sans filtre.

Conclusion, nous pouvons dire que le filtrage non seulement préserve la qualité de la solution du réseau élastique mais peut l'améliorer tout en accélérant les temps de calculs.

3.4.2. L'approche hiérarchique

Cette approche, proposée par Golden & Vakhutinsky (1996) **[30]** a été partiellement inspirée par R. Karp (1977) et T.Sun, P. Meakin, et T. Jossang[1](1993)

L'idée clé de l'algorithme est de diviser le carré unité contenant les villes en plusieurs carrés plus petits et de remplacer les villes dans chaque petit carré (cellule) par leur centre de gravité. Plusieurs itérations de l'algorithme 'réseau élastique' sont effectuées sur le problème agrégé. Le carré unité est ensuite divisé en un nombre plus grand de districts qui seront plus petits que les précédents. Les villes sont agrégées comme précédemment et plusieurs itérations sont appliquées. Et ainsi de suite, jusqu'à ce que chaque cellule ne contienne qu'une seule ville.

Description de l'algorithme

L'algorithme comprend T étapes, à chaque étape W itérations sont effectuées.

Initialement, un anneau de neurones noté : Y(o) est placé de sorte que son centre coïncide avec le centre de gravité des villes.

A l'étape k, le carré unité est divisé en un nombre $s_k=1/\sigma_k$ (supposé entier) de cellules, σ_k étant la largeur de la cellule. On dispose alors de s_k^2 cellules. Soit $U_{pq}^{(k)}$ le nombre de neurones dans la cellule p, q, tel que : $0 \leq p, q \leq s_k^2$

Soit n_k le nombre de cellules non vides.

Si $| U_{pq}^{(k)} | \geq 1$ alors définir le centre de gravité de la cellule pq :

$$x_i^{(K)} = \frac{\sum_{xi \in u_{pq}^{(K)}} xi}{\left| u_{pq}^{(K)} \right|} \quad , \quad i=1, n_K \qquad (3.9)$$

Soit $X^{(k)}$ l'ensemble contenant les centres de gravités $x_i^{(k)}$ à l'étape K.

[1] T. Sun, P. Meakin, and T. Jossang, 1993. A Fast Optimization Method Based on a Hierarchical Strategy for the Traveling Salesman Problem, *Physica A 199*, 232-242.

Si le nombre de neurones dans l'anneau précédant $|Y(k-1)| \leq m_k$, alors l'anneau de l'étape k sera formé de m_k neurones : $|Y(k)| = m_k$.

Tel que : $m_k = \gamma n_k$, l'auteur recommande de prendre γ=2.5.

W itérations sont effectuées sur l'anneau Y(k) afin d'ajuster l'anneau à la nouvelle distribution des centres de gravité.

A chaque étape k, le nombre de cellules devient plus grand jusqu'à ce qu'il y ait autant de cellules que de villes et que chaque cellule ne contienne qu'une seule ville.

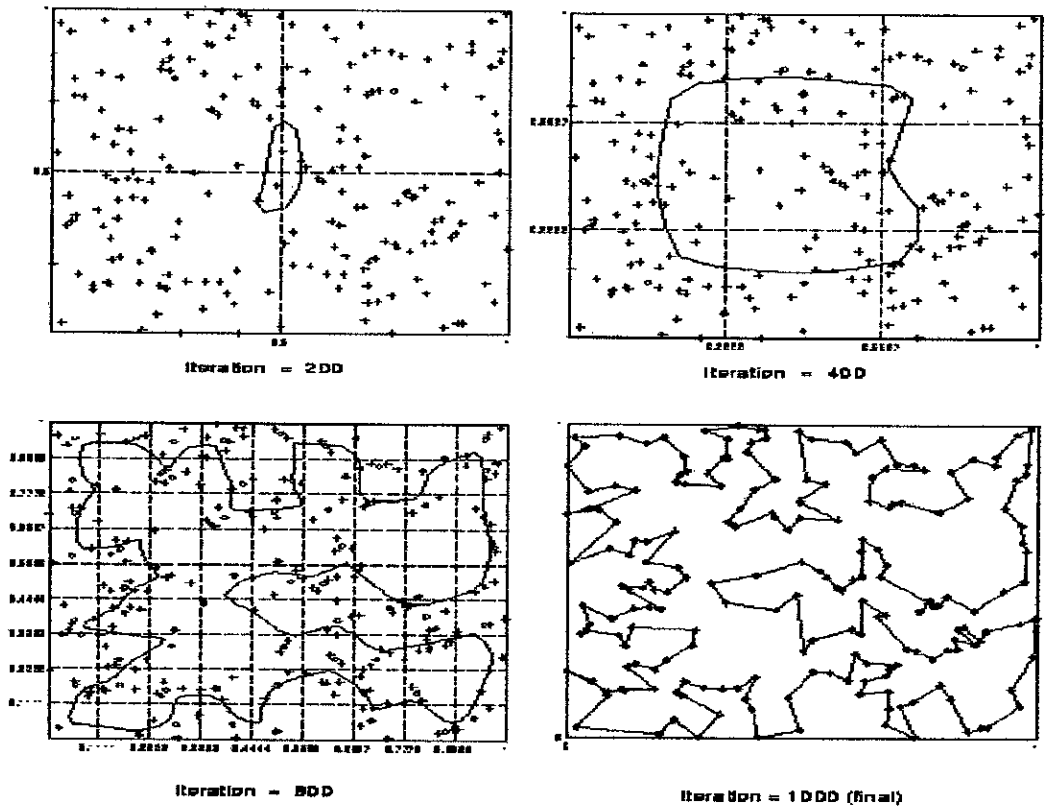


Figure 3.5.: Evolution de l'algorithme hiérarchique sur une instance à 200 villes

(noter que + = ville, o= centre de gravité d'une cellule, ●= une ville unique dans la cellule,) **[30]**

Résultats de l'implémentation

Les instances utilisées lors de la simulation ont été générées aléatoirement dans le carré unité, pour des tailles de problèmes allant de 50 à 500 villes (Les résultats sont donnés en annexe II). Les simulations ont été effectuées sur une station de travail (Work station) SUN SPARC-10. Pour chaque taille de problème,

5 instances ont été générées aléatoirement et les valeurs données en résultats sont des moyennes faites sur ces 5 instances.

Les paramètres W (nombre d'itérations par étape) et T(nombre d'étapes) ont été initialisés à 20 et 50 respectivement.

L'accélération (speed up) apportée par la méthode hiérarchique dans les temps d'exécution est :

SP1= temps (réseau élastique classique) / temps (méthode hiérarchique)

Il a été démontré **[30]** que l'expression analytique de l'accélération était la suivante:

$$S=T\frac{n^2}{\sum_{k=1}^{T} n_k^2} \qquad (3.10)$$

Où : n est le nombre de villes

n_k le nombre de cellule (de villes agrégées) à l'étape k. n_k est estimé d'après la taille des cellules σ_k, les villes étant supposées distribuées uniformément dans la surface considérée.

La figure ci-dessous représente la courbe analytique et la courbe empirique de l'accélération obtenue par la méthode hiérarchique. Nous constatons que la courbe analytique est assez proche des points expérimentaux.

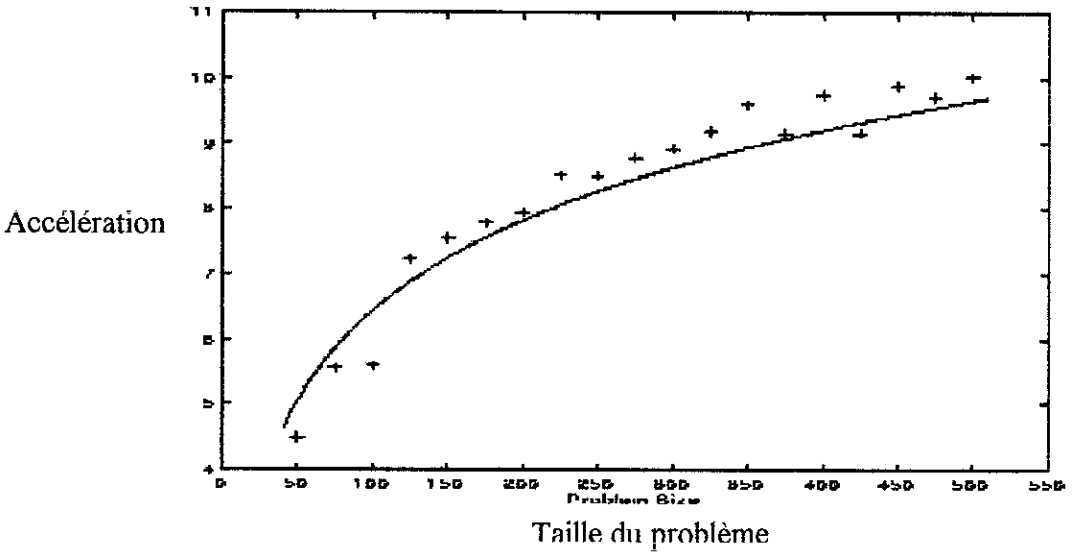


Figure 3.6. Résultats expérimentaux et analytiques de l'accélération en fonction du nombre de villes **[30]**

Accélération par filtrage **[30]**

Un filtre, analogue au γ-filtre est mis en place afin de réduire le temps de calcul. Il n'est plus nécessaire alors de calculer n*m fois $\phi(d, K)=\exp(-d^2/2K^2)$, tel que : $d^2=||x_i-y_j||^2$, pour chaque i et chaque j car l'expérience a montré que seuls les termes tels que $||x_i-y_j||^2 \geq 4K^2$ contribuaient à la somme des $\phi(||x_i-y_j||, K)$

Le filtre fonctionne comme suit : le carré unité a été divisé en $[1/2K]^2$ petits carrés de côté 2K. Avant chaque itération, l'appartenance des villes et neurones aux cellules est déterminée en un temps O(log N) et les ϕ sont calculés uniquement pour les paires x_i et y_j appartenant à la même cellule ou à des cellules adjacentes.

La méthode hiérarchique accélère les calculs du réseau élastique de façon considérable, ses temps de calcul peuvent être jusqu'à 10 fois inférieurs à ceux du réseau élastique classique (N=500) l'accélération des temps de calcul est mesurée par :

SP1=temps de calcul du réseau élastique / temps de calcul de la méthode hiérarchique.

La mise en place du filtre provoque une accélération de l'algorithme de :

SP2=temps de calcul de la méthode hiérarchique avec filtre / temps de calcul de la méthode hiérarchique.

L'accélération totale est alors SP=SP1*SP2. Cette accélération ou le gain de temps peut atteindre 16.42 pour 500 villes c'est à dire que le temps de calcul est 16.42 fois inférieur à celui d'un réseau élastique classique.

La méthode hiérarchique peut améliorer la solution de 4.5% (pour N=400), l'amélioration résultant du filtrage peut atteindre 0.2% (N= 325)

Conclusion

En plus de réduire les temps de calcul, la méthode hiérarchique avec filtre semble améliorer la qualité des solutions obtenues. Cela peut être dû au nombre de villes agrégées qui croît graduellement. L'approche hiérarchique tend alors à produire un anneau de neurones aux angles moins obtus et dont la forme est plus proche d'une courbe que d'un polygone. **[30]**

3.4.3. Le réseau élastique convexe [24]

Cette méthode (Efficient Convex Elastic Net, **ECEN**) a été proposée en 1996 par Al-Mulhem et Al-Maghrabi se situe dans la lignée des approches hybrides qui combinent recherche opérationnelle et réseaux de neurones artificiels. Ces approches hybrides ont suscité un grand intérêt ces dernières années par les améliorations qu'elles peuvent apporter aux méthodes neuronales classiques en termes de qualité de la solution.

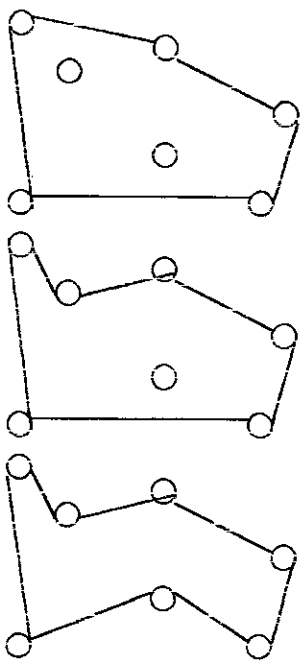
Description de l'algorithme **[24]**

L'algorithme comprend deux phases :

Un réseau élastique convexe qui génère un tour de départ à travers une recherche globale.

Une propriété importante du TSP est que l'ordre dans lequel les villes de l'enveloppe convexe sont visitées dans la solution optimale est le même que celui dans lequel elles sont visitées dans l'enveloppe convexe de départ. (Merill Flood,1956)

Il s'agit, dans la première phase de combiner cette propriété avec l'approche par réseau élastique. Pour ce faire, l'anneau de neurones initial ne se trouve plus au centre de gravité des villes mais constitue l'enveloppe convexe du graphe.



L'anneau de départ fixé sur l'enveloppe convexe des villes

Insertion des villes selon le principe de descente de gradient du réseau élastique

Figure 3.9. Evolution de l'algorithme du réseau élastique convexe

Le réseau élastique convexe met en œuvre deux types de forces :

- les forces d'attraction des autres neurones sur chaque neurone de l'anneau qui maintiennent la cohésion de l'anneau

- et les forces d'attraction des villes sur chaque neurone qui permettent de le déformer.

A chaque itération les villes non incluses dans l'anneau sont présentées au réseau à deux couches : N neurones dans la première et M (M<2N) dans la seconde et donc N*M connections ($N*M<2N^2$). Un processus de compétition se met en place (

dans la deuxième couche) où le neurone gagnant est le neurone le plus proche de la ville présentée, au sens de la distance euclidienne.

Une amélioration non déterministe itérative qui, par une recherche locale, améliore le tour obtenu à la suite de la première phase. Cette procédure fait appel à deux puissants opérateurs d'arrangement ou d'affinage de tournée tels que 2-*optimal*.

Partant du tour de départ, la procédure applique l'un des deux opérateurs afin de générer N nouveaux tours. Ensuit l'un des N nouveaux tours est sélectionné et fait l'objet des améliorations sus citées en tant que tour de départ. Ces opérations ont pour but d'ôter les boucles et les arêtes qui se croisent du tour.

Résultats des simulations **[24]**

Les simulations ont été faites sur des problèmes test classiques de la littérature de recherche opérationnelle.

Problème	Nombre de villes	Longueur optimale	Longueur du tour ECEN	% d'optimalité
GRID 100	100	100	100.8	0.8 %
KAR 100	100	21282	21622.9	1.6 %
LIN 318	318	44169	46231.7	4.7 %
GR 96	96	55209	57634.1	4.4 %
GR 137	137	69853	72150.4	3.2 %

Tableau 3.1 : résultats des simulations sur des problèmes test classiques

Nombre de villes	10 villes	30 villes	50 villes	100 villes
Bornes de Stein	2.42-3.68	4.19-4.92	5.41-5.98	7.65-8.05
ECEN	**2.47**	**4.32**	**5.57**	**8.04**
Space Filling Curve	3.14	4.45	7.03	9.56
Hopfield et Tank	2.92	5.12	6.64	----
Recuit simulé	2.74	4.75	6.13	8.40
Réseau élastique	2.45	4.51	5.58	8.23

Tableau 3.2 : comparaison du ECEN avec d'autres algorithmes sur des instances générées aléatoirement

Les simulations ont été faites sur des instances dont les coordonnées des villes ont été générées aléatoirement dans la carré unité. L'intervalle de la solution optimale a été calculé par la formule de Stein (1977) :

$$0.765\sqrt{N} \leq \text{longueur du tour optimal} \leq (0.765+\frac{4}{N})\sqrt{N}$$

Les résultats de simulation ont montré que le pourcentage d'optimalité de ECEN n'excédait pas 5% et que ses résultats étaient meilleurs que les cinq autres algorithmes témoins (excepté le réseau élastique classique sur 10 villes).

3.5. Synthèse

Nous avons abordé dans ce chapitre trois méthodes constituant des améliorations récentes du réseau élastique pour le TSP.
La première **[6]**, s'intéresse au temps de calcul par la mise en place d'un filtre. Ce filtre est destiné à pallier le handicap majeur des réseaux élastiques qu'est son temps de calcul.

La seconde **[30]**, reprend l'idée du filtre et met en oeuvre une hiérarchisation de la méthode classique. Cette hiérarchisation, non seulement, améliore la qualité des solutions obtenues mais accélère les temps de calcul. Cette accélération à été modélisé et identifiée par une formule analytique **[30]**.
La troisième méthode abordée **[24]**, est une approche hybride combinant des opérateurs d'affinage de tournée avec le réseau élastique convexe. Ces opérateurs d'affinage sont destinés à améliorer la qualité de la solution obtenue par le réseau élastique.

Les réseaux élastiques abordés dans cette partie tirent leur origine des cartes auto-organisées de Kohonen. Ces dernières ont été appliquées au TSP en 1988 avec succès. De puis, cette deuxième classe de méthodes fait, elle aussi, l'objet d'améliorations.

4.1. Introduction

Nous aborderons dans cette partie le phénomène de l'auto-organisation qui tire son origine dans les cartes auto-organisatrices présentes dans le cerveau. Nous commencerons par décrire la théorie des cartes auto-organisatrices de Kohonen, nous pourrons ensuite introduire l'approche de B.Angéniol et al. Nous développerons enfin deux méthodes qui s'inscrivent dans cette lignée et qui constituent des améliorations de la première, à savoir le réseau Flexmap - développé par Fritzke et Wilke- et la compétition harmonique-proposée par Matsuyama.

4.2. Les cartes auto-organisatrices de Kohonen [17]

Les cartes auto-organisatrices (Self Organizing Maps) ont été introduites en 1982 par T.Kohonen. Ce sont des réseaux dans lesquels les neurones sont liés entre eux sur une surface élastique. Le comportement des neurones varie graduellement d'une région de la carte à l'autre. Ce type de structuration est fréquemment observé dans le cerveau où on retrouve souvent des cartes topologiques.

L'intérêt majeur des cartes topologiques est de représenter sur un petit nombre de dimensions la structure présente dans des données de haute dimensionnalité. C'est donc une méthode d'analyse de données performante. Cette théorie et plusieurs autres applications constituent un des principaux champs de recherche dans les réseaux de neurones d'aujourd'hui. Les domaines des applications sont : l'analyse des données, la reconnaissance de formes, l'analyse de la parole, la robotique, les diagnostics médicaux et industriels et l'optimisation combinatoire.

Architecture

Les cartes auto-organisatrices sont constituées d'une seule couche - compétitive- où les neurones sont disposés géométriquement selon une topologie bien définie dans un réseau à n dimensions. Ces neurones forment une grille et sont tous connectés aux entrées. Cette grille peut être à motif hexagonal, carré ou irrégulier.

La structuration topologique de la carte provient d'une mesure de distance d définie sur les neurones du réseau. Cette distance peut être la distance euclidienne. On définit ainsi un voisinage du neurone.

Une carte auto-organisatrice peut être linéaire, bidimensionnelle, tridimensionnelle ou n-dimensionnelle.

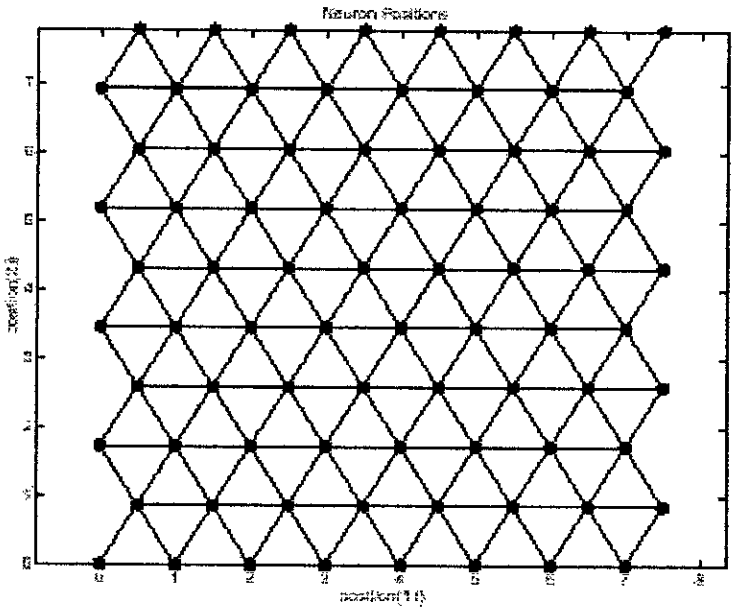


Figure 4.1. une carte auto-organisée à topologie hexagonale. **[mlab]**

Les neurones sont identifiés par leurs vecteurs de poids: $w_i=(w_i^{(1)}, w_i^{(2)}, \ldots, w_i^{(n)})$ ou coordonnées dans l'espace à n dimensions. Chaque neurone est relié aux n composant du vecteur d'entrée.

La figure 4.1 schématise un réseau de Kohonen bidimensionnel, la couche compétitive est connectée à un vecteur d'entrée à deux composantes. Ce vecteur d'entrée prendra différentes valeurs au cours de l'apprentissage.

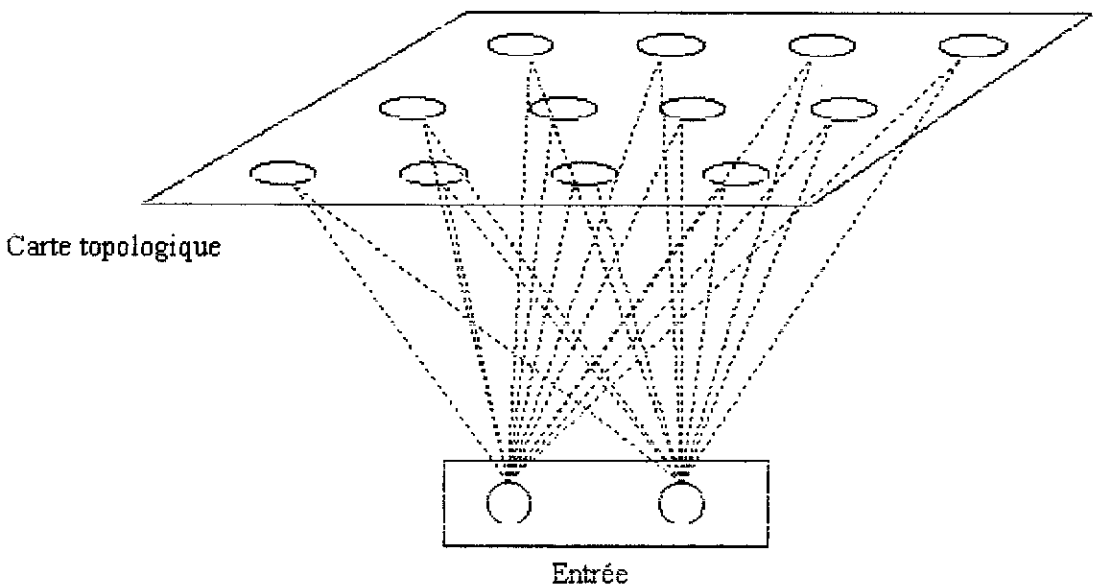


Figure 4.2. Schéma d'une carte auto-organisatrice de Kohonen bidimensionnelle

Apprentissage dans le plan euclidien **[1]**

Nous allons nous Limiter au cas d'une carte bidimensionnelle munie de la métrique euclidienne usuelle. C'est cette configuration du réseau qui s'applique aux problèmes d'optimisation.

Le réseau de Kohonen est un réseau compétitif, il satisfait donc les deux conditions suivantes :

- Les neurones de la couche compétitive doivent réagir différemment aux entrées afin de pouvoir élire un neurone vainqueur. Cette condition peut être satisfaite en choisissant une distribution aléatoire des poids dans le plan.
- Il existe un mécanisme de concurrence entre les neurones où le neurone le plus proche des données d'entrée serait le vainqueur, par exemple.

L'apprentissage dans le réseau de Kohonen peut être supervisé ou non supervisé. Dans la présente étude, nous nous intéresserons à l'apprentissage non supervisé. Dans ce type d'apprentissage, le réseau apprend 'tout seul' sans qu'il soit nécessaire de lui montrer ce qu'il doit apprendre ou retenir. Le réseau identifie lui-même les corrélations entre les données d'entrée et les représentent par des groupes de neurones qui vont se mouvoir pour se rapprocher de ces données. Les neurones, en modifiant leurs poids et donc leur positions dans l'espace, élaborent une représentation des données qui préservent la topologie de ces données.

L'apprentissage ou adaptation aux données d'entrée se fait comme suit : A un instant t, un vecteur de données $x=(x_1, x_2)$ est présenté aux neurones de la carte. Le réseau entre alors dans une **phase compétitive** où le neurone dont le vecteur de poids est le plus proche de x au sens de la métrique choisie est désigné comme le gagnant. Le neurone gagnant n_g est donc celui qui minimise $\|x-w_i\|$, i étant l'indice du neurone.

Vient ensuite la **phase de coopération**. Le neurone gagnant et ses quelques voisins sont alors mis à jour, c'est à dire que leurs vecteurs de poids subiront la transformation suivante :

$$w_i(t+1)=w_i(t)+\varepsilon(t)h_{gi}(t)\left[x(t)-w_i(t)\right]$$

Les poids du neurone sont incrémentés d'une quantité proportionnelle à leur éloignement du vecteur d'entrée : $x(t)\ w_i(t)$. Le facteur de proportionnalité est composé de deux termes :

- La fonction *h* a un rôle central dans le processus d'adaptation des neurones aux données d'entrée. Il s'agit d'une fonction noyau (telle la courbe de

Gauss) qui va déterminer dans quelles proportions les voisins du neurone gagnant seront déplacés.

$$h(t)=\exp(-\frac{\|W_g-W_i\|}{2\sigma(t)^2})$$

- Le paramètre ε est appelé taux d'apprentissage et σ définit la largeur de la fonction noyau. Les paramètres ε et σ sont des fonctions monotones décroissantes. En effet, car pour les besoins de convergence de l'algorithme le taux d'apprentissage et la largeur du voisinage doivent décroître dans le temps. Plus on avance dans l'apprentissage (adaptation aux données) et moins il est nécessaire d'apprendre.

La fonction exponentielle délimite le périmètre concerné par la mise à jour. En effet, le neurone gagnant n'est pas le seul à être déplacé, les neurones se trouvant dans un certain voisinage sont eux aussi déplacés d'une quantité qui dépend de leur éloignement du neurone gagnant. Le voisinage peut être délimité autrement que par une fonction noyau : les neurones constituant les neurones d'une grille, il suffit de définir un voisinage géométrique. Dans ce cas, $h_{gi}(t)$ =1 si le neurone i $\in V_g(t)$, le voisinage du neurone gagnant à l'instant t.

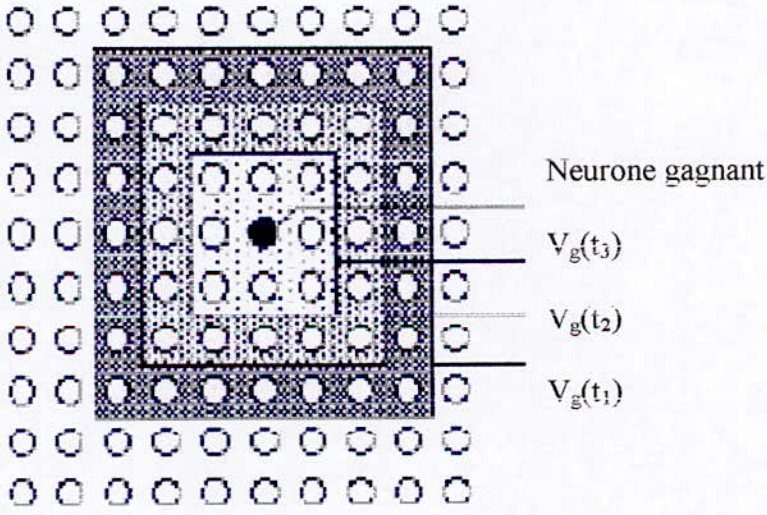


Figure 4.3. Définition des voisinages sur une carte bidimensionnelle sur 3 étapes **[23]**

La figure 4.4. représente une carte au cours du processus d'auto-organisation pendant lequel elle s'adapte aux données d'entrée. Cette adaptation permet à chaque neurone de se spécialiser dans une certaine catégorie de données. A chaque fois qu'un vecteur d'entrée appartenant à la catégorie de données qui concerne le neurone i sera présenté au réseau, celui-ci réagira plus fort que les autres neurones et indiquera alors que le vecteur d'entrée appartient à sa catégorie.

En s'adaptant aux données d'entrée, la carte auto-organisée approxime la densité de distribution des données qui se trouve dans ce cas uniformément distribuée.

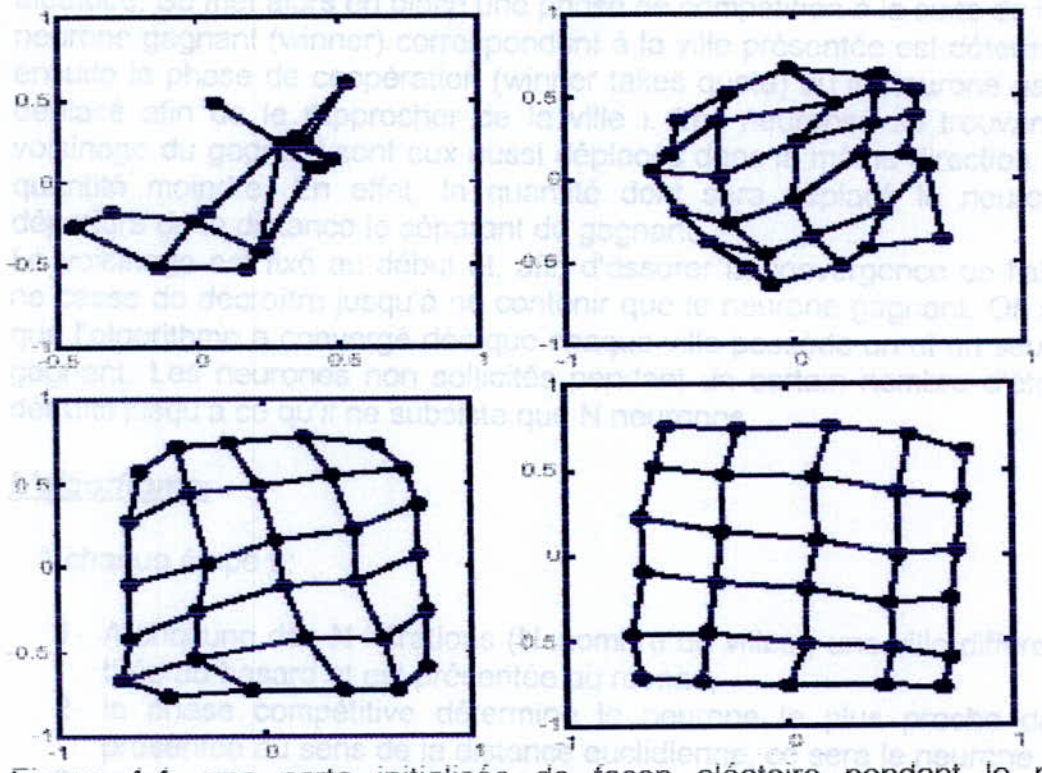


Figure 4.4. une carte initialisée de façon aléatoire pendant le processus d'adaptation. **[23]**

Les cartes topologiques constituent une méthode performante d'analyse de données dans la mesure où elles permettent de représenter sur un petit nombre de dimensions une structure présente dans des données de haute dimensionnalité.
Les cartes de Kohonen peuvent être utilisées dans le cadre de la projection de données, d'approximation de densité ou de classification. Elles ont été utilisées avec succès en reconnaissance de la parole, traitement d'images, robotique, contrôle de processus.

4.3. Application au TSP [29]

En 1988 Bernard Angéniol, Le Texier et de Lacroix Vaubois font appel aux cartes auto-organisatrices de Kohonen pour la résolution du TSP. La carte utilisée est unidimensionnelle, il s'agit d'un anneau de neurones similaire à celui des réseaux élastiques. En entrée, les neurones recevront les coordonnées des villes et s'adapteront à la topologie des villes afin de former en fin d'apprentissage (ou d'entraînement) un circuit proche de l'optimum.
Bien que la topologie soit la même, la règle de mise à jour des neurones diffère de celle des réseaux élastiques.

Certaines variantes de l'algorithme, prévoient la possibilité de dupliquer les neurones gagnants afin que chaque neurone gagnant le soit pour une seule ville uniquement. Le nouveau neurone est inséré entre son jumeau et l'un de ses voisins. Souvent cette création de neurones est accompagnée par la destruction des neurones qui n'ont été déclarés gagnants à aucun moment et pour aucune ville.

La complexité de cet algorithme est $O(N^2)$, dans la mesure où : à chaque étape, les N villes sont présentées au réseau et que pour chaque ville, il faut parcourir l'ensemble des M neurones afin d'en déterminer le gagnant. M étant en général proportionnel à N, du moins pendant un certain temps, le nombre d'opérations est proportionnel à N^2. Nous devons faire remarquer ici, que la constante de proportionnalité comprend le nombre d'étapes à effectuer avant que l'algorithme converge. Ce nombre est en général supérieur à N.

4.4. Améliorations récentes

4.4.1. FEXMAP [12]

Introduit par Fritzk et Wilke (1991), il s'agit d'un réseau de neurones auto-organisé inspiré des cartes auto-organisatrices de Kohonen. Cependant, l'anneau de neurones à une taille variable et s'agrandit au fur et à mesure que l'algorithme avance dans l'exécution.

Cet algorithme, à la différence de celui de Kohonen, utilise une fonction de voisinage constante, il s'agit de déplacer uniquement le neurone gagnant et ses deux voisins (situés de part et d'autre). Le paramètre d'adaptation ε dans le réseau de Kohonen décroît au fur et à mesure que l'algorithme s'exécute alors que FLEXMAP a recours à un taux d'apprentissage constant. La constance de ce paramètre garantit la convergence vers un état stable tel qu'aucun nouveau mouvement de neurones n'est possible, et la satisfaction des deux conditions :

1- les neurones sont proches des villes
2- les voisins de ces neurones le sont aussi

Les auteurs ont défini une variable erreur propre à chaque neurone. Pendant la phase compétitive/coopérative (recherche du neurone gagnant 'vg' et mise à jour de ses coordonnées et de celles de ses voisins) la variable erreur du gagnant est incrémentée par sa distance à la ville C choisie :

$$er(vg) = er(vg) + dist(C, vg)$$

Une valeur élevée de l'erreur pour un neurone donné indique que celui-ci se trouve dans une région où le ratio neurones/villes est bas. Il faut donc insérer des neurones dans cette région étant donné qu'à chaque ville doit correspondre un neurone.

L'algorithme initial

Initialisation : démarrer avec un anneau de trois neurones aléatoirement disposés. Initialiser les variables erreur à 0.

Etape 1 : effectuer un nombre constant de phases compétitives n_c et mettre à jour la variable erreur pour chaque neurone gagnant.

Etape 2 : trouver l'arête t de l'anneau reliant deux neurones v1 et v2 qui maximise l'erreur :

$$er(t)=er(v1)+er(v2)$$

Insérer un nouveau neurone v au milieu de l'arête t et initialiser la variable erreur du nouveau neurone à :

$$er(v)=\frac{[er(v1)+er(v2)]}{3}$$

Diminuer les erreurs de v1 et v2 de sorte que : $er(v)+er(v1)+er(v2)$ reste constant : $er(v1)=2\frac{er(v1)}{3}$ et $er(v2)=2\frac{er(v2)}{3}$

Etape 3 : si chaque ville possède un plus proche neurone différent de ceux des autres villes, alors la solution a été trouvée sinon aller à l'étape 1■

Chaque arête doit être visitée afin de déterminer celle dont l'erreur est maximale. Le nombre d'arêtes étant égal au nombre de neurones il faut O(N) itérations. A l'étape 3, pour chaque ville l'algorithme doit déterminer le neurone le plus proche. Il faut donc O(N) itérations.

Une boucle (étapes 1 à 3) devant être répétée N fois, la complexité de l'algorithme est d'au moins $O(N^2)$

Comment rendre l'algorithme linéaire

En remplacer la recherche globale du neurone (neurone) gagnant, à l'étape 1,par une recherche locale :

Lorsqu'une ville est présentée au réseau une seconde fois, la recherche du gagnant se limitera au neurone gagnant précédant et aux neurones se trouvant dans un voisinage de taille K. Ce K-voisinage étant constant, la recherche prend O(1) itérations (c'est à dire qu'elle n'est pas proportionnelle au nombre de neurones et donc pas au nombre de villes).

A l'étape 2, au lieu de faire une recherche globale sur toutes les arêtes afin de déterminer celle dont l'erreur est maximale, nous considérerons le fait, qu'en général, les arêtes à grande erreur sont adjacentes aux neurones gagnants.

La recherche de l'étape 2 se limitera aux arêtes adjacentes à un neurone gagnant de l'une des n_c phases compétitives de l'étape1. Vu que n_c est constant cette recherche est faite en un temps constant. Cette procédure, si elle n'aboutit pas à l'arête d'erreur maximale, trouve une arête dont l'erreur est importante.

Le test de l'étape 3 peut être remplacé par la procédure suivante :

Lorsqu'un neurone est gagnant pour une ville pendant un certain nombre d'itérations, on 'attache' ce neurone à cette ville. C'est à dire que ce neurone est placé à la position exacte de la ville et ne fait plus l'objet de modifications. La ville est alors retirée de l'ensemble des 'villes libres'. Dès que l'ensemble des villes libres est vide, une solution est atteinte.

Maintenant, chaque boucle est O(1) l'algorithme est alors O(N).

Affinage de la tournée en temps linéaire **[12]**

Pour chaque séquence de k villes, faire les k ! améliorations possibles. k sera pris égal à 4 ce qui produit une amélioration de la solution de 0 à 2%. k étant fixé, le nombre d'opérations pour chaque séquence est O(1), le nombre total d'opérations est alors O(N).

Simulations

L'algorithme Flexmap désigne la succession des procédures linéaires décrites ci-dessus. Flexmap a été implémenté sur des problèmes-test classiques dont la solution optimale est connue. Puis des instances aléatoires de tailles : 700, 1200, 1600 et 2000 ont été générées. Les solutions et temps de calcul sur ces dernières instances ont été comparées à ceux de la procédure 2-Opt qui est $O(N^2)$. 2-Opt étant une procédure d'affinage, elle requière un tour initial. La procédure Nearest Neighbor (NN) a donc été utilisée. **[12]**

Paramètres de l'algorithme **:[12]**

Le nombre de répétitions de la phase compétitive n_c=100.

Le taux d'apprentissage du neurone gagnant $\varepsilon(n_g)$=0.05,

Le taux d'apprentissage ses voisins $\varepsilon(v)$=0.05,

La taille du K-voisinage K=50.

L'attachement est effectué lorsque le neurone est gagnant 100 fois pour une même ville.

Notons que ces paramètres sont fixes et ne doivent nullement être adaptés à la taille du problème.

Résultats des simulations

L'implémentation a été faite sur une station de travail SunAPARCstation2. Les solutions présentées par l'auteur sont les meilleurs obtenues pour chaque problème sur plusieurs exécutions. Les temps d'exécution, quant à eux, sont des temps moyens sur plusieurs simulations. Les exécutions ont été répétées 20 fois pour chaque instance. Le nombre de neurones nécessaires à l'exécution de l'algorithme était de l'ordre de 2.5*N. **[12]**

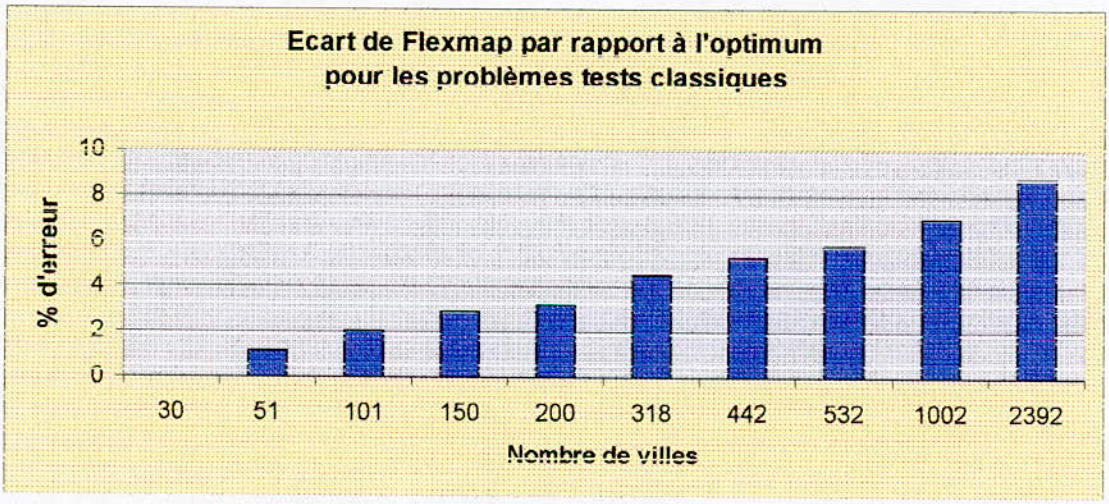


Figure 3.5. Performances de Flexmap sur des problèmes test classiques

Nous constatons que Flexmap n'excède jamais les 9% d'écart par rapport à l'optimum, y compris pour des instances à 2392 villes. Avec des écarts inférieurs à 2% pour l'instance à 51 villes.

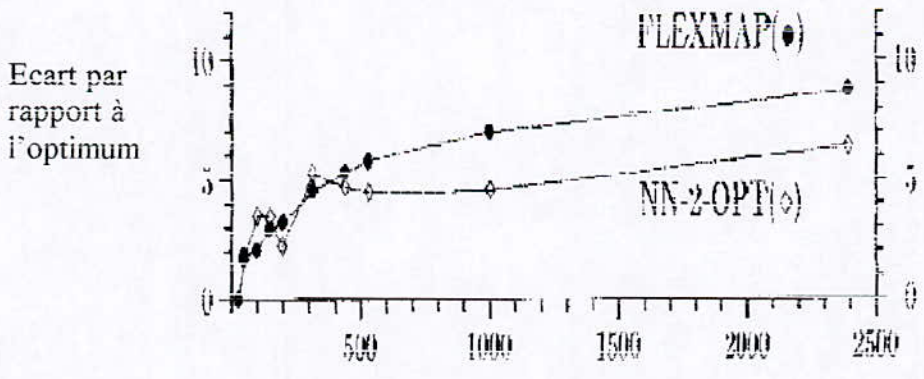


Figure 3.6. (a)

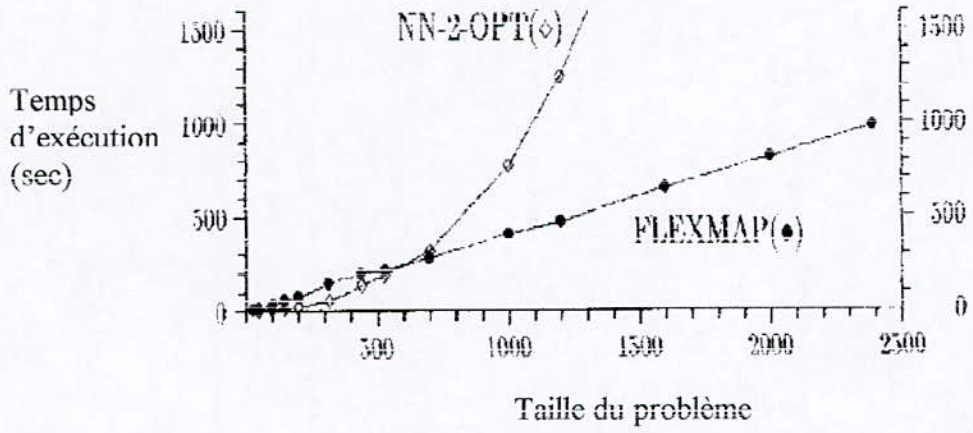


Figure 3.6. (b)

Figure 3.6. Comparaison des performances de Flexmap et de la procédure 2-Opt

(a) En terme de longueur de tour (b) En terme de temps de calcul **[12]**

Concernant la qualité de la solution, les deux algorithmes restent équivalent pour des problèmes de taille inférieure à 500. A partir de 500 villes l'heuristique conventionnelle se démarque pour converger vers un éloignement de 5% de l'optimalité alors que Flexmap tend vers les 9% négativement pour des problèmes de taille supérieure à 1000. Par exemple, pour une instance à 2392 villes, l'heuristique Nearest Neighbor+2-Opt prend 5037 secondes alors que Flexmap ne prend que 980 secondes soit une exécution 5 fois plus rapide que Nearest Neighbor+2-Opt.

Pour les problèmes de petite taille, l'heuristique conventionnelle est plus rapide que Flexmap, les deux algorithmes se rejoignent pour des instances à 700 villes puis Flexmap prend le dessus en termes de rapidité pour devenir deux fois et demi plus rapide pour des instances de taille 1000. A partir de là l'heuristique conventionnelle diverge.

4.4.2. Compétition harmonique

En 1990, Y.Matsuyama **[21]**, tout en utilisant une carte auto-organisée de Kohonen, introduit un second terme dans la fonction d'énergie à minimiser, il s'agit du terme de cohésion que l'on retrouve dans les réseaux élastiques :

$$\sum_{j=1}^{M} \left\| y_j - y_{j+1} \right\|^2$$

La fonction d'énergie à minimiser est alors :

$$E = \bar{f} + \lambda \bar{g} = \frac{1}{N} \sum_{j=1}^{N} \left\| x_i - y_j \right\|^2 + \lambda \sum_{j=1}^{M} \left\| y_j - y_{j+1} \right\|^2$$

Où : $\bar{f}=\frac{1}{N}\sum_{j=1}^{N}\left\|x_i-y_j\right\|^2$ est le coût moyen représentant l'approximation que fait le réseau des données d'entrée. En d'autres termes, $\bar{f}$ est la moyenne des distances séparant les neurones gagnants des villes auxquelles ils doivent la victoire. Cette moyenne est à minimiser dans la mesure où les neurones sont sensés se rapprocher des villes.

$\bar{g}=\sum_{j=1}^{M}\left\|y_j-y_{j+1}\right\|^2$ est le terme de contrainte qui fait en sorte que l'anneau de neurones soit le plus court possible.

A chaque étape, les villes sont successivement présentées au réseau dans un ordre aléatoire. Lorsque la ville xi est présentée, le neurone gagnant est déterminé, il est alors déplacé ainsi que ses voisins. Le voisinage à partir d'une valeur initiale est décroissant jusqu'à ne comprendre que le neurone gagnant en fin d'exécution.

Les neurones sont disposés en anneau, le nombre M est entre 3N et 4N. durant la phase compétitive, l'identification du gagnant se fait à l'aide du carré de la distance euclidienne est utilisé, ce qui- du fait de son caractère monotone- donne le même gagnant que la distance euclidienne.

Le déplacement subit par le neurone j appartenant au voisinage du gagnant sous l'effet de la ville i présentée au réseau est :

$$\Delta y_j=\alpha(t)(x_i-y_j)+\alpha(t)(y_{j+1}-2y_j+y_{j-1})$$

où : ε(t) est le taux d'apprentissage à l'étape t

$\alpha(t)$ est le facteur d'inhibition à l'étape t, $\alpha(t)=\varepsilon(t)\,\lambda(t)$

ε(t) et $\alpha(t)$ sont des nombres positifs et 'petits',

ε(t) est choisi de sorte à croître lentement au fur et à mesure des itérations, tout en étant autorisé à décroître lorsque cela est nécessaire.

$\alpha(t)$ sert à pondérer le deuxième terme du second membre, que l'on retrouve dans les réseaux élastiques et, qui agit pour raccourcir le périmètre de l'anneau. La présence de ce terme donne à l'anneau une forme plus lisse en maintenant la cohésion entre les neurones.

En 1996, **[22]** Matsuyama propose un contrôle dynamique de $\alpha=\varepsilon\,\lambda$

La fonction coût à minimiser étant composée de deux termes :

$$E=\bar{f}+\lambda\bar{g}=\frac{1}{N}\sum_{j=1}^{N}\left\|x_i-y_j\right\|^2+\lambda\sum_{j=1}^{M}\left\|y_j-y_{j+1}\right\|^2$$

L'idée principale de l'algorithme est d'opérer un contrôle dynamique du paramètre λ qui pondère le terme de cohésion ainsi que le taux d'apprentissage ε.

Nous décrivons ici deux des trois façons de contrôler λ parmi les trois proposées par l'auteur :

$$\lambda(t)=\frac{a\,i\,\bar{f}(t)}{\bar{g}(t)} \qquad \text{et} \qquad \lambda(t)=\frac{a\,i\,\varepsilon(t)\bar{f}(t)}{g(t)}$$

Où : a est une constante à définir et t représente le numéro de l'étape.

A chaque étape t : ε est ajusté de la façon suivante : $\varepsilon(t)=Max\{\varepsilon(t)+\Delta\varepsilon(t),\varepsilon_{\max}\}$

Où : $\Delta\varepsilon(t)=0$ si $\bar{f}(t)> \bar{f}(t-1)$ et $\bar{g}(t)< \bar{g}(t-1)$,

$\Delta\varepsilon(t)=\gamma_\varepsilon\frac{(\bar{f}(t)-\bar{f}(t-1))}{\bar{f}(t-1)}$ sinon

avec γ_ε =0.5 si ε(t+1)< ε(t),

γ_ε =0.25 sinon.

L'algorithme **[22]**

Initialisation

1) Initialiser t à 0.
2) Initialiser les coordonnées des neurones sur un anneau
3) Choisir une règle d'ajustement de λ
4) Choisir une fonction de voisinage f (une fonction noyau)
5) Déterminer l'ampleur du voisinage initial et sa formule de décroissance
6) Déterminer un taux de vigilance r0

Présentation des villes au hasard : à chaque étape t, les N villes sont présentées au hasard,

1) Phase de compétition : détermination du neurone gagnant y_j
2) Phase de coopération : les neurones appartenant au voisinage 2.σ sont déplacés comme suit :

$$y_j(t+1)=y_j(t)+\varepsilon(t).f(d,t).(x_i-y_j(t))+\alpha(t).(y_{j+1}(t)-2y_j(t)+y_{j-1}(t))$$

Où d est la distance entre le neurone concerné par le déplacement et le neurone gagnant, $f(d,t)$ est la fonction de voisinage, c'est une fonction noyau.

Test d'arrêt : si chaque ville possède son propre neurone gagnant alors l'algorithme a convergé, Sinon si un neurone possède un pourcentage d'appréhension (catch percentage) r_j>r0 cela signifie qu'il est sollicité par plus d'une ville. Il faut alors dédoubler ce neurone et placer le nouveau neurone à la même position que lui. Ce test est effectué à chaque étape.

Mettre à jour t=t+1, ε(t), α(t) et σ(t).□

Le pourcentage d'appréhension r est défini pour chaque neurone j comme suit :

$$r_j = \frac{f_j}{\overline{f}} \quad \text{où :} \quad f_j = \frac{1}{N} \sum_{j\,gagnant\,de\;i} \left\| y_j - x_i \right\|$$

La variable r est analogue à l'erreur définie dans Flexmap. Ce pendant, r_j est défini sur un neurone j et non sur une arrête, de plus r_j est un ratio. Son numérateur représente la somme des distances d'un neurone gagnant aux villes pour lesquelles il est gagnant. Quant à son dénominateur, il représente la somme des termes que nous venons de définir sur tous les neurones gagnants du réseau.

Résultats des simulations **[22]**

Une seule simulation a été effectuée sur l'instance 'usa 532', la taille de cette instance dont la solution optimale est connue est de 532, elle représente 532 localités des USA.

Les résultas sont consignés dans le tableau 4.1. qui suit.

Méthode	Ecart de l'optimum	Temps de calcul
Compétition Harmonique $\lambda(t)=\frac{at\,\overline{f}(t)}{g(t)}$	2.49%	4.1h
Compétition Harmonique $\lambda(t)=\frac{at\varepsilon(t)\overline{f}(t)}{g(t)}$	3.52%	4.1h
Contrôle statique de λ	3.21%	2.8h
Recuit simulé	4.44%	0.33h
Réseau élastique classique	34.23%	170h
Branch & Cut	8.75%	6h

Tableau 4.1. Résultats obtenus sur l'instance usa532 avec différentes méthodes.

Les calculs concernant la compétition harmonique et le contrôle statique de λ ainsi que le recuit simulé ont été effectués sur une station de travail de 12.5 MIPS (Millions d'instructions par seconde). Le Branch & Cut a été implémenté sur un Cyber 205 Supercomputer. Quant au réseau élastique, aucune précision n'a été donnée par l'auteur concernant la machine utilisée.

Le recuit simulé a nécessité plusieurs essais avant d'atteindre le meilleur rythme de refroidissement.

4.5. Synthèse

Nous avons présenté dans cette partie deux approches relevant des cartes auto-organisatrices de Kohonen. Ces deux approches, tout en reprenant le principe du réseau auto-organisé, innovent par leurs apports.
La première**[12]**, tente de réduire son temps de calcul et son espace de stockage des données. Pour ce faire les auteurs ont considéré un voisinage constant et un anneau de neurone dont la taille évolue en cours d'adaptation. Ils ont aussi défini une erreur sur chaque neurone, cette erreur permet de déterminer où ajouter des neurones.

La seconde méthode**[22]**, prévoit aussi d'ajouter des neurones en cours d'adaptation suivant un principe similaire à celui de la première. Mais l'apport le plus intéressant est celui qui consiste à introduire le terme de pénalité des réseaux élastiques. De plus, l'auteur a proposé un mécanisme d'adaptation dynamique du paramètre pondérant ce terme ainsi que du taux d'apprentissage. Les résultats annoncés semblent prometteurs.

5.1. Introduction

Nous nous proposons dans cette partie de faire le point sur les différentes méthodes présentées. Notre discussion s'articulera, pour chaque méthode présentée autour de trois points : le temps de calcul, la qualité des solutions et la facilité d'implémentation. Ce dernier point concernera essentiellement la détermination des paramètres des algorithmes présentés car ceux-ci semblent jouer un rôle déterminant dans la convergence des méthodes neuronales géométriques.

Nous commencerons par analyser les performances des réseaux élastiques sur des instances générées aléatoirement en ramenant les résultats bruts à la borne inférieure de la solution optimale donnée par Held et Karp (cf. chapitre 1). Nous commenterons plus en détails les résultats obtenus pour les différents réseaux élastiques. Nous ferons également appel à la littérature pour réinterpréter les résultats donnés par Flexmap en matière de linéarité de l'algorithme. Enfin, nous aurons recours à l'implémentation afin de vérifier les performances annoncées de la compétition harmonique. Nous spécifierons les paramètres utilisés lors de l'implémentation ainsi que les apports destinés à accélérer la convergence. Les résultats de l'implémentation seront présentés, analysés et comparés aux autres méthodes neuronales dans la mesure du possible en terme de qualité des solutions et de temps de calcul.

5.2. Les réseaux élastiques

Nous citons ici les problèmes qui peuvent être rencontrés lors de l'implémentation des réseaux élastiques. Ces problèmes ont été soulevés dans **[31]** après une étude détaillée de l'évolution de la fonction d'énergie lors du déroulement de l'algorithme.

Les réseaux élastiques peuvent produirent des solutions irréalisables et ceci pour deus raisons :

- Le facteur température K peut être abaissé trop rapidement. Tout comme pour le recuit simulé, le rythme de refroidissement doit être assez rapide afin d'assurer la convergence de l'algorithme en un temps raisonnable et assez lente pour donner le temps à l'algorithme d'aboutir à une solution réalisable.
- Deux villes très proches peuvent se voir assigner un même neurone après convergence de l'algorithme.

Ces deux inconvénients sont propres à tous les réseaux élastiques et les améliorations que nous allons aborder peuvent être sujettes à ce type de problèmes.

5.2.1. Le réseau élastique avec *γ-filtre*

1-Temps de calcul

Au sein des réseaux élastiques, le filtrage réduit les temps de calcul. L'accélération se fait sentir à partir de 300 villes, elle peut atteindre 5000 secondes (en moyenne) pour des instances à 1000 villes soit un temps de calcul 1.3 fois inférieur à celui d'un réseau élastique sans filtre.

2-Qualité des solutions

Les résultats obtenu sur des instances générées aléatoirement, ont été présentés de façon brute et n'ont pas été comparés à la borne inférieure de la solution optimale (cf. chapitre3). Nous avons repris les meilleurs résultats (obtenus grâce à différents filtres)pour chaque taille de problème et les avons comparés à la borne de Held et Karp.

N>100, $\text{Borne}_{HK}(N) \sim \sqrt{N}(0.70805+\frac{0.52229}{\sqrt{N}}+\frac{1.31572}{N}-\frac{3.07474}{N\sqrt{N}})$

La borne de Held et Karp a été choisie pour sa bonne approximation de la solution optimale.

Dans le graphique suivant, nous présentons les erreurs relatives (en %) des meilleures solutions du *γ-filtre* **[6]** par rapport à la borne_{HK}.

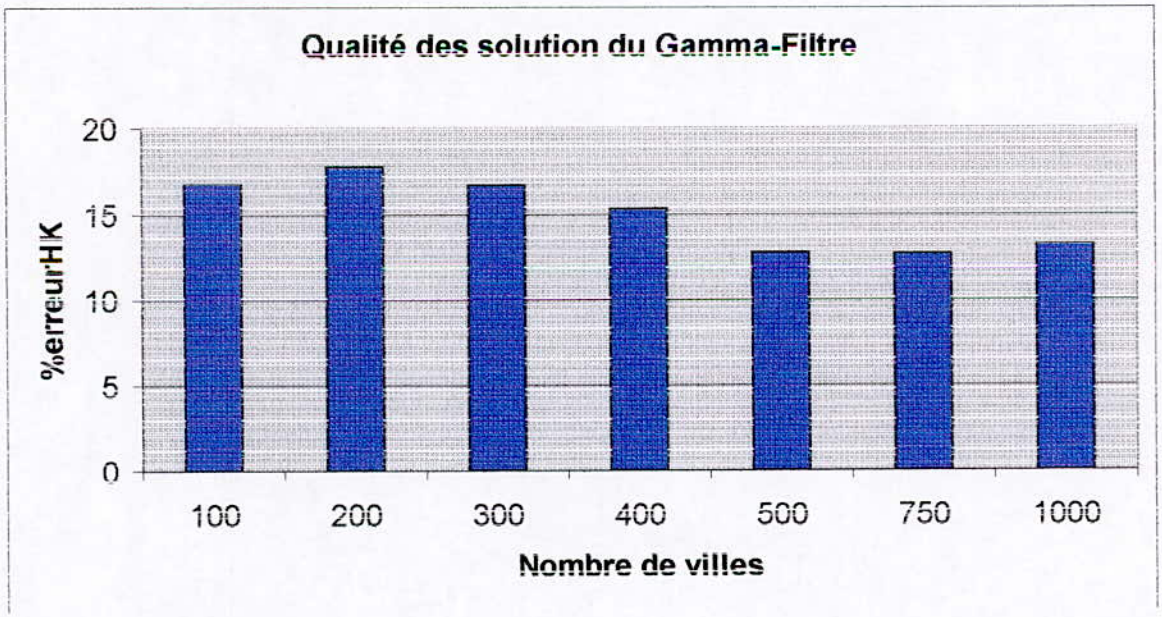


Figure 5.1. Qualité de la solution du γ-filtre

Nous constatons que plus le nombre de villes est important, meilleure est la qualité des solutions données par le *γ-filtre*. Cette méthode serait donc à recommander pour

des problèmes de taille supérieure à 500. Nous faisons ici une tentative d'explication de ces résultats :

Un neurone donné n'étant déplacé que sous l'influence des villes qui lui sont très proches, son déplacement devient fructueux dans la mesure où il n'est pas parasité par les attractions mineures de toutes les autres villes. Il se rapproche donc plus vite des villes pour lesquelles il se spécialisera. Nous rejoignons ici la notion de neurone gagnant des cartes auto-organisées où le neurone le plus proche est déplacé vers la ville à qui il doit la victoire.

Dans le cas des problèmes de grandes tailles la densité des villes est importante il s'ensuit qu'un neurone donné peut être proche de beaucoup de ville à la fois. Mettre en place un filtre limite le nombre des villes considérées comme proches et par conséquent limite les déplacement du neurones à une zone ou direction donnée. De cette façon, les neurones ne sont pas 'ballottés' entres plusieurs zones sans pouvoir se fixer et se spécialiser dans l'une d'elles (et dons être affectés à un ville).

3-Détermination des paramètres

Nous constatons que les paramètres ne sont pas identiques pour les différentes instances du problème. Ils ont été déterminés après tâtonnement et en fonction de la réaction de chaque instance.

Les paramètres α et β pondérant les deux forces agissant sur les neurones ainsi que ε représentant la distance à partir de laquelle un neurone est considéré comme affecté à une ville, ont pris des valeurs différentes selon la taille de l'instance testée. Cela revient à dire que les résultats présentés par les auteurs sont les meilleurs obtenus sur plusieurs essais (avec différents jeux de paramètres). Le nombre initial de neurones ainsi que le rythme de refroidissement de K (température) ont eux aussi fait l'objet de différents essais.

5.2.2. Le réseau élastique hiérarchique

1- Temps de calcul

Nous avons représenté les résultats obtenus par le réseau élastique classique, le réseau hiérarchique et le réseau hiérarchique avec filtre sous forme de graphique (Figure 5.2). Ce graphique montre que la méthode hiérarchique accélère les calculs du réseau élastique de façon considérable, ses temps de calcul peuvent être jusqu'à 10 fois inférieurs à ceux du réseau élastique classique (N=500) l'accélération des temps de calcul est mesurée par :

SP1=temps de calcul du réseau élastique / temps de calcul de la méthode hiérarchique.

La mise en place du filtre provoque une accélération de l'algorithme de :

SP2=temps de calcul de la méthode hiérarchique avec filtre / temps de calcul de la méthode hiérarchique.

L'accélération totale est alors SP=SP1*SP2. Cette accélération ou le gain de temps peut atteindre 16.42 pour 500 villes c'est à dire que le temps de calcul est 16.42 fois inférieur à celui d'un réseau élastique classique.

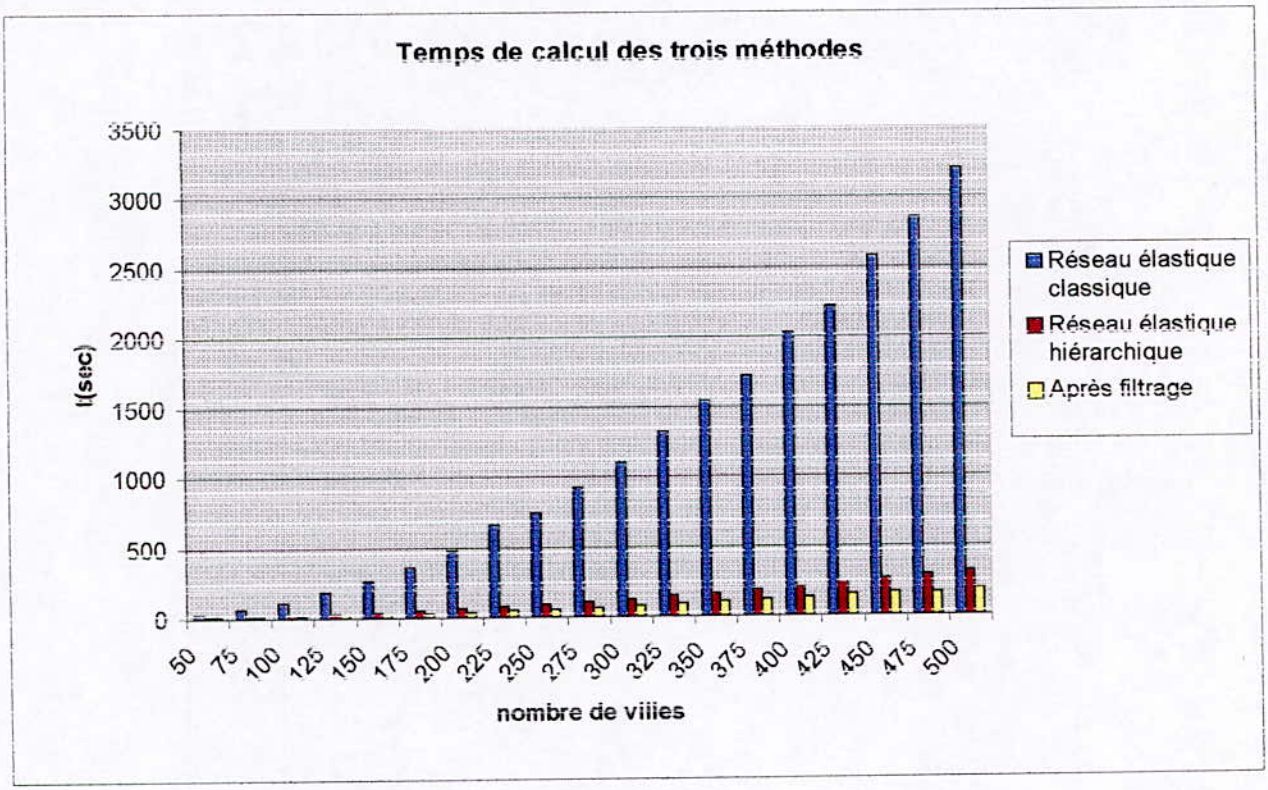


Figure 5.2. Comparaison des temps de calcul des trois méthodes

2- *Qualité des solutions*

Nous avons représenté les longueurs de tour obtenues par les trois méthodes **[30]** sous formes d'histogrammes (Figure 5.3). Nous constatons qu'en plus de réduire les temps de calcul, la méthode hiérarchique améliore la qualité des solutions, il y va de même pour la méthode hiérarchique avec filtre. La méthode hiérarchique peut améliorer la solution de 4.5% (pour N=400), l'amélioration résultant du filtrage peut atteindre 0.2% (N= 325)

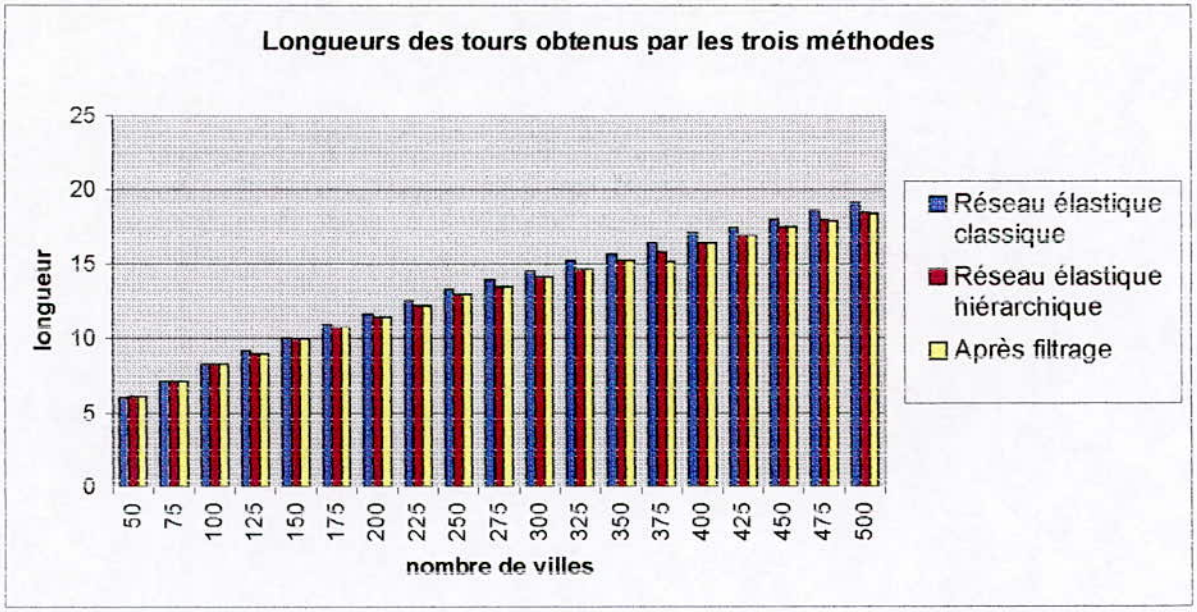


Figure 5.3. Comparaison des solutions obtenues par les trois méthodes **[30]**

Le réseau élastique hiérarchique avec filtre donne des résultats meilleurs que ceux du réseau élastique classique et du réseau hiérarchique sans filtre, en terme de qualité et de rapidité. Ce sont donc ces résultats que nous analyserons. Le graphique suivant (Figure 5.4) présente l'erreur (en %) par rapport à la $borne_{H\text{-}K}$ des solutions.

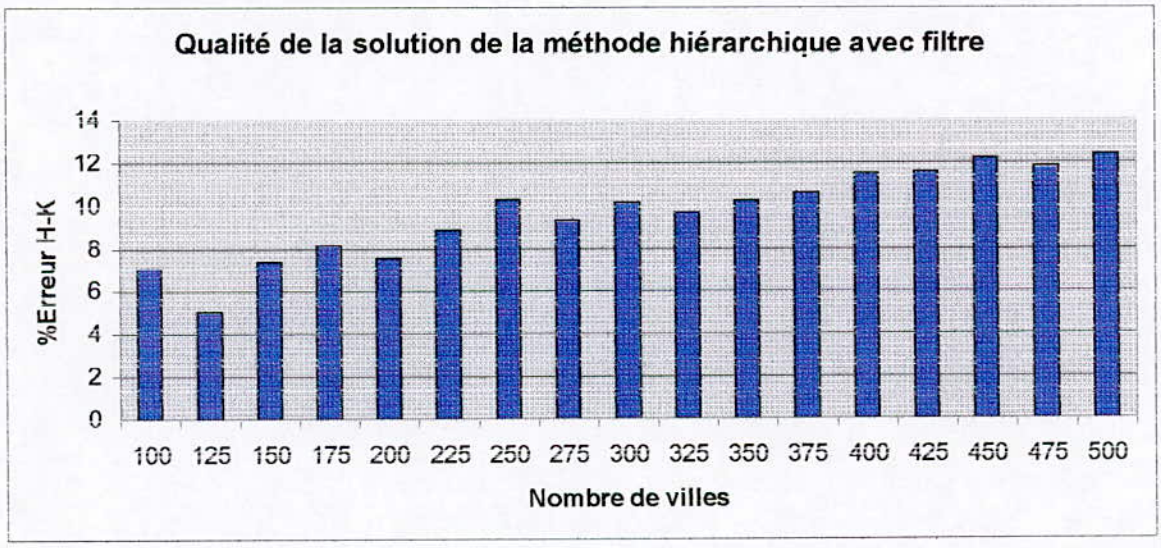


Figure 5.4. Qualité des solutions de la méthode hiérarchique avec filtre **[30]**

La $borne_{H\text{-}K}$ n'étant valable que pour des tailles de problème supérieures à 100, les résultats concernant des problèmes de taille inférieure n'ont pas été pris en compte.

L'écart relatif par rapport à la $borne_{H\text{-}K}$ oscille autour de 10%, il y a une tendance nette vers l'accroissement de cette erreur en fonction de la taille du problème. Cette tendance à la croissance est atténuée par la présence du filtre qui, comme nous

l'avons vu dans la section précédente, tend à améliorer la qualité de la solution lorsque la taille du problème croît.

3- Détermination des paramètres

Concernant les paramètres utilisés dans la méthode hiérarchique, les auteurs nous renvoient aux travaux de J.Y.Potvin[1]. Nous ne connaissons donc pas les paramètres utilisés lors de l'implémentation de la méthode. Le fait de faire référence à cet article montre bien que les auteurs n'ont pas eu recours à un seul jeu de paramètres mais ont dû adapter les paramètres de l'algorithme à la taille des instances testées. Les auteurs **[30]** font aussi référence dans leur article aux travaux de M.W. Simmen[2] (1991) sur la sensibilité du réseau élastique aux paramètres. Tout porte donc à croire que les La sensibilité aux paramètres ainsi que l'absence de formules (théorique ou empirique) pour la détermination de ces paramètres est un handicap important pour les réseaux élastiques.

5.2.3. Le réseau élastique convexe

1-Temps de calcul

Enfin, aucune indication n'a été donnée – du moins dans l'article étudié **[24]**- sur les temps de calcul de la méthode. Mais tout porte à croire qu'il ne rivalise pas avec les deux méthodes précédentes dans la mesure où :

1- Le réseau convexe effectue les calculs sans filtre et, par conséquent, est plus lourd en terme de temps de calcul.
2- Le réseau convexe fait appel à 2-Opt qui est réputée pour sa lenteur.

2-Qualité des solutions

Contrairement aux méthodes précédentes, le réseau convexe n'a pas été testé sur des instances de grande taille : pas plus de 100 villes pour les instances générées aléatoirement et pas plus de 318 sur les problèmes test classiques, nous ne pouvons alors connaître son comportement ou estimer ses performances sur des problèmes de grande taille.

Il faut noter ici que le réseau élastique fait office de procédure de construction de tournée et les bons résultats obtenus sont peut être dus aux opérateurs de réarrangement tels que 2-Opt. Nous ne pouvons donc tirer aucune conclusion quant au fait que le réseau soit initialisé sur l'enveloppe convexe des villes car la procédure 2-Opt prend le relais du réseau de neurone pour améliorer la solution.

---

[1] J.Y. Potvin, 1993. The Traveling Salesman Problem: A Neural Network Perspective, *ORSA Journal on Computing* 5, pp328-348.
[2] M.W. Simmen, 1991. Parameter Sensitivity of the Elastic Net Approach to the Traveling Salesman Problem, *Neural Computation* 3, 363-374.

3-Détermination des paramètres

Aucune indication n'a été donnée sur les paramètres utilisés. Cependant, cette méthode, comme les précédentes, n'échappe pas aux problèmes rencontrés au niveau de la détermination des paramètres du réseau élastique.

5.3. Les cartes auto-organisatrices

Les simulations menées avec les cartes auto-organisatrices ont montré qu'elles convergeaient plus vite que les réseaux élastiques mais qu'elles avaient tendances à produire des tours de qualité moindre. Les cartes auto-organisatrices ont aussi l'avantage de pouvoir appréhender des problèmes de plus grande taille que les réseaux élastiques. **[14]**.

La rapidité des cartes auto-organisatrices est due au fait qu'elles ne déplacent que le neurone gagnant et ses quelques voisins vers la ville présentée alors que les réseaux élastiques déplacent tous les neurones en même temps et sous l'influence de toutes les villes. De plus le voisinage décroît au cours du temps, ce qui revient à déplacer de moins en moins de neurones.

5.3.1. La carte auto-organisatrice Flexmap

1-Temps de calcul

En termes de temps de calcul, FLAXMAP est un algorithme à complexité linéaire (O(N) = constante.N) ce qui est sans précédant pour un algorithme de résolution du TSP. Le temps nécessaire à la résolution d'une instance à 1000 villes sur une station de travail Sun Sparc 2 est de 7mn. **[12]**.

Cependant, d'après **[14]**, la constante de proportionnalité est très élevée ce qui rendrait l'algorithme proche de $O(N^2)$.

2-Qualité des solutions

Pour des problèmes de taille inférieure à 400, les solutions de Flexmap sont proches de celles données par 2-Opt et sont souvent meilleures.

La méthode FLEXMAP est capable de traiter des problèmes de grande taille (jusqu'à 2392 villes) avec un écart de moins de 9% de l'optimalité. Les résultats de Fritzk et Wilke sont incontestables car ils ont été comparés à la vraie solution optimale. De plus des résultats sont assez proches de l'heuristique conventionnelle 2-Opt.

3- Détermination des paramètres

Outre ses résultats, l'intérêt de cette méthode réside dans le fait qu'elle utilise des paramètres constants au lieu de les faire décroître comme l'algorithme d'apprentissage des cartes auto-organisées le prévoit. Le voisinage ainsi que le facteur d'apprentissage sont fixes. Ils le sont non seulement pendant la durée d'adaptation mais aussi quelque soit la taille du problème traité. L'algorithme Flexmap, n'a pas recours explicitement à une fonction noyau fin de pondérer le

déplacement des voisins d'un neurone gagnant mais affecte un taux d'apprentissage neurone gagnant (ε=0.5) qui diffère de celui de ses deux voisins (ε=0.25). **[12]**

5.3.2. La compétition harmonique

1-Temps de calcul

Les caractéristiques des machines utilisées pour l'implémentation de la compétition harmonique et de Flexmap n'étant pas les mêmes, nous pouvons tout de même affirmer que la Flexmap est la plus rapide. En effet, Flexmap met 7mn pour aboutir à une solution sur une instance à 1000 villes alors que la compétition harmonique met 4h6mn sur une instance à 532 villes.

Cela s'explique par le fait que les recherches des gagnants et des arêtes à grande erreur sont locales et que les paramètres de Flexmap sont constants et ne sont pas modifiés à chaque étape. Ce n'est pas le cas de la compétition harmonique qui doit, à chaque étape, ajuster : le voisinage, le taux d'apprentissage, le paramètre λ, les fonctions $\overline{f}$, $\overline{g}$ et de voisinage ainsi que les pourcentages d'appréhension de tous les neurones. Plusieurs tests doivent aussi être effectués : ceux déterminant le gagnant de chaque ville, ceux déterminant si un neurone doit être dédoublé ou pas et ceux déterminant le sort de ε et λ. Il est donc certain qu'en terme de temps de calcul la compétition harmonique n'apporte aucune amélioration.

2-Qualité de la solution

Cette méthode n'a été testée que sur une seule instance, sur instance de taille 532 la méthode **[22]** à produit une solution dont l'écart par rapport à l'optimum est de 2.49%. Les résultats obtenus sont très bons et augurent un bon comportement de la méthode pour des problèmes de plus petite taille. Les simulations ont montré que la formule la plus simple de contrôle de λ c'est à dire $\lambda(t)=\frac{a\overline{f}(t)}{g(t)}$ était la plus efficace.

Ce contrôle des paramètres est sans doute à l'origine de la qualité de la solution obtenue. Le fait que le taux d'apprentissage et le paramètre λ soient modifiés lors du processus d'adaptation peut constituer un avantage de la méthode. Le taux d'apprentissage, par exemple, ne fait plus que décroître inexorablement sans tenir compte de l'état d'avancement du processus d'adaptation. Il est désormais de tendance croissante et il est autorisé à diminuer lorsque cela est nécessaire, à un stade donné du processus d'adaptation.

3-Détermination des paramètres

Le fait d'ajouter des neurones lorsque le pourcentage d'appréhension dépasse un certain seuil pose le problème de la détermination d'un nouveau paramètre : r_0. Une procédure analogue à celle de Flexmap aurait été souhaitable. En effet, ajouter un neurone sur une arête dont l'erreur est maximale est, d'une part, plus conforme aux besoins de l'algorithme et d'autre part, évite d'avoir recours à un paramètre supplémentaire.

Conclusion

Les méthodes Flexmap et de compétition harmonique nous enseignent que les paramètres peuvent être gérés de façons différentes de celle préconisée par l'algorithme des cartes auto-organisées classique.

5.4. Méthode retenue pour l'implémentation

La qualité de la solution obtenue avec les réseaux élastiques est en partie due au terme de pénalité : $\alpha(t)\ (y_{j+1}-2y_j+y_{j-1})$ qui fait en sorte de maintenir la cohésion entre les neurones et par conséquent maintient l'anneau de neurones la plus serré et dons le plus court possible.

La méthode de compétition harmonique de Matsuyama **[22]** faisant appel à la fois aux cartes auto- organisatrices et au terme de pénalité des réseaux élastiques à toutes les chances de converger plus rapidement que les réseaux élastiques tout en obtenant des solutions de bonne qualité.

De plus les performances annoncées de la méthode sur une instance à 532 villes, 2.5% de l'optimum réel ont fait que notre choix s'est porté sur cette méthode. Nous procéderons donc à son implémentation et à des simulations sur une batterie de problèmes test afin d'en vérifier les performances en terme de temps de calcul et de qualité des solutions.

6.1. Introduction

6.2. Présentation du programme

Nous avons commencé par programmer à l'aide du logiciel Matlab 5.3 qui possède une boite à outils performante dans le domaine des réseaux de neurones. Bien que permettant de définir notre propre topologie, la boite à outils ne permettait pas d'ajuster le taux d'apprentissage de façon dynamique. La règle de modification par défaut prévoyait la décroissance de ε pendant une première phase puis son maintien à une certaine valeur dans une seconde phase.

Nous nous somme alors orientés vers l'environnement de programmation Delphi5 (version professionnelle) basé sur le langage de programmation Pascal. Les fonctions et procédures ont toutes été reprogrammées.

Nous avons doté notre programme d'une interface graphique dans laquelle il est possible de visualiser l'anneau de neurones initial, le comportement de l'anneau de neurones à chaque étape ainsi que le tour obtenu épuré de tous les neurones superflus.

L'interface affiche également le nombre de neurones, les valeurs de σ et ε à chaque étape ainsi que le numéro de l'étape.

Pour effectuer une simulation sur un problème test classique, il suffit d'ouvrir le fichier correspondant aux coordonnées des villes ainsi que le fichier correspondant à la solution optimale. Il est également possible d'avoir recours à des instances générées aléatoirement dans le carré unité. Il suffit pour cela d'entrer le nombre de villes dans la case correspondante et de cliquer sur le bouton 'Aléatoire'.

Les dates de début et de fin d'exécution et les pourcentages d'écart par rapport aux bornes de Stein et de Held et Karp s'affichent à la fin de l'exécution. Nous pouvons visualiser la longueur du tour obtenu par la méthode de compétition. Le pourcentage d'erreur par rapport à l'optimum (dans le cas d'un problème test classique) peut être obtenu en cliquant sur le bouton correspondant.

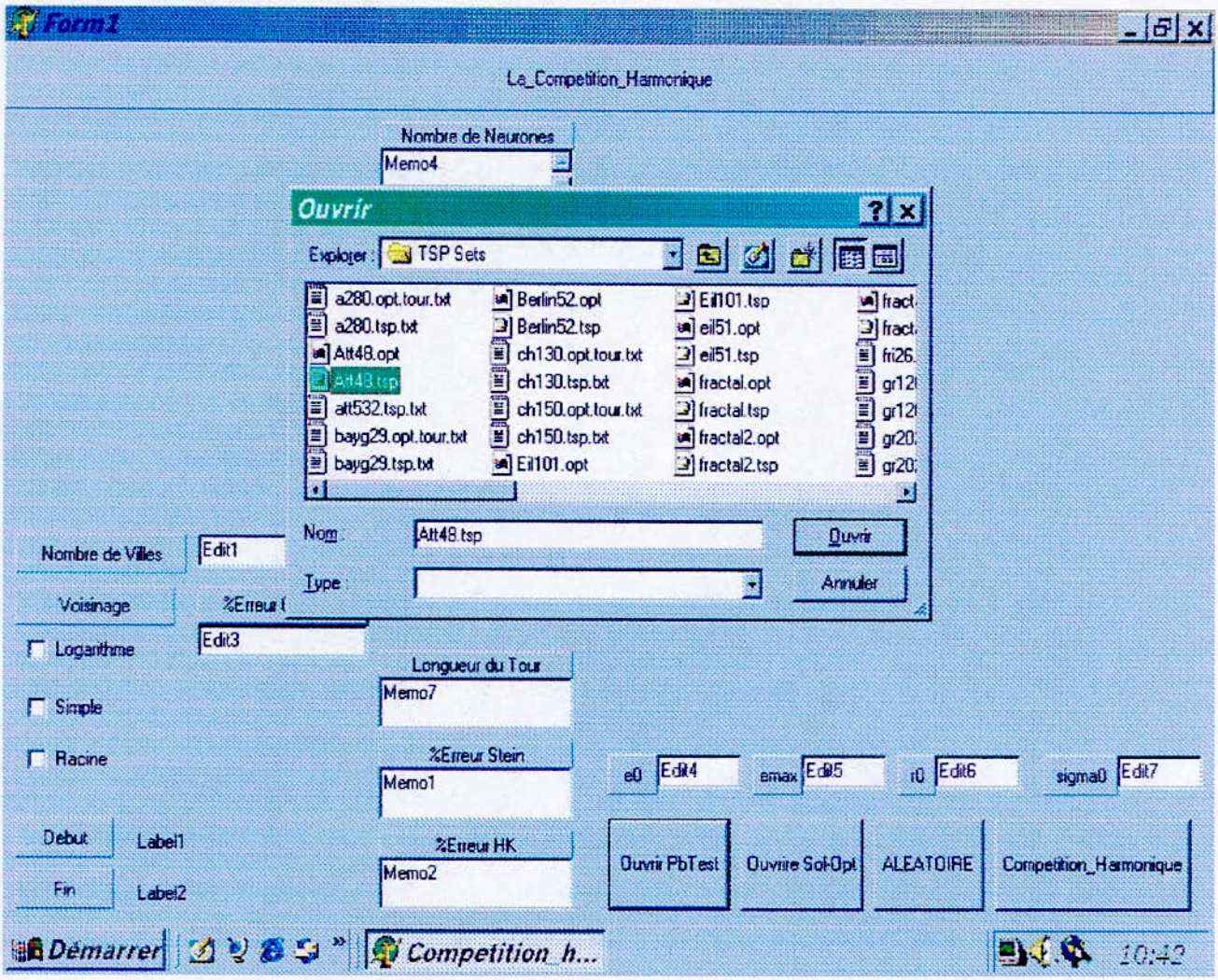


Figure 6.1. L'interface du programme de la méthode de compétition harmonique lors de l'ouverture d'un fichier de problème test.

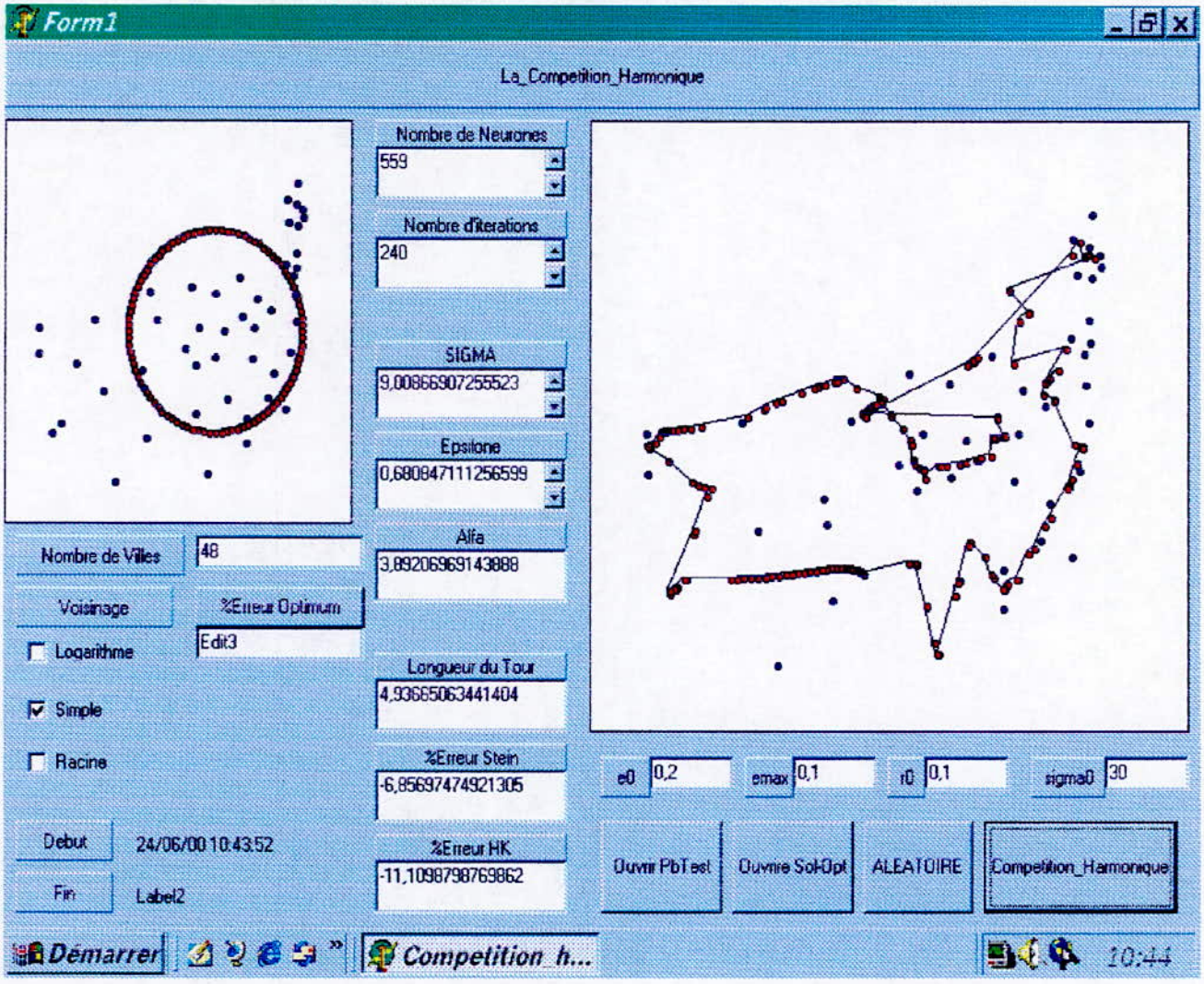


Figure 6.2. Le programme pendant l'exécution de la compétition harmonique

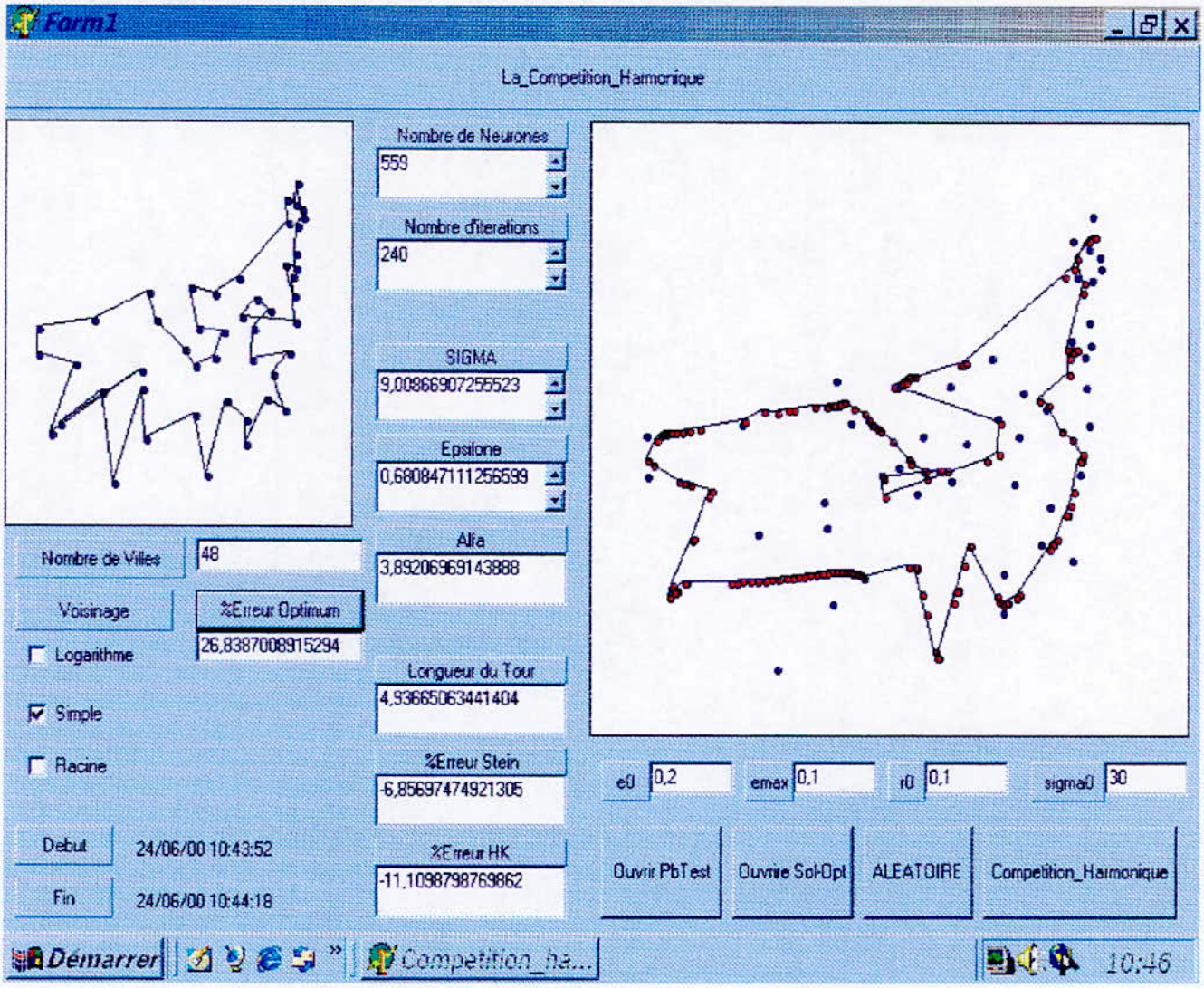


Figure 6.3. Affichage des résultats après convergence de l'algorithme.

6.3. Choix des problèmes test

Les simulations seront effectuées sur deux catégories de problèmes test :

- La première catégorie comprend les problèmes test classiques dont l'optimum réel est connu. Cette catégorie de problèmes servira essentiellement à évaluer les performances de l'algorithme sur les problèmes de petite taille dans la mesure où la borne de Held et Karp ne s'applique qu'aux instances de taille supérieure à 100.
- La seconde catégorie est constituée d'instances générées aléatoirement dans le carré unité. Les solutions obtenues sur ces instances seront comparées à la borne de Held et Karp.

Les problèmes test ont été téléchargés à partir du site Internet de TSPLIB **[33]**.

6.4. Implémentation

Nous avons effectué l'implémentation sur un PC possédant un processeur AMD K6-2 de 400Mhz de fréquence, une mémoire cache de 512 Ko et une RAM de 64 Mo.

Les paramètres du problème

Les valeurs des paramètres et les formules permettant de les mettre à jour à chaque étape sont celles préconisées par l'auteur [22] à savoir :

$\lambda(t) = \bar{f}(t).a.t\ /[\varepsilon(t).\ \bar{g}(t)]]$

$\varepsilon_0 = 0.25$

$\sigma=\sigma_0.(1-s)^t$

$a=2.N^2.10^{-7}$

L'ampleur du voisinage à chaque étape est égale à 2. $\sigma(t)$ de part et d'autre du neurone gagnant. C'est à dire que les 2. $\sigma(t)$ neurones se situant à droite et à gauche du neurone gagnant feront l'objet d'un déplacement. La condition d'appartenance au voisinage s'écrit comme suit :

Le neurone m appartient au voisinage du neurone j si : $|m-j| < 2.\ \sigma(t)$

Nous donnons ici les valeurs de paramètres pour lesquels nous n'avons pas eu d'informations et les explications de nos choix :

Nous fixerons la valeur d'ε_{max} (le seuil en deçà duquel le taux d'apprentissage ε ne doit pas descendre) à 0.1, il s'agit d'un choix arbitraire. Nous n'avons pas effectué d'étude sur ce paramètre mais dans la mesure où le taux d'apprentissage tend à croître, le seuil ε_{max} n'est qu'une sécurité. Nous avons par conséquent pris ε_{max} légèrement inférieur au facteur de départ ε_0.

Nous avons pris σ_0 autour de N/3 (le tiers de la taille du problème), comme cela est préconisé pour l'algorithme de Kohonen classique. **[inter-2]**

Le nombre initial de neurones est fixé à 2N. Ainsi, au lieu de consacrer ses premières étapes à accroître le nombre de neurones, l'algorithme pourra commencer avec un nombre suffisant de neurones et entamer des déplacements efficaces.

6.5. Résultats de l'implémentation et analyse

Lors des simulations, nous avons observé le comportement de l'anneau de neurones et en avons dégagé les remarques suivantes :

- Les neurones se dédoublent de façon trop rapide, l'anneau de neurones devient très vite très dense et les calculs sont ralentis dès les premières itérations. Prenons l'exemple d'une instance aléatoire à 40 villes, le nombre

de neurones atteint 3818 au bout de deux minutes. Les paramètres pour cette instance ont été fixés comme suit : ε_0=0.2, ε_{max}=0.1, σ_0=25 et r_0=0.1. La figure 6.4 illustre le comportement de l'anneau sur l'instance citée.

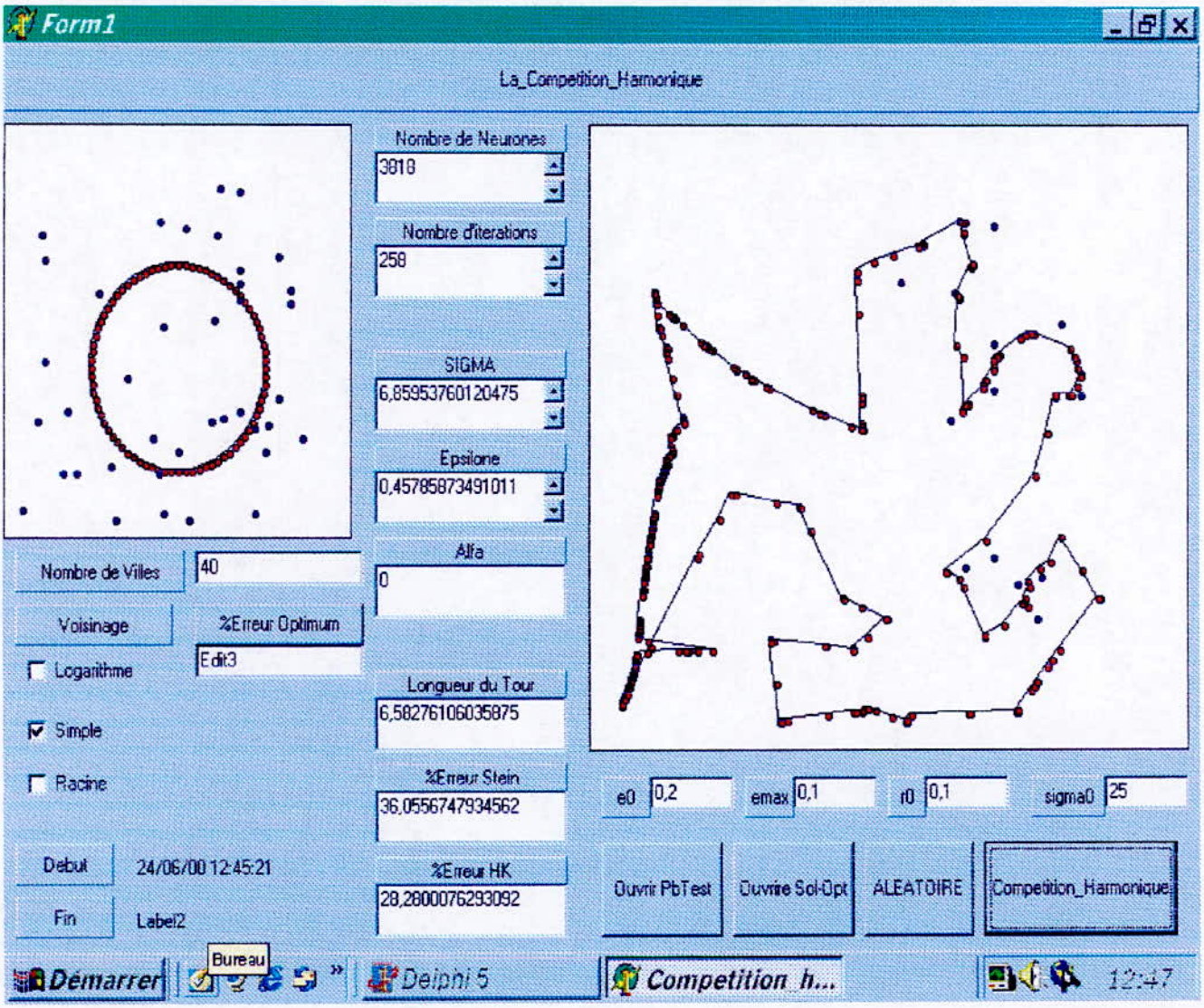


Figure 6.4. Comportement de l'algorithme initial de compétition harmonique.

- Les neurones ajoutés en cours d'exécution étant placés à la même position que le neurone auquel ils doivent la naissance, ils finissent par former des 'paquets'. L'anneau de neurones prend alors l'allure représentée dans la figure 6.5.

- La convergence n'est pas réalisée pour la plupart des instances testées, le programme continue à tourner sans que le nombre de neurone ni leurs emplacements ne soient modifiés.
- Dans le peu de cas où l'algorithme converge, les solutions obtenues sont assez éloignées de la borne inférieure de la solution optimale. L'exemple suivant (Figure 6.5) illustre, les performances de la méthode sur une instance à 100 villes générée aléatoirement. L'écart par rapport à la borne inférieure de la solution optimale se situe autour de 60%. Le temps de calcul est de 8mn, ce temps pourrait être meilleur si l'anneau ne contenait pas autant de neurones. En effet, le temps de calcul est proportionnel à la taille de l'anneau dans la mesure où, à chacune des N itérations de chaque étape, l'anneau doit être parcouru afin de déterminer le gagnant de la ville présentée.

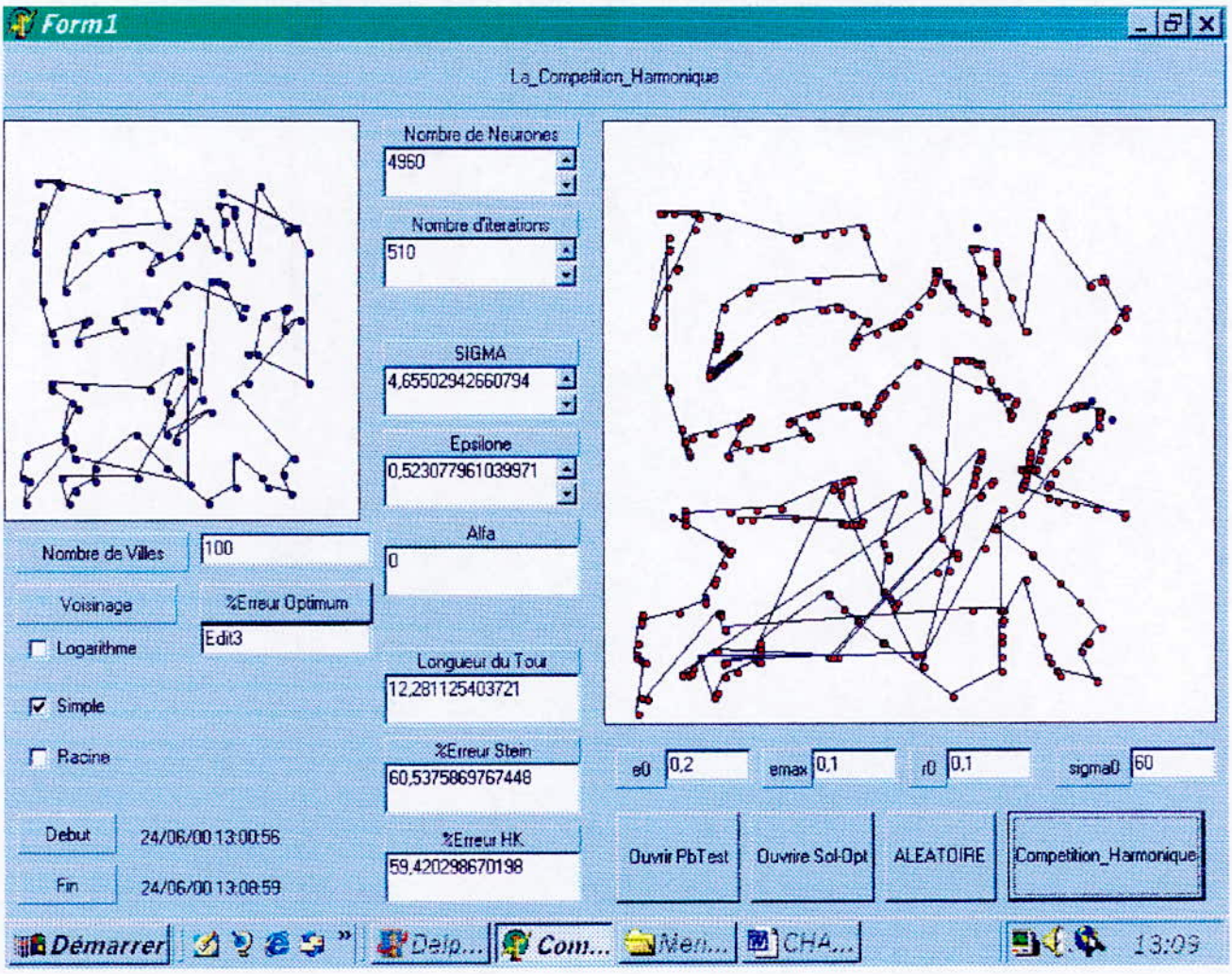


Figure 6.5. Résultat de la compétition harmonique sur une instance à 100 villes.

- Deux facteurs nous paraissent contribuer aussi à la non convergence de l'algorithme :

1- Le seuil r_0 est fixe et ne suit pas le processus d'adaptation. Il en résulte qu'au moment ou l'anneau paraît s'être bien adapté à la configuration des villes, un neurone se situant entre deux villes très proches n'est pas dédoublé et par

conséquent oscille sans fin entre ces deux villes. Ce neurone n'est pas dédoublé car son taux d'appréhension est faible étant donné que la distance qui le sépare de ces deux villes est très faible.

2- Il arrive que le voisinage décroisse jusqu'à atteindre la valeur nulle sans que l'anneau n'aie eu le temps de s'adapter. Cela est dû à une inadéquation entre le voisinage qui décroît sans cesse et la taille de l'anneau qui ne fait que croître. Une solution possible serait de commencer avec un voisinage grand, mais un voisinage trop grand fait diverger l'anneau dès le départ car celui-ci est entièrement déplacé sous l'effet de chaque ville. Le problème de la détermination de σ_0 se pose donc pour chaque taille de problème. De plus d'une instance à une autre l'algorithme peut avoir un comportement différent en raison de la distribution des villes dans le plan.

6.6. Propositions destinées à améliorer la convergence :

Au vu du comportement de la méthode, nous faisons ici quelques propositions qui pourraient améliorer les performances du programme.

6.6.1. Le nombre de neurones

Nous maintenons le nombre de neurone à 2N car un nombre insuffisant de neurones rendrait l'anneau instable dans la mesure où des neurones seront gagnants pour plusieurs villes et seront par conséquent ' ballottés' entre ces villes.

Nous aurions pu envisager la possibilité de réduire le nombre de neurone en éliminant ceux qui ne sont pas déclarés gagnants pendant un certain nombre d'étapes (comme pour Flexmap), ce qui aurait réduit le nombre de calculs à effectuer. Cependant, la compétition harmonique faisant intervenir le terme de pénalité des réseaux élastiques, chaque neurone est attiré par ses deux voisins. Moins il y aura de neurones et plus ceux-ci seront éloignés les uns des autres, la force de cohésion : $\alpha(t)(y_{j+1}-2y_j+y_{j-1})$, pour un neurone donné y_j, étant proportionnelle aux distances entre ce neurone et ses deux voisins n'en sera que plus importante. Il s'ensuit que l'anneau de neurones tendra à rétrécir plutôt qu'à s'étendre pour se rapprocher des villes. C'est pour cette raison que nous n'éliminerons pas de neurones en cours d'exécution.

6.6.2. Mise à jour du nombre de neurones

La mise à jour du nombre de neurone se fait toutes les 10 étapes afin d'éviter qu'ils ne se multiplient trop rapidement. Ce choix est arbitraire et a été motivé par le nombre important d'itérations que le programme effectue (peut atteindre 40 pour un problème de taille22).

Pour qu'un neurone soit dédoublé, il faut qu'il satisfasse la condition suivante :

Condition : Que son taux de vigilance (r) soit supérieur à r0 (catch pourcentage) et qu'il soit non croissant, le test sur la condition de non croissance est effectué sur les deux dernières valeurs de r.

Lors du dédoublement d'un neurone, le 'neurone ajouté' n'est pas placé à la même position que son 'jumeau' mais entre celui ci et l'un de ses deux voisins (comme pour Flexmap). Toutes les 10 étapes, lorsque le nombre de neurones est mis à jour, tous les neurones sont replacés sur l'anneau de sorte qu'ils soient équidistants. Ce choix de positionnement redistribue bien les neurones sur l'anneau afin que le 'neurone ajouté' puisse servir de gagnant à une autre ville que celle du neurone gagnant à qui il doit la naissance.

6.6.3. Le voisinage

Faire décroître le voisinage revient à déplacer de moins en moins de neurones vers chaque ville. Dans la mesure ou le nombre de neurones ne fait que croître, viendra le moment où le voisinage sera nul. A partir de ce moment, seul le neurone gagnant sera déplacé, alors que l'anneau contiendra beaucoup de neurones. Le neurone gagnant reviendra aussi tôt à sa précédente position attiré par ses deux voisins qui apparaissent dans le terme de pénalité : $(y_{j+1}-2y_j+y_{j-1})$

Visuellement, cela se traduit par un anneau de neurone immobile et très dense duquel quelques neurones s'écartent individuellement et reviennent aussitôt à leurs positions. L'anneau ne se déforme plus et stagne. Il s'ensuit que l'algorithme aura des difficultés à converger.

Aussi, pour ajuster le voisinage à la taille de l'anneau de neurones nous proposons les modifications suivantes pour $\sigma(t)$: $\sigma(t) = \sigma_0.(1-s)^t.\sqrt{M}$,

et $\sigma(t) = \sigma_0.(1-s)^t.\ln(M)$, M étant le nombre de neurones.

Ces deux formules sont composées de deux termes :

- Le terme classique $\sigma_0.(1-s)^t$ assurant la décroissance du voisinage à des fins de convergence.
- Le second, $\sqrt{M}$ ou bien $\ln(M)$, fait en sorte que l'ampleur du voisinage accompagne la taille de l'anneau de neurones.

Le paramètre σ_0 sera fixé à 1 dans la mesure où $\sqrt{M_0}$ et $\ln(M_0)$ constituent le voisinage de départ (M_0 étant le nombre initial de neurones). Il ne sera donc plus nécessaire de déterminer le paramètre σ_0.

6.7. Implémentation et résultats

Nous avons procédé n trois étapes :

1- Introduction de la mise à jour des neurones et leur redistribution toutes les 10 étapes en gardant le voisinage décroissant. (variante 1)
2- Introduction du logarithme de M dans la formule de voisinage (variante 2)
3- Introduction de la racine de M dans la formule de voisinage (variante 3)

1- Les simulations sur des problèmes test classiques :

Nous avons procédé à des tests sur les problèmes classiques suivants : Ulysse 16, Ulysse 22, Att 48, Berlin 52, Eil 51, St 70 et Eil 101. pour chaque problème, nous avons effectué 8 simulations. Les moyenne obtenues sur les 8 simulations sont représentées sous forme d'histogrammes (cf. Figure 6.6 et 6.7). Les résultats complets sont dans les annexes III, IV et V. Les paramètres utilisés ont té les mêmes pour les trois variantes et tous les problèmes test à savoir :
Les paramètres : σ_0=1, ε_0=0.2, ε_{max}=0.1, r_0=0.1pour les variantes 2 et 3 et σ_0=N/3 pour la variante 1.

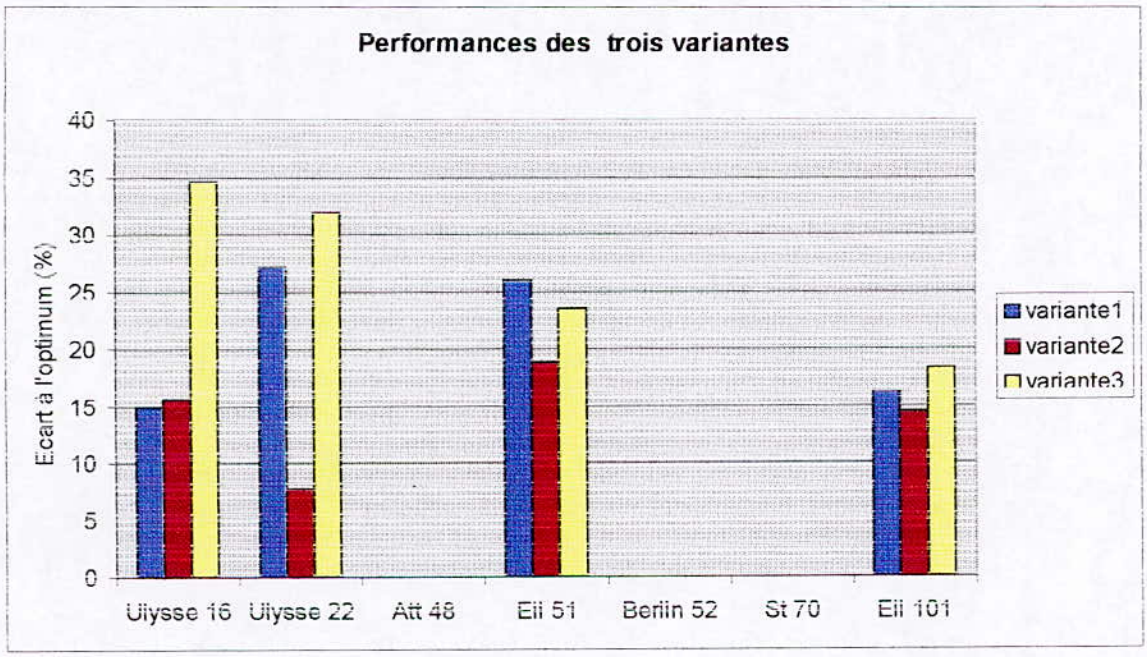


Figure 6.6. Résultats des simulations sur les problèmes test classiques en terme de qualité des solutions.

Les instances : Att 48, Berlin 52 et St 70 n'ont pas été représentées car les trois variantes stagnent au bout d'un certain nombre d'itérations sur instances. Faire varier le voisinage de départ de la variante 1 ne change n'apporte pas d'améliorations. Le nombre de neurones reste constant, les neurones ne se déplacent plus alors que l'algorithme continue de tourner. Une des raisons de cet état de fait est la constance de r_0 qui ne s'adapte pas à l'évolution de l'anneau. Par conséquent, à un certain stade de la simulation, des neurones devraient être dédoublés même s'ils ne satisfont pas la condition dictée par Matsuyama **[22]**. Ces neurones sont très proches d'un amas de villes et leur pourcentage d'appréhension reste inférieur à r_0, il s'ensuit que ces neurones demeureront 'ballottés' entre plusieurs villes à l'infini.

Nous devons faire remarquer ici que les résultats de la compétition harmonique se soient améliorés après introduction de la mise à jour du nombre de neurones toute les 10 itérations et l'équidistance. En effet, la méthode ne rencontre plus autant de problèmes de convergence. Cependant ces résultats restent moyens et bien

éloignés des performances de la méthode sur l'instance à 532 villes. C'est à ce niveau qu'interviennent les paramètres que nous avons fixés. Un meilleur choix de ces paramètres aurait contribué à améliorer les résultas.

Nous voyons bien d'après la figure 6.6 que le voisinage logarithmique produit de meilleures solutions que les deux autres variantes.
Cela peut s'expliquer par le fait que le logarithme est inférieur en valeur à la racine, cela produit un voisinage plus petit mais toujours 'proportionnel' au nombre de neurones. Un tel voisinage semble bien convenir aux instances de petites tailles. De plus, les résultats de la variante 3 semble s'améliorer lorsque la taille du problème croit.

La figure 6.7. illustre les performances des trois variantes en terme de temps de calcul.

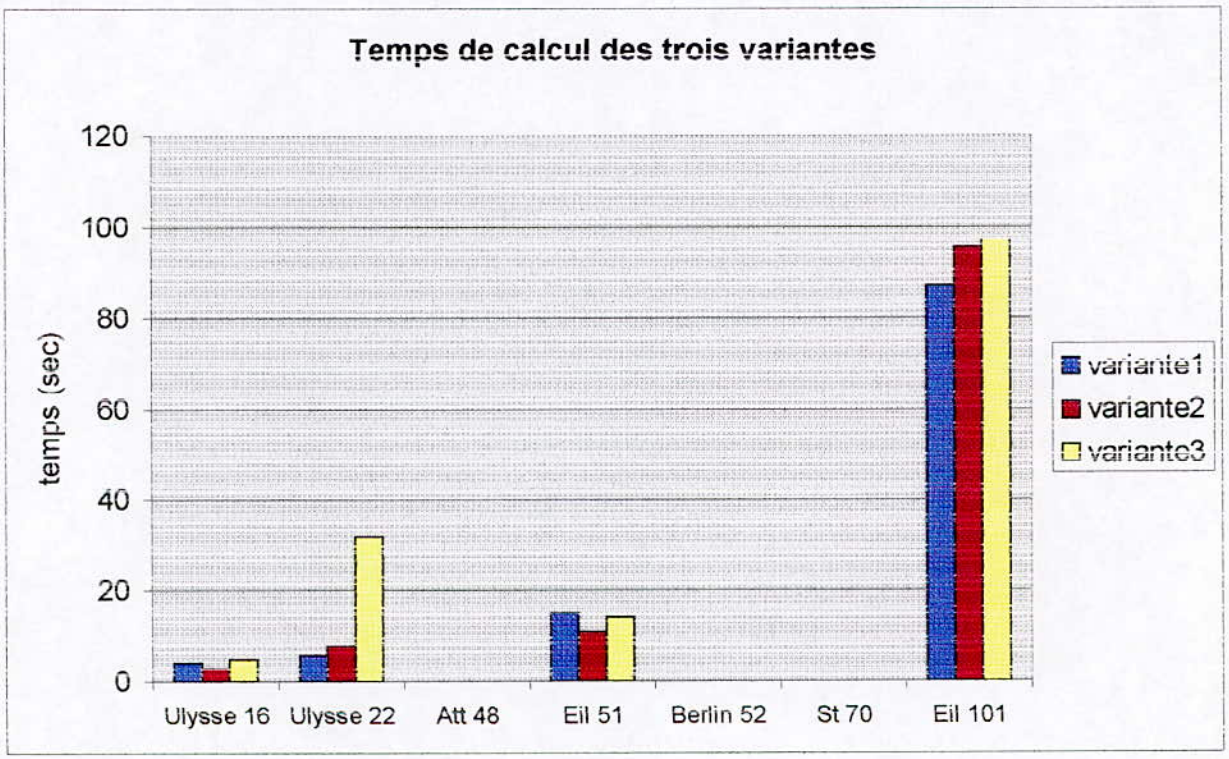


Figure 6.7. Résultats des simulations sur les problèmes test classiques en terme de temps de calcul.

Nous remarquons que la variante 3 est celle qui produit les plus grands temps de calculs. Cependant, hormis le cas de Ulysse 22, l'écart de temps est faible (de l'ordre de la dizaine de secondes pour 101 villes.

Une telle série de simulation ne saurait apporter d'éléments incontestables quant aux performance des trois variantes. Cependant, nous pouvons d'ores et déjà prétendre, pour les raisons citées plus haut, que le voisinage logarithmique est plus approprié aux problèmes de taille inférieure à 100.

2- Les simulations sur des instances aléatoires

Nous avons effectué 10 simulations sur des instances de taille 120 générées aléatoirement. Les valeurs présentées dans les trios tableaux suivants représentent les moyennes des écarts par rapport à la borne de Held et Karp et des temps de calcul sur ces 10 instances :

Résultats sur 10 instances de taille 120 générées aléatoirement:

Variante 1 :

Les paramètres : σ_0=30, ε_0=0.2, ε_{max}=0.1, r_0=0.1

Ecart par rapport à l'optimum %	Temps de calcul (sec)
24.55	203.9

Variante 2 :

Les paramètres : σ_0=1, ε_0=0.2, ε_{max}=0.1, r_0=0.1

Ecart à l'optimum %	Temps(sec)
26.18	72.2

Variante3 :

Les paramètres : σ_0=1, ε_0=0.2, ε_{max}=0.1, r_0=0.1

Ecart par rapport à l'optimum %	Temps de calcul (sec)
21.99	114.6

La supériorité de la variante 2 est nette en terme de temps de calcul, le temps de calcul de la variante 3 lui est 1.6 fois supérieur et celui de la variante1 2.8 fois supérieur.

Quant à la qualité de la solution, la variante 3 semble prendre le dessus suivie par la variante 1 puis la variante 2. Il est probable que la

Le voisinage avec racine semble donc plus approprié aux problèmes dont la taille est de l'ordre de 100.

Conclusion

Nous avons dans cette partie du travail, programmé la méthode de compétition harmonique afin de s'assurer des performances annoncées par l'auteur, à savoir un écart de 2.5% de l'optimum sur une instance à 532 villes. Nous nous sommes heurté au problème de détermination des paramètres initiaux. Nous les avons fixé en nous basant sur des information que nous avons recueillies et quelques raisonnements.

L'observation du comportement de la méthode nous ont amené à proposé un mécanisme de dédoublement des neurones qui a porté ses fruits.
Nous avons proposé deux formules de voisinage destinées à éviter la détermination du paramètre σ_0 et d'accompagner la taille du réseau qui ne fait que croître. Les premiers résultats semblent indiquer le voisinage avec logarithme est plus approprié aux problèmes de petite taille, et la racine aux problèmes de plus grande taille.

Des tests plus poussés, ayant recours à la statistique (la méthode ANOVA par exemple) devraient être menés afin de confirmer ces premières constatations. Le nombre de problèmes test devrait être important et le nombre de simulations également.

Nous faisons remarquer que bien que les voisinages proposés donne d'assez 'bons' résultas, leur mise en œuvre nécessite plus de temps de calcul que le voisinage classique ou qu'un voisinage constant. en effet, il faut calculer à chaque étape un logarithme ou une racine.

Conclusion et perspectives

Au cours de ce travail, nous avons étudié et présenté les méthodes neuronales d'optimisation combinatoire. Parmi ces méthodes, nous avons opté pour la seconde famille, c'est à dire la famille des méthodes géométriques. Au sein de cette famille, nous avons distingué deux classes s'inspirant toutes les deux plus ou moins des cartes auto-organisatrices de Kohonen, à savoir : les réseaux élastiques et les cartes auto-organisatrices.
Ces dernières, contrairement aux réseaux élastiques, ne produisent pas de solutions irréalisables. Cela est dû au fait qu'un neurone gagnant doit être affecté à chaque ville pour que la convergence ait lieu.

Nous avons pu nous apercevoir que les méthodes neuronales géométriques étaient très différentes des heuristiques du TSP. Toutes les autres méthodes répertoriées au chapitre 1 sont des heuristiques de recherche locale au sens large. En effet, ces dernières démarrent toutes d'un tour initial qu'elles sont appelées à améliorer. A l'opposé, les méthodes neuronales aboutissent à un tour en partant de 'rien', en l'occurrence un anneau de neurone initial dans le plan.

Dans la partie consacrée à l'étude des méthodes présentées, nous avons abordé le problème des paramètres initiaux qu'il faut déterminer en fonction de la taille du problème - et probablement - de la configuration des villes dans le plan. Les différentes améliorations des méthodes originelles nous ont enseigné que les lois d'adaptation des paramètres pouvaient différer de celles de l'algorithme originel tout en produisant des solutions de meilleure qualité.

Nous avons tenté, dans la dernière partie de ce travail de proposer deux alternatives à la formule classique du voisinage dans le cas de la compétition harmonique. Nous avons obtenu des résultats préliminaires qui indiquent que des améliorations sont possibles à partir de ces alternatives. Nos résultats devraient être confirmés par des tests plus rigoureux.

Bibliographie

[1] W.Arrouy

Les réseaux de neurones

http://www.ie2.u-psud.fr/~coutris/neural/

[2] Aoufi et Souayeb

Approche Statistique du Problème du Voyageur de Commerce

Projet de Fin d'Etudes. Département du Génie Industriel, ENP 1996.

[3] Azibi et Mecellem

Les réseaux de neurones artificiels : Outil de traitement d'information. Application : Prédiction des prix de pétrole

Projet de Fin d'Etudes. Département du Génie Industriel, ENP 1998.

[4] J.Bartholdi

Space filling curves, fractals, etc.

http://www.isye.gatech.edu/people/faculty/John_Bartholdi/

[5] Bhide & Kabuka

A Real Time Solution for the Traveling Salesman Problem using a Boolean neural network

IEEE International Conference on Neural Networks Vol2.1993. pp 1096-1103.

[6] M.C.S.Boeres, L.A.V. de Cavalho & V.C.Barbosa

A Faster Elastic Net for the Traveling Salesman Problem

International Joint Conference on Neural Networks. Vol 2. 1992. pp 215-220.

[7] D.J.Burr

An Improved Elastic Net Method for the Traveling Salesman Problem

IEEE International Conference on Neural Networks.1988. pp 69-76.

[8] S.Cavalieri & M.Russo

Solving Constraint and Optimisation Satisfaction Problems by a Neuro-Fuzzy Approach

IEEE transactions on Systems, Men & Cybernetics, Part B.Vol29,N°6, December 1999. pp 618-620.

[9] M.M.Corsini

Réseaux de Neurones Artificiels

http://durandal.mass.u-bordeaux2.fr/~corsini/ANN/main/. 1998.

[10] E.Davalo, P.Naïm

Des réseaux de neurones artificiels

Editions Eyrolles 1989.

[11] M.Dorigo et L.M.Gambardella

Ant Colonies for the Traveling Salesman Problem
ftp://iridia.ulb.ac.be/pub/mdorigo/journals/IJ.15-BIOSYS97.pdf. 1996.

[12] B.Fritzk & P.Wilke

FLEXMAP- a Neural Network for the TSP with Linear Time and Space Complexity
TSPBIB. 1991.

[13] A.H.Gee & R.W.Prager

Limitation of Neural Networks for Solving Traveling Saleman Problem

IEEE transactions on Neural Networks,Vol 6,N°1, January 1995. pp280-282.

[14] Johnson, Mc Geoch

The traveling salesman problem: a case study in local optimisation. Version préliminaire du chapitre paru dans: Local serach in combinatorial optimisation

Aarts, Lenstra, John Wiley & sons. 1997.

[15] D.S. Johnson, L.A.McGeoch & E.E.Rothberg

Asymptotique Experimental Analysis for the HELD-Karp Traveling Salesman Problem

Proceedings of the 7th annual ACM-SIAM Symposium on Discrete algorithms. 1996. pp 341-350. (TSPBIB)

[16] G.Kindervater, J.K.Lenstra & M.Savelsberg

Sequential and Parallel Local Search for the Time-Constrained Traveling Salesman Problem

TSPBIB

[17] T.Kohonen

Self Organizing Maps

Springer Verlag. 1997

[18] H.Lafoux

Une page d'histoire et *Fondements biologiques*

http://www-dapnia.cea.fr/Spp/Experiences/OPAL/bib/nnhistory.html

[19] Lawler, Lenstra, Rinooy Kan and Shmoy

The traveling salesman problem

John Wiley & sons. 1985.

[20] O.C.Martin, S.W.Otto

Combining Simulated Annealing with Local Search Heuristic

TSPBIB. 1994.

[21] Y.Matsuyama

Competitive Self Organization and Combinatorial Optimisation: Applications to the Traveling Salesman Problem

International Joint Conference on Neural Networks.1990. pp 819-824.

[22] Y.Matsuyama

Harmonic Competition: A Self-Organizing Multiple Criteria Optimisation

IEEE transactions on Neural Networks,Vol 7,N°3, May 1996. pp 652-668.

[23] Documentation PDF du logiciel **MatLab 5.3**.

[24] M.Al-Mulhem & T.Al-Maghrabi

An Efficient Convexe Elastic Net to Solve the Euclidean Traveling Salesman Problem

IEEE transactions on Systems, Men & Cybernetics, Part B.Vol28,N°4,August 1998. pp 618-620.

[25] J.C.Perez

La révolution des ordinateurs neuronaux

Hermes 1990.

[26] C.Peterson

Parallele Distributed Approach to Combinatorial Optimisation: Benchmark Studies on Traveling Salesman Problem

'Artificial Neural Networks, Concept and Theory' by Pankaj Mehra & Benjamin Wah, The IEEE Computer Society Press,1992.

[27] I.Peterson

The Traveling Monkey.

http://www.maa.org/mathland/mathland_7_7.html.1997

[28] Rolland & Steven

Final project

http://wayne.cs.nthu.edu.tw/~roland/fuzzy/

[29] Roland Fox, Huei-Juen Lin, Fang-Yi Jiang

Traveling Salesman Problem by Neural Approach

http://wayne.cs.nthu.edu.tw/~roland/nn/

[30] A.Vakhutinsky and B.L.Golden

A Hierarchical Strategy for Solving Traveling Salesman Problems Using Elastic Nets

Journal of Heuristics,1: 67-76 (1995). http://www.glue.umd.edu/~avakh/hier/hier.html

[31] J.Van Den Berg & J.Geselsschap

An Analyse of Various Elastic Algorithms

TSPBIB.

Source de certains articles :

[32] TSPBIB: http://www.ing.unlp.edu.ar/cetad/mos/TSPBIB_home.html

Source des problèmes test:

[33]TSPLIB:http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB.html

Site mirroire: ftp://softlib.rice.edu/pub/tsplib/tsp/

Le réseau de Hopfield

Le modèle de Hopfield fut présenté en 1982 et a contribué à relancer les recherches sur les réseaux de neurones. Ce modèle est basé sur celui des verres de Spin de Ising mais aussi sur le principe des mémoires associatives. Il est aussi appelé réseau associatif.

Le modèle de Ising: **[10]**

Il s'agit d'un modèle très simplifié des interactions magnétiques entre atomes. Chaque atome est décrit par la direction de son moment magnétique (spin) dont les valeurs sont limitées à 1 et -1. Chaque atome apportera sa contribution au champ magnétique ambiant et de ce fait influencer son voisin.

Soit : S_i le spin de l'atome i.
J_{ij} l'effet de l'atome j sur l'atome i.

La contribution h_i de l'atome i au champ magnétique est:

$$h_i = \sum_{i \neq j} J_{ij} S_i$$

Hypothèses: pas d'auto activation des neurones $J_{ii}=0$, symétrie des connexions $J_{ij}=J_{ji}$

L'énergie du système s'écrit: $E=\frac{-\sum_{i,j} J_{ij} S_i S_j}{2}$

E représente la somme des interactions entre les spins des atomes et l'effet de leur champ magnétique.

Description du réseau **[10]**

Le réseau de Hopfield est un réseau dont les neurones sont complètement interconnectés. Les neurones du réseau suivent le paradigme du neurone de McCulloch et Pittts dont les deux états possibles sont $V_i=1$ et -1 (ou 0 et 1) la fonction d'activation du neurone est la fonction Signe ou la fonction de Heaviside.

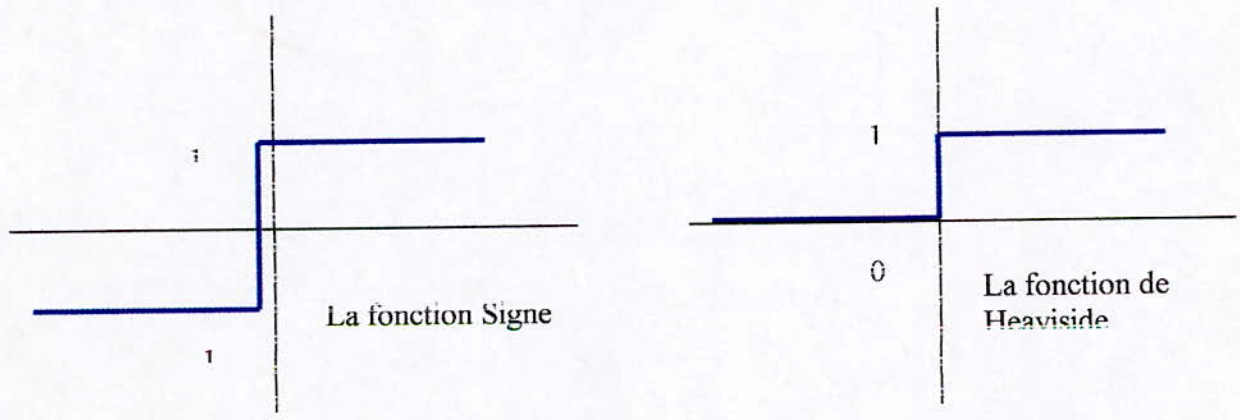


Figure I : Les fonctions d'activation du réseau de Hopfield

Le poids de la connexion entre deux neurones i et j est T_{ij}. L'entrée totale du neurone est donc $E_i=\sum_j T_{ij}V_j$

L'état du neurone à l'instant t+1 :

$V_i(t+1)=1$ si $E_i-\sum_j T_{ij}V_j>0$

$V_i(t+1)=0$ sinon

Concernant la dynamique de mise à jour des états, elle peut être faite aléatoirement, de façon asynchrone par rapport aux autres neurones. Elle peut aussi être séquentielle, c'est à dire que tous les neurones sont mis à jour les uns après les autres dans un ordre défini. Ou encore synchrone lorsque tous les neurones sont mis à jour simultanément (sur une machine parallèle).

L'état du réseau est caractérisé par l'état de ses N neurones V_i, il peut donc être représenté par une séquence (vecteur ou matrice) de N bits.

Apprentissage[10]

Le réseau de Hopfield est un réseau récurrent : un vecteur d'entrée est présenté au réseau qui produit un vecteur de sortie, ce vecteur de sortie est réinjecté dans le réseau comme vecteur d'entrée et ainsi de suite jusqu'à ce que le réseau converge vers un état stable. L'apprentissage ici est qualifié d'apprentissage par cœur car le réseau doit mémoriser un certain nombre d'états V^s (vecteur à N composants) ou *prototypes*. Ces états sont :

- Des états stables du réseau
- Des attracteurs vers lesquels va converger le réseau lorsqu'on lui présente des états proches des ces attracteurs.

Le processus d'apprentissage (ajustement des poids) se fait comme suit :

- On démarre avec un ensemble de connexions nulles (hypothèse de la table rase),

  - on force le réseau dans un état V^s (matrice initiale)
  - on examine tous les couples de neurones (i, j) et on incrémente T_{ij} de ΔT_{ij} suivant la règle de Hebb (renforcer les connexions des neurones actifs ou inactifs simultanément et affaiblir les connexions des neurones ayant des états différents)

V_i^s	V_j^s	ΔT_{ij}
1	1	+1
1	0	-1
0	1	-1
0	0	+1

En présentant successivement tous les exemples, on aboutit aux poids :

$T_{ij}=\sum_s V_i^s V_j^s$ pour les connexions qui sont renforcées

$T_{ij}=0$ pour les autres connexions.

Applications du réseau de Hopfield

Le réseau de Hopfield s'applique entre autres à la reconnaissance de caractères manuscrits et à l'optimisation combinatoire.

	ulysse22	simple
n° de la simula	Temps (sec)	Ecart à l'optim
1	4	3,53
2	7	43
3	2	80,5
4	8	1,63
5	5	41,56
6	4	38,8
7	9	6,13
8	8	1,82
Moyenne	5,875	27,12125

	ulysse16	simple
n° de la simula	Temps (sec)	Ecart à l'optim
1	4	15,96
2	3	4,91
3	7	11,21
4	4	1,97
5	5	18,17
6	5	21,96
7	3	27
8	2	18,2
Moyenne	4,125	14,9225

	eil51	simple
n° de la simula	Temps (sec)	Ecart à l'optim
1	20	27,34
2	12	29,46
3	13	17,82
4	11	25,24
5	14	18,21
6	16	30,76
7	20	37,66
8	16	21,37
Moyenne	15,25	25,9825

	eil101	simple
n° de la simula	Temps (sec)	Ecart à l'optim
1	85	24,66
2	96	16,22
3	78	25,71
4	98	7,21
5	103	14,58
6	80	13,8
7	46	20,36
8	110	7,55
Moyenne	87	16,26125

Résultats sur 10 instances de taille 120 générées aléatoirement:

Les paramètres : σ_0=60, ε_0=0.2, ε_{max}=0.1, r_0=0.1

Ecart à l'optimum %	Temps(sec)
17,49	213
20,8	217
22,02	226
27,89	178
19,74	198
21,03	144
40,52	150
17,49	315
27,87	174
30,68	224
24,553	203,9

	ulysse22	logarithme
n° de la simula	Temps (sec)	Ecart à l'optim
1	7	2,2
2	9	12,07
3	3	9,98
4	2	13,9
5	4	5,81
6	5	4,82
7	4	3,26
8	7	10,04
Moyenne	5,125	7,76

	ulysse16	logarithme
n° de la simula	Temps (sec)	Ecart à l'optim
1	2	8,59
2	3	24,76
3	3	13,54
4	5	21,73
5	3	12,66
6	3	9,52
7	1	19,15
8	4	15,07
Moyenne	3	15,6275

	eil51	logarithme
n° de la simula	Temps (sec)	Ecart à l'optim
1	12	7,22
2	8	13,03
3	11	13,22
4	14	26,39
5	8	14,12
6	14	23,64
7	9	23,55
8	12	29,44
Moyenne	11	18,82625

	eil101	logarithme
n° de la simula	Temps (sec)	Ecart à l'optim
1	88	8,56
2	95	13,8
3	103	14,09
4	100	13,74
5	90	18,17
6	93	13,51
7	95	12,73
8	101	20,77
Moyenne	95,625	14,42125

Résultats sur 10 instances de taille 120 générées aléatoirement:

Les paramètres : σ_0=1, ε_0=0.2, ε_{max}=0.1, r_0=0.1

Ecart à l'optimum %	temps
34,36	23
17,01	94
18,89	81
19,76	98
28,52	100
29,99	65
27,48	60
31,98	62
35,04	58
18,79	81
26,182	72,2

	ulysse22	racine
n° de la simulation	Temps (sec)	Ecart à l'optimum (%)
1	17	32,67
2	14	48,86
3	7	25,76
4	16	39,75
5	13	65,84
6	13	11,87
7	13	22,34
8	9	8,5
Moyenne	12,75	31,94875

	ulysse16	racine
n° de la simulation	Temps (sec)	Ecart à l'optimum (%)
1	3	26,1
2	2	9,19
3	8	43
4	2	29,53
5	2	11
6	11	34,04
7	4	84,07
8	6	40,24
Moyenne	4,75	34,64625

	eil51	racine
n° de la simulation	Temps (sec)	Ecart à l'optimum (%)
1	15	29,66
2	13	14,22
3	16	27,9
4	8	15,26
5	6	16,77
6	24	42,77
7	13	15,28
8	18	26,33
Moyenne	14,125	23,52375

	eil101	racine
n° de la simulation	Temps (sec)	Ecart à l'optimum (%)
1	120	14,58
2	72	18,09
3	96	19,22
4	71	15,57
5	101	19,23
6	101	14,02
7	105	13,56
8	111	32,66
Moyenne	97,125	18,36625

Résultats sur 10 instances de taille 120 générées aléatoirement:

Les paramètres : σ_0=1, ε_0=0.2, ε_{max}=0.1, r_0=0.1

n° de la simulation	Ecart à l'optimum %
1	25,32
2	17,95
3	17,4
4	28,08
5	23,04
6	6,87
7	19,27
8	20,97
9	33,72
10	27,24
Moyenne	21,986