MASTER EN SCIENCES INFORMATIQUES

Méthodes non-déterministes en optimisation combinatoire

Ledoyen, Pierre

Award date:
1991

Awarding institution:
Universite de Namur

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique

Rue Grandgagnage, 21
B-5000 NAMUR (Belgium)

Méthodes non-déterministes en optimisation combinatoire

Mémoire présenté en vue de l'obtention
du diplôme de Maître en Informatique

Pierre Ledoyen

Promoteur : J-L. Leclercq

Année académique 1990-1991

REMERCIEMENT

Avant d'entamer la rédaction de ce travail, nous tenons à remercier Monsieur J-P. Leclercq qui nous a suggéré cette étude et qui, par sa coopération, nous a permis de le mener à bien. Nous aimerions également remercier les personnes qui nous ont soutenus et aidés lors de l'élaboration de ce mémoire.

Facultés Universitaires Notre-Dame de la Paix
Institut d'Informatique
rue de Bruxelles 61, B-5000 NAMUR
Tél. 081-72.41.11
Télex 59222 facnam-b
Téléfax 081-23.03.91

Méthodes non-déterministes en optimisation combinatoire

Pierre Ledoyen

Résumé

Dans ce travail, nous avons étudié trois méthodes : le recuit simulé, la recherche tabou et les algorithmes génétiques. Toutes les trois sont assez récentes. Elles sont également toutes trois non-déterministes, c'est-à-dire qu'elles recourent à l'aléatoire. Elles ont été utilisées dans divers domaines, que ce soit pour des problèmes de placement, de conception de circuits intégrés, en physique, en biologie... Les deux applications qui ont servis de tests proviennent de l'optimisation combinatoire, à savoir, le problème du voyageur de commerce et celui de la coloration des sommets d'un graphe. Chaque méthode se voit consacrer un chapitre au cours duquel sont expliqués les points caractéristiques, puis la façon dont nous les avons implémentées, et enfin, les résultats des tests effectués. La comparaison de ces trois méthodes donne un avantage au recuit simulé. Quant à la recherche tabou, nous n'avons pas obtenu de résultats concluants. Enfin, malgré les résultats négatifs en coloration des sommets d'un graphe, les algorithmes génétiques apparaissent comme aussi efficace que le recuit simulé..

Abstract

In this work, we have introduced three methods : simulated annealing, tabu search and genetic algorithms. These methods, wich are recently emerged, are non-determinist because of they are based on randomization. They have been used in varied domains, for example : placement problem, intégrated circuits design, physics, biology ... The applications we experimented, i.e. the travelling salesman problem and the graph colouring problem, are induced by combinatorial optimization. Every chapter presents a method, its characteristics, the implementation and the results of the conducted tests. The comparison of the methods give advantage to simulated annealing. No conclusive results have been obtained for tabu search. Despite of negative results for graph colouring problem, genetic algorithms have been showed to be as efficient as simulated annealing.

Mémoire de maîtrise en Informatique
Septembre 1991
Promoteur : J-L. Leclercq

INTRODUCTION

Par le présent travail, nous voulons d'abord présenter trois méthodes non-déterministes, à savoir le recuit simulé, la recherche tabou et les algorithmes génétiques. Les points communs, ainsi que les différences entre ces trois méthodes, sont mis en évidence. Ensuite, nous les comparons en les appliquant à deux problèmes d'optimisation combinatoire.

Le cadre de ce travail, ainsi que les notions principales, sont décrits dans le premier chapitre. Nous expliquons également les deux applications traitées, à savoir le problème du voyageur de commerce et celui de la coloration des sommets d'un graphe. La fin du premier chapitre présente la recherche locale. Cette méthode sert, en effet, de base à deux des trois méthodes étudiées, à savoir le recuit simulé et la recherche tabou.

Le deuxième chapitre est consacré au recuit simulé. Nous y décrivons l'analogie qui est à la base de cette méthode. Il est alors possible d'en déduire un algorithme en pseudo-code. Ensuite, nous expliquons comment le recuit simulé peut être formalisé, ce qui permet d'obtenir un théorème de convergence asymptotique. Nous envisageons alors l'implémentation, ce qui nécessite l'explication de ce que représente un plan de refroidissement. Enfin, les résultats numériques concernant les deux applications choisies sont décrits.

La recherche tabou est présentée dans le deuxième chapitre, selon le même cannevas : description de la méthode, algorithme en pseudo-code, implémentation et résultats.

Le troisième chapitre, quant à lui, est consacré aux algorithmes génétiques. Tout comme le recuit simulé, cette méthode provient d'une analogie avec un phénomène naturel, à savoir, l'évolution.

Nous pouvons alors conclure ce travail en rappelant brièvement les caractéristiques de chaque méthode, et en évoquant les résultats obtenus.

CHAPITRE 1 : OPTIMISATION COMBINATOIRE

1.1. Introduction

" Résoudre un problème d'optimisation combinatoire revient à trouver la solution " la meilleure " ou " optimale " parmi un nombre fini ou infini dénombrable de solutions " (Papadimitriou [14]).

Ces dernières décennies, de nombreux problèmes de ce type ont émergé dans divers domaines : informatique, management, engineering ... De nombreux efforts ont été apportés pour mettre au point des méthodes de résolution. Que ce soit la programmation linéaire, non-linéaire, à variable entière ou dynamique, toutes ces méthodes ont été fortement développées dans ce but. La théorie de la complexité a mis à jour certaines limites. Dès lors, pour résoudre des problèmes NP-complets de taille assez grande, l'alternative est en général la suivante. Soit nous utilisons une méthode qui nous garantit de trouver la solution optimale au risque de prendre un temps de calcul assez long, soit nous essayons d'une manière ou d'une autre d'accélérer cette convergence au risque de perdre au niveau de la qualité de la solution obtenue.

Ce choix se marque à deux niveaux : d'une part, certaines méthodes ont recours à l'aléatoire et forment la classe des méthodes non déterministes, tandis que les autres sont appelées exactes ou déterministes. D'autre part, pour certaines méthodes, la convergence vers la solution optimale est prouvée : il s'agit des algorithmes, les autres étant des heuristiques.

Insistons sur le fait que ces deux caractéristiques sont indépendantes et donc qu'il existe des algorithmes non-déterministes tout comme il existe des heuristiques déterministes. C'est la classe des méthodes non-déterministes qui nous intéresse dans le cadre de ce travail et les trois méthodes dont nous allons parler en font partie.

Un autre classement se base sur le fait qu'une méthode est soit adaptée à un problème et ne peut servir dans d'autres cas, soit elle est générale et peut résoudre différents types de problèmes. Il s'agit respectivement des méthodes spécifiques et génériques. Les trois méthodes traitées sont génériques Il s'agit du recuit simulé, de la recherche tabou et des algorithmes génétiques.

Nous allons à présent formaliser la notion de problème d'optimisation combinatoire. Il importe de souligner ici que nous nous limitons à des ensembles finis de solutions et que nous supposons que la qualité d'une solution est quantifiable. Après cela, nous expliquerons les deux applications que nous avons implémentées pour la comparaison des différentes méthodes. Il s'agit

du problème du voyageur de commerce et celui de la coloration de graphe. Nous finirons par décrire les méthodes de recherches locales qui nous seront utiles dans la suite.

1.2. Problèmes d'optimisation combinatoire

Définition 1.1.
Une instantiation d'un problème d'optimisation combinatoire est caractérisée par une paire (S,f) où S, l'espace des solutions, est l'ensemble fini de toutes les solutions possibles et où f, la fonction d'évaluation, est une fonction définie sur S et à valeurs dans la droite réelle.

Définition 1.2.
Un problème d'optimisation combinatoire est soit un problème de maximisation soit de minimisation et est caractérisé par un ensemble d'instantiations.

Définition 1.3.
Une solution optimale d'un problème d'optimisation combinatoire est un élément i_{opt} de S. Dans le cas d'une minimisation, le problème est de trouver une solution i_{opt} appartenant à S et telle que
$f(i_{opt}) \leq f(i)$, pour tout i appartenant à S.
Dans le cas d'une maximisation, la condition ci-dessus se transforme en
$f(i_{opt}) \geq f(i)$, pour tout i appartenant à S.

La distinction entre un problème d'optimisation combinatoire et ses instantiations est celle qui existe entre, par exemple, le problème du voyageur de commerce en général (défini au point 1.3.1) et un problème particulier entre N villes fixées, c'est-à-dire dont les données initiales sont connues : les N villes et les distances entre elles.

Une telle solution i_{opt} est dénommée de plusieurs façons suivant les cas : solution globalement optimale, solution optimale ou optimum, soit solution minimale ou minimum, soit solution maximale ou maximum. S_{opt} dénote l'ensemble des solutions optimales. Dans la suite de ce travail et notamment lors de l'explication des méthodes de résolution, nous supposerons, sauf mention explicite du contraire, que le problème est une minimisation. Cela peut se faire sans perdre l'aspect général de cet exposé car une maximisation revient à une minimisation en changeant simplement le signe de la fonction d'évaluation.

Souvent, les solutions d'un problème d'optimisation doivent aussi satisfaire à un ensemble de contraintes. Il peut être utile pour résoudre ce type de problèmes de partitionner l'ensemble des solutions en deux parties : les solutions qui vérifient ces contraintes, autrement dit les solutions

réalisables, et celles qui ne les vérifient pas, c'est-à-dire les solutions non-réalisables. Cela amène à étendre la définition d'une instantiation d'un problème d'optimisation.

Définition 1.4.
Une instantiation d'un problème d'optimisation combinatoire est caractérisée par un triplet (S,S',f) où S, l'espace des solutions, est l'ensemble de toutes les solutions, S' celui des solutions réalisables et où f, la fonction d'évaluation, est une fonction définie sur S et à valeurs dans la droite réelle. La définition d'une solution est obtenue en remplaçant, dans le cas d'une minimisation, la condition par
$f(i_{opt}) \leq f(i)$, pour tout i appartenant à S',
et, dans le cas d'une maximisation, par
$f(i_{opt}) \geq f(i)$, pour tout i appartenant à S'.

Voyons à présent en quoi consistent les deux problèmes d'optimisation combinatoire qui nous permettront de comparer les méthodes étudiées.

1.3. Applications

Nous allons décrire les deux applications, à savoir le voyageur de commerce et la coloration des sommets d'un graphe, qui seront utilisées pour réaliser l'implémentation des diverses méthodes.

1.3.1. Le voyageur de commerce

C'est un exemple classique de problème d'optimisation combinatoire car il est simple à expliquer sans pour autant être facile à résoudre. L'énoncé est le suivant :

" Soit une liste de N villes et un moyen de calculer la distance entre chaque couple de villes; le but est d'établir le trajet du voyageur de commerce tel qu'il passe une et une seule fois par chaque ville et qu'il revienne à son point de départ tout en minimisant la distance parcourue. "

La donnée de N villes et d'une matrice d de taille N*N, tel que d (p,q) représente la distance entre les villes p et q, permet de caractériser une instantiation d'un problème d'optimisation combinatoire. En effet, S, l'ensemble des solutions, est constitué de l'ensemble des permutations des N villes et la fonction f calcule la distance d'une solution à partir de la matrice d. Remarquons que, dans le cas du voyageur de commerce, les mots trajet, tour, circuit et solution ont la même signification.

Notons aussi qu'il suffit d'un moyen de calculer la distance entre deux villes; la matrice en

tant que telle n'est pas nécessaire. Par exemple, si la distance euclidienne est utilisée, les coordonnées des villes permettent ce calcul des distances.

Un des aspects intéressants du voyageur de commerce réside en la possibilité d'estimer la distance d'un tour minimal. Cela permet de tester les méthodes.

Par expérience, nous savons que pour N suffisamment grand, la distance minimale tend vers $0{,}749 \sqrt{N} \sqrt{A}$ où A est l'aire de la région où se trouvent les villes.[3]

1.3.2. La coloration des sommets d'un graphe

Tout comme le voyageur de commerce, la coloration des sommets d'un graphe intervient dans plusieurs applications comme par exemple la composition des horaires.

Notons que, par facilité, nous parlerons dans la suite, de coloration de graphe, sans préciser qu'il s'agit des sommets. Il existe en effet aussi des problèmes de coloration des arêtes d'un graphe ou des faces d'un graphe planaire. Cependant, ces points ne seront pas envisagés.

Pour un énoncé clair, nous nous situons en théorie des graphes.

" Soit G= (X,V) un graphe tel que X est l'ensemble des sommets et V celui des arêtes. Colorer un graphe consiste à attribuer une couleur à chaque sommet, de sorte que deux sommets reliés par une arête aient une couleur différente. Le but est de trouver une coloration minimale, c'est-à-dire celle qui utilise le moins de couleurs possible. "

Contrairement au voyageur de commerce, le nombre minimal de couleurs est difficile à estimer. Il est clair que ce nombre est inférieur au degré du graphe augmenté de un. Il existe aussi des bornes inférieures, mais en général, elles n'apportent pas suffisamment d'informations.

Pour ce qui est d'une instantiation, l'ensemble des solutions S peut être constitué de toutes les colorations possibles. Comme nous le verrons plus loin, il est intéressant d'y ajouter des solutions non-réalisables. En effet, une coloration revient en fait à partitionner les sommets en ensembles indépendants et d'attribuer une couleur à chacun de ces ensembles. L'ensemble S devient, dès lors, l'ensemble des partitions des sommets. Une solution est réalisable si la partition est une coloration, c'est-à-dire si la partition procure des ensembles indépendants.

Le choix de la fonction f n'est pas aussi évident que pour le voyageur de commerce. Il est clair que le nombre de couleurs utilisées ainsi que le nombre d'arêtes monochromes, c'est-à-dire qui relient deux sommets de la même couleur, interviennent dans ce choix.

Signalons également qu'il existe une manière différente de procéder. Elle consiste à se

donner au départ le nombre de couleurs permises et d'essayer de trouver une coloration qui y correspond. La fonction d'évaluation est alors égale au nombre d'arêtes monochromes et le but est d'arriver à l'annuler. Insistons sur le fait qu'il ne s'agit plus tout à fait d'un problème de minimisation, car le nombre de couleurs est donné, mais d'un problème de satisfaisabilité. La question devient un effet : " est-il possible de trouver une coloration des sommets avec k couleurs, k étant donné ? "

Contre cette manière de procéder, une réaction pourrait être de reprocher de devoir déterminer le nombre de couleurs à l'avance. Cependant, d'une part, comme nous le constaterons par après, et ce même dans le cas de l'optimisation, une approximation à priori de ce nombre est nécessaire. D'autre part, rien n'empêche, en fonction de la réussite ou de l'échec, de recommencer en modifiant la valeur de k.

Ces deux optiques différentes seront appliquées et les résultats obtenus nous permettront de voir si l'une des deux est plus performante que l'autre.

Voyons à présent en quoi consistent les méthodes de recherches locales qui servent de base au recuit simulé et à la recherche tabou. Nous expliquerons également comment peut s'appliquer la recherche locale aux deux problèmes présentés ci-dessus.

1.4. La recherche locale

1.4.1. Stratégie

Cette méthode est une heuristique basée sur la stratégie d'amélioration itérative. Celle-ci consiste à partir d'une solution initiale, à appliquer un opérateur de changement jusqu'à la découverte d'une solution qui améliore la fonction d'évaluation. Cette dernière remplace alors la solution courante et le processus recommence jusqu'à ce qu'aucune amélioration ne soit trouvée. Il y a ainsi parcours de l'ensemble des solutions à la recherche de l'optimum.

La stratégie " diviser pour règner " consiste par contre à diviser le problème en sous-problèmes plus facilement traitables et de les résoudre séparément. Les solutions des diverses parties sont alors remises ensemble pour former une solution du problème initial.

Dans ce cas, nous travaillons donc sur des " morceaux " de solutions, tandis que dans le premier cas, seules des solutions globales sont envisagées.

1.4.2. Voisinage

Tout en appliquant la stratégie d'amélioration itérative, la caractéristique de la recherche locale est d'explorer le voisinage de la solution courante. Voyons ce que cette notion de voisinage recouvre.

Définition 1.5.
Le voisinage S_i d'une solution i de S est un ensemble d'éléments de S considérés comme directement accessibles à partir de i.

Le mot " considérés " dans la définition provient du fait que, pour un problème donné, il y a plusieurs façons de déterminer le voisinage d'une solution.

Voici un exemple : en coloration de graphe, une solution voisine d'une solution i est obtenue en changeant un sommet de couleur et le voisinage de i est l'ensemble de ces solutions obtenues en modifiant la couleur d'un sommet. Nous aurions très bien pu décider qu'une solution voisine est obtenue en changeant la couleur de deux sommets.

A ce voisinage est associé un mécanisme de génération : c'est un moyen de sélectionner une solution j parmi les solutions voisines de i. Notons que nous ne considérons que des voisinages symétriques, c'est-à-dire tel que si i appartient au voisinage de j, alors j appartient au voisinage de i. Nous pouvons dès lors parler de deux solutions voisines sans ajouter qui est voisin de l'autre.

1.4.3. Algorithme

Le principe de la recherche locale est simple. Cet algorithme progresse en explorant le voisinage de la solution courante i afin de l'améliorer. En fait, à chaque itération, une solution j voisine de i est générée. Si elle améliore la fonction d'évaluation, cette solution j devient courante, sinon l'algorithme continue avec i. Lorsqu'aucune amélioration ne peut être obtenue de cette manière, l'algorithme se termine. La solution finale est donc meilleure que toutes celles de son voisinage. Cet algorithme débute par le choix d'une solution initiale. En voici une version en pseudo-pascal.

```
Procedure recherche locale;
begin
   initialisation(i);
            (* choix d'une solution initiale *)
   repeat
            génération(j dans S_i);
                  (* application du mécanisme de génération *)
            if f(j) < f(i) then i:=j
                  (* si j améliore la fonction d'évaluation, j remplace
                     la solution courante *)
   until f(j) ≥ f(i), pour tout j dans S_i
            (* aucune amélioration n'est possible pour l'algorithme *)
end;
```

1.4.4. Point fort et point faible

Pour caractériser les qualités et les défauts, introduisons la notion d'optimum local.

Définition 1.6.:
Soit (S, f) une instance d'un problème d'optimisation. Une solution i est localement optimale ou un optimum local si et seulement si
$f(j) \geq f(i)$, pour tout j dans S_i.

C'est là le point faible de la recherche locale. Elle fournit un optimum local mais rien ne permet en général de le comparer à l'optimum recherché. De plus, la solution finale d'une recherche locale dépend fortement de la solution initiale et dans la plupart des cas, il n'est pas possible de trouver une règle ou même des indications pour effectuer un choix approprié de cette solution initiale.

Il apparaît clairement que l'efficacité de la recherche locale dépend du choix de la solution initiale et du choix du voisinage.

Le grand avantage de cette méthode est qu'elle peut s'appliquer facilement à beaucoup de problèmes d'optimisation combinatoire. Il suffit de spécifier les solutions, une fonction d'évaluation et une structure de voisinage (le mécanisme de génération découle de cette structure).

Plusieurs possibilités existent pour éviter le point faible de la recherche locale tout en gardant

son principe de base et ses qualités. En voici trois :

- exécuter plusieurs fois la recherche locale avec différentes solutions initiales. Il est clair que si cet algorithme est exécuté un nombre suffisant de fois, il finit par trouver la solution optimale. Il suffit de prendre toutes les solutions comme solution initiale. Il est tout aussi clair que cette possibilité est à rejeter pour un problème complexe.
- prendre une structure de voisinage plus complexe en introduisant certaines connaissances relatives au problème traité. Ainsi une plus grande partie de l'espace des solutions est visité. Cependant, le risque est alors grand que l'algorithme devienne adapté à un problème particulier et perde donc sa généralité.
- accepter dans certaines limites des détériorations de la fonction d'évaluation pour essayer d'éviter de rester bloqué dans un optimum local.

Le recuit simulé et la recherche tabou suivent cette troisième possibilité. Le principe des algorithmes génétiques est assez éloigné de cette recherche locale. Dans certains cas, nous pourrions dire que le principe de ces algorithmes génétiques est d'effectuer la recherche locale sur plusieurs solutions en parallèles avec à chaque étape un échange d'informations entre ces solutions.

1.4.5. Implémentation

Le but de ce paragraphe n'est pas de réaliser l'implémentation de la recherche locale pour la tester, mais d'expliquer quelques points qui seront utiles dans le cadre du recuit simulé, de la recherche tabou et, dans une moindre mesure, des algorithmes génétiques.

Pour chaque application, nous allons discuter tour à tour de la façon de représenter les solutions, d'initialisation, du mécanisme de génération et de la fonction d'évaluation.

1. Le voyageur de commerce

Une instantiation du problème est caractérisée par le nombre N de villes, la matrice D des distances entre chaque couple de villes. Nous supposons donc avoir un tableau à deux dimensions de N^2 éléments qui représente cette matrice, tandis que N est une constante.

Les villes sont numérotées de 1 à N et un tour est représenté par les villes dans leur ordre de visite, c'est-à-dire une suite de nombres de N éléments. Une solution est dès lors représentée par un tableau à une dimension.

L'initialisation peut s'opérer de diverses manières. Une première est d'arranger au hasard le circuit en utilisant un générateur aléatoire de nombres. Il suffit alors de veiller à ne pas prendre deux fois la même ville.

Une deuxième consiste à choisir d'abord une ville, puis de prendre chaque fois la ville la plus proche. Suivant la ville de départ, les tours obtenus ainsi peuvent être fort différents.

La troisième façon ressemble à la précédente, si ce n'est que le choix de la ville la plus proche est limité à un sous-ensemble. Par exemple si N=50, le choix est restreint à dix villes. Ces différentes possibilités seront utilisées par les différentes méthodes.

Pour la notion de voisinage, nous utilisons le k-change de LIN, noté aussi k-opt.[1] Ce mécanisme de génération consiste à obtenir un nouveau tour en supprimant k-arêtes du tour initial et en les remplaçant par k autres, afin de réobtenir un circuit.

Voyons ce que cela représente si k=2, cas que nous utiliserons plus loin. La démarche est la suivante, illustrée par le figure 1.1.

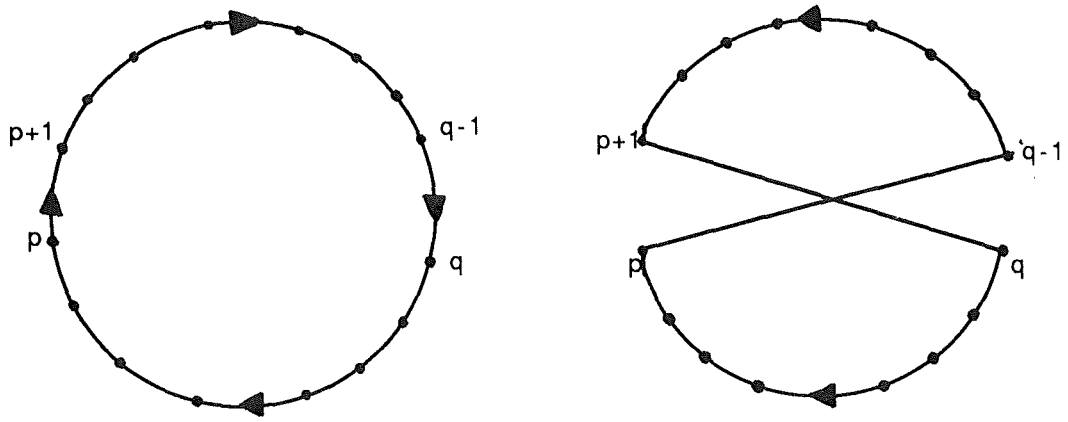

Figure 1.1. : Le 2-change de LIN.

Soit le tour initial 1, 2, ..., N. Soit p,q deux villes telles que q ne soit pas visitée juste après p, c'est-à-dire q≠p+1 " Pour la clarté de l'explication, nous supposons p < q.

Le tour final qui résulte du 2-change de LIN est alors 1, 2, ...p, q-1, q-2, ... p+1, q, q+1 ... N.

Cela revient à aller de p à la ville qui précédait q puis d'inverser le parcours jusqu'à arriver à p+1, c'est-à-dire la ville qui suivait p. De là, le trajet repart vers q et reprend le même ordre qu'au départ.

Dès lors, le voisinage de i est constitué de tous les tours obtenus en appliquant le 2-change

de LIN et le mécanisme de génération consiste à choisir au hasard deux villes p et q telles que q ne succède pas à p.

La fonction d'évaluation représente la distance totale du tour. Cependant, par souci d'optimisation, le calcul complet de cette fonction est effectué le moins souvent possible. En général, lorsqu'une nouvelle solution est générée, il est possible de ne calculer que la variation de la fonction, notée Δf. Ce Δf correspond donc à la différence de la fonction d'évaluation entre la solution proposée et la solution courante. S'il est négatif, cela entraîne un changement de solution courante.

Dans le cadre de l'implémentation, nous avons utilisé la distance euclidienne pour le calcul de la fonction d'évaluation.

Les différentes villes sont donc caractérisées par leurs coordonnées. De plus, nous avons choisi de situer ces villes dans le carré unité, c'est-à-dire un carré dont les côtés sont de longueur unitaire.

Cela a comme conséquence une certaine symétrie dans le problème. En effet, vu ce choix, passer d'une ville à l'autre " coûte " autant que de faire le chemin inverse. Insistons sur le fait que cette symétrie ne provient pas du problème du voyageur de commerce en général, mais du cas particulier que nous avons décidé de traiter.

Le calcul de la variation Δf s'en trouve facilité, car le fait d'inverser une partie du parcours ne change en rien son coût. Pour obtenir la fonction d'évaluation du nouveau circuit, il suffit de soustraire, à la fonction de l'ancien tour, la distance de p à p+1 et de q-1 à q puis d'y ajouter la distance de p à q-1 et de p+1 à q.

La nouvelle valeur est donc obtenue par le calcul de quatre distances plutôt que de N.

2. La coloration de graphe

Une instantiation du problème est caractérisée par le nombre N de sommets et la matrice d'adjacence du graphe. De la même manière que pour le voyageur de commerce, nous numérotons les sommets de 1 à N, ainsi que les couleurs de 1 à Δ. Suivant l'optique choisie comme expliqué au point 1.3.2. , Δ représente soit une borne supérieure du nombre minimal de couleurs, soit le nombre maximum de couleurs permises.

Une solution est dès lors représentée par un tableau de N éléments qui prennent des valeurs comprises entre 1 et Δ. Le fait que le $i^{ème}$ élément ait la valeur j correspond au fait que le sommet i possède la couleur j.

Pour ce qui est de l'initialisation, une couleur est tirée au hasard pour chaque sommet. Nous aurions pu essayer de minimiser le nombre d'arêtes monochromes introduites lors de cette opération, mais nous n'avons pas retenu cette possibilité.

Le voisinage d'une solution a été expliqué comme exemple précédemment. Rappelons qu'il s'agit de l'ensemble des partitions obtenues en changeant un sommet de couleur. Le mécanisme de génération revient alors à choisir deux nombres qui correspondent l'un à un sommet, l'autre à une couleur, tout en s'assurant que cette couleur ne soit pas celle du sommet choisi.

L'ajout des solutions non-réalisables dans la définition d'une instantiation trouve sa justification ici.

En effet, cela permet de définir le mécanisme de génération comme ci-dessus, car il suffit de s'assurer que la nouvelle solution soit une partition. Sans cet ajout, il faudrait vérifier que cette nouvelle solution soit une coloration.

Il existe des mécanismes de génération qui vérifient cette condition. Les solutions non-réalisables sont alors inutiles. C'est l'approche utilisée notamment par Morgenstein et Shapiro [1] et leur concept de " Kempe Chains " . Nous avons cependant opté pour la première façon de procéder.

Le choix de la fonction d'évaluation dépend de l'approche suivie. Si le nombre de couleurs est donné et qu'il s'agit en fait de satisfaisabilité, il est clair qu'il faut annuler le nombre d'arêtes monochromes. Il suffit dès lors de poser f égal à ce nombre.

Par contre, si nous considérons le problème de minimisation tel quel, il faut que le nombre de couleurs utilisées intervienne également. Le choix de la fonction d'évaluation devient dès lors plus complexe. Il s'agit en effet de savoir quelle importance donner à chacun de ces deux éléments.

Différents poids peuvent être attribués en fonction du nombre de couleurs utilisées et du nombre d'arêtes monochromes. Le choix de ces poids est assez complexe. Il ne faut en effet pas trop pénaliser les arêtes non voulues, sinon la recherche risque de s'arrêter dès qu'une coloration est obtenue. D'autre part, de trop faibles poids conduiront à une solution où le nombre de couleurs a été minimisé, mais qui n'est pas une coloration.

Deux approches différentes ont été utilisées. Avant de les expliquer, introduisons quelques notations. Rappelons que Δ est le nombre maximum de couleurs permises.

Trouver une coloration revient donc à trouver une partition en ensembles indépendants.

Notons $V_1, V_2, ..., V_\Delta$ les ensembles formant une partition et E_i, $1 \leq i \leq \Delta$, l'ensemble des arêtes monochromes, c'est-à-dire les arêtes dont les deux extrémités appartiennent à V_i.

Une première idée est d'attribuer une pénalité λ à chaque arête monochrome. La fonction d'évaluation est alors égale au nombre de couleurs utilisées auquel est ajouté le nombre d'arêtes monochromes, multiplié par le facteur λ. Mais ce choix n'est pas assez fin et ne permet pas une bonne minimisation. Les solutions obtenues utilisent beaucoup de couleurs sauf si λ est égal à un, car dans ce cas, les solutions ne sont plus des colorations.

Pour améliorer le processus, nous avons agit de la manière suivante : Nous nous fixons un nombre Δ' de couleurs espérées, un peu comme une borne inférieure. Chaque sommet qui prend une couleur entre $\Delta'+1$ et Δ est " pénalisé " . En fait, nous utilisons un ensemble de poids $W = (w_1, w_2, ..., w_\Delta)$ tel que $w_i = 1$, $1 \leq i \leq \Delta'$ et $w_i = 2*i$ si $\Delta' < i \leq \Delta$. La première partie de la fonction d'évaluation consiste, dès lors, à multiplier les sommets par le poids correspondant à leur couleur puis à les additionner. Vu le choix des w_i, les Δ' premières couleurs sont favorisées par rapport aux autres. De plus, le choix des w_i croissants est utile pour permettre la minimisation du nombre de couleurs utilisées.

La deuxième partie de la fonction d'évaluation reste la même, c'est-à-dire le nombre d'arêtes monochromes multiplié par le facteur λ.

Signalons que λ doit être plus grand que $2*\Delta$, sinon il y a un risque d'obtenir des solutions qui ne sont pas des colorations.

Nous avons également utilisé une toute autre façon de définir la fonction d'évaluation. Elle provient de Aarts et Korst [1] . Ils se basent sur un théorème qui démontre que si les poids $w_1, ...w_\Delta$ vérifient certaines relations, résoudre le problème de graphe est équivalent à trouver une partition P en ensembles indépendants telle que

$$g(P)=\sum_{i=1}^{\Delta} w_i \mid V_i \mid$$

est maximal.

Notons que cette fonction revient à multiplier les sommets par le poids de leur couleur puis de les additionner.

A partir de ce théorème, en relachant les hypothèses, ils choisissent les poids tels que $w_{j+1} < 2*w_j - w_1$, $j=1, ..., \Delta-1$.

Dès lors, la fonction d'évaluation revient à prendre la fonction g définie ci-dessus et à soustraire la somme des arêtes monochromes qui ont été multipliées par un facteur λ et par le

poids de la couleur correspondante. Cela donne

$$f(P)=\sum_{i=1}^{\Delta} w_i \, (\, | V_i | - \lambda \, | E_i | \,)$$

Le but est dès lors de maximiser cette fonction.

Ces deux façons de définir la fonction d'évaluation seront testées avec les méthodes étudiées. Au-delà de cette alternative, il est intéressant de constater la relative complexité du problème de coloration de graphe par rapport au voyageur de commerce. C'est pour cela que nous avons choisi ces deux applications. La seconde sert en quelque sorte d'étalon pour comparer les méthodes, tandis que la première essaye de voir comment il est possible d'utiliser ces méthodes dans des cas moins favorables.

CHAPITRE 2 : LE RECUIT SIMULE

2.1. Introduction

La méthode du recuit simulé provient d'un analogie entre le processus bien connu en physique du solide, le recuit, et la résolution de problèmes en optimisation combinatoire. C'est Kirkpatrick, Gelatt et Vecchi [11] d'une part, et Cerny [6] d'autre part qui, de façon indépendante, ont introduit cette méthode au début des années 80.

Dans un premier temps, nous allons décrire le processus physique du recuit et l'algorithme de Métropolis qui permet une simulation d'une partie de ce processus. Après cela, nous introduirons la méthode du recuit simulé en expliquant l'analogie utilisée. Une fois cet aperçu intuitif terminé, nous décrirons brièvement comment modéliser mathématiquement le recuit, ce qui permet d'obtenir quelques résultats théoriques dont ceux relatifs à la convergence asymptotique de la méthode.

Nous pouvons ensuite passer à l'implémentation. Pour cela, la notion de plan de refroidissement est étudiée, c'est-à-dire comment arriver à une convergence vers une bonne solution en un temps fini. Pour terminer, deux applications concrètes illustreront l'utilisation du recuit simulé : le voyageur de commerce et la coloration de graphe.

2.2. Le recuit et l'algorithme de Métropolis

Le processus du recuit consiste à obtenir les états de basse énergie d'un solide qui se trouve dans un bain chaud. Deux étapes le caractérisent :

- accroître la température du bain jusqu'à ce que le solide fonde. A cet état, les molécules du solide s'arrangent entre elles de manière aléatoire.
- décroître très lentement la température du bain pour atteindre un " état de base " du solide. A cet état, les molécules du solide sont arrangées de manière très structurée.

Ce processus est délicat et pour obtenir cet état de base, il faut que la température maximale atteinte soit suffisamment élevée et que le refroidissement se fasse avec précaution. Sinon, l'état obtenu du solide est en fait un état métastable. C'est ce qui se passe dans la trempe, processus inverse du recuit, qui consiste à abaisser brutalement la température du bain.

La phase de refroidissement doit être très lente pour qu'à chaque valeur T de la température, le solide puisse atteindre un équilibre thermique. Cet équilibre est caractérisé par la distribution de Boltzmann qui donne la probabilité que le solide soit dans un état i d'énergie E_i, à la température T, c'est-à-dire

$$Pr\{X = i\} = 1/Z(T) * \exp(- E_i / K_b*T)$$

où X est une variable stochastique qui décrit l'état courant du solide
K_b est la constante de Boltzmann

Z(T) est un facteur de normalisation, souvent appelé fonction de partition.

Lorsque T approche de la valeur nulle, seuls les états de basse énergie ont une probabilité non nulle d'apparaître.

Dès 1953, l'algorithme de Métropolis a été proposé pour simuler la recherche d'un équilibre thermique par le solide dans un bain chaud à température fixe. Il est basé sur des méthodes Monte Carlo. Il consiste à générer une suite d'états du solide de la façon suivante :

" soit l'état courant i d'énergie E_i; appliquons un mécanisme de perturbation qui par une petite modification donne un état j d'énergie E_j. La probabilité que l'état j soit accepté comme état courant est égale à 1 si il y a eu diminution d'énergie c'est-à-dire si $\Delta E = E_j - E_i$ est négatif, et est égale à l'exponentielle de $(-\Delta E / K_b*T)$ sinon, ce qui peut s'écrire

$$P_t\{\text{accept j}\} = \begin{cases} 1 & \text{si } \Delta E \leq 0 \\ \exp(-\Delta E / K_b*T) & \text{sinon."} \end{cases}$$

Cette règle d'acceptation est connue sous le nom de critère de Métropolis.

Pour simuler tout le processus du recuit, une idée est d'itérer l'algorithme de Métropolis pour des valeurs de T décroissantes. C'est cette idée qui est à la base du recuit simulé.

2.3. Le recuit simulé

A présent, expliquons l'analogie entre un solide dans un bain chaud et un problème d'optimisation combinatoire. Ceci nous permettra de simuler le recuit pour résoudre des problèmes tels le voyageur de commerce et la coloration de graphe.

Cette analogie s'effectue comme suit :

- chaque solution du problème d'optimisation est assimilée à un état du solide
- la fonction d'évaluation est assimilée à l'énergie
- un paramètre t, appelé paramètre de contrôle, est introduit pour jouer le rôle de

la température.

Dès lors, réduire lentement la température du bain afin d'atteindre un état de base du solide, état caractérisé par une faible énergie, revient à réduire lentement le paramètre de contrôle afin d'atteindre une solution optimale du problème, solution caractérisée par une valeur minimale de la fonction d'évaluation. C'est là le principe du recuit simulé.

Voyons maintenant ce que l'algorithme de Métropolis apporte au recuit simulé. En fait, à chaque valeur du paramètre de contrôle, il va nous permettre de générer une séquence de solutions pour atteindre un " équilibre " . L'algorithme du recuit simulé est ainsi construit comme l'itération de l'algorithme de Métropolis pour des valeurs décroissantes du paramètre de contrôle. Pour cela, nous avons besoin d'un mécanisme de génération qui, à partir d'une solution courante, nous donne une autre solution proche de la première : c'est la notion de voisinage que nous avons vue avec la recherche locale au premier chapitre. De plus, si i est la solution courante et j la solution voisine, le critère de Métropolis devient

$$\Pr\{\text{accept } j\} = \begin{cases} 1 & \text{si } \Delta f = f(j) - f(i) \leq 0 \\ \exp(-\Delta f / t) & \text{sinon} \end{cases}$$

Ces éléments sont les points principaux du recuit simulé et nous permettent d'en construire l'algorithme.

2.4. Algorithme

Traduire la démarche explicitée ci-dessus en un algorithme, tel est le but de ce paragraphe.

Une transition est définie comme l'action qui transforme une solution courante, c'est-à-dire l'application du mécanisme de génération suivi de l'application du critère d'acceptation. Nous pouvons à présent décrire l'algorithme du recuit simulé en pseudo-pascal. Soient t_k la valeur du paramètre de contrôle et L_k le nombre de transitions générées à la k ^{ème} itération de l'algorithme de Métropolis, soient i et j des solutions.

```
procedure recuit_simulé;
begin
        initialisation(i,t0,L0);
                (* choix d'une solution initiale ainsi que des valeurs initiales des
                   paramètres t et L *)
        k:=0;
        repeat
                for l:=1 to Lk do
                        (*boucle qui effectue le nombre de transitions voulues *)
                begin
                        génération(j dans Si);
                                (* application du mécanisme de génération *)
                        calcul_de_df;
                                (* calcul de la variation Δf *)
                        if df ≤ 0 then i:=j
                                else
                                if exp(-df/tk) > random then i:=j
                                        (* application du critère d'acceptation *)
                end;
                k:=k+1;
                calcul_longueur(Lk);
                calcul_paramètre(ck)
                        (* les paramètres doivent permettre d'atteindre un équilibre à
                           chaque itération *)
        until critère_d'arrêt
                (* décision d'arrêter ou non l'algorithme *)
end;
```

Signalons tout d'abord que l'initialisation de i, le mécanisme de génération, le calcul de la variation de Δf ainsi que le choix de la fonction d'évaluation s'effectuent de la même manière que celle décrite pour la recherche locale.

Ce qui est nouveau par contre, ce sont la gestion du paramètre de contrôle et du nombre de transitions générées, le critère d'arrêt ainsi que la façon d'accepter une nouvelle solution. Notons à ce propos, que, comme indiqué dans l'algorithme, le critère d'acceptation est utilisé en comparant la probabilité d'accepter j à un nombre aléatoire.

A ce stade, nous pouvons constater que le comportement de cet algorithme dépend surtout de deux paramètres : le paramètre de contrôle t et le nombre de transitions générées L.

Insistons sur le fait que ces deux paramètres ne sont pas indépendants l'un de l'autre. En effet, la variation de t correspond à la façon dont la température est abaissée. Les valeurs de t simulent les différents paliers de température où le solide recherche l'équilibre thermique. Or L est le nombre de transitions à générer pour atteindre cet équilibre.

Dès lors, plus les paliers de température sont espacés, ce qui correspond à une décroissance rapide du paramètre t, plus grand doit être le nombre de transitions générées, sans quoi le risque d'obtenir un état métastable (c'est-à-dire une solution non-optimale) s'agrandit.

Inversément, plus les valeurs de t sont rapprochées, plus vite est atteint l'équilibre, donc moins de transitions sont nécessaires. Nous verrons lors de l'implémentation, comment nous avons tenu compte de cette relation entre les deux paramètres.

Certes, la façon de choisir la solution initiale et la fonction d'évaluation peuvent aussi modifier le comportement de l'algorithme. Mais ces choix sont plus liés à l'application traitée qu'au processus de recuit.

Le critère d'arrêt a aussi son importance. Il doit permettre de repérer quand la situation devient figée, c'est-à-dire quand une solution optimale est trouvée. Cette tâche n'est pas aussi évidente qu'elle n'y paraît. La façon de l'effectuer, ainsi que la gestion des deux paramètres sont regroupées dans ce que nous appelons un plan de refroidissement. Cette notion sera développée au point 2.6. .

De la même manière que la recherche locale, le recuit simulé utilise la technique d'amélioration itérative à laquelle est cependant ajouté une technique aléatoire qui permet une certaine détérioration de la fonction d'évaluation. En fait, l'intensité de ces détériorations varie proportionnellement au paramètre de contrôle t . Au début, presque tous les changements sont acceptés. Plus t diminue, moins nombreux sont les changements permis de telle sorte que lorsque t approche de zéro, seules les améliorations modifient la solution.

C'est ce mécanisme qui permet au recuit simulé de ne pas être bloqué dans un extrémum local. De cette manière, le recuit simulé peut être vu comme une généralisation de la recherche locale en permettant d'accepter des solutions qui détériorent la fonction d'évaluation. D'autre part, si dans cet algorithme, le paramètre de contrôle est constamment maintenu à la valeur nulle, nous retrouvons une version de la recherche locale, ce qui nous permet de dire que cette recherche locale ressemble au processus physique de trempe.

2.5. Formalisation et convergence asymptotique

C'est la théorie des chaînes de Markov qui est utilisée pour modéliser mathématiquement le recuit simulé. Cette théorie permet d'établir un théorème de convergence asymptotique, dont nous parlons plus loin. Le but de ce paragraphe n'est pas de démontrer mathématiquement une quelconque propriété mais de montrer les bases de la démarche suivie et d'en retirer les résultats intéressants en vue de l'implémentation.

Définition 2.1

Soit un système qui évolue au cours du temps et un ensemble fini d'états E_1, ,.E_n dans lesquels le système peut se trouver. Soit $E(t_i)$ l'état du système au temps t_i. L'évolution de ce système est représentée par une chaîne de Markov si les probabilités de transition d'un état $E(t_i)$ à un état E(tj) ne dépendent pas des états antérieurs. Soit X(t) une variable stochastique qui décrit l'état au temps t_k, soient E_i,E_j deux résultats, alors la probabilité de transition est définie comme

$$P_{ij}(k) = Pr\{ X(t) = E_j / X(t-1) = E_i \}.$$

La matrice P composée des éléments définis ci-dessus est appelée matrice de transition.

Une chaîne de Markov est homogène si la probabilité de transition est indépendante de temps c'est-à-dire si

pour tout i,j, $P_{ij}(t)$ est indépendant de t.

Sinon, elle est inhomogène.

Appliquons ces définitions au recuit simulé. L'ensemble des états correspond à l'ensemble des solutions. Vu l'algorithme, il est clair que la probabilité qu'une solution apparaisse à la k ème transition dépend seulement de la solution à la (k-1) ème transition. D'ailleurs les probabilités de transitions peuvent être définies comme suit en supposant t_k constant :

$$\text{pour tout i,j} \quad P_{ij}(k) = P_{ij}(t_k) = \begin{cases} G_{ij}(t_k) * A_{ij}(t_k) & \text{si } i \neq j \\ 1 - \sum_{l \in S, l \neq j} P_{il}(t_k) & \text{sinon} \end{cases}$$

où G_{ij} est la probabilité de génération et est définie par

$$\text{pour tout i,j} \quad G_{ij}(t_k) = G_{ij} = \chi_{si}(j) / |S_i|$$

A_{ij} est la probabilité d'acceptation et est définie par

$$\text{pour tout i,j} \quad A_{ij}(t_k) = \exp(-\Delta f^+ / t_k)$$

avec pour tout $a \in R$, $a^+ = a$ si $a > 0$ et $a^+ = 0$ sinon.

G est appelé la matrice de génération et A celle d'acceptation. Les recherches théoriques qui portent sur le recuit généralisent ces définitions en permettant de prendre n'importe quelles matrices G et A pour autant que la matrice P définie ainsi corresponde à une matrice de transition. Les formules données ci-dessus sont celles qui décrivent le recuit simulé tel qu'il a été expliqué.

Prouver que l'algorithme se termine avec une solution optimale revient dès lors à prouver que

Il existe k tel que $P\{X(k) \in S_{opt}\} = 1$.

Cependant, seule une convergence asymptotique est démontrée, ce qui donne

$\lim \; P\{X(k) \in S_{opt}\} = 1$.

Il y a deux façons différentes de démontrer ce résultat [1], [13]. Dans la définition des P_{ij}, t est supposé constant. Or, dans le recuit, t_k décroît. Les deux démonstrations proviennent de deux façons de résoudre ce problème. Il y a en effet deux formulations distinctes du recuit simulé. Soit nous le considérons comme un algorithme homogène, c'est-à-dire que l'algorithme est décrit par une séquence de chaînes de Markov homogènes. Chaque chaîne est générée pour une valeur fixe de t et t décroît entre deux chaînes successives. Soit nous le considérons comme un algorithme inhomogène, c'est-à-dire que l'algorithme est décrit par une seule chaîne de Markov inhomogène. t décroît alors entre deux transitions successives.

Dans le cas homogène, la convergence asymptotique est prouvée sous les conditions suivantes :

- chaque chaîne de Markov est de longueur infinie
- certaines conditions sont satisfaites pour les matrices A et G (ces conditions sont satisfaites pour les matrices que nous avons définies)
- $\lim \; c_k = 0$.

Le fait que chaque chaîne de Markov doive être de longueur infinie et donc que L_k soit infini, est non seulement impossible à réaliser mais surtout gênant car difficilement approximable.

Dans le cas inhomogène, la convergence asymptotique est prouvée sous les conditions suivantes :

- certaines conditions sont satisfaites pour les matrices A et G (ces conditions sont satisfaites pour les matrices que nous avons définies)
- $\lim \; c_k = 0$.
- sous certaines conditions supplémentaires pour A, la vitesse de convergence de la séquence des c_k n'est pas plus rapide que $O((\log k)^{-1})$

Cette deuxième manière est plus intéressante car nous supposons que t décroît tout en lui

laissant la possibilité de rester constant pendant un certain nombre de transitions. Dans ce cas, ces transitions constituent en fait une chaîne de Markov homogène de longueur finie. Cela est plus intéressant pour l'implémentation. Dans les deux cas, il reste le problème de trouver une condition d'arrêt qui permette d'obtenir un optimum. Une étude plus approfondie de ces paramètres et de l'implémentation se trouve au paragraphe suivant.

Cependant, avant de terminer cette partie théorique, nous signalons qu'outre ces théorèmes de convergence, d'autres études sont effectuées au sujet du recuit simulé afin d'en étudier le comportement et d'améliorer la stratégie des choix des paramètres, c'est-à-dire le plan de refroidissement. Certaines de ces recherches étudient plus en profondeur l'analogie avec la physique statistique et introduisent des notions de moyenne, de variance de la fonction d'évaluation ainsi que l'entropie. D'autres analysent l'espace des solutions sur base de l'ultramétricité. D'autres encore recourent aux verres de spin ... Il y a là tout un domaine de recherche ayant pour but d'améliorer le recuit simulé. Des études ont aussi été effectuées pour adapter le recuit à l'optimisation continue. Ceci termine le bref aperçu des recherches théoriques, essentielles pour le développement du recuit simulé. Nous pouvons envisager l'implémentation.

2.6. Plan de refroidissement

Nous venons de voir le théorème de convergence asymptotique. Le but de ce paragraphe est de montrer comment nous pouvons implémenter le recuit simulé afin d'obtenir, en un temps fini, l'optimum recherché ou du moins une " bonne " approximation. Nous expliquons ici ce que représente un plan de refroidissement, plan dont nous avons déjà parlé. Les aspects de l'implémentation liés à l'application traitée, c'est-à-dire les procédures d'initialisation et de génération, seront expliquées en même temps que les applications.

Comme nous l'avons vu, deux paramètres, t et L, jouent un rôle primordial. Le but de ce plan de refroidissement est de déterminer le comportement de ces deux paramètres en tenant compte de la convergence asymptotique.

Définition 2.2.

Un plan de refroidissement spécifie

- une séquence finie de valeurs du paramètre de contrôle c'est-à-dire
  1 une valeur initiale t_0
  2 une fonction de décroissance de ces valeurs
  3 une valeur finale utilisée dans le critère d'arrêt
- un nombre fini de transitions à générer pour chaque valeur de t c'est-à-dire
  4 une longueur finie de chaque chaîne de Markov.

Le plan de refroidissement comporte donc quatre éléments.

Une manière d'arrêter l'algorithme est de se fixer à priori une valeur finale du paramètre de contrôle. C'est la raison d'être du troisième point du plan de refroidissement. Cependant, comme nous le verrons, il y a d'autres façons de réaliser le critère d'arrêt.

Signalons, d'une part, qu'Aarts et van Laarhoven [13] proposent un plan de refroidissement qui se base sur des notions physiques, telle l'entropie, et sur des notions utilisées lors de la démonstration du théorème de convergence asymptotique.

D'autre part, van Laarhoven [13] fait un relevé assez détaillé de plans de refroidissement proposés par divers auteurs.

Dans ce qui suit, nous expliquerons les diverses possibilités qui ont été envisagées.

Nous nous sommes inspirés entre autre du plan de refroidissement proposé par Kirkpatrick, Gelatt et Veccchi [11]. Celui-ci a l'avantage d'être basé sur des règles " conceptuellement simples " . Celui de Aarts et van Laarhoven, par contre, est plus élaboré et repose sur des bases théoriques mais est aussi nettement plus complexe.

Les différents points du plan de refroidissement sont examinés ci-dessous.

2.6.1. Valeur initiale t_0.

Rappelons que la première phase du recuit est d'accroître la température du bain jusqu'à ce que le solide fonde. Cet état est caractérisé par un arrangement aléatoire des molécules.

Se basant sur ce fait, Kirkpatrick propose de choisir une valeur initial t_0 suffisamment grande, telle que n'importe quelle transition soit acceptée.

Nous avons procédé d'une manière différente. En nous basant sur le critère d'acceptation, nous avons choisi t_0 de sorte qu'une variation d'un certain pourcentage ait une probabilité d'être acceptée plus grande que un demi.

La valeur initiale du paramètre de contrôle est donc telle que

$$\exp(-\Delta f / t_0) = 0{,}5$$

c'est-à-dire $t_0 = -\Delta f / \ln(0{,}5)$.

où Δf représente la variation voulue. Différents pourcentages seront testés lors de

l'implémentation.

2.6.2. Fonction de décroissance du paramètre de contrôle

La manière la plus fréquente de faire varier le paramètre de contrôle est de le multiplier par un facteur constant. Cela donne

$t_{k+1} = \alpha * t_k$, k=1,2,...

Les valeurs habituelles de ce facteur α sont comprises entre 0,8 et 0,99.

Rappelons ici que la façon dont décroît le paramètre de contrôle est lié au nombre de transitions générées, c'est-à-dire à la longueur des chaînes de Markov qui sera traité au point quatre.

2.6.3. Critère d'arrêt

Une première possibilité est de se fixer au départ un nombre d'itérations. Cela nécessite bien sûr de pouvoir prévoir à l'avance la valeur du paramètre de contrôle telle qu'aucune amélioration ne soit possible pour l'algorithme. Cependant, cette façon de faire ne sera pas utilisée.

Une deuxième possibilité est de s'arrêter lorsque la fonction d'évaluation de la solution obtenue à la fin de la chaîne de Markov, c'est-à-dire après les Lk transitions, reste inchangée pendant un nombre déterminé d'itérations consécutives.

Par expérience, nous avons déterminé que cinq ou six itérations constantes suffisaient pour mettre fin à l'algorithme, celui-ci ne pouvant plus améliorer la solution courante.

La troisième possibilité ressemble à la deuxième. En fait, en observant le déroulement du recuit simulé lors des différents tests, nous nous sommes rendus compte que, en moyenne pendant les vingt dernières itérations, l'algorithme oscille entre deux, parfois trois, solutions. Il faut en fait attendre que le paramètre de contrôle atteigne une valeur siffisamment basse pour que la solution voisine, mais un peu moins bonne, ne soit plus acceptée.

Dès lors, une variable a été ajoutée pour contenir le minimum des valeurs de la fonction d'évaluation des solutions obtenues à la fin des chaînes de Markov, et le critère d'arrêt teste que la valeur de cette variable reste inchangée pendant cinq ou six itérations consécutives.

2.6.4. Longueur des chaînes de Markov

Rappelons qu'à chaque température, le système doit pouvoir atteindre son équilibre. C'est en fonction de cela qu'il faut choisir la longueur des chaînes de Markov. Un argument intuitif nous suggère que cet équilibre est restauré à condition qu'un nombre suffisant de transitions ait été accepté. Cependant, puisque la probabilité d'acceptation dépend du paramètre de contrôle et dès lors décroît, cela conduirait à ce que L tende vers l'infini, lorsque t s'approche de la valeur nulle.

Il faut donc fixer une borne supérieure. Celle-ci est, en général, de la même taille que le voisinage d'une solution.

Le choix de la longueur des chaînes de Markov revient donc à fixer une valeur initiale, une façon d'augmenter et une borne supérieure pour le nombre de transitions générées. Remarquons que ces choix sont guidés non seulement par le souci d'obtenir des solutions de bonne qualité, mais aussi parfois, par contrainte au niveau du temps d'exécution.

2.7. Implémentation

Comme nous l'avons mentionné lors de la description de l'algorithme, certaines parties sont communes à la recherche locale. C'est ainsi que les points énumérés ci-après ont été expliqués au point 1.4.5. . Il s'agit de la représentation des solutions, de leur initialisation, du mécanisme de génération et de la fonction d'évaluation. L'implémentation de ces parties et donc de la façon de calculer f et Δf ainsi que de changer de solution, s'effectue de la même manière que pour la recherche locale.

D'un autre côté, les aspects propres au recuit telle la gestion du paramètre de contrôle, du nombre de transitions et du critère d'arrêt ont été détaillés lors du plan de refroidissement. Leur implémentation ne pose aucun problème.

Ce qui est intéressant à signaler ici, ce sont les deux parties qui forment le programme du recuit simulé. D'une part, il y a un ensemble de procédures propres à l'application. D'autre part, l'autre partie du programme s'occupe des aspects propres au recuit simulé.

Dès lors, changer de problème d'optimisation implique de modifier la première partie, mais permet de conserver la deuxième intacte. Par contre, la partie propre à l'application peut être utilisée par différentes méthodes. C'est le cas de la recherche locale, du recuit simulé et, comme nous le verrons plus tard, de la recherche tabou.

L'implémentation du recuit simulé ne pose à présent plus aucun problème. Il nous reste à détailler les résultats obtenus pour les deux applications traitées.

2.8. Résultats

Indiquons, pour commencer, les points pratiques qui sont valables pour les trois méthodes. Tout d'abord, les programmes ont été réalisés en Turbo Pascal 5.0. . Ensuite, les différents tests ont été réalisés sur des P.C.386 SX. Enfin, sauf mention du contraire, la taille (N) des problèmes est fixée à cinquante.

2.8.1. Le voyageur de commerce

Rappelons d'abord quelques données du problème. Le nombre de villes est donc fixé à cinquante. L'estimation du minimum, donnée au point 1.3.1. donne une valeur de 5,30. En effet, nous avons choisi de positionner les villes dans un carré unité. L'aire A de la région vaut donc un.

Remarquons, dès à présent, que cette estimation est valable pour N suffisamment grand et que nous pouvons nous demander à quel point elle est réaliste dans le cas de cinquante villes.

La première comparaison porte sur les deux moyens de réaliser le critère d'arrêt. Le test A utilise la valeur minimum de la fonction d'évaluation tandis que le test B utilise la valeur courante de cette fonction. La première conclusion est qu'il n'y a aucune différence en ce qui concerne la distance de la solution finale. Par contre, le test A nécessite moins d'itérations et donc moins de temps que le test B. Le tableau 2.1. illustre cette comparaison. L'option du test A est donc utilisée dans les autres tests. La dernière ligne du tableau fournit la valeur moyenne de la colonne.

nombre d'itérations		temps d'exécution	
A	B	A	B
85	97	111	114
67	103	97	119
67	112	97	125
82	102	109	118
90	121	114	131
78	107	106	121

Tableau 2.1.: Critère d'arrêt.

Nous allons, à présent, comparer différentes gestions du paramètre de contrôle et du nombre de transitions générées à chaque itération. Dans un premier temps, nous avons fixé le comportement de t pour essayer différentes valeurs pour L. C'est ainsi que nous avons donné 0,30 comme valeur initiale de t et 0,95 comme valeur de α. (Pour rappel, le paramètre de contrôle est multiplié par ce facteur α à chaque itération).

Pour le paramètre L, trois valeurs sont à déterminer, à savoir une valeur initiale L_0 , une valeur finale L_f et un facteur ß de croissance. En fait, L croît jusqu'à atteindre la valeur finale, après quoi il reste constant.

Dans un premier temps, L_f est fixé à 2352, c'est-à-dire (n-1) * (n-2), ce qui correspond au double de la taille du voisinage. De plus, ß prend la valeur 1,05. La valeur initiale est alors choisie telle que L atteigne la valeur finale en cinquante itérations, ce qui conduit à prendre L_0=230. Cette première version a été comparée à d'autres, obtenues en modifiant quelque peu ces différentes valeurs. Cela donne la série des tests suivants :

test A : $L_0 = 230$ ß = 1,05 $L_f = 2352$
B : $L_0 = 300$ ß = 1,05 $L_f = 2352$
C : $L_0 = 100$ ß = 1,05 $L_f = 2352$
D : $L_0 = 230$ ß = 1,10 $L_f = 2352$
E : $L_0 = 230$ ß = 1,10 $L_f = (2352) / 2$

La première comparaison est effectuée pour le même problème, mais en initialisant le générateur aléatoire de dix façons différentes. Les résultats sont présentés par le tableau 2.2.

A	B	C	D	E
6,1440	6,1038	6,2175	6,2290	6,2349
6,2096	6,1872	6,4375	6,1038	6,2389
6,2292	6,1473	6,1578	6,1440	6,3897
6,1431	6,1404	6,1872	6,1432	6,3270
6,1542	6,1506	6,2501	6,0962	6,2175
6,2577	6,1431	6,2218	6,1038	6,1765
6,1605	6,1818	6,2100	6,1876	6,2020
6,3369	6,1566	6,2772	6,1038	6,1701
6,3172	6,2203	6,2307	6,3672	6,3680
6,0962	6,1820	6,2096	6,1425	6,1135
6,2048	6,1622	6,2399	6,1621	6,2437

Tableau 2.2.: comparaison du paramètre L.

En moyenne, ce sont les résultats de B et de D qui sont les meilleurs. Remarquons qu'il y a une plus grande dispersion dans les valeurs finales de D que dans celles de B. Le nombre d'itérations, par contre, est en moyenne le même (80) pour les deux tests. Cela a comme conséquence que le temps d'exécution de B est en moyenne plus rapide que celui de D (respectivement 60 et 76 secondes), car dans ce dernier, le paramètre L atteint plus vite la valeur finale, et donc doit générer plus de transitions. A ce stade, il est cependant prématuré de tirer une conclusion.

Dans un deuxième temps, la valeur initiale de t a été modifiée. Signalons que prendre t_0=0,30 revient à assurer une probabilité d'acceptation égale à un demi pour des variations Δf de l'ordre de 0,21. Cela correspond à 3,5% de 6,09, la meilleure solution trouvée. D'autres valeurs ont été envisagées, à savoir t_0=0,50 (5,7 %) ou t_0=0,90 (10%). Nous avons aussi essayé d'autres valeurs de L_0 pour ß=1,10. Cela donne la suite des tests suivants :

test F t_0= 0,50 L_0 = 300 ß = 1,05
G t_0 = 0,50 L_0 = 230 ß = 1,10
H t_0 = 0,50 L_0 = 300 ß = 1,10
I t_0 = 0,50 L_0 = 150 ß = 1,10
J t_0 = 0,90 L_0 = 100 ß = 1,20

Les résultats sont présentés par le tableau 2.3.

F	G	H	I	J
6,1463	6,1038	6,2836	6,1127	6,1012
6,2875	6,1038	6,0962	6,1556	6,1625
6,1899	6,0983	6,0962	6,1625	6,0962
6,1940	6,1199	6,0962	6,1625	6,1038
6,1730	6,1915	6,2117	6,1038	6,2090
6,0962	6,1431	6,1440	6,0962	6,1327
6,1038	6,0962	6,1481	6,0962	6,0962
6,2521	6,1038	6,2646	6,1730	6,0962
6,2199	6,1022	6,1765	6,0962	6,1038
6,1556	6,2899	6,1038	6,2067	6,3441
6,1818	6,1352	6,1621	6,1365	6,1446

Tableau 2.3.: Comparaison du paramètre t.

La première constatation qui découle de ces résultats, est que les tests G,I et J donnent une meilleure solution finale que les autres, y compris ceux de la comparaison précédente. Il s'agit pourtant de rester prudent. Les différences trouvées ne sont pas énormes. Certes, les résultats de E

sont à peu près deux pourcents plus élevés que ceux de G. Mais que dire de la différence entre G et F (0,75%) ou G et H (0,4%) ?

En regardant de plus près, nous constatons, tout d'abord, que les résultats finaux de G et I sont les moins dispersés. Ensuite, les résultats de G et J sont fort semblables, sauf pour les problèmes 2 et 8. Enfin H et I ont obtenu trois fois le minimum atteint par le recuit simulé. Il est, dès lors, complexe de tirer des conclusions.

Ces deux séries de tests font aussi ressortir le lien qu'il existe entre les paramètres t et L. En effet, B et F ne diffèrent que par la valeur de t_0, tout comme D et G. Or, les résultats donnent des valeurs similaires pour B et D, tandis qu'ils font apparaître un léger avantage de G par rapport à F.

C'est là l'illustration de la complexité du choix de ces paramètres. Les conclusions hâtives sont risquées !

Ensuite, nous avons effectué une troisième comparaison. Pour cela, nous avons choisi dix problèmes différents. C'est l'occasion de tester la méthode par rapport à l'estimation de l'optimum. Les tests suivants ont été réalisés :

R1 $t_0 = 0,30$ $L_0 = 300$ $ß = 1,05$
R2 $t_0 = 0,50$ $L_0 = 150$ $ß = 1,10$
R3 $t_0 = 0,90$ $L_0 = 100$ $ß = 1,10$
R4 $t_0 = 0,90$ $L_0 = 100$ $ß = 1,05$
R5 $t_0 = 0,50$ $L_0 = 300$ $ß = 1,05$

Les résultats sont présentés par le tableau 2.4.

R1	R2	R3	R4	R5
5,5841	5,5971	5,5841	5,5740	5,6030
5,4355	5,4439	5,4438	5,4438	5,6172
6,2882	6,1484	6,2824	6,1398	6,2925
6,1369	6,2547	6,1178	6,1185	6,0847
5,7618	5,5823	5,6837	5,8124	5,5824
5,6502	5,6502	5,6502	5,6948	5,6948
6,2851	6,2692	6,4128	6,2767	6,2066
5,5181	5,6005	5,5153	5,5153	5,6052
5,4726	5,3681	5,4000	5,5002	5,4445
6,1675	6,0553	6,0221	6,0221	6,0707
5,8300	5,7975	5,8112	5,8088	5,8802

Tableau 2.4.: Comparaison du recuit simulé.

Comme lors de la comparaison précédente, il est difficile de tirer des conclusions du tableau 2.4.

Ce qui ressort le plus clairement, c'est l'influence du problème sur le résultat. En effet, il y a moins de dispersion dans les résultats d'un problème (une ligne) que d'un test (une colonne) . De même, si nous regardons le meilleur et le moins bon résultat obtenu par chaque test, nous constatons que les problèmes 2 et 9 se partagent les meilleures solutions finales, tandis que les problèmes 3 et 7 prennent les moins bonnes.

L'influence de génération aléatoire est difficile à déterminer. En fait, chaque problème initialise ce générateur différemment, comme dans les premières comparaisons. Nous pouvons nous demander quelle influence cela peut avoir sur les résultats. La faible dispersion obtenue dans les lignes du tableau 2.4. ne se retrouve pas dans les tableaux 2.2. et 2.3. . Cela tenderait à montrer la faible influence de ce générateur. Cependant, la dispersion entre les colonnes de ces mêmes tableaux 2.2. et 2.3. ne provient que de cette initialisation différente.

Cela nous suggère de conclure que ce générateur a une influence sur les résultats obtenus, mais que dans les comparaisons effectuées, cette influence est limitée.

Finalement, nous avons testé l'influence de l'initialisation sur le résultat final. Nous avons repris les valeurs des paramètres de R2. L'initialisation tentée recherche la ville la plus proche, soit dans vingt villes (R7), soit dans toutes les villes R8. Dans R8, toutefois, la valeur initiale est abaisée (t_0= 0,15). Cela correspond au fait que le système est déjà ordonné et qu'il n'est plus nécessaire de partir d'une température si élevée.

Comme nous pouvons le voir dans le tableau 2.5. , les résultats sont à nouveau très proches. Le principal intérêt de cette initialisation est le gain de temps. En effet, le tableau 2.5. fournit aussi le temps moyen d'exécution et le nombre moyen d'itérations. Cela permet de constater que R8 prend en moyenne un peu plus de la moitié du temps d'exécution de R2.

Quelque soit le test, les valeurs obtenues sont assez éloignées de l'estimation de l'optimum. Il y a en effet presque 10% de différence. Deux remarques nuancent ce fait. D'une part, comme mentionné ci-dessus, l'estimation vaut pour N suffisamment grand. D'autre part, le recuit simulé, comme les autres méthodes, est réputé avoir un meilleur comportement lorsque N est plus grand que cent.

R2	R7	R8
5,5971	5,5927	5,5740
5,4439	5,4531	5,4356
6,1484	6,2721	6,1398
6,2547	6,2829	6,1336
5,5823	5,9246	5,6750
5,6502	5,6602	5,6781
6,2692	6,2011	6,3898
5,6005	5,6215	5,6052
5,3681	5,4938	5,5040
6,0553	5,9987	6,0221
5,7975	5,8201	5,8157

Tableau 2.5.: Comparaison des initialisations.

Nous pouvons , à présent, passer à la coloration de graphe. Il est clair que nous tiendrons compte des informations que nous venons de retirer.

2.8.2. Coloration de graphe

Rappelons qu'il s'agit de colorer les cinquante sommets d'un graphe dont la matrice d'adjacence est donnée. Toutes les comparaisons se font en testant la méthode sur dix problèmes différents. Signalons, à ce sujet, que ces dix problèmes ne sont pas quelconques. En fait, les matrices d'adjacence sont telles que la densité des graphes correspondants est proche de N/2.

Une remarque concernant les résultats est encore nécessaire. En effet, les fonctions d'évaluation, comme nous l'avons expliqué, dépendent de la détermination de poids. Dès lors, le résultat intéressant n'est pas de savoir quelle valeur a prise la fonction d'évaluation, mais plutôt combien de couleurs ont été utilisées, ou dans le cas de la satisfaisabilité, combien de solutions sont des colorations.

Dans un premier temps, les poids définis par Aarst et Korst, qui transforment le problème en une maximisation, sont utilisés. De plus, le nombre maximum Δ de couleurs permises est fixé à quinze. Cela donne les poids suivants : $w_1 = 16400$, $w_2 = 16399$, $w_3 = 16397$, $w_4 = 16393$, $w_5 = 16385$, $w_6 = 16369$, $w_7 = 16337$, $w_8 = 16273$, $w_9 = 16145$, $w_{10} = 15889$, $w_{11} = 15377$, $w_{12} = 14353$, $w_{13} = 12305$, $w_{14} = 8209$, $w_{15} = 17$

Le facteur de pénalité λ est fixé à 2 et la taille du voisinage est égale à (Δ-1)*N, ce qui donne 700.

La première comparaison porte sur la gestion de L et de t. Nous avons tout d'abord fixé L à une valeur constante. Donc $L_f=L_0$. Les valeurs proposées sont 700 et 2000. Différentes gestions de t sont alors tentées. Cela donne la série des tests suivants :

RC1 L = 2000 t_0 = 200 α = 0,95
RC2 L = 2000 t_0 = 1000 α = 0,9
RC3 L = 2000 t_0 = 10000 α = 0,9
RC4 L = 700 t_0 = 10000 α = 0,9
RC5 L = 700 t_0 = 50000 α = 0,8
RC6 L = 700 t_0 = 50000 α = 0,9

Les résultats sont présentés dans le tableau 2.6.

RC1	RC2	RC3	RC4	RC5	RC6	RC7
10	10	10	11	11	11	11
12	11	10	11	12	11	11
10	11	10	11	11	11	11
11	10	11	10	11	11	11
11	10	11	11	12	11	11
11	12	11	11	12	12	11
11	11	10	10	11	11	10
11	11	10	11	12	11	11
11	10	11	12	12	11	11
11	10	11	11	11	11	10
10,9	10,6	10,5	10,9	11,5	11,1	10,8

Tableau 2.6.: Comparaison en coloration de graphe.

Le test RC7 est différent, car il laisse varier L. Ses paramètres sont donc : L_0=700 , L_f=2000 , ß=1,05 , t_0=10000 , α=0,9.

Les deux meilleurs tests sont donc RC2 et RC3. Ils arrivent tous les deux à produire cinq colorations en dix couleurs. RC2 a une moins bonne moyenne due à une coloration en douze couleurs.

Signalons, d'une part, que la valeur moyenne de la fonction d'évaluation est de 816001 pour RC3, 815187 pour RC2 et 814057 pour RC5. De plus, les différentes valeurs obtenues pour cette

fonction se situent entre 812439 et 817381.

Ce qui est assez surprenant, c'est qu'il est possible de trouver deux solutions, telles que la première utilise moins de couleurs, mais qu'elle ait une valeur inférieure quant à la fonction d'évaluation. Il est aussi possible de trouver deux colorations qui sont identiques à une renumérotation près des couleurs, mais qui sont différentes quant à la fonction d'évaluation.

En fait, le choix des poids favorise les grands ensembles indépendants, quitte à devoir terminer par de très petits ensembles. Or Bollobas [2] montre qu'il est préférable d'essayer de trouver une partition en ensembles de tailles semblables. Ce n'est certes pas le cas ici, et pourtant, comme nous le verrons plus tard, c'est cette façon de procéder qui donne les meilleurs résultats. Ajoutons que ce que préconise Bollobas est valable pour un nombre de sommets importants, ceci en est peut-être l'explication.

Signalons d'autre part que les valeurs choisies pour t_0 correspondent à des variations Δf de 9% si t_0=50000 et 1,8% si t_0=10000.

Dans un deuxième temps, tout en gardant les mêmes poids, nous avons donné la valeur 10 ou 11 à Δ. Cela revient au problème de satisfaisabilité, comme cela a été évoqué au point 1.3.2. . Dès lors, le résultat intéressant est le nombre de colorations obtenues.

Cela nous amène à effectuer les trois tests qui suivent :

RC8 $\Delta = 10$ $t_0 = 2000$
RC9 $\Delta = 10$ $t_0 = 1000$
RC10 $\Delta = 11$ $t_0 = 4000$

Pour les trois tests, le paramètre L est fixé à 2000. De plus, les valeurs des poids changent en fonction de Δ et les valeurs t_0 correspondent à des variations de 5,5% pour RC8 et RC10, et 2,7 % pour RC9.

RC8 donne trois colorations, et pour les sept autres partitions, il y a en tout 19 arêtes monochromes, ce qui donne à peu près 3 arêtes en moyenne.

RC9, par contre, donne deux colorations et 23 arêtes monochromes, ce qui revient à la même moyenne.

RC10, quant à lui, donne sept colorations, dont une en dix couleurs, et 4 arêtes monochromes.

Ces résultats sont donc moins bons que ceux obtenus par RC3.

Finalement, l'autre façon de choisir les poids, décrite au point 1.3.2. est utilisée. Le nombre

de couleurs permises est quinze et le facteur de pénalité λ est quarante. De plus, t_0=10000 et α=0,90. Le premier RC11 laisse varier L de 700 à 2000 avec ß=1,05, tandis que la deuxième RC12 bloque L à 700, et le troisième prend la valeur 2000.

Après cela, tout en gardant L=2000, nous avons changé la valeur t_0 . Vu le changement des poids, cette valeur n'est plus adaptée. Cela donne les tests suivants :

RC13 $t_0 = 100$
RC14 $t_0 = 50$
RC15 $t_0 = 200$

Les résultats sont présentés par le tableau 2.7.

RC11	RC12	RC13	RC14	RC15
11	12	11	12	11
11	11	11	11	11
12	12	12	12	13
11	11	10	12	11
11	12	11	10	11
12	13	11	11	11
11	12	11	11	11
11	12	12	11	11
11	10	10	11	11
11	11	10	10	10
11,2	11,6	10,9	11,1	11,1

Tableau 2.7.: Comparaison en coloration de graphe.

Nous pouvons conclure que c'est la valeur 100 qui donne le meilleur résultat. Cependant, ce choix de poids s'avère moins efficace que celui fait avant.

Pour terminer, signalons qu'il n'était pas possible de tester toutes les combinaisons des paramètres. Comme nous l'avons remarqué, la gestion de L et t est assez complexe. Ajoutons à cela, qu'il est possible d'utiliser le recuit simulé comme une méta-méthode pour optimiser ces paramètres. Pour cela, nous pensons cependant qu'il faut travailler avec des ordinateurs plus puissants.

2.9. Conclusion

Une des caractéristiques principales du recuit simulé est que cette méthode provient d'un processus physique. Le but est de modéliser le mieux possible ce phénomène, afin de pouvoir l'appliquer à des problèmes d'optimisation.

Une autre caractéristique importante est la démonstration de convergence. Cela permet de dire que le recuit simulé est un algorithme.

De plus, l'analogie physique sert de base à de nombreuses recherches théoriques. Le comportement du recuit simulé est ainsi étudié en profondeur afin d'obtenir l'optimum en un temps fini.

C'est là que se situe le problème: la convergence est asymptotique. Il faut, dès lors, définir des plans de refroidissement qui permettent d'espérer des résultats proches de l'optimum, si pas égaux à celui-ci.

C'est ainsi que la gestion des paramètres t et L est assez complexe. Nous avons eu l'occasion de le souligner à plusieurs reprises, ainsi que de mettre en évidence le lien qui existe entre ces deux paramètres. Par contre, l'implémentation de l'algorithme est simple. La programmation ne recourt à aucune technique spéciale. De plus, le but de chaque partie de l'algorithme est clair et simple à réaliser.La mise au point des paramètres se fait alors en changeant seulement les valeurs de certaines constantes.

Pour ce qui est des résultats, notre avis est plus mitigé. D'une part, parmi les trois méthodes étudiées, c'est celle du recuit simulé qui fournit les meilleures solutions pour les deux applications traitées.

D'autre part, par rapport à l'estimation de l'optimum du problème du voyageur de commerce, les résultats du recuit simulé sont moins bons. Une des causes en est la taille du problème. Cependant, pour des problèmes plus importants, nous pensons qu'il est préférable de disposer de machines plus puissantes.

En conclusion, le recuit simulé est une méthode simple à implémenter, un peu plus complexe à mettre au point et qui donne de bons résultats, surtout pour des problèmes de plus grandes envergures.

CHAPITRE 3 : LA RECHERCHE TABOU

3.1. Introduction

Le principe de base de la recherche tabou est de parcourir une partie du voisinage (au sens vu lors de la recherche locale) de la solution courante et d'en choisir la meilleure. Cette dernière devient solution courante, qu'elle améliore ou non la fonction d'évaluation. A cela s'ajoute une liste de mouvements, dont le statut est tabou et qui sert à empêcher de revenir sur ses pas. Ces deux mécanismes doivent permettre d'éviter de rester bloqué dans un optimum local.

Dans un premier temps, nous allons décrire ces deux mécanismes en détail. Ensuite, nous expliquerons pourquoi il est parfois utile qu'un mouvement puisse se défaire de son statut tabou et comment cela peut se réaliser. Nous pouvons alors élaborer un algorithme . Le choix des différents paramètres est ensuite traité avant d'illustrer cet algorithme sur les deux applications choisies.

3.2. Voisinage et liste tabou

Comme pour la recherche locale et le recuit simulé, les notions de voisinage et de mécanisme de génération à partir d'une solution courante sont utilisés. Voici la démarche générale : Supposons avoir une solution i courante et son voisinage S_i. Générons alors un voisinage partiel V_{pi}, sous-ensemble de S_i et choisissons la meilleure d'entre-elles, soit j. Alors j remplace i que ce mouvement améliore ou non la fonction d'évaluation. Cela permet donc d'accepter une certaine détérioration de la fonction d'évaluation.

Cependant, cela n'est pas suffisant pour éviter de rester bloqué dans un optimum local. En effet, si i est un tel optimum, nous choisirons une solution j moins bonne. Mais à l'étape suivante, il est possible que i soit sélectionnée dans le Vpj, et dès lors, que i soit choisi à ce moment. Il faut donc trouver un moyen qui empêche de revenir à une situation que l'on vient de visiter sous peine de rester dans une région au détriment des autres, sans être sûr qu'une solution optimum s'y trouve. C'est le but de la liste tabou.

Cette liste comprend en fait un nombre fixe de mouvements dont le statut est tabou, c'est-à-dire qu'ils ne peuvent être choisis pour remplacer la solution courante. D'une manière générale, cela se passe comme suit : chaque fois qu'une solution j est choisie pour remplacer une solution i, le mouvement inverse, c'est-à-dire le passage de j à i, est ajouté à la fin de la liste tabou tandis que

le premier, c'est-à-dire le plus ancien, est retiré de cette liste. Ce statut n'est donc que provisoire. En fait, ce n'est pas le passage de j à i qui devient tabou, mais c'est la partie de i qui s'est modifiée pour devenir j qui ne peut reprendre son ancienne valeur. Illustrons par un exemple : prenons la coloration de graphe. Le passage de i à j s'effectue en changeant de couleur un sommet, noté x. x passe d'une couleur à une autre. Le mouvement tabou est alors de rendre la couleur initiale au sommet x quelques soient les couleurs des autres sommets, qu'ils soient ou non de la même couleurs qu'en i.

A	B	C	D	E
3	2	2	1	3
3	1	2	1	3
2	1	2	1	3
2	3	2	1	3
2	3	3	1	3
2	2	3	1	3

Figure 3.1. : La liste tabou.

La figure 3.1. illustre l'utilisation de cette liste tabou. Ce tableau représente les couleurs des sommets A, B, C, D, E pour six applications du mécanisme de génération. Le dernier mouvement est en fait tabou car il redonne la couleur 2 au sommet B même si, depuis lors, le sommet A a aussi changé de couleur.

D'autre part, il arrive que les mouvements soient divisés en différents types suivant leur direction. Alors, à chaque direction correspond une liste tabou. Par exemple, en programmation linéaire en nombre entier, une variable se voit attribuer une valeur adjacente à sa valeur courante. Il y a alors deux types de mouvements : les augmentations et les diminutions. Supposons que la variable x soit dans la liste des augmentations tabous pour être passée de la valeur 3 à la valeur 4. L'usage de cette liste sert alors à empêcher que cette variable ne reprenne la valeur 3 mais rien ne lui interdit de prendre la valeur 5. En coloration de graphe par contre, une seule liste suffit car un mouvement consiste à changer de couleur un sommet : la seule chose à empêcher est que ce sommet ne reprenne sa couleur initiale.

3.3. Comment passer outre son statut tabou ?

Le chemin pour arriver à la solution optimale n'est pas toujours le plus direct. Ainsi en coloration de graphe, il peut être utile de changer temporairement de couleur un sommet x pour permettre de modifier la couleur de quelques autres sommets puis de redonner sa couleur initiale au sommet x. Sans ce " déplacement temporaire " , les autres modifications seraient refusées vu

la trop grande détérioration de la fonction d'évaluation que cela entraînerait. Pour cela, il faut permettre, dans certains cas, à un mouvement de passer outre son statut tabou pour lui permettre d'être choisi. C'est le but de la fonction Asp introduite par Glover.[8]

Cette fonction Asp(z) peut être interprétée comme le niveau espéré que la fonction d'évaluation va atteindre comme valeur suivante lorsque la valeur courante est z. Cette fonction est donc définie sur les valeurs de la fonction d'évaluation. Cependant, que ce soit par translation, échantillonnage ou autre méthode, z est supposé prendre des valeurs entières entre 1 et U. z = 0 est alors une borne inférieure de l'optimum de f et toutes les valeurs supérieures à U sont considérées comme égales à U.

Initialement, nous posons Asp(z) = z-1. Cela veut dire que nous espérons au moins une amélioration d'une unité (cette unité peut représenter différentes grandeurs suivant la méthode utilisée pour amener les valeurs de f entre 1 et U). Améliorer d'une unité correspond donc à la plus petite amélioration attendue de la fonction d'évaluation, étant donné que nous avons supposé que cette fonction est à valeurs dans N.

L'utilisation pratique de cette fonction est la suivante. Un mouvement s(i-j) peut passer outre son statut tabou si l'amélioration que ce mouvement procure à la fonction d'évaluation est meilleure que celle espérée c'est-à-dire si $f(j) \leq Asp(z)$ avec z = f(i). Le signe $\leq$ provient du fait que nous supposons avoir un problème de minimisation.

Voyons à présent comment mettre à jour cette fonction. Remarquons que cette mise à jour s'effectue à chaque changement de solution, qu'un mouvement tabou soit en cause ou non. Supposons que la solution courante i a z comme valeur de la fonction d'évaluation et que le mouvement considéré amène à une solution j et une valeur z'. Dès lors, nous espérons que tous les mouvements futurs à partir de la valeur z atteignent au moins z'-1, si z' est plus petit que Asp(z) ce qui peut s'écrire

$$Asp(z) = \min (Asp(z), z'-1)$$

Par contre, si z' est plus grand que z, nous obligeons que les mouvements futurs à partir de z', atteignent au moins z-1, ce qui peut s'écrire :

$$Asp(z') = \min (Asp(z'), z-1)$$

Donc si un mouvement déteriore la solution courante, cela revient à obliger à tous les mouvements futurs d'obtenir une fonction d'évaluation meilleure qu'elle n'était avant cette déterioration. Cette condition n'est bien sûr imposée que si le mouvement veut passer outre son statut tabou. La figure 3.2. illustre ce point.

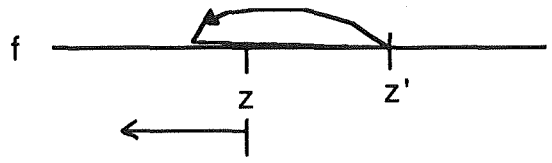

Figure 3.2. : La fonction Asp.

Il faut encore assurer le caractère non-croissant de la fonction, ce qui peut impliquer des mises à jour de la fonction pour les valeurs comprises entre z et z'.

Le domaine de définition de la fonction Asp peut ne pas être les valeurs de la fonction d'évaluation. Parfois, il s'agira d'une quantité d'amélioration attendue pour une certaine variable y. Prenons l'exemple du voyageur de commerce et du 2-change de Lin. y représente alors la longueur de la plus petite arête détruite. L'initialisation devient alors Asp(y) = 1 pour tout y, et la mise à jour se fait par les formules

$$Asp(y) = \max(Asp(y), z-z'+1)$$
$$Asp(y') = \max(Asp(y'), z'-z+1)$$

où y' est la longueur de la plus petite arête ajoutée et z, z' sont respectivement les valeurs de la fonction d'évaluation avant et après le mouvement.

3.4. Algorithme

Nous venons d'expliquer les deux éléments principaux de la recherche tabou, à savoir la liste tabou et le moyen de se défaire de ce statut. A présent, il nous est possible de comprendre le déroulement de l'algorithme présenté ci-dessus. Signalons que i, j, s représentent des solutions.

(i,j) représente le mouvement qui assure la transition de i à j

liste est la liste qui contient un nombre fixe de mouvements dont le statut est tabou.

k est le compteur d'itération.

```
procedure recherche_tabou;
begin
        initialisation(i,liste);
              (*consiste à choisir une solution initiale i ainsi que des mouvements
                 pour former la liste tabou*)
        k:=0;
        repeat
          constitution- voisinage- partiel (Vpi)
             (* consiste à générer un voisinage partiel de i,
               c'est-à-dire un nombre fixe de voisins j
               tels que (i,j) ne soit pas tabou ou f(j)≤ Asp (i) *)
          meilleur- voisin ( Vpi,s);
             ( * détermination de la meilleure solution de Vpi*)
          mettre- à- jour (liste, Asp);
                        (* mise à jour de la liste tabou et de la fonction Asp
                     comme expliqué précédemment *)
           i:=s;          (* changement de la solution courante, que s soit
                             meilleur ou non que i *)
           k:=k+1
      until critère- arrêt
              (* consiste à tester si l'algorithme peut se terminer *)
end;
```

Cet algorithme suggère quelques commentaires. Tout d'abord, son comportement est influencé par le choix des deux paramètres suivants :

- la taille de la liste tabou

- le nombre de voisins générés

Le critère d'arrêt, quant à lui, sert à déterminer si la solution obtenue est un optimum global. Cependant, cet algorithme n'obtient pas toujours cet extrémum : le blocage autour d'un optimum local est possible.

Le critère d'arrêt doit donc déterminer quand l'algorithme n'est plus capable d'améliorer la solution trouvée. La manière de réaliser cet arrêt est semblable à celle utilisée par le recuit simulé.

L'initialisation consiste d'une part à choisir une solution initiale soit au hasard, soit d'une manière plus ou moins déterminée. Nous verrons lors des tests, l'influence de ce choix sur le

résultat. D'autre part, l'initialisation de la liste tabou consiste à générer le nombre voulu de mouvements et de leur donner le statut tabou. La façon d'agir dans ce cas ne modifie en rien le comportement de l'algorithme.

Par souci d'optimisation, la détermination du meilleur voisin s'effectue en même temps que la constitution du voisinage partiel. Pour ce qui est de cette constitution, il y a une autre possibilité que de générer un nombre fixe de voisins. Elle consiste à s'arrêter dès qu'un voisin améliore la solution courante. Le nombre maximum de voisins n'est dès lors atteint que dans les autres cas.

La liste à jour de la liste tabou et de la fonction Asp ont déjà été expliquées. Toutes les parties de l'algorithme sont ainsi décrites. Il nous reste à voir comment les réaliser et comment choisir les paramètres.

3.5. Implémentation

Comme nous l'avons déjà mentionné, les parties propres à l'application sont communes à la recherche locale et au recuit simulé. Rappelons qu'il s'agit de la représentation des solutions, de leur initialisation, de la fonction d'évaluation et du mécanisme de génération. C'est ainsi qu'au niveau implémentation, nous avons réalisé une unité en Turbo Pascal qui regroupe les procédures relatives à l'application et qui est utilisée par les deux méthodes.

Voyons à présent les aspects propres à la recherche tabou. Il y a d'abord les choix des paramètres. Pour ce qui est de la taille de la liste tabou, Glover [8] préconise dans tous les cas de prendre sept. Nous avons dès lors testé des valeurs entre cinq et dix.

La taille d'un voisinage partiel doit bien sûr être inférieure à celle du voisinage lui-même. Les valeurs testées seront en fait assez proches de cette borne supérieure.

Le reste de l'implémentation ne nécessite que quelques commentaires. D'abord la constitution du voisinage partiel recourt bien entendu au mécanisme de génération. A cela s'ajoute une procédure pour tester si le mouvement proposé est tabou et dans ce cas, s'il peut passer outre ce statut. De même, il faut veiller à ce que le voisinage partiel soit constitué d'éléments différents.

Ensuite, la gestion de la fonction Asp et de la liste tabou a été détaillée ci-avant et ne pose aucun problème particulier à l'implémentation.

Enfin, en ce qui concerne les listes tabous, remarquons que pour éviter de devoir mémoriser trop d'éléments, il est parfois suffisant de conserver une partie du mouvement. Par exemple, pour le voyageur de commerce, si le 2-change de LIN est utilisé, un nouveau tour est engendré en détruisant deux arêtes non adjacentes et en ajoutant les deux seules arêtes qui permettent de

retrouver une solution réalisable différente de celle de départ. Dès lors, garder en mémoire une seule des deux arêtes détruites empêchera le mouvement inverse de s'effectuer.

Nous pouvons à présent passer aux tests réalisés pour les deux applications proposées.

3.6. Résultats

Nous allons, à présent, effectuer différents tests dans les mêmes conditions que pour le recuit simulé. Remarquons toutefois que le comportement de la recherche tabou dépend de deux paramètres, à savoir la taille de la liste tabou, noté tl, et le nombre de voisins, noté V. C'est évidemment moins complexe que le recuit simulé, ce qui explique qu'il y ait moins de tests effectués.

3.6.1. Voyageur de commerce

Dans un premier temps, la comparaison porte sur le nombre de voisins. Pour cela, nous avons fixé la taille de la liste à la valeur 7. De plus, nous avons repris le critère d'arrêt du recuit simulé. Néanmoins, le dernier test (T4) diffère de T3, parce qu'il oblige la valeur minimum à rester constante pendant dix itérations au lieu de cinq, avant d'arrêter l'algorithme.

Cela donne la série de tests suivants :

T1 V = 750 tl = 7
T2 V = 500 tl = 7
T3 V = 1000 tl = 7
T4 V = 1000 tl = 7 et arrêt modifié

Les résultats présentés par le tableau 3.1. présentent une grande régularité. Au niveau d'un même problème (ligne du tableau) , les solutions finales sont très proches. Dans certains cas, il n'y a même aucune différence, quelque soit le test. D'ailleurs, les comparaisons deux à deux des différentes colonnes ne montrent que quelques changements. C'est cependant T4 qui, dans tous les cas, s'en sort avec le meilleur résultat. Il est clair, cependant, que ce nombre de voisins n'influence pas d'une manière déterminante le comportement de la recherche tabou.

T1	T2	T3	T4
6,1565	6,1565	6,1687	6,1565
5,5794	5,5794	5,5794	5,4771
6,5235	6,5235	6,5235	6,5235
6,2763	6,3211	6,1587	6,1587
5,8497	5,8674	5,8497	5,8497
6,0705	6,0705	6,0705	6,0705
6,4075	6,4075	6,4075	6,4075
5,9822	5,9822	5,9822	5,9822
5,7114	5,6877	5,7111	5,6868
6,2279	6,2279	6,2279	6,2279
6,0785	6,0824	6,0679	6,0540

Tableau 3.1. : Comparaison du paramètre V.

Dans un deuxième temps, la comparaison porte sur la taille de la liste. Les valeurs de 5 à 9 ont ainsi été utilisées. Le nombre de voisins est fixé à 750. La série des tests est la suivante :

T5 V = 750 tl = 5
T6 V = 750 tl = 6
T7 V = 750 tl = 8
T8 V = 750 tl = 9

T5	T6	T7	T8
6,0766	6,1602	6,1687	6,0839
5,5794	5,5794	5,5794	5,5794
6,4663	6,4449	6,6543	6,6546
6,3480	6,3985	6,3894	6,7140
5,9525	5,9524	6,4698	6,4698
6,4067	6,4067	5,8190	5,8090
6,6677	6,2577	6,3334	6,2679
5,7299	6,4916	6,0991	6,0821
5,5809	5,5809	5,8478	5,8478
6,4876	6,4992	6,2279	6,1783
6,1296	6,1771	6,1579	6,1687

Tableau 3.2. : Comparaison du paramètre tl.

Les résultats présentés par le tableau 3.2. sont fort différents de ceux de la première

comparaison. Il est clair que la taille de la liste a une influence plus grande que le nombre de voisins. Les différences entre les diverses colonnes sont plus marquées. Remarquons que c'est toujours le deuxième problème qui récolte les meilleurs résultats.

Le test T1 obtient la meilleure moyenne, mais il n'arrive pas à devancer les autres pour tous les problèmes. Il semble, cependant, que la recommandation de Glover [8] de prendre Tl=7 soit bonne.

Finalement, nous avons comparé deux autres versions de la recherche tabou. La première (T9) génère à chaque itération, les nombres de voisins voulus (V). Pour rappel, la version précédente (T1 à T8) arrête la génération de voisins dès qu'il en trouve un qui améliore la solution courante.

La deuxième (T10) utilise une autre initialisation. Celle-ci consiste à prendre la ville la plus proche dans un groupe de vingt. Cette manière d'agir est la même que celle testée pour le recuit simulé.

Pour ces deux tests, les paramètres ont été fixés à 750 pour V et 7 pour tl. Les résultats, présentés par le tableau 3.3., mettent en évidence que la nouvelle initialisation détériore le comportement de la recherche tabou, et n'est donc pas à prendre en compte. Les résultats de T9, bien que meilleurs que ceux de T8, n'apportent pas non plus l'amélioration escomptée. De plus, le temps d'exécution moyen est de 53 secondes contre 13 pour T1. Le nombre moyen d'itérations est par contre de 59 pour T9 et de 145 pour T1.

T1	T9	T10
6,1565	5,7554	6,2377
5,5794	6,1602	6,0526
6,5235	6,4377	6,3360
6,2763	6,3658	6,8908
5,8497	6,1278	5,8300
6,0705	5,9133	5,9455
6,4075	6,3080	6,4373
5,9822	5,9222	5,9066
5,7114	5,6674	5,7211
6,2279	6,5139	6,4489
6,0785	6,1172	6,1806

Tableau 3.3. : Comparaison des initialisations.

En ce qui concerne les temps d'exécution, remarquons qu'ils sont nettement inférieurs à

ceux obtenus pour le recuit simulé. Pour rappel, R2 met en moyenne 80 secondes. Par contre, les résultats obtenus sont supérieurs, R2 obtenant 5,7975.

C'est pour cela que nous avons essayé la version T9, en espérant que si la recherche tabou prenait plus de temps à passer d'une solution à une autre, cela lui permettrait d'éviter les optima locaux, et ainsi pouvoir s'améliorer. Les résultats obtenus ne vont cependant pas dans le même sens!

Nous passons à la coloration de graphe où nous pouvons constater les mêmes problèmes.

3.6.2. Coloration de graphe

Comme pour le recuit simulé, il s'agit de colorer les cinquante sommets d'un graphe. Dans un premier temps, nous utilisons les poids d'Aarts et Korst. La remarque du point 2.8.2., concernant la fonction d'évaluation, reste valable ici. Nous nous intéressons donc au nombre de couleurs utilisées.

La première comparaison utilise certains résultats obtenus pour le voyageur de commerce. C'est ainsi que la taille de la liste tabou est fixée à 7 et le nombre de voisins à 600. Rappelons que la taille du voisinage est (Δ-1) * N.

Ce qui change, c'est le nombre d'arrêt (Na), c'est-à-dire le nombre de fois que la valeur minimum de la fonction d'évaluation doit rester constante pour arrêter l'algorithme.

Le facteur de pénalité est fixé à λ=2. Les tests suivants sont effectués :

TC1 $\Delta = 15$ Na = 15
TC2 $\Delta = 15$ Na = 50
TC3 $\Delta = 12$ Na = 15

Après cela, nous avons testé d'autres valeurs des paramètres V et tl. Cela donne les tests :

TC4 $\Delta = 15$ Na = 15 tl = 5 V = 600

TC5 $\Delta = 12$ Na = 15 tl = 7 V = 500

TC6 $\Delta = 12$ Na = 15 tl = 7 V = 400

Les résultats sont présentés dans le tableau 3.4. La dernière ligne indique le nombre de solutions qui ne sont pas des colorations.

Comme nous pouvons le constater, les tests avec Δ=12 n'obtiennent pas toujours des colorations. C'était aussi le cas pour le recuit simulé. Cela indique que pour utiliser les poids

d'Aarts et van Laarhoven, il faut fixer un Δ plus grand que le nombre de couleurs espéré.

Pour les tests avec Δ=15, le TC2 améliore deux colorations de TC1 et donne donc les meilleurs résultats.

TC1	TC2	TC3	TC4	TC5	TC6
11	11	11	11	11	11
12	11	12	12	12	12
12	12	12	12	12	12
13	13	12	12	12	12
12	12	12	12	12	12
13	13	12	14	12	12
12	11	12	12	12	12
12	12	12	13	12	12
11	11	12	12	12	12
12	12	12	12	12	12
12	11,8	11,9	12,2	11,9	11,9
0	0	2	0	3	5

Tableau 3.3. : Comparaison en coloration de graphe.

Signalons que le temps d'exécution moyen est de 61 secondes pour TC1, 168 pour TC2 et 78 pour TC3. Le nombre moyen d'itérations est de 111 pour TC1, 165 pour TC2 et 100 pour TC3.

L'autre façon de choisir les poids n'a pas été utilisée car, lors des tests pour le recuit simulé, elle s'est avérée moins efficace.

Comme pour le voyageur de commerce, les résultats obtenus par la recherche tabou sont un peu moins bons que ceux du recuit simulé. Signalons en plus, que nous avons ajouté un compteur pour connaître le nombre de fois que la fonction Asp permettait à un mouvement de passer outre son statut tabou.

Pour le voyageur de commerce, le nombre de mouvements tabous varie entre 60 et 110. La fonction ne permet en général à aucun mouvement de passer outre son statut tabou. En fait, pour dix utilisations de la méthode, un seul mouvement profite de la fonction Asp.

Pour la coloration de graphe, le nombre de mouvements tabous varie entre 30 et 175. Sur dix utilisations de la méthode, la fonction Asp permet entre 0 et 14 mouvements de passer outre leur statut, ce qui fait un maximum de deux mouvements par exécution.

Cette fonction Asp n'est visiblement pas au point. Soit elle ne sert à rien, soit les auteurs qui l'utilisent ne veulent pas dévoiler ses secrets.

3.7. Conclusion

Contrairement aux deux autres méthodes, la recherche tabou n'a pas de lien direct avec un processus physique. C'est en fait une amélioration de la recherche locale.

De plus, c'est une méthode apparemment moins utilisée que les autres. Les articles à son sujet sont plus rares. Les recherches théoriques sont moins importantes.

L'implémentation de cette méthode est un peu plus complexe que celle du recuit simulé. La fonction Asp et la liste tabou doivent être réalisées avec quelques précautions. La programmation reste malgré tout assez simple.

Le nombre de paramètres étant assez réduit, la mise au point ne pose aucun problème. Pourtant, les résultats ne suivent pas.

Comment améliorer le comportement de la recherche tabou ? C'est pour nous une énigme. Nous avons essayé les diverses modifications qui nous paraissaient possibles.

Le seul point qui reste à élucider, c'est le rôle de la fonction Asp. La façon dont nous l'avons réalisée n'a pas été très utile. Il reste à supposer que les auteurs qui l'ont utilisée ne dévoilent pas tout à son sujet... Est-ce tabou?

En conclusion, la recherche tabou est donc une méthode assez simple à implémenter et à mettre au point, mais les résultats ne sont pas très concluants.

CHAPITRE 4 : LES ALGORITHMES GENETIQUES

4.1. Introduction

C'est par analogie avec la théorie de l'évolution en biologie que J.Holland [10], le premier, conçut les algorithmes génétiques. L'idée de base de ces algorithmes est qu'une population d'organismes en évolution peut être perçue comme un processus de recherche. Tout comme une société de fourmis ou un volume macroscopique d'un gaz, une telle population en évolution est un système dynamique complexe et la propriété la plus importante d'un tel système est qu'il est capable d'auto-organisation. Par exemple, pour établir un chemin entre une source de nourriture et leur nid, ce n'est pas une fourmi en particulier qui décide, mais c'est par leurs interactions locales que les fourmis s'organisent entre-elles. Elles peuvent ainsi s'adapter à différentes situations.

Cette propriété d'auto-organisation a deux conséquences intéressantes : la flexibilité et la robustesse. Pour illustrer la flexibilité, prenons l'exemple suivant : Si vous posez un obstacle sur le chemin emprunté par les fourmis, elles sont capables, en peu de temps, de trouver un chemin qui le contourne. Elles s'adaptent donc au nouvel environnement. D'autre part, si vous retirez une fourmi, le comportement de la société entière n'en est que peu modifié. Cela nous amène à dire qu'ôter un élément n'affecte pas fondamentalement la performance du système . Nous parlons ici de la robustesse.

Nous allons d'abord décrire comment cette évolution peut être perçue comme un processus de recherche puis nous formaliserons quelques notions comme un génotype, une allèle ... ce qui nous permettra de décrire la structure d'un algorithme génétique. Nous expliquerons ensuite les deux phases principales. Il s'agit de la sélection dont le but est de reproduire les " bons " éléments et de la variation qui cherche à introduire du changement via des opérateurs génétiques. Cela nous permettra de détailler l'algorithme introduit précédemment. Nous terminerons alors par l'implémentation des algorithmes génétiques pour les deux applications étudiées, le voyageur de commerce et la coloration de graphe.

4.2. Evolution et recherche

La description des faits les plus importants de l'évolution en biologie va nous permettre de percevoir le processus de recherche sous-jacent. Nous soulignons que cette description n'a pas pour but d'expliquer scientifiquement l'évolution mais d'en retirer les points qui nous intéressent pour la suite.

Les caractéristiques d'un individu sont déterminées par ses chromosomes et plus particulièrement par les gènes qui le composent. En fait, chacun de ces gènes peut prendre différentes " valeurs " appelées " allèles " . Par exemple, un gène responsable de la couleur des cheveux peut être noir, châtain, blond ...Notons que c'est parfois l'association de plusieurs gènes qui déterminent une caractéristique : la couleur des yeux par exemple.

L'ensemble des gènes forme un génotype. La notion d'adéquation d'un individu ou par assimilation d'un génotype, représente une mesure de performance évaluant l'adaptation de l'individu à son environnement. Dès lors, l'évolution consiste en une recherche d'individus de mieux en mieux intégrés dans leur entourage. Plus techniquement, l'évolution est donc une optimisation de l'adéquation, c'est-à-dire une recherche d'optima dans un espace multidimensionnel constitué par les gènes. L'adéquation est l'équivalent de la fonction d'évaluation.

Remarquons que l'effet d'un allèle (d'un gène) peut dépendre fortement de la présence ou de l'absence d'autres allèles. Ce phénomène porte le nom d'épistasis. Dès lors l'optimisation ne peut se faire gène par gène mais l'évolution est une recherche d'ensemble d'allèles " co-adaptés " .

Deux processus constituent la base de cette évolution : la sélection et la variation. La sélection naturelle correspond au fait que les individus les mieux adaptés, c'est-à-dire ceux dont l'adéquation à l'environnement est la meilleure, ont le plus de chances de survivre. La variation, quant à elle, introduit du changement dans les descendants. C'est grâce à cela que les enfants ne sont pas les copies conformes des parents et que nous sommes tous différents, ce qui est nécessaire à l'évolution.

Mentionnons, pour terminer, que si nous regardons autour de nous, nous pouvons conclure que la capacité d'adaptation des individus est grande. Il nous reste à bien formaliser les différents aspects pour pouvoir utiliser cette puissance de recherche, ce qui n'est pas le moindre travail.

4.3. Formalisation

Le but recherché est de décrire d'une façon générale un algorithme génétique. La partie implémentation s'occupera des problèmes plus techniques : représentation, choix de f, ...

Un génotype est représenté par une chaîne a_1, a_2, ..., a_n d'éléments : le ième élément a_i correspond à l'allèle du ième gène. La longueur n des génotypes est constante. L'ensemble des génotypes est noté A et est défini par

$A = \{ a_1, a_2, , a_n$: a_i est l'allèle du ième gène $\}$

L'adéquation est une fonction f telle que

$f : A \longrightarrow R$

Une population d'individus et donc de génotypes est observée à différents moments. B(t) représente une telle population à l'instant t et M sa taille, considérée constante. A certains moments, il n'est utile que de connaître la valeur de certains gènes. C'est pourquoi la notation est introduite. Dès lors, a_1 a_4 désigne l'ensemble des génotypes de A tels que le gène 1 est porteur de l'allèle a_1 et le gène 4 de l'allèle a_4 tandis que les autres gènes peuvent prendre n'importe quelle valeur. Ainsi a_1 a_2 a_3 a_4 a_5 a_6 et a_1 b_2 b_3 a_4 b_5 b_6 appartiennent à ce sous-ensemble contrairement à a_1 a_2 a_3 b_4 a_5 a_6. Ce sous-ensemble est aussi appelé un schéma et est noté K. Cette notion de schéma est importante pour représenter un ensemble d'allèles co-adaptés comme nous le verrons plus tard. Soit un schéma K tel que $i_1 < i_2 < ... < i_k$ soient les positions où il n'y a pas de . La longueur l(K) est définie par la valeur $i_k - i_1$. Par exemple, a_3 a_6 a_9 est de longueur six.

Voici à présent la structure d'un algorithme génétique. Il y a quatre phases.

1 Initialisation : t = 0 et choix de M génotypes pour former B(0)

2 Sélection des génotypes en fonction de leur adéquation pour former B'(t)

3 Variation : application des opérateurs génétiques aux génotypes de B'(t) pour former B(t+1)

4 t=t+1 et retour à 2.

Ce petit aperçu nous suggère que le comportement des algorithmes génétiques dépend de différents choix : les paramètres (M,n), l'adéquation, la représentation d'un génotype. Les problèmes liés à ces choix seront étudiés dans la partie implémentation. Nous allons cependant d'abord expliquer les phases deux et trois en supposant ces choix faits. Le but recherché de ces deux phases est qu'au fur et à mesure des générations (c'est-à-dire des cycles des différentes phases) croissent la proportion d'allèles ou d'ensemble d'allèles qui améliorent l'adéquation des génotypes.

4.4. Sélection

Le but de cette étape est de déterminer les génotypes qui peuvent se reproduire et dans quelles proportions ils peuvent le faire. Pour cela, une population fictive B'(t) est constituée, telle que chaque génotype de B(t) se retrouve dans B'(t) autant de fois qu'il peut avoir de descendants, ce nombre dépendant de son adéquation. Or, la taille des populations est constante. Il faut donc

attribuer M places en rapport avec l'adéquation. Pour cela, chaque $A_i \in B(t)$ a une chance égale à

$$f(A_i) / \sum_{j=1}^{M} f(A_j)$$

de prendre chaque place de B'(t). Le nombre attendu de descendants dans B'(t) d'un génotype A_i de B(t) est dès lors égal à

$$M\, f(A_i) / \sum_{j=1}^{M} f(A_j)$$

Si deux génotypes de B(t) sont tels que l'adéquation de l'un est deux fois plus grande que celle de l'autre, il y aura deux fois plus de descendants du premier que du deuxième. En outre (*) se réécrit

$$f(A_i) / \bar{f} \quad \text{avec } \bar{f} = \sum_{j=1}^{M} f(A_j)$$

f étant la moyenne de l'adéquation de la population. Le nombre de descendants d'un génotype est donc égal au rapport entre l'adéquation de ce génotype et la moyenne, ce qui correspond au but de la sélection.

Regardons à présent l'effet de la sélection sur les schémas. Soit $M_k(t)$ ($M'_k(t)$) le nombre d'instances du schéma K dans B(t) (B'(t)). Dès lors nous avons

$$M'_k(t) = \sum_{A_i \in K} f(A_i) / \bar{f} = (\bar{f}_K / \bar{f}) * M_k(t)$$

$$\text{avec } \bar{f}_K = \sum_{A_i \in K} f(A_i) / M_k(t)$$

Donc le nombre d'instances d'un schéma évolue suivant le rapport entre l'adéquation moyenne du schéma et celle de la population. Cela montre que la sélection modifie la proportion d'instances des schémas en fonction de leur adéquation. C'est ce mécanisme qui permet aux allèles co-adaptés, qui sont en fait représentés par des schémas, de se répandre dans la population.

Enfin, chaque génotype est une instance de 2^n schémas : il suffit de substituer un ou plusieurs allèles par . Il y a donc entre 2^n et $M\, 2^n$ schémas dans une population. En effet, vu que plusieurs génotypes peuvent appartenir à un même schéma, il n'est pas possible de déterminer le nombre exact. La phase de sélection traite donc implicitement un grand nombre de schémas. Ce phénomène est appelé " parallélisme intrinsèque " .

4.5. Variation

Le but de cette phase est d'introduire du changement dans les génotypes. Au niveau des schémas, cela veut dire, d'une part, introduire des nouveaux et, d'autre part, propager les " bons " schémas existants. Pour réaliser cette phase, des opérateurs génétiques sont utilisés : ils génèrent de nouveaux génotypes à partir d'anciens. Il y a habituellement trois opérateurs : le croisement, l'inversion et la mutation. Remarquons que suivant l'application traitée, il est parfois préférable de modifier ces opérateurs standards tout en respectant leurs buts. Nous allons détailler ces trois opérateurs.

4.5.1. L'opérateur de croisement

Le principe de cet opérateur est d'engendrer des descendants à partir de deux parents. Vu la taille constante de la population, il ne peut y avoir que deux descendants, ni plus ni moins. Cet opérateur est inspiré du croisement en biologie qui consiste à échanger un segment d'allèles entre une paire de génotypes. Le principe est le suivant :

soient $A = a_1, a_2, ..., a_n$ et $B = b_1, b_2, ...,b_n$ les deux génotypes parents,
choisir un nombre z au hasard entre 1 et n-1,
former les deux génotypes descendants, C et D, en échangeant les allèles à droite de la position z, ce qui donne
$C = a_1, a_2, ..., a_z, b_{z+1}, ..., b_n$ et $D = b_1, b_2, ..., b_z, a_{z+1}, ..., a_n$

Pour autant que A et B diffèrent au moins pour un allèle, cet opérateur a deux effets :

— créer des nouveaux schémas
si $a_z = b_z$ et $a_{z+1} <> b_{z+1}$, alors seul C appartient au schéma ... $a_z b_{z+1}$ ... .
— conserver les schémas existants, par exemple a_1 a_2 ... .

De même que la sélection, cet opérateur traite un grand nombre de schémas en les conservant, les créant ou les détruisant. Il est possible de montrer que cet opérateur, de même que les deux autres, n'altère pas le travail de la sélection, à savoir la propagation des schémas dont l'adéquation est au-dessus de la moyenne. Certains donnent le nom de " théorème du schéma " au théorème qui montre ce fait.

Les schémas les plus courts (la notion de longueur a été définie) ont le plus de chances de survivre vu qu'il y a moins de possibilités que le point de croisement se trouve " à l'intérieur " du schéma, c'est-à-dire entre a_{i1} et a_{ik} si a_{i1} est le premier et a_{ik} le dernier allèle différent de dans le schéma. Cet aspect est important pour les applications. La meilleure

prolifération des schémas courts est primordiale pour la convergence de l'algorithme.

Notons enfin que cet opérateur de croisement est le plus puissant des trois et, dès lors, le plus utilisé. C'est en général, l'aspect clé de la réussite de l'implémentation d'un algorithme génétique.

4.5.2. L'opérateur d'inversion

Le but de cet opérateur est de modifier la position de certains gènes afin de permettre à des allèles de former un schéma court et d'augmenter leur chance de se déplacer ensemble lors du croisement. C'est cet opérateur qui permet à des groupes d'allèles co-adaptés de se répandre au travers de la population. Cependant, pour cela, il faut pouvoir retrouver à quel gène appartient tel allèle. Par exemple, si A = a_1, a_2, a_3, a_4, ..., a_n est transformé en B = a_1, a_3, a_2, a_4, ..., a_n , rien ne nous dit que le deuxième allèle de B se rapporte au troisième gène. La solution est d'introduire un index et alors un génotype a_1, a_2, ..., a_n est représenté par les permutations de $(1,a_1)$ $(2,a_2)$ ... (n,a_n) : il y a donc plusieurs représentations pour un même génotype.

Le principe de l'opérateur est le suivant :

soit A = (i_1,a_{i1}) (i_2,a_{i2}) ... (i_n,a_{in}) ,

choisir deux nombres x et y au hasard entre 0 et n+1 tels que $x \leq y$,

former un nouveau génotype en inversant le segment strictement compris entre les positions x et y, ce qui donne

B = (i_1,a_{i1}) (i_2,a_{i2}) ...(x,a_x) $(y-1,a_{y-1})$ $(y-2,a_{y-2})$ ... $(x+1,a_{x+1})$ (y,a_y) ... (i_n,a_{in})

Cette nouvelle représentation est incompatible avec l'opérateur de croisement tel qu'il a été décrit. En effet, si A = $(1,a_1)$ $(2,a_2)$ $(3,a_3)$ et B = $(1,b_1)$ $(3,b_3)$ $(2,b_2)$ sont les parents et si le croisement s'effectue à la position deux, cela donne comme descendants C = $(1,a_1)$ $(2,a_2)$ $(2,b_2)$ et D = $(1,b_1)$ $(3,b_3)$ $(3,a_3)$. Il est clair que ni C ni D ne sont valables. Plusieurs solutions sont possibles. Une d'entre elles est de réarranger les génotypes par ordre croissant, puis d'effectuer le croisement et enfin de remettre les gènes dans leur ordre initial. Cependant, cette représentation est plus lourde à manipuler et entraîne donc un surcoût. De plus, l'effet de cet opérateur est proche de celui de croisement, ce qui a pour conséquence qu'il est moins souvent utilisé tel quel.

Il y a certains cas où cet opérateur peut être utilisé sans avoir besoin de la représentation avec index. Cela arrive si les gènes ont la même signification et qu'ils prennent leurs valeurs dans le même ensemble d'allèles. Par exemple, pour le voyageur de commerce, un génotype représente les villes dans leur ordre de visite. Dès lors, chaque gène est porteur d'une des villes. Remarquons que dans ce cas, le but initial de construire des schémas courts est remplacé par le but de

l'opérateur de croisement : générer des instances d'anciens et de nouveaux schémas.

4.5.3. L'opérateur de mutation

Le but de cet opérateur est de réintroduire du matériel génétique perdu. En effet, les deux opérateurs précédents répandent les schémas et les génotypes les plus performants et donc en éliminent d'autres. Cela a cependant comme conséquence que certains allèles soient perdus parce qu'ils étaient mal placés. Rien ne prouve que ces allèles ne pourraient pas contribuer à améliorer l'adéquation s'ils étaient associés à d'autres allèles qu'à ceux avec lesquels ils l'étaient lors de leur destruction. De plus, il se peut que certains allèles ne soient pas présents au départ. L'initialisation ne garantit pas, en effet, de fournir tout le matériel génétique existant. Le but de la mutation est de réintroduire ces allèles car les deux autres opérateurs en sont incapables. Le principe est le suivant :

soit $A = a_1, a_2, ..., a_n$,

pour i = 1 à n, remplacer a_i par un autre allèle du gène i avec une probabilité P_M.

Cet opérateur est donc un opérateur de fond qui permet de conserver toute la diversité d'une population. La probabilité P_M est en général faible car le but est d'introduire du changement et non de bouleverser la situation existante.

4.6. Algorithme

A présent, nous avons décrit les phases deux et trois de la structure d'un algorithme génétique. Nous pouvons rassembler toutes ces informations pour décrire un algorithme en pseudo-pascal.

Signalons que A_1 représente un génotype

B(t) représente une population de M génotypes

t représente le temps et est en fait un compteur d'itérations.

```
Procédure algorithme_génétique;
begin
    initialisation (B(0));
      (*choix de M génotypes*)
    t:=0;
      repeat
      choix_descendants (B(t),B'(t));
            (* consiste à déterminer le nombre de descendants de chaque génotype*)
      for l:=1 to M do
          begin
          sélection(A_l, A'_j);
                (* choix d'un génotype A'_j de B'(t) pour faire partie de B(t+1)*)
          if (l est pair) then croisement (A_l, A_l-1, P_C);
          inversion (A_l,P_I);
          mutation (A_l, P_M)
          end;
      k:=k+1
      until critère d'arrêt.
```

Voyons à présent ce que recouvrent les différentes parties de cet algorithme.

Tout d'abord, l'initialisation de la population revient à choisir M solutions initiales de la même manière que pour le recuit simulé et la recherche tabou. De même, le critère d'arrêt a le même but que lors des autres méthodes.

La phase deux de sélection est en fait réalisée ici en deux fois. Il y a tout d'abord la partie choix_descendants qui permet de constituer la population B'(t) en calculant le nombre de

descendants de chaque génotype de B(t) d'après les formules vues au point 4.4. . La partie sélection quant à elle, choisit dans B'(t) un génotype pour lui appliquer les opérateurs génétiques et le mettre dans B(t+1).

En fait, la population B'(t) n'est pas indispensable. Il suffit de constituer un vecteur qui contienne le nombre voulu de descendants de chaque génotype. Dès lors, la partie choix_descendants détermine ce vecteur, tandis que la partie sélection l'utilise pour choisir dans B(t) les génotypes qui formeront B(t+1).

Aux trois opérateurs génétiques ont été ajoutées des probabilités P_C, P_I et P_M. P_M a déjà été introduit tandis que les P_C et P_I sont là pour indiquer qu'il est possible de fixer une probabilité d'application des deux opérateurs. Cela permet de mieux réguler les effets des opérateurs.

Avant de terminer, il nous importe de souligner un point. Il existe plusieurs versions de l'algorithme génétique. En fait, cet algorithme est fort dépendant de l'application traitée. Dès lors, suivant les cas, l'un ou l'autre opérateur génétique ne sera pas utilisé ou sera modifié. Par exemple, Holland utilise un croisement un peu différent. Au lieu de générer deux descendants, il n'en crée qu'un et garde un des parents.

D'autres exemples se trouvent entre autre dans [7] Les modifications apportées aux opérateurs y sont plus importantes. Cela nous amène à relever les points qui caractérisent tout algorithme génétique. Ce sont

1. une représentation des solutions sous forme de génotype

2. un moyen de créer une population initiale

3. une fonction d'évaluation qui caractérise les génotypes selon leur " adéquation "

4. un processus de sélection qui reproduit les génotypes en fonction de leur adéquation

5. des opérateurs génétiques qui modifient la composition des génotypes durant la reproduction

6. des valeurs pour les différents paramètres ($M,P_C,P_I,P_M,...$)

Notons qu'à notre avis, ce sont les points un et cinq qui sont les plus importants à trouver lorsqu'une application doit être résolue par un algorithme génétique. Nous verrons lors de l'implémentation ce que ces points sont devenus pour les deux applications traitées.

4.7. Implémentation

Nous expliquons, dans un premier temps, les points communs aux deux applications, c'est-à-dire la phase de sélection et le critère d'arrêt, puis nous passerons en revue les autres points pour chaque application séparément.

Le critère d'arrêt s'inspire du recuit simulé et de la recherche tabou. En fait, une variable contient la valeur minimum de la fonction d'évaluation des M génotypes à la fin de chaque reproduction. L'algorithme s'arrête lorsque cette variable reste inchangée pendant un certain nombre d'itérations. Notons que ce nombre doit être plus élevé que pour les deux autres méthodes étudiées. Cela est du au temps de propagation des schémas à travers la population.

En effet, lorsqu'un génotype a trouvé une meilleure solution, il faut lui laisser le temps de " communiquer " avec les autres génotypes pour voir si, sur base de ces nouvelles informations, ces derniers ne peuvent aussi s'améliorer.

La phase de sélection s'effectue en deux fois comme indiqué dans l'algorithme.

Dans un premier temps, le but est de calculer le nombre de descendants pour chaque génotype. Cependant, les formules données au point 4.4. sont valables pour une maximisation. Changer la fonction d'évaluation de signe ne convient pas , vu que celle-ci doit être positive. Dès lors, nous nous sommes tournés vers l'inverse de la fonction.

Nous avons alors rencontré un autre problème : celui de la différenciation.

En effet, il faut que le procédé de calcul du nombre de descendants soit suffisamment fin pour reconnaître les bons génotypes. Or, prendre l'inverse de la fonction d'évaluation risque de conduire à des nombres trop petits et trop proches l'un de l'autre. Le choix s'est dès lors porté sur l'inverse d'une puissance de la différence entre la fonction d'évaluation et une valeur inférieure au minimum recherché. Pour le voyageur de commerce, l'approximation de la distance minimale donnée au point 1.3.1. a été prise comme valeur tandis que pour la coloration de graphe, cette valeur est zéro.

Il reste encore un problème typique des algorithmes génétiques. C'est que la comparaison de l'adéquation d'un génotype à la moyenne de la population ne donne pas un nombre entier. Or le nombre de descendants doit être entier. Il " suffit " d'arrondir, mais cela ne peut être fait n'importe comment. Il est en effet reconnu que ce problème est en partie lié au problème de convergence prématurée des algorithmes génétiques vers des optima locaux. C'est pourquoi nous avons essayé deux manières d'arrondir.

La première consiste à prendre l'arrondi commercial, ce qui revient à prendre la partie entière du nombre auquel un demi a été ajouté. Pour avoir au moins M descendants, le dernier génotype se voit attribuer le nombre de descendants manquants, ce nombre n'étant pas toujours en relation avec son adéquation. Il arrive par exemple, que plus de M places aient été attribuées. Le dernier se voit, dans ce cas, privé de descendant. Notons que c'est alors la deuxième partie de la sélection qui s'arrange pour qu'il y ait exactement M descendants.

La deuxième manière consiste à prendre les parties entières des nombres calculés. Cela amène évidemment à moins de M descendants. Les dernières places sont alors attribuées en triant les parties décimales des nombres. Les premiers dans cet ordre voient alors leur nombre de descendants augmenté de un.

Le deuxième temps de la sélection est alors très simple. Il suffit de choisir un génotype dont le nombre de descendants n'est pas nul; puis une fois la reproduction effectuée, de diminuer de un ce nombre, puisque le génotype a eu droit à un descendant.

C'étaient les deux points communs aux applications. Voyons à présent les autres points pour le voyageur de commerce, puis pour la coloration de graphe.

4.7.1. Le voyageur de commerce.

Pour le voyageur de commerce, l'implémentation est assez simple. Seul l'opérateur de croisement demande un peu plus de recherche.

La représentation d'une solution est en fait la même que pour la recherche locale. Il s'agit des villes dans leur ordre de visite, chaque ville ayant reçu un numéro. Dès lors, l'initialisation de ces solutions et la fonction d'évaluation ne posent plus de problème. La différence par rapport à la recherche locale, c'est qu'il faut gérer M génotypes à la fois. Par exemple, ils ne peuvent avoir la même configuration de départ.

Signalons que cette représentation donne un sens à la prolifération des schémas courts qui améliorent l'adéquation. En effet, cela correspond à répandre un groupe de villes à visiter dans un certain ordre, tel que cette visite soit peu " coûteuse " quant à la fonction d'évaluation, c'est-à-dire soit de courte distance.

Passons aux opérateurs génétiques. Pour commencer, signalons que la mutation n'a pas de sens pour cette application. En effet, à tout moment, chaque génotype possède les N villes et il n'y a donc pas de matériel génétique qui se perde.

L'opérateur d'inversion quant à lui peut être appliqué comme décrit au point 4.5.2. et de plus, sans devoir ajouter d'index. En effet, chaque allèle représente une ville, et donc inverser un partie d'un génotype redonne une solution. En effet si A= 1,2,3,4,5,6,7,8,9,10 et que les points d'inversion sont x=3 et y=7, cela donne B= 1,2,3,6,5,4,7,8,9,10. Remarquons que cette inversion revient en fait à appliquer le 2-change de LIN. Pour s'en convaincre, il suffit de comparer cet exemple avec celui du point 1.4.5.1. avec x=p et y=q.

Il reste enfin l'opérateur de croisement. Contrairement à l'inversion, l'opérateur décrit au point 4.5.1. ne peut s'appliquer ici. En effet, si A= 1,2,3,4,5,6,7,8,9,10 et B= 1,2,3,6,5,4,7,8,9,10 et si le point de croisement x =5, cela donne C= 1,2,3,4,5,4,7,8,9,10 et D= 1,2,3,6,5,6,7,8,9,10.

Ces deux génotypes ne représentent bien sûr pas des solutions réalisables. Il nous faut donc trouver un autre moyen.

L'opérateur de croisement choisi est le suivant. Un sous-tour est choisi dans le premier génotype. Nous regardons ensuite si les villes qui forment ce sous-tour ne forment pas aussi dans le deuxième génotype, un autre sous-tour. Dans ce cas, il y a échange de ces sous-tours. Regardons sur un exemple, ce que cela donne :

Soit A= 1,2,3,4,5,6,7,8,9,10 et B= 9,7,3,6,4,5,2,8,1,10. Le sous-tour 4,5,6 de A se retrouve dans B en 6,4,5 tandis que le sous-tour 1,2,3 est introuvable dans B. Dès lors, le croisement donne C= 1,2,3,6,4,5,7,8,9,10 et D= 9,7,3,4,5,6,2,8,1,10.

Notons pour terminer cette explication des opérateurs génétiques, que ceux-ci résultent d'un choix fait par l'utilisateur et que nous ne prétendons pas que ce soit la seule manière de faire.

Nous allons maintenant passer à la coloration de graphe où nous verrons que le choix de la représentation des solutions et des opérateurs est moins facile que pour le voyageur de commerce.

4.7.2. La coloration de graphe

Deux approches différentes ont été envisagées. La première s'inspire du voyageur de commerce, mais les résultats obtenus sont très mauvais. C'est pourquoi une seconde approche a été prise en compte, sans beaucoup plus de succès.

La première version consiste à ranger les sommets dans un certain ordre de la même manière que les villes pour le voyageur de commerce.

Le sens de cet ordre est donné par la fonction d'évaluation. C'est une pratique courante dans

les algorithmes génétiques. Il y a ainsi une sorte de décodage du génotype pour calculer son adéquation.

En effet, la fonction d'évaluation en tant que telle a été expliquée pour la recherche locale. C'est celle qui attribue des poids décroissants aux couleurs et qui, dès lors, doit être maximisée. (voir le point 1.4.5.2.)

Il reste le décodage. Celui-ci revient en fait à créer des ensembles indépendants. Pour cela, les premiers sommets dans l'ordre donné par le génotype et qui n'ont pas d'arête entre eux, forment le premier ensemble, et ainsi de suite. Cependant, nous avons fixé un nombre limite de couleurs. C'est pourquoi, le dernier ensemble contient les derniers sommets, même s'ils ne sont pas indépendants. Cela est illustré sur la figure 4.1.

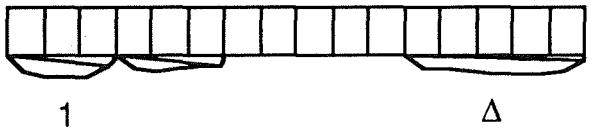

Figure 4.1. : Décodage d'un génotype.

Les ensembles 1 à Δ-1 sont indépendants.

Ce décodage essaye de donner sens à la notion de schéma. Ainsi, des sommets voisins dans un génotype sont co-adaptés s'il n'y a pas d'arête entre eux.

Les opérateurs, quant à eux, ont été repris de l'algorithme génétique appliqué au voyageur de commerce.

Cette implémentation a donc essayé de réutiliser ce qui avait été fait pour une autre application. Comme nous le verrons dans les tests, cela s'avère très mauvais. C'est là une particularité des algorithmes génétiques. C'est que leur implémentation dépend de l'application.

La deuxième version se base aussi sur une partition de sommets. La grande différence, cependant, réside dans le fait que ces ensembles seront traités par les opérateurs eux-mêmes et non plus par le décodage.

Le représentation de base reste la même. Le génotype contient les sommets dans un certain ordre. Cependant, à cela, s'ajoute un ensemble de bornes qui délimitent les ensembles. Un exemple est illustré à la figure 4.2. où les b_i représentent les bornes. Dès lors, les sommets colorés en j sont ceux qui se trouvent à partir de la borne b_j, incluse, jusqu'à la borne b_{j+1}, exclue.

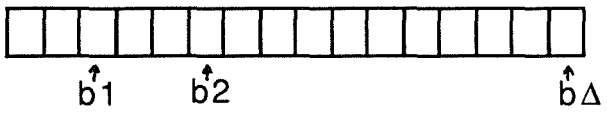

Figure 4.2. : Ajout de bornes à un génotype.

Dès lors, si $b_j = b_{j+1}$ cela signifie qu'aucun sommet ne possède cette couleur.

La fonction d'évaluation est égale à l'addition du nombre de couleurs utilisées et du nombre d'arêtes monochromes multiplié par un facteur de pénalité.

Comme pour la première version, la notion de prolifération des bons schémas courts a un sens. Il s'agit d'ensembles de sommets indépendants.

Il nous reste à expliquer les opérateurs génétiques choisis. La mutation trouve un sens ici. Il s'agit de réintroduire des couleurs perdues. En effet, les bornes doivent respecter une contrainte supplémentaire. En fait, il n'est pas permis d'utiliser la couleur j+1 si la couleur j n'est pas utilisée. Cela revient au même pour la fonction d'évaluation et cela permet de trouver les couleurs non-utilisées à la fin. Dès lors, réintroduire une couleur peut s'effectuer en attribuant au dernier sommet, la première couleur non-utilisée. Cela se fait en diminuant de un la valeur de la borne de cette couleur.

L'inversion a été conservée telle que dans la première version. Cet opérateur inverse dès lors un segment du génotype. Cela se fait sans changer les bornes. Le descendant possède donc le même nombre de couleurs, mais pas le même nombre d'arêtes monochromes, sauf cas exceptionnel.

L'opérateur de croisement tient compte des " bons " ensembles. En effet, à chaque ensemble d'un génotype est associé le nombre d'arêtes qui relient deux sommets de cet ensemble (ce sont en fait les arêtes monochromes). Dès lors, l'opérateur de croisement va d'abord aller chercher les sommets des ensembles indépendants du premier génotype, puis du deuxième. Puis, il fera de même avec les ensembles comportant une arête monochrome, et ainsi de suite jusqu'à former un descendant. Pour cela, bien entendu, avant d'accepter un sommet, il faut vérifier qu'il ne soit pas déjà dans le nouveau génotype. Cet opérateur est schématisé à la figure 4.3.

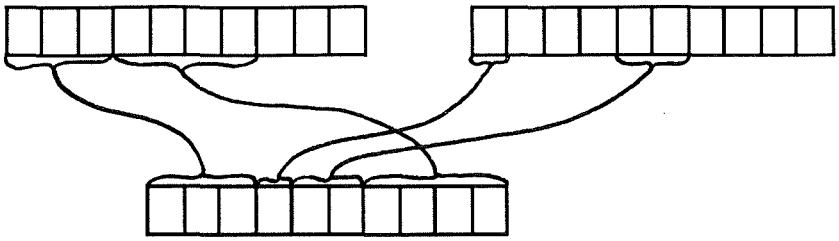

Figure 4.3. : Opérateur de croisement.

Pour le deuxième descendant, il suffit d'inverser l'ordre de prise en compte des deux parents. Cela donnera, en effet, un résultat différent.

Nous pouvons, à présent, passer aux résultats des différents tests effectués pour les deux applications.

Pour le choix des paramètres, nous mentionnons les résultats de Grefenstette [16]. Il a en fait utilisé un méta-algorithme génétique avec comme méta- génotype, les différents paramètres d'un algorithme génétique. La taille de la population pouvait varier entre 10 et 160 , par incrément de 10, le taux de croisements entre 0,25 et 1 , par incrément de 0,05 , et le taux de mutations entre 0 et 1 avec croissance exponentielle. Ses résultats qui rejoignent ceux de Dejong [16] donnent des valeurs standards pour les paramètres (M=50; P_C=0,6; P_M=0,001), ces valeurs devant " assurer une bonne recherche pour une variété de problèmes.

Nous supposons que l'absence de probabilité d'inversion provient du surcoût qu'entraîne l'opérateur d'inversion, dans bien des cas, à cause de l'ajout de l'index.

4.8.Résultats

Nous allons à présent effectuer les comparaisons concernant les algorithmes génétiques dans les mêmes conditions que les deux autres méthodes. Le comportement de ces algorithmes génétiques dépend de six paramètres : M, le nombre de génotypes; P_C, P_I et P_M, les probabilités liées aux opérateurs génétiques; Na, le nombre d'arrêt, et enfin Nmax, le nombre maximum d'itérations. Rappelons d'abord que Na correspond au nombre d'itérations pendant lesquelles la valeur minimum doit rester constante pour mettre fin à la méthode, et ensuite que Nmax est lié au temps maximum d'exécution.

4.8.1. Voyageur de commerce.

Le problème de base est toujours le même: cinquante villes situées dans le carré unité. Comme nous l'avons expliqué, l'opérateur de mutation n'a pas de sens et il reste donc cinq paramètres. De plus, la première version de l'algorithme génétique n'utilise que l'opérateur d'inversion avec une probabilité fixée à 0,40. La première comparaison porte donc sur les trois autres paramètres. La série de tests est la suivante :

G1	M = 40	Na = 30	Nmax = 500
G2	M = 40	Na = 30	Nmax = 1000
G3	M = 40	Na = 50	Nmax = 1000
G4	M = 40	Na = 70	Nmax = 1000
G5	M = 50	Na = 70	Nmax = 1000

Les résultats sont présentés par le tableau 4.1. . L'amélioration continue de G1 à G4 est évidente et permet de conserver 70 et 1000 comme valeurs des deux paramètres.

G1	G2	G3	G4	G5
6,9366	6,2265	5,9294	5,8962	8,5075
7,0115	6,6090	6,0571	5,8495	6,0155
8,5596	7,4423	7,4423	7,4423	6,4285
13,5982	13,5982	13,5982	7,4118	7,6903
8,9490	8,9490	5,8846	5,8846	6,7969
12,1370	12,1370	7,0459	7,0459	6,6942
10,0885	10,0885	10,0885	10,0885	7,4726
9,6223	9,6223	6,7420	6,7420	6,0633
6,2714	6,1000	5,7892	5,6799	6,2413
8,8739	8,8739	8,2995	6,7156	6,9229
9,2048	8,9647	7,6877	6,8756	6,8833

Tableau 4.1. : Comparaison d'algorithmes génétiques.

La comparaison entre G4 et G5 est plus délicate. Derrière une même moyenne de 6,88 se cachent des comportements fort différents. En effet, G4, par rapport à G5, obtient quatre résultats (contre 0) en dessous de 6 et quatre autres résultats (contre trois) au-dessus de 7. Les résultats sont donc plus dispersés en G4 qu'en G5.

Nous avons, dès lors, choisi de continuer avec les paramètres de G4. Nous espérons, en effet, qu'en modifiant quelque peu l'algorithme, nous éviterons ce problème de convergence

prématurée, et dès lors, que les bons résultats de G4 seront prépondérants.

Remarquons que, lors de cette première série, nous avons utilisé la quatrième puissance dans le calcul du nombre de descendants. Nous avons aussi essayé de prendre la deuxième puissance, mais cela donne une valeur moyenne de 11,31 pour les mêmes valeurs de paramètres que G4. Nous avons donc abandonné cette possibilité.

Pour la deuxième comparaison, nous introduisons l'opérateiur de croisement. Notons que nous ne fixons pas de probabilité P_c. En fait, chaque fois que l'opérateur est appliqué, nous choisissons un sous-tour dans le premier génotype. Le fait que celui-ci se retrouve ou non dans l'autre parent, décide de l'application ou non de l'opérateur. La probabilité P_c varie donc d'un test à un autre.

Jusqu'à présent, c'est l'arrondi commercial qui a été utilisé. Le test G8 essaye l'autre manière avec le tri des parties décimales.

Cela donne la série des tests suivants :

G6 m = 40 Na = 30 Nmax = 1000
G7 m = 40 Na = 70 Nmax = 1000
G8 m = 40 Na = 70 Nmax = 1000

Les résultats sont présentés par le tableau 4.2.

G4	G6	G7	G8
5,8962	8,9577	7,2336	5,8372
5,8495	11,6828	6,3361	6,4658
7,4423	11,1511	6,3232	7,3184
7,4118	9,7960	6,7145	6,6531
5,8846	10,8791	6,2236	5,8381
7,0459	9,3490	5,9617	6,4426
10,0885	10,0399	10,0040	7,9606
6,7420	10,5328	10,5328	6,0977
5,6799	9,1543	6,2749	6,4680
6,7156	8,8584	7,3371	6,6822
6,8756	10,0401	7,2977	6,5764

Tableau 4.2. : Comparaison d'algorithmes génétiques.

Les tests G6 et G7 donnent de moins bons résultats que, respectivement, G2 et G4. L'ajout de l'opérateur de croisement apparaît donc avoir, à première vue, un effet négatif.

Le changement dans le calcul des descendants a, par contre, une influence bénéfique. Il améliore nettement le résultat de G7. G8 est le premier test qui ne comporte aucune valeur au-dessus de 8. Cependant, les valeurs obtenues restent encore éloignées par rapport à celles des autres méthodes.

La troisième comparaison porte sur l'initialisation et sur le paramètre P_c. Notons que les paramètres Na et Nmax restent à présent fixés aux valeurs 70 et 1000. Les différentes initialisations utilisées sont celles expliquées lors de la recherche locale. Il s'agit de choisir la ville la plus proche, soit dans un groupe de 10 (G9) , soit de 20 (G10) , soit parmi toutes les villes (G11).

G8	G9	G10	G11
5,8372	6,2770	6,1680	6,3814
6,4658	6,4987	6,4231	5,9656
7,3184	7,1129	6,7765	6,4691
6,6531	7,1551	6,6294	6,7664
5,8381	6,2344	6,0215	5,8505
6,4426	6,2522	5,7945	5,8257
7,9606	7,0117	6,6481	6,7543
6,0977	6,2699	6,6292	5,6149
6,4680	6,7962	6,4740	5,6673
6,6822	7,1588	6,3677	6,5638
6,5764	6,5767	6,3932	6,1859

Tableau 4.3. : Comparaison d'algorithmes génétiques.

D'un autre côté, nous avons essayé de maîtriser la paramètre P_c.

Pour rappel, jusqu'à présent, l'opérateur de croisement chosissait un sous-tour dans un des parents et essayait de le retrouver dans l'autre parent. Ce que nous avons réalisé, c'est faire varier le nombre de sous-tours essayés par application de l'opérateur. Ce nombre était donc maintenu à un.

Dans la nouvelle version, il peut changer. Pour contrôler les variations de ce nombre, nous introduisons deux compteurs qui permettent d'estimer le taux de croisements effectués. Lorsque cette estimation dépasse de plus de cinq pourcents le taux voulu, le nombre de sous-tours diminue de un. De même, lorsqu'il est trop bas, ce nombre de sous-tours augmente. Les trois tests effectués se distinguent par la valeur du taux de croisements voulu. Les valeurs essayées sont 20% (G12), 30% (G13) et 40% (G14). Signalons encore que ces trois tests utilisent l'initialisation de G11.

Les résultats sont présentés par le tableau 4.4

G11	G12	G13	G14
6,3814	6,0185	6,0758	6,0717
5,9656	5,6023	5,6629	5,6304
6,4691	7,0465	6,4872	6,6738
6,7664	6,7094	6,6069	6,3778
5,8505	6,2539	5,7952	7,0154
5,8257	5,6959	5,8153	6,8221
6,7543	6,5356	6,4496	6,7154
5,6149	5,5656	5,8562	6,2627
5,6673	6,4474	5,7537	5,7025
6,5638	6,4274	6,7972	6,4070
6,1859	6,2302	6,1300	6,3679

Tableau 4.4. : Comparaison d'algorithmes génétiques.

Des trois manières d'initialiser, c'est la troisième qui donne les meilleurs résultats, même par rapport à G8. Remarquons ici, que l'effet de ces initialisations a été jugé négatif pour la recherche tabou et le recuit simulé.

Leur réussite dans les algorithmes génétiques peut paraître étonnante à première vue. Mais il ne faut oublier qu'il y a ici M génotypes et donc M solutions initiales. Le risque qu'une solution initiale amène à un optimum local peut donc être minimisé, ce qui n'est pas le cas des autres méthodes qui dépendent d'une seule solution initiale.

Pour ce qui est de l'opérateur de croisement, le taux de 30% est le seul à apporter une amélioration. Signalons que nous avons calculé le taux effectif de croisement. Il est de 75% pour G11, de 24% pour G12, de 34% pour G13 et de 45% pour G14. Nous pouvons donc conclure qu'un taux modéré de croisements est à recommander.

Finalement, une dernière comparaison a été effectuée. En effet, les premiers essais de l'opérateur de croisement ont été négatifs. Ce n'est que le changement dans la manière d'arrondir qui a permis alors d'améliorer l'algorithme génétique. Dès lors, nous avons testé l'algorithme utilisé par G13 en lui retirant l'opérateur de croisement. C'est donc aussi l'algorithme de G4 avec changement d'arrondi. Cela donne le test G15.

Après cela, nous avons changé la valeur de PI pour la fixer à 0,5 (G16) . Pour finir, nous avons modifié le critère d'arrêt en obligeant à chaque fois d'effectuer mille itération. Pour cela,

nous avons repris P_I=0,4 5 (G17).

Les résultats sont présentés par le tableau 4.5.

G13	G15	G16	G17
6,0758	5,9725	5,6881	5,9321
5,6629	5,5623	5,6199	5,5623
6,4872	6,5047	6,4672	6,5047
6,6069	6,4767	6,6140	6,4037
5,7952	5,8811	5,8088	5,8676
5,8153	5,9849	6,0710	5,9362
6,4496	6,4599	6,6538	6,4590
5,8562	6,1416	5,9019	5,9037
5,7537	5,6422	5,6845	5,6422
6,7972	6,3635	7,5691	6,2958
6,1300	6,0989	6,2078	6,0507

Tableau 4.5. : Comparaison d'algorithmes génétiques.

Les résultats de G15 sont assez surprenants. Ils indiquent, en effet, un léger avantage par rapport à G13. Soulignons bien que cela ne signifie pas que l'opérateur de croisement qui doit permettre l'échange d'informations entre les génotypes est d'un effet négatif.

Cela signifie donc que c'est l'opérateur de croisement tel qu'il a été choisi et implémenté, ne répond pas à ce qui lui est demandé.

Le test G16 montre simplement que la valeur 0,40 est meilleure que 0,50 pour P_I.

Le test G17 montre, quant à lui, que la valeur 70 est encore trop petite pour le nombre d'arrêt, et qu'il faut mieux fixer au départ le nombre d'itérations. Signalons que le nombre moyen d'itérations est de 401 pour G15. Pour ce qui est du temps d'exécution moyen, il est de 29 secondes pour G15, de 72 pour G17 et de 70 pour G13.

Cela montre que l'opérateur de croisement est " gourmand " en temps et que y ajouter la procédure d'arrêt de G17 risque d'entraîner des temps d'exécution assez longs.

4.8.2. Coloration de graphe

Les résultats obtenus pour la coloration de graphe se sont révélés catastrophiques. Même avec Δ=15, aucune coloration n'a pu être obtenue.

La première version utilisée est celle qui reprend des opérateurs utilisés pour le voyageur de commerce. L'échec de cette solution nous a peu surpris. Les opérateurs utilisés ne sont en effet pas adaptés au problème traité. Signalons qu'en moyenne, il restaient 140 arêtes monochromes dans le résultat final. Modifier les paramètres n'a que très peu modifié les résultats.

La deuxième version semble mieux adaptée à la coloration de graphe. Les opérateurs ont été construits en tenant compte du problème. Malheureusement, les résultats ne furent pas meilleurs.

Nous avons jugé inutile de détailler davantage les résultats obtenus, vu leur mauvaise qualité.

Il est certes possible de construire d'autres opérateurs génétiques. Nous sommes cependant sceptiques quant à leur efficacité. Le problème de la coloration des sommets d'un graphe nous paraît difficile à résoudre à l'aide des algorithmes génétiques.

4.9. Conclusion

Comme le recuit simulé, cette méthode provient d'une analogie avec un phénomène naturel, à savoir l'évolution des êtres vivants. Le problème principal réside dans la modélisation en opérateurs des transformations qui s'effectuent au niveau des chromosomes.

Car si le recuit simulé est une sorte de méta-algorithme qui se greffe sur la recherche locale et qui s'adapte assez facilement à différents problèmes, les algorithmes génétiques sont, quant à eux, plus complexes et leur réalisation dépend en partie de l'application traitée.

Trouver une représentation du problème sous forme de chromosomes et définir des opérateurs qui permettent un échange d'informations entre les génotypes afin de permettre la prolifération des bons schémas, sont, quant à eux, les deux problèmes les plus importants.

Le principal effort de recherche doit être fourni avant de passer à l'implémentation et la mise au point. Ces dernières ne posent, en effet, pas de problème. Mais, comme nous avons pu le constater lors des comparaisons effectuées, pour le voyageur de commerce, un opérateur peut avoir un effet négatif.

CONCLUSION

Le recuit simulé et les algorithmes génétiques sont deux méthodes non-déterministes qui modélisent toutes les deux un phénomène naturel. En ce qui concerne la première méthode, il s'agit d'un processus de physique en thermodynamique, tandis que pour la seconde, c'est de l'évolution naturelle qu'il est question.

Toutes les deux sont largement répandues et appliquées dans de nombreux domaines, dont entre autres, l'intelligence artificielle.

La recherche tabou, quant à elle, est de la même famille que le recuit simulé. Ces deux méthodes sont, en effet, basées sur la recherche locale. Cependant, la recherche tabou semble moins utilisée.

Le but de ce travail était de comparer ces trois méthodes en les appliquant à deux problèmes d'optimisation combinatoire.

Au niveau des résultats obtenus, c'est sans aucun doute, le recuit simulé qui s'est révélé le plus performant. C'est une méthode dont la mise en oeuvre ne pose aucun problème, même si la gestion de certains paramètres, qui déterminent son comportement, est un peu plus complexe.

Notre avis au sujet de la recherche tabou est plus mitigé. Certes, sa réalisation est assez simple et c'est la méthode qui s'est révélée être la plus rapide en temps d'exécution. Mais les résultats sont décevants. D'autant plus, qu'avec les éléments dont nous disposons, nous ne voyons pas comment nous pourrions améliorer cette recherche.

Le cas des algorithmes génétiques est différent. Cette méthode nécessite une recherche plus approfondie pour sa réalisation. Certes, l'échec en coloration de graphe montre que cette méthode ne s'adapte pas à tous les problèmes. Nous sommes persuadés, cependant, qu'il y a moyen d'améliorer les résultats obtenus pour le voyageur de commerce.

Pour terminer, signalons que subsistent diverses possibilités d'exploration , dont par exemple:

- utiliser les algorithmes génétiques pour optmiser les paramètres du recuit simulé.
- analyser le problème de la convergence prématurée des algorithmes génétiques.
- analyser l'influence des paramètres du recuit simulé.
- résoudre des problèmes de tailles plus importantes (il paraît préférable de travailler avec des ordinateurs plus puissants.) .

Beaucoup d'auters présentent une méthode en particulier. Peu les comparent entre elles. C'est ce que nous avons essayé de réaliser sans idée préconçue.

Chaque méthode a été l'objet des mêmes efforts de recherche et a engendré les résultats que nous venons de présenter.

BIBLIOGRAPHIE

[1] AARST, E.H., KORST, J., **Simulated Annealing and Boltzmann Machines**, John Wiley, Chichester, 1989.

[2] BOLLOBAS, B.,THOMASON, A., **Random Graphs of Small Order, Random Graphs '83**, Annals of Discrete Mathematics n°28, p.47-97, 1985.

[3] BONOMI, E., LUTTON, J.-L., **The N-City Travelling Salesman Problem : Statistical Mechanics and the Metropolis Algorithm**, SIAM Revue, Vol.26, n°4, p. 551-568, 1984.

[4] BONOMI, E., LUTTON, J.-L., **Simulated Annealing Algorithm for the Minimum Weighted Perfect Euclidean Matching Problem**, RAIRO Recherche Opérationnelle - Operations Research, Vol.20, n°3, p.177 à 197, 1986.

[5] BONOMI, E., LUTTON, J.-L., **Le Recuit Simulé**, Pour la Science, n°129, 1988.

[6] CERNY, V., **Thermodynamical Approach to the Traveling Salesman Problem : an Efficient Simulation Algorithm**, Journal of Optimization Theory and Applications, n°45, p.41-51, 1985.

[7] DAVIS, L., **Genetic Algorithms and Simulated Annealing**, Morgan Kaufmann, Los Altos, California, 1987.

[8] GLOVER, F., **Future Paths for Integer Programming and Links to Artificial Intelligence**, C.A.A.I. Report 85-8, University of Colorado, Boulder CO, 1985.

[9] HERTZ, A., de WERRA, A., **Using Tabu Search Techniques for Graph Coloring**, Computing, Springer Verlag, n°39, p.345-351, 1987.

[10] HOLLAND, J.-H., **Adaptation in Natural and Artificial Systems, An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence**, University of Michigan Press, Ann Arbor, 1975.

[11] KIRKPATRICK, S., GELATT, C.-D., VECCHI, M.-P., **Optimization by Simulated Annealing**, Science, n°220, p.671 à 680, 1983.

[12] KUIK, R., SALOMON, M., VAN WASSENHOOVE, L.N., MAES, J., **Linear Programming, Simulated Annealing and Tabu Search Heuristics for Multi Level**

Lotsizing with a Bottleneck, Erasmus Universiteit, Rotterdam, 1989.

[13] van LAARHOVEN, P.J.M. and AARTS, E.H., **Simulated Annealing : Theory and Applications**, D. Reidel, Dordrecht, Holland, 1987.

[14] PAPADIMITRIOU, C.H., STEIGLITZ, K., **Combinatorial Optimization : Algorithms and Complexity**, Prentice Hall, New York, 1982.

[15] SIARRY, P., DREYFUS, G., **La Méthode du Recuit Simulé : Théorie et applications**, I.D.S.E.T., Paris, 1988.

[16] SPIESSENS, P., **Genetic Algorithms : Introduction, Applications and Extensions**, A.I. MEMO n°88-19, Vrije Universiteit Brussel, 1988.

Table des matières