UNIVERSITE MOHAMED BOUDIAF DE M'SILA

Faculté des Mathématiques et de l'Informatique

Département de Mathématiques

MEMOIRE DE FIN D'ETUDE

Présenté pour l'obtention du Diplôme de **MASTER**

Domaine : Mathématiques et Informatique

Filière Mathématiques**.**

Option : Mathématiques

Par

Aiche khadidja

Sujet

Sur l'optimisation combinatoire

Devant le jury :

Mr. Dilmi mustapha	MCB. Univ de M'sila	Président
Mr. Selt omar	MCA. Univ de M'sila	Encadreur
Mr. Gagui bachir	MCA. Univ de M'sila	Examinateur

Promotion : 2018 / 2019

شكر وعرفان

أتوجه بالشكر والحمد لله سبحانه وتعالى الذي أعانني وسهل لي إكمال هذا العمل.

كما أتقدم بجزيل الشكر و العرفان إلى أستاذي المشرف " سالت عمر " الذي أرشدني وتابعني في إنجاز هذا العمل.

كما لا يفوتني أن أتقدم بالشكر إلى كل الأساتذة، والزميلات على ما قدموه لي.

إلى كل من ساهم في هذا البحث من قريب أو بعيد، أشكركم مرة أخرى.

إهـــداء

إلى الوالدين الكريمين حفظهما الله و أطال في عمرهما.

إلى أخواتي و إخوتي الأعزاء.

الى زوجي ونور حياتي وولدي العزيز.

الى عائلة زوجي من كبير الى صغير .

إلى كل صديقاتي ورفيقات دربي.

إلى كل الزملاء و الزميلات وكل من مد لي يد العون والمساعدة.

أهدي هذا الجهد العلمي

TABLE OF CONTENT :

Chapitre 3

Introduction

Introduction general:

Les méthodes de résolutions des problèmes d'optimisation combinatoire puissent dans toutes les techniques de l'optimisation combinatoires (programmation mathématique, programmation dynamique, théorie des graphes …).

Ces méthodes garantissent en général l'optimalité de la solution fournie.

Mais les algorithmes dont la complexité n'est pas polynomiale ne peuvent pas être utilisés, des problèmes de grande taille, d'où la nécessité de construire des méthodes de résolution approchée, qu'on appelle méthodes heuristiques, efficaces pour les problèmes NP- difficiles, dans la suite nous présenterons les méthodes de résolutions les plus connues.

Chapitre 1

1. Problèmes d'optimisation combinatoire:

1.1 Définition

La théorie de la complexité est construite à partir des problèmes de décisions.

Pour les problèmes d'optimisation, il s'agit d'optimiser une mesure sur un ensemble de solutions réalisables. Un problème d'optimisation Π sera défini par la donnée de:

- Un ensemble I d'instances.
- Une fonction **S** telle que pour tout **x ∈ I**, **S(x)** représente l'ensemble des solutions réalisables pour **x**.
- Une fonction **m** à valeur entière définie sur tous les couples **x ∈ I** et **y ∈ S(x)**. cette mesure **m** représente la fonction objectif d'une solution **y** l'instance **x**.
- Un objectif ∈ {min, max} précisant si Π est un problème de minimisation ou de maximisation.

A toute instance **x**, on associe une solution **y*(x)** et la valeur **m*(x)** définissant un optimum. Si Π est un problème de minimisation, **m*(x)=min {m(x, y) | y ∈S(x)}** et **y*(x)** est une solution vérifiant **m(x, y*(x)) = m*(x)** [RAP 02].

A un tel problème d'optimisation Π, on peut associer 2 versions différentes selon la nature du résultat attendu (fonction calculée par un algorithme de résolution):

- Π_C (par défaut Π), le problème de construction. Pour chaque instance **x**, on doit fournir en sortie la valeur **m*(x)** et une solution **y*(x)** réalisant cet optimum.

- Π_D, le problème de décision associé. Une instance de Π_D est une instance **x** de Π plus entier **K**. La question est: existe-il une solution **y** vérifiant **m(x, y) ≤ K** pour le problème de **min (resp. m(x, y) ≥ K pour max)**.

les problèmes d'optimisation ne sont pas des problèmes de décision, et ne peuvent donc appartenir à NP, à fortiori à NP-complet. La notion de réduction entre ces problèmes s'impose pour définir intuitivement q'un problème Π est au moins aussi difficile qu'un problème Π′ qui se réduit à Π.

Un problème Π′ se réduit à Π si l'on peut écrire une procédure pour résoudre Π′ en connaissant une sous-procédure pour résoudre Π. On peut alors définir la NP-difficulté de la manière suivante .

1.2 Définition:

Un problème Π est dit NP-difficile ou NP-dur, ssi seulement pour tout problème $\Pi_0 \propto \Pi$.

Pour un problème d'optimisation Π. On a clairement $\Pi_D \propto \Pi_C$. Pour cette raison on utilise en pratique la propriété suivante [RAP 02].

1.3 Propriété:

Soit Π un problème d'optimisation. Si son problème de décision associé $\Pi_D \in$ NP-complet, alors $\Pi \in$ NP-difficile.

La plupart des problèmes d'ordonnancement sont NP-difficile car ils sont des problèmes d'optimisation. Toutefois il existe quelques cas particuliers pour les quels on a trouvé des algorithmes polynomiaux, comme pour le problème :

2. La complexité d'un algorithme:

Evaluer la complexité d'un algorithme consiste à trouver en fonction de la taille des données, le nombre d'opération élémentaires nécessitées par cet algorithme.

3. Un algorithme polynomial:

Est un algorithme dont le nombre d'opérations élémentaires nécessaire pour résoudre un exemple de taille **n** est une fonction polynomiale en **n**.

4. Algorithme non polynomial:

Est un algorithme dont le nombre d'opérations n'est pas donné par un polynôme de **n**.

5. Un algorithme efficace:

On dit qu'un algorithme est efficace si le nombre des opérations nécessaires pour résoudre un problème est donné par une fonction polynomiale d'un paramètre caractérisant la taille du problème considéré.

5.1 Problème de décision:

Est un problème dont les résultats ne peuvent prendre que l'une des deux valeurs: VRAI ou FAUX.

- **La classe NP:**

La classe des problèmes NP (NP pour Non détermination Polynomiale) est la classe des problèmes des décisions qui peuvent être résolus par une machine de Turing non déterministe en temps polynomial.

Parmi la classe de problème NP, on distingue deux grandes classes:

- La classe des problèmes polynomiaux (la classe P) et la classe des problèmes NP- Complet (la classe NPC).

- **La classe P:** Un Problème est dit appartenir à la classe P s'il existe un algorithme polynomial pour le résoudre. On dit que les problèmes de la classe P sont faciles.

- **La classe NP-Complet:**

Un problème de décision est dit NP-Complet si tout problème de la classe NP peut se rendre polynomialement à lui.

- Tous les problèmes NP- difficiles ne sont pas de difficulté identique. On rencontre par exemple des problèmes qui ne peuvent pas être résolus en temps polynomial avec un codage binaire, mais qui peuvent l'être avec un codage unaire, ces problèmes sont dits NP-difficiles au sens ordinaire ou simplement NP- difficiles, les algorithmes pour cette classe de problèmes sont appelés pseudo-polynomiaux, pour d'autres problèmes, on peut ne pas trouver d'algorithme polynomial, quelque soit le codage, ceux-là sont dits NP- difficiles au sens fort.

Chapitre 2

1. Les méthodes exactes:

La résolution se ramène à l'exploration de l'ensemble des séquences possibles à travers d'un arbre de recherche. Le temps de calcul nécessaire d'une telle méthode augmente en général exponentiellement avec la taille du problème à résoudre. Elle sont issues de la recherche opérationnelle, le but de ces méthodes est de trouver en un temps de calcul, le plus court possible, la solution optimale du problème, ainsi la résolution se ramène à l'exploration de l'ensemble de séquences possibles au travers d'un arbre de recherche.

Pour améliorer l'efficacité de la recherche, on utilise des techniques pour calculer des bornes permettant d 'élaguer le plus tôt possible des branches conduisent à un échec, de plus on emploi des heuristiques pour guider les choix de variables et de valeurs durant l'exploration de l'arborescence. Parmi les méthodes exactes on distingue.

1.1 Enumération complète:

On recherche un ordonnancement optimale à travers tous les ordonnancements possibles, l'énumération complète génère les ordonnancements possibles. Dans le cas des problèmes d'ordonnancement non préemptif sur une machine, il existe **n!** séquences d'exécution des tâches possibles, par conséquent pour le problème de **m** machines correspondant il y a $\mathbf{(n!)^m}$ séquences d'exécution possible, ce nombre est très grand. Et bien qu'en pratique à cause des contraintes du problème, beaucoup

d'ordonnancement peuvent être inadmissibles et que des procédures d'élimination peuvent parfois être utilisées pour voir si la non optimalité d'un ordonnancement implique la non-optimalité d'autre, non encore gênés, le temps de résolution peut être prohibitivement long même si on utilise le plus puissant et le plus rapide des ordinateurs, il est donc clair que, chercher une solution optimale utilisant l'énumération complète n'est pas convenable même pour des problèmes de petite taille. C'est pour cela qu'il était nécessaire de faire appel à des méthodes intelligentes.

1.2 Analyse combinatoire: (Cette méthode examine l'effet d'un changement mineur sur une séquence particulière, sur la valeur de cette séquence).

L'analyse combinatoire peut mener à des algorithmes très efficaces qui produisent un ordonnancement optimal en un nombre prévisible d'étapes où de dernier grandit au plus polynomialement en fonction de la taille du problème, cette méthode examine l'effet d'un changement mineur sur une séquence particulière, sur la valeur de cette séquence.

1.3 Programmation dynamique:

C'est une méthode fondée sur une approche récursive, s'applique à n'importe quel problème qui peut être divisé en une séquence de problèmes, la solution d'un problème d'ordre **n** s'exprime simplement en fonction de solutions de problèmes d'ordre**(n – 1)**.

La méthode de la programmation dynamique est une méthode générique pour les problèmes d'optimisation et en particulier il a été prouvé que c'est un outil à la fois souple et puissant pour traiter les problèmes de l'ordonnancement **[ABD 87]**, elle a été appliquée pour la première fois aux problèmes d'ordonnancement par **CARP** et **[HEL 62]**, et ensuite par **[LAW 64]**.

Cette méthode consiste à décomposer le problème posé en sous problèmes (ou phases), puis à établir une équation récursive exprimant la solution ou du sous problème d'ordre **K** en fonction de celle du sous problème d'ordre **(K- 1)**. Ainsi, le dernier sous problème serait d'ordre **(n)**. en l'occurrence, c'est le problème à résoudre dans l'exemple ci-dessous nous illustrons l'approche de la programmation dynamique.

1// f_i, ou on minimise le coût totale $\sum f(C_J)$

Cette fonction récursive a été suggérée par **HEL** et **KARP** et ensuite **[Law 64]**

soit **J ⊆ I = {1, 2, 3……..,n}**un ensemble de taches et **f *(J)** le coût total minimal de l'ordonnancement des taches de **J** sur la période $\left[0, \sum_{i \in j} P_i\right]$,

l'objectif est de trouver **f*(J)** en résolvant les équations récursives.

$$\mathbf{f^*(J) = \min_{i \in j} \{f^*(J-\{i\}) + f_i \left(\sum_{i \in J} Pi\right)\}.}$$

qui sont initialisés par $\mathbf{f^*(\varnothing) = 0}$

La programmation dynamique est une méthode d'énumération implicite, meilleur que l'énumération complète, mais dont les besoins en espèce mémoire sont souvent très grands.

1.4 Méthode par séparation et évaluation:

Ces techniques ont été appliquées pour la première fois aux problèmes d'ordonnancement par LominickiI et Ignal..

Cette méthode représente les solutions possibles par un "arbre de recherche" qu'il faut explorer du haut en bas, en se servant d'une borne inférieure (Lower Bound), de la solution optimale, déterminée au préalable, on évolue à chaque nœud plus que la borne inférieure et n'explorer que celles qui améliorent cette borne inférieure, cette méthode permet donc de réduire le nombre de cas à explorer. Ce nombre dépend essentiellement de la qualité de la borne inférieure adoptée au départ.

- **L'arbre de recherche:**

L'arbre de recherche est la représentation schématique permettant de recenser systématiquement et explicitement toutes les **n!** permutations de **n** tâches. elle et apparut comme un ensemble de nœuds disposés en couches

hiérarchiques et numérotées respectivement de 0 à (n-1) du plus haut au plus bas niveau.

La couche 0 contient un seul nœud, dit nœud racine, qu'on note **S**. La couche **n-1** est composé de **n!** nœud dits nœuds feuilles, les nœuds des autres couches sont dits nœuds internes.

Chaque nœud est un ensemble d'ordonnancement. En particulier **S** est l'ensemble de tous les ordonnancements (Card(S) = n!), et tout nœud – feuille est un singleton nous désignons par I_y l'ensemble des taches déjà affectées dans le nœud **Y** ($I_S = \phi$ et $I_F = I$ si **F** est nœud – feuille) par 0 un nœud de la couche **i**, et par **D** un nœud de la couche **i+1** où **i ≠ n-1**

D est dit successeur (0) s'il en dérive directement c'est-à-dire que $I_D - I_0 = \{t\}$, où (t) est la tâche de branchement du nœud 0 au nœud **D** et que les tâches sont disposées identiquement dans **D** que dans 0.

Tout nœud possède **$(I - I_0)$** successeurs immédiats qui forment une partition de lui.

Un nœud est un descendant d'un autre s'il en dérive directement ou indirectement.

Tout nœud est le sommet d'un sous arbre de l'arbre de recherche.

Tout sous arbre contient tous les descendants de son sommet, par conséquent, pour supprimer un sous arbre, il suffit de supprimer son sommet.

En explorant l'arbre de recherche, la méthode par séparation et évaluation utilise une procédure de séparation pour partitionner successivement les nœuds.

Si le passage d'un nœud(0) à un successeur immédiat **D** est incompatible avec les contraintes du problème à résoudre, le nœud **D** sera supprimé. Sinon la méthode par séparation et évaluation permet de décider d'explorer ou de supprimer le sous-arbre de sommet **D**. Cet arbre est supprimé s'il a été jugé inutile de l'explorer car il ne contient aucune solution meilleur que celle disponible jusqu'à présent la méthode par séparation et évaluation est un exemple typique de l'approche d'énumération implicite, qui peut trouver une solution optimale en examinant systématiquement des ensembles de solutions admissibles.

- **Procédure de séparation:**

Cette procédure décrit la méthode utilisée pour partitionner un sous ensemble de solutions possibles, les procédures de séparations les plus usuelles sont:

- Branchement en arrière (backwards branching):

Les taches sont affectées une par une à partir de la fin.

- Branchement en avant (forwards branching):

Les tâches sont affectées une par une du début.

- A chaque étape, une tache est choisie pour être effectuée soit au début ou à la fin selon une certaine méthode heuristique basée sur les données des problèmes (voir Hairari [HAR 81], chapitre 8 et 10)

- A chaque étape, une tâche est choisie pour être affectée soit avant soit après une autre tâche. Une heuristique peut être utilisée pour déterminer cette paire de tâches (voir Hariri, [HAR 81]).

Pour minimiser le critère **f** d'un problème d'ordonnancement particulier, la méthode par séparation et évaluation utilise une procédure de calcul d'une borne inférieure dite fonction d'évaluation par défaut.

- **Procédure de calcul d'une borne inférieure:**

Il s'agit d'une fonction définie sur l'ensemble des nœuds de l'arbre de recherche, et qu'on note par **Lb** si **Y** un nœud quelconque, nous devrions être capable de calculer

Lb(Y) et avoir: $\forall y \in \mathrm{Y}\ Lb(\mathrm{Y}) \leq f(y)$

Parmi les méthodes de calcul de bornes inférieures pour les problèmes d'ordonnancement on distingue (po

- Relaxation des contraintes du problème.

- Relaxation lagrangienne des contraintes
- Relaxation "State Space" de la programmation dynamique ([CHR 81])
- Relaxation de l'objectif [ABD 87].

- **Borne supérieure:**

Si aucune solution admissible n'est connue, la borne supérieure **UB** est initialisée jusqu'à ce que la première solution admissible soit trouvée.

Pour un nœud **Y** n'ayant pas encore été supprimé, on calcule sa borne inférieure

Lb (Y). Si **Lb (Y) ≥ UB** alors on supprime **Y**, sinon **Y** est conservé en vue de l'explorer éventuellement ultérieurement.

L'arbre de recherche se réduit remarquablement lorsque les bornes inférieures sont aussi grandes que possibles pour poursuivre la recherche, le prochain nœud à explorer est choisi selon une stratégie de recherche.

- **Stratégie de recherche:**

Une stratégie de recherche permet de sélectionner dans l'arbre le prochain nœud à explorer, soit **Y** ce nœud cherché. Par convention le premier nœud à être exploré est le nœud racine **S**. généralement le nœud **Y** peut être choisi comme étant:

1- Le nœud ayant la plus petite borne inférieure.

2- Le nœud le plus récemment crée.

3- Le nœud ayant la plus petite borne inférieure parmi les nœuds des récemment crées

Les stratégies (2) et (3) sont la stratégie profondeur d'abord, alors que la stratégie (1) est la stratégie largeur d'abord.

La stratégie profondeur d'abord nécessite peut (très peut) d'espace mémoire, mais une quantité considérable de calculs. Par contre la stratégie largeur d'abord nécessite moins de calculs car elle choisi la branche à explorer d'une façon plus intelligente et donc, trouve une solution optimale plus vite. Malheureusement, cette performance exige des besoins plus grands en espace mémoire.

Quelque soit la stratégie de recherche utilisée, le déroulement de la méthode par séparation et évaluation sera terminée une fois que l'arbre de recherche est entièrement exploré et l'ordonnancement essai (trial Schedule) qui reste sera optimal.

- **Amélioration de l'efficacité:** il y a plusieurs façons pour améliorer l'efficacité de la méthode par séparation et évaluation, citons quelques ânes:

a/ Chois d'une stratégie de recherche: la stratégie utilisée est un déterminant important du chois temps nécessaire pour résoudre un problème, généralement largeur d'abord trouve un ordonnancement optimal plus vite qu'une stratégie profondeur d'abord [FRE 82]

b/ Choix de bornes inférieures:

La qualité des bornes inférieures est aussi un facteur important dans le temps de calcul nécessaire pour résoudre un problème, la borne inférieur **Lb (Y)** du nœud **Y** est bonne si elle

n'est pas trop petite que la plus petite valeur du critère d'optimisation au niveau de ce nœud, la méthode par séparation et évaluation trouve généralement un ordonnancement optimal après avoir examiner moins de nœuds avec bonnes bornes inférieures qu'avec mauvaises. Les bornes inférieures éliminant les nœuds qui sont en haut de l'arbre, ce qui réduit considérablement la recherche. Car lorsque un nœud situé en haut de l'arbre est éliminé, plusieurs nœuds subséquents sont éliminés au même temps.

c/ Règle de dominance:

Si on peut montrer qu'une solution optimale peut toujours être générée sans trouver un nœud particulier de l'arbre de recherche, ce nœud est dit dominé et peut être éliminé.

En résumé, les règles de dominance sont utiles dans la réduction du.

- Besoins en espace mémoire sur l'ordinateur
- Temps de calcul (car elles évitent des calculs pour les descendants de tout nœud éliminé).

d/ Initialisation de la borne supérieur (UB):

démarrer avec un ordonnancement presque-optimal (Near Optimal Schedule) peut entraîner l'élimination de beaucoup de nœuds lors des premières étapes de la recherche. On utilise des méthodes heuristiques, pour trouver un ordonnancement presque optimal, dans le cas échéant, un ordonnancement admissible.

Chapitre 3

1. Méthodes approchées:

Les méthodes approchées constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale, en effet, ces méthodes sont utilisées depuis longtemps par de nombreux praticiens.

Pour les grands problèmes NP-durs, il est même probablement impossible de trouver une solution optimale durant une durée de vie humaine, donc, dans le cas où on ne peut pas trouver un ordonnancement optimal au bout d'un temps raisonnable, on ne doit pas quitter l'analyse mais plutôt utiliser notre expérience pour trouver en temps polynomial une solution réalisable, la meilleur que possible.

Les méthodes approchées sont fondées principalement sur diverses heuristiques, souvent spécifiques à un type de problème, les métaheuristiques constituent une autre partie importante des méthodes approchées et ouvrent des voies très intéressantes en matière de conception des heuristiques pour l'optimisation combinatoire.

1.1 Les méthodes de voisinage:

Les méthodes de voisinage sont fondées sur la notion de voisinage.

Définition:

Soit **X** l'ensemble des configurations admissibles d'un problème, on appelle voisinage toute application $N : x \to 2^x$, on appelle mécanisme d'exploration du voisinage toute procédure qui précise comment la recherche passe d'une configuration: **S ∈ X** à une configuration **S′ ∈ N(s)**.

On dit que la configuration **S** est un optimum (minimum) local par rapport au voisinage **N** si:

$\forall S' \in N(S) : f(S) \leq f(S')$

Une méthode typique de voisinage débute avec une configuration initiale, et réalise ensuite un processus itératif qui consiste à remplacer la configuration courante par l'un de ses voisins en tenant compte de la fonction coût, ce processus s'arrête et retourne la meilleur configuration trouvée quand la condition d'arrêt est réalisée, cette condition peut être généralement une limite pour le membre d'itérations.

1.2 Algorithmes évolutifs:

Ces algorithmes sont basés sur le principe de processus d'évolution naturelle et les mécanismes d'évolution des espèces vivantes. Un algorithme évolutif typique est composé de trois éléments essentiels:

- une population constitué de plusieurs individus représentant des solutions du problème donné.
- Un mécanisme d'évaluation de l'adaptation de chaque individu de la population à l'égard de son environnement extérieur.
- Un mécanisme d'évolution composé d'opérateurs permettant d'éliminer certains individus et de produire de nouveaux individus à partir des individus sélectionnés.

Un algorithme évolutif débute avec une population initiale souvent générée aléatoirement et répète un cycle dévolution composée de trois étapes séquentielles.

- Mesurer l'adaptation (la qualité) de chaque individu de la population par le mécanisme d'évaluation.
- Sélectionner une partie des individus. Produire de nouveaux individus ou des recombinaisons d'individus sélectionnés.

Ce processus se termine quand la condition d'arrêt est vérifiée, par exemple quand un membre maximum de génération est atteint. Selon l'analogie de l'évolution naturelle, la qualité des individus de la population devrait tendre à s'améliorer ou fur et à mesure des processus.

- Un algorithme évolutif comporte: un ensemble d'opérateur tel que la sélection, la mutation et éventuellement le croisement.

La sélection a pour objectif de choisir les individus qui vont pouvoir suivre et se reproduire pour transmettre leur caractéristiques à la génération suivante elle est basée généralement sur la conservation des individus les mieux adaptés et l'élimination des moins adaptés.

Le croisement cherche à combiner les caractéristiques des individus parents pour créer des individus enfants dans la génération futur et la mutation effectuée de légères modifications de certains individus.

On peut distinguer deux grandes classes d'algorithmes évolutifs: les algorithmes génétiques et les colonies de fourmis. Ces méthodes se différencient par leur manières de présenter l'information et la façon de faire évoluer la population d'une génération à l'autre.

a- Algorithme génétique: cette classe de méthodes est basée sur une imitation des phénomènes d'adaptation des êtres vivants. Les algorithmes génétiques fonctionnent sur une analogie avec la reproduction des êtres vivants.

De manière générale les algorithmes génétiques utilisent un même principe une population d'individus (correspondant à des solutions) évoluées en même temps comme dans l'évolution naturelle en biologie pour chacun des individus, on mesure sa faculté d'adaptation en milieu extérieur par le fitness. Les algorithmes génétiques s'appuient alors sur trois fonctionnalités.

- La sélection: qui permet de favoriser les individus qui ont un meilleur fitness pour nous le fitness sera le plus sauvent la valeur de la fonction objectif de la solution associée à l'individu.

- Le croissement: qui combine deux solutions parents pour former un ou deux enfants

(offspring) en essayant de conserver les bonnes caractéristiques des solutions parents.

- La mutation: qui permet d'ajouter de la diversité à la population en mutant certaines caractéristiques (gènes) d'une solution.

La représentation des solutions (le codage) est un point critique de la réussite d'un algorithme génétique. Il faut bien sûr qu'il s'adapte le mieux possible au problème et à l'évaluation d'une solution.

Algorithme (1): un algorithme génétique simple

1: Initialisation: générer une population initiale **P** de solutions de taille **|P| = n**

2: Répéter

3: Sélectionner: choisir deux solutions **X** et **X′** par une technique de sélection

4: Croisement: combiner les deux solutions parents **X** et **X′** pour former une solution enfant y

5: Mutation de **y** sous conditions

6: Choisir une solution individuelle y′ pour être remplacé dans la population

7: Remplacer **y′** par y dans la population

8: Jusqu'à critère d'arrêt satisfait.

b- Algorithmes de colonies de fourmis:

Cette nouvelle métaheuristique imite le comportement de fourmis cherchant de la nourriture. A chaque fois qu'une fourmi se déplace elle laisse sur la trace de son passage une odeur (la phéromone) comme la fourmi est rarement une exploratrice isolée avec plusieurs de ses congénères elle explore la région en quête de nourriture face à l'obstacle, le groupe des fourmis explorent les deux cotés de l'obstacle et se retrouvent, puis elles reviennent au nid avec de la nourriture. Les autres fourmis qui veulent obtenir de la nourriture elles aussi vont emprunter le même chemin si celui-ci se sépare face à l'obstacle, les fourmis vont alors emprunter préférablement le chemin sur lequel le phéromone sera la plus forte mais la phéromone étant une odeur elle s'évapore. Si peu de fourmis empruntant une trace, il est possible que ce chemin ne soit plus valable au bout d'un moment, il en est de même si des fourmis exploratrices empruntant un chemin plus long (pour contournement de l'obstacle par exemple). Par contre, si le chemin est fortement emprunté, chaque nouvelle fourmi qui passe redépose un peu de phéromone et renforce ainsi la trace, donnant alors à ce chemin une plus grande possibilité d'être emprunté.

C- Algorithme 2: Métaheuristique ACO:

1: Initialisation: Créer une population initiale de fourmis

2: Répéter

3: Pour chaque fourmi faire

4: Construire une solution par une procédure de construction à l'aide des traces de phéromones.

5: Mise à jour des traces de phéromones basées sur la qualité des solutions trouvées.

6: Fin pour

7: Jusqu'à critère d'arrêt satisfait

2. Les méthodes hybrides:

l'idée essentielle de cette hybridation consiste à exploiter pleinement la puissance de recherche de méthodes voisinage et de recombinaison des algorithmes évolutifs sur une population de solution, un tel algorithme utilise, une ou plusieurs méthodes de voisinage sur les individus de la population pendant un certain membre d'itération ou jusqu'à la découverte d'un ensemble d'optima locaux et invoque ensuite un mécanisme de recombinaison pour créer un nouveau individu comme pour les algorithmes génétiques spécialisés, la recombinaison doit impérativement être adaptée au problème traité.

Les algorithmes hybrides sont sans doute parmi les méthodes les plus puissantes malheureusement le temps de calcul nécessaire pouvant devenir prohibitif à cause des membres d'individus manipulés dans la population.

2.1 Méthode de descente:

c'est l'une des heuristiques de recherche locale les plus simples. Elle consiste à rechercher dans le voisinage de la solution courante, une solution de coût plus faible. Elle procède ainsi jusqu'à arriver à un optimum local.

A partir d'une solution trouvée par une heuristique par exemple, on peut très facilement implémenter des méthodes de descente. Ces méthodes s'articulent toutes autour d'un principe simple. Partir d'une solution existante, chercher une solution dans le voisinage et accepter cette solution si elle améliore la solution courante.

L'algorithme 1 présente le squelette d'une méthode de descente simple. A partir d'une solution initiale **x**, on choisit une solution **x′** dans le voisinage **N(x)** de **x**. Si cette solution est meilleure que **x**, **f(x′)** alors on accepte cette solution comme nouvelle solution **x** et on recommence le processus jusqu'à ce qu'il n'y ait plus aucune solution améliorante dans le voisinage de **x**.

2.2 Algorithme 1 Descente simple:

1: initialisation: trouver une solution initiale **x**

2: répéter

3: recherche de voisinage: trouver une solution $\mathbf{x' \in N(x)}$

4: si $\mathbf{f(x') < f(x)}$ alors

5: $\mathbf{x' := x}$

6: fin si

7: jusqu'à $\mathbf{f(y) \geq f(x), \forall y \in N(x)}$

L'avantage principal de ces méthodes réside dans leur grande simplicité et leur rapidité. Toutefois elles comportent deux obstacles majeurs qui limitent considérablement leur efficacité:

- suivant la taille et la structure du voisinage **N(x)** considéré, la recherche de la meilleur solution voisine est un problème qui peut être aussi difficile que le problème initiale;
- une méthode de descente est incapable de progresser au –delà du premier minimum local rencontré. Or les problèmes d'optimisation combinatoire comportent typiquement de nombreux optima locaux pour lesquels la valeur de la fonction objectif peut être fort éloignée de la valeur optimale.

Pour faire faces à ces carences, la solution la plus simple est la méthode de relance aléatoire qui consiste à générer une nouvelle configuration de départ de façon aléatoire et à recommencer une descente. Une autre solution consiste à accepter des voisins de même performance que la configuration courante. Cette approche permet à la recherche de se déplacer sur les plateaux, mais n'est pas suffisante pour ressortir de tous les optima locaux. D'autres raffinements plus élaborés sont également possibles, par exemple: l'introduction de voisinages variables, et les techniques de réduction ou d'élargissement du voisinage.

3. Recuit simulé:

Cette classe de méthodes d'optimisation s'inspire des méthodes de simulation de Métropolies (années 50) en mécanique statistique. L'analogie historique s'inspire du recuit des métaux en métallurgie: un métal refroidi trop vite présente de nombreux défauts microscopiques, c'est l'équivalent d'un optimum local pour un problème d'optimisation combinatoire. Si on le refroidi lentement, les atomes se réarrangent, les défauts disparaissent, et le métal a alors une structure très ordonnée, équivalent à un optimum global.

L'analogie avec une méthode d'optimisation est trouvée en associant une solution à un état métal, son équilibre thermodynamique est la valeur de la fonction objectif de cette solution. Passer d'un état du métal à un autre correspond à passer d'une solution à une solution voisine.

Pour passer à une solution voisine, il faut respecter l'une des deux conditions:

- Soit le mouvement améliore la qualité de la solution précédente, **i.e.** en minimisation, la variation de coût négative ($\Delta C < 0$).
- Soit le mouvement détériore la qualité de la solution précédente et la probabilité **p** d'accepter un tel mouvement est inférieur à une valeur dépendante de la température courante **t** ($\mathbf{p < e^{-\Delta C / t}}$).

Le schéma de refroidissement de la température est une des parties les plus difficiles à régler dans ce cas. Ces schémas sont cruciaux pour l'obtention d'une implémentation efficace. Un refroidissement trop rapide mènerait vers un optimum local pouvant être de très mauvaise qualité. Un refroidissement trop lent serait très coûteux en temps de calcul. Le réglage des différents paramètres (température initiale, nombre d'itérations par palier de température, décroissance de la température, …) peut donc être long et difficile. Sans être exhaustif, on rencontre habituellement trois grandes classes de schémas de refroidissement [HAO 99]:

- Réduction par paliers: chaque température est maintenue égale pendant un certain nombre d'itérations, et décroît ainsi par paliers.
- Réduction continue: la température est modifiée à chaque itération.
- Réduction polynomial: la température décroît à chaque itération avec des augmentations occasionnelles.

L'algorithme 2 présente les principales caractéristiques d'un recuit simulé.

3.1 Algorithme 2 Recuit simulé:

1: initialisation: trouver une solution initiale **x**, poser une température initiale **t**

2: répéter

3: recherche de voisinage: trouver une solution $\mathbf{x' \in N(x)}$

4: déterminer $\mathbf{\Delta C = f(x') - f(x)}$

5: obtenir $\mathbf{p \sim U(0.1)}$

6: si $\mathbf{\Delta C < 0}$ ou $\mathbf{e^{-\Delta C/t} > p}$ alors

7: $\mathbf{x' := x}$

8: fin si

9: réduire la température **t** selon un schéma de refroidissement

10: jusqu'à un critère d'arrêt satisfait.

4. Comparaison entre méthodes exactes et méthodes approchées (voir la table 01)

Le tableau ci-dessous représente quelques différences entre les méthodes exactes et les méthodes approchées.

Méthode approchée	Méthode exacte
Efficacité pour les problèmes de grande taille	Efficacité pour des cas particuliers des problèmes de taille raisonnable
Espace mémoire raisonnable	Espace mémoire considérable
Durée de résolution très petite par rapport aux méthodes exactes	Durée de temps de résolution est généralement considérable pour les problèmes de grande taille
Aucune garantie d'optimalité	Garantie d'optimalité
Pratiquement simple à implanter et à comprendre	Pratiquement difficile à implanter et à comprendre.

Table (01)

CONCLUSION

Conclusion générale

Dans ce mémoire nous avons abordé les problème d'optimisation combinatoire .

Nous avons rappelé les méthodes de solutions exactes et les méthodes de solutions approchées avec leurs avantages et leurs inconvénients. Avevc un comparison Puisque la majorité de ce genre de problèmes est NP-difficile ainsi un algorithme polynomial pour déterminer une solution exacte est impossible d'être établis.

Références bibliographiques

Références bibliographiques:

[1] Carlier J. et Chrétienne P. 1988. Problème d'ordonnancement, modélisation,
complexité / algorithmes. Masson.

[2] I.Dumitresco and T. Stutzle, 2003. Combinations of Local Search and Exact
Algorithms, Evo Workshops, 211 – 223, Springer Verlag, Berlin

[3] Fred Glove and Said Hanafi, 1999, Tabou Search and Finite Convergence, in
Discrete Applied Mathematics.
de GERARD.

[4] Jin-Kao Hao et al., 1999, Métaheuristique pour l'optimisation Combinatoire et l'affectation sous contraintes, Revue d'Intelligence Artificielle vol. N° 1999

[5] M. Haouari and T. Ladhari, 2003, A Branch – and – Bound – Based Local Search
Method for the Flow Shop Problem, JORS, 54, 1076 – 1084

Résumé:

Le travail de cet mémoire concerne sur l'optimisation combinatoire avec représentations les méthodes de résolutions.

ملخص

الهدف من هذه المذكرة هو تقديم دراسة للتحليل التوفيقي مع تقديم بعض طرق الحل الدقيقة والتقريبية للمسائل الرياضياتية والحاسوبية.