N° d'ordre : 01/2003-M/IN

République Algérienne Démocratique et Populaire

Ministère de la Recherche et de l'Enseignement Supérieur

Université des Sciences et de la Technologie

Houari Boumediene

USTHB/Alger

Faculté d'Electronique et d'Informatique

Département d'Informatique

THESE

Présentée pour l'obtention du grade de

Magister En Informatique

Spécialité : Intelligence artificielle et bases de données avancées

Par :

Mlle IBRI Sarah

Parallélisation du système de colonie de fourmis et application au problème MAX-W-SAT

Soutenue publiquement le : 22/02/2003, devant le jury composé de :

Président	USTHB	Professeur	MR KHELLADI Abdelkader
Directrice de thèse	USTHB	Professeur	Mme DRIAS Habiba
Examinateur	U.Toulouse1	Professeur	MR SIBERTIN-BLANC Christophe
Examinatrice	USTHB	Maître de Conférence	Mme YOUCEF-ETTOUMI Fatiha
Examinatrice	INI	Chargée de cours	Mlle BENATCHBA Karima

Table des matières

Page

Introduction générale …………………………………………………………. 1

Chapitre I : Le problème MAX-SAT

1. Notions de complexité ……………………………………………………... 3
2. Les classes des problèmes ………………………………………………….. 3
3. Le problème SAT ………………………………………………………… 4
   3.1 Définition du problème SAT ……………………………………………… 4
   3.2 Les variantes du problème SAT ………………………………………... 4
   3.3 Le problème MAX-SAT…………………………………………………… 5
   3.4 Domaines d'application du problème SAT …………………………... 5
   3.5 Les méthodes de résolution ……………………………………………. 6
4. Conclusion ……………………………………………………………….. 13

Chapitre II : La méta_heuristique ACO

1. Le principe des fourmis réelles …………………………………………….. 14
2. L'inspiration des fourmis réelles …………………………………………… 15
3. La méta_heuristique ACO ………………………………………………….. 16
4. Les algorithmes ACO ……………………………………………………. 19
   4.1 Système de fourmis 'AS' ……………………………………………… 19
   4.2 Les variantes de AS ……………………………………………………. 20
   4.3 Le système de colonie de fourmis 'ACS' …………………………… 21
5. Conclusion …………………………………………………………………... 23

Chapitre III : ACS pour le MAX-W-SAT

1. Stratégie1 ………………………………………………………………… 24
   Algorithme général…………………………………………………. 24
   Initialisation de la phéromone……………………………………… 25
   Génération des solutions initiales…………………………………... 25
   Construction de la solution………………………………………. 26
   Procédure de recherche locale……………………………………… 27
   Mise à jour de la phéromone ……………………………………….
2. Stratégie 2 ………………………………………………………………… 28
3. Conclusion ………………………………………………………………. 29

Chapitre IV: Parallélisation de ACS-SAT

1. le calcul parallèle……………………………………………………….. 30
   Concept de processus…………………………………………….. 30
   Relations entre processus……………………………………… 30
   Synchronisation des processus………………………………… 31
2. ACO parallèle…………………………………………………….. 32
3. Parallélisation de ACS-SAT………………………………………….. 33
   Méthode synchrone……………………………………………. 33
   Méthode asynchrone…………………………………………... 35
4. Environnement parallèle…………………………………………. 37
5. Conclusion……………………………………………………… 38

Chapitre V: Tests et évaluation

1. Détermination des paramètres…………………………………… 39
2. Comparaison des deux stratégies séquentielles…………………….. 44
3. Communication entre les sous colonies de fourmis…………………. 45
4. Comparaison entre les différents algorithmes………………………. 46
5. Comparaison avec d'autres méta-heuristiques……………………… 48
6. conclusion……………………………………………………… 49

Conclusion générale …………………………………………………….. 50
Annexe………………………………………………………………. 52
Bibliographie ……………………………………………………………... 56

Résumé
Le système de colonies de fourmis "ACS :Ant Colony System" est une approche très importante de la méta-heuristique ACO "Ant Colony Optimization" dédiée à la résolution des problèmes d'optimisation et qui a été appliquée avec succès au problème de voyageur de commerce "TSP : Travelling Salesman Problem".
La satisfiabilité propositionnelle "SAT" est un problème important de la théorie de la complexité et de l'intelligence artificielle. Il consiste à déterminer, pour une formule en forme normale conjonctive "CNF : Conjunctive Normal Form", l'existence d'une instanciation de ses variables pour laquelle la formule est évaluée à vrai.
Le problème de satisfiabilité maximale pondérée "MAX-W-SAT" est une version d'optimisation très importante du problème SAT, qui cherche une assignation aux variables qui maximise la somme des poids des clauses satisfaites dans une formule logique CNF ; ce problème est aussi NP-COMPLET.
Une première étape de ce travail, consiste à étudier l'efficacité de ACS dans la résolution du problème de satisfiabilité maximale pondérée "MAX-W-SAT" ; ceci revient à proposer une adaptation de toutes les règles constituant cette approche aux éléments caractérisant le problème en question.
Les algorithmes ACO contiennent un degré de dépendance faible, ce qui rend leur parallélisation facile et bénéfique. De ce fait, nous nous intéressons aussi à l'étude des possibilités de parallélisation de l'algorithme ACS proposé dans la première étape, tout en discutant leurs effets et leurs différences.

Mots clés
Optimisation combinatoire, satisfiabilité propositionnelle, SAT, MAX-SAT, méta-heuristique, ACO, ACS, recherche locale, calcul parallèle, synchronisation, processus.

Abstract
The ant colony system "ACS" is a very important approach of the ACO "Ant Colony Optimization" meta-heuristic; it has been successfully applied to the travelling salesman problem "TSP".
The satisfiability problem "SAT" takes an important place in the computational complexity and in the artificial intelligence. It consists of finding an assignment to a set of Boolean variables to satisfy a formula in a conjunctive normal form "CNF" (a CNF formula is a conjunction of clauses; each clause is a disjunction of literals). In the weighted max-sat "MAX-W-SAT" each clause of the formula has a weight, and the objective becomes to maximize the sum of satisfied clauses weights.
In this work we aim to study the efficiency of ACS in the solution of the weighted max-sat problem "max-w-sat". This will require an adaptation of all the rules of this approach to the elements which characterize our problem.
The ACO algorithms contain a low dependence level, this feature make them easy and beneficent to parallelize. Thereby, we will also study various ways to parallelize the ACS algorithm proposed in the first step of this work, including discussion about their impacts and their differences.

Keywords
Combinatorial optimization, propositional satisfaction, SAT, MAX-SAT, meta-heuristic, ACO, ACS, local search, parallel computation, synchronization, processus.

La satisfiabilité propositionnelle "SAT" est un problème important de la théorie de la complexité et de l'intelligence artificielle. C'est le premier problème qui a été montré NP-COMPLET. Il consiste à déterminer, pour une formule en forme normale conjonctive "CNF : Conjunctive Normal Form", l'existence d'une instanciation de ses variables pour laquelle la formule est évaluée à vrai.
Le problème de satisfiabilité maximale pondérée "MAX-W-SAT" est une version d'optimisation très importante du problème SAT, il cherche une assignation aux variables qui maximise la somme des poids des clauses satisfaites dans une formule logique CNF ; ce problème est aussi NP-COMPLET.

Développer des algorithmes exactes et approchés pour le problème SAT, peut mener à des approches générales pour résoudre les problèmes d'optimisation combinatoire. En effet, tout problème NP-COMPLET est réductible au problème SAT ; en plus, certains problèmes codés sous forme normale conjonctive et résolus grâce à un algorithme dédié à SAT trouvent une solution plus rapidement que par des outils spécifiques dans leur codage d'origine**[29]**.

A moins que NP=P, le problème SAT, ainsi que plusieurs d'autres problèmes d'optimisation combinatoire ne puissent être résolus par un algorithme exact en un temps raisonnable ; par conséquent, les méta-heuristiques sont devenues la solution la plus adéquate.
Parmi les méta-heuristiques qui existent, il y a le recuit simulé, les algorithmes génétiques, la recherche tabou, la recherche dispersée et les réseaux de neurones ; la plupart de ces méthodes sont dérivées de la nature et ont été appliquées avec succès au problème SAT.

L'optimisation par les colonies de fourmis "ACO : Ant Colony Optimization " est un nouveau membre dans la classe des méta-heuristiques inspirées des phénomènes naturels. Elle a été développée récemment par Coloroni, Dorigo et Maniezzo pour résoudre le problème de voyageur de commerce "TSP : Travelling Salesman Problem" ; ensuite appliquée avec succès à d'autres problèmes standards d'optimisation combinatoire comme l'affectation quadratique, le routage des véhicules, job shop, coloration de graphe et le problème de l'emploi du temps.
C'est une approche basée population, elle est inspirée du comportement des fourmis réelles, et exploite la rétroaction positive aussi bien que la recherche avide. Elle consiste en un ensemble de règles de transitions et de mises à jour appliquées par les fourmis artificielles qui coopèrent pour arriver à une bonne solution au problème à résoudre **[13]**.

Puisque le problème MAX-W-SAT appartient à la classe des problèmes NP-DIFFICILE, on peut prétendre qu'il n'est pas possible de lui appliquer le calcul parallèle, car la notion NP-DIFFICILE de ce problème fait que le temps d'exécution au pire des cas ne peut jamais être réduit en un temps polynomial, à moins que nous ayant un nombre de processeurs exponentiel. Cependant la complexité du temps moyen des méta-heuristiques est généralement polynomiale, et lorsque des solutions sub-optimales ou proches de l'optimales sont acceptables, il existe des algorithmes pour trouver les solutions désirées en un temps polynomial ; dans de tels cas, le calcul parallèle peut significativement augmenter la taille des problèmes à résoudre **[24]**.

Les approches ACO sont basées population où une population d'agents (fourmis artificielles) est utilisée pour trouver le but désiré ; de telles approches sont naturellement convenables pour le calcul parallèle. Cette caractéristique inhérente constitue la motivation principale pour l'amélioration de la performance des algorithmes ACO en utilisant la parallélisation.

L'idée principale dans la majorité des implémentations des algorithmes de fourmis exploitant le parallélisme est de diviser la colonie de fourmis en sous colonies, et d'affecter chacune d'elles à un processeur pour évoluer en parallèle.
Pendant le calcul, des échanges d'informations entre les sous colonies d'un algorithme de fourmis parallèle sont nécessaires ; et c'est la fréquence de la communication, l'information échangée entre les sous colonies et le mode de synchronisation adopté qui font la différence entre les divers travaux proposés dans ce domaine.

La suite de cette thèse est structurée comme suit :

- Une étude du problème de satisfiabilité avec ses variantes, et les différentes méthodes proposées dans la littérature pour sa résolution est détaillée dans le premier chapitre.
- Dans le deuxième chapitre nous évoquerons la meta-heuristique ACO avec ses différentes versions d'améliorations.
- Le troisième chapitre est consacré à l'application de l'algorithme ACS au problème MAX-W-SAT, en proposant deux stratégies différentes.
- Des notions primordiales sur le calcul parallèle, suivies par deux méthodes de parallélisation synchrone et asynchrone de l'algorithme ACS seront détaillées dans le quatrième chapitre.
- Le dernier chapitre est consacré à l'expérimentation des performances des différentes implémentations, tout en effectuant les tests nécessaires pour le réglage des paramètres empiriques et la comparaison des résultats numériques.

Résumé

Le système de colonies de fourmis “ACS :Ant Colony System” est une approche très importante de la méta-heuristique ACO “Ant Colony Optimization” dédiée à la résolution des problèmes d’optimisation et qui a été appliquée avec succès au problème de voyageur de commerce “TSP : Travelling Salesman Problem”.

La satisfiabilité propositionnelle “SAT” est un problème important de la théorie de la complexité et de l’intelligence artificielle. Il consiste à déterminer, pour une formule en forme normale conjonctive “CNF : Conjunctive Normal Form”, l’existence d’une instanciation de ses variables pour laquelle la formule est évaluée à vrai.

Le problème de satisfiabilité maximale pondérée “MAX-W-SAT” est une version d’optimisation très importante du problème SAT, qui cherche une assignation aux variables qui maximise la somme des poids des clauses satisfaites dans une formule logique CNF ; ce problème est aussi NP-COMPLET.

Une première étape de ce travail, consiste à étudier l’efficacité de ACS dans la résolution du problème de satisfiabilité maximale pondérée “MAX-W-SAT” ; ceci revient à proposer une adaptation de toutes les règles constituant cette approche aux éléments caractérisant le problème en question.

Les algorithmes ACO contiennent un degré de dépendance faible, ce qui rend leur parallélisation facile et bénéfique. De ce fait, nous nous intéressons aussi à l’étude des possibilités de parallélisation de l’algorithme ACS proposé dans la première étape, tout en discutant leurs effets et leurs différences.

Mots clés

Optimisation combinatoire, satisfiabilité propositionnelle, SAT, MAX-SAT, méta-heuristique, ACO, ACS, recherche locale, calcul parallèle, synchronisation, processus.

Abstract

The ant colony system “ACS” is a very important approach of the ACO “Ant Colony Optimization” meta-heuristic; it has been successfully applied to the travelling salesman problem “TSP”.

The satisfiability problem “SAT” takes an important place in the computational complexity and in the artificial intelligence. It consists of finding an assignment to a set of Boolean variables to satisfy a formula in a conjunctive normal form “CNF” (a CNF formula is a conjunction of clauses; each clause is a disjunction of literals). In the weighted max-sat “MAX-W-SAT” each clause of the formula has a weight, and the objective becomes to maximize the sum of satisfied clauses weights.

In this work we aim to study the efficiency of ACS in the solution of the weighted max-sat problem “max-w-sat”. This will require an adaptation of all the rules of this approach to the elements which characterize our problem.

The ACO algorithms contain a low dependence level, this feature make them easy and beneficent to parallelize. Thereby, we will also study various ways to parallelize the ACS algorithm proposed in the first step of this work, including discussion about their impacts and their differences.

Keywords

Combinatorial optimization, propositional satisfaction, SAT, MAX-SAT, meta-heuristic, ACO, ACS, local search, parallel computation, synchronization, processus.

Parallélisation du système de colonie de fourmis et application au problème MAX-W-SAT

Résumé

Le MAX-W-SAT est un problème NP-COMPLET qui dérive du problème SAT, il cherche une assignation aux variables qui maximise la somme des poids des clauses satisfaites dans une formule logique CNF.

Le système de colonies de fourmis "ACS" est une approche importante de la méta-heuristique ACO, qui a été développée pour la première fois par Dorigo pour résoudre le problème de voyageurs de commerce.

Une première étape de ce travail, consiste à étudier l'efficacité de ACS dans la résolution du problème MAX-W-SAT ; ceci revient à trouver une bonne adaptation de toutes les règles constituant cette approche aux éléments caractérisant le problème en question.

Les algorithmes ACO contiennent un degré de dépendance faible, ce qui rend leur parallélisation facile et bénéfique. De ce fait, nous proposons deux méthodes différentes pour paralléliser l'algorithme ACS-SAT.
L'idée de base des deux méthodes consiste à partager la colonie de fourmis entre plusieurs processus de façon équilibrée, en ayant une sous colonie pour chaque processus, et de lancer les processus en parallèle.
Le mode de synchronisation et la façon que les processus utilisent pour communiquer, font la différence entre les deux méthodes synchrone et asynchrone que nous proposons.

Les deux étapes du travail sont suivies par une phase expérimentale qui nous permet d'évaluer les algorithmes proposés, et de révéler d'autres aspects importants liés à la communication entre les sous colonies de l'algorithme parallèle.

Réalisé par :
Melle Ibri Sarah

Proposé et dirigé par :
Mme Drias Habiba

Le problème de satisfiabilité occupe une position centrale dans l'étude de la complexité, et sa résolution mène à solutionner beaucoup d'autres problèmes importants d'optimisation combinatoire; pour cette raison il a été beaucoup étudié. Dans la suite de ce chapitre nous définissons le problème SAT avec ses différentes versions d'optimisation ainsi que les méthodes qui existent pour sa résolution.

1. Notions de complexité

La distinction entre les bons algorithmes et les mauvais en temps de calcul peut être faite en exécutant les programmes correspondants sur machine: c'est la complexité empirique **[16]**. Une meilleure façon pour faire cette distinction est la complexité théorique, elle est définie comme une fonction mathématique qui permet de donner le temps d'exécution des algorithmes en déterminant le nombre maximum d'instructions exécutées en fonction de la taille de la donnée du problème.

Un algorithme dont la fonction de complexité peut être bornée par un polynôme est un algorithme à temps polynomial, sinon il est dit à temps exponentiel.

2. Les classes des problèmes

- ✔***Les problèmes NP:*** Ce sont les problèmes de décision (à qui il faut répondre par oui ou par non) dont les algorithmes correspondants peuvent, en un temps polynomial, vérifier sur une machine non déterministe qu'une interprétation de l'instance du problème est effectivement une solution, mais qui demandent dans le pire des cas un temps exponentiel pour la trouver **[9]**.
- ✔***Les problèmes P:*** Un problème de décision appartient à la classe P s'il existe un algorithme déterministe en un temps polynomial pour résoudre ce problème ($P \subseteq NP$).
- ✔***Les problèmes NP-COMPLETS:*** Sont les problèmes les plus difficiles de la classe NP. Un problème est NP-complet quand tous les problèmes NP lui sont réductibles. Si on trouve un algorithme polynomial pour un problème NP-complet, on trouve alors automatiquement une résolution polynomiale de tous les problèmes de la classe NP.

THEOREME DE COOK: Le problème SAT est NP-COMPLET.

SAT est le premier exemple des problèmes NP-COMPLETS et tient une position centrale dans l'étude de la complexité.

✔***Les problèmes NP-DIFFICILES***: Un problème est NP-DIFFICILE s'il est au moins difficile qu'un problème NP-COMPLET; ceci veut dire que tous les problèmes NP-COMPLETS sont NP-DIFFICILES[16].

3. Le problème SAT

La satisfiabilité propositionnelle est un problème important de la théorie de la complexité et de l'intelligence artificielle. Il consiste à déterminer pour une formule du calcul propositionnel, l'existence d'une instanciation de ses variables pour laquelle la formule est évaluée à vrai. SAT considère souvent la forme normale conjonctive (CNF) des formules.

Une formule CNF est une conjonction de clauses, elles même disjonction de littéraux. Un littéral est une variable propositionnelle ou sa négation et ne peut prendre que la valeur vrai ou faux.

3.1 Définition du problème SAT

Soit X={x1,x2,……,xn} un ensemble de n variables booléennes.

C={c1,c2,…….,cm}un ensemble de m clauses.

$$C_i = \bigvee_{k=1}^{|Ci|} L_k$$

Chaque clause est une disjonction de littéraux

La question à une donnée SAT est de déterminer une instanciation (une assignation) aux variables de X pour que toutes les clauses soient satisfaites simultanément.

3.2 Les variantes du problème SAT

✔**Problème SAT de décision**

Il est décrit par une instance de SAT, et permet de répondre par "oui" si la donnée est satisfiable, par "non" si elle est contradictoire.

✔ **Problème SAT de recherche**

Consiste a exhiber certaines solutions d'une instance SAT, Il permet de répondre au problème de décision, il est donc au moins aussi difficile que le problème de décision.

✔ **Problème de dénombrement**

Permet de déterminer le nombre de solutions d'une instance SAT.

Suivant le type de l'instance SAT on peut distinguer :

- **Problème K-SAT**

C'est le cas où toutes les clauses sont de longueur K (càd chaque clause contient K littéraux), noté K-SAT.

Un intérêt est porté aux valeurs 2 et 3 de K. En effet 3 est la plus petite valeur de K pour laquelle K-SAT est NP-complet, et 2-SAT peut être résolu en un temps linéaire.

- **Problème HORN-SAT**

Représente des problèmes où chaque clause a au plus une variable sans négation, résolu également en un temps linéaire.

3.3 Problème MAX-SAT

C'est un problème dédié à la recherche d'une solution partielle satisfaisant le plus grand nombre de clauses d'une formule CNF.

MAX-SAT est un problème NP-COMPLET même lorsque les clauses ne contiennent que 2 littéraux.

- **Problème MAX-W-SAT(weighted MAX_SAT)**

Une valeur entière $W(C_i)$ est associée à chaque clause C_i définissant son poids.

Le problème consiste donc à déterminer une instanciation qui maximise la somme des poids des clauses satisfaites.

3.4 Domaines d'applications du problème MAX-SAT

Le problème de satisfiabilité est un problème central dans l'intelligence artificielle, la logique mathématique, et l'optimisation combinatoire. Les problèmes de la vision par ordinateur, conception VLSI, bases de données,

raisonnement automatisé et de fabrication, nécessitent tous la résolution des instances du problème de satisfiabilité.

Citons deux exemples très connus:

Dans le domaine des systèmes experts on a besoin de vérifier la consistance des bases de connaissances (ou BDs) après leur mise à jour. Si la BC n'est pas consistante on essaye de trouver le plus petit ensemble d'implications qui doit être supprimé pour récupérer la consistance, ceci est obtenu en résolvant une certaine instance du problème MAX-SAT **[17]**.

Dans **[9]** une hybridation d'un problème SAT et un problème MAX-SAT a été utilisée pour résoudre le problème de minimisation du nombre de croisement dans les graphes à niveaux (pour obtenir une bonne lisibilité dans la représentation des graphes à niveaux). Où on utilise le problème SAT pour éliminer tous les croisements entre les arcs de deux niveaux adjacents: le problème dit "level planar embedding", et l'utilisation du problème MAX-SAT pour minimiser le nombre de croisements entre les arcs de deux niveaux successifs: "K-level crossing minimization problem".

3.5 Les méthodes de résolution

Il existe deux alternatives pour résoudre un problème MAX-SAT: soit on cherche l'optimum en risquant l'explosion combinatoire, ou bien on se contente d'une solution proche de l'optimale trouvée en un temps raisonnable.

Pour la première alternative on a :

3.5.1 Les algorithmes exactes (ou complets): où on utilise les méthodes arborescentes, la programmation linéaire ou la programmation dynamique.

Davis et Putnam ont introduit une méthode énumérative dite" Generate & test" **[9]**, dans laquelle les variables sont instanciées une par une . Après chaque instanciation d'une variable, on simplifie la formule en éliminant les clauses qui contiennent le littéral évalué à vrai par cette affectation, ainsi que toutes les occurrences des littéraux évalués à faux par cette affectation. Ensuite on fait les tests suivants:

Si la formule ne contient aucune clause alors return (satisfiable).

Si la formule contient une clause vide alors return (non satisfiable).

Sinon continuer l'instanciation des autres variables.

Après le test si la valeur retournée est “non satisfiable” alors on réaffecte à la dernière variable instanciée la négation de la valeur assignée auparavant, et on refait le test.

Pour la deuxième alternative on a:

3.5.2 Les algorithmes approximatifs: Pour éviter l’explosion combinatoire des algorithmes exacts, les chercheurs se sont dirigés vers le développement des algorithmes C-approximatifs pour la résolution du problème MAX-SAT.

Soit la solution optimale S* d’un problème d’optimisation, un algorithme C-approximatif est un algorithme qui produit en un temps polynomial une solution S tel que $S >= CS^*$ $(0<C<1)$ **[31]**.

✔**Les algorithmes constructifs:** sont basés sur des heuristiques et permettent de construire une solution réalisable approchée.

- **Les algorithmes de David Johnson:** Ce sont les deux premiers algorithmes approximatifs pour le MAX-SAT.

Greedy algorithm johnson1: Dans cet algorithme on choisit à chaque étape le littéral qui occure dans le plus de clauses. Si le littéral sélectionné est positif, la variable correspondantes est fixée à vrai, sinon à faux.

Les clauses satisfaites par le littéral sont supprimées de la formule, l’algorithme s’arrête quand la formule est satisfaite ou toutes les variables ont été assignées.

Johnson a monté que cet algorithme est k/k+1-approximatif (k est le nombre minimum de littéraux dans les clauses)

Greedy algorithm johnson2: Johnson a introduit un second algorithme qui améliore la performance du premier en obtenant un algorithme 2/3-approximatif.

L’algorithme associe une masse $W(Ci)=2^{-|Ci|}$ à chaque clause. Il procède en choisissant à chaque étape une variable Y qui n’a pas été déjà assignée à une valeur, ensuite détermine la classe où elle apparaît positivement et celle où elle apparaît négativement. La valeur assignée à Y favorise les clauses dont la somme de leurs masses est la plus grande (car les grandes clauses sont plus faciles à satisfaire que les petites).

- **Les algorithmes aléatoires:** Il existe plusieurs algorithmes aléatoires, le plus général est “Genrandom” qui s’applique sur des clauses pondérées.

L’algorithme répète les étapes suivantes jusqu’à arriver à la satisfiabilité de la formule ou bien l’instanciation de toutes les variables.

- Choisir une variable X_i aléatoirement.
- Fixer X_i à vrai avec une probabilité P_i.
- Calculer l’ensemble des clauses satisfaites par cette instanciation C’.
- Calculer $\sum_{C_j \in C'} W(C_j)$.

Il a été prouvé qu’il est possible de trouver les valeurs convenables de P_i pour obtenir un algorithme aléatoire ¾-approximatif.

- **La procédure GRASP:** “Greedy Randomised Adaptative Search Procedure”

Chaque itération GRASP consiste en une phase de construction et une phase de recherche locale. La meilleure solution des deux phases est gardée comme résultat Dans la phase de construction une solution réalisable est construite. A chaque itération de la construction, le choix de l’élément à rajouter est déterminé en ordonnant tous les éléments dans une liste candidate suivant une fonction greedy. Cette fonction mesure l’avantage de la sélection de chaque élément. L’heuristique est adaptative car les avantages (bénéfices) associés à chaque élément sont mises à jour à chaque itération de la phase de construction pour refléter les changements apportés par la sélection de l’élément précèdent, Le composant probabiliste est caractérisé par le choix aléatoire de l’un des meilleurs éléments de la liste candidate et pas nécessairement le meilleur. La solution générée par la construction GRASP n’est pas garantie à être localement optimale, cependant il est souvent bénéfique d’appliquer une recherche locale pour essayer d’améliorer chaque solution construite **[40][34]**.

Procédure GRASP()
1. Initialisation : $x^*=\infty$;
2. Tant que(critère d'arrêt non satisfait)faire
   - Construire une solution aléatoire greedy x ;
   - Trouver l'optimum local $x^{\sim}$ dans le voisinage N(x)de x;
   - Si $f(x^{\sim})<f(x^*)$ alors $x^*= x^{\sim}$;

3.return la meilleure solution trouvée x^*.

Figure1. La procedure GRASP

Procédure construction()
1.Initialisation de la solution $S=\emptyset$;
2.Tantque (construction non terminée) faire
   - En utilisant une fonction greedy établir la liste RCL ;
   - Choisir un élément 's' de RCL aléatoirement ;
   - Mettre 's' dans la solution i.e $S=S\cup\{s\}$;
   - Changer la fonction greedy pour prendre en compte S après la m.à.j ;

3.Return la solution x qui correspondant à S ;

Figure2. La phase de construction de GRASP

Procédure recherche locale()
Input : une solution x, un voisinage N(x).
Output : une solution $x^{\sim}$ optimale localement.
1.Tantque (x n'est pas optimale localement)faire
   - Trouver $x^{\sim}\in N(x)$ tel que $f(x^{\sim})<f(x)$. //le voisinage est obtenu en appliquant des flips sur une ou plusieurs variables.
   - $x= x^{\sim}$;

2. return la solution optimale localement $x^{\sim}$.

Figure3. La phase de recherche locale de GRASP

✔**Les algorithmes itératifs:** Partent d'une solution initiale complète (pas forcement très bonne) et cherchent à l'améliorer par des transformations successives.

- **La procédure G-SAT:** La procédure G-SAT de base commence d'une solution générée aléatoirement et change de façon répétée l'assignement d'une variable qui conduit à la plus grande diminution dans le nombre total des clauses non satisfaites.

Procedure G-SAT()
Repeat (Max-tries) fois
Begin
S une instanciation aléatoire.
Repeat (Max-flips) fois (généralement max-flips = N)
- Si S satisfait C alors return (S), stop.
- Soit Xi la variable dont le changement de son assignement donne la plus grande diminution dans le nombre de clauses de C non satisfaites par S.
- S = S avec l'assignement inverse de Xi.

End
End

Figure4. La procédure G-SAT

Deux caractéristiques importantes de G-SAT doivent être notées **[31]**:

➢ Le passage à la solution suivante se fait même si elle est mauvaise que la solution courante ce qui permet d'échapper aux optimums locaux.

➢ Quand la recherche échoue de trouver une solution dans l'espace local, G-SAT redémarre la recherche avec une nouvelle instanciation.

- **Le recuit simulé:** C'est une méthode inspirée par la technique expérimentale du recuit utilisée par les métallurgistes pour obtenir un état solide bien ordonné d'énergie minimale **[19][34]**. L'idée de base est de simuler le comportement d'un ensemble d'atomes initialement équilibré à une température T et soumis à refroidir. Il est bien connu que le refroidissement rapide bloquerait le système dans un état de haute énergie correspondant à un déséquilibre, alors qu'un refroidissement lent conduira le système dans un état équilibré de faible énergie.

Procédure recuit simulé ()
1. generer une solution initiale et une température T aléatoirement
2. tantque(T>0)faire
(a)répéter (nb-iter) fois
- générer un voisin aléatoirement(flipper une variable X_i aléatoire) et évaluer la variation du nombre de clauses non satisfaites ΔE.
- Si $\Delta E>0$ (amelioration de la solution) alors màj la solution courante par la nouvelle solution.
- Si $\Delta E<=0$ alors màj la solution courante par la nouvelle solution avec une probabilité $e^{-\Delta E/\delta T}$ (où δ est une constante).

(b) diminuer la temperature T.
3. Return la solution finale.

Figure5. Procédure du recuit simulé

- **La recherche tabou:** Démarre d'une solution initiale **[34][23]**, et à chaque itération le meilleur voisin est choisi même s'il est mauvais que la solution courante. Pour éviter de cycler une structure de données appelée liste tabou est utilisée, elle contient des informations sur les dernières itérations de l'algorithme et permet de prohiber les solutions récemment visitées. Un élément de base de TS est le critère d'aspiration qui détermine si un mouvement est admissible malgré qu'il soit dans la liste tabou. Des stratégies d'intensification, de diversification, d'oscillation stratégique et de chemin reliant sont utilisées pour améliorer la performance de l'algorithme.

- **Les algorithmes génétiques:** Ils sont basés sur une analogie avec l'évolution des espèces, se fondent sur deux opérateurs: le croisement et la mutation. A chaque itération l'algorithme combine (croise) des solutions (des parents) de la population courante pour générer une nouvelle population (enfants); ensuite une phase de sélection est appliquée pour choisir les meilleurs individus (solutions) parmi les parents et les enfants **[34]**.

- **La recherche dispersée (Scatter search) :** La recherche dispersée est une méta-heuristique récente basée population **[15][17]**, elle consiste à chaque itération à générer une population diversifiée à partir d'une solution initiale appelée semence. De cette population un ensemble "Refset" des meilleures solutions en terme de qualité et de diversité est choisi.

  L'ensemble Refset est ensuite partitionné en sous-ensembles, auxquels des combinaisons linéaires sont appliquées pour créer de nouvelles solutions ; contrairement aux AGs, la combinaison dans la recherche dispersée peut se faire entre 2,3 ou plusieurs solutions.

  Les nouvelles solutions générées sont améliorées par des processus heuristiques ; une solution améliorée peut remplacer la mauvaise solution dans Refset si elle est meilleure en qualité sinon en diversité.

  La génération de nouvelles solutions s'arrête lorsque l'ensemble Refset ne change pas ; dans ce cas, une nouvelle itération est commencée en utilisant la meilleure solution de Refset comme semence.

Des stratégies de bruits aléatoires sont utilisées dans la recherche dispersée pour bénéficier des avantages que présentent les méthodes aléatoires dans la recherche d'une bonne solution.

- **ACO "Ant Colony Optimization":** C'est une nouvelle méta-heuristique**[5]** appropriée pour résoudre les problèmes difficiles de l'optimisation combinatoire. Elle est basée population et inspirée de la nature, exploitant la rétroaction positive aussi bien que l'information locale, et a été appliquée avec succès à une variété de problèmes d'optimisation combinatoire.

ACO consiste en un ensemble d'agents coopérants (fourmis artificielles) et un ensemble de règles qui déterminent la génération, la mise à jour et l'usage de l'information globale et locale dans le but de trouver de bonnes solutions **[6]**.

4. CONCLUSION

La résolution du problème MAX-SAT nécessite des méthodes approchées surtout lorsqu'il s'agit d'une instance de grande taille. Les méta-heuristiques constituent la plus grande partie de ces méthodes et se sont avérées très efficaces.

Le chapitre suivant est consacré aux différents algorithmes de la méta-heuristique ACO qui constituent un nouvel axe de recherche.

La méta-heuristique ACO est une approche assez récente proposée pour la première fois par Marco Dorigo en 1992. Elle est inspirée par l'observation des colonies de fourmis réelles **[14]**.

Un comportement intéressant des colonies de fourmis est le fait qu'elles soient capables de trouver les plus courts chemins entre la source de leur nourriture et leur fourmilière, malgré qu'elles soient aveugles. Cependant une fourmi seule en est incapable.

1. Le principe des fourmis réelles

En marchant de leurs sources de nourriture à la fourmilière et vice versa, les fourmis déposent sur terre une substance nommée "la phéromone", en formant dans leurs chemins une piste de phéromone. Les fourmis peuvent sentir la phéromone et quand elles choisissent leurs chemins, elles tendent à choisir ceux qui sont marqués par une grande concentration de phéromone.

Dans le but d'étudier le comportement des colonies de fourmis **[13]**, une expérience a été réalisée par GAUSS en 1989(**figure 6).** Elle consiste à lier la fourmilière d'une colonie de fourmis et une source de nourriture par un double pont de deux branches de longueurs différentes, les fourmis sont ensuite laissées libres pour se déplacer entre la fourmilière et la source ; et le pourcentage des fourmis qui choisissent l'une ou l'autre branche est observé dans le temps.

On a remarqué qu'après une phase de transition initiale (où les deux branches ont été choisies par le même nombre de fourmis), la branche la plus courte était la plus souvent sélectionnée. En effet les premières fourmis qui arrivent à la source de nourriture sont celles qui ont pris la plus courte branche, alors quand ces fourmis commencent leur voyage de retour, plus de phéromone sera présente sur la petite branche que sur la longue branche, stimulant ainsi les fourmis successives à choisir la plus petite branche.

On doit noter que les fourmis accomplissent leur comportement en utilisant une forme de communication indirecte, via la phéromone déposée, connue par la "stygmergie". Il est possible de parler de la communication "stygmergetique" à chaque fois qu'il y a une communication indirecte modélisée par:

- Des modifications physiques des endroits de l'environnement.
- Accès local à ces endroits par les agents communicants.

Un autre aspect important doit être souligné, c'est le couplage entre l'autocatalysie (rétroaction positive) et l'évaluation implicite des solutions; c-à-d plus le chemin est court, plus tôt la phéromone est déposée par les fourmis, plus les fourmis utilisent le plus court chemin.

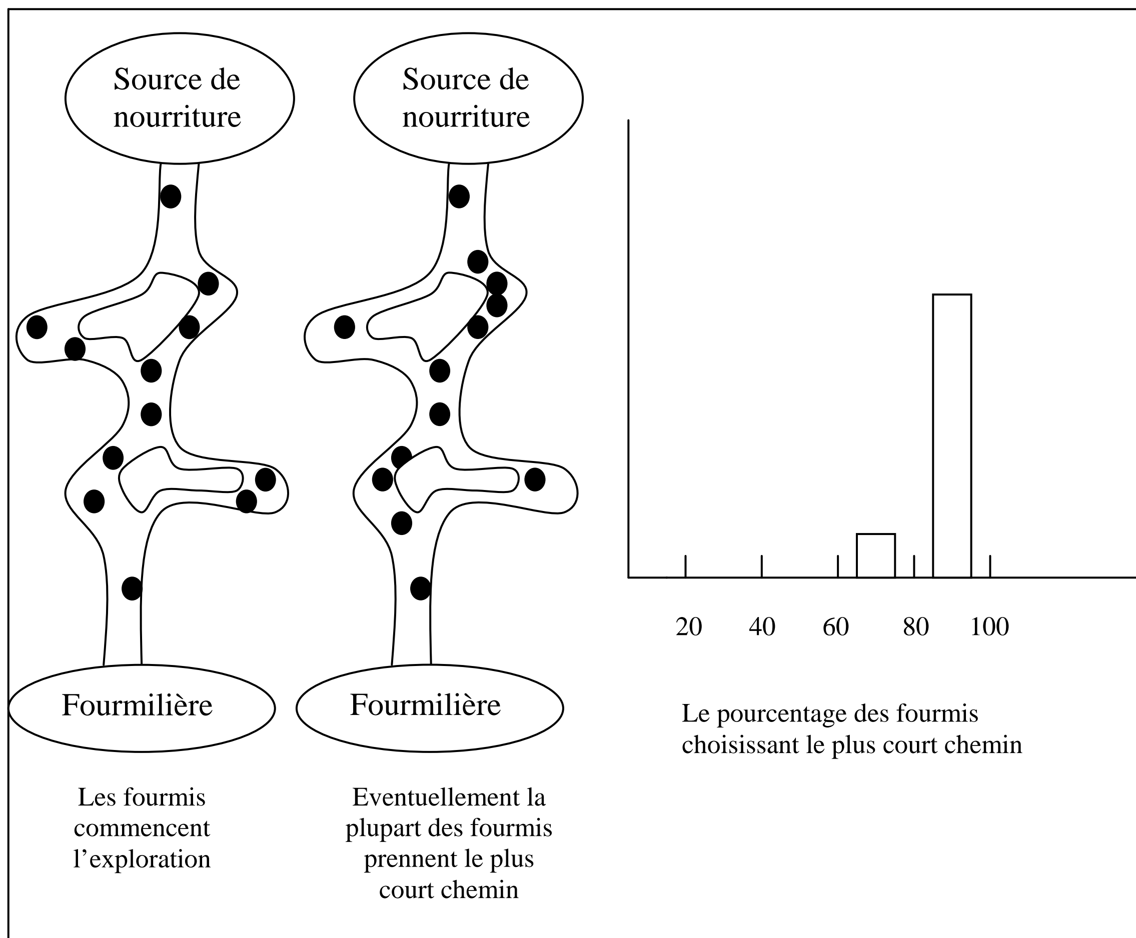


Figure6. Expérience de GAUSS 1989**[13]**

2. L'inspiration des fourmis réelles :

- Comme pour les fourmis réelles, les algorithmes de fourmis sont composés d'une colonie d'entités (des fourmis artificielles) concurrentes et asynchrones coopérant globalement pour trouver une bonne 'solution' au problème en considération.

- Alors que les fourmis réelles déposent dans les endroits visités une substance chimique, la phéromone, les fourmis artificielles changent quelques informations numériques sauvegardées localement dans les endroits du problème qu'elles visitent (la détermination des endroits où la phéromone est déposée dépend du problème étudié, par exemple, dans le TSP la phéromone est déposée sur les arcs du graphe représentant l'instance du problème).Par analogie cette information est appelée 'information phéromone'.
- Dans les algorithmes ACO un mécanisme d'évaporation similaire à l'évaporation de la phéromone réelle modifie l'information phéromone dans le temps. L'évaporation permet de diriger la recherche vers de nouvelles directions sans être contraint par les décisions du passé.
- La recherche d'une solution de coût optimal pour les fourmis artificielles correspond à la recherche du chemin le plus court pour les fourmis réelles.
- Les fourmis artificielles comme les fourmis réelles construisent des solutions en appliquant une politique de décision probabiliste pour se déplacer à travers des états adjacents.

Les fourmis artificielles ont des caractéristiques qui ne se trouvent pas chez les fourmis réelles :

- Les fourmis artificielles ont un état interne privé qui contient la mémoire des actions du passé.
- Les fourmis artificielles déposent une quantité de phéromone qui est une fonction de la qualité de la solution trouvée ce qui ne reflète pas le comportement des fourmis réelles.
- Les algorithmes ACO peuvent être enrichis par des compétences supplémentaires comme l'optimisation locale, retour arrière, ce qui n'est pas le cas pour les fourmis réelles.

3. La méta-heuristique ACO

La méta-heuristique ACO peut être appliquée aux problèmes d'optimisation discrète caractérisés par :

- $C=\{c_1,c_2,\ldots\ldots\ldots,c_{nc}\}$ un ensemble fini de composants.
- $L=\{L_{cicj}/(ci,cj)\in C\times C\}$un ensemble fini de connections(transitions)possibles entre les éléments de C, $|L|<=nc^2$.
- J_{cicj} est le coût associé à la connexion $L_{cicj.}$
- $\Omega\equiv\Omega(C,L,t)$ un ensemble fini de contraintes définies sur les éléments de C et L à l'instant t.
- $S=<c_i,c_j,\ldots\ldots..,c_k,\ldots..>$ une séquence des éléments de C appelée aussi « état » du problème. Ayant deux états S_1 et S_2, on peut dire que S_2 est le voisin de S_1 si S_2 peut être atteint depuis S_1 par une seule étape logique. Le voisinage d'un état S est noté N(S).
- $\Psi=< c_i,c_j,\ldots\ldots..,c_k,\ldots..>$ une séquence des éléments de C est une solution au problème si elle satisfait toutes les contraintes Ω du problème.
- J_Ψ est le coût de la solution Ψ.

Considérons le graphe G=(C, L) associé à une instance d'un problème d'optimisation donné. Les algorithmes ACO peuvent être utilisés pour trouver des chemins (séquences) réalisables dans G de coût minimum tout en respectant les contraintes Ω **[14]**.

Par la suite, nous utilisons une population de fourmis artificielles pour résoudre collectivement le problème d'optimisation en considération.

- Chaque fourmi construit une solution commençant d'un état initial, en se déplaçant à travers les connections de L.
- L'information collectée par les fourmis durant le processus de recherche, est mémorisée dans les pistes de phéromone T_{ij} associées aux connections L_{ij} traversées par ces fourmis. Les connections peuvent aussi avoir une valeur heuristique $\eta_{ij,}$ représentant une information à priori sur l'instance du problème.
- Une fourmi k a une mémoire M^k qui est éventuellement utilisée pour mémoriser des informations sur le chemin tracé, ou pour construire des solutions réalisables.
- Une fourmi k située dans un nœud (état) i se déplace vers un nœud j $\in N(i)$ en utilisant une règle de décision probabiliste. Cette dernière est une fonction de

l'information de phéromone, des valeurs de l'heuristique, de la mémoire M^k et des contraintes du problème.

- En se déplaçant d'un nœud i vers un nœud j la fourmi peut mettre à jour la phéromone T_{ij} sur l'arc (i,j), ceci est appelé la mise à jour « online step by step ».
- Une fois que la fourmi construit sa solution, elle peut retracer le même chemin pour mettre à jour la phéromone sur les arcs traversés avec une quantité relative à la qualité de la solution trouvée ; ce qui est appelé la mise à jour « online delayed ».
- Après la construction de la solution et la mise à jour de la phéromone, la fourmi 'meurt', c-à-d elle est supprimée du système **[13,14]**.

La méta-heuristique ACO peut avoir un comportement supplémentaire appelé « les actions du démon ». Un démon peut sur la base de l'observation de toutes les solutions générées par les fourmis déposer de la phéromone supplémentaire « offline », tout en respectant la phéromone « online » déposée par les fourmis.

Les trois activités principales d'un algorithme ACO (la génération et l'activité des fourmis, l'évaporation de la phéromone et les actions du démon) peuvent nécessiter une certaine synchronisation qui est accomplie par un scheduler des activités.

```
Procédure ACO-méta-heuristique()
tantque (critere-terminaison-non-satisfait)
        Scheduler des activités
            Generation-et-activité-des-fourmis() ;
            Evaporation-de-pheromone() ;
            Actions-du-daemon() ; {optional}
      fin scheduler des activités
 Fin_tantque
Fin procedure
Procedure  generation-et-activté-fourmis()
    tantque(resources-disponibles)
         Scheduler-la-creation-nouvelle-fourmi() ;
         Nouvelle-fourmi-active() ;
    Ftq.
Fin procedure

Procedure nouvelle-fourmi-active() {cycle de vie de fourmi}
 Initialiser-la-fourmi() ;
 M = mise-à-jour-memoire-fourmi() ;
tantque (état-courrent != état-cible)
 P=calculer-probabilités-de-transition(M,contraintes-probléme) ;
 Etat-suivant=appliquer-politique-decision(P,problem-contraintes) ;
 Se-deplacer-à -l'état-suivant(état-suivant)
 Si (mise-à-jour-de-phéromone online step-by-step)
    Deposer-la-pheromone-sur-le-lien-visité() ;
 Fsi
 M= mise-à-jour-état-interne() ;
Ftq
Si (mise-à-jour-de-pheromone online-delayed)
   Evaluer-la-solution () ;
   Déposer-la-pheromone-sur-tous-les-arcs-visités() ;
```

4. Les algorithmes ACO :

Dans les algorithmes ACO les règles de transition, de mise à jour et d'évaporation dépendent du problème traité, pour cette raison nous essayerons de les présenter en relation avec le problème de voyageur de commerce "TSP" qui est la première application des algorithmes ACO.

Un TSP peut être représenté par un graphe complet orienté avec poids G= (V,A,d) avec l'ensemble des nœuds V={1,2, …,n },l'ensemble des arcs A : (i,j)$\in$V$\times$V, et la fonction poids d : A $\rightarrow$N associant un poids entier positif d_{ij} à chaque arc (i,j) donnant la distance entre les villes i et j. Le but est donc de trouver un plus court cycle hamiltonien dans le graphe. La fonction f du coût est donnée par la somme des poids des arcs qui appartiennent à un tour S.

4.1. Système de fourmis 'AS' :

4.1. Système de fourmis 'AS' : C'est le premier algorithme ACO qui a été conçu pour résoudre le TSP **[12]**. Dans AS les fourmis artificielles sont distribuées sur les villes, et chaque fourmi décide indépendamment de la ville suivante à visiter jusqu'à ce qu'elle complète son tour. Chaque fourmi doit garder les villes déjà visitées pour ne pas y revenir, c'est la raison de la nécessité de la mémoire.

Règle de transition : La probabilité avec laquelle une fourmi K choisit de partir de la ville i vers la ville j à la $t^{iéme}$ itération est :

$$P_{ij}^{k}(t) = \frac{[T_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{l \in N_i^K}[T_{il}(t)]^{\alpha}[\eta_{il}]^{\beta}} \quad \text{...(1)}$$

N_i^k est l'ensemble des voisins de i non encore visités.

$T_{ij}(t)$ est la quantité de phéromone sur l'arc(i,j) à l'instant t.

$\eta_{ij} = 1/d_{ij}$ est la valeur de l'heuristique du déplacement de la ville i vers la ville j

α,β sont les paramètres de contrôle des poids relatifs à la phéromone et à l'heuristique.

Le rôle des paramètres α et β est expliqué comme suit : si α=0 , les villes les plus proches sont les plus probables à être choisies, ceci correspond à un algorithme stochastique avide. Si au contraire β=0, seulement l'amplification de la phéromone est utilisée, et ceci conduit à une stagnation rapide. Pour cela un compromis entre les valeurs de l'heuristique et de la piste de phéromone est nécessaire.

Règle de mise à jour : L'évaporation et l'ajout de la phéromone sont implémentés par la règle suivante : $T_{ij}(t) = (1-\rho)T_{ij}(t)+\Delta T_{ij}(t)$ …………………….(2).

$\Delta T_{ij}(t) = \sum_{k=1}^{m} \Delta T_{ij}^{K}(t)$, m est le nombre de fourmis à chaque iteration.

$$\Delta T_{ij}^{K}(t) = \begin{cases} 1/L^{K}(t) & \text{si } (i, j) \in T^{k}(t) \\ 0 & \text{sinon} \end{cases}$$

$\rho \in [0,1]$ est le coefficient d'altération (d'évaporation) de la phéromone.

$T^k(t)$ est le tour construit par la fourmi k à l'instant t, et L^k représente sa longueur.

Les expériences ont montré que l'algorithme AS était performant seulement pour les problèmes de petites tailles, alors que pour les grands problèmes, les meilleures solutions connues ne sont atteintes qu'après un très grand nombre d'itérations. Pour cette raison on a vu l'apparition des versions améliorantes de AS.

4.2. Les variantes de AS

*a. AS avec la stratégie élitiste "ASelite" :*L'idée de la stratégie élitiste dans le contexte de AS **[4,13]** est de donner un appui supplémentaire au meilleur chemin déjà trouvé à chaque itération quand les pistes de phéromone sont mises à jour. Ce chemin est traité comme si un certain nombre de fourmis élitistes l'ont choisi, car il est prometteux que quelques arcs de ce chemin soient une partie de la solution optimale.

L'effet de cette stratégie diminue dans le cas où les longueurs des tours conçus par les fourmis se rapprochent et spécialement quand plusieurs fourmis voyagent sur des bons chemins qui sont sub-optimaux. On a essayé de pallier à ce problème par le AS avec rangement.

b. AS avec stratégie élitiste et rangement "ASrank" : Dans cette stratégie après que toutes les fourmis génèrent leurs tours , elles sont rangées par la longueur des tours, et la contribution de chaque fourmi à la mise à jour de la phéromone est proportionnelle à son rang. En plus, seulement les w meilleures fourmis sont autorisées à mettre à jour la phéromone **[4]**.

c. MAXMIN AS : Il est clair **[41]** qu'une bonne performance de AS pour le TSP doit inclure plus d'exploitation des meilleures solutions trouvées durant la recherche et un mécanisme pour éviter la stagnation rapide de la recherche. Pour cette raison, MMAS diffère de AS en trois aspects **[13] :**

- Les pistes de phéromone sont mises à jour (offline) par le démon : Les arcs qui ont été utilisés par la meilleure fourmi dans l'itération reçoivent de la phéromone supplémentaire.
- Les valeurs de la phéromone sont limités par un intervalle $[T_{min}, T_{max}]$.
- Les pistes de pheromone sont initialisées à leurs valeurs maximales T_{max}.

On a constaté que MMAS appliqué au TSP trouve de meilleurs tours que AS. Cependant pour les grandes instances, les solutions ne sont pas garanties d'être localement optimales **[41].** Ainsi des procédures de recherche locales sont rajoutées pour les tours construits.
Dans ces procédures pour une ville donnée tout le voisinage est scruté et le meilleur voisin qui améliore la solution courante, s'il en existe, est choisi.

4.3. Le système de colonie de fourmis "ACS" : l'algorithme ACS a été introduit par Dorigo et Gambardella en 1996 pour améliorer la performance de AS, Il possède quelques différences importantes **[13]**:

1. La mise à jour de la piste de phéromone est faite par le démon "offline" ; elle concerne "S^+" la meilleure solution de l'itération ou la meilleure solution trouvée depuis le début de l'algorithme. Elle est appliquée selon la règle :

$T_{ij} = (1-\rho)\ T_{ij}(t) + \rho\ \Delta T_{ij}(t)$ ……………………(3).

ρ est le paramètre d'altération de la phéromone.

$\Delta T_{ij}(t) = 1/L^+$

L^+ est la longueur de "S^+".

2. Les fourmis utilisent une règle de décision différente appelée 'règle proportionnelle pseudo aléatoire'. Soit q une variable aléatoire uniformément distribuée sur [0,1], et $q_0 \in [0,1]$ un paramètre réglable. La règle proportionnelle pseudo aléatoire utilisée par la fourmi k située au nœud i pour choisir le nœud $j \in N_i^k$ pour y aller est :

si $q <= q_0$ alors

$$P_{ij}^{k}(t) = \begin{cases} 1 & \text{si } j = \arg\max \{[T_{ij}(t)][\eta_{ij}]^{\beta}\} \\ 0 & \text{sinon} \end{cases} \quad \ldots\ldots\ldots\ldots\ldots(4)$$

sinon $(q > q_0)$

$$P_{ij}^{k}(t) = \frac{[T_{ij}(t)][\eta_{ij}]^{\beta}}{\sum_{l \in N_i^K} [T_{il}(t)][\eta_{il}]^{\beta}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots(5)$$

argmax représente l'élément (la ville) j qui possède la plus grande valeur $[T_{ij}(t)][\eta_{ij}]^{\beta}$
Cette règle de décision a une double fonction ; quand $q<= q_0$ la règle exploite la connaissance disponible sur le problème alors que quand $q> q_0$ elle opère une exploration favorisée.

3. Les fourmis effectuent seulement une mise à jour online étape par étape de la phéromone pour favoriser l'émergence d'autres solutions que la meilleure la plus connue en appliquant la règle suivante :

$T_{ij}(t) = (1-\rho)\, T_{ij}(t) + \rho T_0 \quad [0<\rho<=1]$……………..……(6)

T_0 est la valeur initiale de la phéromone.

Une fourmi se déplaçant d'une ville i vers j met à jour la phéromone sur l'arc (i,j) par la règle(6).

4. ACS exploite une structure de données appelée « liste candidate », qui est une liste des villes préférées à être visitées à partir d'une ville donnée. Quand une fourmi veut se déplacer d'une ville i, elle choisit une ville parmi celles de la liste candidate ; les autres villes ne sont examinées que si toutes les villes de la liste candidate ont été visitées .

ACS a été testé sur des TSPs différents, et comparé à d'autres méta-heuristiques. Sa performance était la meilleure en terme de qualité de solution ainsi que de temps CPU nécessaire.

5. Conclusion

Dans ce chapitre nous avons évoqué les aspects essentiels de la méta-heuristique ACO qui sont inspirés du comportement des fourmis réelles. Nous avons utilisé le problème de voyageur de commerce 'TSP' comme exemple pour expliquer les différentes règles nécessaires à la mise en œuvre des algorithmes ACO. Dans le chapitre qui suit nous allons proposer une adaptation de l'algorithme ACS au problème MAX-W-SAT.

L'efficacité des algorithmes ACO dépend énormément de la façon d'utilisation de l'information phéromone : les éléments où elle est déposée, le moment et la quantité avec laquelle elle est mise à jour, ainsi que son poids par rapport à l'heuristique dans la règle de décision.
Dans ce chapitre nous allons détailler tous ces points essentiels, en proposant deux stratégies différentes pour adapter l'algorithme ACS au problème MAX-W-SAT.
La première stratégie consiste à faire démarrer chaque fourmi d'une solution initiale complète générée aléatoirement, et de lui appliquer des flips répétitifs afin de construire une solution finale ; alors que la deuxième consiste à démarrer d'une solution initialement vide, et de l'étendre progressivement en lui rajoutant à chaque itération un littéral jusqu'à arriver à une solution complète.

1. Stratégie1

Dans cette stratégie le problème MAX-W-SAT est caractérisé par:

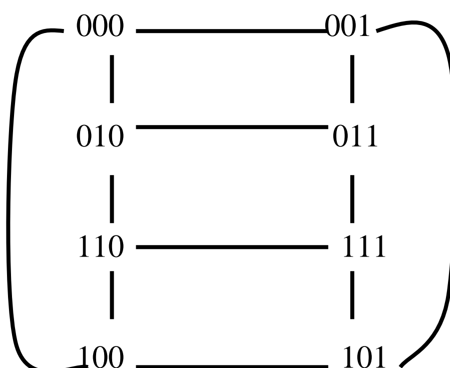
- chaque solution S de l'instance MAX-W-SAT est une séquence de N bits (littéraux); chaque bit i (0/1) représente la valeur de la variable booléenne X_i de l'instance du problème.
- Un ensemble C de toutes les solutions possibles, c.à.d toutes les combinaisons possibles avec N variables booléennes $|C|=2^N$.
- Un ensemble L des connections entre les éléments de C. deux solutions S_1, S_2 de C sont connectées par un lien si et seulement si elles différent d'un seul bit. $|L|=N*2^{N-1}$.
- Le déplacement d'une solution S_1 vers sa voisine S_2 se fait par le flip d'une seule variable de S_1 "1-flip".
- Le coût (la fonction objective) d'une solution S "coût(S)" est la somme des poids des clauses non satisfaites par S.

  $$coût(S) = \sum_{cl \notin Sat(S)} poids(cl) \qquad \text{Sat(S) est l'ensemble des clauses satisfaites par S.}$$

Cette stratégie consiste à appliquer l'algorithme ACS à une instance du problème MAX-W-SAT, avec l'objectif de minimiser la somme des poids des clauses non satisfaites (équivalent à maximiser la somme des poids des clauses satisfaites).

Exemple

Avec une instance de trois variables logiques, on aboutira à un graphe de 8 sommets, (chacun d'eux représente une solution possible) et 12 arrêtes (chaque arrête relit deux solutions ayant un seul littéral différent) :



1.1. L'algorithme général

Debut

Initialiser la phéromone.

<u>*Pour*</u> i=1 <u>*à*</u> MaxIter <u>*faire*</u>

<u>*Pour*</u> k=1 <u>*à*</u> NbAnts <u>*faire*</u>

Générer une solution initiale S_0 aléatoirement.

Construire une solution S^k.

Evaluer la solution S^k.

Mettre à jour la phéromone "online delayed".

<u>*Fait*</u>

Déterminer la meilleure solution trouvée dans cette itération.

Mettre à jour la phéromone "offline".

<u>*Fait*</u>

Fin

1.2 Initialisation de la phéromone

La phéromone est déposée sur chaque littéral de l'instance ; elle représente l'information liée à l'utilisation précédente de ce littéral ; en effet plus la quantité de phéromone est grande, plus la probabilité du choix de cet élément dans la construction des prochaines solutions est grande. L'information de phéromone est implémentée en utilisant une table de taille N×2 (N est le nombre des variables et 2 sont les valeurs {0,1} que peut prendre chaque variable). Cette table est initialisée à une valeur petite τ_0=0.1.

1.3 Génération des solutions initiales

Chaque fourmi génère sa solution initiale S_0 aléatoirement en donnant à toutes les variables de l'instance une valeur de vérité *false*, elle génère un nombre aléatoire *nb*<=N (N est le nombre de variables de l'instance), ensuite elle choisit aléatoirement *nb* variables auxquelles elle change la valeur de vérité à *true*.

1.4 Construction de la solution

Chaque fourmi k démarre d'une solution initiale S_0 générée aléatoirement, et construit sa solution S^k itérativement en se déplaçant à travers une séquence de solutions voisines en appliquant des flips.

A chaque itération, la fourmi choisit une variable X_i à flipper avec une probabilité P_{xi}^k en utilisant la règle de décision proportionnelle pseudo aléatoire suivante :

Soit q_0 un paramètre réglable dans [0,1] et q une variable aléatoire uniformément distribuée sur [0,1].

Si q<=q_0 alors :

$$P_{xi}^k(t) = \begin{cases} 1 & \text{si (i, j)} = \text{argmax}\left\{phero[i,j]^\alpha \, heur[i,j]^\beta\right\} \\ 0 & \text{sinon} \end{cases} \quad \text{.........................(1)}$$

sinon

$$P_{xi}^k(t) = \frac{phero[i,j]^\alpha \, heur[i,j]^\beta}{\sum_{x_l \in N^k} phero[l,m]^\alpha \, phero[l,m]^\beta} \qquad \forall x_l = 1 - m \text{..................(2)}$$

X_i=1-j à l'instant t. ($j \in \{0,1\}$).Le calcul de la probabilité de la variable X_i se fait donc en utilisant la valeur de phéromone et de l'heuristique de son complément.
α, β sont les paramètres qui contrôlent respectivement la phéromone et l'heuristique. N^k est l'ensemble des variables qui ne sont pas tabou.

$$heur[i,j] = \frac{\sum_{cl \in sat(xij)} poid(cl)}{\text{poids total des clauses.}} \quad ....(3)$$

Sat(X_{ij})est l'ensemble des clauses satisfaites lorsque la variable X_i=j.

L'utilisation de cette règle de décision permet de contrôler le degré de l'exploration de la recherche. En effet, l'augmentation de la valeur de q_0 mène à une concentration de la recherche sur les littéraux des meilleures solutions (car dans ce cas, le littéral qui possède la plus grande valeur $phero[i,j]^{\alpha}$ $heur[i,j]^{\beta}$ est choisi); alors que de petites valeurs de q_0 permettent le choix des autres littéraux, favorisant ainsi l'exploration de la recherche.

Procedure de construction

Entrée : une solution initiale S_0

Sortie : une solution construite S^k

Debut

Pour i=1 *à* NbFlip *faire*

 Générer une variable aléatoire (q).

 Si (q<=q0) *alors*

  Déterminer la variable "x" qui n'est pas tabou et qui a la probabilité maximale.

  Flipper"x"

  Insérer "x" dans la liste tabou.

 Sinon

  Choisir une variable "x" non tabou aléatoirement.

  Calculer sa probabilité P(x) par la formule (2).

  Générer une valeur aléatoire VAL∈[0,1].

  Si P(x)> VAL alors

   Flipper"x"

   Insérer "x" dans la liste tabou.

  Fsi

 Fsi

Fait.

Fin.

Après chaque flip d'une variable "x", elle est insérée dans une liste taboue, pour ne pas la choisir plus d'une fois pendant un nombre fini d'itérations; le but est d'éviter de boucler pendant la construction de la solution.

La performance des algorithmes ACO peut être considérablement améliorée par une recherche locale appliquée aux solutions construites par les fourmis. Dans notre algorithme avant chaque mise à jour de la phéromone une recherche locale est appliquée à la solution concernée par la mise-à-jour en utilisant la procédure suivante.

1.5 Procedure de recherche locale

Entrée : une solution S à améliorer.
Sortie : une solution S*.
Debut
S*=S.
pour step=1 *à* MaxStep *faire*
Choisir une clause non satisfaite “cl” de S aléatoirement.
Choisir une variable X_i de cl aléatoirement
Flipper (X_i)
Si coût(S)<coût(S*) *alors* S*=S *fsi.*
Fait
Fin.

1.6 Mise à jour de la phéromone

Elle consiste en une évaporation de la phéromone pour tous les littéraux, suivie par un ajout concernant seulement les littéraux de la solution en question.

- L'évaporation est appliquée par la règle suivante
  phero[i,j]=(1-ρ)*phero[i,j] …………………… ………… (4)∀i=1..N et ∀j=0..1.
  $\rho \in [0,1]$ est le coefficient d'altération de la phéromone.

- La quantité de phéromone rajoutée aux littéraux qui composent la solution S* obtenue par la recherche locale est proportionnelle au coût(S*). Dans la mise à jour online retardée, l'ajout est appliqué par la règle :
  phero[i,j]=phero[i,j]+ρ*(1-(cout(S*)/poids-total-des-clauses))......(5) $\forall (X_i=j) \in S^*$.

  Alors que dans la mise à jour offline est appliquée par la règle :
  phero[i,j] = phero[i,j]+ ρ*(coût(bestsol)/coût(S*)) ……..(6) $\forall (X_i=j) \in S^*$.
  bestsol est la meilleure solution trouvée depuis le début de l'algorithme.

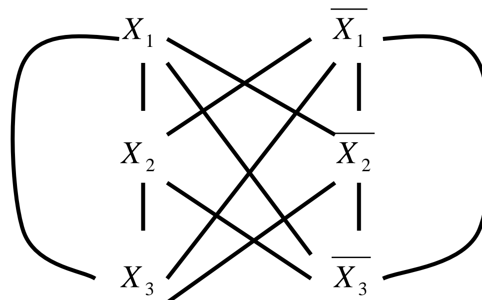
2. Stratégie2

Cette stratégie diffère de la première principalement dans la méthode de construction de la solution. Alors que dans la première les fourmis se déplaçaient d'une solution vers une autre en flippant une variable à chaque itération; dans cette stratégie elles se déplacent d'un littéral vers un autre en choisissant successivement des littéraux à rajouter à la solution en construction, jusqu'à arriver à une solution finale complète. Cette façon de procéder change la façon par laquelle le problème max-w-sat est caractérisé :

- L'ensemble C des composants contient tous les littéraux possibles ;
  C= {c_{ij} $\forall$i=1..N et j=0..1} ($c_{i1}=X_i$ et $c_{i0}=\overline{X_i}$). |C|=2*N.
- L'ensemble L des connections entre les éléments de C. deux composants c_{ia}, c_{jb} de C sont connectés si et seulement si i≠j. |L|=(2*N)(N-1).
- Une solution S au problème max-w-sat est une séquence <c_i,c_j,…,c_k,…> de N composants de C
- Deux solutions S_1, S_2 sont voisines si S_2 peut être obtenue par le flip d'un seul composant (littéral) de S_1 "1-flip".
- Le coût d'une solution S est la somme des poids des clauses non satisfaites ; le but est donc d'avoir une solution de coût minimal.

Exemple

Avec une instance de trois variables logiques, on aboutira à un graphe de 6 sommets, (chacun d'eux représente un littéral) et 12 arrêtes :



Le même algorithme ACS de la première stratégie a été appliqué dans la deuxième avec deux différences principales : la première réside dans la façon de construction des solutions; en effet, dans cette stratégie les fourmis construisent leurs solutions de façon incrémentale. Chaque fourmi "k" commence la construction de la solution à partir d'un composant choisi aléatoirement; à chaque itération elle décide du composant c_{ij} à rajouter à sa solution en appliquant la règle de transition pseudo aléatoire décrite dans la première stratégie.

Pour assurer que la solution construite soit réalisable, le composant choisi et son complément sont insérés dans une liste taboue pour ne pas être choisis encore une fois ; Il est nécessaire alors dans ce cas que la taille de la liste taboue soit 2*N.

la seconde différence est la mise à jour online retardée qui est remplacée par la mise à jour online étape par étape; à chaque fois qu'une fourmi rajoute un composant c_{ij} à sa solution, elle applique la règle de mise à jour suivante :

phero[i,j]=(1-ρ)*phero[i,j]+ ρ* τ_0 ……….(7) (τ_0=0.1 est la valeur initiale de phéromone)

Procedure de construction
Debut
Choisir un composant “c_{ij}” aléatoirement pour commencer la construction.
Insérer “c_{ij}” et son complément dans la liste tabou.
Appliquer la règle (7).
Pour i=1 *à* N-1 *faire*
Générer une variable aléatoire (q)
Si (q<=q0) *alors*
Déterminer le composant “c_{ij}” qui n’est pas tabou et qui a la probabilité maximale.
Sinon
Choisir un composant “c_{ij}” aléatoirement avec une probabilité calculée par la formule (2).
Fsi
Rajouter “c_{ij}” à la solution en construction.
Insérer “c_{ij}” et son complément dans la liste tabou.
Appliquer une mise à jour étape par étape (formule7).
Fait.
Fin.

3. Conclusion

Dans ce chapitre nous avons décrits deux façons différentes pour adapter l’algorithme ACS au problème MAX-W-SAT.
L’efficacité de chacune d’elles ne peut émerger qu’après la détermination des valeurs les plus convenables des paramètres empiriques. Ceci sera détaillé dans le chapitre5 où plusieurs tests seront fait pour régler les paramètres nécessaires, et comparer les performances des deux stratégies.

La complexité de calcul des algorithmes ACO séquentiels, entrave leur utilisation pour résoudre les grands problèmes. Par exemple pour le problème de voyageur de commerce, avec un nombre de fourmis m=n (n est le nombre de ville) on a une complexité d'ordre $O(m^3)$**[5]**.
La génération et l'évaluation des fourmis (solutions), sont les parties les plus essentielles des algorithmes séquentiels et sa source principale du coût de calcul. Elles offrent un degré de dépendance faible, ce qui rend la structure de l'algorithme bien adéquate pour une exécution concurrente.
Dans la suite de ce chapitre nous allons présenter quelques notions essentielles du calcul parallèle suivies par les méthodes de parallélisation de l'algorithme ACS.

1. le calcul parallèle

Parmi les applications dont la réalisation fait appel à la programmation parallèle il y a les systèmes d'exploitation ; l'algorithmique numérique utilisée souvent dans les applications avioniques, le traitement d'images, la biologie moléculaire, l'analyse sismique…etc ; les applications d'intelligence artificielle qui doivent en général traiter des bases de données (ou des bases de connaissances) de tailles considérables en des temps raisonnables, ou qui font appel aux techniques heuristiques pour des buts d'optimisation ; et d'autres applications qui doivent respecter des contraintes temporelles absolument draconiennes comme l'ordonnancement des taches, la compréhension de la parole, la planification du déplacement des robots. Pour toutes ces applications le calcul parallèle est le seul moyen d'obtenir une performance acceptable **[3]**.

1.1 Concept de processus

Le concept de processus est apparu initialement dans les systèmes informatiques pour décrire l'exécution d'un programme séquentiel, avec ses données, par un processeur. Ce concept est inhérent au calcul parallèle car ce dernier est le travail d'un ensemble de processus, chacun avec son propre algorithme, et parfois exécutant le même algorithme sur des données différentes.

1.2 Relations entre processus

Dans un programme parallèle, les processus peuvent être :

1.2.1 Indépendants : ce qui permet d'activer tous les processus simultanément et de les exécuter parallèlement. Le programme se termine avec l'achèvement de tous ses processus.

1.2.2 En compétition : on parle de processus en compétition lorsqu'ils sont en conflit par l'utilisation de ressources qui sont toujours en nombre limité dans le système. Les solutions à ce problème reposent sur une sérialisation convenable des diverses exécutions des processus en compétition.

1.2.3 En coopération : c'est le cas où un ensemble de processus participent à une tâche commune et qui doivent échanger des informations pour mener à bien cette tache. On peut distinguer :

- *la coopération par partage :* lorsque des processus d'une application peuvent mettre à jour des données que d'autres processus peuvent

également consulter et modifier. Pour conserver la cohérence de ces données, il est nécessaire d'en contrôler l'accès par des mécanismes d'exclusion mutuelle.

- *Coopération par échange explicite d'information :* se fait généralement par des primitives de communication (d'envoie et de réception de messages).

1.3 Synchronisation des processus

Selon qu'une primitive de communication implique l'arrêt du processus l'exécutant (primitive bloquante) ou la continuation de ce processus (primitive non bloquante), on peut définir divers modes de synchronisation **[3]**.

1.3.1 Mode synchrone

C'est le cas où la communication ne peut être effectuée que lorsque le processus expéditeur est prêt à effectuer l'opération d'envoie au processus destinataire, et que le destinataire lui-même est prêt à recevoir l'information (envoi bloquant et réception bloquante).

1.3.2 Mode asynchrone

Dans ce mode, dés que l'expéditeur est prêt à envoyer un message, il effectue cet envoi, même si le récepteur n'est pas prêt à recevoir le message. Le récepteur devant consommer un message ne se bloque que si celui-ci n'a pas été produit au préalable, dans le cas contraire, il consomme le message et continue son exécution (envoi non bloquant, réception bloquante).

Il existe un troisième mode de synchronisation plus ésotérique qui pourrait se rapprocher du traitement d'interruption, où lorsqu'un récepteur demande à recevoir un message et que ce dernier n'est pas prêt, il continue en séquence (envoi non bloquant et réception non bloquante).

Dans un programme parallèle, les processus peuvent être crées statiquement ou dynamiquement. Dans le premier cas les processus sont en nombre fixe et sont crées une fois pour toutes lors de l'initialisation du programme. Alors que dans la création dynamique le nombre de processus présents lors de l'exécution d'un programme peut varier d'une exécution à une autre en fonction des données traitées.

2. ACO parallèle

Dans un algorithme de fourmis parallèle, la colonie de fourmis peut être divisée en sous colonies, et chacune d'elles est affectée à un processeur. Apres chaque itération locale (une génération de fourmis), les sous colonies échangent les informations concernant les solutions trouvées ; Ensuite, chaque sous colonie calcule la nouvelle matrice de phéromone et continue son exécution. La plupart des implémentations parallèles des algorithmes ACO suivent cette approche, et diffèrent seulement dans la granularité, et si les calculs des nouvelles matrices de phéromone sont effectués localement au niveau de chaque sous colonie, ou centralement par un processeur maître qui distribue les nouvelles matrices aux esclaves.

- La parallèlisation "fine grained" faite par Blondi et Bondanza **[5][35]** avec un modèle maître/esclave où chaque fourmi était affectée à un processeur n'a pas donné des résultats expérimentaux impressionnants ; et devient très mauvaise en augmentant la dimension du problème, ceci est dû à la fréquence et au volume élevés de la communication. De meilleurs résultats ont été obtenus par la deuxième implémentation "coarse grained" faite par Bullheimer **[5][13][35]** où une sous colonie de fourmis est assignée à un processeur.
- Une autre approche "coarse grained" a été proposée aussi par Bullheimer**[5]**, elle consiste à n'effectuer un échange d'information entre les sous colonies de fourmis qu'après k générations de fourmis. Alors que cette approche est attendue à réduire la communication globale considérablement, de bonnes et prometteuses valeurs obtenues durant les itérations locales peuvent être ignorées par les autres sous colonies ; pour cette raison, le rapport des itérations locales/globales doit être choisi rigoureusement.
- Dans **[42]** Stutzle a comparé la qualité des solutions obtenues par plusieurs courtes exécutions indépendantes, avec la qualité d'une seule longue exécution. Il a conclu que sous certaines conditions, les courtes exécutions donnent de meilleurs résultats. Les courtes exécutions indépendantes ont aussi l'avantage d'être facilement faites en parallèle, et il est aussi possible d'utiliser différents ensembles de paramètres pour chacune d'elles.

D'autres recherches ont été faites pour étudier le type et la qualité de l'information qui doit être échangée entre les sous colonies et comment cette information doit être utilisée pour la mise à jour de la matrice de phéromone.

- Kruger et Merkle ont prouvé qu'il est préférable d'échanger seulement les meilleures solutions trouvées que de faire un échange de toutes les matrices de phéromone. Et dans **[35]** Middendorf, Reishle et Schmerk ont comparé l'échange de la meilleure solution globale ; l'échange circulaire des meilleures solutions locales (un voisinage virtuel est établi entre les sous colonies formant un anneau orienté) ; l'échange circulaire des migrants (les processus forment un anneau orienté virtuel et seulement les m meilleures fourmis sont échangées) et l'échange circulaire des meilleures solutions locales avec migrants. Ils ont trouvé que les meilleurs résultats sont obtenus lorsque les meilleures solutions locales sont échangées entre les colonies voisines (c-a-d la deuxième et la quatrième méthode d'échange). Ces résultats renforcent ceux obtenus par Kruger, confirmant q'un échange de petites quantité d'informations est plus utile pour la recherche.

3. parallélisation de ACS-SAT

Dans la suite de ce chapitre, nous allons proposer deux méthodes différentes pour paralléliser l'algorithme ACS-SAT de la première stratégie décrite dans le chapitre3.

L'idée de base des deux méthodes consiste à partager la colonie de fourmis entre plusieurs processus, en ayant une sous colonie pour chaque processus, et de lancer les processus en parallèle.

L'équilibrage de charge entre N processus est accompli en assignant au processus j (j=1à N) les fourmis m_i (i=1à nbAnt) tel que j = i mod N. Ainsi chaque processus aura la même charge, et chaque sous colonie d'un processus se comporte comme une colonie complète.

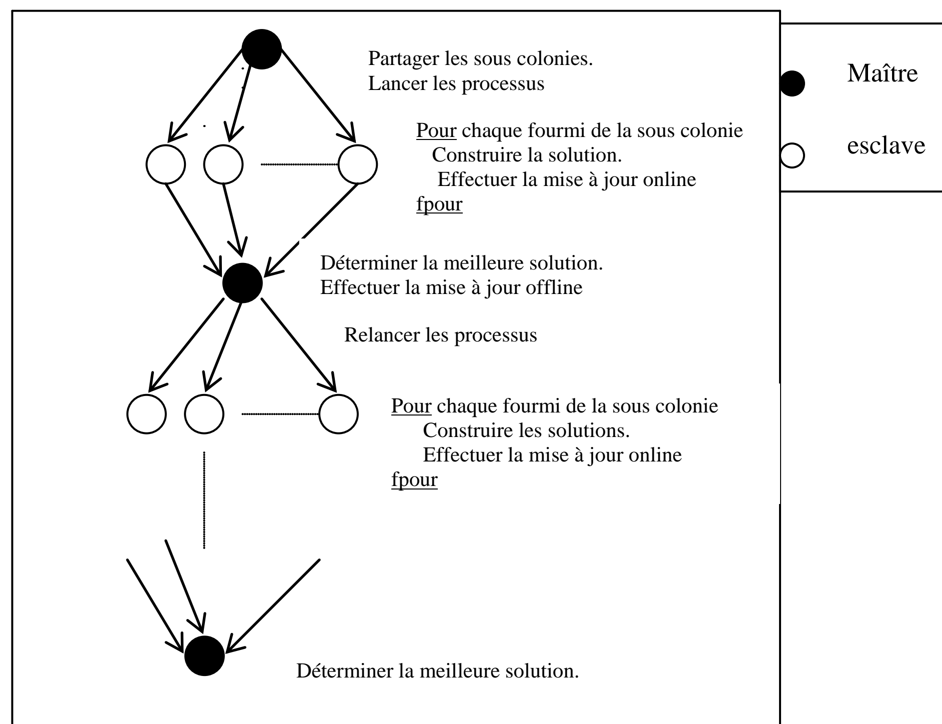
Le mode de synchronisation et la façon que les processus utilisent pour communiquer, font la différence entre les deux méthodes synchrone et asynchrone proposées.

3.1 Méthode synchrone

Cette méthode consiste à utiliser le calcul parallèle synchrone, qui exige des points de synchronisation entre les processus, tout en particularisant le rôle de l'un comme maître et des autres comme esclaves.

Le processus maître crée l'ensemble des processus esclaves, et leurs assigne les fourmis pour commencer à travailler (construire des solutions et mettre à jour la phéromone) en parallèle.

Lorsque tous les esclaves terminent, le processus maître cherche la meilleure solution trouvée par ces derniers, applique la mise à jour offline de la phéromone, ensuite, relance les esclaves de nouveau.



Méthode synchrone

La communication utilisée entre les processus dans cette méthode n'est pas explicite. Elle est fait par partage de mémoire, en gardant la matrice de phéromone dans une zone critique où une seule mise à jour est permise à la fois.

Algorithme synchrone

<table>
<tr>
<td>
//code exécuté par le maître
Début
. Initialiser la matrice de phéromone.
. Créer les processus esclaves
. Affecter les fourmis aux processus.
<u>Pour</u> i = 1 <u>à</u> MaxIter <u>faire</u>
. Lancer les processus esclaves.
. Attendre la terminaison des esclaves.
. Rechercher la meilleure solution trouvée.
. Mettre à jour offline la phéromone.
<u>Fait</u>
. Arrêter tous les processus esclaves.
Fin
</td>
<td>
//code exécuté par l'esclave "i"
Début
<u>Tantque</u> non fin <u>faire</u>
Attendre le signal du père
<u>Pour</u> toute fourmis "k" de "i" <u>faire</u>
. Générer une solution initiale
. Construire la solution S^k
. Mettre à jour online la phéromone.
<u>Fait</u>
<u>Fait</u>
Fin
</td>
</tr>
</table>

Les processus esclaves partagent la matrice de phéromone, et chaque processus doit accéder en lecture et en écriture à la matrice ; ceci est un problème similaire à celui des lecteurs/rédacteurs où les contraintes suivantes doivent être respectées pour maintenir la cohérence de la matrice :

- Nombre de rédacteurs =0 <u>et</u> nombre de lecteurs>=0.
- Nombre de rédacteurs =1 <u>et</u> nombre de lecteurs=0.

Le traitement de tel problème sous entend des mécanismes de synchronisation et d'exclusion mutuelle. L'utilisation des moniteurs permet de garantir l'exclusion mutuelle ; et pour la synchronisation on utilisera les signaux qui sont modélisés en java par les méthodes "wait()" (qui met un processus en attente passive) et "notify()" (qui réveille un processus en attente).
Chaque lecture (écriture) doit être précédée par un appel de la fonction StartRead (StartWrite) et suivie par la fonction EndRead (EndWrite).

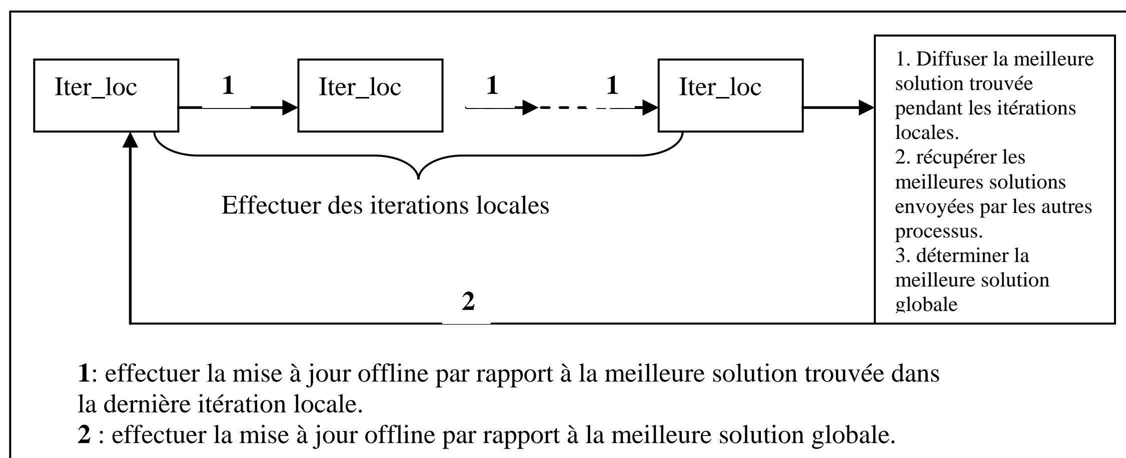
StartRead() ***Début*** tantque (ecriture=vrai) faire Wait()//attente passive Fait Nblect :=nblect+1 Si (nblect=1) alors lecture=vrai fsi ***Fin***	**StartWrite()** ***Début*** tantque ((lecture =vrai) ou (ecriture=vrai)) faire Wait() Fait Ecriture :=vrai ***Fin***
EndRead() ***Début*** nblect :=nblect-1 si (nblect=0) alors lecture=faux fsi notifyAll()//reveiller tous les processus en attente ***Fin***	**EndWrite()** ***Début*** ecriture:=faux notifyAll() ***Fin***

3.2 Méthode Asynchrone

La méthode de parallélisation asynchrone permet une communication directe entre les processus tout en évitant les points de synchronisation de la première méthode, théoriquement, ceci provoquera la réduction du temps d'exécution.

Chaque processus dans cette méthode, effectue un nombre d'itérations locales à la sous colonie qui lui est assignée. Lorsqu'il termine, il envoie la meilleure solution trouvée localement à tous les autres processus, et reçoit les meilleures solutions proposées par les autres.
Chaque processus choisit la meilleure solution entre la sienne et celles qui lui sont envoyées, et effectue la mise à jour offline de la phéromone, ensuite continue le calcul en commençant de nouvelles itérations locales.

Avec plus d'espace mémoire, nous pouvons éviter les points de synchronisation dus à l'accès simultané à la matrice de phéromone ; et ceci en créant une matrice pour chaque processus (sous colonie de fourmis), le but est de réduire le temps d'exécution.



Étapes suivies par chaque processus dans la méthode asynchrone

Algorithme asynchrone

```
//code exécuté par chaque processus i
Début
  cpt :=0
  Pour i=1 à MaxIter faire
     Pour toute fourmi k du processus i faire
       . Générer une solution initiale
       . Construire la solution s^k
       . Mettre à jour online de la phéromone.
     Fait
     cpt :=cpt +1
     Si cpt = LocalIter alors
       . Échanger les informations avec les autres processus
       . Rechercher la meilleure solution.
       . cpt :=0 ;
     fsi
    Mettre à jour la phéromone offline
Fait
```

- LocalIter est le paramètre empirique qui détermine la fréquence de communication entre les processus.

4. Environnement parallèle

Selon les besoins fonctionnels des développeurs et l'environnement d'exécution de leurs applications, ils ont le choix entre l'utilisation de PVM (Parallel Virtual Machine) et MPI (Message Passing Interface).

La base de conception de PVM était la notion de machine virtuelle ; un ensemble de hôtes hétérogénes connectés par un réseau apparaît logiquement à l'utilisateur comme un seul large ordinateur parallèle. Dans la conception de PVM la portabilité était considérée beaucoup plus importante que la performance car les recherches étaient concentrées sur les problèmes de tolérances aux pannes et d'hétérogéinité de la machine virtuelle.
Au contraire à PVM, l'impulsion de développement de MPI était que chaque concepteur de processeur massivement parallèle MPP, créait ses propriétés de passage de message ; dans ce scénario il était impossible d'écrire une application parallèle portable. MPI est voulue être un standard de technique de passage de messages qui peut être utilisé sur les différents MPPs.
L'utilisation des MPPs revient au besoin d'une très grande performance, ceci est devenu le but de conception de MPI. Pour cette raison MPI est attendue d'être toujours plus rapide que PVM sur les MPPs.

Si une application va être développé et exécutée sur un seul MPP, alors il est conseillé d'utiliser MPI pour sa grande performance de communication et sa portabilité sur d'autres MPPs. MPI a un ensemble plus riche de fonctions de communication, alors elle est favorisée quand une application est structurée pour exploiter des modes de communication spéciaux non disponibles dans PVM ; l'exemple le plus cité est l'envoie non bloquant.
Quelques sacrifices ont été fait dans la spécification MPI dans le but de maintenir une performance de communication supérieure. Les plus notables sont le manque d'interopérabilité entre les implémentations MPI ; et l'impossibilité d'écrire des applications tolérantes aux pannes en MPI.
Parce que PVM est construite sur le concept de la machine virtuelle, elle est favorisée quand l'application à développer va se dérouler sur un ensemble d'hôtes liés, particulièrement lorsque les hôtes son hétérogènes. PVM contient des gestionnaires de ressources et des fonctions de contrôle des processus ce qui est nécessaires pour créer des applications portables qui s'exécutent sur des groupes (clusters) de Workstations et de MPPs.
Plus l'ensemble des hôtes est grand, plus la caractéristique de tolérance aux pannes de PVM devient importante. Ce qui permet d'écrire de longues applications PVM qui peuvent continuer à s'exécuter même quand des hôtes ou des tâches échouent, ou lorsque les charges changent dynamiquement, chose qui est extrêmement importante pour le calcul distribué hétérogène **[22]**.

5. Conclusion

Nous avons proposé dans ce chapitre deux algorithmes, à savoir, synchrone et asynchrone pour paralléliser la première stratégie du chapitre3.
Les deux algorithmes consistent à partager la colonie de fourmis entre plusieurs processus de façon équitable, et de lancer les processus pour évoluer en parallèle. Ils diffèrent dans la façon dont les processus se synchronisent, ainsi que dans la méthode et la fréquence de communication entre eux. Les deux algorithmes seront comparés dans le chapitre suivant, en exposant les résultats numériques obtenus.

Dans ce chapitre, notre intérêt portera sur l'analyse de performance des algorithmes décrits dans les chapitres précédents. Des tests seront effectués sur des instances MAX-W-SAT de la classe "Johnson" ; ces problèmes sont caractérisés par un ensemble de 100 variables, et un nombre de clauses qui varie entre 800 et 950 ; toutes les clauses ont un poids positif inférieur à 1000.
Vu l'absence d'une machine parallèle, nous nous sommes contenté d'un PC (Pentium 533MHZ, 128MO de RAM) fonctionnant sous le système d'exploitation LINUX pour évaluer les différentes méthodes ACS qui ont été implémentées en JAVA (voir annexe).

Les tests seront fait pour :

- Régler les paramètres empiriques qui ont une grande influence sur la performance des algorithmes. Une valeur de paramètre est jugée bonne si elle donne les meilleurs résultats en terme de qualité de solution comme premier critère, et en temps d'exécution comme second critère.
- Montrer l'importance de la communication entre les sous colonies de fourmis dans la méthode parallèle asynchrone, et son influence sur la qualité de la solution.
- Effectuer des comparaisons entre les résultats des algorithmes proposés dans ce travail, ainsi que ceux obtenus par d'autres solveurs SAT.

Nous n'avons pas étudié l'accélération des algorithmes parallèles dans ce travail, à cause de l'indisponibilité de machine parallèle ; et leur exécution sur une machine monoprocesseur ne fait qu'augmenter le temps d'exécution en augmentant le degré du parallélisme.

1. Détermination des paramètres

Les paramètres empiriques des deux stratégies séquentielles, de la règle de décision et des règles de mise à jour jouent un rôle très important dans la qualité des solutions construites.
Dans cette phase nous effectuons une série de tests pour fixer les valeurs des paramètres qui ont une grande influence sur la performance des l'algorithmes. Toutes les exécutions ont été faites avec 10 fourmis, et les résultats mentionnés sont atteints par environ 10 exécutions

- q_0 est le paramètre qui définit le taux d'intensification et de diversification de l'algorithme. Les grandes valeurs de q0 permettent d'intensifier la recherche car elles favorisent les variables qui ont une probabilité maximale ; alors que les petites valeurs dirigent l'algorithme vers de nouvelles régions, permettant ainsi de diversifier la recherche. q0 a été fixé à 0.9 pour la première stratégie et à 0.95 pour la seconde.

Strategie1

instance	0.1	0.5	0.7	**0.9**	1
Jnh1	419276	420732	420892	420925	420330
Jnh10	417753	419290	420513	420840	418668
Jnh17	418626	419267	420625	420925	420797
Jnh210	391898	393204	394238	394238	393778
Jnh215	391423	392879	393904	394000	393263
Jnh220	393225	393202	394021	394200	392482
Jnh301	442615	443726	444703	444842	444215
Jnh306	441457	442969	444779	444838	442713
Jnh310	441099	442419	443968	444065	443004

Table1. Les valeurs obtenues pour les différentes valeurs de q0 de la strategie1.

Strategie2

instance	0.1	0.5	0.7	0.8	0.9	**0.95**	1
Jnh1	419441	419453	420021	420024	420079	420115	419712
Jnh10	410145	416731	417399	417626	418891	419115	420131
Jnh15	412156	414691	416031	418596	419932	420057	420046
Jnh201	390393	392841	393337	392781	394135	393686	393896
Jnh210	384939	387697	387850	391824	392430	393533	393213
Jnh220	389670	389401	392492	392040	392973	393594	393362
Jnh301	439659	443589	441450	442094	443854	444282	444210
Jnh306	438191	440611	441337	441505	442823	443377	443049
Jnh310	437755	439949	439236	440860	441674	442558	442175

Table2. Les valeurs obtenues pour les différentes valeurs de q0 de la strategie2.

- ***eRate*** représente le taux d'évaporation de la phéromone. Ce paramètre contrôle la quantité de phéromone évaporée et rajoutée à chaque mise à jour. Les grandes valeurs de eRate permettent un taux élevé d'évaporation et de rajout de phéromone et vis versa. le paramètre a été fixé à 0.7 pour la première stratégie et 0.8 pour la seconde.

Strategie1

instance	0.1	0.3	0.5	**0.7**	0.9
Jnh1	420330	420829	420925	420925	420915
Jnh10	418668	420531	420354	420840	420590
Jnh5	419284	420488	420197	420584	420522
Jnh215	393263	393951	393951	394000	394150
Jnh220	392482	393985	394129	394200	394038
Jnh210	393778	394238	394238	394238	394238
Jnh303	442263	444004	444244	444299	444246
Jnh306	442713	444377	444638	444838	444527
Jnh310	443004	444122	443812	444065	444161

Table3. Les valeurs obtenues pour les différentes valeurs de eRate de la stratégie1.

Strategie2

instance	0.3	0.5	0.7	**0.8**	0.9
Jnh1	420780	420545	420747	420722	420806
Jnh10	420154	419740	420529	420225	420216
Jnh15	419993	420104	420167	420201	420059
Jnh201	394238	393976	394161	394238	394154
Jnh205	393975	393993	393975	393975	393897
Jnh212	393662	393540	393927	394004	393958
Jnh215	393607	393353	393621	393722	393321
Jnh301	444517	443956	444668	444562	444537
Jnh306	444533	444416	444515	444559	444233
Jnh310	443569	443556	443530	443889	443873

Table4. Les valeurs obtenues pour les différentes valeurs de eRate de la stratégie2.

- ***NbFlip*** est le nombre de flips effectués pendant la construction de la solution dans la première stratégie. Il est fixé à N*2/3 (N est le nombre de variables).

Strategie1

instance	N/3	N/2	**2*N/3**	N
Jnh1	420366	420819	420915	420856
Jnh10	419032	420389	420590	420264
Jnh15	419369	419960	420510	420463
Jnh210	393469	394238	394238	394238
Jnh215	392212	393613	393951	393768
Jnh220	392911	393937	394169	394147
Jnh306	443902	444137	444642	444838
Jnh310	442068	443674	443947	443909

Table5. Les valeurs obtenues pour les différentes valeurs de NbFlip.

- ***MaxStep*** est le nombre d'itérations effectuées dans la recherche locale. Il est fixé à 30 pour la première stratégie et à 60 pour la deuxième.

Strategie1

instance	10	20	**30**	40
Jnh1	420925	420889	420925	420909
Jnh8	420280	420463	420342	419806
Jnh10	420649	420417	420840	420786
Jnh210	394238	394238	394238	394238
Jnh215	393858	393967	394000	394014
Jnh220	394178	394108	394200	394038
Jnh303	444129	443954	444299	444299
Jnh306	444838	444726	444838	444838
Jnh310	444313	444097	444065	444203

Table6. Les valeurs obtenues pour les différentes valeurs de MaxStep de la stratégie1.

2. Strategie2

instance	10	20	30	40	50	**60**	70
Jnh1	420120	420087	420187	420650	420391	420811	420585
Jnh10	419895	420157	420007	420113	420216	420225	420116
Jnh15	420052	420057	419976	420101	419464	420121	419975
Jnh210	393951	394238	394199	393951	394238	394238	394238
Jnh212	393134	393463	393645	393705	393748	393805	393753
Jnh220	393839	393690	393754	393944	393961	393985	393451
Jnh301	444749	444626	444470	444194	444854	444586	444711
Jnh306	444144	444515	444629	444315	444515	444559	444310
Jnh310	443449	442835	442934	443261	443168	443742	442996

Table7. Les valeurs obtenues pour les différentes valeurs de MaxStep de la stratégie2.

- ***MaxIter*** est le nombre d'itérations de l'algorithme, fixé à 160 pour la première stratégie et à 80 pour la seconde.

Strategie1

instance	100	120	140	**160**	180
Jnh1	420856	420812	420856	420925	420925
Jnh10	420463	420416	420430	420840	420590
Jnh17	420776	420925	420925	420925	420925
Jnh210	394238	394238	394238	394238	394238
Jnh215	393951	393756	393942	394000	394150
Jnh220	393985	394129	394238	394129	394129
Jnh301	444786	444790	444790	444842	444786
Jnh306	444592	444467	444646	444838	444838
Jnh310	443731	444256	444001	444065	444277

Table8. Les valeurs obtenues pour les différentes valeurs de MaxIter de la stratégie1.

Strategie2

instance	20	40	60	**80**	100
Jnh1	420616	420104	420492	420811	420806
Jnh10	419649	419585	420023	420216	419836
Jnh17	420066	420571	420551	420925	420711
Jnh205	393750	393857	393698	393964	393903
Jnh217	393819	393967	394229	394229	394229
Jnh220	393533	393580	393842	393988	394009
Jnh301	443842	443654	444238	444684	444583
Jnh306	444016	444292	444326	444559	444472
Jnh308	442611	442858	443419	443650	443639

Table9. Les valeurs obtenues pour les différentes valeurs de MaxIter de la stratégie2.

- ***(α,β)*** ces deux paramètres contrôlent respectivement l'influence de la phéromone et de l'heuristique dans la règle de décision. Ils sont fixés à (1,0) pour les deux stratégies.

Strategie1

instance	(1,1)	(2,1)	(1,2)	(5,1)	(1,5)	(8,1)	(1,0)
Jnh1	420856	420879	420806	420909	420770	420806	**420925**
Jnh10	420581	420464	420273	420581	419630	420590	**420840**
Jnh12	420861	420724	**420925** (39,34)	**420925** (44,78)	420405	420706	**420925** (37,1)
Jnh17	420790	420830	420776	**420925** (50,52)	420078	420892	**420925** (36,42)
Jnh5	420515	420526	420474	420426	419076	420609	**420742**
Jnh205	394075	394075	393975	**394238** (53,17)	393824	**394238** (36,4)	**394238** (34,39)
Jnh215	393951	393951	393951	393951	393498	393951	**394150**
Jnh301	444700	444744	444638	444790	444465	**444842** (40,6)	**444842** (36,10)
Jnh303	**444360**	443936	444139	444146	442692	444351	444157
Jnh306	444671	**444838** (37,92)	**444838** (35,15)	**444838** (50,90)	444579	**444838** (44,01)	**444838** (37,65)
Jnh310	443668	444031	443963	443668	441641	444037	**444313**

Table10. Les valeurs obtenues pour les différentes valeurs de α et β de la stratégie1. (Les valeurs entre parenthèse représentent le temps d'exécution).

Strategie2

instance	(1,1)	(2,1)	(4,1)	(6,1)	(8,1)	(1,2)	(1,4)	**(1,0)**
Jnh1	420465	420585	420404	420585	420780	420581	420095	420811
Jnh10	419703	419109	419884	420333	419550	419629	419031	420225
Jnh15	419736	419542	420057	419871	419755	420097	419677	420121
Jnh17	420299	420489	420188	420525	420388	420388	420273	420925
Jnh205	393975	393879	393964	393972	394032	393972	393484	393975
Jnh210	393951	393717	393951	393678	393951	393762	393557	394238
Jnh220	393904	393882	393904	393963	393930	393765	393560	394086
Jnh301	444381	444690	444483	444037	443919	443932	443445	444711
Jnh306	442694	443804	444375	444144	444004	444356	443855	444559
Jnh310	442694	443859	442207	443105	443351	443859	442160	443889

Table11. Les valeurs obtenues pour les différentes valeurs de α et β de la stratégie2.

On remarque des tables 10 et 11 que de bonnes solutions sont obtenues quand la valeur de α est beaucoup plus grande que celle de β (5,1),(6,1) et (8,1). Il est clair aussi que les valeurs (1,0) donnent les meilleurs résultats en qualité de solutions ainsi qu'en temps d'exécution ; ces valeurs correspondent à une recherche sans heuristique. On peut donc déduire que l'utilisation de l'heuristique dans les deux stratégies ACS n'est pas aussi bénéfique que son utilisation pour le TSP, où elle était cinq fois plus importante que l'information phéromone.

2. Comparaison des deux stratégies séquentielles

Après avoir fixer les paramètres empiriques des deux stratégies proposées, nous allons comparer leurs résultats numériques dans la table suivante.
La première valeur dans la colonne des solutions correspond à la meilleure solution trouvée (solution maximale), et la seconde à la moyenne de 20 exécutions.

instance	Strategie1		Strategie2	
	solution	Tps-exec	solution	Tps-exec
Jnh1	420925 420829	26.56	420811 420285	75.53
Jnh10	420786 420210	28.62	420225 419619	66.81
Jnh12	420925 420419	26.77	420704 419877	67.10
Jnh14	420733 420469	29.54	420544 419750	68.12
Jnh19	420497 420002	30.37	420349 419312	59.80
Jnh202	394044 393747	28.53	393676 393051	66.62
Jnh205	394238 393955	23.81	393975 393460	6611
Jnh207	394229 393858	28.24	393834 393156	67.11
Jnh210	394238 393935	24.91	394238 393348	53.56
Jnh212	394227 393985	28.49	394004 393315	59.85
Jnh214	394163 393358	28.49	393860 392790	65.35
Jnh217	394238 394154	26.06	394238 393539	63.42
Jnh219	394053 393498	29.19	393708 393014	66.19
Jnh220	394205 394048	28.89	394086 393380	64.64
Jnh301	444842 444668	31.26	444711 443932	64.64
Jnh302	444419 443624	32.17	443554 443539	68.80
Jnh304	444533 444083	32.16	444061 442724	67.12
Jnh307	444083 443520	31.46	443805 442943	68.01
Jnh309	444578 444225	30.22	444257 443407	67.42
Jnh310	444274 443686	31.64	443889 442454	67.66

Table12. Comparaison des deux stratégies séquentielles.

Les valeurs de la table montrent que les meilleurs résultats sont obtenus par la première stratégie de l'algorithme ACS pour presque toutes les instances testées, alors que la deuxième stratégie n'atteint les bonnes valeurs que pour quelques instances avec des temps d'exécutions nettement plus grands.

Ceci montre la différence qui peut exister entre deux stratégies d'un même algorithme ACO, chose due à l'impact de la méthode de construction des solutions sur la qualité des résultats obtenus.
Vu sa performance, la première stratégie a été choisie par la suite pour lui appliquer les méthodes de parallélisation synchrone et asynchrone proposées dans le chapitre4.

3. La communication entre les sous colonies de fourmis

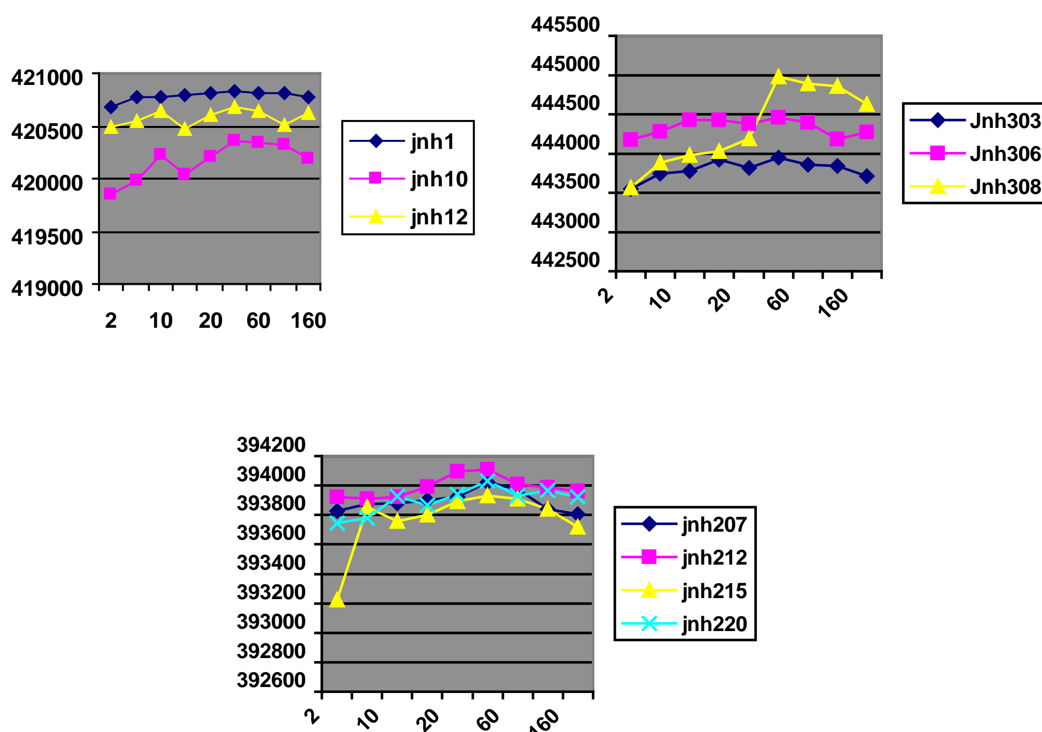
Dans l'algorithme parallèle asynchrone, le degré de communication entre les sous colonies de fourmis est contrôlé par le paramètre empirique "LocalIter" ; il représente le nombre d'itérations locales effectuées avant chaque échange d'information. Les petites valeurs de localIter impliquent un taux élevé de communication et vis versa. Nous rappelons que l'information diffusée par chaque sous colonie lors d'une communication est la meilleure solution trouvée pendant les dernières itérations locales qu'elle a effectué.

Vu le manque de précision rencontré en comparant les meilleures solutions trouvées pour les différentes valeurs de LocalIter, nous avons dû utiliser la solution moyenne de 20 exécutions ; cette mesure nous a aidé à choisir les bonnes valeurs du paramètre de communication, et de conclure son influence sur la qualité des solutions atteintes.

	2	5	10	16	20	30	60	80	160
1	420925 420676	420889 420781	420925 420782	420915 420791	420925 420819	420925 **420832**	420909 420803	420925 420802	420873 420782
10	420529 419843	420446 419974	420656 420229	420529 420040	420590 420212	420590 **420357**	420564 420338	420590 420318	420578 420194
12	420925 420491	420724 420546	420925 420645	420925 420470	420925 420595	420925 **420675**	420871 420648	420819 420513	420871 420614
14	420505 420165	420716 420244	420765 420475	420747 420332	420608 420434	420824 420459	420781 **420498**	420824 420345	420700 420313
207	394020 393825	394164 393873	394178 393881	394065 393900	394238 393922	394238 **394024**	394229 393976	394142 393838	393950 393735
212	394192 393923	394159 393909	394111 393990	394227 393990	394227 394095	394238 **394107**	394216 394011	394111 393986	394227 393962
215	393829 393224	394150 393862	394075 393757	394082 393799	394082 393892	394150 393852	394150 **393911**	394104 393842	394150 393720
220	394140 393744	394076 393779	394140 393927	394086 393864	394102 393940	394238 **394029**	394238 393929	394153 393965	394083 393922
303	444133 443548	444084 443745	443975 443776	444272 443925	444351 443818	444338 **443949**	444191 443860	444174 443840	444018 443711
306	444838 444177	444726 444282	444838 444424	444783 444428	444838 444377	444829 **444457**	444838 444397	444597 444181	444838 444274
308	444107 443565	444266 443879	444367 443980	444568 444019	444258 444030	444460 **444188**	444263 443983	444222 443894	444362 443858

Table13. L'influence de la frequence de communication entre les sous colonies de fourmis.

Remarque : tous les résultats des algorithmes parallèles (synchrone et asynchrone) sont obtenus en utilisant 8 processus.



On constate, pour presque toutes les instances testées, qu'en augmentant les valeurs de "LocalIter" (ce qui provoque la réduction de la fréquence des échanges d'informations) la moyenne des solutions trouvées s'améliorent ; mais au delà de la valeur 30 elle commence à se dégrader. De ce fait la valeur choisie pour ce paramètre est 30 ; cette valeur représente donc le meilleur rapport des itérations locales/globales pour notre algorithme asynchrone, où le taux d'intensification et de diversification est équilibré.

4. Comparaison entre les différents algorithmes

En comparant les résultats numériques (table14) des différents algorithmes de ce travail, on peut remarquer que :

- Les solutions maximales atteintes pour les deux méthodes séquentielle et parallèle asynchrone se rapprochent, et sont toutes les deux assez performantes ; alors que la version parallèle asynchrone de l'algorithme parallèle atteint des moyennes meilleures que celles de la version séquentielle.
- Le temps d'exécution des algorithmes parallèles est supérieur que celui du séquentiel ; ceci est paradoxal à ce qui est attendu théoriquement, et est dû à leur exécution sur une machine séquentielle.
- L'algorithme parallèle synchrone peut atteindre de bonnes solutions maximales, mais avec des moyennes moins intéressantes que celles des deux versions asynchrone et séquentielle.
- Le temps de réponse de la méthode synchrone est plus grand que celui de l'asynchrone ; ceci revient au nombre élevé de points de synchronisation de la méthode synchrone.

Ces remarques nous permettent de prétendre qu'avec la disponibilité d'une machine parallèle on peut avoir un algorithme parallèle au moins aussi performant que l'algorithme séquentiel, avec des temps d'exécution plus réduits.

	Séquentiel (strategie1)		Synchrone		Asynchrone	
	Solution	tps-exec	Solution	tps-exec	Solution	tps-exec
Jnh1	420925 420829	26.56	420925 420706	110.97	420925 **420832**	88.10
Jnh12	420925 420419	26.77	420925 420348	102.21	420925 **420671**	87.82
Jnh14	420733 **420469**	29.54	420824 420346	113.67	420824 420460	87.10
Jnh16	420911 420698	30.17	420914 420569	114.61	420911 **420751**	93.40
Jnh19	420497 420002	30.37	420455 419846	122.19	420497 **420254**	88.76
Jnh201	394238 **394232**	10.90	394238 394177	61.09	394238 394205	49.83
Jnh203	394119 393397	29.16	393766 393164	124.46	394105 **393631**	83.95
Jnh208	394159 393598	27.78	394013 393501	122.49	394159 **393675**	82.16
Jnh211	393863 393302	28.81	393979 393225	121.01	393836 393470	84.23
Jnh214	394163 393413	27.68	394013 393472	120.55	394163 **393495**	82.69
Jnh217	394238 394154	26.06	394238 394185	108.46	394238 **394215**	87.54
Jnh219	394053 393498	29.19	394059 393527	129.45	394053 393643	84.01
Jnh301	444842 444668	31.26	444842 444501	128.67	444842 **444678**	94.48
Jnh303	444299 443634	32.94	444351 443823	108.35	444318 **443960**	84.60
Jnh305	443714 443017	32.83	443471 442788	122.28	443857 **443291**	95.30
Jnh307	444083 443520	31.46	444083 443515	128.87	444083 **443690**	91.01
Jnh309	444578 444225	30.22	444578 **444249**	126.35	444578 444227	98.02

Table14. Comparaison des différents algorithmes implémentés.

5. comparaison avec d'autres méta-heuristiques

	Sol-optimale	ACS-SAT		GRASP		SS-SAT	
		Solution	Tps-exe	Solution	Tps-exe	Solution	Tps-exe
Jnh1	420925	**420925**	**26.56**	420632	1525.2	420892	203.99
Jnh10	420840	**420786**	**28.62**	420463	964.4	420479	901.47
Jnh11	420753	**420701**	**113.77**	420398	1330.1	420141	439.82
Jnh12	420925	**420925**	**26.77**	420518	564.1	420701	250.65
Jnh13	420816	**420816**	**29.95**	420592	567.0	420716	930.12
Jnh14	420824	**420824**	**113.67**	420401	982.6	420491	298.22
Jnh15	420719	420580	88.33	420429	899.2	**420632**	624.63
Jnh16	420919	**420914**	**114.61**	420809	18.7	420889	716.51
Jnh17	420925	**420925**	**29.36**	420712	1062.8	420794	401.95
Jnh18	420795	**420775**	**90.52**	420289	560.5	420404	129.64
Jnh19	420759	**420497**	**30.37**	420307	287.3	420330	403.10
Jnh201	394238	**394238**	**10.90**	394154	1099.6	**394238**	161.72
Jnh202	394170	**394044**	**28.53**	393680	261.2	393924	269.99
Jnh203	394119	**394119**	**29.16**	393446	1352.3	393875	294.10
Jnh205	394238	**394238**	**23.88**	393890	1163.5	394060	203.37
Jnh207	394238	**394229**	**28.24**	394030	41.8	394228	313.58
Jnh208	394159	**394159**	**27.78**	393859	890.1	393771	137.19
Jnh209	394238	**394238**	**28.17**	393959	1740.0	**394238**	463.31
Jnh210	394238	**394238**	**24.91**	393950	219.3	394067	394.46
Jnh211	393979	**393979**	**121.01**	393529	281.8	393742	408.04
Jnh212	394238	**394227**	**28.49**	394011	171.5	394082	758.00
Jnh214	394163	**394163**	**27.68**	393737	1272.2	394152	1159.15
Jnh215	394150	**394150**	**27.04**	393818	350.8	393942	202.51
Jnh216	394226	**394144**	**121.78**	394042	1215.7	393806	181.43
Jnh217	394238	**394238**	**26.06**	**394232**	438.8	**394238**	799.40
Jnh218	394238	**394238**	**22.56**	394009	825.2	394189	166.47
Jnh219	394156	**394059**	**129.45**	393792	1308.3	393800	530.35
Jnh220	394238	**394205**	**28.89**	393951	1055.1	393985	597.70
Jnh301	444854	**444842**	**31.26**	444612	347.6	**444842**	1267.63
Jnh302	444459	**444459**	**32.17**	443906	46.6	443895	698.77
Jnh303	444503	**444351**	**108.35**	444063	1046.7	444223	437.91
Jnh304	444533	**444533**	**32.16**	444310	142.5	**444533**	1175.98
Jnh305	444112	443857	95.30	**444112**	1465.8	443594	463.33
Jnh306	444838	**444838**	**96.44**	444603	1003.0	444515	604.53
Jnh307	444314	**444083**	**91.01**	443836	972.3	443662	288.14
Jnh308	444724	**444568**	**95.51**	444215	607.9	444250	768.57
Jnh309	444578	**444578**	**98.02**	444273	564.6	444483	662.58
Jnh310	444391	444274	32.64	444010	1290.4	**444313**	240.01

Table15. Comparaison des qualités de solutions et des temps d'exécutions.

Dans la table15 nous comparons les meilleures solutions obtenues par les algorithmes ACS (séquentiel, synchrone et asynchrone) et ceux de Grasp et de la Scatter Search. Les valeurs de la table montrent l'efficacité et la performance des algorithmes ACS pour presque toutes les instances testées.

6. Conclusion

D'après les résultats numériques obtenus dans ce chapitre, les algorithmes séquentiel (strategie1), parallèle synchrone et parallèle asynchrone de ACS s'avèrent bien efficaces pour la résolution du problème MAX-W-SAT. Nous avons aussi pu constater que la version synchrone était plus gourmande en temps de calcul que les deux autres méthodes, et que l'algorithme asynchrone, avec une bonne exploitation de la communication, peut arriver à une meilleure performance, tout en réduisant le temps d'exécution synchrone.

Plusieurs travaux de recherche ont été réalisés dans le domaine de la parallélisation des méta-heuristiques ces dernières années ; leurs résultats étaient intéressants et variaient en fonction de la méta-heuristique, du modèle de la parallélisation adopté et du type de la machine parallèle utilisée. Ces études ont montré que les méta-heuristique les plus convenables et qui peuvent être très performantes par la parallélisation sont celles basées population où la recherche est accomplie en utilisant un ensemble d'agents artificiels ; ce qui est la cas pour la méta-heuristique ACO.

La méta-heuristique d'optimisation par les colonies de fourmis "ACO" est une approche d'optimisation inspirée du comportement des fourmis réelles, elle consiste à lancer plusieurs fourmis artificielles pour la recherche d'une solution optimale ; tout en appliquant des règles qui dirigent le déplacement des fourmis d'un état vers un autre, et aussi pour mettre à jour les pistes empruntées par chacune d'elles.
Le premier algorithme ACO était le système de fourmis "AS" qui a été dédié à la résolution du problème de voyageur de commerce "TSP". Plusieurs versions améliorantes ont été ensuite proposées, comme la AS avec la stratégie élitiste "ASELITE", la version basée rangement "ASRANK", le Max-Min AS "MMAS" et le système de colonies de fourmis "ACS".

Dans cette étude nous avons utilisé l'algorithme ACS ; l'une des meilleures versions de ACO pour résoudre le problème de satisfiabilité maximale pondérée "MAX-W-SAT". Dans ce but nous avons développé deux stratégies séquentielles qui diffèrent dans la façon utilisée par les fourmis pour construire les solutions et mettre à jour la phéromone. La comparaison entre les résultats des deux stratégies a montré la grande influence des différences déjà cités et leur impact sur la qualité des résultats obtenus.

Une seconde étape de ce travail était l'étude de la parallélisation de l'algorithme ACS ; en effet, la structure des algorithmes ACO contient un aspect parallèle naturel du fait que chaque fourmi construit sa solution indépendamment des autres, ce qui rend l'algorithme facile à paralléliser en permettant l'amélioration de sa performance.

Le modèle de parallélisation des algorithmes ACO le plus utilisé est celui qui consiste à partager la colonie de fourmis en sous colonies, et d'affecter chacune d'elles à un élément de calcul différent ; les détails concernant les informations échangées entre les sous colonies, la fréquence de communications entre elles et les modes de synchronisation utilisés sont à l'origine des différences de performances pouvant être obtenues.
Les deux méthodes de parallélisation que nous avons proposées et implémentées rejoignent ce modèle. Alors que la première est synchrone et consiste à faire des échanges d'informations par partage de mémoire ; la deuxième est une méthode asynchrone permettant une communication explicite, faite entre les sous colonies à une fréquence déterminée empiriquement.

La phase expérimentale nous a permis de révéler l'importance et l'influence de la communication entre les sous colonies sur la qualité de la recherche, et de déduire que, dans le cas de notre algorithme parallèle asynchrone, une communication modérée est plus bénéfique qu'une communication intensifiée.
De plus, les tests faits sur les différentes instances et comparés aux résultats des autres méta-heuristiques montrent l'efficacité et la performance de l'algorithme ACS dans ses deux états séquentiel et parallèle dans la résolution du problème MAX-W-SAT.

D'ailleurs une étude plus poussée avec des tests effectués sur une machine parallèle, et appliqués à des instances MAX-W-SAT de plus grandes tailles, pourrait éventuellement apporter plus de conclusions sur l'accélération et l'adaptation de nos algorithmes parallèles ; chose qui n'a pas été réalisée à cause des limites des temps et de matériel.
Et vu la diversité des solutions atteintes par les algorithmes ACO, et l'aspect non déterministe de la recherche dans cette méta-heuristique, il serait intéressant d'envisager une parallélisation par des exécutions parallèles indépendantes avec l'utilisation de différentes valeurs de paramètres pour chacune d'elles ; ceci pourrait faciliter la tâche de l'évaluation des performances.

1. Le langage JAVA

Java est un langage orienté objet multi-plate-forme, conçu pour être facilement appris. Les caractéristiques fondamentales du langage permettent de définir java plus précisément :

- Java s'affranchit des plates-formes : il fonctionne en mode interprété, par opposition aux langages compilés, et peut s'exécuter sur de nombreux systèmes d'exploitation.
- Java est résolument basé sur la technologie objet et emprunte de nombreux éléments au C++.
- Le langage propose un mode de fonctionnement adapté aux applications réseaux et Internet.
- Java offre la capacité de créer simplement des applications multitâches.

L'un des objectifs majeurs de java est de réunir l'industrie informatique derrière un standard de développement afin de s'affranchir des particularités des systèmes d'exploitations. Cela revient à offrir une portabilité et une capacité de réutilisation maximales (d'où le célèbre maxime de Sun :
"Ecrivez une fois, exécutez n'importe où", traduction de " Write once, run anywherer").

Aussi les applications java fonctionnent au sein de machines virtuelles, véritables interpréteurs assurent la neutralité et l'isolation du langage vis-à-vis des systèmes d'exploitation.

2. les threads

Java est un langage multi processus, c'est-à-dire capable d'exécuter plusieurs séquences d'instructions (tâches) concurremment. Chaque séquence est appelée Thread (processus légers).

Création des threads

Il existe deux moyens pour créer des threads dans un programmes java ; les deux méthodes passent par l'écriture d'une méthode run() décrivant les instructions que doit exécuter un thread. Cette méthode est soit intégrée dans une classe dérivée de la classe Thread, soit intégrée dans n'importe qu'elle classe qui doit alors implémenter l'interface Runnable. La seconde méthode est très utile car elle permet d'ajouter les fonctionnalités des threads à une classe existante, dans une classe dérivée.

Start() : est une méthode qui permet de démarrer effectivement le thread sur lequel elle est invoquée, ce qui va provoquer l'appel de la méthode run() du thread. Cet appel est obligatoire pour démarrer l'exécution d'un thread. En effet la création d'un objet de la classe thread ou d'une classe dérivée de la classe thread ne fait que créer un objet sans appeler la méthode run().

Synchronisation des threads

Tous les threads d'une même machine virtuelle partagent le même espace mémoire, peuvent donc avoir accès à n'importe quelles méthode ou variable d'objets existants. Ceci est très pratique mais dans certains cas, nous pouvons avoir besoin d'éviter que deux threads n'aient accès n'importe quand à certaines données. De tels cas nécessitent des mécanismes de synchronisation et d'exclusion mutuelles. Le mécanisme d'exclusion mutuelle présent dans l'environnement java des le moniteur, que l'on implémente grâce au modifieur "Synchronized" ; et la synchronisation peut être assurée par les signaux qui sont modélisées par les méthodes wait() et notify().

• ***Utilisation de Synchronized :*** soient une ou plusieurs méthodes METHODEI() déclarées SYNCHRONIZED, dans une classe CLASSE1 et un objet OBJET1 de la classe CLASSE1 : comme tous objet java comporte un verrou permettant d'empêcher que deux threads différents n'aient accès simultané à un même objet , quand l'une des méthodes METHODEI() est invoquée sur OBJET1 et que celui-ci est déjà verrouillé par un autre thread , le système met le thread courant dans l'état bloqué, tant que OBJET1 est verrouillé. Une fois que OBJET1 est d déverrouillé, le système remet ce thread dans l'état exécutable, pour qu'il puisse essayer de verrouiller OBJET1 et exécuter METHODEI().

• ***Utilisation de Wait() et Notify() :*** comme il est expliqué dans le paragraphe précèdent , SYNCHRONIZE permet d'éviter que plusieurs threads aient accès en même temps à un même objet, mais ne garantit pas l'ordre dans lequel les threads vont exécuter ces méthodes. Pour cela, il existe plusieurs méthodes de la classe Objet qui permettent de mettre en attente volontairement un thread sur un objet (avec les méthodes wait()), et de prévenir des threads en attente sur un objet que celui-ci est à jour (avec les méthodes notify() ou notifyAll()).
Quand wait() est invoquée sur un objet OBJET1, le thread courant perd le contrôle et est mis en attente, et l'ensemble des verrous d' OBJET1 est retiré. Comme chaque objet, java mémorise l'ensemble des threads mis en attente sur lui, le thread courant est ajouté à la liste des threads en attente de OBJET1. OBJET1 étant déverrouillé, un des threads bloqués parmi ceux qui désiraient verrouiller OBJET1, peut passer dans l'état exécutable et exécuter une méthode ou un bloc SYNCHRONIZED sur OBJET1.

Un thread THREAD1 mis en attente est retire de la liste d'attente de OBJET1, quand une des trios raisons suivantes survient:

1. THREAD1 a été mis en attente en donnant en argument à wait() un délai qui a fini de s'écouler.
2. le thread courant a invoqué Notify() sur OBJET1, et THREAD1 a été choisi parmi tous les threads en attente.
3. le thread courant a invoque NotifyAll() sur OBJET1.

THREAD1 est mis alors dans l'état exécutable, et essaye de verrouiller objet1, pour continuer son exécution. Quand il devient le thread courant, l'ensemble des Verrous qui avait été enlevé d'objet1 à l'appel de wait(), est remis sur objet1, pour que thread1 et objet1 se retrouvent dans le même état qu'avant l'invocation de wait().

Remarque : un thread mis en attente en utilisant la méthode wait() sans arguments sur un objet, ne peut redevenir exécutable qu'une fois qu'un autre thread a invoqué Notify() ou notifyAll() sur ce même objet.

3. Exemple

Dans cet exemple nous montrons comment les threads se synchronisent pour accéder à la matrice de phéromone.
Ils appellent les méthodes startWrite(), endWrite() (startRead(), endRead()) d'un même objet "obj" de la classe Synchronisation lors d'une écriture (lecture) dans la matrice de phéromone.

```
public class Synchronisation {
  private int compLect;
  private boolean lect;
  private boolean ecrit;

  // le constructeur de la classe
  public Synchronisation(){
    compLect=0;
    lect=false;
    ecrit=false;}

  public synchronized void startRead(){
   while(ecrit==true)
   {
    try
    {wait();}
    catch(InterruptedException e){ }
   }
   compLect=compLect+1;
   if(compLect ==1){lect=true;}
  }

  public synchronized  void endRead(){
   compLect = compLect -1;
   if(compLect ==0){lect=false;}
   notifyAll();
  }
```

```
public synchronized void startWrite(){
   while (lect==true||ecrit==true)
   {
    try{wait();}
    catch(InterruptedException e){}
   }
   ecrit=true;
  }

  public synchronized void endWrite(){
   ecrit=false;
   notifyAll();
  }
}//fin de la classe synchronized
```

//procédure de mise à jour (cas d'écriture)

```
public void onlineupdate()
{
 obj.startWrite();
 evaporate();
 add();
 obj.endWrite();
}
```

Bibliographie

[1] J.Aguilar.
A general Ant Colony Model to solve combinatorial
optimization problems. 99.

[2] V.Bachelet, P.Preux, E.G Talbi
Parallel hybrid meta-heuristics: application to the quadratic
assignment problem.
In parallel proceeding of the parallel optimization colloquium (Versailles, France), 1996.

[3] J.P.Banatre
La programmation parallèle.
Editions Eyrolles, Octobre 1990.

[4] B.Bulheimer, R.F.Hartl, C.Strauss.
A new rank based version of the ant system, a computational study.
Technical report POM -03/97, institute of management science, university of Vienna 1997.
Accepted for publication in the annals of operations research.

[5] B.Bulheimer, G.Kotsis, C.Strauss.
Parallelisation strategies for the ant system.
Technical report POM 9/97, university of Vienna,1997. To appear in: Kluwer series of applied optimization (selected papers of HPSNO'97), A.Murli, P.Pardalos and G.Toraldo, editors.

[6] A.Colroni, M.Dorigo, V.Manniezo.
Distributed optimization by ant colonies.
In proceedings of the first European conference on artificial life, pages 134-142 Elsevier, 1992.

[7] A.Colroni, M.Dorigo, V.Manniezzo.
An investigation of some properties of an "Ant Algorithm".
In PPSN'92 pages 509-520, Elsevier 1992.

[8] O.Cordon, I.Deviana, F.Herrera, L.Moreno.
A new ACO model integrating evolutionary computation concepts:
the best-worst ant system.
From Ant Colonies to Artificial Ants: Second International Workshop on Ant Algorithms (ANTS'2000). Brussels (Belgium), 2000, pp. 22-29.

[9] P.Damien
Methodes SAT pour un problème d'optimisation dans les graphes.
Université de Metz October 2000.

[10] P.Delisle, M.Krajecki, M.Gravel, C.Gagné
Parallel implementation of an ant colony optimization meta-heuristic with OpenMP.
International conference on parallel architectures and compilation techniques, proceeding of the 3rd European workshop on Open MP (EWOMP'2001), Barcelone, Espagne.

[11] K.Doerner, R.F.Hartl, M Reimann.
Cooperative ant colonies for optimizing resource allocation in transportation.
E-j.w.Boers et al editors- evolutionary workshop 2001, LNCS 2037, pp 70-79, Springer Verlag.

[12] M.Dorigo.
Ant Colony System: A cooperative learning approach to the Travelling Salesman Problem.
IEEE transactions on evolutionary computation, 1(1): 53-66 1997.

[13] M.Dorigo, G.Di Caro, L.M.Gambardella.
Ant algorithms for discrete optimization.
To appear in artificial life vol5, N°3 137-172, 1999.

[14] M.Dorigo, G.Dicaro.
Ant colony optimisation: A new meta_heuristic.
Proceeding of the congress on evolutionary computation volume2
Pages 1470-1477, 1999.

[15] H.Drias
Genetic Algorithm versus Scatter Search And solving Hard MAX-W-SAT problems.
In proc of IWANN' 2001, Lectures Notes in Computer Science,LNCS, Springer-Verlag, (June 2001) Grenada, Spain, to appear.

[16] H.Drias.
Approche probabiliste du dénombrement des solutions du problème de satisfiabilité.
Thése de doctorat 01/93 E/I USTHB.

[17] H.Drias, M. Khabzaoui.
Scatter Search with Random Walk Strategy for SAT and MAX-W-SAT problems.
In proc of IEA-AIE'2001, LNAI 2070, Springer , Budapest Hongrie, (June 2001) 35-44

[18] E.Ekin, T Yakhma.
A case study of adapting ant system to optimization problems.
In the 10th Turkish symposium on Artificial Intelligence and Neuronal Networks (TAINN'2001).

[19] B.Ernesto.
Le recuit simulé.
Article “pour la science” N°125 Juillet 88.

[20] C.Gagne, M.Gravel, W.Price.
A look-ahead addition to the ant colony optimization
metaheuristic and its application to an industrial sceduling problem.
Proceedings of the – 4th Metaheuristics International Conference (MIC’2001), Porto, Portugal, pp. 79-84.

[21] L.M.Gambardella, M.Dorigo.
An ant colony System hybrized with a new local search
for the sequential ordering problem.
Technical report 11-97, IDSIA, Lugano, CH, 1997.

[22] G.A.Geist, J.A.Kohl, P.M.Papadopoulos
PVM and MPI A comparison of features
Calculateurs paralleles, Volume8, pages 137-150, 1996.

[23] F.Glover
Tabu search fundamentals and uses.
Technical report, University of Colorado, 1994.

[24] A.Y.Grama and V.Kumar
A survey of parallel search algorithms for discrete optimization problems.
Department of computer science, university of Minnesota,1992

[25] Holger H.Hoos
SAT encoding, search space structure, and local search performance.
Proceedings of IJCAI-99, pp 296-302, Morgan Kaufman,1999.

[26] Holger H.Hoos and T.Stutzle.
Some surprising regularities in the behavior of stochastic local search.
Proceedings of CP-98, pp 470, 1998.

[27] Holger H.Hoos
On the runtime behavior of stochastic local search algorithms for SAT.
Proceedings of AAAI-99, pp 661-666, MIT press, 1999.

[28] S.Joy, J,T,Mitchel, B.Bochers.
Solving MAX_SAT and weighted MAX_SAT problems using Branch_and_cut.,February 28, 98.
Accepted for publication in the *Journal of Combinatorial Optimization*.

[29] B. jurkowiak et Chu Min Li.
Parallelisation d’une recherche arborescente pour les problèmes de satisfiabilité.
In *Actes de RenPar 2001*, Paris.

[30] D.Karabog, A. Kalinli and N.Karaboga.
Touring Ant Colony algorithm with frequency based memory for continuous optimization.
In the 10th Turkish symposium on Artificial Intelligence and Neuronal Networks (TAINN'2001).

[31] H.Khedimi.
Contribution to the resolution of MAX_SAT problem by taboo search strategies.
Thése de magister 05/2000 M/IN USTHB.

[32] V.Manniezo, A.Carbonaro.
Ant colony optimization: An overview.
In C.Ribeiro(eds) Essays and Surveys in Metaheuristics, Kluwer,
pages 21-44, 2001.

[33] V.Manniezo, A.Carbonaro, M. Golfarelli and S. Rilz
An ants algorithm for optimizing the materialization of fragmented views in data warehouses: preliminary results.
Lecture notes in computer science, Volume 2037, Springer Verlag Berlin, Heilderberg, pp 80-89, 2001.

[34] T.Mavridou, PM.Pardalos, I.Pistoulis, MGC.Resende.
Parallel search for combinatorial optimization: Genetic algorithms, Simulated annealing, Tabu search and Grasp.
Lecture Notes in computer science 980:317-331, 1995.

[35] M.Middendorf, F.Reishle, H.Schmeck
Information exchange in multi-colony ant algorithms.
In: Parallel and Distributed Computing, Proceedings of the 15 IPDPS 2000 Workshops, Rolim, J. (Hrsg.), Third Workshop on Biologically Inspired Solutions to Parallel Processing Problems (BioSP3), Cancun, Mexico, 1.5.2000, Springer, LNCS, 1800, 2000, S. 645-652

[36] P.M.Pardalos, L.Pistoulis and M.G.C.Resende.
A parallel grasp for Max-Sat problems.
Applied Parallel Computing, Industrial Computation and Optimization, Third International Workshop, Lyngby, Denmark, Springer, PARA 96: 575-585.

[37] M.Randall.
A general parallel tabu search algorithm for combinatorial optimization problems.
Technical Report 99-03, School of Information Technology, *Bond University*.

[38] M.Randall and A.Tonkes
Intensification and diversification strategies in ant colony search.
Technical Report 00-02, School of Information Technology, *Bond University*.

[39] A.Roli, C.Blum.
Critical parallelization of local search for Max-Sat.
In AI*IA 2001: Advances in Artificial Intelligence, LNAI 1792, pp. 147-158, ISBN 3-540-42601-9.

[40] MGC.Resende, L.S.Pistoulie, PM.Pardalos.
Approximate solution of weighted MAX_SAT problems using Grasp.
Discrete Applied Mathematics March 2000, Volume 100 Issue 1-2.

[41] T.Stutzle, H.Hoos.
Improving the ant system: A detailed report on the MAX_MIN ant system.
In proceeding of the international conference on artificial neuronal and genetic algorithms pages 245-249. Springer Verlag, wien 1997.

[42] T.Stutzle.
Parallelization strategies for ant colony optimization.
Proceedings of PPSN-V, fifth International Conference on Parallel Problem Solving from Nature, pages 722-731 Springer Verlag 1998.