*REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE*

*MINISTERE DE L'ENSEIGNEMENT SUPERIEUR*
*ET DE LA RECHERCHE SCIENTIFIQUE*

Université des Sciences et de la Technologie
Houari Boumediene (U.S.T.H.B.)

Faculté de Génie Electrique et d'Informatique

Département d'Informatique

Thèse

Présentée à l'U.S.T.H.B.
pour l'obtention du diplôme de Magister
en Informatique

**Par: M$^{elle}$ Dalila BOUGHACI**

# Méta- heuristiques hybrides pour les problèmes MAX-W-SAT difficiles

Soutenue le 02 juin 2001

Devant le jury composé de:

Mme. Boumghar F, Maître de conférences (USTHB)……..…Présidente
Mme. Drias H, Professeur (USTHB ) ……Directrice de thèse
Mr. Saad R, Maître de conférences (Boumerdès)..……..Examinateur
Mr. Azzoune H, Docteur d'état ( USTHB)…….……Examinateur
Mme. Benatchba K, Chargée de cours (INI). …….……...Invitée

N° d'ordre : 01/2001-M/IN

# Table des matières

**Chapitre 3 :** La recherche dispersée et la recherche locale guidée



**Chapitre 4 :** Proposition d'algorithmes de résolution de MAX-W-SAT basés sur la recherche dispersée et la recherche locale guidée

# Résumé


Cette thèse traite un problème réputé difficile qui est la satisfiabilité maximale pondérée (**MAX-W-SAT** en abrégé**)**. Etant donné un ensemble C de clauses sous forme disjonctive définies sur un ensemble X de variables logiques, à chaque clause Ci un poids Wi est associé. Le problème **MAX-W-SAT** consiste à trouver une assignation aux variables de X qui maximise la somme des poids des clauses satisfaites.

Le problème **MAX-W-SAT** suscite un intérêt particulier car il possède plusieurs applications notamment en raisonnement automatique et en démonstration de théorèmes. De plus, La résolution de ce problème a une très grande importance dans le domaine de l'intelligence artificielle, en particulier, dans l'élimination de l'incohérence des bases de connaissances dans les systèmes experts.

L'accent est mis dans ce travail sur les méta- heuristiques ainsi que les voies qu'ouvrent ces approches dans l'optimisation combinatoire. Il a pour but de clarifier le langage obscure qui entoure les concepts sous-jacents, et d'expliciter l'apport précieux qu'offrent actuellement ces méthodes dans l'intelligence artificielle.

Parmi ces méthodes, on trouve la recherche dispersée et la recherche locale guidée, qui bousculent les méthodes classiques et établissent de nouvelles stratégies de recherche.

La recherche dispersée appelée en anglais «**SCATTER SEARCH**» est une méta-heuristique évolutionniste basée population, elle génère de nouvelles solutions en commençant par un ensemble de solutions appelées solutions références. L'approche fonctionne avec un opérateur de combinaison permettant d'obtenir des solutions admissibles.

La recherche locale guidée est une approche basée pénalité. Elle permet, en principe, d'éviter l'optimum local. C'est une amélioration de la recherche locale, elle modifie la fonction objective dans le but d'approcher l'optimum global.

Dans notre étude, nous avons proposé deux solutions au problème **MAX-W-SAT.** La première est basée sur la recherche dispersée alors que la seconde est basée sur la recherche locale guidée. De plus, nous essaierons pour notre part d'améliorer les performances de ces derniers algorithmes en présentant quelques orientations générales pour une hybridation pertinente de quelques heuristiques connues dans la littérature informatique basées sur la recherche tabou, et la procédure GSAT.



**Mots clés :**

Méta- Heuristique, Heuristique, Satisfiabilité Maximale Pondérée, Recherche Dispersée, Recherche Locale Guidée, NP- Complet, Intensification, Diversification, Bruit aléatoire, Marche aléatoire.

# Introduction générale

Un grand nombre de problèmes d'optimisation sont NP- complets. En conséquence, une énumération exhaustive pour trouver la solution optimale est la plupart du temps impossible. On définit alors des heuristiques qui sont des méthodes qui s'approchent en un temps raisonnable d'une «bonne» solution au problème.

Parmi ces méthodes, on trouve la recherche dispersée et la recherche locale guidée qui constituent une alternative intéressante aux techniques classiques pour la résolution des problèmes d'optimisation **[GLM 00][Lag 99]**, en particulier le problème de satisfiabilité(**SAT** en abrégé).

Le problème SAT de la logique booléenne est le premier problème démontré NP- complet. C'est une satisfiabilité propositionnelle : on donne une formule de calcul propositionnel (une conjonction de clauses, où une clause est une disjonction de littéraux et un littéral est une variable ou sa négation), on décide s'il y a une assignation à ses variables qui rend la formule vraie.

D'autre part, le problème SAT concerne surtout le domaine de l'intelligence artificielle puisqu'il est en relation avec celui du raisonnement déductif. De plus, d'autres formes de raisonnement incluant le raisonnement par défaut, diagnostic, planification et l'interprétation d'images font un appel direct à la satisfiabilité.

Dans notre étude, on s'intéresse à une autre variante du problème SAT, appelée satisfiabilité maximale pondérée «**MAX-W-SAT**» qui associe à chaque clause Ci un poids Wi. Le but consiste à trouver une assignation aux variables qui maximise la somme des poids des clauses satisfaites.

Vu le rôle important que présente le problème SAT, plusieurs méthodes ont été développées pour le résoudre : les méthodes exactes et les méthodes approchées.

La figure1 met en parallèle les méthodes représentatives développées en recherche opérationnelle et en intelligence artificielle, avec à titre indicatif la date approximative d'apparition de chaque méthode.

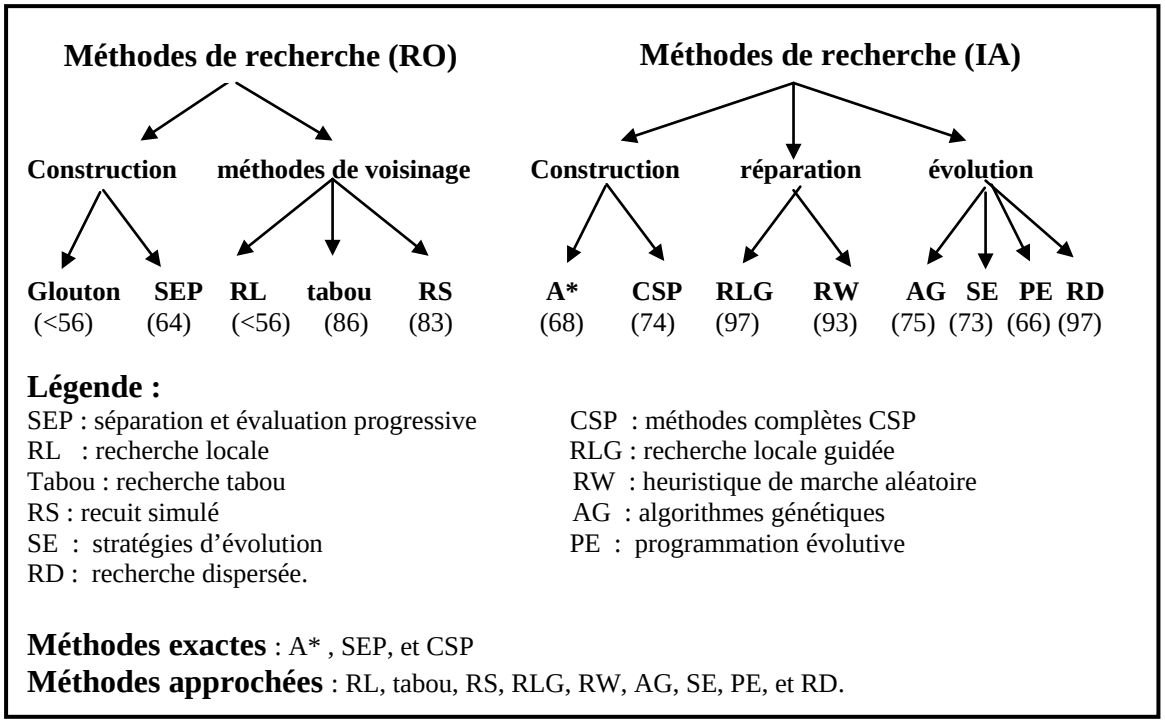


**Figure1. Classement des méthodes de résolution.**

D'une manière générale, les méthodes de résolution suivent quatre approches différentes pour la recherche d'une solution : l'approche de construction, l'approche de relaxation telle que la relaxation de base en programmation linéaire, l'approche de voisinage et l'approche d'évolution. Ces méthodes font partie de deux groupes de nature différente : le premier groupe comprend les méthodes exactes, et le second groupe comprend les méthodes approchées.

Le principe essentiel d'une méthode exacte consiste généralement à énumérer, souvent de manière implicite, l'ensemble des solutions de l'espace de recherche. Parmi ces méthodes, nous citons les méthodes de séparation et évaluation et la procédure de Davis et Putnam**[BF 97]**. Les méthodes exactes ont permis de trouver des solutions optimales pour des problèmes de taille raisonnable. Cependant, ces méthodes rencontrent généralement des difficultés face aux applications de taille importante.

Les méthodes approchées constituent une alternative très intéressante pour traiter les problèmes d'optimisation de grande taille si l'optimalité n'est pas primordiale. On peut citer les méthodes gloutonnes et l'amélioration itérative : par exemple, la procédure GRASP**[RF 96]**, la procédure GSAT**[SHKC 94]**, et la recherche locale.

Ces dernières années, des progrès importants ont été réalisés avec l'apparition d'une nouvelle génération de méthodes approchées puissantes et générales, souvent appelées méta- heuristiques.

Les méta- heuristiques sont représentées essentiellement par les *méthodes de voisinage* comme le recuit simulé**[Sia 95]**, et la recherche tabou**[Glo 95],** et les *algorithmes évolutifs* comme les algorithmes génétiques**[Hol 92]** et la recherche dispersée**[GLM 00]**. Grâce à ces méta- heuristiques, on peut proposer aujourd'hui des solutions approchées pour des problèmes d'optimisation de plus grande taille.

Notre travail s'intéresse aux méta- heuristiques, en particulier, la recherche dispersée et la recherche locale guidée.

La recherche dispersée appelée en anglais «**SCATTER SEARCH**» est une méta- heuristique basée population, elle génère de nouvelles solutions en commençant par un ensemble de solutions appelées solutions références. L'approche fonctionne avec un opérateur de combinaison permettant d'obtenir des solutions admissibles.

La recherche locale guidée est une approche basée pénalité. Elle permet, en principe, d'échapper à l'optimum local**[Vou 97]**. C'est une amélioration de la recherche locale, elle modifie la fonction objective dans le but d'approcher l'optimum global.

En d'autres termes, la recherche dispersée et la recherche locale guidée procèdent par améliorations successives. La première approche démarre d'une population initiale variée, en explorant des zones ***diverses*** de l'espace de recherche, et tente de l'***améliorer*** tout au long du processus de résolution, alors que la recherche locale guidée est initialisée avec une configuration initiale réalisable qu'elle cherche à améliorer à chaque itération à l'aide d'un algorithme itératif de recherche locale. Pour échapper aux optima – locaux la recherche locale guidée le fait en utilisant une fonction objective augmentée permettant de pénaliser les solutions déjà visitées, la recherche dispersée utilise la diversification et l'amélioration.

L'objet de notre étude consiste à appliquer la recherche dispersée et la recherche locale guidée au problème **MAX-W-SAT**. De plus, nous essaierons pour notre part d'améliorer les performances de ces derniers algorithmes en présentant quelques orientations générales pour une hybridation pertinente de quelques heuristiques connues dans la littérature informatique basées sur la recherche tabou, GSAT et d'autres heuristiques qui vont être exposées dans la suite, et cela dans le cadre de la résolution du problème **MAX-W-SAT**.

***Notre document sera organisé comme suit :***

Le premier chapitre est consacré aux généralités et aux principaux concepts du problème SAT.

Dans le deuxième chapitre, nous présentons trois méta- heuristiques classiques qui calculent des solutions acceptables : la recherche tabou, le recuit simulé et les algorithmes génétiques.

Nous décrivons au troisième chapitre deux méthodes récentes apparues ces dernières années dans le but de résoudre efficacement les problèmes d'optimisation difficiles : la recherche dispersée et la recherche locale guidée.

Le quatrième chapitre fait un tour d'horizon sur les solutions apportées au problème **MAX-W-SAT** considéré et de manière plus approfondie, deux méta- heuristiques ont été conçues et mises en œuvre à savoir la recherche dispersée et la recherche locale guidée.

Au cinquième chapitre, nous exposons les différents tests numériques réalisés sur plusieurs types de problèmes.

Nous présentons dans le sixième chapitre d'autres algorithmes itératifs : la recherche tabou avec bruit aléatoire, la procédure GRASP, la procédure GSAT avec marche aléatoire, la recherche dispersée hybride, et la recherche locale guidée hybride.

Une étude comparative entre les méta- heuristiques citées ci –dessus est établie au septième chapitre dans le but de tester la performance de la recherche dispersée et de la recherche locale guidée et de prouver leur efficacité, ainsi que leur capacité à résoudre le problème **MAX-W-SAT**.

Enfin, la conclusion générale porte sur les résultats obtenus et dresse quelques perspectives sur la suite de notre travail.

Une annexe est ajoutée à la fin pour élucider les notions non approfondies dans les chapitres.

# Chapitre 1 :

# Le problème de satisfiabilité SAT

## 1.1. INTRODUCTION :

Ces dernières années, les méta- heuristiques ont connu un succès grandissant dans les domaines de la recherche opérationnelle et de l'intelligence artificielle. Elles ont prouvé leur efficacité et leur performance dans de nombreux problèmes d'optimisation combinatoire à savoir le problème du voyageur de commerce, les problèmes d'affectation **[Vou 97][VT 98]**, les problèmes d'ordonnancement ainsi que le problème **SAT [Dri 93]**.

Le problème de satisfiabilité (**SAT**) est l'ancêtre des problèmes NP- complets. Un problème est dit NP complet si son temps d'exécution s'exprime théoriquement comme une fonction exponentielle de la taille de donnée, ce qui implique l'inexistence des algorithmes polynômiaux pour le résoudre en un temps polynômial.

Beaucoup de problème NP- complets appartiennent à divers domaines, tels que la théorie des graphes (le problème du voyageur de commerce, le problème du circuit hamiltonien et le problème de coloration d'un graphe…), la géométrie, les jeux, la théorie des langages et l'ordonnancement des tâches. Ces problèmes peuvent être formulés comme un problème de satisfiabilité ou une de ses variantes.

D'autre part, le problème **SAT** a un impact sur le domaine de l'intelligence artificielle notamment dans l'élimination de l'incohérence des bases de connaissances dans les systèmes experts, où les implications de la base de connaissance peuvent être transformées sous forme de clauses en utilisant des formules bien connues de l'algèbre de Boole.

Dans ce qui suit, nous allons définir d'une manière formelle le problème de satisfiabilité SAT. Nous citons certaines de ses variantes et nous présentons quelques méthodes de résolution.

## 1.2. NOTATION ET FORMULATION DU PROBLEME :

Soient :

- X un ensemble de n variables booléennes ;

- C un ensemble de m clauses dont chacune est la disjonction d'un ensemble de littéraux associés aux variables de X, tel que chaque littéral est une variable xi ou sa négation $\neg$i;

On définit formellement le problème SAT comme une formule logique écrite sous forme normale conjonctive(conjonction de clauses) et on cherche une assignation aux variables de X qui rend cette formule vraie.

$$\mathbf{F} = \bigwedge_{1 \le i \le m} \mathbf{Ci} \quad \textbf{avec :} \quad \mathbf{Ci} = \bigvee_{k \in \{1,\ldots n\}} \mathbf{L_{ik}}, \quad \mathbf{L_{ik}} \in \{x_j, \neg j\},\ j=1,..n.$$

***Une instance de SAT est présentée comme suit :***

$$\begin{cases} C_1 = L_{11} \vee L_{12} \vee \ldots\ldots\ldots\ldots\ldots \vee L_{1r1} \\ C_2 = L_{21} \vee L_{22} \vee \ldots\ldots\ldots\ldots\ldots \vee L_{2r2} \\ \\ C_m = L_{m1} \vee L_{m2} \vee \ldots\ldots\ldots\ldots\ldots \vee L_{m\,r\,m} \end{cases}$$

Où :

n est le nombre de variables booléennes xi ;
m est le nombre de clauses ;
r est la longueur d'une clause, c'est –à- dire le nombre de littéraux figurant dans cette clause ;
$L_{ik}$ est un littéral ;
$r_1, r_2, \ldots\ldots\ldots r_m$ sont respectivement les longueurs des clauses
$C_1, C_2 \ldots\ldots\ldots C_m$ .

Une instance est satisfaite quand toutes ces clauses sont vraies, sinon elle sera contradictoire.

Le but est de trouver une assignation aux variables, c'est-à–dire pour chaque variable xi affecter la valeur vraie =1 ou faux=0, qui satisfasse la conjonction de clauses.

Quand il n'existe pas d'assignation qui rend la formule **F** vraie, on s'intéresse beaucoup à d'autres variantes du problème SAT.

## 1.3. QUELQUES VARIANTES DU PROBLEME SAT [Dri 93] [BP 97] [BP 96][MSG 97]:

Le problème SAT peut se trouver sous plusieurs formes. Parmi ces variantes nous citons :

- ***Le problème SAT* de *décision :***

  Il permet de répondre par «oui» si la donnée est satisfaite, par «non» si celle- ci est contradictoire.

- ***Le problème SAT de recherche des solutions d'une donnée SAT :***

  Il consiste en l'exhibition de certaines solutions de la donnée SAT. La résolution de ce problème permet de répondre au problème de décision.

- ***Le problème de dénombrement des solutions d'une donnée SAT:***

  Il s'agit de déterminer le nombre de solutions de l'instance SAT. La résolution de ce problème permet aussi de répondre au problème de décision associé à SAT.

A ces variantes on peut rajouter les problèmes MAX-SAT, MAX-R-SAT et MAX-W-SAT qui seront présentés comme suit :

- ✓ **Le problème MAX-SAT :**

Dans le cas où il n'existe pas une assignation qui satisfait toutes les clauses (donnée contradictoire), le problème SAT peut être exprimé en un autre problème appelé **MAX-SAT**. Dans un tel problème on cherche une assignation qui maximise la somme des clauses satisfaites.

- ✓ **Le problème MAX-R-SAT :**

Dans ce problème toutes les clauses ont une taille inférieure ou égale à R. L'objectif revient aussi à maximiser la somme des clauses satisfaites.

✓ **Le problème MAX-W-SAT :**

Le problème auquel on s'intéresse ici est la satisfiabilité maximale pondérée « **MAX-W-SAT**» qui associe à chaque clause Ci un poids Wi. Le but consiste à trouver une assignation aux variables qui maximise la somme des poids des clauses satisfaites.

***On définit formellement le problème MAX-W-SAT comme suit :***

Soient :

- $X = \{x_1, x_2, \ldots, x_n\}$ un ensemble de n variables booléennes ;
- $C = \{c_1, c_2, \ldots, c_m\}$ un ensemble de m clauses formées de littéraux construits à partir de l'ensemble X ;
- A chaque clause ci un poids positif Wi est associé.
- $\mathbf{ci} = \bigvee_{k \in \{1,\ldots n\}} \mathbf{Li_k}, \quad \mathbf{Li_k} \in \{x_j, \neg x_j\}, j=1,\ldots.n.$
- $\mathbf{F} = \sum_{i=1}^{m} W_i , \ i \in \{ \text{clauses satisfaites}\}.$

⟹ On cherche une assignation aux variables de X qui maximise **F.**

**Le problème MAX-W-SAT consiste à trouver une instanciation qui maximise la somme des poids des clauses satisfaites.**

## 1.4. METHODES DE RESOLUTION :

Les problèmes SAT et MAX-SAT sont NP- complets. Plusieurs méthodes ont été développées pour les résoudre: les méthodes constructives, les méthodes exactes et les méthodes approchées.

### 1.4.1. Méthodes constructives :

Les méthodes constructives sont basées sur des heuristiques et permettent de construire une solution approchée. Nous citons : les algorithmes de JOHNSON. En 1974, JOHNSON a proposé deux heuristiques de type Glouton john1 et john2. **[Joh 74] [JG 79]**.

- ***La procédure heuristique John1*[Joh 74] *:***

Cette procédure permet de construire une solution approchée au problème SAT d'une façon progressive. Elle considère à chaque itération le littéral qui apparaît dans le plus de clauses. Ce littéral étant à vrai toutes les clauses où il apparaît sont vraies ; ainsi la solution sera construite à partir des valeurs booléennes pour lesquelles ces littéraux sont vrais.

**Algorithme de johnson (john1) :**

```
// C  : ensemble de clauses.
// U  : ensemble des variables booléennes.
// Ens-s : ensemble de clauses satisfaites.
// G : ensemble de clauses – Ens-s.
// Pour chaque littéral l, u(l) est la variable correspondante.
// R est l'assignation aux variables.
Début
  Ens-s = ϕ ;
  G = C ;
  U= { u  \ u variable dans C } ;
    répéter
    Choisir un littéral l ;     // u(l) ∈ U et l apparaît dans le maximum
                                   de clauses de G.
    // Soient {Cl1,……,Clk } les clauses dans lesquels l apparaît.
    Ens-s  = Ens-s ∪ {Cl1,……,Clk };
    G   = G -  {Cl1,……,Clk };
    Si  l  vrai    alors    u(l) = vrai Sinon   u(l) = faux ;
     U   = U  - { u(l) };

    Jusqu'à ce qu'aucun littéral l,
              avec u(l)∈U ne soit contenu dans une clause de G.
  Si  U ≠ϕ   alors     pour tout u ∈U faire u = vrai
  Retourner le résultat  R

Fin
```

- ***La procédure heuristique John2*** **[JG 79]** ***:***

Cette procédure est une amélioration de la première, elle prend en compte la forme positive ou négative du littéral. Elle considère à chaque itération le littéral qui apparaît dans le plus de clauses. Ce dernier sera vrai si la somme des clauses où il apparaît positif est supérieure à la somme des clauses où il apparaît négatif sinon il sera mis à faux.

**Algorithme de johnson (john2) :**

// C : ensemble de clauses.
// U : ensemble des variables booléennes.
// Ens-s : ensemble de clauses satisfaites.
// G : ensemble de clauses – Ens-s.
// Pour chaque littéral l, u(l) est la variable correspondante.
// R est l'assignation aux variables.

**Début**

Ens-s = $\phi$ ;
G = C ;
U= { u \ u variable dans C } ;
Assigner pour chaque clause $C_i$ une masse $W(c_i)= 2^{-|C_i|}$

**répéter**
Choisir un littéral l ; // u(l) $\in$ U et l apparaît dans le maximum de clauses de G.
// Soit CT les clauses de G dans lesquels l apparaît positif .
// Soit CF les clauses de G dans lesquels l apparaît négatif.

**Si** $\sum_{c_i \in ct} W(c_i) \geq \sum_{c_i \in cf} W(c_i)$ **alors**
u(l) ← vrai
Ens-s = Ens-s $\cup$ CT;
G = G - CT;
Pour toute $C_i \in$CT faire
$W(c_i) \leftarrow 2^* W(c_i)$

**Sinon**
u(l) ← faux
Ens-s = Ens-s $\cup$ CF;
G = G - CF;
Pour toute $C_i \in$CF faire
$W(c_i) \leftarrow 2^* W(c_i)$

**Jusqu'à** ce qu'aucun littéral l, avec u(l)$\in$U ne soit contenu dans une clause de G.
**Si** U $\neq\phi$ **alors** pour tout u $\in$U faire u = vrai
**Retourner** le résultat R

Fin

- ***La procédure GRASP* [RF 96] [BP 98] *:***

C'est une méthode itérative où chaque itération consiste en deux phase, une phase de construction et une phase de recherche locale. Dans la première phase, on construit itérativement une solution qui sera améliorée par la recherche locale dans la deuxième phase.

Plus précisément, dans la phase de construction on utilise une liste candidate contenant toutes les transitions possibles ordonnées selon une certaine fonction Glouton. La solution courante est construite aléatoirement à partir de la liste candidate dans le but d'obtenir une configuration différente à chaque itération.

La fonction Glouton prend en compte l'assignation qui vient d'être faite. Elle est conçue pour maximiser la somme des poids des clauses déjà non satisfaites qui sont devenues satisfaites après une assignation donnée.

***Les deux phases de la procédure GRASP sont présentées comme suit :***

**Phase de construction :**

*Une solution est construite en exécutant la procédure suivante :*

*procédure* construction ()

// n : le nombre des variables booléennes

**Pour** k=1 → n **Faire**
- Créer la liste restreinte des assignations candidates;
- Sélectionner aléatoirement une transition ;
- Affecter une valeur de vérité à la variable choisie ;
- Appliquer la fonction Glouton

**Fait**

**Phase d'amélioration :**

Afin d'améliorer la solution construite par la première procédure, **GRASP** fait appel à une recherche locale. Cette dernière effectue des transformations successives sur la configuration courante dans le but de trouver une solution adjacente. Si le coût de la configuration voisine est meilleur que celui de la solution courante, cette dernière sera remplacée par cette solution voisine, sinon elle sera gardée. Ce processus est réitéré jusqu'à ce qu'aucune amélioration dans la configuration courante ne pourrait être faite

***L'algorithme de la recherche locale :***

// X : la solution construite dans la première phase ;
// Aml = faux, si aucune amélioration ne pourrait être faite dans la configuration courante ;

Aml := vrai ;

**Répéter**
    X' = Meilleur voisin ;
    **Si** coût(X') > coût(X) **alors** X= X' **Sinon** Aml = faux ;

**Jusqu'à** Aml =faux
**Retourner** la meilleure solution X ;

**Par conséquent, on aura l'algorithme GRASP suivant :**

**Etape 1 : Initialisation**

- nbiter = 0 ; /* itération courante */
// nbmax : nombre maximum d'itérations

**Etape 2 : Processus itératif**
***Tant que*** (nbiter < nbmax)
***Faire***
    - nbiter = nbiter + 1;
    - Construction() ;
    - Recherche locale() ;
***Fait***
***Fin***

### 1.4.2. Méthodes exactes :

Elles permettent de construire une solution exacte en utilisant soit la programmation linéaire ou bien la programmation dynamique ou les méthodes arborescentes. Pour le problème **SAT**, nous citons la procédure de Davis et Putnam.

- **La procédure de Davis et Putnam[BF 97] :**

Le principe de la procédure de **Davis** et **Putnam** (**DP** en abrégé) revient à réduire l'instance de **SAT** jusqu'à l'obtention d'une sous instance, à partir de laquelle il devient possible de répondre à la satisfiabilité.

L'algorithme de la procédure de **Davis** et **Putnam** est basé sur les deux règles suivantes :

**- La règle de la clause unitaire :**

Une **clause unitaire** est une clause qui contient un seul littéral. Par convention, une **clause vide** est une clause qui ne contient aucun littéral, et elle est forcément fausse.

*La règle de la clause unitaire est appliquée comme suit :*

Considérer les clauses unitaires et fixer leur littéral à vrai, puis supprimer toutes les clauses qui contiennent ce littéral **lit**, ainsi que toutes les occurrences négatives de **lit** dans les clauses restantes. Cette règle est appliquée jusqu'à ce que la donnée soit vide ( indiquant que la donnée **SAT** est satisfiable), ou qu'elle contient une clause vide ( indiquant que la donnée **SAT** est non satisfiable).

**- La règle d'éclatement :**

Dans le cas où la règle de la clause unitaire n'est pas applicable, l'algorithme de la procédure de **Davis** et **Putnam** s'appuie sur la règle d'éclatement. Cette dernière consiste à choisir une variable **u**, qui n'a pas de valeur de vérité et lui affecter une valeur. Si l'appel de la procédure de Davis et Putnam sur la formule simplifiée retourne non satisfiable, la variable **u** est réinitialisée à la valeur opposée et la procédure **DP** est appliquée de nouveau.

*Après avoir présenté les deux règles utilisées par la méthode de Davis et Putnam , nous donnons dans ce qui suit la procédure **DP** :*

**Procédure DP :**

- Soit un ensemble de clauses S défini sur un ensemble de variables U ;
- Si S est vide alors retourner « satisfiable »
- Si S contient une clause vide alors retourner « non satisfiable »

***Règle de clause unité :***

- Si S contient une clause unité C alors affecter à la variable mentionnée la valeur de vérité qui satisfait C, et retourner le résultat de l'appel de **DP** sur la formule simplifiée.

***Règle d'éclatement « Splitting » :***

- Sélectionner une variable u de U qui n'a pas de valeur de vérité ;
- Affecter une valeur de vérité à cette variable, et appliquer **DP** sur la formule simplifiée ;
- Si cet appel retourne «satisfiable» retourner «satisfiable» ;
- Sinon initialiser u à la valeur opposée à celle assignée auparavant, et retourner le résultat de l'appel de **DP** sur la formule simplifiée de nouveau.

### 1.4.3. Méthodes itératives :

Les algorithmes itératifs démarrent d'une configuration initiale et essayent de l'améliorer en effectuant des transformations successives sur la configuration courante. Ce processus est réitéré jusqu'à un certain nombre d'itérations. Parmi ces méthodes nous citons: la procédure **GSAT**, la recherche locale, la technique de recuit simulé, les algorithmes génétiques, et la recherche tabou.

- **La procédure GSAT [SML 92] [SHKC 94]:**

La procédure de base GSAT commence par une solution aléatoire, puis procède de façon répétitive à changer l'affectation de la variable qui mène à la plus grande baisse dans le nombre total des clauses non satisfaites.

*Cette procédure se présente de la manière suivante :*

```
Soit un ensemble de clauses C
// MAX-ITER : nombre d'itérations ;
// MAX-ESS : nombre d'essais ;
// n : nombre de variables ;
Pour i :=1 à MAX-ESS :
 faire
     Soit X une assignation aléatoire
     Pour i := 1 à Max-ITER
     faire
          Si X satisfait C, alors retourner X et arrêter
          Soit xi une variable dont le changement de
          l'assignation donne la plus grande baisse dans le
          nombre total de clauses de C qui ne sont pas
          satisfaites par X
          X= X avec la valeur de xi inversée.
     fait
  fait
```

Les paramètres MAX-ITER et MAX-ESS déterminent respectivement combien de transitions la procédure entreprendra avant d'abandonner et recommencer, et combien de temps cette recherche peut être recommencée avant de quitter.

La procédure GSAT permet de trouver une instanciation pour laquelle on a le maximum de clauses satisfaites. Néanmoins, elle pourrait échouer à trouver une assignation alors qu'elle existe. Pour cet effet, deux extensions de la procédure GSAT ont été introduites : GSAT avec bruit aléatoire et GSAT avec marche aléatoire.

- ***GSAT avec marche aléatoire :***

Une stratégie de la **«marche aléatoire»** a été ajoutée à l'algorithme de la procédure **GSAT** pour modifier la règle de choix du voisin, et ça, dans l'algorithme **GSAT avec marche aléatoire.**

***Cet algorithme est donné comme suit [BKC 93]*** *:*

```
Pour i : = 1 à MAX-ESS
 faire
    X := assignation aléatoire
   Pour j : = 1 à MAX-ITER
    faire
          Si Randomnbr() < p alors
              Var := variable quelconque ∈ à une clause non satisfaite
             Sinon var := la variable dont Δf est le plus grand
           FLIP(var)
       Fait
Fait
```

Où :
Randomnbr() : génère un nombre aléatoire entre 0 et 1.
**FLIP**(var) : inverse la variable var (var : = 1- var).
**Δf** : coût d'une configuration.
**MAX-ESS** : nombre d'essais.
**MAX-ITER** : nombre d'itérations.

***Principe de résolution :***

Vu que l'approche contient certaines étapes avec un choix aléatoire, un certain nombre d'essais est exécuté(MAX-ESS), où Chaque essai consiste en un nombre d'itérations qui égale à MAX-ITER.

A chaque itération une variable est choisie par deux critères différents :

- Avec une probabilité p, choisir une variable d'une clause non satisfaite et inverser sa valeur.
- Avec une probabilité 1-p, appliquer l'algorithme GSAT standard.

- ***GSAT avec bruit aléatoire :***

Le principe de cette approche, consiste à faire la sélection aléatoire de la variable à inverser à partir de l'ensemble de toutes les variables et non uniquement celles qui apparaissent dans les clauses non satisfaites. Par conséquent, on aura la procédure suivante :

Procédure **GSAT** avec bruit aléatoire

**Pour** i←1 **à** MAX-ESS **faire**

**Pour** I :=1 to MAX-ITER **faire**

- Avec une probabilité p, choisir aléatoirement une variable ;
- Avec une probabilité 1-p, suivre le schéma de **GSAT** standard ;

**Fait**

**Fait**

Où :
**MAX-ESS** : nombre d'essais.
**MAX-ITER** : nombre d'itérations.

- **La recherche locale :**

La recherche locale démarre d'une solution initiale possible et essaie de l'améliorer, en cherchant une solution meilleure dans le voisinage courant. Un voisinage d'une solution X correspond à des éléments adjacents à X dont chacun est atteint par un changement dans la configuration courante.

Le processus de recherche est réitéré jusqu'à ce qu'aucune amélioration dans la solution courante ne pourrait être faite.

*L'algorithme général de la recherche locale est le suivant:*

Soit une solution initiale x ;

Amelior =vrai,

**Répéter**

x' = Meilleur Voisin(x) ;

**Si** coût(x) < coût(x') **alors** x = x', **Sinon** Amelior =faux **fsi**

***Jusqu'à*** A*melior= faux*.

**Fin**

- **La recherche tabou [Her 95] [Glo 95] :**

La recherche tabou démarre d'une solution initiale aléatoire, puis le meilleur voisin, dans le voisinage courant, est choisi pour être le point courant suivant même s'il est de qualité inférieure au point précédant. Cependant, cette stratégie à tendance à revenir sur des solutions déjà visitées ; c'est pourquoi la recherche tabou utilise une mémoire à court terme; la liste tabou. Cette liste conserve une information sur les dernières itérations de l'algorithme qui permet de se prémunir contre les cycles courts.

- **Les algorithmes génétiques [Gol 89] [Hol 92]:**

C'est un algorithme inspiré de l'évolution naturelle des êtres vivants, il ne considère pas un seul point courant, comme la recherche tabou, mais il fait évoluer toute une population de points. L'algorithme génétique fonctionne avec deux opérateurs : la recombinaison ou croisement et la mutation.

A partir d'une population de points dénommés les parents, cet algorithme construit une nouvelle population (les enfants) en combinant plusieurs parents et en appliquant des modifications aléatoires (mutation).

La phase de sélection produit la population pour l'itération suivante en choisissant les meilleurs points parmi les parents et les enfants.

Habituellement, l'algorithme génétique converge, c'est à dire , la population tend à perdre sa diversité, il perd de son efficacité ; c'est pourquoi la convergence est souvent utilisée comme critère d'arrêt. Toutefois il est fréquent de stopper la recherche après un nombre donné d'itérations ( encore dénommées générations).

- **Le recuit simulé [Sia 95] :**

Le recuit simulé est une méthode heuristique inspirée de la thermodynamique ou plus exactement de la physique statique.

Elle est fondée sur une analogie avec le recuit physique des solides, qui consiste à chauffer un matériau à haute température et à le faire refroidir très lentement afin de laisser le système atteindre son énergie minimale.

L'algorithme du recuit simulé génère aléatoirement une nouvelle solution, à partir d'une solution initiale. La nouvelle solution est acceptée ou rejetée, selon un choix effectué par la génération d'un nombre aléatoire, contrôlé par un paramètre T qui est analogue à la température dans le processus physique du recuit.

## 1.5. CONCLUSION :

Dans ce chapitre, nous avons étudié le problème de la satisfiabilité, certaines de ses variantes, ainsi que quelques méthodes existantes pour sa résolution.
Dans ce qui suit, nous nous intéresserons au problème de la satisfiabilité maximale pondérée MAX-W-SAT. Nous allons contribuer à sa résolution, en utilisant des méthodes basées sur les méta- heuristiques.

# Chapitre 2 :

# La recherche tabou, les algorithmes génétiques et le recuit simulé

## 2.1. INTRODUCTION :

La signification exacte d'un problème NP- complet est que jusqu'à présent il n'existe pas d'algorithmes déterministes pour le résoudre en un temps polynomial. Pour contourner cet obstacle, les chercheurs se sont intéressés à une classe d'algorithmes très intéressante permettant de trouver des solutions acceptables aux problèmes NP- complets en un temps raisonnable, ce sont les Méta-heuristiques.

Ces dernières se caractérisent par leur conception et leur philosophie, elles reposent sur des outils mathématiques comme les fonctions à optimiser et de contrôle, qui visent à éviter les solutions de mauvaise qualité.

Ces algorithmes constituent actuellement les algorithmes itératifs produisant les meilleurs résultats. La recherche tabou, les algorithmes génétiques et le recuit simulé font partie intégrante de ces algorithmes.

## 2.2. LA RECHERCHE TABOU :

Introduite par **Fred Glover** en 1977 **[Glo 89]** et développée plus tard**[Glo 90]**, la recherche tabou est considérée comme étant une puissante procédure d'optimisation, capable d'organiser et de diriger les opérations d'une méthode subordonnée. Indépendamment, **Hansen** et **Jaumard** ont développé une méthode très similaire pour résoudre un problème particulier d'optimisation combinatoire qui est le problème de la satisfiabilité maximale**[HJ 89]**.

### 2.2.1. DOMAINES D'APPLICATION DE LA RECHERCHE TABOU :

La méthode de la recherche tabou a remporté un immense succès dans la résolution de nombreux problèmes d'optimisation combinatoire. En effet, cette méthode à été adaptée à des problèmes d'ordonnancement, de productique, de distributique, et d'optimisation dans les graphes.

La table ci-dessous met en relief les différents domaines d'application de la recherche tabou :

| |
|---|
| **SCHEDULING :** Scheduling, job-shop, flow-shop. |
| **CONCEPTION :** Conception assistée par ordinateur,<br>Conception de réseaux de transport. |
| **ALLOCATION :** Affectation quadratique. |
| **INTELLIGENCE ARTIFICIELLE ET LOGIQUE : Satisfiabilité maximale**,<br>Réseau de neurones. |
| **TECHNOLOGIE :** distribution de l'énergie électrique. |
| **TELECOMMUNICATIONS :** Conception de réseaux pour les services. |
| **PRODUCTION :** manufacture flexible. |
| **OPTIMISATION DE GRAPHE :** Partitionnement de graphes,<br>coloration de graphes |
| **DOMAINES D'APPLICATION DE LA RECHERCHE TABOU** |

### 2.2.2. DEFINITION :

La définition littérale du mot tabou est l'interdiction, pour un groupe de gens particulier, de certaines paroles ou actions pour des raisons sociales ou religieuses. Cependant, la méthode de recherche tabou est définie autrement. C'est une procédure suboptimale caractérisée par sa simplicité et sa façon de construire une mémoire dans laquelle se trouvent les solutions réalisables recherchées.

L'information fournie par cette mémoire assure que :

- La procédure de recherche continue ses investigations et ce dans le but de trouver de nouvelles solutions.

- La procédure de recherche ne sera pas piégée dans des optima- locaux et ce en les évitant.

C'est une procédure qui opère par améliorations successives, elle est conçue pour converger vers l'optimum global du problème considéré. Elle commence par une solution initiale et tente de l'améliorer au fur et a mesure que la recherche progresse.

En outre, la recherche tabou est caractérisée par sa grande capacité d'échapper aux optima- locaux en utilisant une mémoire à court terme appelée : « liste tabou ». elle permet également le retour à des solutions déjà explorées qui à travers une autre direction peuvent mener à de meilleures solutions. Cette façon de reéxploiter ces solutions se fait grâce à une valeur dite : « valeur d'aspiration ».

### 2.2.3. NOTATION ET FORMULATION DU PROBLEME [Glo 95][LG 93] :

La recherche tabou peut être présentée comme un processus itératif, dans lequel plusieurs acteurs ont contribué en développant des techniques qui permettent de résoudre plusieurs catégories de problèmes d'optimisation combinatoire.

On définit le problème à résoudre comme suit :

X est un ensemble fini de solutions réalisables et f une fonction objective définie dans X à valeurs réelles.

$$\begin{aligned} f : \quad & X \rightarrow R \\ & s \rightarrow f(s) \end{aligned}$$

Le but est de trouver une solution $s^*$ tel que $f(s^*) = \mathbf{MIN}_{s \in X} f(s)$

Ce problème peut être représenté sous cette forme :

$$(P) \begin{cases} \mathbf{MIN}_{s \in X} f(s) \end{cases}$$

Pour chaque solution s de X , on introduit V(s) comme étant l'ensemble des solutions voisines de s. Le processus tabou démarre d'une solution initiale réalisable quelconque s.

On définit une procédure itérative pour atteindre la solution optimale de (P) comme un processus qui passe d'une solution s à une solution s' dans X, en le répétant autant de fois que nécessaire.

Dans le but de garder la simplicité de la procédure, le passage d'une solution à une autre solution, doit se faire par le biais d'une transformation notée : TR.
Soient M l'ensemble de transformations et $Ms \subset M$ l'ensemble des transformations qui donnent une solution s' = TR(s) dans X, donc on définit le voisinage de la solution s par : $V(s) = \{ s' / \exists TR \in Ms : s' = TR(s) \}$.

### 2.2.4. DESCRIPTION DE LA RECHERCHE TABOU :

L'idée principale de la recherche tabou est d'accepter une solution même si son coût est supérieur à celui de la solution courante. En d'autres termes, si le voisinage V(s)de la solution s ne contient pas de meilleures solutions que s, présenté dans V(s).

En effet, la technique tabou détermine à chaque itération une nouvelle solution parmi l'ensemble des solutions voisines de la solution courante et tel que son coût est le plus faible, ceci permet de choisir le moins mauvais des voisins, c'est à dire celui qui donne un accroissement aussi faible que possible de la fonction objective **[Her 95]**.

L'inconvénient que présente une méthode basée sur cet unique principe est que : si un minimum local est atteint, un mouvement d'une solution s vers une solution s' tel que f(s) > f(s') peut provoquer le mouvement inverse à l'itération suivante, puisque s est voisin de s' et f(s') < f(s). On risque ainsi de cycler autour de ce minimum local. C'est pour cette raison que la méthode tabou s'appuie sur un deuxième principe qui consiste à garder en mémoire les dernières solutions visitées et à interdire le retour vers celles-ci pour un nombre fixé d'itérations, le but étant de donner assez de temps à l'algorithme pour lui permettre de sortir de toute «vallée» contenant un minimum local **[ Her 95]**.

Ainsi, la méthode tabou conserve à chaque étape une liste de solutions tabous dite : «liste tabou ». L'espace nécessaire pour sauvegarder un ensemble de solutions tabous peut s'avérer gourmand en place mémoire. Pour cette raison, il est préférable d'interdire uniquement un ensemble de mouvements qui ramènerait à une solution déjà visitée. Ces mouvements interdits sont appelés :**«mouvements tabous»**.

### 2.2.5. PARAMETRES DE CONTROLE DE LA RECHERCHE TABOU [Her 95]:

- ***Le voisinage :***

La recherche tabou nécessite que pour toute solution $s \in X$, il existe un sous-ensemble appelé V(s) qui consiste en les solutions réalisables qui peuvent être atteintes par s en un seul mouvement.

Pour certains problèmes, le voisinage V(s) de la solution courante s a une grande taille et le seul moyen de déterminer la solution s' minimisant f sur V(s) est de passer en revue l'ensemble V(s) tout entier, on préfère alors générer un sous-ensemble $V^* \subset V(s)$ ne contenant qu'un échantillon de solutions voisines à s et on choisit la solution $s' \in V^*$ de valeur f(s') minimale.

- ***La liste tabou :***

Dans la recherche tabou la meilleure solution s' dans V(s) est acceptée (même si f(s') > f(s), en tenant compte de certaines conditions ), par conséquent, on peut facilement retomber sur une solution s déjà explorée, pour éviter de cycler autour de cette dernière, on interdit les modifications qui conduisent vers des solutions déjà visitées, pour cela, on introduit une liste LT appelée : liste tabou dans laquelle toute solution non permise ou interdite est recensée **[Glo 95]**.

En d'autres termes, cette liste de longueur K(fixe ou variable) est utilisée pour conserver les derniers mouvements réalisés sous forme de couple (s, s') c'est à dire quand un mouvement est fait de s vers s', on introduit le mouvement qui a donné s' a la fin de la liste LT et on rejette le plus ancien, ainsi le retour à s' n'est pas autorisé pendant une durée fixe (pendant $|LT| = K$ itérations) dite : tabou tenure. La solution s' est dite «solution tabou » et tout mouvement de s vers s' est un «mouvement tabou ».

La meilleure solution s' est gardée dans LT si $s' \notin LT$. Cette condition est appelée «condition tabou » et les solutions gardées dans LT ont un caractère de «statut tabou »**[LG 93]**.

- ***Fonction d'aspiration :***

Lors du choix de la meilleure solution $s' \in V(s)$, il est possible que l'on ait à départager plusieurs candidats donnant certes une même valeur à la fonction objective, mais ne nous dirigeant pas tous vers un optimum global. Il est ainsi parfois souhaitable de pouvoir retourner vers une solution visitée s, malgré le fait qu'elle fasse partie de liste LT des solutions taboues, ceci afin d'exploiter une nouvelle région voisine de s.

Pour cette raison, la méthode tabou fait intervenir un nouvel ingrédient appelé ***fonction d'aspiration*** et définie sur toutes les valeurs de la fonction objective :

Lorsqu'une solution s' voisine de la solution s fait partie de LT et satisfait de plus l'aspiration (c'est à dire f(s') < A(f(s))), on lève le statut tabou de cette solution s' et elle devient donc candidate lors de la sélection du meilleur voisin de s. En général, A(f(s)) prend la valeur de la meilleure solution s* rencontrée (on «aspire » donc à déterminer une solution meilleure que s*).

- ***Critère d'arrêt :***

Pour assurer la finitude de la procédure tabou, un critère d'arrêt est introduit comme suit : Si aucune amélioration n'est produite pour s* pendant un nombre maximum d'itérations, on peut alors stopper la recherche.

En d'autres termes, le processus itératif peut être interrompu quand on est très proche de la solution optimale f*, pour ce faire, on fixe un nombre maximum nbmax d'itérations entre deux améliorations de la meilleure solution s* rencontrée. Dés que nbmax itérations sont effectuées sans améliorations, le processus s'arrête.

Dans certaines applications, il est possible de déterminer une borne inférieure f* de f(s) s $\in$X. nbmax est défini de telle sorte que l'espace des solutions soit parcouru d'une manière exhaustive. De là, un autre critère d'arrêt est retenu, il est présenté comme suit :
On arrête le processus quand f(s*) atteint f* où s* est la meilleure solution trouvée jusqu'ici [**Her 95]**.

**L'organigramme illustrant le processus tabou est le suivant :**

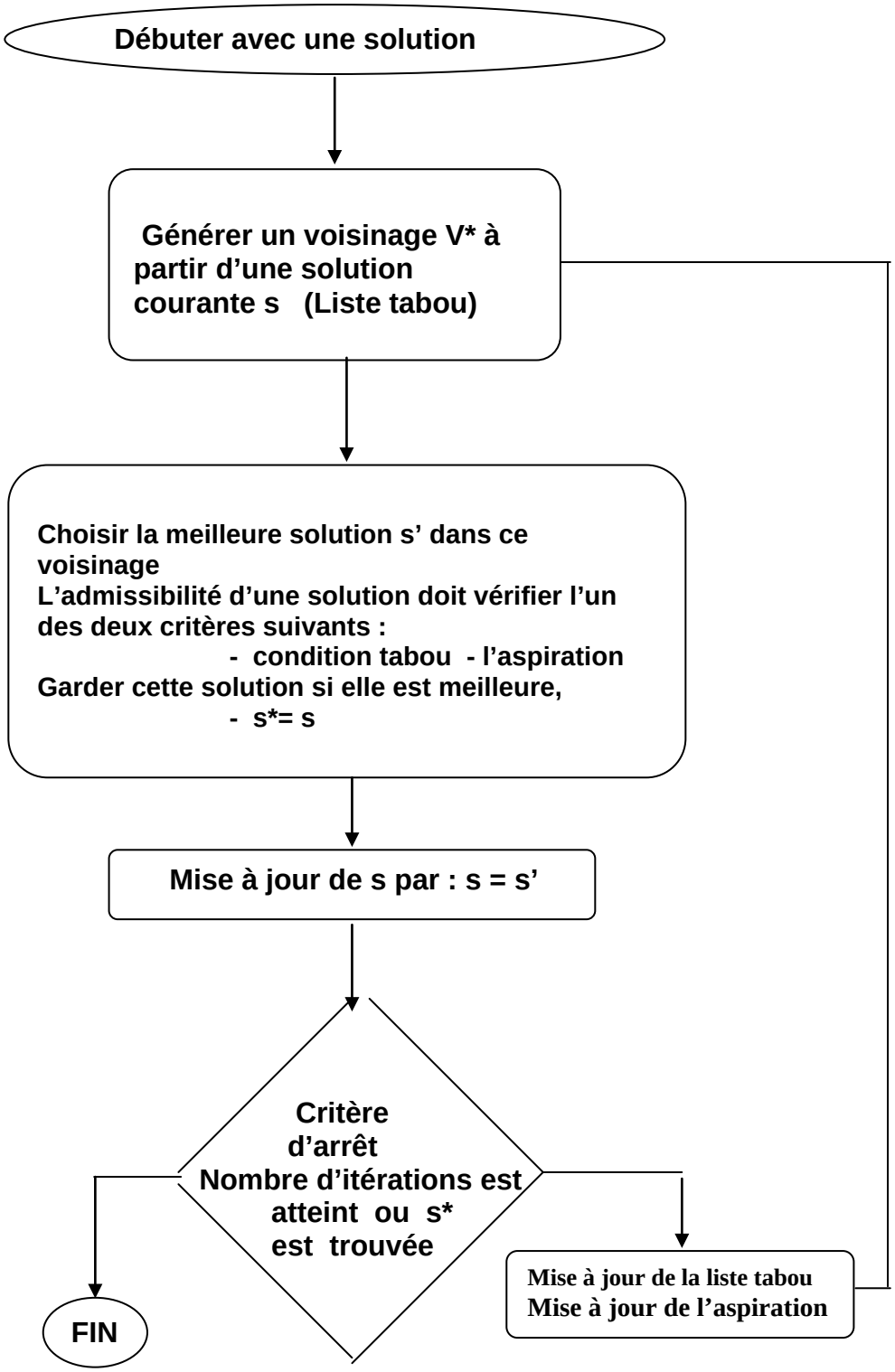


**Figure 2.1. Organigramme de la recherche tabou**

### 2.2.6. LA NOTION DE MEMOIRE DANS LA RECHERCHE TABOU[Glo 96] [LG 93] :

Une importante distinction dans la recherche tabou provient en différenciant entre la mémoire à court terme et la mémoire à long terme. Chaque type de mémoire est accompagné de ses propres stratégies. La plus utilisée, la mémoire à court terme. Elle fournit le chemin des solutions qu'elle a échangé durant le passé récent, elle est appelée : mémoire à base récente**[Glo 95]**. Cette mémoire est exploitée en assignant la désignation tabou aux solutions récemment visitées. Ceci empêche certaines solutions du passé récent d'être revisitée.

Dans quelques applications, les composantes de la mémoire à court terme sont suffisantes pour produire des solutions de haute qualité. Cependant, en général, la recherche tabou devient fermement significative par l'inclusion de mémoire à long terme et ses stratégies. L'utilisation de la mémoire à long terme ne requiert pas de longues exécutions avant que ses bénéfices deviennent visibles. Souvent, ces améliorations se manifestent et permettent la recherche de solution dans un court espace de temps.

La fonction de mémorisation à long terme consiste en une stratégie semblable à la stratégie de mémorisation à court terme, sauf qu'au lieu de centrer la recherche sur des régions contenant des bonnes solutions déjà trouvées, elle guide le processus de recherche à des régions qui ont un aspect différent de ceux déjà examinés.

Le principe de la mémoire à long terme consiste à pénaliser les solutions déjà visitées par le nombre de fois de leur apparition, multiplié par un facteur $\lambda$ permettant d'augmenter leur fonction objective. Cela est réalisé en utilisant une mémoire dite **«mémoire basée fréquence»**. Cette dernière consiste en l'enregistrement des caractéristiques d'un nombre de meilleures solutions sélectionnées et générées durant le processus de recherche.

Les stratégies de la mémoire à long terme prennent avantages de l'histoire de la recherche pour atteindre deux principaux buts : l'intensification régionale et la diversification globale.

### 2.2.7. STRATEGIES DE LA RECHERCHE TABOU :

- **Stratégies d'intensification :**

Elles sont désignées pour intensifier la recherche dans une région spécifique, elles sont basées sur la modification des règles de choix pour encourager les combinaisons de mouvements et les caractéristiques des solutions trouvées bonnes historiquement.

L'approche opère en enregistrant et en comparant les caractéristiques d'un nombre choisi de meilleures solutions générées durant une période particulière de la recherche.

- **Stratégies de diversification :**

Contrairement à l'intensification, la diversification guide le processus aux régions qui marquent des différences entre celles examinées auparavant.

L'approche diffère des autres méthodes qui cherchent la diversité en générant une recherche heuristique désignée pour produire un nouveau point de départ. Ces stratégies sont basées, généralement, sur la modification des règles de choix pour envoyer les attributs avec une reprise partielle ou totale du processus.

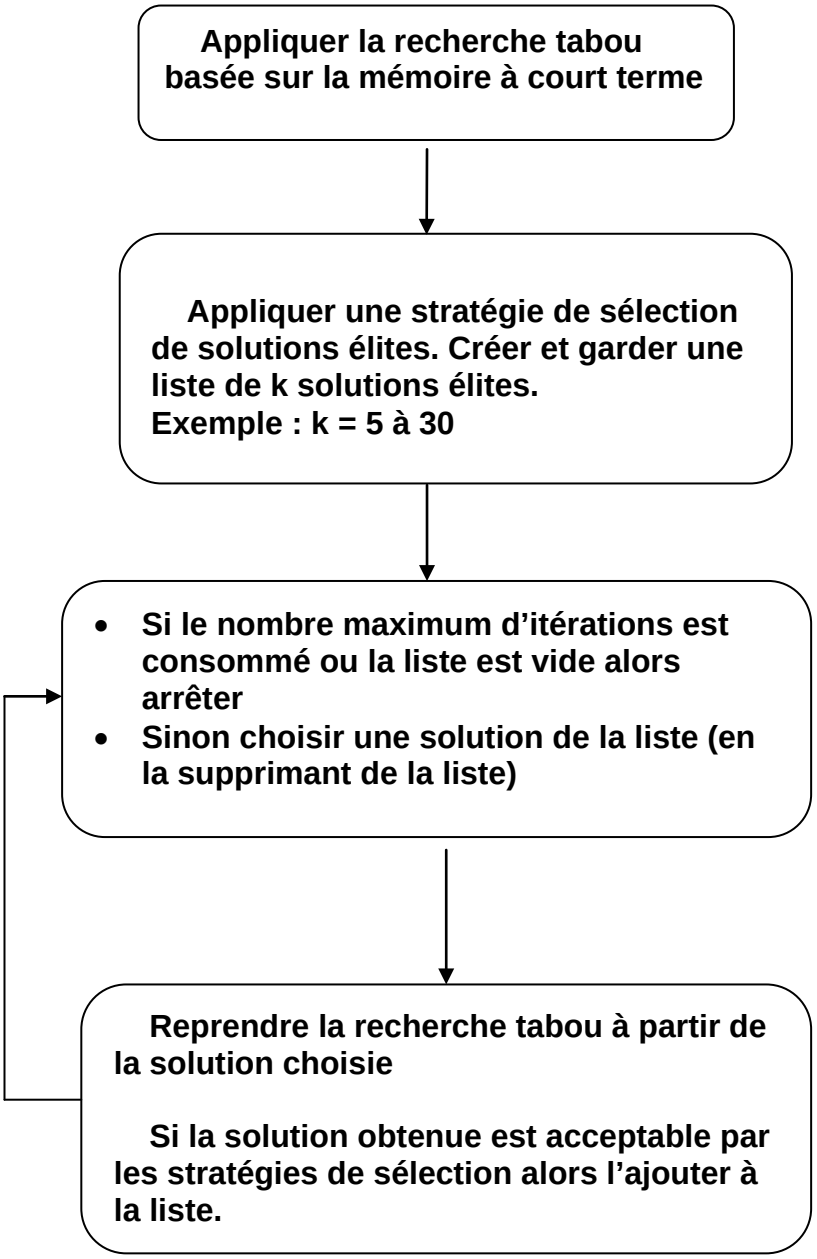


**Figure 2.2. Une stratégie d'intensification**

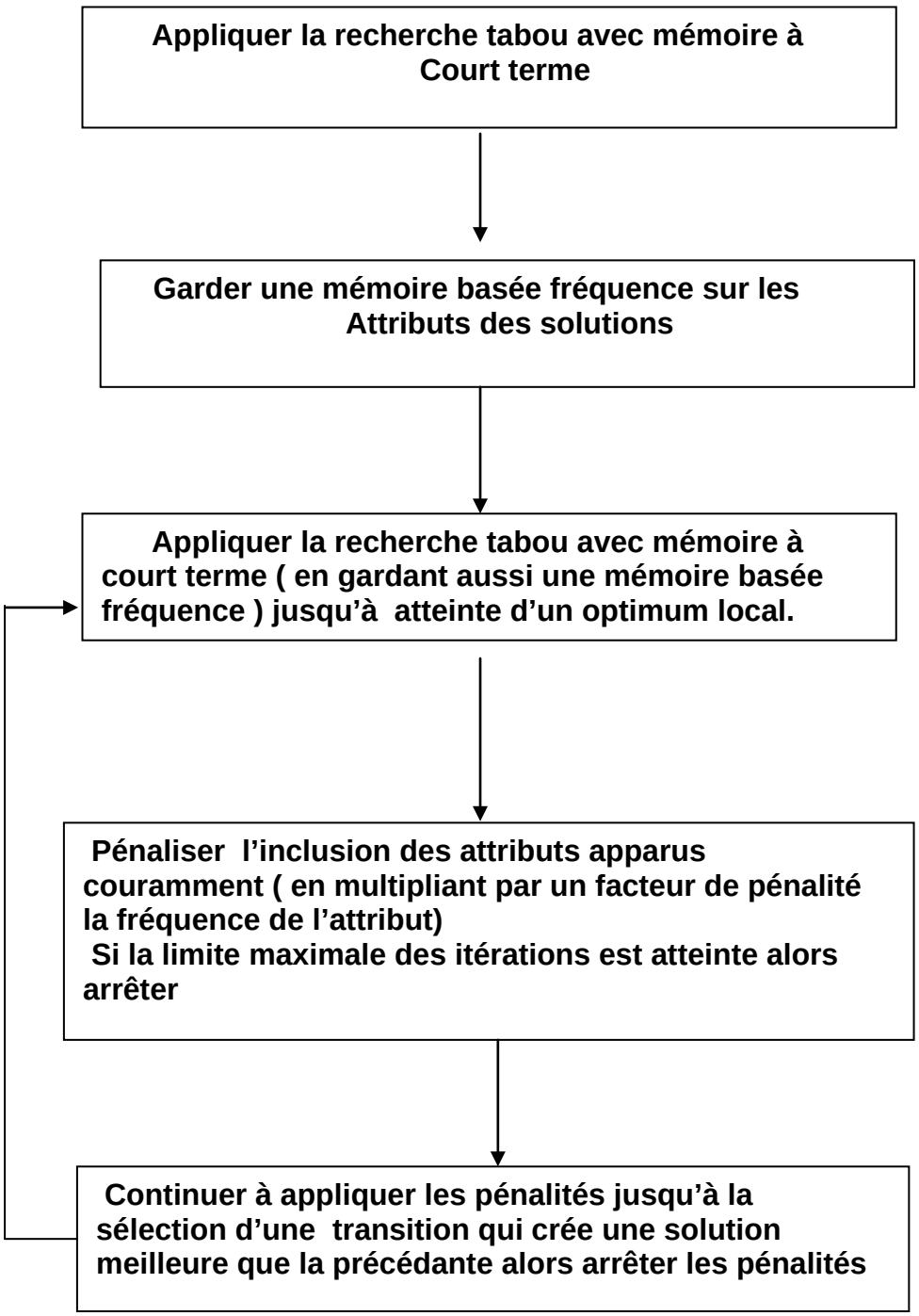


**Figure 2.3. Une stratégie de diversification**

- **Stratégies d'oscillation :**

La stratégie d'oscillation est étroitement liée aux origines de la recherche tabou, elle fournit un moyen pour réaliser une interaction efficace entre l'intensification et la diversification. Cette approche tend à faire varier la direction de recherche et la région visitée, et cela par le contrôle des évaluations des mouvements dans quelques voisinages de la solution courante.

Dans le cas où l'espace des solutions comporte des sous espaces disjoints, la stratégie d'oscillation fournit un mécanisme pour traverser les régions inadmissibles ( ne vérifiant pas les contraintes du problème), elle permet de créer des processus de recherche étendus qui peuvent admettre des solutions inadmissibles durant la recherche, mais elles sont pénalisées. Le composant d'oscillation de la procédure fait varier cette pénalité à chaque fois dans le but d'assurer une diversité efficace en cours de recherche.

- **Le chemin reliant :**

Une intégration utile des stratégies d'intensification et de diversification se produit dans l'approche appelée en anglais **«path relinking » [Glo 95][LG 93].** Cette approche génère de nouvelles solutions en explorant les trajectoires qui connectent les solutions élites, en commençant par une de ces solutions appelées solution initiale, et en générant un chemin dans l'espace du voisinage qui conduit vers d'autres solutions appelées solutions guides.

Cette approche peut être vue comme une stratégie qui cherche à unir les attributs des solutions de haute qualité, en favorisant ces attributs dans les mouvements choisis. Cependant, au lieu d'encourager seulement l'inclusion de tels attributs, l'approche de chemin reliant subordonne toutes les considérations dans le but de choisir les mouvements qui introduisent les attributs des solutions guides et de créer une bonne composition d'attributs dans la solution courante.

**Le chemin reliant**

1. **Identifier la structure de voisinage et associer une solution au chemin reliant .**
2. **Sélectionner une collection de deux ou plusieurs solutions de références et identifier celle qui va être une solution initiale et celles qui vont être des solutions guides.**
3. **Déplacer la solution initiale vers les solutions guides, générer une ou plusieurs solutions intermédiaires.**

**Figure 2.4. Une stratégie de chemin reliant**

- **Stratégies de la liste candidate : [ Glo 95]**

La qualité d'une solution peut dépendre de stratégies de la liste candidate appropriées. L'importance de ces approches est souvent négligée, malgré qu'elles soient fondamentales pour la recherche tabou pour effectuer des choix judicieux.

L'examination du voisinage complet avec la recherche tabou fournit généralement une solution de bonne qualité, mais peut être coûteuse en temps de calcul. Pour cette raison, il est souvent important d'appliquer la recherche tabou en conjonction avec une stratégie qui isole les régions de voisinage contenant des mouvements avec de bonnes caractéristiques en les mettant dans une liste de candidats. Ceci est une façon d'accélérer la recherche. Nous citons, dans ce qui suit quelques exemples de ces stratégies qui sont particulièrement utiles.

***- Stratégie de décomposition de voisinage :***

L'approche consiste à décomposer le voisinage en sous-ensembles, à chaque itération. Un seuil d'aspiration est appliqué pour limiter la fréquence de sélection des mouvements les moins attractifs. Généralement, cette stratégie permet de réduire le temps de calcul et de générer une certaine diversité dans la recherche.

- ***Stratégie de l'évaluation élite*** :

Elle consiste à sauvegarder une collection des meilleurs mouvements dans la liste candidate. A chaque itération, les mouvements appartenant à cette liste sont examinés en premier, suivi d'un sous-ensemble de voisinage. Ensuite, les mouvements de la liste candidate qui ne sont pas attractifs sont remplacés, périodiquement, après un nombre spécifié d'itérations, ou bien quand leurs qualités dans la liste sont détériorées sous un seuil choisi, une large portion du voisinage courant est explorée, afin de reconstruire la liste candidate.

## 2.3. LES ALGORITHMES GENETIQUES [Gol 89]:

Un algorithme génétique repose sur le principe de l'évolution naturelle d'organismes en respectant les phénomènes d'hérédité et la loi de survie énoncée par Darwin, selon lesquels, dans une population d'individus, ce sont les plus forts, c'est à dire les mieux adaptés à leur environnement qui survivent et pourront donner une descendance.

Le processus d'évolution se fait par deux mécanismes : la sélection naturelle et la reproduction.

### 2.3.1. La représentation chromosomique :

Le chromosome code une solution au problème, il représente un point de l'espace de recherche. Dans le cas du codage binaire, utilisé dans les premières versions des algorithmes génétiques, un chromosome est représenté par un vecteur dont les éléments appartiennent à {0,1}. Si la fonction à optimiser possède n variables, le chromosome sera constitué de la concaténation des n codages binaires. Il faut noter qu'il peut parfois être très difficile, de coder des solutions de cette manière. Le choix du codage a un impact sur les performances de l'algorithme génétique. En effet on doit choisir un codage qui n'est pas complexe dans le sens où le temps du décodage est minimal.

### 2.3.2. La population initiale :

La population initiale d'individus /solutions est l'ensemble des individus qui représentent des solutions d'un problème donné. Cette dernière est générée aléatoirement.

### 2.3.3. Fonction d'évaluation :

La fonction d'adaptation, décode un individu puis elle lui associe un coût. Un AG tend à maximiser ce coût, en permettant aux chromosomes ayant une grande valeur d'être retenus puis croisés( étape de reproduction).

### 2.3.4. Les opérateurs génétiques :

Les opérateurs génétiques définissent la manière dont les individus se recombinent et s'agencent pendant la phase de reproduction.

#### *1- La sélection :*

La phase de sélection spécifie les individus de la population qui doivent survivre. La méthode de base, appelée roue de loterie, attribue à chaque individu Vi une probabilité de survie Pi proportionnelle à son adaptation dans la population.

Lors de la phase de sélection, les individus sont sélectionnés aléatoirement en respectant les probabilités Pi associées pour former la population de la nouvelle

génération. Ceci s'effectue par le calcul d'une probabilité de sélection cumulée Qi, telle que :

$$\mathbf{Qi} = \sum_{J=1}^{i} Pj, \quad \mathbf{Pi} = \text{adaptation } Vi \,/\, \sum_{j=1}^{n} \text{adaptation } Vj$$

, où n est la taille de la population.

Puis, on génère aléatoirement un réel r sur l'intervalle [0,1].

L'individu Vi est sélectionné lorsque Qi-1 < r <= Qi. De ce fait les individus les mieux adaptés sont sélectionnés plusieurs fois et les plus faibles rarement voire jamais. Ce processus est réitéré n fois.

### *2- Le Croisement :*

Le croisement est défini comme suit (voir Figure 2.5):

Soient deux individus v1 et v2 juxtaposées, le croisement les coupe en un point choisi aléatoirement et produit deux nouveaux individus V'1 et V'2 après avoir échangé les parties coupées. Les deux individus V1 et V2 participant à cette opération sont assimilés aux parents, et les deux chaînes résultantes aux descendants.

### *3- La mutation :*

La mutation est un changement aléatoire d'une ou plusieurs positions d'un individu. L'opérateur de croisement devient moins efficace avec le temps, car les individus deviennent similaires. C'est à ce moment que le phénomène de mutation prend toute son importance. Chaque position de l'ensemble des chaînes de la population a une probabilité Pm de subir une mutation à chaque génération. Cet opérateur permet de sortir des optima locaux.

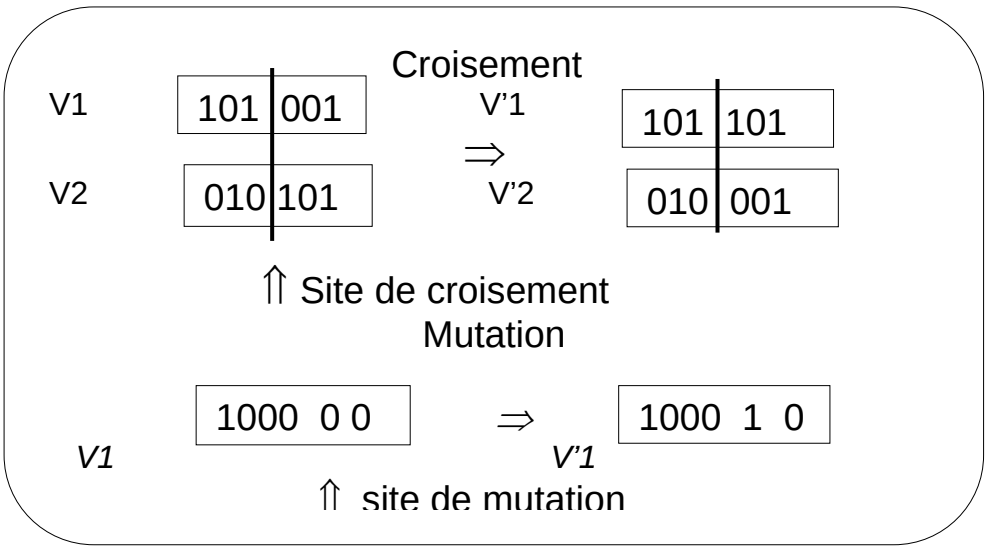


**Figure 2.5. Les opérateurs génétiques de croisement et de mutation**

***Les différentes étapes de l'AG sont les suivantes :***

- Générer aléatoirement une population initiale d'individus.
- Evaluation : Affecter à chaque individu son coût.

***Répéter***

- Sélection : Etablir une liste de paires d'individus susceptibles de se reproduire, le critère de sélection favorise les meilleurs.
- Reproduction : Appliquer les opérateurs génétiques à toutes les paires d'individus sélectionnées.
- Evaluation : Affecter à chaque individu son coût.
- Remplacement : Produire la nouvelle population en remplaçant les individus de manière à favoriser encore les meilleurs.

***Jusqu'à un certain critère d'arrêt.***

## 2.4. LE RECUIT SIMULE [Sia 95]:

Cette méthode est inspirée de la technique du «recuit » qui consiste à chauffer un métal jusqu'à l'état liquide, puis on le refroidit très lentement en marquant des paliers de température de durées suffisantes. La stratégie de la baisse contrôlée de la température conduit à un état solide cristallisé, qui est un état stable, correspondant à un minimum d'énergie.

La méthode de recuit simulé consiste à transposer ce procédé à la résolution d'un problème d'optimisation combinatoire : la fonction de coût du problème, analogue à l'énergie du métal, est alors minimisée, avec l'introduction d'une «température »qui est dans ce cas un simple paramètre de contrôle de l'algorithme jouant le rôle de la température du système physique dans la technique du recuit.

Dans cette méthode, on part d'une configuration initiale représentant une solution réalisable du problème et d'une température T. Ensuite, on fait subir à la configuration une modification élémentaire, si cette transformation a pour effet de diminuer la fonction coût du problème, elle est acceptée. Si au contraire elle provoque une augmentation $\Delta E$ de la fonction coût, elle est acceptée tout de même avec la probabilité $\exp(-\Delta E)$ où exp dénote la fonction mathématique exponentielle. En appliquant itérativement cette règle d'acceptation, on engendre une séquence de configurations. Après qu'un certain nombre d'itérations a été atteint (qui dépend de la taille du problème), on réduit la température T et on effectue une nouvelle série d'itérations à cette nouvelle température. On répète le cycle de refroidissement et des transitions jusqu'à un certain critère d'arrêt qui correspond à une température finale faible(proche de zéro).

### 2.4.1. Algorithme du recuit simulé :

L'algorithme du recuit simulé est présenté comme suit :

**Etape 1 Initialisation :**

On part d'une configuration initiale, on calcule son coût $E_0$.
$T_0 = T_{max}$ température initiale

**Etape 2 Itération :**

Le système se trouve à la température $T_i$ et possède l'énergie $E_i$.

1- On génère une nouvelle configuration voisine.
Soit $E_{i+1}$ le coût associé à la nouvelle configuration.
$\Delta E = E_{i+1} - E_i$

2- On accepte la nouvelle configuration avec la probabilité P :
$P(T_i, \Delta E) = MIN (1, exp(-\Delta E/T))$

3- On répète cette itération jusqu'à ce que le nombre d'itérations dépasse $N_T$ (aller à 2).

**Etape3 mise à jour et test d'arrêt :**

$T_i = a * T_{i-1}$ /* a le taux de décroissance de la température */
Si $T_i \geq T_{min}$ alors aller à 2 /* $T_{min}$ est la température minimale */

D'après cet algorithme, on comprend le rôle confié à la température. A haute température, $exp(-\Delta E/T)$ est voisin de 1, donc la plupart des configurations sont acceptées, l'algorithme équivaut à une simple marche aléatoire dans l'espace de recherche. A basse température, $exp(-\Delta E/T)$ est voisin de 0, donc la plupart des configurations augmentant la fonction coût (l'énergie) sont refusées. A température intermédiaire, l'algorithme autorise de temps en temps des transformations qui dégradent la fonction coût, il laisse ainsi au système une chance de s'extraire d'un minimum local **[Sia 95]**.

### 2.4.2. L'organigramme de l'algorithme du recuit simulé :

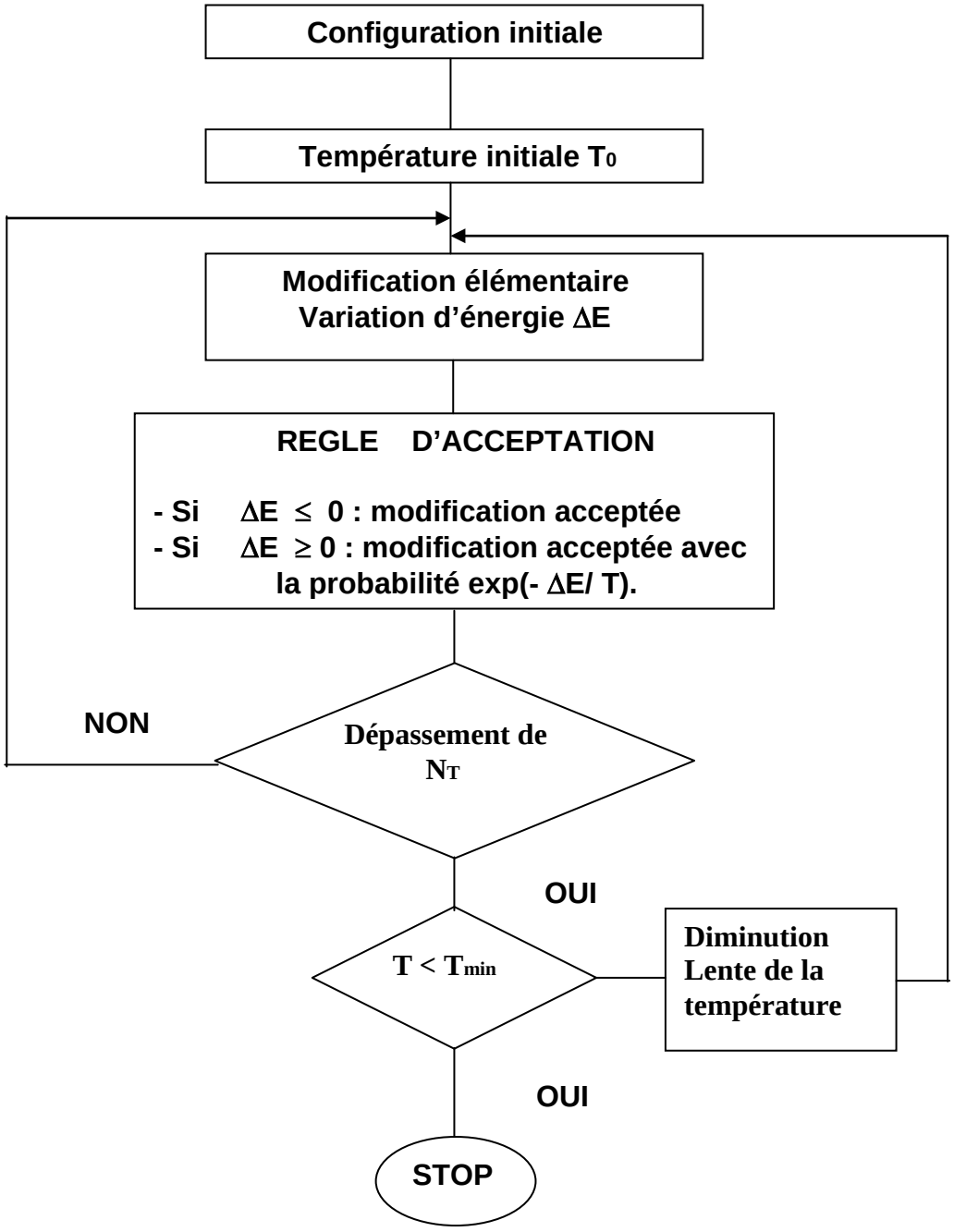


**Figure 2.6. Organigramme de l'algorithme de recuit simulé.**

## 2.5. CONCLUSION :

Ce chapitre est consacré aux méta- heuristiques classiques, qui sont des algorithmes itératifs produisant des bons résultats pour les problèmes d'optimisation difficiles. Le chapitre suivant est consacré à examiner deux autres méta- heuristiques récentes : la recherche dispersée et la recherche locale guidée.

# Chapitre 3 :

# La recherche dispersée et la recherche locale guidée

## 3.1. INTRODUCTION :

La recherche dispersée et la recherche locale guidée sont des méta-heuristiques apparues ces dernières années avec une ambition commune : résoudre efficacement les problèmes d'optimisation combinatoire en particulier ceux qui sont classés NP- difficiles.

En effet la plupart de ces méthodes sont basées sur le pilotage d'un point courant dans un certain domaine. Les déplacements de ce point sont alors décidés de façon à optimiser une certaine fonction objective. Très souvent, la structure du domaine considéré ne permet pas d'établir un lien fort entre les notions d'optimalité globale et d'optimalité locale. De ce fait, divers artifices de contrôle existent, qui visent à éviter au point courant de s'enfermer sur un optimum local de mauvaise qualité. Ces artifices de contrôle tendent à autoriser des déplacements dégradant temporairement le point courant, moyennant un tirage au sort ( le recuit simulé), une mutation (les algorithmes génétiques) ou bien une mémorisation des positions déjà visitées (la recherche tabou).

Dans ce qui suit nous présentons deux autres méthodes récentes et nous donnons d'autres mécanismes utilisés pour échapper à l'optimum local : la recherche dispersée qui s'appuie sur la diversification et l'amélioration et la recherche locale guidée qui utilise une fonction objective augmentée.

## 3.2. LA RECHERCHE DISPERSEE [Lag 99][GLM 00] [GLMC 99]:

La recherche dispersée appelée en anglais «**SCATTER SEARCH** » est une méta- heuristique destinée à résoudre au mieux les problèmes dits d'optimisation difficiles : elle fait évoluer une population de points (solutions au problème) générée via un générateur de diversification.

De plus, la recherche éparpillée établit une liste des meilleurs individus «**ensemble référence**» en permettant aux éléments ayant un coût élevé ou une disparité assez importante d'être retenus puis croisés ou combinés (**étape de combinaison**). Le critère de sélection favorise les meilleurs, c'est-à-dire les individus de haute qualité en regardant leur coût ou les solutions diverses de l'espace de recherche.

Autrement dit, la recherche dispersée unit un ensemble préféré de points dits points références pour produire les points élites qui vont devenir la source des nouveaux points références en utilisant un opérateur de combinaison permettant de construire des solutions admissibles. Le processus d'évolution est réitéré jusqu'à ce qu'aucune combinaison n'améliore la qualité ou la diversité de l'ensemble des points références.

### 3.2.1. DESCRIPTION DE LA RECHERCHE DISPERSEE [GLM 00] :

La recherche dispersée est une méta-heuristique à stratégies d'évolution. Elle intègre les idées majeures des méthodes évolutionnistes ( population de solutions, recombinaison de solutions).

Un algorithme génétique est aussi une approche évolutionniste, il présente plusieurs inconvénients liés à sa structure classique **[Tal 95]**. Nous citons :

- L'incapacité du processus de recherche d'exploiter au mieux certaines régions particulières. Ceci est dû au fait que les opérateurs génétiques de mutation et de croisement agissent tous les deux d'une manière aveugle.

- La convergence prématurée après un certain nombre d'itérations. Ceci est dû au fait que les individus deviennent de plus en plus identiques au fur et à mesure que le nombre de générations augmente. La manière avec laquelle agissent les opérateurs génétiques est une cause principale de cet inconvénient, car comme dans ce cas il faut augmenter le taux de diversification pendant une période suffisante. Chose difficile a réaliser du fait que les opérateurs génétiques classiques restreignent significativement l'effet du contexte, ce sont des opérateurs à effet de contexte neutre **[Tal 95]**.

Dans le but de pallier ces principaux inconvénients, la recherche dispersée s'appuie sur un principe qui consiste à générer la population de départ à l'aide d'une stratégie de diversification. De plus, elle autorise des synergies avec d'autres méthodes itératives, qui peuvent être intégrées dans le processus de recherche.

Dans cette approche, on génère de nouvelles solutions en explorant des trajectoires qui lient plusieurs solutions de départ, appelées solutions de haute qualité, et cela en démarrant par l'une de celles-ci, appelée solution initiale, puis en générant un chemin dans l'espace voisin qui mène vers les autres solutions, dites solutions diverses. Les solutions diverses et de haute qualité sont appelées solutions références.

Le schéma suivant présente une instance de la recherche dispersée :

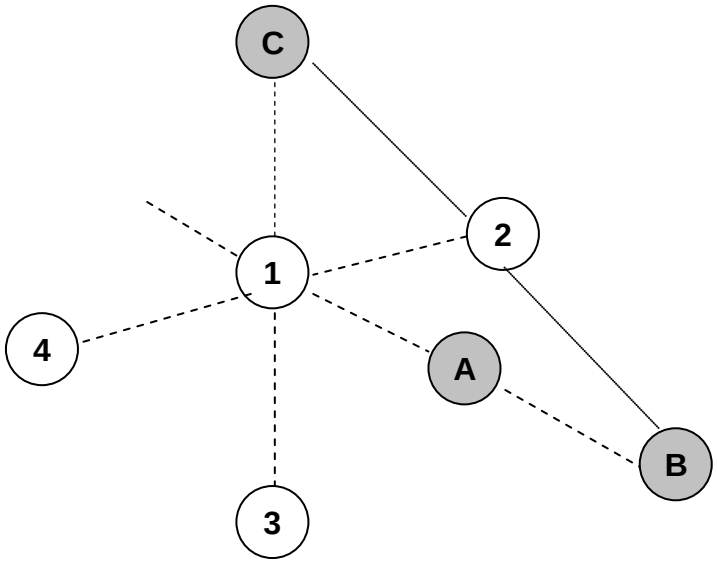


***Figure 3.1. Schéma représentant une instance de la recherche dispersée.***

On remarque les trois solutions de référence A, B et C. Des solutions intermédiaires sont générées en démarrant par l'une des solutions de référence, puis en générant un chemin dans l'espace voisin qui mène vers les autres solutions. Ceci est effectué en utilisant un opérateur de combinaison. La combinaison joue un rôle presque semblable à celui d'un opérateur de croisement, et cela du fait qu'elle consiste en la construction d'une ou plusieurs solutions combinées en sortie partageant certains attributs avec les solutions références en entrée. Les nouvelles solutions seront améliorées et réintégrées dans l'ensemble des points références. Les points 1, 2, 3, et 4 sont les nouvelles solutions combinées.

La recherche dispersée peut contribuer à bien gérer le compromis entre la diversification et l'intensification. Pour favoriser l'intensification, les solutions références sont choisies de telle sorte qu'elles parviennent d'une région commune «solutions de haute qualité ». Pour favoriser la diversification, les solutions références sont choisies de telle sorte qu'elles parviennent de différentes régions «solutions diverses » **[GLM 00].** Cette façon de faire, augmente les chances d'atteindre des solutions optimales.

***Les différentes étapes de la recherche dispersée sont les suivantes :***

- Générer une population initiale variée ;
- Extraire un ensemble des meilleurs individus qui vont construire les points références ;
- Grouper les points références «combinaison ».

### 3.2.2. LA POPULATION INITIALE :

La population initiale est l'ensemble des individus qui représentent des solutions d'un problème donné. Contrairement aux algorithmes génétiques où les solutions de départ sont générées aléatoirement, la recherche dispersée utilise une stratégie de diversification, pour avoir une population variée qui permettra d'explorer des zones diverses de l'espace de recherche.

A chaque solution un coût correspondant à la fonction objective est associé. La population de départ est ordonnée suivant la fonction objective en commençant par les meilleurs individus ayant un coût assez élevé.

### 3.2.3. CONSTRUCTION DES SOLUTIONS DE REFERENCE R :

Dans cette phase on spécifie les individus de la population initiale qui doivent se combiner et produire une nouvelle génération.

En effet le choix d'une solution pour l'ensemble **R** «solutions références » est un point essentiel dans la recherche dispersée, pour lequel une bonne réflexion est nécessaire. Il existe deux stratégies de choix de l'ensemble **R :**

Celle qui consiste à choisir un ensemble de solutions et de les évaluer suivant la fonction objective pour pouvoir déduire les solutions de haute qualité, l'autre consiste à choisir un ensemble de solutions et de les évaluer suivant la disparité pour pouvoir déduire les solutions diverses**[LMC 99]**.

Par conséquent, on aura deux sous-ensembles R1 de taille B1 et R2 de taille B2, tel que les B1 solutions sont de haute qualité et les B2 solutions sont les points les plus éloignés les uns des autres dans l'espace de recherche.

En d'autres termes, l'ensemble R est une liste de longueur B = B1+B2, utilisée pour conserver les meilleures solutions. A chaque évolution on introduit la solution combinée, si elle est meilleure, et on rejette la plus mauvaise.

### 3.2.4. RENOUVELLEMENT DE LA POPULATATION :

Le renouvellement de la population, autrement dit la création d'une nouvelle génération, est obtenu par itération de l'algorithme de recherche dispersée qui va créer de nouveaux points et en détruire d'autres. A partir de l'ensemble R, la recherche dispersée génère des sous-ensembles appelés aussi points élites qui vont construire des solutions combinées en utilisant un opérateur de combinaison.

La recombinaison consiste à grouper les éléments de R par paire, par triplet, et ainsi de suite, c'est-à-dire avec un ensemble R de taille B on peut générer **4 types** de solutions définis comme suit :

**Type 1** : Il consiste à grouper les éléments de R par paire, puis, pour chaque paire de solutions appliquer l'opérateur de combinaison, améliorer et insérer la solution générée dans R.

**Type 2**: Les solutions seront combinées trois à trois, en prenant un élément de type1 et on lui rajoute une meilleure solution qui n'apparaît pas dans le couple de solutions courant.

**Type 3** : Les solutions seront combinées quatre à quatre. Pour chaque élément de type2, lui rajouter une bonne solution, puis, pour chaque quadruplet obtenu, appliquer l'opérateur de combinaison

**Type 4** : Les i meilleurs points de R seront groupés pour donner d'autres Solutions combinées, où i = 5 jusqu'à B, c'est –à-dire le type 4 comprend des sous–ensembles de taille 5,….jusqu'à B meilleurs éléments.

**Exemple :**

Soit l'ensemble R de taille 5 suivant :

X(1) : (0, 0, 0, 0, 0, 0, 0, 0,0)
X(2) : (1, 0, 1, 0, 1, 0, 1, 0, 1)
X(3) : (1, 0, 0, 1, 0, 0, 1, 0, 0)
X(4 ) : (1, 0, 0, 0 ,1, 0, 0, 0, 1)
X(5 ): (1, 1, 1, 1 ,1, 1, 1, 1, 1)

A partir de cet ensemble, 4 types de solutions seront générés et peuvent être présentés comme suit :

| TYPE | Sous-ensembles |
|---|---|
| 1 | (1,2) (1,3) (1,4) (1,5)<br>(2,3) (2,4) (2,5)<br>(3,4) (3,5)<br>(4,5) |
| 2 | (1,2,3) (1,2,4) (1,2,5)<br>(1,3,4) (1,3,5)<br>(1,4,5) |
| 3 | (1,2,3,4) (1,2,3,5)<br>(1,2,4,5) |
| 4 | (1,2,3,4,5) |

**Figure 3.2. Les solutions élites**

Où 1, 2, 3, 4, 5 sont les solutions de l'ensemble R.

On aura les quatre types suivants :

- **Le type 1** : Contient 10 sous–ensembles, qui vont participer à la combinaison. Chaque élément va construire une troisième solution à l'aide d'un opérateur structuré.

- **Le type 2** : Contient 6 sous–ensembles. Chaque élément dans cet ensemble est formé à partir de l'ensemble de type précédant, en évitant la redondance.

- **Le type 3** : Pour construire un élément de type 3, il suffit de prendre un élément de type 2 et lui rajouter la meilleure solution dans R et qui n'apparaît pas dans le triplet courant.

- **Le type 4** : Afin de créer un élément de type 4, i solutions de R seront groupées, où i =5 jusqu'à B solutions. Dans notre cas, on aura un seul élément de taille 5 puisque le B est égal à 5.

### 3.2.5. LA COMBINAISON :

L'aveuglance exagérée des opérateurs génétiques classiques est à l'origine de plusieurs problèmes, tels que la convergence prématurée de l'algorithme génétique. Donc, il est intéressant d'utiliser un opérateur plus évolué prenant en considération le contexte du problème.

La combinaison structurée est un opérateur permettant de produire des solutions admissibles, à partir d'un ensemble de solutions appelées solutions élites.

Dans la recherche dispersée, la combinaison dépend de la fonction objective à optimiser «contexte du problème » ainsi que de la représentation utilisée pour modéliser les solutions du problème.

Précisément, sur l'identification d'une collection d'une ou plusieurs solutions élites pour guider le chemin d'une solution donnée, des poids sont assignés aux attributs des solutions guides, comme mobile pour être sélectionnés.

Des gros poids sont assignés aux attributs qui surviennent dans un grand nombre de solutions guides permettant de donner une accentuation des solutions de haute qualité ou bien avec des caractéristiques spéciales.
Quelques conceptions d'attributs peuvent partager des degrés de similarité, et dans ce cas, il peut être utile de garder la solution vectorielle, aussi prévoir «des votes» pour favoriser ou décourager les attributs particuliers.

**Exemple** : « problème d'ordonnancement »

On entend par ordonnancement, le placement d'un ensemble de tâches qui peuvent être indépendantes ou soumises à des contraintes de précédence, sur un ensemble de processeurs. Ce placement doit minimiser la durée totale de l'exécution.

Soient les deux solutions initiales :

X(1) = (1, 2, 3, 4, 5, 6, 7, 8, 9) W(1) = 2/3
X(2) = (1, 8, 6, 4, 9, 3, 5, 2, 7) W(2) = 1/3

Où : X(i) est une solution de référence et W(i) son poids.

A partir de ces deux points de référence, on essaie de trouver une nouvelle solution X(3) de la manière suivante :

- Soit la relation de précédence (i, j) tel que i est la tâche initiale, trouver le meilleur successeur de i, j à partir de X(1) et X(2).

- A chaque point de référence X(i), une note «voteX(i)» est associée.

Dans notre cas, le nouveau vecteur X(3) contient au début la valeur 1. Donc le problème revient à chercher un j, le successeur de 1, à partir de X(1) et X(2).

**Initialement X(3) = (1, , , , , , , ), i= 1,j = ?.**

**Cas 1** : d'après X(1), j = 2 avec voteX(1) =1 et voteX(2) =0
X(3) = (1, 2, , , , , , , )

**Cas 2** : d'après X(2), j = 8 avec voteX(1) =0 et voteX(2) =1
X(3) = (1, 8, , , , , , , )

**Etape d'évaluation :**

Afin de sélectionner un successeur, on doit minimiser la formule donnée par GLOVER comme suit :

$$\text{Minimise} \sum_{q=1}^{\text{Taille de R}} \left| \text{contribution}(k:q) - W(k) \right| \quad \textbf{|Glo 94]}$$

on aura donc :

**Cas 1** : $|1- 2/3| + |0- 1/3| = 2/3.$

**Cas 2** : $|0- 1/2| + |1- 1/3| = 4/3.$

$\Rightarrow$ j = 2 c'est à dire c'est le premier cas qu'est sélectionné.
$\Rightarrow$ X(3) = (1, 2, , , , , , , )

En réitérant le même processus, on aura à la fin la solution X(3) suivante :

(1, 2, 7, 8, 9, 3, 4, 5, 6)

### 3.2.6. METHODE D'AMELIORATION :

Dans cette méthode, on part d'une configuration représentant une solution réalisable du problème obtenue par la méthode de combinaison. Ensuite, on fait subir à cette solution combinée des modifications élémentaires (série de transformations qui a pour effet d'améliorer le coût de la solution combinée).
Afin d'avoir une solution meilleure, on applique itérativement cette méthode d'amélioration qui engendra une séquence de solutions. On répète le cycle d'amélioration et des transitions jusqu'à un certain critère d'arrêt qui correspond à un certain nombre d'itérations.

**Exemple :**

Dans l'optimisation combinatoire, la recherche locale est la méthode la plus utilisée pour améliorer une solution donnée. Elle peut être présentée comme suit :

- Soit s une solution à améliorer ;

***Répéter***
- Générer un ensemble V* $\subseteq$ N(s) de solutions voisines de s ;
- Choisir la meilleure solution s' $\in$V*
- ***si*** f(s) < f(bestsol) ***alors*** bestsol = s' ;

***Jusqu'à un certain nombre d'itérations.***

### 3.2.7. LA DIVERSITE:

Afin d'augmenter la probabilité d'atteindre un optimum de haute qualité, les solutions de l'ensemble R sont choisies suivant leur fonction objective «Solutions de haute qualité et suivant leur disparité «Solutions diverses».

En ce qui concerne la diversité, la recherche dispersée est une méthode qui cherche la diversité en générant une recherche heuristique désignée pour produire un nouveau point. La diversification guide le processus aux régions qui marquent des différences entre celles examinées auparavant.

Pour avoir une population variée on utilisera une méthode assurant la diversité à travers le calcul d'une distance «disparité» mesurée entre les solutions de haute qualité et les autres solutions générées dans l'espace de recherche.

**Exemple :**

Soient deux solutions vectorielles X et Y, la distance entre X et Y sera mesurée comme suit :

// X solution vectorielle de n variables
// Y solution vectorielle de n variables

Dist = 0 ;

**Pour** i=1 à n **faire** Si (X[i] <> Y[i]) alors Dist = Dist +1 ;

Distance (X, Y) = Dist

⇨ **// Distance (x, y) = la différence entre les bits //**

### 3.2.8. L'ALGORITHME DE LA RECHERCHE DISPERSEE [Lag 99]:

Une version de l'algorithme de la recherche dispersée est présentée comme suit :

**INITIALISATION :**

- Engendrer une population initiale P en utilisant une stratégie de diversification
- Affecter à chaque solution son coût.
- Ordonner la population suivant la fonction objective

// Itermax nombre maximum d'itérations

**ITERATION :**

**Pour** i=1 à itermax **Faire**

- Etablir une liste de solutions de taille B appelée liste de points références R, le critère de sélection favorise les meilleures.
- L'ensemble R est de taille B = B1+B2 où

// Les B1 solutions sont de haute qualité
// Les B2 solutions sont diverses

Nouveau = Vrai

**Tant que** (Nouveau)
**Faire**

- Calculer la taille de l'ensemble des solutions élites Maxsubset ;
- Nouveau = faux ;

**Pour** i=1 à Maxsubset **Faire**

- Générer les différents types de solutions
- Appliquer l'opérateur de combinaison pour obtenir une nouvelle solution s
- Améliorer la solution s
- Evaluer la solution améliorée s*.

**Si** (coût (s*) > coût(B1 ème solution de R) et s* n'apparaît Pas dans R) **alors**

- Insérer s*
- Ecraser la mauvaise
- Nouveau = Vrai ;

**Sinon**

**Si** (disparité (s*) > disparité(B2ème solution de R) et s* n'apparaît Pas dans R) **alors**

- Insérer s*
- Ecraser la mauvaise
- Nouveau = Vrai;

**Fait**

**Fait**

- Régénérer la population en remplaçant les solutions de manière à favoriser encore les meilleures.

**Fait**

## 3.3. LA RECHERCHE LOCALE GUIDEE [Vou 97] [VT 98]:

La recherche locale guidée est considérée comme étant une puissante procédure d'optimisation, capable d'échapper aux optima- locaux.

La recherche locale guidée a remporté un immense succès dans de nombreux problèmes d'optimisation combinatoire à savoir le problème du voyageur de commerce, les problèmes d'affectation, et les problèmes d'ordonnancement **[Vou 97]**. Ces problèmes sont caractérisés par un haut degré de complexité dû à la présence d'un nombre assez important d'optima- locaux dans l'espace de recherche. Pour cette raison il n'existe pas d'algorithmes pouvant générer une solution optimale en un temps polynomial.

L'heuristique de recherche locale s'arrête au premier minimum local qui peut être très loin du minimum global recherché. C'est pourquoi la recherche locale guidée a été mise au point.

Pour pouvoir ressortir des optima–locaux une autre variante de la recherche locale a été développée et qui est une amélioration de la première dans le sens où elle modifie la fonction objective dont le but est de pénaliser les solutions de mauvaise qualité et d'approcher le minimum global.

Dans ce qui suit, nous présentons tout d'abord la méthode de base dite «recherche locale». Le point suivant est consacré à la description et la conception d'une approche basée sur la recherche locale afin de trouver des bons résultats pour les problèmes d'optimisation combinatoire : recherche locale guidée.

### 3.3.1. LA RECHERCHE LOCALE :

Un algorithme itératif de recherche locale part d'une configuration initiale (une solution au problème) et essaie de l'améliorer. L'amélioration consiste à effectuer des transformations successives sur la configuration initiale. Si le coût d'une configuration générée est meilleur que celui de la configuration courante, cette dernière sera remplacée par cette nouvelle configuration, sinon elle sera gardée. Ce processus est réitéré jusqu'à ce qu'aucune transformation ne réduise le coût de la configuration courante.

La recherche locale est une méthode itérative initialisée avec une solution qu'elle cherche à améliorer à chaque itération.

En effet, les algorithmes itératifs de recherche locale conduisent très souvent à des solutions suboptimales. Dans le but d'échapper aux optima- locaux, d'autres techniques ont été développées telles que : la recherche locale guidée qui produit une solution acceptable.

*Ci dessous une version de l'algorithme itératif de recherche locale :*

**Début**
- Générer une configuration initiale S
- **Tant que** (coût (S) > 0)
  **Faire**
  - Générer une configuration voisine S'
  - **Si** coût (S) < coût (S') **alors** S= S'

  **Fait**
- retourner (S)

**Fin**

### 3.3.2. LA RECHERCHE LOCALE GUIDEE [VT 98]:

La recherche locale guidée est une méta- heuristique destinée à résoudre efficacement les problèmes d'optimisation combinatoire. Elle permet, en principe, d'éviter le piéage des optima- locaux de la fonction objective moyennant une pénalisation de certaines caractéristiques dans la solution.

En effet, la recherche locale guidée est une variante de la recherche locale. Elle modifie la fonction objective dans le but d'approcher le minimum global.

Le principe de la recherche locale guidée revient à pénaliser certaines solutions jugées mauvaises par le nombre de fois de leurs apparitions multiplie par un facteur $\lambda$ permettant d'augmenter leur fonction objective. Cela est réalisé en utilisant une série de formules présentées dans ce qui suit.

*Formule 1 :* **Ini(S) =**
- **1 Si, Pour la configuration S, la mauvaise caractéristique est présente ;**
- **0 Sinon.**

*Formule 2 :* **C(i) = poids de la caractéristique i**

*Formule 3 :* **Utilité(i) = Ini(S) * C(i) / 1+P(i)**

Où :

**P(i)** : Est la pénalité ou encore un compteur de fréquence. Au début P(i) est initialisée à zéro, quand la caractéristique i est mauvaise P(i) est mis à jour.

**Ini(S)** : Indique si la caractéristique i est mauvaise ou pas.

**Utilité(i)** : Permet de pénaliser les solutions déjà visitées.

Donc le problème revient à pénaliser les caractéristiques de haute utilité. Cette façon de faire permet de pénaliser les solutions déjà visitées ce qui conduira à une exploration plus diversifiée de l'ensemble de recherche.

***Par conséquent, on aura la fonction objective augmentée suivante :***

$$H(S) = G(S) + \lambda \sum_{i=1}^{m} Ini(S) * P(i)$$

Où :

H : La fonction objective augmentée ;
G : Le coût de la solution S
S : La configuration courante ;
Ini(S) : L'indicateur ;
P(i) : La pénalité ;
m : Le nombre de caractéristiques.
$\lambda$ : Le facteur de pénalisation
$\lambda$ > 1 : Recherche diversifiée
$\lambda$ < 1 : Recherche intensifiée.

### 3.3.3. L'ALGORITHME DE LA RECHERCHE LOCALE GUIDEE [VT 98]:

L'algorithme de la recherche locale guidée est présenté comme suit :

**INITIALISATION :**

- Générer aléatoirement une solution initiale S
- Affecter à la solution générée son coût.

// G est la fonction objective standard
// Itermax nombre maximum d'itérations
// H = la fonction objective augmentée
// m = nombre de caractéristiques

**Pour** i=1 à m **Faire** P(i)=0 ;

**ITERATION :**

**Pour** i=1 à Itermax **Faire**

- S := recherche locale(S, H)

// appel de l'algorithme de recherche locale avec les paramètres H et S
// la recherche locale retourne une solution S qui est l'optimum local

**Pour chaque** ( mauvaise caractéristique fi)
**Faire**

- Calculer son utilité
  **Utilité(fi) = Infi(S) * C(fi) / 1+P(fi)**
- Soit utilité- max égale à l'utilité maximale

**Si** (utilité(fi) = utilité- max) **alors** P(fi) = P(fi)+1 **finsi**

**Fait**

**Fait**

Retourner la meilleure solution avec le meilleur coût

**FIN**

### 3.4. CONCLUSION :

Dans le but de trouver une solution aussi bonne que possible, différentes méta-heuristiques ont été développées. Certains de leurs outils de base sont :

- La randomisation, utilisée pour explorer des zones diverses de l'espace de recherche.
- La mémorisation, pour guider plus efficacement la recherche de la solution.
- L'intensification et la diversification, afin d'augmenter les chances d'avoir une solution de bonne qualité.

Dans le chapitre qui suit, nous proposons deux algorithmes pour la résolution du problème **MAX-W-SAT**. L'un est basé sur la recherche dispersée et l'autre est basé sur la recherche locale guidée.

# Chapitre 4 :

# Proposition d'algorithmes de résolution de MAX-W-SAT basés sur la recherche dispersée et la recherche locale guidée

## 4.1. INTRODUCTION :

La recherche dispersée et la recherche locale guidée sont des méta-heuristiques introduites dans le cadre de ce travail dans le but de résoudre efficacement le problème de la satisfiabilité maximale pondérée **MAX-W-SAT** qui est un problème important de l'intelligence artificielle.

En se basant sur la stratégie de diversification, la recherche dispersée génère un ensemble de solutions variées qui seront considérées comme les solutions candidates à améliorer, en utilisant une méthode d'amélioration.

La recherche locale guidée commence à appliquer une recherche locale simple pour obtenir un point de départ, généralement un optimum local. Afin d'échapper à cet optimum, la recherche locale guidée modifie la fonction objective en pénalisant les solutions de mauvaise qualité.

Dans ce qui suit, nous proposons deux solutions au problème MAX-SAT. La première est basée sur la recherche dispersée alors que la seconde est basée sur la recherche locale guidée.

## 4.2. ADAPTATION DE LA METHODE DE RECHERCHE DISPERSEE AU PROBLEME ETUDIE :

Pour pouvoir appliquer la recherche dispersée on doit passer par les étapes suivantes :

- Générer un ensemble de solutions dispersées «Population variée P» ;
- Avoir une fonction objective qui permet d'évaluer les solutions ;
- Choisir une certaine manière de générer l'ensemble $R \subset P$ afin de choisir Un ensemble de solutions meilleures dites solutions de référence;
- Avoir un opérateur de combinaison qui nous permet de construire d'autres Solutions ;
- Définir le test d'arrêt.

### 4.2.1. LA POPULATION INITIALE :

Afin d'avoir une population variée, les solutions initiales sont générées à l'aide d'une stratégie de diversification. Cette dernière, permettra d'explorer des zones diverses de l'espace de recherche.

Une solution au problème **MAX-W-SAT** est un vecteur X de taille n, tel que n est le nombre de variables où chaque variable X[i] est une valeur booléenne(0 ou 1).

**Exemple :**

Soit la donnée SAT suivante :

$$\begin{cases} c_1 : x_1 + \xi_2 + \xi_3 \\ c_2 : \xi_1 + x_2 + x_3 \\ c_3 : x_2 + x_3 \end{cases}$$

Pour :

Un ensemble de clauses $C = \{ c_1, c_2, c_3 \}$,
Un ensemble de variables $X = \{ x_1, x_2, x_3 \}$

Le tableau V, ci-dessous contient respectivement les valeurs 1, 0, 1. Les valeurs attribuées respectivement aux variables $x_1$, $x_2$, $x_3$ représentent une solution de la donnée SAT.

V = | 1 | 0 | 1 |  V[1]= $x_1$=1, V[2]= $x_2$= 0, V[3] =$x_3$=1.

- **Méthode de diversification :**

La méthode de diversification utilisée dans la génération de la population initiale considère deux types de solutions :

**Type 1** : il consiste à choisir un ensemble de cases repérées par un indice h et leur attribuer une valeur booléenne.

**Type 2** : afin de générer des solutions diverses, le type 2 consiste à Complémenter les solutions de type 1, ce qui permettra d'avoir des solutions différentes de celles générées dans le type1.

***Ceci est réalisé en utilisant la méthode de diversification suivante :***

```
Pour   i=1  → sizepop Faire  // sizepop est la taille de la population P
Pour   j=1  → n  Faire       // n est le nombre de variables de la donnée SAT
P[i][j]=0                    // P initialement à zéro

Tant que   j ≤ sizepop  Faire
 Début
  h=1

    Tant que   j ≤ n  Faire
      | Début
      |    P[i][j] = 1 -  P[i][j];    // type 1
      |    j   = j + h;
      |    Fait
  h= h+1;
  i = i+1 ;

 Si  i ≤ sizepop alors
                |
                |        Pour   i=1  → n Faire
                |        P[i][j] = 1 -  P[i-1][j];   // type 2
                  Fait
 i = i+1 ;

 Fait
```

**Exemple :**

Pour n=10, sizepop=10, on aura la population suivante :

P= $\begin{Bmatrix} 1,1,1,1,1,1,1,1,1,1 \\ 1,0,1,0,1,0,1,0,1,0 \\ 1,0,0,1,0,0,1,0,0,1 \\ 1,0,0,0,1,0,0,0,1,0 \\ 1,0,0,0,0,1,0,0,0,0 \\ 0,0,0,0,0,0,0,0,0,0 \\ 0,1,0,1,0,1,0,1,0,1 \\ 0,1,1,0,1,1,0,1,1,0 \\ 0,1,1,1,0,1,1,1,0,1 \\ 0,1,1,1,1,0,1,1,1,1 \end{Bmatrix}$

### 4.2.2. LA FONCTION OBJECTIVE :

Une solution est caractérisée par une fonction objective permettant de mesurer sa qualité. Cette dernière est donnée par la somme des poids des clauses satisfaites.

### 4.2.3. LA GENERATION DE L'ENSEMBLE R :

La recherche dispersée constitue un moyen pour réaliser une interaction efficace entre la diversification et l'amélioration. A chaque étape, une liste de solutions diverses et de haute qualité est conservée pour construire une nouvelle génération.

La qualité d'une solution est mesurée par la fonction objective alors que la diversité est mesurée par la fonction distance suivante :

- **Fonction de dispersion :**

Soient X, Y deux solutions dans l'espace de recherche. La distance entre X et Y est calculée comme suit :

distance = 0 ;

**Pour** i=1 → n **Faire**
**Début**

**Si** X[i] ≠ Y[i] **alors** distance = distance +1 ;

**Fait**

**Exemple :**

Soient les deux solutions suivantes :

X = { 1 , 1 , 1 , 1 , 1, 1 ,1 , 1} n= 8 ;

Y = { 0 , 0 , 0 , 0 , 0, 0 ,0 , 0}

Distance (X, Y) = 1+1+1+1+1+1+1+1= 8

Après avoir présenté les deux fonctions utilisées pour évaluer les solutions de l'ensemble de recherche (le coût et la disparité), on va décrire dans ce qui suit une construction de l'ensemble R qui comprend les points références.

Les points références sont les points de haute qualité plus les points éloignés dans l'espace de recherche. Ils seront construits comme suit :

- Ordonner la population initiale suivant la fonction objective ;
- Sélectionner les B1 premières solutions de P pour pouvoir déduire les solutions de haute qualité ;
- Sélectionner les B2 solutions diverses en exécutant le processus suivant :

  Soient :

  - Sizepop la taille de la population P
  - B = B1+B2 la taille de l'ensemble R
  - Initialement R contient les B1 solutions de haute qualité

  Pour chaque solution X $\in$ (P-R),

  - Calculer la distance entre X est les Solutions de R
  - Sauvegarder la distance minimale

  Ordonner les solutions de (P-R) suivant la disparité (distance) en commençant par celles qui possèdent une grande distance ;

  Sélectionner les B2 premières solutions de l'ensemble (P-R) qui vont construire les solutions dispersées.

- Rajouter les B2 solutions sélectionnées à l'ensemble R pour le compléter.

### 4.2.4. LES SOLUTIONS ELITES :

Les points références vont participer à la construction d'un ensemble de solutions dites solutions élites. Avec un ensemble R de taille B on peut générer **4 types** de solutions définis comme suit :

**Type 1** : Il consiste à grouper les éléments de R deux à deux, puis, pour chaque paire de solutions appliquer l'opérateur de combinaison.

**Type 2**: Les solutions seront combinées trois à trois, en prenant un couple de solutions de type1 et on lui rajoute une meilleure solution qui n'apparaît pas dans l'élément courant.

**Type 3** : Les solutions seront combinées quatre à quatre. Pour chaque quadruplet obtenu, appliquer l'opérateur de combinaison

**Type 4** : Les i meilleurs points de R seront groupés, où i = 5 jusqu'à B éléments

### 4.2.5. LA CONSTRUCTION DES SOLUTIONS COMBINEES :

Définir un bon opérateur de combinaison est une étape délicate dans l'implémentation de la recherche dispersée. Cet opérateur dépend de la fonction objective utilisée ainsi que de la représentation binaire de la solution.

- **L'opérateur de combinaison :**

Soient X, Y deux solutions dans l'ensemble R «des points références». La solution combinée Z est construite comme suit :

**Pour** i=1 → n **Faire**
**Début**

**Si** X[i] = Y[i] **alors** Z[i] = Y[i] // selon la représentation
**Sinon**
donner une valeur pour Z[i] qui maximise la somme des poids des clauses satisfaites // en regardant la fonction objective
**Fait**

**Exemple :**

Soient :

Positif [i ] = la somme des poids des clauses où x[i] apparaît positif

Négatif [i ] = la somme des poids des clauses où x[i] apparaît négatif

On suppose la donnée SAT suivante :

$$\begin{cases} x_1 + \xi_2 + \xi_3 & w1=100 \\ \xi_1 + x_2 + X3 & w2=200 \\ X2 + X3 & w3=50 \end{cases}$$

Pour n=3, $C = \{ c_1, c_2, c_3 \}$, $X = \{ x_1, x_2, x_3 \}$

On aura les deux tableaux suivants :

Positif = {100, 250, 250}

Négatif = {200, 100, 100}

Soient les deux solutions suivantes :

$X = \{ 1 , 1 , 0 \}$

$Y = \{ 0 , 0 , 0 \}$

La solution combinée Z sera construite comme suit

$X[1] \neq Y[1]$ → positif [1] < Négatif [1] → Z[1]=0 ;

$X[2] \neq Y[2]$ → positif [2] > Négatif [2] → Z[2]=1 ;

X[1] =Y[1] → Z[3]=X[3]=0 ;

Par conséquent Z={ 0, 1, 0 }

### 4.2.6. L'AMELIORATION:

Afin d'approcher un optimum de haute qualité, la recherche dispersée s'appuie sur un deuxième principe qui consiste à améliorer la solution combinée en utilisant une méthode itérative. La méthode la plus utilisée est la recherche locale qui converge généralement vers un optimum local.

***La méthode proposée est donnée comme suit :***

**INITIALISATION :**

// S : la solution combinée
- Affecter à la solution combinée son coût.

// somme des poids des clauses satisfaites
// Itermax nombre maximum d'itérations

**ITERATION :**

**Pour** i=1 à itermax
**Faire**
- Générer le voisinage de S
- Inverser une variable dans les clauses non satisfaites
- Evaluer la configuration générée
- Si elle est meilleure remplacer la configuration initiale par cette dernière

**Fait**
Retourner la meilleure solution avec le meilleur coût
**FIN**

### 4.2.7. LE CRITERE D'ARRET:

Il nous faut encore définir une condition d'arrêt.

La recherche dispersée est stoppée après un certain nombre d'itérations ou lorsqu'on atteint la borne supérieure f* de la fonction objective.

### 4.2.8. L'ALGORITHME DE LA RECHERCHE DISPERSEE :

L'algorithme de la recherche dispersée est présenté comme suit :

**INITIALISATION :**

- Engendrer une population initiale P en utilisant une stratégie de diversification
- Affecter à chaque solution son coût.
- Ordonner la population suivant la fonction objective

// Itermax nombre maximum d'itérations

**ITERATION :**

**Pour** i=1 à itermax **Faire**

- Etablir une liste de solutions de taille B.

// le critère de sélection favorise les meilleures.

- L'ensemble R est de taille B = B1+B2 où
- Les B1 solutions sont de haute qualité
- Les B2 solutions sont diverses

Nouveau = Vrai

**Tant que** (Nouveau)
**Faire**

- Calculer la taille de l'ensemble des solutions élites Maxsubset ;
- Nouveau = faux ;

**Pour** i=1 à Maxsubset **Faire**

- Générer les différents types de solutions
- Appliquer l'opérateur de combinaison pour obtenir une nouvelle solution s
- Améliorer la solution s
- Evaluer la solution améliorée s*.

**Si** (coût (s*) > coût(B1 ème solution de R)) et s* n'apparaît Pas dans R **alors**
- Insérer s*
- Ecraser la mauvaise
- Nouveau = Vrai ;

**Sinon**

**Si** (disparité (s*) > disparité(B2ème solution de R)) et s* n'apparaît Pas dans R **alors**
- Insérer s*
- Ecraser la mauvaise
- Nouveau = Vrai ;

**Fait**

**Fait**

- régénérer la population, en remplaçant les solutions de manière à favoriser encore les meilleures.

**Fait**

## 4.3. ADAPTATION DE LA METHODE DE RECHERCHE LOCALE GUIDEE AU PROBLEME ETUDIE :

Pour pouvoir appliquer la recherche locale guidée on doit passer par les étapes suivantes :

- Générer une solution initiale ;
- Appliquer la recherche locale
- Avoir une fonction objective augmentée qui permet de pénaliser certaines solutions ;
- Définir le test d'arrêt

### 4.3.1. LA SOLUTION INITIALE:

La recherche locale guidée démarre d'une configuration initiale générée aléatoirement. La représentation binaire est celle utilisée dans l'algorithme de recherche dispersée (vecteur de n variables booléennes).

$X = \{ x_1, x_2, \ldots\ldots\ldots, x_n \}$ , $x_i$ est égal à 0 ou 1.

### 4.3.2. LA RECHERCHE LOCALE:

C'est une méthode itérative où chaque itération consiste à améliorer une configuration courante. L'amélioration consiste à effectuer des transformations successives(inverser un bit) sur la configuration initiale. Si le coût d'une configuration générée est meilleur que celui de la configuration courante, cette dernière sera remplacée par cette nouvelle configuration, sinon elle sera gardée. Ce processus est réitéré jusqu'à ce qu'aucune transformation ne réduise le coût de la configuration courante.

La méthode itérative utilisée est une recherche locale simple présentée comme suit :

**INITIALISATION :**

// S : la solution initiale
- Affecter à la solution son coût //nombre de clauses non satisfaites.

// Itermax nombre maximum d'itérations

**ITERATION :**

**Pour** i=1 à itermax
**Faire**
- Générer le voisinage de la solution S
- Inverser une variable dans les clauses non satisfaites
- Evaluer la configuration générée
- Si elle minimise le coût alors remplacer la configuration initiale par la configuration trouvée

**Fait**
Retourner l'optimum local et son coût
**FIN**

### 4.3.3. LA FONCTION OBJECTIVE AUGMENTEE:

La fonction objective permet de mesurer la qualité d'une solution. Pour notre problème MAX-SAT cette fonction est donnée par la somme des clauses non satisfaites. Donc elle consiste à minimiser le nombre de clauses non satisfaites. Afin d'échapper aux optima- locaux, la fonction coût a été modifiée en ajoutant une partie permettant d'augmenter le coût des solutions déjà visitées, ce qui impliquera une exploration plus diversifiée de l'espace de recherche.

***La fonction objective augmentée est donnée par la formule suivante :***

$$H(S) = G(S) + \lambda * \sum_{I=1}^{m} Ini(S)* P(i)$$

Où :

H : Fonction objective augmentée ;
G : Coût de la solution S = nombre de clauses non satisfaites;
S : Configuration courante ;
$In_{i(S)}$ : Indicateur ;
P(i) : Pénalité ;
m : Nombre de clauses ;
$\lambda$ : Facteur de pénalisation ;
$\lambda > 1$ : Recherche diversifiée ;
$\lambda < 1$ : Recherche intensifiée.

- **Principe de résolution :**

La recherche locale guidée consiste à pénaliser les clauses non satisfaites par le nombre de fois de leurs apparitions dans le but d'augmenter la fonction objective des solutions de mauvaise qualité. Cela est réalisé en utilisant une série de formules présentées dans ce qui suit :

*Formule 1 :*

$$Ini(S) = \begin{cases} 1 & \text{Si, pour la configuration S,la clause i est non satisfaite ;} \\ 0 & \text{Sinon.} \end{cases}$$

*Formule 2 :* **C(i) = poids de la clause i**

*Formule 3 :* **Utilité(i) = Ini(S) * C(i) / 1+P(i)**

Où :

**P(i)** : Est la pénalité ou encore un compteur de fréquence. Au début P(i) est initialisée à zéro, quand la clause i est non satisfaite P(i) est mis à jour.

**Ini(S)** : Indique si la clause i est satisfaite ou pas.

**Utilité(i)** : Permet de pénaliser les solutions déjà visitées.

Donc le problème revient à pénaliser les clauses non satisfaites ayant une utilité assez grande. Cette façon de faire permet de pénaliser les solutions déjà visitées, ce qui conduira vers une exploration plus diversifiée de l'ensemble de recherche et augmente les chances de trouver un optimum de haute qualité.

### 4.3.4. L'ALGORITHME DE LA RECHERCHE LOCALE GUIDEE :

Une version de l'algorithme de la recherche locale guidée est la suivante :

**INITIALISATION :**

- Générer aléatoirement une solution initiale S
- Affecter à la solution générée son coût // nombre de clause non satisfaite.

// G est la fonction objective standard
// Itermax nombre maximum d'itérations
// H = la fonction objective augmentée
// m = nombre de clauses

**Pour** i=1 à m **Faire** P(i)=0 ;

**ITERATION :**

**Pour** i=1 à Itermax **Faire**

- S := recherche locale(S, H)

// appel de l'algorithme de recherche locale avec les paramètres H et S
// la recherche locale retourne une solution S qui est le minimum local

**Pour chaque** ( clause non satisfaite fi)
**Faire**

- Calculer l'utilité de chaque clause non satisfaite
  **Utilité(fi) = Infi(S) * C(fi) / 1+P(fi)**
- Soit utilité- max égale à l'utilité maximale

**Si** (utilité(fi) = utilité- max) **alors** P(fi) = P(fi)+1

**Fait**
**Fait**
Retourner la meilleure solution avec le meilleur coût
**FIN**

### 4.4. CONCLUSION :

Dans ce chapitre, nous avons proposé deux algorithmes pour la résolution du problème **MAX-W-SAT** basés sur la recherche dispersée et la recherche locale guidée. Dans le prochain chapitre, ces méthodes seront testées à travers des expérimentations numériques pour mesurer leur performance.

# Chapitre 5 :

# Résultats expérimentaux

## 5.1. INTRODUCTION :

Afin de mesurer les performances des méta- heuristiques mises en œuvre, des tests ont été effectués sur quelques instances dont on sait apprécier les solutions optimales. Nos algorithmes ont été implémentés sous la plate forme suivante :

- **Machine** : Pentium 300 MMX, 32 MO de RAM.
- **Système d'exploitation** : Windows 95.
- **Langage de programmation** : C++ BUILDER.

***Dans nos expériences, nous avons désigné les différentes solutions par :***

- **SS_tcpu** : Temps d'exécution de la recherche dispersée.
- **GLS_tcpu** : Temps d'exécution de la recherche locale guidée.
- **SS** : Somme des poids des clauses satisfaites donnée par la recherche dispersée.
- **GLS** : somme des poids des clauses satisfaites donnée par la recherche locale guidée.

***La comparaison est basée sur deux paramètres :***

- la somme des poids des clauses satisfaites.
- Le temps CPU (temps de calcul).

## 5.2. RESULTATS :

L'efficacité d'une méta- heuristique dépend largement d'un bon ajustement de ses paramètres. Ceux-ci sont fixés suivant les expérimentations effectuées (Voir annexe) et les valeurs prises sont celles pour lesquelles il existe un compromis entre la qualité de la solution obtenue par la méta- heuristique et le temps d'exécution de l'algorithme.

Pour la recherche dispersée, la taille de la population (sizepop), la taille de l'ensemble R (B), nombre d'itérations de la recherche locale (Aiter), et Maxiter (nombre d'itérations) sont les paramètres à ajuster.

Pour la recherche locale guidée les paramètres de contrôle sont : Maxiter(nombre maximum d'itérations), nombre d'améliorations de la recherche locale (Aiter) et $\lambda$ le facteur de diversification.

Suite à des tests intensifs, les paramètres de ces deux méta- heuristiques sont fixés comme suit :

- **La recherche dispersée :** Sizepop=100, B1=5, B2= 5, Maxiter=3, et Aiter=100.
- **La recherche locale guidée** : Maxiter= 100, Aiter=1000, et $\lambda$=1.

***Instances MAX-SAT*** :

Dans nos expériences, nous avons considéré deux modèles :

**Modèle 1** : le nombre de clauses est égal à 800, le nombre de variables est égal à 100 ( Jnh201, Jnh202, Jnh203, Jnh205, Jnh207, Jnh208, Jnh209, Jnh210, Jnh211, Jnh212, Jnh214, Jnh215, Jnh216, Jnh217, jnh218) et la somme des poids des clauses est égale à 394238.

**Modèle 2** : le nombre de clauses est égal à 850, le nombre de variables est égal à 100 ( Jnh1, Jnh10, Jnh11, Jnh12, Jnh13, Jnh14, Jnh15, Jnh16, jnh17, Jnh18, Jnh19) et la somme des poids des clauses est égale à 420925.

Ces instances sont prises du site Internet «**Http://www.research.att.com/-mgcr/data/index.html**» dont les solutions optimales sont connues. Ce sont des problèmes difficiles de la classe Johnson.

Les résultats obtenus par la recherche dispersée et la recherche locale guidée sont les suivants :

**Modèle 1 :**

| Benchmark | Optimale | SS | SS_tcpu | GLS | GLS_tcpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 394238 | 6.44 | 394238 | 21.07 |
| Jnh202 | 394170 | 394022 | 2067.29 | 393818 | 3120.00 |
| Jnh203 | 394199 | 394135 | 2379.43 | 393206 | 3383.07 |
| Jnh205 | 394238 | 394238 | 1840.49 | 393903 | 1916.82 |
| Jnh207 | 394238 | 394045 | 1975.44 | 393585 | 1883.44 |
| Jnh208 | 394159 | 393934 | 2219.86 | 392936 | 2080.54 |
| Jnh209 | 394238 | 393971 | 1916.71 | 393812 | 1824.38 |
| Jnh210 | 394238 | 394238 | 293.56 | 394238 | 334.33 |
| Jnh211 | 393979 | 393372 | 2296.44 | 392921 | 2189.21 |
| Jnh212 | 394238 | 394029 | 2046.90 | 393993 | 1952.04 |
| Jnh214 | 394163 | 393377 | 2079.27 | 393341 | 1974.02 |
| Jnh215 | 394150 | 393451 | 2130.24 | 393872 | 2042.12 |
| Jnh216 | 394226 | 393399 | 2231.51 | 393391 | 2115.18 |
| Jnh217 | 394238 | 394238 | 650.46 | 394238 | 1606.38 |
| Jnh218 | 394238 | 394238 | 605.65 | 394238 | 914.47 |

**Figure 5.1. Les résultats trouvés par la recherche dispersée et la recherche locale guidée pour le modèle 1 :**

La recherche dispersée et la recherche locale guidée donnent de bons résultats mais les solutions trouvées par la recherche dispersée sont nettement meilleures que celles fournies par la recherche locale guidée. L'histogramme suivant le montre avec clarté :

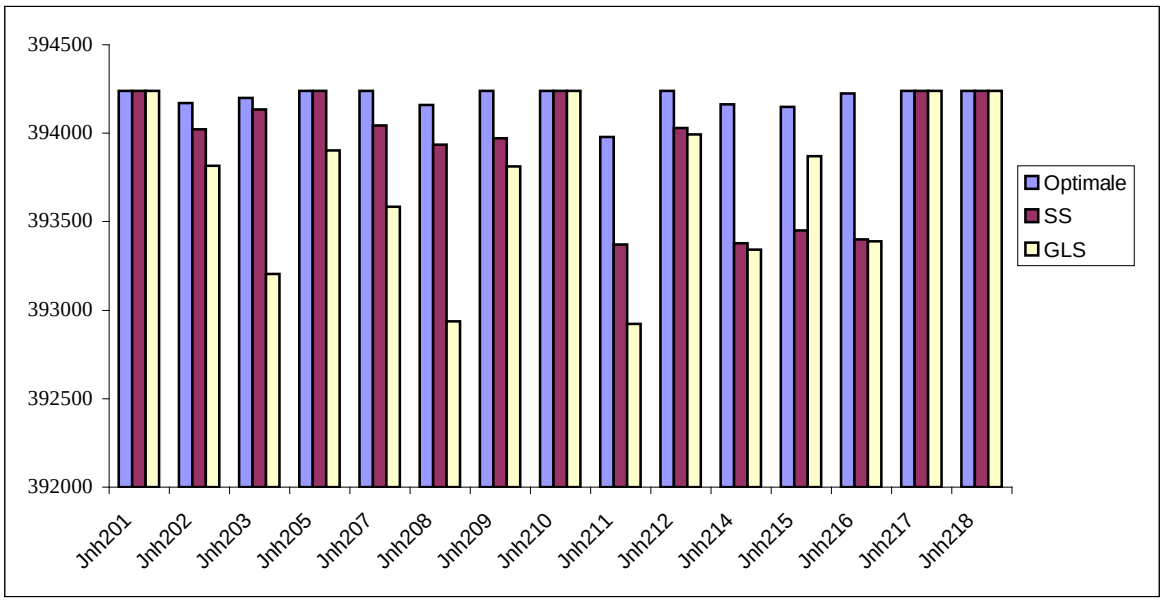


**Figure 5.2. Comparaison entre les solutions SS, les solutions GLS et les solutions optimales pour le modèle1.**

**Modèle 2 :**

La table ci-dessous, présente les résultats déterminés par la recherche dispersée et la recherche locale guidée pour le modèle 2 :

| Benchmark | Optimale | SS | SS_tcpu | GLS | GLS_tcpu |
|---|---|---|---|---|---|
| Jnh1 | 420925 | 420925 | 1159.88 | 420806 | 1782.72 |
| Jnh10 | 420840 | 420165 | 1860.38 | 419420 | 2432.27 |
| Jnh11 | 420753 | 420274 | 2404.32 | 420413 | 2300.99 |
| Jnh12 | 420925 | 420706 | 2407.00 | 420925 | 1220.29 |
| Jnh13 | 420816 | 420302 | 2593.60 | 419491 | 3249.01 |
| Jnh14 | 420824 | 420510 | 2555.30 | 420545 | 2738.84 |
| Jnh15 | 420719 | 419671 | 2680.10 | 419004 | 43800.2 |
| Jnh16 | 420919 | 420821 | 2380.91 | 420063 | 2294.55 |
| Jnh17 | 420925 | 420925 | 2080.00 | 420830 | 2226.41 |
| Jnh18 | 420795 | 420692 | 2576.90 | 419525 | 2467.08 |
| Jnh19 | 420759 | 420107 | 2521.19 | 419468 | 2428.60 |

**Figure 5.3. Les résultats trouvés par la recherche dispersée et la recherche locale guidée pour le modèle 2 :**

On remarque bien que la recherche dispersée et la recherche locale guidée trouvent l'optimum pour certaines instances et pour les autres, l'écart à l'optimum n'est pas très grand. L'histogramme qui suit permet de comparer ces algorithmes en matière de performance :

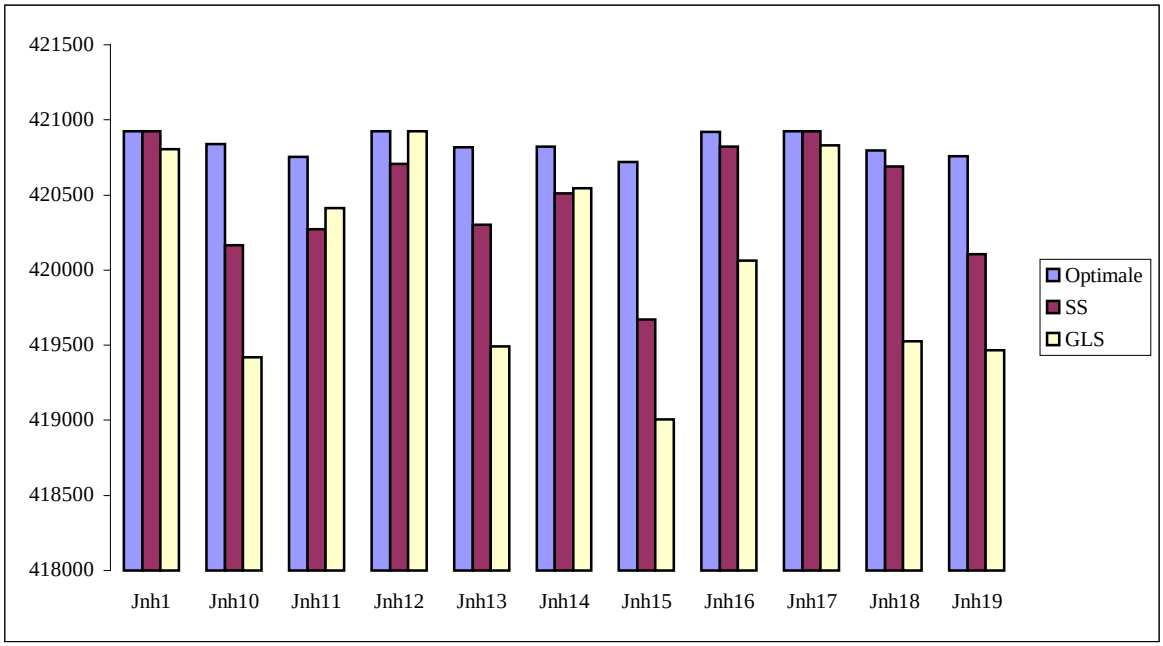


**Figure 5.4. Comparaison entre les solutions SS, les solutions GLS et les solutions optimales pour le modèle2.**

**Commentaire :**

Les tableaux ci- dessus montrent les résultats obtenus par la recherche dispersée et la recherche locale guidée ainsi que les solutions optimales.

Il est bien clair dans ces résultats que le degré de difficulté de la résolution du problème varie d'un problème à un autre. Les problèmes Jnh201 et Jnh210 possèdent un degré de difficulté plus petit et les deux algorithmes aboutissent en un temps réduit contrairement aux autres problèmes (Jnh208 et Jnh211) où les deux algorithmes prennent plus du temps pour trouver une solution de bonne qualité. Cela est dû essentiellement à la nature des problèmes, car Jnh201 (par exemple) ne comporte pas des clauses contradictoires contrairement aux problèmes Jnh208 et Jnh211. Ces derniers comportent des clauses qui ont plus de chances d'être contradictoires, c'est –à- dire, qui ne pourront jamais être satisfaites en même temps par la même affectation.

Ce qu'il faut noter aussi en ce qui concerne les solutions trouvées, c'est l'écart entre ces dernières et la solution optimale qui est négligeable dans certaines instances, ce qui prouve la puissance de la recherche dispersée et la recherche locale guidée dans la résolution du problème MAX-W-SAT.

## 5.3. INTERPRETATION DES TESTS EFFECTUES :

Dans nos tests, nous avons utilisé des données MAX-SAT dont le degré de difficulté est assez important ceci est dû essentiellement à la nature de ces problèmes et le nombre important d'optima-locaux dans l'espace de solutions. Ces instances se caractérisent par un nombre de clauses compris entre 800 et 900 et un nombre de variables égal à 100.

Pour la recherche dispersée, nous remarquons d'après les tests effectués que la taille de la population et la taille de l'ensemble R sont deux paramètres très importants pour la robustesse de la recherche dispersée. Ils fournissent un degré de diversification significatif à la recherche.

En effet, lorsqu'on augmente B et sizepop, la recherche devient efficace mais le temps CPU croit. Pour cela le réglage de ces paramètres est une opération importante dans la recherche et le bon ajustement conduira nécessairement à des résultats satisfaisants.

De même la recherche locale guidée procure généralement une solution de bonne qualité. Toutefois, cette méthode représente quelques inconvénients comme le temps de calcul qui est excessif pour avoir une solution de haute qualité. Le paramètre Maxiter utilisé pour la recherche locale guidée joue un rôle dans la qualité des solutions trouvées. Si celui-ci est important la recherche devient efficace mais perd en rapidité d'exécution, et à l'inverse il diminue l'efficacité mais accélère la recherche.

Nous constatons que la recherche dispersée donne d'excellents résultats. Elle atteint l'optimum plus rapidement que la recherche locale guidée. Les temps sont compris entre 0.50 et 3000 secondes (obtention d'une solution optimale).

En effet, la recherche dispersée tient sa supériorité du fait de l'intégration de plusieurs techniques de diversification et d'amélioration. Ainsi le générateur de la population augmente la probabilité de trouver un optimum de bonne qualité rapidement (Jnh201), alors que la recherche locale guidée utilise sa fonction objective augmentée pour éviter l'optimum local trouvé par la recherche locale et essaie de retomber sur un optimum global après un certain nombre d'itérations.

# Chapitre 6 :

# Approches hybrides

## 6.1. INTRODUCTION :

Dans le but de valider les différents algorithmes développés, des tests ont été effectués sur différents problèmes **MAX-W-SAT**.

Afin que la qualité des solutions trouvées soit plus significative, nous utilisons les algorithmes de comparaison suivants:

- La procédure GSAT avec marche aléatoire.
- La recherche tabou avec bruit aléatoire.
- La procédure GRASP.
- La recherche dispersée avec marche aléatoire.
- La recherche dispersée avec bruit aléatoire.
- La recherche locale guidée avec marche aléatoire.
- La recherche locale guidée avec bruit aléatoire.

## 6.2. La procédure GSAT avec marche aléatoire:

La procédure **GSAT** développée par Selman **[SML 92]** permet d'obtenir une instanciation en commençant par une assignation aléatoire puis procède à changer l'assignation de la variable qui mène à la plus grande baisse dans le nombre total des clauses non satisfaites. C'est une heuristique très puissante, elle permet d'obtenir des résultats très satisfaisants. Afin d'améliorer ses performances, une stratégie de **«marche aléatoire»** a été ajoutée à l'algorithme pour modifier la règle de choix du voisin, et ça, dans l'algorithme **GSAT avec marche aléatoire**.

***Cet algorithme se présente de la manière suivante [BP 96]*** *:*

```
Pour i : = 1 à MAX-ESS
faire
 X := assignation aléatoire
 Pour j : = 1 à MAX-ITER
 faire
      Si Randomnbr() < p alors
          Var := variable quelconque ∈ à une clause non satisfaite
      Sinon  var := la variable dont ∆f est le plus grand
      FLIP(var)

 Fait
Fait
```

Où :

Randomnbr () : est une procédure qui génère un nombre aléatoire entre 0 et 1.
FLIP(var) : inverse la variable var (var : = 1- var).
∆f : coût d'une configuration.
MAX-ESS : nombre d'essais.
MAX-ITER : nombre d'itérations.

A chaque itération une variable est choisie par deux critères différents :

- La variable est choisie aléatoirement d'une clause non satisfaite, avec une probabilité p.

- En se basant sur la fonction objective f, choisir la variable avec le plus grand ∆f.

### 6.3. La recherche tabou avec bruit aléatoire[Dri 99]:

Le choix de la configuration voisine peut avoir un effet considérable aussi bien sur le temps de recherche que sur la qualité de la solution finale. Dans l'algorithme de la recherche tabou, la solution voisine est obtenue en choisissant la meilleure transition non tabou. Une amélioration possible serait de choisir cette solution à partir d'un algorithme de recherche tabou avec bruit aléatoire. Il introduit un bruit représenté par une probabilité p.

Avec une probabilité p, la configuration voisine est choisie aléatoirement. Avec une probabilité 1-p, la solution adjacente est construite en se basant sur la règle de choix du voisin de la recherche tabou.

Ci-dessous une version de l'algorithme itératif de la recherche tabou avec bruit aléatoire :

***L'algorithme tabou avec le bruit aléatoire :***

**Etape 1 : Initialisation**

- Choisir une solution initiale générée aléatoirement s = s0 ;
- nbiter = 0 /* itération courante */ ;
- bestiter = 0 /* itération ayant donné la meilleure solution */ ;
- bestsol = s0 /* meilleure solution */ ;
- LT = $\Phi$ /* liste tabou initialisée à vide */ ;
- P /* est le bruit */

**Etape 2 : Processus itératif**

***Tant que*** (nbiter- bestiter < nbmax)
***Faire***

***Si (RandomNumber () < p ) alors /*randomNumber fournit un nombre */***
***/* aléatoire compris entre 0 et 1 */***

- nbiter = nbiter + 1 ;
- Générer un ensemble V* $\subseteq$ N(s) de solutions voisines de s ;
- Choisir la meilleure solution s'$\in$V*telle que s'$\notin$LT ou satisfait le critère d'aspiration ;
- Mettre à jour la liste tabou ;

***Sinon s= n'importe quel voisin de la solution courante***

- s = s' ;
- ***si*** f(s) < f(bestsol) ***alors*** bestsol = s' ; bestiter = nbiter ;

***Fait***
***Fin***

## 6.4. La procédure GRASP :

La procédure GRASP (Greedy Randomized Adaptive Search Procedure) combine deux phases : une phase de construction aléatoire (Glouton) pour construire une solution initiale et une phase d'amélioration locale. Le processus de construction et d'amélioration est réitéré pour un certain nombre d'itérations.

Dans la phase de construction, les transitions possibles sont ordonnées dans une liste candidate. La solution est construite aléatoirement à partir de la liste candidate, en choisissant pour une variable à assigner une valeur de vérité.

***La construction***:

Pour k=1 → n
**Faire**
- Créer la liste candidate
- Sélectionner aléatoirement une transition
- Affecter une valeur de vérité pour la variable
- Appliquer la fonction Glouton.

**Fait**

Afin d'améliorer la solution construite par la première procédure, GRASP fait appel à une recherche locale après chaque construction.

***La procédure GRASP est présentée comme suit [BP 98]*** *:*

**Etape 1 : Initialisation**
nbiter = 0 /* itération courante */ ;
nbmax /* nombre maximum d'itérations */ ;

**Etape 2 : Processus itératif**

**Tant que** (nbiter < nbmax)

**Faire**
- nbiter = nbiter + 1;
- Construction() ;
- Amélioration();

**Fait**

**FIN**

## 6.5. Approches hybrides :

Dans cette section, nous proposons quatre hybridations d'algorithmes basées sur l'introduction des stratégies de bruit et de marche aléatoires dans la recherche dispersée et la recherche locale guidée.

L'hybridation des méta- heuristiques en question avec les stratégies de bruit et de marche aléatoires se présente comme suit :

### 6.5.1. La recherche dispersée avec marche aléatoire :

Dans cette hybridation, le passage à la prochaine solution se fait selon les deux critères suivants :

- La variable est tirée aléatoirement d'une clause non satisfaite, avec une probabilité p.
- La configuration voisine est obtenue en appliquant la recherche dispersée, avec une probabilité 1-p.

***Cet algorithme est donné comme suit :***

```
Pour  i : = 1  à   MAX-ESS
Faire
|
|     X :=  meilleure assignation dans la population
|
|    Pour   j : = 1  à   MAX-ITER
|    Faire
|    |   Si  Randomnbr() < p    alors
|    |   Var :=  variable quelconque appartenant à une clause non satisfaite
|    |   FLIP(var)
|    |
|    |    Sinon
|    |     Recherche dispersée() ;
|      Fait
Fait
```

Où :

Randomnbr() est une procédure qui génère un nombre aléatoire entre 0 et 1.
MAX-ESS : nombre d'essais.
MAX-ITER : nombre d'itérations.
FLIP(var) : inverse la variable var.

### 6.5.2. La recherche dispersée avec bruit aléatoire :

Une version de l'algorithme de recherche dispersée avec bruit aléatoire est donnée comme suit :

```
Pour i : = 1 à MAX-ESS
Faire

        X := meilleure assignation dans la population

        Pour j : = 1 à MAX-ITER
        Faire
            Si Randomnbr() < p  alors
                                Var := variable quelconque
                                FLIP(var)

             Sinon
                        Recherche dispersée() ;
         Fait
Fait
```

Où :

Randomnbr() est une procédure qui génère un nombre aléatoire entre 0 et 1.
MAX-ESS : nombre d'essais.
MAX-ITER : nombre d'itérations.
FLIP(var) : inverse la variable var.

Cette approche possède un paramètre très important qui est le bruit. En augmentant la valeur de p on accélère la recherche mais on diminue de son efficacité, et en diminuant on ralentit la recherche mais on gagne en efficacité. La valeur optimale de p est positionnée empiriquement entre 0.5 et 0.6.

### 6.5.3. La recherche locale guidée avec marche aléatoire :

Dans sa forme standard, la recherche locale guidée est guidée par une fonction objective augmentée, et le critère de base est d'accepter un voisin s'il satisfait plus de clauses. A quelle ampleur les résultats seront affectés si nous changeons la règle de choix du voisin, en rajoutant des stratégies de bruit ou de marche aléatoires.

L'algorithme de la recherche locale guidée simple, a été modifié dans le but de choisir un voisin suivant des critères différents, et ça, dans les algorithmes suivants :

***L'algorithme de recherche locale guidée avec marche aléatoire se présente de la manière suivante :***

```
Pour  i : = 1  à   MAX-ESS
Faire
        X := assignation aléatoire

        Pour  j : = 1  à   MAX-ITER
        Faire
            Si  Randomnbr() < p    alors
          Var :=  variable quelconque appartenant à une clause non satisfaite
            FLIP(var)

            Sinon
              Recherche locale guidée() ;.
        Fait
 Fait
```

Randomnbr() est une procédure qui génère un nombre aléatoire entre 0 et 1.
MAX-ESS : nombre d'essais.
MAX-ITER : nombre d'itérations.
FLIP(var) : inverse la variable var.

Cette hybridation consiste à choisir la variable à inverser par les deux critères donnés comme suit :

- Avec une probabilité p, la variable est choisie aléatoirement à partir d'une clause non satisfaite.

- Avec une probabilité 1-p, appliquer la recherche locale guidée.

### 6.5.4. La recherche locale guidée avec bruit aléatoire:

L'introduction de la stratégie de bruit aléatoire dans la recherche locale guidée a donné l'algorithme suivant :

```
Pour  i : = 1  à   MAX-ESS
Faire
        X := assignation aléatoire ;
        Pour  j : = 1  à   MAX-ITER
            Faire
                    Si  Randomnbr() < p    alors
                            Var :=  variable quelconque.
                             FLIP(var)

                    Sinon   Recherche locale guidée().
             Fait
 Fait
```

Où :

Randomnbr() est une procédure qui génère un nombre aléatoire entre 0 et 1.
MAX-ESS : nombre d'essais.
MAX-ITER : nombre d'itérations.
FLIP(var) : inverse la variable var.

Dans la recherche locale guidée avec bruit aléatoire, la solution voisine est obtenue selon les mêmes critères cités dans la recherche dispersée avec bruit aléatoire.

La configuration adjacente est choisie aléatoirement, avec une probabilité p, ou en appliquant la recherche locale guidée, avec une probabilité 1-p.

## 6.6. CONCLUSION :

Nous venons de voir qu'ils existent différents modes d'exploration de l'espace de solutions. Les méta- heuristiques se différencient selon la manière dont elles font varier l'intensité de l'exploration au cours de recherche.

Dans la méthode GSAT, par exemple, le mécanisme de base repose sur une règle dite agressive consistant à choisir systématiquement le meilleur voisin. Une manière efficace d'améliorer GSAT de base est l'option de marche aléatoire qui consiste à alterner les mouvements de base et des mouvements totalement aléatoires.

Une autre stratégie pour explorer consiste à mémoriser au cours de la recherche des caractéristiques des régions visitées et à introduire un mécanisme permettant de s'éloigner de ces zones. C'est ce que fait la méthode tabou avec la liste tabou.

Les stratégies de bruit et de marche aléatoires ont été développées pour la recherche dispersée et la recherche locale guidée dans ce chapitre. Seuls les expérimentations intensives peuvent nous renseigner si de telles stratégies augmentent la performance de ces méta- heuristiques. Le chapitre suivant fait l'objet en effet de cette préoccupation.

# Chapitre 7 :

# Validation des approches développées

## 7.1. INTRODUCTION :

Comme pour toute méta- heuristique, deux types d'expérimentations doivent être menées : la première catégorie concerne le réglage des paramètres des algorithmes et la seconde teste la performance des approches considérées.

## 7.2. REGLAGE DES PARAMETRES :

Avant de se lancer dans l'étude comparative entre les différentes approches présentées, le réglage de certains paramètres s'impose. Les résultats empiriques sont montrés dans les tables suivantes :

| Symbole | Valeur |
|---|---|
| Tabou tenure | 7 |
| Maxiter | 10000 |
| P | 0.5 |

**Table 1 : Paramètres de la recherche tabou avec bruit aléatoire**

| Symbole | Valeur |
|---|---|
| N_essais | 10 |
| Maxiter | 1000 |
| P | 0.5 |

**Table 2 : Paramètres du GSAT avec marche aléatoire**

| Symbole | Valeur |
|---|---|
| Aiter | 1000 |
| Maxiter | 1000 |
| P | 0.5 |

**Table 3 : Paramètres de la recherche locale guidée avec marche aléatoire**

| Symbole | Valeur |
|---|---|
| Aiter | 1000 |
| Maxiter | 1000 |
| P | 0.5 |

**Table 4 : Paramètres de la recherche locale guidée avec bruit aléatoire**

| Symbole | Valeur |
|---|---|
| Sizepop | 100 |
| B1 | 5 |
| B2 | 5 |
| Aiter | 10 |
| Maxiter | 10000 |
| P | 0.5 |

**Table 5 : Paramètres de la recherche dispersée avec marche aléatoire**

| Symbole | Valeur |
|---|---|
| Sizepop | 100 |
| B1 | 5 |
| B2 | 5 |
| Aiter | 10 |
| Maxiter | 1000 |
| P | 0.5 |

**Table 6 : Paramètres de la recherche dispersée avec bruit aléatoire**

## 7.3. RESULTATS :

Les expériences numériques sur des instances **MAX-W-SAT** difficiles ont fourni les résultats résumés dans la série des tables montrées sur les figures apparaissant ultérieurement.

❖ Les résultats trouvés par la procédure **GSAT** avec marche aléatoire et la recherche tabou avec bruit aléatoire sont exposés dans la table suivante :

| Benchmark | Optimale | GsatW | GS_tcpu | TabouW | Tab_tcpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 394238 | 15.24 | 394238 | 143.45 |
| Jnh202 | 394170 | 393155 | 240.36 | 392837 | 242.58 |
| Jnh203 | 394199 | 392804 | 274.01 | 392485 | 275.26 |
| Jnh205 | 394238 | 393959 | 233.79 | 393752 | 235.07 |
| Jnh207 | 394238 | 393049 | 243.71 | 392708 | 230.28 |
| Jnh208 | 394159 | 392296 | 255.82 | 392380 | 254.41 |
| Jnh209 | 394238 | 392933 | 223.52 | 392992 | 225.30 |
| Jnh210 | 394238 | 393864 | 229.41 | 394238 | 222.46 |

**Figure 7.1. Les solutions trouvées par la recherche tabou avec bruit aléatoire et GSAT avec marche aléatoire.**

La procédure **GSAT** avec marche aléatoire et la recherche tabou avec bruit aléatoire donnent des solutions assez bonnes. Dans ces résultats, on remarque que le problème Jnh201 possède un degré de difficulté plus petit et les deux algorithmes aboutissent en un temps réduit. Les solutions fournies par la recherche dispersée et la recherche locale guidée sont les meilleures.

L'histogramme suivant permet de comparer les algorithmes testés jusqu'à présent en matière de performance.

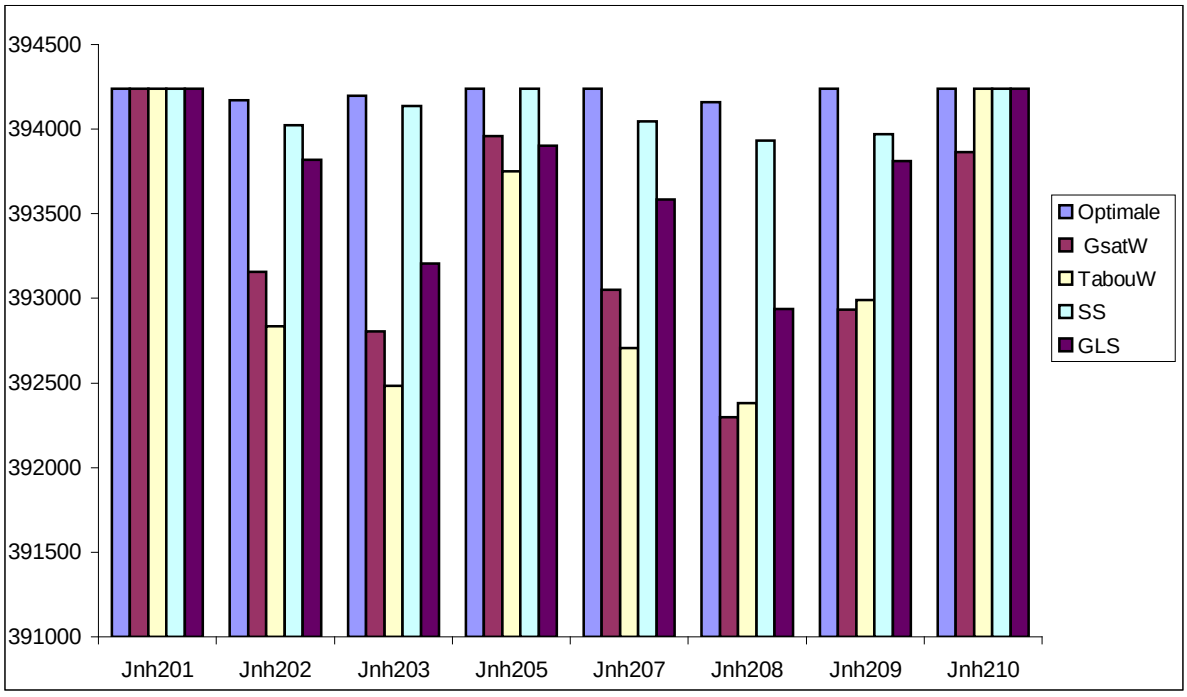


**Figure 7.2. Comparaison entre la recherche dispersée, la recherche locale guidée, la recherche tabou avec bruit aléatoire et GSAT avec marche aléatoire.**

D'après les tables représentant les tests effectués sur quelques instances **MAX-W-SAT** difficiles, on constate que l'algorithme de recherche dispersée s'est avéré très efficace en qualité de solutions par rapport aux autres méthodes développées.

❖ Sur la table suivante, nous reportons les résultats déterminés par la procédure **GRASP** :

| Benchmark | Optimale | GRASP | TCPU | SS | SS_tcpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 394154 | 310.4 | 394238 | 6.44 |
| Jnh202 | 394170 | 393680 | 312.2 | 394022 | 2067.29 |
| Jnh203 | 394199 | 393446 | 351.2 | 394135 | 2379.43 |
| Jnh205 | 394238 | 393890 | 327.8 | 394238 | 1840.49 |
| Jnh207 | 394238 | 394030 | 304.7 | 394045 | 1975.44 |
| Jnh208 | 394159 | 393893 | 355.2 | 393934 | 2219.86 |
| Jnh209 | 394238 | 393959 | 339.0 | 393971 | 1916.71 |
| Jnh210 | 394238 | 393950 | 318.5 | 394238 | 293.56 |

**Figure 7.3. Les solutions trouvées par GRASP [BP 98].**

A partir de la table ci-dessus, on remarque que la recherche dispersée trouve l'optimum global pour certaines instances, et pour les autres l'écart à l'optimum n'est pas très grand. Les résultats respectifs sont meilleurs que ceux fournis par la procédure **GRASP**, concernant la qualité des solutions. L'histogramme suivant le montre avec clarté.

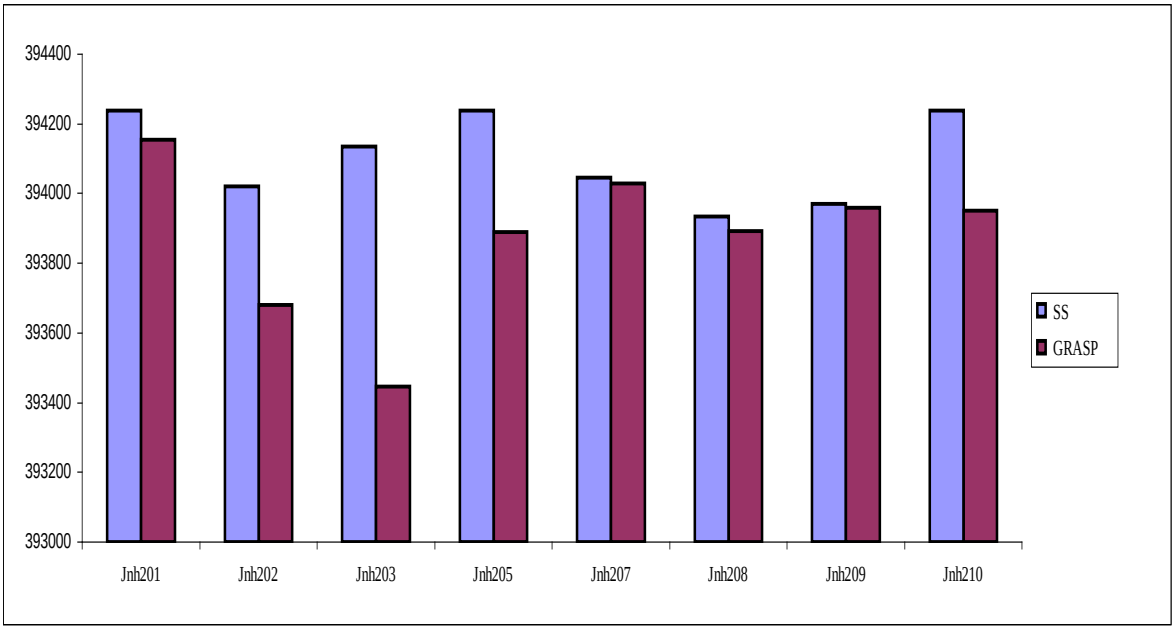


**Figure 7.4. Comparaison entre la recherche dispersée et la procédure GRASP.**

❖ La figure 7.5 montre la comparaison entre la recherche dispersée avec marche aléatoire et la recherche dispersée avec bruit aléatoire. L'histogramme qui suit schématise les résultats de la comparaison :

| Benchmark | Optimale | SSW | Swtcpu | SSnoise | SSN_tcpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 394238 | 28.48 | 394238 | 25.12 |
| Jnh202 | 394170 | 392788 | 256.05 | 394044 | 1086.39 |
| Jnh203 | 394199 | 392795 | 293.14 | 393498 | 1198.62 |
| Jnh205 | 394238 | 393300 | 250.23 | 393903 | 338.74 |
| Jnh207 | 394238 | 393262 | 245.67 | 393745 | 758.63 |
| Jnh208 | 394159 | 392608 | 272.03 | 393822 | 384.74 |
| Jnh209 | 394238 | 392819 | 237.29 | 393459 | 919.42 |
| Jnh210 | 394238 | 393951 | 240.91 | 394238 | 395.43 |
| Jnh211 | 393979 | 391993 | 284.32 | 392986 | 526.55 |
| Jnh212 | 394238 | 392710 | 251.17 | 394074 | 610.96 |
| Jnh214 | 394163 | 392825 | 256.98 | 393077 | 608.92 |
| Jnh215 | 394150 | 393188 | 264.44 | 393951 | 478.22 |
| Jnh216 | 394226 | 392661 | 277.07 | 393882 | 860.10 |
| Jnh217 | 394238 | 394238 | 165.42 | 394197 | 499.25 |
| Jnh218 | 394238 | 393763 | 251.90 | 394238 | 285.55 |
| Jnh1 | 420925 | 420609 | 234 | 420806 | 658.16 |
| Jnh10 | 420840 | 418550 | 316.04 | 419603 | 674.21 |
| Jnh11 | 420753 | 419341 | 296.49 | 419658 | 481.34 |
| Jnh12 | 420925 | 418947 | 307.79 | 420701 | 484.94 |
| Jnh13 | 420816 | 418447 | 320.62 | 418891 | 598.83 |
| Jnh14 | 420824 | 419145 | 313.44 | 419358 | 589.05 |
| Jnh15 | 420719 | 419032 | 322.52 | 419800 | 848.66 |
| Jnh16 | 420919 | 419501 | 291.38 | 420746 | 914.16 |
| Jnh17 | 420925 | 419959 | 289.65 | 420925 | 225.89 |
| Jnh18 | 420795 | 418710 | 317.79 | 419746 | 680.99 |
| Jnh19 | 420759 | 418765 | 310.52 | 419746 | 567.79 |

**Figure 7.5. Les solutions trouvées par la recherche dispersée avec marche aléatoire et la recherche dispersée avec bruit aléatoire.**

Pour la plupart de jeux d'essai :

- Soit la qualité de la solution est dans les deux cas comparable.

- Soit la recherche dispersée avec bruit aléatoire fournit des résultats meilleurs que ceux trouvés par la recherche dispersée avec marche aléatoire.

L'avantage principal de ces méthodes réside dans leur grande simplicité et leur rapidité. Mais les solutions produites sont souvent de qualité médiocre.

Quant à la recherche dispersée, ses performances en qualité des solutions obtenues sont très bonnes.

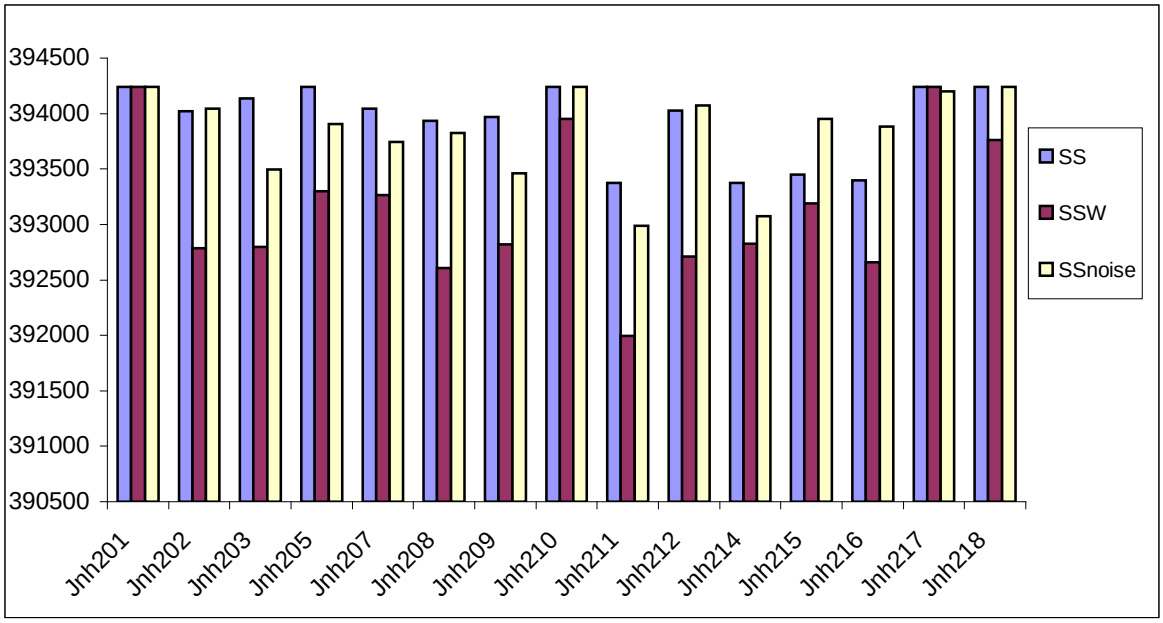


**Figure 7.6. Comparaison entre la recherche dispersée, la recherche dispersée avec marche aléatoire et la recherche dispersée avec bruit aléatoire, pour modèle 1.**

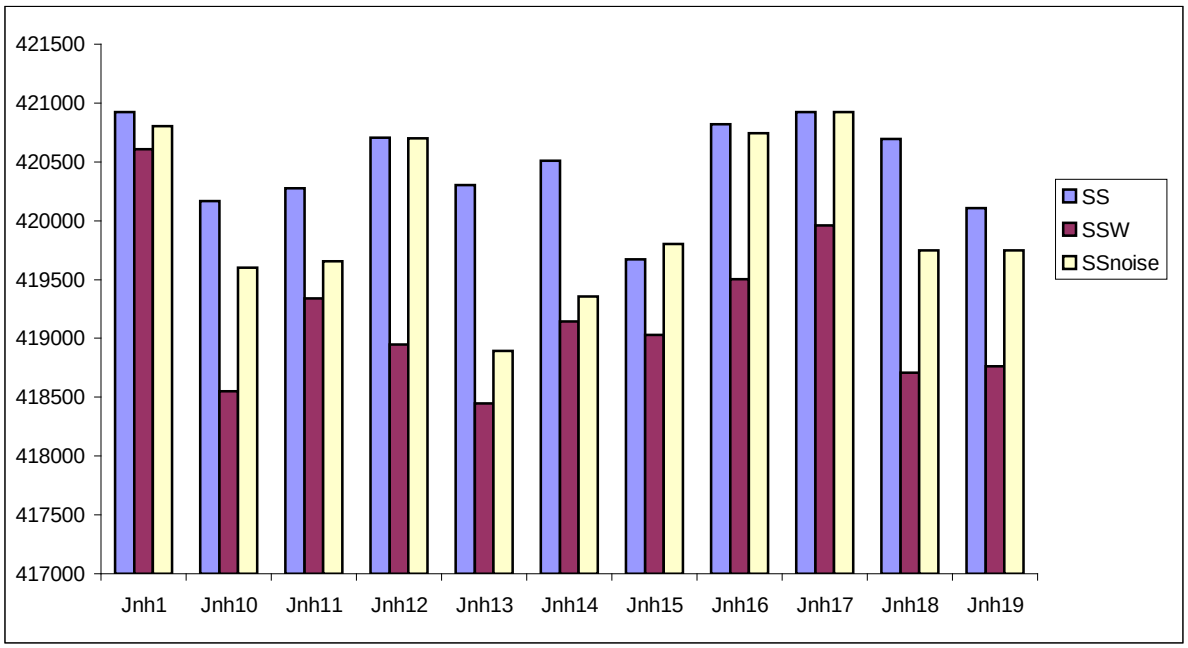


**Figure 7.7. Comparaison entre la recherche dispersée, la recherche dispersée avec marche aléatoire et la recherche dispersée avec bruit aléatoire, pour modèle 2.**

La recherche dispersée avec bruit aléatoire donne des solutions qui sont en général meilleures que celles trouvées par la recherche dispersée avec marche aléatoire, voire parfois excellentes (l'optimum global). Ceci en ce qui concerne la qualité de la solution obtenue et le temps de calcul qui lui est approprié. La

comparaison entre la recherche dispersée avec marche aléatoire, la recherche dispersée avec bruit aléatoire et la recherche dispersée montre des résultats meilleurs pour la recherche dispersée en ce qui concerne la qualité de solutions.

- Les algorithmes développés ont été testés sur un certain nombre de problèmes **MAX-W-SAT** difficiles. Les résultats trouvés par la recherche locale guidée avec bruit aléatoire et la recherche locale guidée avec marche aléatoire sont donnés comme suit :

| Benchmark | Optimale | GLSW | GW_cpu | GLSN | GN_cpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 394238 | 78.60 | 394238 | 40.14 |
| Jnh202 | 394170 | 393157 | 215.03 | 393197 | 175.97 |
| Jnh203 | 394199 | 392538 | 240.03 | 392463 | 227.34 |
| Jnh205 | 394238 | 393816 | 135.63 | 393563 | 248.27 |
| Jnh207 | 394238 | 393593 | 244.07 | 393296 | 301.62 |
| Jnh208 | 394159 | 392338 | 292.66 | 392706 | 188.44 |
| Jnh209 | 394238 | 393141 | 145.37 | 393524 | 145.82 |
| Jnh210 | 394238 | 394238 | 15.72 | 394238 | 86.30 |

**Figure 7.8. Les solutions trouvées par la recherche locale guidée avec bruit aléatoire et la recherche locale guidée avec marche aléatoire.**

Les résultats expérimentaux obtenus montrent que la qualité des solutions trouvées par la recherche locale guidée avec bruit aléatoire est comparable à celle des solutions obtenues par la recherche locale guidée avec marche aléatoire.

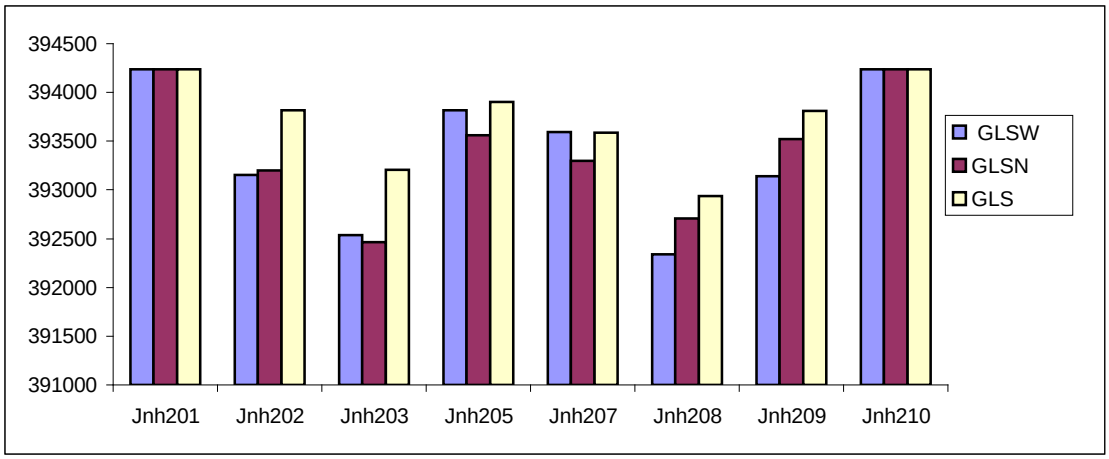


**Figure 7.9. Comparaison entre la recherche locale guidée, la recherche Locale guidée avec marche aléatoire et la recherche locale Guidée avec bruit aléatoire.**

D'après l'histogramme donné ci-dessus, on voit bien que les stratégies de bruit et de marche aléatoires n'ont pas incité à l'augmentation de la performance de la recherche locale guidée.

## 7.4. CONCLUSION:

Dans ce chapitre nous avons présenté les différents résultats obtenus des expérimentations numériques des approches développées tout au long de notre projet.

Nous avons vu que parmi les algorithmes développés, la recherche dispersée et la recherche locale guidée ont prouvé leur performance grâce à l'inclusion de plusieurs techniques de diversification et d'amélioration. Les résultats respectifs sont acceptables et meilleurs que ceux de GSAT avec marche, recherche tabou avec bruit et GRASP. De plus, si nous avions à choisir un algorithme parmi ceux développés, nous suggérons la recherche dispersée et la recherche locale guidée sans marche et sans bruit car l'inclusion de ces stratégies n'a pas été sensible à la robustesse de ces méthodes.

# Conclusion générale et perspectives

Les problèmes d'optimisation combinatoire auxquels nous nous sommes intéressés sont réputés difficiles : consistant en la recherche d'une solution de meilleure qualité dans un ensemble fini mais de très grande cardinalité donc non énumérable. A leur appartenance à la classe des problèmes NP- Complets, s'ajoute également le fait que souvent n'ont été résolus que des problèmes de petite taille.

Ce travail concerne le problème de la satisfiabilité maximale pondérée(**MAX-W-SAT** en abrégé) qui est le problème central de la NP- Complétude. Il présente un double intérêt : d'une part, il apporte une réponse à de nombreux problèmes de décision se posant souvent, de façon cruciale, dans des domaines très divers. D'autre part, il nous a permis de montrer comment construire des algorithmes performants pour des problèmes très ardus.

En effet, notre démarche a d'abord consisté à appliquer au traitement de ce problème deux méta- heuristiques à savoir la recherche dispersée et la recherche locale guidée.

Nous avons ensuite conçu, étudié et expérimenté des approches hybrides basées sur la recherche tabou, la procédure GSAT, la recherche dispersée, et la recherche locale guidée, dans le but de prouver la performance des méta- heuristiques dans la résolution du problème **MAX-W-SAT.**

Dans la première partie, nous avons proposé deux algorithmes de résolution de **MAX-W-SAT** basés sur la recherche dispersée et la recherche locale guidée.

La recherche dispersée est une méta- heuristique à stratégies d'évolution, elle intègre les idées majeures des méthodes évolutionnistes (population de solutions, recombinaison de solutions), tout en possédant les avantages suivants:

- Elle dispose d'heuristique efficace pour aider à la génération d'une bonne population de départ ;

- Elle autorise des synergies avec d'autres méthodes (algorithmes itératifs), qui peuvent être intégrées dans la phase d'amélioration ;

- Elle se rapproche d'avantage de la formulation du problème (dans la combinaison).

La recherche locale guidée est une méta- heuristique basée pénalité. Elle intègre les idées majeures des algorithmes itératifs de recherche locale(solution initiale, améliorations successives), tout en possédant les avantages suivants :

- Elle augmente la fonction objective, dans le but d'éviter l'optimum local.

- Elle se rapproche d'avantage de la formulation du problème (dans la pénalisation des solutions de mauvaise qualité).

Les expérimentations reportées au chapitre 5, nous ont permis de clarifier ce que l'on peut attendre de ces deux méta- heuristiques, et de confirmer l'existence des méthodes capables de produire d'excellents résultats.

Ce qu'il faut noter aussi en ce qui concerne les solutions trouvées, c'est l'écart entre ces dernières et la solution optimale qui est négligeable dans certaines instances, ce qui prouve la puissance de la recherche dispersée et la recherche locale guidée dans la résolution du problème **MAX-W-SAT.**

Dans la deuxième partie, nous avons fait une étude comparative des approches développées. Elle s'appuie sur l'étude réalisée à propos des performances des algorithmes de recherche dispersée et de recherche locale guidée ; nous avons été amenés à dégager des notions essentielles comme celles de diversification et d'amélioration (Stratégies utilisées pour trouver de bonnes solutions), et à mettre en évidence le rôle joué par la fonction d'évaluation augmentée de la recherche locale guidée( fonction utilisée afin d'échapper à l'optimum local).

Parmi les approches développées, la recherche dispersée et la recherche locale guidée ont prouvé leur performance grâce à l'inclusion de plusieurs techniques de diversification et d'intensification. Les résultats respectifs sont acceptables et meilleurs que ceux de GSAT avec marche aléatoire, recherche tabou avec bruit aléatoire, GRASP, recherche dispersée avec bruit aléatoire, recherche dispersée avec marche aléatoire, recherche locale guidée avec bruit aléatoire, et recherche locale guidée avec marche aléatoire.

# Perspectives :

Nos recherches vont naturellement se prolonger, dans les deux voies suivantes :

- Prolonger l'étude des méta -heuristiques proposées. En général, il reste à élaborer des algorithmes parallèles performants pour résoudre les problèmes **MAX-W-SAT** difficiles.

- Appliquer les approches développées à d'autres problèmes d'optimisation combinatoire.

# Références :


[BKC 93] Bart. S & Kautz. A & Cohen. B
Local Search Strategies for Satisfiability Testing
Presented at the Second DIMACS challenge on
Cliques, coloring, and Satisfiability, oct, 1993.

[BF 97] Borchers. B & Furman. J
A two phase Algorithm for Max-Sat and
Weighted Max-sat Problems (1997)

[BP 96] Battiti. R & Protasi. M
Reactive Search, a history – based heuristic for
MAX-SAT : theory and applications,
Rutgers University, march 1996.

[BP 97] Battiti. R & Protasi M
Solving Max-Sat With Non Oblivious Functions and
History based Heuristics. Dimacs series in
discrete mathematics and theoretical computer science.
N°35, AMS and ACM Press, 1997.

[BP 98] Battititi R & Protasi M
Approximate Algorithms and Heuristics for Max-Sat;
HandBook of Combinatorial Optimization(vol.1) D.-Z Du
and P.M Pardalos(Eds) pp. 77-148 1998

[BT 95] Battiti R, Tecchiolli. G;
Local Search With Memory. Benchmarking RTS.
Operations Research Spectrum, 1995.

[Dri 93] Drias. H
Approche probabiliste du dénombrement des solutions
du problème MAX-SAT
Thèse de doctorat,1993

[Dri98] Drias. H,
New Algorithm For Counting and Checking Satisfiability and a restricted
Unfavorable Region For Polynomial Average Time
IEEE 1998. 0-7803-4547-9/98.

[Dri 99] Drias. H
Randomness in heuristics : An Experimental investigation for the Maximum Satisfiability Problem in Proc of IWANN-99, lectures notes in artificial intelligence Spain, June 1999.

[Gol 89] Goldberg. D.E
Genetic algorithms in search optimization and machine learning The university of Alabama, Edison Wesley, 1989.

[Glo 86] Glover. F
Future paths for integer programming and links to Artificial intelligence operational search, vol 31, 1986

[Glo 89] Glover. F
Taboo search : Part I, ORSA, journal on computing, 1989

[Glo 90] Glover. F
Taboo search : Part II, ORSA, journal on computing, 1990

[Glo 94] Glover. F
Taboo search for nonlinear and parametric optimization links to genetic algorithms
graduate school of business, University of Colorado, Boulder to appear in discrete applied mathematics, 1994

[Glo 95] Glover. F
Taboo search fundamentals and uses, journal of heuristic, 1995

[Glo 96] Glover. F
Taboo search and adaptive memory programming advances, application and challenges, computer science and operation research, 1996

[GLM 00] Glover..F & Laguna.M & Marti. R
Scatter search
University of Colorado, USA, 2000

[GLMC 99] Glover.F & Laguna.M & Marti. R & Campos.V
An experimental evaluation of Scatter search for The linear ordering problem, April 23, 1999

[HJ 89] Hansen P, Jaumard B.
Algorithms For Maximum satisfiability Problem.

[Her 95] Hertz .A
La méthode tabou appliquée aux problèmes d'ordonnancement, APII, vol 29, n° 4- 5, 1995

[Hol 92] Holland. J
Les algorithmes génétiques pour la science n°9 1992

[JG 79] Johnson.S & Garey. R
Computers and Intractability.
Guide to theory of the completeness.(1979).

[Joh74] Johnson. S,
Approximation algorithms for combinatorial problems, Journal of Computer and System Sciences 9(1974), 256-278.

[LG 93] Laguna .M & Glover. F
Taboo search modern heuristic techniques for
Combinatorial problem, C. Reeves, 1993.

[Lag 99] Laguna. M
SCATTER SEARCH
Graduate school of business, University of
Colorado, Boulder August 16, 1999

[LMC 99] Laguna.M & Marti. R & Campos. V
Scatter search for the linear ordering problem
University of Colorado, Boulder February 2, 1999

[MSG97] Mazure. B, Sais L, Gregoire.E;
Taboo Search For Sat, 1997.

[KS93] Kirkpatrick S, Selman B;
Critical Behavior in The Satisfiability Of Random Boolean formulae.

[RF96] Resende.MGC and T.A.Feo;
A grasp for satisfiability; Proc. of the second Dimacs Algorithm Implementation Challenge on cliques, coloring sansatisfiability ; Dimacs series on discrete
Mathematics and theoretical Computer science, n°26, 19-96, pp:499-520.

[SHKC 94] Selman B, Henry A, Kautz Z, Cohen B;
Noise Strategies For Improving Local Search.
Proceeding of AAAI-94, 1994.

[SML92] Selman B, Mitchell D, levesque H;
Hard And Easy Distribution Of Sat Problems.
In Proceedings Of The Tenth National Conference On Artificial Intelligence(AAAJ' 92), 459-465.

[SML92] Selman B, Mitchell D, levesque H;
A New Method For Solving Hard Satisfiability
Problems. In Proceeding Of Tenth Natinal Conference On Artificial
Intelligence(AAAI'92),440,446.

[Sia 95] Siarruy .P
La méthode du recuit simulé : théorie et applications.
Automatique productique, informatique industrielle
Vol 29 N° 4/1995.

[Tal 95] Talbi. EG
Algorithmes génétiques parallèles : techniques et
Applications. Publication université de Lille 1995.

[Vou 97] Voudouris .C
Guided local search for combinatorial optimization
problems, PHD thesis, department of computer science,
University of Essex, colchester, July 1997

[VT 98] Voudouris .C & Tsang. EPK
Guided local search and its application to the traveling
salesman problem, European journal of operational research,
march 1998.

# Annexe :

## Récapitulatifs des tests

Les tableaux qui suivent donnent les résultats des différents tests effectués afin de déterminer les paramètres des méta- heuristiques mises en œuvre tout au long de notre étude.

Les tests ont été effectués sur différentes instances **MAX-W-SAT** dont les solutions optimales sont connues

Vu que les algorithmes développés contiennent certaines étapes avec un choix aléatoire (bruit, marche, solution initiale), pour chaque jeu d'essai, dix(10) exécutions ont été réalisées. La meilleure solution obtenue pour chaque algorithme ainsi que sa durée d'exécution sont sélectionnés.

*Dans ce qui suit, nous avons désigné les différentes solutions par :*

- **SS** : Somme des poids des clauses satisfaites donnée par la recherche dispersée.
- **SS_tcpu** : Temps d'exécution de la recherche dispersée.
- **SSW** : Somme des poids des clauses satisfaites donnée par la recherche dispersée avec marche aléatoire.
- **SSW_tcpu** : Temps d'exécution de la recherche dispersée avec marche aléatoire.
- **SSnoise** : Somme des poids des clauses satisfaites donnée par la recherche dispersée avec bruit aléatoire.
- **SSN_tcpu** : Temps d'exécution de la recherche dispersée avec bruit aléatoire.
- **GLS** : Somme des poids des clauses satisfaites donnée par la recherche locale guidée.
- **GLS_tcpu** : Temps d'exécution de la recherche locale guidée.
- **GLSN** : Somme des poids des clauses satisfaites donnée par la recherche locale guidée avec bruit aléatoire.
- **GN_cpu** : Temps d'exécution de la recherche locale guidée avec bruit aléatoire.
- **GLSW** : Somme des poids des clauses satisfaites donnée par la recherche locale guidée avec marche aléatoire.
- **GW_cpu** : Temps d'exécution de la recherche locale guidée avec marche aléatoire.
- **TabouW** : Somme des poids des clauses satisfaites donnée par la recherche tabou avec bruit aléatoire.
- **Tab_tcpu** : Temps d'exécution de la recherche tabou avec bruit aléatoire.

- **GSATW** : Somme des poids des clauses satisfaites donnée par la procédure GSAT avec marche aléatoire.
- **Gsat_tcpu** : Temps d'exécution de la procédure **GSAT** avec bruit

aléatoire.

- **Détermination des Paramètres de la recherche dispersée :**

Le but de cette évaluation est de mesurer l'évolution de la qualité de la solution obtenue par la recherche dispersée, pour des populations de tailles différentes, et de déterminer le nombre maximum d'itérations qui représente le critère d'arrêt de l'algorithme.

| Benchmark | SS | SS_cpu | Benchmark | SS | SS_cpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 97.26 | Jnh217 | 394238 | 1008.06 |
| Jnh202 | 394044 | 1226.03 | Jnh218 | 394238 | 150.94 |
| Jnh203 | 394135 | 1441.79 | Jnh1 | 420909 | 1316.91 |
| Jnh205 | 394238 | 364.87 | Jnh10 | 420165 | 1860.38 |
| Jnh207 | 394006 | 1207.59 | Jnh11 | 420274 | 1772.86 |
| Jnh208 | 393842 | 1347.46 | Jnh12 | 420706 | 1748.86 |
| Jnh209 | 394057 | 1201.20 | Jnh13 | 420302 | 1866.45 |
| Jnh210 | 394238 | 123.59 | Jnh14 | 420510 | 1825.21 |
| Jnh211 | 393372 | 1393.20 | Jnh15 | 419285 | 1907.53 |
| Jnh212 | 394074 | 1231.54 | Jnh16 | 420401 | 1738.11 |
| Jnh214 | 393205 | 1289.80 | Jnh17 | 420797 | 1647.98 |
| Jnh215 | 393951 | 1533 | Jnh18 | 419868 | 1891.74 |
| Jnh216 | 393462 | 1620.87 | Jnh19 | 420107 | 1786.74 |

**Figure 1. Les résultats trouvés par la recherche dispersée pour Maxiter=20, B1=5, B2=5, tpop=100, et Aiter=10.**

| Benchmark | SS | SS_cpu | Benchmark | SS | SS_cpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 15.30 | Jnh217 | 393817 | 543.30 |
| Jnh202 | 392355 | 606.28 | Jnh218 | 394238 | 205.57 |
| Jnh203 | 393462 | 696.39 | Jnh1 | 420806 | 547.82 |
| Jnh205 | 393816 | 587.76 | Jnh10 | 419812 | 749.47 |
| Jnh207 | 393131 | 580.37 | Jnh11 | 419645 | 731.75 |
| Jnh208 | 393064 | 634.26 | Jnh12 | 420225 | 718.92 |
| Jnh209 | 393868 | 592.92 | Jnh13 | 418574 | 762.43 |
| Jnh210 | 394238 | 261.75 | Jnh14 | 419395 | 744.16 |
| Jnh211 | 392035 | 672.63 | Jnh15 | 419152 | 770.08 |
| Jnh212 | 393712 | 595.42 | Jnh16 | 420412 | 692.24 |
| Jnh214 | 392455 | 612.57 | Jnh17 | 420602 | 681.35 |
| Jnh215 | 393453 | 621.37 | Jnh18 | 419448 | 755.95 |
| Jnh216 | 392203 | 660.44 | Jnh19 | 419144 | 737.19 |

**Figure 2 . Les résultats trouvés par la recherche dispersée pour Maxiter=3, B1=3, B2=2, tpop=50, et Aiter=100.**

| Benchmark | SS | SS_cpu | Benchmark | SS | SS_cpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 5.03 | Jnh217 | 394102 | 954.36 |
| Jnh202 | 394044 | 1088.43 | Jnh218 | 394238 | 304.02 |
| Jnh203 | 392937 | 1233.58 | Jnh1 | 420806 | 954.61 |

| | | | | | |
|---|---|---|---|---|---|
| Jnh205 | 393958 | 1022.88 | Jnh10 | 419518 | 1311.70 |
| Jnh207 | 394171 | 1141.53 | Jnh11 | 419790 | 1240.01 |
| Jnh208 | 392773 | 994.19 | Jnh12 | 420706 | 1279.55 |
| Jnh209 | 393868 | 267.49 | Jnh13 | 420187 | 1339.79 |
| Jnh210 | 394238 | 1197.92 | Jnh14 | 420160 | 1303.86 |
| Jnh211 | 392185 | 1056.67 | Jnh15 | 419783 | 1348.46 |
| Jnh212 | 393970 | 1056.67 | Jnh16 | 420663 | 1230.21 |
| Jnh214 | 393006 | 1085.44 | Jnh17 | 420231 | 1193.31 |
| Jnh215 | 394002 | 1125.31 | Jnh18 | 419933 | 1327.09 |
| Jnh216 | 393224 | 1146.67 | Jnh19 | 419514 | 1288 |

**Figure 3. Les résultats trouvés par la recherche dispersée pour Maxiter=3, B1=3, B2=2, tpop=100, et Aiter=100.**

| Benchmark | SS | SS_cpu | Benchmark | SS | SS_cpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 23.77 | Jnh217 | 394202 | 1448.75 |
| Jnh202 | 393961 | 1605.75 | Jnh218 | 394189 | 1578.61 |
| Jnh203 | 392444 | 1847.99 | Jnh1 | 420744 | 1447.30 |
| Jnh205 | 393975 | 1555.12 | Jnh10 | 419382 | 1982.26 |
| Jnh207 | 394229 | 1554.52 | Jnh11 | 420113 | 1882.13 |
| Jnh208 | 393021 | 1706.86 | Jnh12 | 420645 | 1904.10 |
| Jnh209 | 393912 | 1490.46 | Jnh13 | 420400 | 2019.51 |
| Jnh210 | 394238 | 176.50 | Jnh14 | 420106 | 1973.51 |
| Jnh211 | 392380 | 1787.21 | Jnh15 | 419323 | 2046.66 |
| Jnh212 | 394117 | 1593.59 | Jnh16 | 419982 | 1839.88 |
| Jnh214 | 392927 | 1615.92 | Jnh17 | 42408 | 1805.30 |
| Jnh215 | 393593 | 1654.43 | Jnh18 | 419098 | 2006.20 |
| Jnh216 | 393395 | 1734.13 | Jnh19 | 419240 | 1944.51 |

**Figure 4 . Les résultats trouvés par la recherche dispersée pour Maxiter=3, B1=5, B2=5, tpop=50, et Aiter=100.**

D’après les tables ci-dessus représentant les solutions trouvées par la recherche dispersée, on constate que la taille de la population est grande meilleur est la qualité du résultat.

| Benchmark | SS | SS_cpu | Benchmark | SS | SS_cpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 26.31 | Jnh217 | 393937 | 300.02 |
| Jnh202 | 392908 | 334.83 | Jnh218 | 394189 | 329.58 |

| | | | | | |
|---|---|---|---|---|---|
| Jnh203 | 393475 | 383.61 | Jnh1 | 420728 | 301.45 |
| Jnh205 | 393534 | 323.99 | Jnh10 | 419015 | 409.99 |
| Jnh207 | 393206 | 321.44 | Jnh11 | 419575 | 328.50 |
| Jnh208 | 392568 | 356.91 | Jnh12 | 420091 | 398.85 |
| Jnh209 | 393097 | 310.10 | Jnh13 | 418810 | 419.30 |
| Jnh210 | 394238 | 24.56 | Jnh14 | 419769 | 414.84 |
| Jnh211 | 391587 | 373.51 | Jnh15 | 419389 | 422.75 |
| Jnh212 | 393683 | 325.89 | Jnh16 | 420029 | 382.52 |
| Jnh214 | 392511 | 368.92 | Jnh17 | 420830 | 377.76 |
| Jnh215 | 392927 | 344.88 | Jnh18 | 419064 | 418.38 |
| Jnh216 | 392606 | 360.02 | Jnh19 | 419047 | 403.72 |

**Figure 5. Les résultats trouvés par la recherche dispersée pour Maxiter=1, B1=2, B2=2, tpop=40, et Aiter=100.**

| Benchmark | SS | SS_tcpu | Benchmark | SS | SS_tcpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 6.44 | Jnh217 | 394238 | 650.46 |
| Jnh202 | 394022 | 2067.29 | Jnh218 | 394238 | 605.65 |
| Jnh203 | 394135 | 2379.43 | Jnh1 | 420925 | 1159.88 |
| Jnh205 | 394238 | 1840.49 | Jnh10 | 420165 | 1860.38 |
| Jnh207 | 394045 | 1975.44 | Jnh11 | 420274 | 2404.32 |
| Jnh208 | 393934 | 2219.86 | Jnh12 | 420706 | 2407.00 |
| Jnh209 | 393971 | 1916.71 | Jnh13 | 420302 | 2593.60 |
| Jnh210 | 394238 | 293.56 | Jnh14 | 420510 | 2555.30 |
| Jnh211 | 393372 | 2296.44 | Jnh15 | 419671 | 2680.10 |
| Jnh212 | 394029 | 2046.90 | Jnh16 | 420821 | 2380.91 |
| Jnh214 | 393377 | 2079.27 | Jnh17 | 420925 | 2080.00 |
| Jnh215 | 393451 | 2130.24 | Jnh18 | 420692 | 2576.90 |
| Jnh216 | 393399 | 2231.51 | Jnh19 | 420107 | 2521.19 |

**Figure 6. Les résultats trouvés par la recherche dispersée pour Maxiter=3, B1=5, B2=5, tpop=100, et Aiter=100.**

| Benchmark | SS | SS_tcpu | Benchmark | SS | SS_tcpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 8.64 | Jnh217 | 394238 | 418.33 |
| Jnh202 | 393912 | 745.65 | Jnh218 | 394238 | 175.31 |
| Jnh203 | 394135 | 868.05 | Jnh1 | 420661 | 745.65 |
| Jnh205 | 393975 | 694.36 | Jnh10 | 420651 | 895.07 |

| | | | | | |
|---|---|---|---|---|---|
| Jnh207 | 393862 | 786.07 | Jnh11 | 420120 | 933.30 |
| Jnh208 | 393483 | 953.77 | Jnh12 | 420549 | 811.17 |
| Jnh209 | 394094 | 687.27 | Jnh13 | 420666 | 1036.40 |
| Jnh210 | 394238 | 12.72 | Jnh14 | 420058 | 1087.70 |
| Jnh211 | 393574 | 1023.97 | Jnh15 | 420314 | 1084.40 |
| Jnh212 | 394136 | 837.84 | Jnh16 | 420871 | 860.40 |
| Jnh214 | 393612 | 828.35 | Jnh17 | 420892 | 793.85 |
| Jnh215 | 393942 | 765.53 | Jnh18 | 419938 | 1147.65 |
| Jnh216 | 393805 | 821.90 | Jnh19 | 419244 | 980.78 |

**Figure 7. Les résultats trouvés par la recherche dispersée pour Maxiter=6, B1=2, B2=3, tpop=20, et Aiter=100.**

Suivant les tables ci-dessus, on remarque que la qualité de la solution s'améliore plus rapidement dans les premières itérations que dans les suivantes. A partir d'un certain seuil, il est inutile d'augmenter le nombre d'itérations, car les variations deviennent très petites. D'après ces tests, on peut fixer ce nombre à trois(3) itérations.

- **Détermination des Paramètres de la recherche Locale guidée :**

L'objectif de cette évaluation est de déterminer le nombre maximum d'itérations qui représente le critère d'arrêt, le nombre d'améliorations de la recherche locale, ainsi que le facteur de diversification $\lambda$.

| Benchmark | GLS | GLS_cpu | Benchmark | GLS | GLS_cpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 21.07 | Jnh217 | 394238 | 1606.38 |
| Jnh202 | 393818 | 3383.07 | Jnh218 | 394238 | 914.47 |
| Jnh203 | 390206 | 1916.82 | Jnh1 | 420806 | 1782.72 |
| Jnh205 | 393903 | 1883.44 | Jnh10 | 419420 | 2432.27 |
| Jnh207 | 393585 | 1884.45 | Jnh11 | 420413 | 2300.99 |
| Jnh208 | 392936 | 2080.54 | Jnh12 | 420925 | 1220.29 |
| Jnh209 | 393812 | 1824.38 | Jnh13 | 419491 | 3249.01 |
| Jnh210 | 394238 | 334.33 | Jnh14 | 420545 | 2738.84 |
| Jnh211 | 392921 | 2189.30 | Jnh15 | 419004 | 43800.2 |
| Jnh212 | 393993 | 1952.04 | Jnh16 | 420063 | 2294.55 |
| Jnh214 | 393341 | 1974.02 | Jnh17 | 420830 | 2226.41 |
| Jnh215 | 393872 | 2042.12 | Jnh18 | 419525 | 2467.08 |
| Jnh216 | 393391 | 2115.18 | Jnh19 | 419468 | 2428.60 |

**Figure 8. Les résultats trouvés par la recherche locale guidée pour Maxiter=100, $\lambda$=1, et Aiter=10000.**

| Benchmark | GLS | GLS_cpu | Benchmark | GLS | GLS_cpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 21.17 | Jnh217 | 393268 | 177.05 |
| Jnh202 | 393815 | 197.89 | Jnh218 | 393692 | 194.99 |
| Jnh203 | 392597 | 225.55 | Jnh1 | 420806 | 176.41 |
| Jnh205 | 393371 | 189.76 | Jnh10 | 419378 | 243.02 |
| Jnh207 | 392660 | 189.21 | Jnh11 | 418660 | 245.50 |

| | | | | | |
|---|---|---|---|---|---|
| Jnh208 | 392241 | 208.24 | Jnh12 | 419776 | 236.27 |
| Jnh209 | 393279 | 181.96 | Jnh13 | 419002 | 248.71 |
| Jnh210 | 393791 | 258.64 | Jnh14 | 419131 | 244.70 |
| Jnh211 | 392921 | 219.77 | Jnh15 | 418323 | 252.11 |
| Jnh212 | 393730 | 194.94 | Jnh16 | 419452 | 226.46 |
| Jnh214 | 393341 | 147.15 | Jnh17 | 420331 | 221.35 |
| Jnh215 | 393872 | 202.24 | Jnh18 | 418960 | 245.86 |
| Jnh216 | 392641 | 210.73 | Jnh19 | 418968 | 240.44 |

**Figure 9. Les résultats trouvés par la recherche locale guidée pour Maxiter=10, λ=1, et Aiter=1000.**

| Benchmark | GLS | GLS_cpu |
|---|---|---|
| Jnh201 | 394238 | 22.01 |
| Jnh202 | 393815 | 493.79 |
| Jnh203 | 392597 | 564.23 |
| Jnh205 | 393379 | 478.14 |
| Jnh207 | 393169 | 472.73 |
| Jnh208 | 392241 | 520.90 |
| Jnh209 | 393568 | 454.18 |
| Jnh210 | 394237 | 334.93 |

**Figure 10. Les résultats trouvés par la recherche locale guidée pour Maxiter=50, λ=1, et Aiter=500.**

| Benchmark | GLS | GLS_cpu |
|---|---|---|
| Jnh201 | 394238 | 22.08 |
| Jnh202 | 393815 | 513.61 |
| Jnh203 | 392597 | 587.13 |
| Jnh205 | 393379 | 495.42 |
| Jnh207 | 393169 | 490.18 |
| Jnh208 | 392241 | 540.65 |
| Jnh209 | 393568 | 471.00 |
| Jnh210 | 394238 | 347.51 |

**Figure 11. Les résultats trouvés par la recherche locale guidée pour Maxiter=50,Aiter=500, et λ=2.**

| Benchmark | GLS | GLS_cpu |
|---|---|---|
| Jnh201 | 394238 | 21.96 |
| Jnh202 | 393815 | 512.53 |
| Jnh203 | 392597 | 587.43 |
| Jnh205 | 393379 | 495.42 |
| Jnh207 | 393169 | 490.18 |
| Jnh208 | 392241 | 540.65 |
| Jnh209 | 393568 | 471.00 |

| | | |
|---|---|---|
| Jnh210 | 394238 | 347.19 |

**Figure 12. Les résultats trouvés par la recherche locale guidée pour Maxiter=50,Aiter=500, et λ=0,5.**

| Benchmark | GLS | GLS_cpu |
|---|---|---|
| Jnh201 | 393838 | 13.81 |
| Jnh202 | 392086 | 22.08 |
| Jnh203 | 391610 | 23.63 |
| Jnh205 | 393304 | 20.04 |
| Jnh207 | 391960 | 19.74 |
| Jnh208 | 392241 | 22.08 |
| Jnh209 | 393279 | 18.42 |
| Jnh210 | 393101 | 19.86 |

**Figure 13. Les résultats trouvés par la recherche locale guidée pour Maxiter=10,Aiter=100, et λ=1,5.**

| Benchmark | GLS | GLS_cpu |
|---|---|---|
| Jnh201 | 394238 | 1.62 |
| Jnh202 | 392086 | 2.13 |
| Jnh203 | 391610 | 2.63 |
| Jnh205 | 390197 | 2.40 |
| Jnh207 | 391817 | 2.12 |
| Jnh208 | 391782 | 2.44 |
| Jnh209 | 391084 | 2.06 |
| Jnh210 | 390281 | 2.25 |

**Figure 14. Les résultats trouvés par la recherche locale guidée pour Maxiter=10,Aiter=10, et λ=3.**

| Benchmark | GLS | GLS_cpu |
|---|---|---|
| Jnh201 | 393838 | 13.85 |
| Jnh202 | 392086 | 28.88 |
| Jnh203 | 391610 | 23.69 |
| Jnh205 | 393304 | 24.67 |
| Jnh207 | 391960 | 19.91 |
| Jnh208 | 392241 | 18.54 |
| Jnh209 | 393579 | 22.46 |
| Jnh210 | 393101 | 20.49 |

**Figure 15. Les résultats trouvés par la recherche locale guidée pour Maxiter=10,Aiter=100, et λ=0.2.**

L'effet du facteur de diversification λ sur la recherche locale guidée est montré sur les figures 8, 9, 10, 11, 12, 13, 14, et 15. On constate qu'une valeur λ plus élevée n'implique pas une solution meilleure. Concernant le nombre d'itérations, on remarque que plus on l'augmente, plus la qualité de la solution est meilleure et plus le temps d'exécution de l'algorithme croît.

- **Détermination des Paramètres de la recherche dispersée avec marche aléatoire et de la recherche dispersée avec bruit aléatoire :**

Un nombre important de paramètres sont utilisés dans la méthode de recherche dispersée hybride( nombre d'itérations maximal, taille de la population, la taille de l'ensemble R, et la probabilité P). Les paramètres choisis sont ceux pour lesquels un compromis a été constaté entre le temps d'exécution de l'algorithme et la qualité de la solution trouvée. Ces paramètres sont fixés comme suit :

- Paramètres de l'algorithme de la recherche dispersée avec bruit aléatoire : Sizepop=100, B1=5, B2= 5, Maxiter=10, Aiter=1000, et p=0.5.

- Paramètres de l'algorithme de la recherche dispersée avec marche aléatoire: Sizepop=100, B1=5, B2= 5,Maxiter=10, Aiter=1000, et p=0.5.

| Benchmark | Optimale | SSW | Swtcpu | SSnoise | SSN_tcpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 394238 | 28.48 | 394238 | 25.12 |
| Jnh202 | 394170 | 392788 | 256.05 | 394044 | 1086.39 |
| Jnh203 | 394199 | 392795 | 293.14 | 393498 | 1198.62 |
| Jnh205 | 394238 | 393300 | 250.23 | 393903 | 338.74 |
| Jnh207 | 394238 | 393262 | 245.67 | 393745 | 758.63 |
| Jnh208 | 394159 | 392608 | 272.03 | 393822 | 384.74 |
| Jnh209 | 394238 | 392819 | 237.29 | 393459 | 919.42 |
| Jnh210 | 394238 | 393951 | 240.91 | 394238 | 395.43 |
| Jnh211 | 393979 | 391993 | 284.32 | 392986 | 526.55 |
| Jnh212 | 394238 | 392710 | 251.17 | 394074 | 610.96 |
| Jnh214 | 394163 | 392825 | 256.98 | 393077 | 608.92 |
| Jnh215 | 394150 | 393188 | 264.44 | 393951 | 478.22 |
| Jnh216 | 394226 | 392661 | 277.07 | 393882 | 860.10 |
| Jnh217 | 394238 | 394238 | 165.42 | 394197 | 499.25 |
| Jnh218 | 394238 | 393763 | 251.90 | 394238 | 285.55 |
| Jnh1 | 420925 | 420609 | 234 | 420806 | 658.16 |
| Jnh10 | 420840 | 418550 | 316.04 | 419603 | 674.21 |
| Jnh11 | 420753 | 419341 | 296.49 | 419658 | 481.34 |
| Jnh12 | 420925 | 418947 | 307.79 | 420701 | 484.94 |
| Jnh13 | 420816 | 418447 | 320.62 | 418891 | 598.83 |
| Jnh14 | 420824 | 419145 | 313.44 | 419358 | 589.05 |
| Jnh15 | 420719 | 419032 | 322.52 | 419800 | 848.66 |
| Jnh16 | 420919 | 419501 | 291.38 | 420746 | 914.16 |
| Jnh17 | 420925 | 419959 | 289.65 | 420925 | 225.89 |
| Jnh18 | 420795 | 418710 | 317.79 | 419746 | 680.99 |
| Jnh19 | 420759 | 418765 | 310.52 | 419746 | 567.79 |

**Figure 16. Les résultats trouvés par la recherche dispersée avec marche aléatoire et la recherche dispersée avec bruit aléatoire pour Maxiter=10000, B1=5, B2=5, tpop=100, Aiter=10, et P=0.5.**

| Benchmark | Optimale | SSW | Swtcpu | SSnoise | SSN_tcpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 394238 | 15.38 | 394019 | 23.20 |
| Jnh202 | 394170 | 392256 | 30.48 | 391271 | 30.43 |
| Jnh203 | 394199 | 390147 | 34.95 | 391798 | 33.47 |
| Jnh205 | 394238 | 392095 | 30.04 | 392365 | 29.98 |
| Jnh207 | 394238 | 392404 | 29.88 | 393296 | 29.24 |
| Jnh208 | 394159 | 392575 | 33.07 | 390797 | 32.40 |
| Jnh209 | 394238 | 393329 | 29.55 | 394187 | 28.46 |
| Jnh210 | 394238 | 393604 | 30.56 | 391349 | 30.04 |
| Jnh211 | 393979 | 391941 | 33.89 | 390949 | 33.37 |
| Jnh212 | 394238 | 393378 | 30.91 | 392364 | 30.00 |
| Jnh214 | 394163 | 391984 | 31.13 | 391200 | 31.01 |
| Jnh215 | 394150 | 390910 | 32.15 | 392447 | 31.44 |
| Jnh216 | 394226 | 391822 | 33.01 | 391949 | 32.13 |
| Jnh217 | 394238 | 392530 | 28.73 | 391941 | 28.54 |
| Jnh218 | 394238 | 391541 | 31.28 | 393763 | 29.97 |
| Jnh1 | 420925 | 419636 | 28.81 | 419408 | 28.73 |
| Jnh10 | 420840 | 416970 | 36.81 | 417221 | 36.95 |
| Jnh11 | 420753 | 418944 | 35.28 | 417839 | 35.04 |
| Jnh12 | 420925 | 418165 | 35.57 | 418419 | 35.45 |
| Jnh13 | 420816 | 417750 | 35.91 | 417437 | 37.19 |
| Jnh14 | 420824 | 417813 | 37.25 | 418873 | 37.18 |
| Jnh15 | 420719 | 417965 | 37.28 | 416643 | 37.66 |
| Jnh16 | 420919 | 418769 | 35.63 | 417582 | 33.75 |
| Jnh17 | 420925 | 419104 | 35.51 | 418446 | 34.90 |
| Jnh18 | 420795 | 417322 | 37.81 | 417570 | 37.61 |
| Jnh19 | 420759 | 417298 | 36.83 | 417434 | 36.51 |

**Figure 17. Les résultats trouvés par la recherche dispersée avec marche aléatoire et la recherche dispersée avec bruit aléatoire pour Maxiter=20, B1=5, B2=5, tpop=100, Aiter=10, et P=0.5.**

- **Détermination des Paramètres de la recherche locale guidée avec marche aléatoire et de la recherche locale guidée avec bruit aléatoire :**

Les meilleurs résultats ont été obtenus pour une valeur de Maxiter= 100, une valeur de Aiter=1000, et une valeur de P=0.5.

| Benchmark | Optimale | GLSW | GW_cpu | GLSN | GN_cpu |
|---|---|---|---|---|---|
| Jnh201 | 394238 | 394238 | 78.60 | 394238 | 40.14 |
| Jnh202 | 394170 | 393157 | 215.03 | 393197 | 175.97 |
| Jnh203 | 394199 | 392538 | 240.03 | 392463 | 227.34 |
| Jnh205 | 394238 | 393816 | 135.63 | 393563 | 248.27 |
| Jnh207 | 394238 | 393593 | 244.07 | 393296 | 301.62 |
| Jnh208 | 394159 | 392338 | 292.66 | 392706 | 188.44 |
| Jnh209 | 394238 | 393141 | 145.37 | 393524 | 145.82 |
| Jnh210 | 394238 | 394238 | 15.72 | 394238 | 86.30 |

**Figure 18. Les résultats trouvés par la recherche locale guidée avec marche aléatoire et la recherche locale guidée avec bruit aléatoire pour Maxiter = 1000, P=0.5, et Aiter= 1000.**

| Benchmark | GLSW | GW_cpu | GLSN | GN_cpu |
|---|---|---|---|---|
| Jnh201 | 369791 | 0.77 | 394238 | 10.23 |
| Jnh202 | 374372 | 0.94 | 391891 | 19.77 |
| Jnh203 | 391833 | 23.57 | 392463 | 11.21 |
| Jnh205 | 392893 | 10.60 | 392738 | 19.11 |
| Jnh207 | 366714 | 1 | 391545 | 19.39 |
| Jnh208 | 391336 | 11.46 | 390569 | 10.43 |
| Jnh209 | 393141 | 10 | 393524 | 9.06 |
| Jnh210 | 393135 | 10.47 | 393536 | 9.01 |

**Figure 19. Les résultats trouvés par la recherche locale guidée avec marche aléatoire et la recherche locale guidée avec bruit aléatoire pour Maxiter = 50, P=0.5, et Aiter= 500.**

- **Détermination des Paramètres de la recherche tabou avec bruit aléatoire:**

Les paramètres choisis sont ceux pour lesquels un bon compromis a été trouvé entre le coût d'exécution de l'algorithme et la qualité de la solution obtenue.

| Benchmark | TabouW | Tab_tcpu |
|---|---|---|
| Jnh201 | 394238 | 143.45 |
| Jnh202 | 392837 | 242.58 |
| Jnh203 | 392485 | 275.26 |
| Jnh205 | 393752 | 235.07 |
| Jnh207 | 392708 | 230.28 |
| Jnh208 | 392380 | 254.41 |
| Jnh209 | 392992 | 225.30 |
| Jnh210 | 394238 | 222.46 |

**Figure 20. Les résultats trouvés par la recherche tabou avec bruit aléatoire pour Maxiter=10000, P=0.5, et tenure=7.**

| Benchmark | TabouW | Tab_tcpu |
|---|---|---|
| Jnh201 | 394238 | 11.27 |
| Jnh202 | 391583 | 23.97 |
| Jnh203 | 390908 | 28.07 |
| Jnh205 | 391940 | 24.09 |
| Jnh207 | 392404 | 23.34 |
| Jnh208 | 391383 | 25.27 |
| Jnh209 | 392992 | 21.94 |
| Jnh210 | 393827 | 22.11 |

**Figure 21. Les résultats trouvés par la recherche tabou avec bruit aléatoire pour Maxiter=1000, P=0.5 , et tenure=7.**

| Benchmark | TabouW | Tab_tcpu |
|---|---|---|
| Jnh201 | 394238 | 11.27 |
| Jnh202 | 391583 | 23.98 |
| Jnh203 | 390908 | 28.07 |
| Jnh205 | 391940 | 24.10 |
| Jnh207 | 392404 | 23.34 |
| Jnh208 | 391383 | 25.28 |
| Jnh209 | 392992 | 21.94 |
| Jnh210 | 393827 | 22.10 |

**Figure 22. Les résultats trouvés par la recherche tabou avec bruit aléatoire pour Maxiter=1000, P=0.5, et tenure=3.**

- **Détermination des Paramètres du GSAT avec marche aléatoire :**

Les figures 23 et 24, illustrent l'effet du paramètre Maxiter sur la procédure GSAT. On remarque que plus on augmente Maxiter, plus la qualité de la solution est meilleure, et plus le temps de calcul de l'algorithme croît.

| Benchmark | GSATW | Gsat_tcpu |
|---|---|---|
| Jnh201 | 394238 | 15.24 |
| Jnh202 | 393155 | 240.36 |
| Jnh203 | 392804 | 274.01 |
| Jnh205 | 393959 | 233.79 |
| Jnh207 | 393049 | 243.71 |
| Jnh208 | 392296 | 255.82 |
| Jnh209 | 392933 | 223.52 |
| Jnh210 | 393864 | 229.41 |

**Figure 23. Les résultats trouvés par GSAT avec marche aléatoire pour Maxiter=1000, P=0.5, et Num-ess=10.**

| Benchmark | GSATW | Gsat_tcpu |
|---|---|---|

| | | |
|---|---|---|
| Jnh201 | 394238 | 140.19 |
| Jnh202 | 393439 | 240659 |
| Jnh203 | 392816 | 2875.35 |
| Jnh205 | 393975 | 2338.34 |
| Jnh207 | 393649 | 2301.65 |
| Jnh208 | 392651 | 2558.96 |
| Jnh209 | 393746 | 2225.98 |
| Jnh210 | 394238 | 456.23 |

**Figure 24. Les résultats trouvés par GSAT avec marche aléatoire pour Maxiter=10000,P=0.5, et Num-ess=10.**

**CONCLUSION:**

Après cette expérimentation, les paramètres de chaque méta- heuristiques citées ci-dessus sont fixés aux valeurs suivantes :

- Paramètres de l'algorithme de la recherche tabou avec bruit aléatoire : Tenure = 7, Maxiter= 10000 et P=0.5.

- Paramètres de l'algorithme du GSAT avec marche aléatoire : Maxiter=1000, NUM-ess= 10, et P=0.5.

- Paramètres de l'algorithme de la recherche dispersée : Sizepop=100, B1=5, B2= 5,Maxiter=3, et Aiter=100.

- Paramètres de l'algorithme de la recherche locale guidée : Maxiter= 100, Aiter=1000, et $\lambda$=1.

- Paramètres de l'algorithme de la recherche dispersée avec bruit aléatoire : Sizepop=100, B1=5, B2= 5, Maxiter=10, Aiter=1000, et p=0.5.

- Paramètres de l'algorithme de la recherche dispersée avec marche aléatoire: Sizepop=100, B1=5, B2= 5,Maxiter=10, Aiter=1000, et p=0.5.

- Paramètres de l'algorithme de la recherche locale guidée avec bruit aléatoire : Maxiter= 100, Aiter=1000, P=0.5, et $\lambda$=1.

- Paramètres de l'algorithme de la recherche locale guidée avec marche aléatoire: Maxiter= 100, Aiter=1000, P=0.5, et $\lambda$=1.