UNIVERSITÉ DU QUÉBEC À MONTRÉAL

MÉTHODE SAT ET ALGORITHME DPLL APPLIQUÉS À UN PROBLÈME DE RECHERCHE OPÉRATIONNELLE

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE

PAR

NABILA RAHMOUNE

AOÛT 2006

REMERCIEMENTS

Ce travail de recherche n'aurait pas vu le jour sans la contribution de nombreuses personnes à qui j'aimerais adresser des remerciements.

Je tiens tout d'abord à exprimer toute ma gratitude à mon directeur de recherche, le professeur Roger Villemaire, pour son soutien durant ce programme de maîtrise. Il m'a non seulement témoigné de la confiance et de la générosité durant les moments difficiles, mais il a été un inestimable guide tout au long de cette recherche. Je le remercie encore une fois pour sa patience, sa disponibilité et pour ses pertinents conseils et je lui suis extrêmement reconnaissante de m'avoir permis de faire mes premiers pas dans le monde de la recherche. Je lui dis un grand merci.

J'adresse également mes sincères remerciements à monsieur Hantao Zhang, pour avoir rapidement répondu à nos questions au sujet du logiciel SATO.

Je remercie vivement « examinateur 1 et examinateur 2 »pour l'intérêt qu'ils portent au sujet dont ce travail fait l'objet et pour l'honneur qu'ils me font en le lisant. Merci d'apporter vos cautions scientifiques à ce travail.

Je désire aussi remercier mon époux Amine, qui a su m'encourager, me soutenir et faire beaucoup de concessions afin que mon travail se déroule dans de bonnes conditions.

Je tiens à remercier, vivement mes amis, Sylvain Hallé et Ngantchaha Ngougue Ghislain, pour leurs conseils et supports.

Enfin, je tiens à reconnaître le support financier du conseil de recherches en sciences naturelles et en génie du Canada.

Merci beaucoup à tous.

À toi qui me manque tant, ma très chère soeur Malika.

À mon beau frère Ali.

Je dédie ce manuscrit et mes deux années de travail.

Que Dieu tout puissant vous accorde sa sainte miséricorde et vous accueille en son vaste paradis.

TABLE DES MATIÈRES

LISTE DES TABLEAUX vii

LISTE DES FIGURES viii

RÉSUMÉ ix

INTRODUCTION 1

CHAPITRE I
LOGIQUE PROPOSITIONNELLE ET PROBLÈME SAT 4

1.1 La logique propositionnelle 4

1.1.1 Syntaxe 4

1.1.2 Sémantique 6

1.2 La forme normale conjonctive (FNC) 8

1.2.1 Transformation 9

1.3 Le problème SAT 12

1.4 Conclusion 13

CHAPITRE II
ALGORITHMES DPLL POUR SAT 14

2.1 Définitions 14

2.2 Méthodes énumératives pour SAT 15

2.2.1 Davis-Putnam (DP) 16

2.2.2 Davis Putnam Logemann Loveland (DPLL) 18

2.2.3 Heuristiques de branchement 21

2.3 Retour-arrière (Backtracking) chronologique (CBT) 23

2.4 Retour ponctuel (Backjumping) (BJ) 24

2.5 L'apprentissage (Learning) 25

2.6 Quelques solutionners pour le problème SAT 26

2.7 Les TL-Clauses (True Literal) 28

2.7.1 Motivation 29

2.7.2 Particularité de SATO 29
2.7.3 Résolution des conflits aux TL-clauses 31
2.8 Conclusion 33
CHAPITRE III
PROBLÉMATIQUE 34
3.1 Description du problème d'ordonnancement de véhicules 34
3.1.1 Processus de l'ordonnancement du problème d'ordonnancement de véhicules 35
3.1.2 Problème à résoudre 37
3.2 Modalités pratiques 37
3.3 Conclusion 38
CHAPITRE IV
RÉSOLUTION DU PROBLÈME D'ORDONNANCEMENT DE VÉHICULES . 39
4.1 Méthodologie de résolution 39
4.2 Encodage en FNC/TL 40
4.2.1 Contraintes arithmétiques de base 41
4.2.2 Contraintes de purges 42
4.2.3 Contraintes de ratio 43
4.3 Optimisation multi-objectifs et approches de résolution 44
4.3.1 HPRC-LPRC-PCC 45
4.3.2 HPRC-PCC-LPRC 46
4.3.3 PCC-HPRC-LPRC 47
4.4 Conclusion 48
CHAPITRE V
RÉSULTATS EXPÉRIMENTAUX ET ANALYSE 50
5.1 Réalisation 50
5.2 Analyse 51
5.2.1 HPRC-LPRC-PCC 51
5.2.2 HPRC-PCC-LPRC 54
5.2.3 PCC-HPRC-LPRC 56
5.3 Conclusion 57

CONCLUSION 58

APPENDICE A
CODES DES INSTANCES 60

Références Bibliographiques 64

LISTE DES TABLEAUX

4.1 Complexité des TL-clauses 44

4.2 Complexité des contraintes clausales 44

5.1 Toutes Contraintes de Ratio, Coefficient de Purge 1.9 52

5.2 Toutes Contraintes de Ratio, Coefficient de Purge 1.3 53

5.3 Toutes Contraintes de Ratio, Coefficient de Purge 2.9 54

5.4 Toutes Contraintes de Ratio, Coefficient de Purge 3.9 54

5.5 Contraintes de Ratio Prioritaire, Coefficient de Purge 1.09 55

5.6 Types des Couleurs, Coefficient de Purge 1.05 56

A.1 Codes des instances de l'approche *Toutes Contraintes de Ratio* (TCR) . 61

A.2 Codes des instances de l'approche *Contraintes de Ratio Prioritaires* (CRP) 62

A.3 Codes des instances de l'approche *Types des Couleurs* (TC) 63

LISTE DES FIGURES

2.1 Arbre binaire représentant une affectation avec 3 variables 16

2.2 Algorithme de Davis-Putnam 17

2.3 Algorithme de Davis Putnam Logemann Loveland 19

2.4 Algorithme de Propagation Unitaire 20

2.5 Algorithme Simplifier 20

2.6 Exemple d'arbre de clauses avec SATO 28

2.7 Exemple d'une TL-clause 32

2.8 Solution de l'exemple 32

RÉSUMÉ

La littérature fait état des travaux de recherches qui ont été menés pour la résolution des problèmes d'ordonnancement de production. La complexité de ces problèmes rend nécessaire l'emploi de stratégies de recherche de solutions évoluées. Parmi celle-ci figure le formalisme du calcul propositionnel, le plus souvent sous forme normale conjonctive (FNC) associé au problème de satisfiabilité (SAT). Le présent travail de recherche a pour but d'intégrer les formalismes d'approches de résolution des problèmes SAT pour la résolution du problème d'ordonnancement de production, soit le *problème d'ordonnancement de véhicules*, proposé dans le cadre du challenge ROADEF'2005.

Dans un premier temps, les principaux algorithmes pour la résolution de problème SAT sont présentés, particulièrement les algorithmes basés sur le retour en arrière tels que le retour-arrière (Backtracking) et le retour ponctuel (Backjumping) étendus sur les TL-clauses (*True-Literal clauses*).

Ce travail de recherche couvre le développement de trois approches de résolutions du problème SAT appliquées au *problème d'ordonnancement de véhicules*. Pour chaque approche un encodage en FNC/TL traduisant les contraintes du problème ainsi que l'objectif à optimiser sont effectués. Ces FNC/TL sont générées en format DIMACS à l'aide du logiciel développé par l'auteur. Ensuite, une stratégie de résolution est décrite, en fixant à chaque fois l'objectif à optimiser. Dans la première approche, le problème est traité globalement. Les deux autres approches subdivisent le problème initial en sous-problèmes. Finalement une comparaison des trois approches est décrite.

Les instances du problème proposées par le challenge ROADEF'2005 sont utilisées comme base d'évaluation des approches développées. Les résultats obtenus sont comparés aux meilleurs résultats obtenus par le gagnant du challenge ROADEF'2005, à l'aide du logiciel suggéré par le challenge, soit *exeCarSeq*. Une analyse détaillée des résultats montre que notre stratégie de résolution du *problème d'ordonnancement de véhicules* est une voie prometteuse.

Mots clés : Forme normale conjonctive; Problème de satisfiabilité; Problème d'ordonnancement de véhicules; TL-clauses; Encodage en FNC/TL

INTRODUCTION

Le soucis majeur de chaque gestionnaire d'une usine d'automobiles est de pouvoir fournir à son client le véhicule demandé, le jour prévu et au meilleur coût possible. Les efforts sont faits à tous les niveaux de la fabrication : aussi bien pour la gestion des commandes, que pour la production du véhicule ou la livraison de celui-ci.

Les usines d'assemblage n'échappent pas à cet effort. Une attention permanente est portée à la diminution des coûts et à la réduction des délais de fabrication. L'usine se compose de trois ateliers successifs : la tôlerie, la peinture et le montage. On cherche à obtenir la meilleure qualité de fabrication au moindre coût.

Une bonne gestion de ces ateliers est très importante, dans la mesure où elle permet de faire des gains en ressources (matière, personnel,..). Par exemple, au niveau de l'atelier de peinture, nous pouvons procéder au regroupement des voitures de même couleur, afin de ne pas changer de teinte à chaque fois et ainsi économiser les coûts du rinçage des pistolets. Quand à l'atelier de montage, nous essayons de ranger les véhicules en fonction de la quantité de travail qu'ils demandent, en espaçant ceux pour lesquels la tâche sera plus longue du fait de leurs options. Ceci est fait pour que les opérateurs ne se retrouvent pas surchargés à certains moments ; quand le cas se produit, il faut qu'un deuxième opérateur intervienne pour aider le premier. Éviter ce problème de surcharge momentané permet de réduire les effectifs nécessaires au bon fonctionnement de l'atelier.

Un ordonnancement est réalisé afin de répartir uniformément la charge de travail dans la journée. Cela permet de bien ordonner les véhicules pour les ateliers, notamment pour le montage.

Notre travail se situe dans le cadre d'une recherche d'un meilleur ordonnancement des

véhicules dans la chaîne de la production. C'est dans cette optique, que le présent travail sera réalisé en visant l'étude des algorithmes de résolution de problème de la satisfiabilité (SAT) d'une formule booléenne sous forme normale conjonctive (FNC) et leur adaptation au *problème d'ordonnancement de véhicules*, sujet de notre recherche.

Le problème SAT consiste à déterminer si une formule propositionnelle mise sous forme FNC est satisfaite. SAT est un problème NP–complet et un paradigme de base dans beaucoup de domaines d'intelligence artificielle. De nombreuses études théoriques et pratiques ont porté sur le problème SAT. Elles ont permis de créer des algorithmes, souvent basés sur l'algorithme énumératif de Davis, Putnam Logemann et Loveland (DPLL) dont les implémentations sont de plus en plus efficaces, tel que le solutionner SATO4 version 2 basé sur l'algorithme DPLL, étendu sur les TL-clauses (*True-Literal clauses*), que nous utiliserons pour atteindre notre objectif.

Nous allons montrer comment on peut transformer le *problème d'ordonnancement de véhicules*, qui est en fait un problème de recherche opérationnelle, en un problème SAT, en le résolvant par un outil efficace comme SATO4.2.

Ce mémoire est organisé comme suit. Le chapitre 1 est consacré à la logique propositionnelle. Nous commençons par définir les concepts de bases. Nous accompagnons cette étude d'un état de la question succinct autour du problème SAT et nous décrivons ensuite le système SATO.

Le chapitre 2 présente les méthodes de résolution du problème SAT basées sur l'algorithme DPLL, en particulier ceux basés sur le retour en arrière, le retour-arrière (Backtracking) et le retour ponctuel (Backjumping) étendus sur les TL-clauses. Nous terminons le chapitre par une définition du principe des TL-clauses, pierre angulaire d'optimisation de l'algorithme de SATO4.2.

Le chapitre 3 passe en revue la problématique. Il présente une description du *problème d'ordonnancement de véhicules* dans le cadre du challenge ROADEF'2005. Ce chapitre décrit les exigences du problème, les contraintes à satisfaire, les objectifs à optimiser et

enfin les données nécessaires pour l'application numérique.

Au chapitre 4, nous présentons une traduction du problème en un problème SAT. Pour ce faire, nous encodons les différentes contraintes du problème en FNC/TL. Nous décrivons par la suite les trois méthodes étudiées pour la résolution de ce problème, suivies, d'une analyse des points forts et des lacunes de ces trois méthodes.

Finalement, le chapitre 5 est dédié à la description des étapes suivies permettant la résolution du problème et l'obtention de l'ordonnancement proprement dit des véhicules satisfaisant aux mieux l'objectif fixé au départ. Ensuite, nous présentons pour chacune des trois approches présentées au chapitre 4, une analyse comparative des résultats obtenus à ceux présentés par le challenge ROADEF'2005, dont le but est de déterminer les performances de nos méthodes développées.

La conclusion de ce mémoire permettra de porter un jugement général de la performance des trois méthodes de résolution de problème SAT et d'identifier des avenues futures de recherches intéressantes.

CHAPITRE I

LOGIQUE PROPOSITIONNELLE ET PROBLÈME SAT

Dans ce chapitre, nous étudions le formalisme du calcul propositionnel en définissant les différents concepts nécessaires à la compréhension des éléments contenus dans nos méthodes de résolutions, nous abordons la forme normale conjonctive et par la suite, nous introduisons la notion du problème de satisfaction d'une formule de la logique propositionnelle.

1.1 La logique propositionnelle

La logique propositionnelle est une partie de la logique traitant des propositions qui sont des affirmations pouvant être vraies ou fausses. En effet, la logique propositionnelle se trouve être une manière simplitiée de traduire une grande quantité d'informations.

Afin de se familiariser avec les bases de la logique propositionnelle, nous rappelons sa syntaxe et sa sémantique que nous trouvons dans la plupart des sources traitant du sujet [Cori, Lascar, 2003].

1.1.1 Syntaxe

Définition 1. *Une variable propositionnelle est une variable prenant les valeurs de vérité vrai ou faux. Elle représente la valeur logique d'une proposition. L'ensemble des variables propositionnelles est noté* χ.

Définition 2. *L'ensemble des connecteurs logiques pour la logique propositionnelle est*

défini par l'ensemble $\mathcal{P} = \{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.

- *La négation, noté* $\neg\ p$ *d'une proposition* p *est la proposition, qui est vraie quand* p *est fausse et fausse quand* p *est vraie. La négation est un connecteur unaire.*

  Quant aux connecteurs binaires, nous en avons quatre:

- *La conjonction (le « et » logique) de deux propositions* p *et* q*, notée* $p \wedge q$*, est une proposition qui est vraie dans le seul cas où* p *et* q *sont vraies toutes les deux.*

- *La disjonction (inclusive)(le « ou » logique) de deux propositions* p *et* q *est la proposition, notée* $p \vee q$*, qui est fausse dans le seul cas où* p *et* q *sont fausses toutes les deux.*

- *Le conditionnel (l'implication logique) de deux propositions* p *et* q *est la proposition, notée* $p \rightarrow q$*, qui est fausse dans le seul cas où* p *est vraie et* q *est fausse.*

- *Le biconditionnel (double implication) de deux propositions* p *et* q *est la proposition, notée* $p \leftrightarrow q$*, qui est fausse dans les deux cas où* p *et* q *n'ont pas la même valeur de vérité.*

Définition 3. *L'ensemble des formules propositionnelles* $\mathcal{FP}$ *est défini comme suit :*

1. $\chi \subseteq \mathcal{FP}$,
2. *vrai et faux* $\in \mathcal{FP}$,
3. *si* $\mathcal{A} \in \mathcal{FP}$ *alors* $(\mathcal{A})$ *est une formule,*
4. *si* $\mathcal{A} \in \mathcal{FP}$ *alors* $\neg\ \mathcal{A}$ *est une formule,*
5. *si* $\mathcal{A}$ *et* $\mathcal{B} \in \mathcal{FP}$ *alors* $(\mathcal{A} \wedge \mathcal{B})$, $(\mathcal{A} \vee \mathcal{B})$, $(\mathcal{A} \rightarrow \mathcal{B})$ *et* $(\mathcal{A} \leftrightarrow \mathcal{B}) \in \mathcal{FP}$.

Dans tous ce qui suit, nous nous n'utiliserons pas les parenthèses de façon stricte si il n'y a pas risque de confusion.

Soit p_k, q_k, $k = 1, \ldots, n$, n variables propositionnelles, nous avons que :

- $\bigwedge_{k=1}^{n}(p_k \vee q_k) \equiv (p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge \cdots \wedge (p_n \vee q_n)$,
- $\bigvee_{k=1}^{n}(p_k \wedge q_k) \equiv (p_1 \wedge q_1) \vee (p_2 \wedge q_2) \vee \cdots \vee (p_n \wedge q_n)$.

Définition 4. *Un littéral est une variable propositionnelle $x \in \chi$ (littéral positif) ou sa négation $\neg x$ (littéral négatif). L'ensemble des littéraux est noté $\mathcal{L}$. Nous noterons par $\bar{\ell}$ l'inverse du littéral ℓ, qui est défini comme suit : Lorsque ℓ est une variable x, $\bar{\ell}$ est $\neg x$. Lorsque ℓ est la négation $\neg x$ d'une variable x, alors $\bar{\ell}$ est x.*

Définition 5. *Une clause est une disjonction de littéraux. Nous considerons aussi souvent qu'une clause est tout simplement l'ensemble de ses littéraux.*

Définition 6. *Une clause est dite :*

1. *Unaire (unitaire) ou mono-littérale, si elle ne contient qu'un seul littéral.*
2. *Binaire, si elle contient deux littéraux.*
3. *n-aire, si elle contient exactement n littéraux.*

Nous notons $|c|$ le nombre de littéraux distincts de la clause c.

Définition 7. *Une clause vide est une clause qui ne contient aucun littéral. Elle est aussi appelée contradiction, est notée $\perp$ ou encore $\square$.*

Définition 8. *Deux clauses c_1 et c_2 se résolvent si et seulement si il existe un littéral ℓ, tel que $\ell \in c_1$ et $\bar{\ell} \in c_2$. La clause $(c_1 - \{\ell\}) \cup (c_2 - \{\bar{\ell}\})$ est appelée la résolvante en ℓ de c_1 et c_2.*

1.1.2 Sémantique

La sémantique de la logique propositionnelle repose principalement sur : la valeur de vérité (vrai (1) ou faux (0)) que peut prendre une variable propositionnelle. Quant à la valeur de vérité d'une formule, elle est déterminée par une assignation de la valeur de vérité à l'ensemble des variables propositionnelles, appelées variables booléennes.

Définition 9. *Une affectation partielle est une fonction* $\mathcal{B}$*:* $\mathcal{W} \rightarrow \{0,1\}$ *avec* $\mathcal{W} \subseteq \chi$*. Certaines des variables peuvent ne pas obtenir de valeur de vérité.*

Définition 10. *Une variable instanciée par une affectation partielle est une variable ayant une valeur de vérité.*

Définition 11. *Une affectation complète pour une formule* Φ *est une fonction* $\mathcal{A}$*:* $\chi_\Phi \rightarrow \{0,1\}$ *où* χ_Φ *représente l'ensemble des variables de* Φ*. Toutes les variables sont instanciées. L'ensemble des affectations complètes relative à une formule* Φ *est notée* $\mathcal{S}$ *et son cardinal est* $2^{card(\chi_\Phi)}$*, où* $card(\chi_\Phi)$ *est le nombre de variables distinctes apparaissant dans* Φ*.*

Définition 12. *Pour une affectation complète* $\mathcal{A}$ *pour une formule* Φ*, nous définissons le fait que* $\mathcal{A}$ *satisfait* Φ*, noté* $\mathcal{A} \models \Phi$*, récursivement de la façon suivante.*

1. *Pour une variable propositionnelle* x
   - $\mathcal{A} \models x$ *, si* $\mathcal{A}(x) = 1$
2. *Pour des formules propositionnelle* p *et* q
   - $\mathcal{A} \models \neg p$ *, si* $\mathcal{A} \not\models p$
   - $\mathcal{A} \models p \wedge q$ *, si* $\mathcal{A} \models p$ *et* $\mathcal{A} \models q$
   - $\mathcal{A} \models p \vee q$ *, si* $\mathcal{A} \models p$ *ou* $\mathcal{A} \models q$
   - $\mathcal{A} \models p \rightarrow q$ *, si lorsque* $\mathcal{A} \models p$ *nous avons que* $\mathcal{A} \models q$
   - $\mathcal{A} \models p \leftrightarrow q$ *, si* $\mathcal{A} \models p$ *si et seulement si* $\mathcal{A} \models q$

Définition 13. *La formule propositionnelle* Φ *est dite satisfiable s'il existe au moins une affectation* $\mathcal{A}$ *telle que* $\mathcal{A} \models \Phi$*.*

Définition 14. *La formule propositionnelle* Φ *est dite insatisfiable si elle n'est pas satisfiable.*

Définition 15. *Une tautologie est une formule toujours satisfaite, quelles que soient l'affectation complète choisie.*

Définition 16. *Deux formules p, q sont dites équivalentes, noté $p \equiv q$, si pour toute affectation $\mathcal{A}$ complète pour p et pour q, nous avons que $\mathcal{A} \models p$ si et seulement si $\mathcal{A} \models q$.*

Définition 17. *Une formule p est vide ou contradictoire si elle n'est pas satisfaisable.*

Une formule propositionnelle peut s'écrire sous forme normale disjonctive ou conjonctive. Nous nous intéressons à cette dernière.

1.2 La forme normale conjonctive (FNC)

Définition 18. *Une formule propositionnelle est en forme normale conjonctive, nous utiliserons l'acronyme FNC, si c'est une conjonction de clauses. Une telle formule propositionnelle est écrite sous forme clausales. Seuls les connecteurs $\neg$, $\wedge$ et $\vee$ ainsi que les variables propositionnelles composent une formule FNC.*

Propriété 1. *Une formule propositionnelle en forme normale conjonctive est vraie si et seulement si toutes ses clauses sont vraies. Une formule en forme normale conjonctive peut donc être vue comme un ensemble de clauses devant toutes être vraies pour que la formule soit vraie.*

Le résultat suivant montre l'importance des formes normales conjonctives en logique propositionnelle.

Théorème 1. *Toute formule de la logique propositionnelle admet une forme normale conjonctive qui lui est logiquement équivalente.*

Définition 19. *Un littéral ℓ (resp. $\bar{\ell}$) est dit pur dans une formule en forme forme normale conjonctive si et seulement si $\bar{\ell}$ (resp. ℓ) n'apparaît dans aucune clause de la formule.*

Proposition 1. *Si c_1 et c_2 sont des clauses d'une FNC Φ, telles que $\ell \in c_1$, $\bar{\ell} \in c_2$, la résolvante $(c_1 - \{\ell\}) \cup (c_2 - \{\bar{\ell}\})$ est satisfaite par toute affectation complète $\mathcal{A}$ qui satisfait Φ.*

Démonstration

Écrivons $c_1 = \alpha \vee \ell$ et $c_2 = \beta \vee \bar{\ell}$ où α est une disjonction de littéraux ne contenant pas ℓ et β une disjonction de littéraux ne contenant pas $\bar{\ell}$.

$\mathcal{A}$ doit satisfaire soit ℓ soit $\bar{\ell}$.

- Si $\mathcal{A}$ satisfait ℓ, nous avons que comme $\mathcal{A}$ satisfait c_2, $\mathcal{A}$ doit satisfaire β et donc finalement aussi la résolvante $c_1 - \{\ell\}) \cup (c_2 - \{\bar{\ell}\})$.
- Si $\mathcal{A}$ satisfait $\bar{\ell}$, d'une façon similaire $\mathcal{A}$ satisfait α et donc aussi la résolvante.

Nous allons dans la section suivante montrer comment nous pouvons transformer une formule de la logique propositionnelle en forme normale conjonctive, ce qui donne une démonstration du théorème précédent. Ces transformations sont importantes pour nous, car nous les utiliserons pour traduire notre *problème d'ordonnancement de véhicules* en forme normale conjonctive.

1.2.1 Transformation

Rappelons quelques équivalences logiques usuelles, que nous utilisons pour faire les transformations nécessaires dans le but d'obtenir une formule écrite sous forme normale conjonctive. Soit p et q et r trois formules, nous avons alors :

1. $p \rightarrow q \equiv (\neg p \vee q)$
2. $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$
3. $p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$
4. $\neg(p \vee q) \equiv (\neg p \wedge \neg q)$
5. $\neg(p \wedge q) \equiv (\neg p \vee \neg q)$
6. $(p \wedge q) \vee r \equiv (p \vee r) \wedge (q \vee r)$

7. $p \vee (q \wedge r) \equiv (p \vee q) \wedge (p \vee r)$

8. $\neg\neg p \equiv p$

Ces règles peuvent être utilisées pour transformer une formule en forme normale conjonctive. Premièrement, $p \leftrightarrow q \equiv (p \rightarrow q) \wedge (q \rightarrow p)$ et $p \leftrightarrow q \equiv (p \wedge q) \vee (\neg p \wedge \neg q)$ permettent de remplacer les $\rightarrow$ et $\leftrightarrow$ par les connecteurs $\neg$, $\wedge$ et $\vee$. Deuxièmement, l'application répétée de $\neg(p \vee q) \equiv (\neg p \wedge \neg q)$ et $\neg(p \wedge q) \equiv (\neg p \vee \neg q)$ permet de pousser les négations jusqu'aux variables propositionnelles. Finalement, $(p \wedge q) \vee r \equiv (p \vee r) \wedge (q \vee r)$ permet de pousser un $\vee$ en dessous d'un $\wedge$.

Remplacer une formule par une autre formule équivalente ne change pas la valeur de vérité globale [Cori, Lascar, 2003].

Théorème 2. *Considérons une formule F, une sous-formule G de F et une formule H logiquement équivalente à G. Alors la formule $\grave{F}$, obtenue à partir de F en substituant à la sous-formule G la formule H, est logiquement équivalente à F.*

Nous allons montrer par un exemple comment effectuer cette transformation.

Exemple

Soit p, q, r et s quatres littéraux, nous proposons de transformer les formules suivantes, que nous utilisons plus loin, en forme normale conjonctive .

1. $\mathcal{F}_1$: $(p \rightarrow (q \leftrightarrow \neg r)) \wedge (r \leftrightarrow \neg s)$

   $\mathcal{F}_1 \equiv (\neg p \vee (q \leftrightarrow \neg r) \wedge (r \rightarrow \neg s) \wedge (\neg s \rightarrow r))$

   $\mathcal{F}_1 \equiv (\neg p \vee ((q \rightarrow \neg r) \wedge (\neg r \rightarrow q)) \wedge (r \rightarrow \neg s) \wedge (\neg s \rightarrow r))$

   $\mathcal{F}_1 \equiv (\neg p \vee ((\neg q \vee \neg r) \wedge (r \vee q)) \wedge (\neg r \vee \neg s) \wedge (s \vee r))$

   Notons que : $((\neg q \vee \neg r) \wedge (r \vee q)) \equiv (\neg q \wedge r) \vee (\neg r \wedge q)$

   Nous obtenons donc:

$(\neg p \vee (\neg q \wedge r) \vee (\neg r \wedge q)) \wedge (\neg r \vee \neg s) \wedge (s \vee r)$

Si nous appliquons l'équivalence (7) à $\neg p \vee ((\neg q \vee \neg r) \wedge (r \vee q))$, nous obtiendrons la formule $\mathcal{F}_1$ écrite en FNC.

$\mathcal{F}_1 : (\neg p \vee \neg q \vee \neg r) \wedge (\neg p \vee r \vee q) \wedge (\neg r \vee \neg s) \wedge (s \vee r)$

2. $\mathcal{F}_2 : (\bigvee_{k=1}^{n} \neg(p_k \leftrightarrow q_k)) \rightarrow r$

   $\mathcal{F}_2 \equiv \neg(\bigvee_{k=1}^{n} \neg(p_k \leftrightarrow q_k)) \vee r$, par l'équivalence (1)

   $\mathcal{F}_2 \equiv (\bigwedge_{k=1}^{n}(p_k \leftrightarrow q_k)) \vee r$, par l'équivalence (8)

   $\mathcal{F}_2 \equiv (\bigwedge_{k=1}^{n}((p_k \wedge q_k) \vee (\neg p_k \wedge \neg q_k))) \vee r$, par l'équivalence (3)

   $\mathcal{F}_2 \equiv \bigwedge_{k=1}^{n}(((p_k \wedge q_k) \vee \neg p_k) \wedge ((p_k \wedge q_k) \vee \neg q_k)) \vee r$, par l'équivalence (7)

   $\mathcal{F}_2 \equiv (\bigwedge_{k=1}^{n}((p_k \vee \neg p_k) \wedge (q_k \vee \neg p_k) \wedge (p_k \vee \neg q_k) \wedge (q_k \vee \neg q_k))) \vee r$, par l'équivalence (6)

   $\mathcal{F}_2 \equiv (\bigwedge_{k=1}^{n}((q_k \vee \neg p_k) \wedge (p_k \vee \neg q_k))) \vee r$

   La formule $\mathcal{F}_2$ écrite en FNC.

   $\mathcal{F}_2 : \bigwedge_{k=1}^{n}((\neg p_k \vee q_k \vee r) \wedge (p_k \vee \neg q_k \vee r))$

3. $\mathcal{F}_3 : q \rightarrow \bigvee_{k=1}^{n}(p_k \leftrightarrow r_k)$

   $\mathcal{F}_3 \equiv \neg q \vee (\bigwedge_{k=1}^{n}(p_k \leftrightarrow r_k))$, par l'équivalence (1)

   $\mathcal{F}_3 \equiv \neg q \vee (\bigwedge_{k=1}^{n}((p_k \wedge r_k) \vee (\neg p_k \wedge \neg r_k)))$, par l'équivalence (3)

   $\mathcal{F}_3 \equiv \bigwedge_{k=1}^{n} \neg q \vee (((p_k \wedge r_k) \vee \neg p_k) \wedge ((p_k \wedge r_k) \vee \neg r_k))$, par l'équivalence (7)

   $\mathcal{F}_3 \equiv \bigwedge_{k=1}^{n} \neg q \vee (((p_k \vee \neg p_k) \wedge (r_k \vee \neg p_k)) \wedge ((p_k \vee \neg r_k) \wedge (r_k \vee \neg r_k))$, par l'équivalence (6)

   $\mathcal{F}_3 \equiv \bigwedge_{k=1}^{n} \neg q \vee ((\neg p_k \vee r_k) \wedge (p_k \vee \neg r_k))$, par l'équivalence (7)

   nous obtenons la formule $\mathcal{F}_3$ écrite en FNC.

   $\mathcal{F}_3 : \bigwedge_{k=1}^{n}((\neg p_k \vee \neg q \vee r_k) \wedge (p_k \vee \neg q \vee \neg r_k))$

1.3 Le problème SAT

Une attention particulière sera accordée au problème de décision (SAT), que nous définissons maitenant et qui est lié à notre méthode de résolution pour le *problème d'ordonnancement de véhicules.*

Le problème de satisfaction d'une formule de la logique propositionnelle, désigné par SAT, consiste à vérifier si une formule propositionnelle écrite sous forme clausales est satisfaisable ou pas.

La logique propositionnelle permet de définir plusieurs autres problèmes : Le roblème d'optimisation (MAX-SAT) qui est de déterminer une affectation qui maximise le nombre de clauses satisfaites d'une formule en forme normale conjonctive, ou encore de problème de comptage (#SAT) qui est de déterminer le nombre d'affectations qui satisfont une formule en FNC. Le problème de la satisfiabilité maximale (MAX-SAT) correspond à minimiser le nombre de clauses fausses. En effet, pour une formule propositionnelle écrite en FNC, il s'agit de trouver une affectation maximisant le nombre de clauses vraies. Quant au problème #SAT, il correspond à compter le nombre de modèles d'une formule, [Tompkins, Hoos, 2004].

Le problème SAT [Gent, Walsh, 1993] occupe également une place importante parmi les problèmes NP-complets [Cook, 1971]. Il fait partie à ce titre des six problèmes fondamentaux NP-complets à savoir: la couverture de sommets, le circuit hamiltonien, la clique, le partitionnement de graphes et le partitionnement tridimensionnel.

De plus, plusieurs problèmes connexes sont actuellement traités par transformation en SAT, comme par exemple certains problèmes de satisfaction de contraintes ou certains problèmes de programmation linéaire en nombres entiers booléens. Dans ce cas, les solutions basées sur SAT peuvent donner des résultats intéressants.

1.4 Conclusion

Dans ce chapitre, nous avons étudié les concepts de bases de la logique propositionnelle, utiles dans notre étude. En effet, la logique propositionnelle constitue la base de toutes les approches logiques développées. Elle se trouve aussi être un formalisme très utile pour représenter de nombreux problèmes en informatique. En outre, nous avons montré comment une formule propositionnelle peut être transformée en forme normale conjonctive. Donc nous pouvons souvent nous réduire à ne considérer que de telles formules.

Nous avons également décrit le problème de décision SAT, qui consiste à vérifier si une formule écrite sous forme normale conjonctive est satisfaisable ou non.

Bien que le problème SAT appartienne à la classe des problèmes NP-complets, réputé être des problèmes difficiles, de nombreuses méthodes de résolution ont été proposées et fournissent par ailleurs des résultats souvent satisfaisants dans la pratique. Nous nous contentons de présenter dans le suivant chapitre l'algorithme le plus appliqué en pratique, soit l'algorithme DPLL pour le problème SAT.

CHAPITRE II

ALGORITHMES DPLL POUR SAT

Ce présent chapitre est consacré à la présentation de quelques algorithmes et d'heuristiques, qui à ce jour, ont permis d'obtenir des résultats remarquables pour le problème SAT.

Nous commençons par quelques définitions nécessaires aux méthodes de résolutions que nous présentons ensuite, suivi par la présentation de quelques solutionners (logiciels)pour le problème SAT. Nous présentons également les TL-clauses qui permettent d'appliquer l'algorithme DPLL aux expressions arithmétiques.

2.1 Définitions

Définition 20. *Une instance de SAT est définie par un ensemble de variables propositionnelles.* $\chi = \{x_1, ..., x_n\}$, *et un ensemble* $C = \{c_1, ..., c_m\}$ *de clauses.*

Définition 21. *Une instanciation est une affectation de valeur v (0/1) à une des variables* x_i. *L'instanciation est alors considérée consistante, si elle ne viole aucune des clauses du problème. Une sous-séquence d'instanciation ou une instanciation partielle de taille i est une instanciation consistante de i variables de l'ensemble* χ. *Une instanciation est considérée complète si elle porte sur toutes les variables de* χ, *et dans le cas contraire, elle est considérée comme partielle.*

Définition 22. *Une solution est une instanciation complète consistante. En effet, une solution consiste à une affectation des variables, telle que toutes les clauses soient satisfaites (au moins un littéral par clause a la valeur vrai).*

Définition 23. *Un conflit est un cas où l'affectation d'une valeur v à la variable x_i viole une des clauses c du problème. Cela se produit lorsqu'une affectation partielle ne peut pas être prolongée à une affectution complète satisfaisant toutes les clauses.*

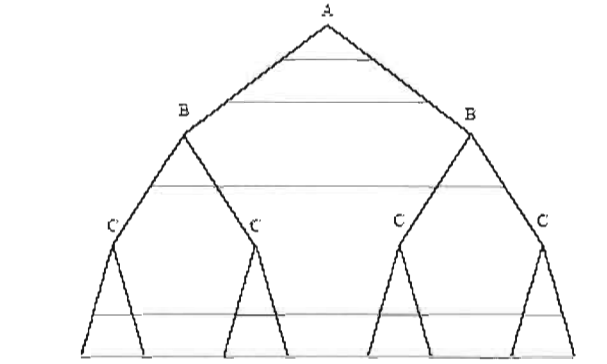
Après avoir présenté les définitions, nous présentons les méthodes de résolution.

2.2 Méthodes énumératives pour SAT

Actuellement, il existe plusieurs méthodes pour résoudre les problèmes de type SAT. Nous les regroupons en deux catégories : les méthodes incomplètes, basées sur le principe de « générer puis tester », et les méthodes complètes, dites énumératives. Les méthodes incomplètes ont l'avantage de souvent pouvoir trouver rapidement une solution quand elle existe, mais leur inconvénient majeur est de ne pas pouvoir démontrer l'inconsistance. Nous ne nous intéressons dans ce chapitre qu'aux méthodes complètes, qui sont celles que nous allons appliquées.

Quand nous utilisons les méthodes complètes, cela ne veut pas dire que nous explorons toutes les affectations possibles. En effet, le nombre d'affectations pour une instance comportant n variables est 2^n. Pour réduire le nombre d'affectations à tester, les méthodes exactes génèrent les affectations en traitant les variables l'une après l'autre. Aussitôt qu'une affectation partielle engendre une clause fausse pour le problème SAT alors il n'est plus nécessaire de continuer à compléter cette affectation partielle. Ce processus permet la réduction du nombre d'affectations évaluées, mais ne permet toutefois pas de travailler sur des instances de grande taille. Une représentation courante de ce processus se fait sous la forme d'un arbre binaire présentée à la figure 2.1. Chaque sommet représente une variable et le fils gauche (resp. droit) d'un sommet indique l'assignation de la variable représentée par le sommet à V (resp. F). Une feuille de cette arbre représente donc une affectation.

Le tableau de la figure 2.1 donne en effet les 8 (2^3) affectations possibles, la lecture du tableau se fait en colonne.



A	V	V	V	V	F	F	F	F
B	V	V	F	F	V	V	F	F
C	V	F	V	F	V	F	V	F

Figure 2.1 Arbre binaire représentant une affectation avec 3 variables

Il est bien entendu que l'ordre dans lequel les variables sont choisies occupe un rôle très important, car il permet de couper des branches plus tôt. En outre, à chaque étape le choix d'une variable peut être quelconque parmi les variables non affectées, donc des heuristiques peuvent être utilisées. De même, des mécanismes telles que la propagation unitaire ou le calcul de borne inférieure permettent de couper les branches plus rapidement en anticipant sur les prochaines évaluations.

2.2.1 Davis-Putnam (DP)

Davis et Putnam sont les premiers à avoir proposé une méthode exacte. Le principe de cette dernière est assez simple, il est présenté par l'algorithme de la figure 2.2. Prenons deux clauses c_1 et c_2 de la formule Φ telles que $\ell \in c_1$ et $\bar{\ell} \in c_2$, nous ajoutons la résolvante $c_1 - \{\ell\} \cup c_2 - \{\bar{\ell}\}$. Si cette clause est une tautologie, il n'est pas nécessaire de l'ajouter. Après l'ajout de toutes les résolvantes en ℓ, toutes les clauses contenant ℓ sont supprimées. Davis et Putnam ont démontré que la formule Φ est insatisfiable si et seulement si, une clause vide (c'est-à-dire fausse) apparaît après que toutes les résolutions possibles aient été effectuées [Davis, Putnam, 1960].

Procedure 1. *DP(Φ)*

Donnees : une formule Φ en FNC

Resultat : satisfiabilite de Φ

debut

pour chaque variable ℓ apparaissant dans Φ

faire

Resolvantes $= \emptyset$;

pour chaque couple de clauses (c_1, c_2) *de* $\mathcal{C}_\Phi$

faire

si $\ell \in c_1$ *et* $\bar{\ell} \in c_2$ *alors*

$c = c_1 - \{\ell\} \cup c_2 - \{\bar{\ell}\}$; « *creation d'une nouvelle clause* »

si c n'est pas une tautologie alors

Resolvantes $=$ *Resolvantes* $\cup$ *c*; « *ajouter a c l'ensemble des resolvantes en* ℓ »

fin

fin

fin

$Clauses_\ell = \{\gamma \in \mathcal{C}_\Phi \mid \ell \in \gamma \text{ ou } \bar{\ell} \in \gamma\}$; « *ensemble de clauses contenant la variable* ℓ »

$\Phi = \Phi$ - $Clauses_\ell$;

$\Phi = \Phi \cup$ *Resolvantes*;

fin

si Φ contient une clause vide alors

retourner "insatisfiable";

sinon retourner "satisfiable";

fin

Figure 2.2 Algorithme de Davis-Putnam

2.2.2 Davis Putnam Logemann Loveland (DPLL)

Le principal problème de l'algorithme DP est de produire un nombre énorme de résolvantes, ce qui limite passablement la taille des FNC qui peuvent être utilisées. Une seconde difficulté est que cette méthode détermine l'existence d'une solution sans la donner explicitement. Pour palier à ces difficultés, Davis, Logemann et Loveland ont créé l'algorithme DPLL. Ce dernier remplace le mécanisme de résolution de l'algorithme DP par la séparation du problème en deux sous problèmes. En effet, l'idée est de construire un arbre binaire dans lequel chaque sommet représente un appel récursif à la procédure DPLL. Pour les feuilles, elles représentent l'arrivée de la procédure à une contradiction lors de la recherche en trouvant une clause vide, ou encore la satisfiablité d'une formule [Li, Anbulagan, 1997], [Davis et al., 1962].

Cette recherche utilise un retour-arrierè (Backtracking) que nous présenterons plus en détail dans la section 2.3, afin de remonter dans l'arbre de recherche et ainsi tester de nouvelles branches. La procédure DPLL est détaillée dans l'algorithme de la figure 2.3.

L'algorithme que nous venons d'énoncer utilise la propagation unitaire, présentée par l'algorithme de la figure 2.4, qui consiste à l'utilisation des clauses unitaires. Pour le problème SAT, l'unique littéral de cette clause doit être rendu vrai, car dans le cas contraire, une clause fausse apparaît. La simplification de la formule devient donc aisée et ceci en utilisant la règle du littéral unique [Loveland, 1978]. Quand une variable ℓ est mise à vrai (resp. faux) alors toutes les clauses contenant le littéral ℓ (resp. $\bar{\ell}$) sont supprimées et toutes les occurrences du littéral $\bar{\ell}$ (resp. ℓ) sont retirées des clauses les contenant. Cette méthode permet la simplification de la formule, comme il a été montré par l'algorithme de la figure 2.5. La propagation unitaire utilise cette règle sur les variables des clauses unitaires jusqu'à épuisement des simplifications possible.

Plusieurs variantes d'algorithme DPLL ont été proposées, la différence entre chacune d'elles demeure dans le choix de l'heuristique permettant de sélectionner une variable à étudier, dans la méthode de retour-arrière (backtracking) ainsi que dans les structures de

Procedure 2. *DPLL(Φ)*

Donnees : une formule Φ en FNC

Resultat : satisfiabilite de Φ et une solution ℓ saisfaisable

debut

si $\Phi = \emptyset$

retourner "satisfiable"

fin

Φ = PropagationUnitaire(Φ);

si Φ contient une clause vide alors

retourner "insatisfiable";

fin

Selection d'une variable ℓ dans $\mathcal{C}_\Phi$ grâce a l'heuristique utilisee;

si DPLL($\Phi \cup \{\ell\}$) retourne satisfiable alors

retourner "satisfiable"

sinon retourner DPLL($\Phi \cup \{\bar{\ell}\}$);

fin

fin

Figure 2.3 Algorithme de Davis Putnam Logemann Loveland

Procedure 3. *PropagationUnitaire*

Donnees : une formule Φ *en FNC*

Resultat : Φ *simplifiee*

debut

tant que il n'y a pas de clause vide et qu'une clause unitaire c existe dans Φ *faire*

Considerer c une clause unitaire et ℓ *son litteral non-affecte;*

Φ = *Simplifier(*Φ, ℓ*);*

fin

Retourner Φ

fin

Figure 2.4 Algorithme de Propagation Unitaire

Procedure 4. *Simplifier(*Φ,ℓ*)*

Donnees : une formule Φ *en FNC, un litteral* ℓ

Resultat : Φ *simplifiee*

debut

pour chaque clause c de Φ

faire

si ($\ell \in c$)*alors* $\Phi = \Phi$-$\{c\}$;

sinon si ($\bar{\ell} \in c$) *alors c=c-*ℓ;

fin

fin

Retourner Φ

fin

Figure 2.5 Algorithme Simplifier

données utilisées. Nous nous proposons d'en présenter quelques unes dans la prochaine section.

2.2.3 Heuristiques de branchement

Le choix de l'heuristique pour l'algorithme DPLL est d'une importance capitale. En effet, un mauvais choix peut conduire à une exploration totale de l'arbre de recherche alors qu'un meilleur choix permettra de tronquer certaines branches, et dans le meilleur des cas nous ne parcourons qu'une unique branche. Nous décrivons dans ce qui suit les trois heuristiques les plus utilisées [Audemard, Benhamou, Siegel, 1999], [Li, Anbulagan, 1997] et [Hooker et Vinay, 1995].

- Heuristique $MOMS$ (Maximum Occurrences in clauses of Minimum Size), maximum d'occurrences dans les clauses de tailles minimales qui est une des heuristiques les plus simples. Elle sélectionne la variable ayant le plus d'occurrences dans les clauses de la plus petite taille. Ce choix est motivé par le fait qu'il favorise la propagation unitaire.

  Exemple 1

  Soit la formule $\Phi : (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_4 \wedge \neg x_3) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee x_4 \vee x_6) \wedge (\neg x_2 \vee x_5 \vee x_6) \wedge (x_2 \vee \neg x_5 \vee x_3)$

  Les clauses de plus petite taille étant $(\neg x_1 \vee x_2)$ et $(x_1 \vee \neg x_5)$ alors l'heuristique MOMS choisira la variable x_1.

- Heuristique JW (Jeroslow-Wang), cette heuristique est basée sur le même principe que celle de MOMS. Elle favorise la sélection des variables apparaissant dans les clauses de petites tailles. En outre, la possibilité qu'une variable soit sélectionnée par JW est inversement proportionnelle à la taille des clauses dans lesquelles elle apparaît. JW utilise une fonction J qui prend en entrée un littéral ℓ et retourne un poids pour ce littéral comme suit :

  $J(\ell) = \sum_{c \in \mathcal{C}_\Phi / \ell \in c} 2^{-|c|}$

où $|c|$ est la taille de la clause c appartenant à l'ensemble $\mathcal{C}_\Phi$. La variable x sélectionnée par l'heuristique JW est celle qui maximise $J(x) + J(\neg x)$.

Exemple 2

Soit la formule $\Phi : (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee x_4 \wedge \neg x_3) \wedge (x_1 \vee \neg x_5) \wedge (x_2 \vee x_4 \vee x_6) \wedge (\neg x_2 \vee x_5 \vee x_6) \wedge (x_2 \vee \neg x_5 \vee x_3)$

Avec la fonction J, nous obtenons :

$J(x_1)= 0.25$, $J(\neg x_1)= 0.25$, $J(x_2)= 0.5$, $J(\neg x_2)= 0.25$, $J(x_3)= 0.125$, $J(\neg x_3)= 0.125$, $J(x_4) = 0.25$, $J(\neg x_4)= 0$, $J(x_5)= 0.125$, $J(\neg x_5) = 0.375$, $J(x_6)= 0.25$, $J(\neg x_6)= 0$. La variable x_2 avec $J(x_2) + J(\neg x_2) = 0.75$ est sélectionnée par l'heuristique JW .

- Heuristique UP (Unit Propagation), les heuristiques $MOMS$ et JW sélectionnent une variable en fonction de la situation courante lors de la recherche. Il est donc plus intéressant de sélectionner une variable en prévoyant à l'avance son influence sur l'évolution de la recherche. L'heuristique UP est basée sur ce principe. En effet, elle prévoit l'évolution de la formule si une variable particulière est sélectionnée. L'influence de la variable est définie par un poids de la variable x, noté $\mathcal{W}$, effectué par la procédure de propagation unitaire.

  Pour chaque variable non instanciée x de la formule Φ étudiée, nous ajoutons la clause unitaire (c) et $(\neg c)$ à $\mathcal{C}_\Phi$ et nous effectuons indépendamment les deux propagations unitaires ce qui permet de calculer le poids de x comme suit :

  $\mathcal{W}(x) = \sum_{c \in \mathcal{C}_\Phi / \neg x \in c} 5^{-|c|}$ et $\mathcal{W}(\neg x) = \sum_{c \in \mathcal{C}_\Phi / x \in c} 5^{-|c|}$

  où $|c|$ est la taille de la clause c appartenant à l'ensemble $\mathcal{C}_\Phi$.

  Dans les exemples déjà présentés pour les heuristiques MOMS et JW, la variable sélectionnée par l'heuristique UP sera x_1 ou x_2.

2.3 Retour-arrière (Backtracking) chronologique (CBT)

L'algorithme DPLL procède à l'affectation au fur et à mesure d'une valeur booléenne à une variable. Il vérifie à chaque étape si une clause vide (contradiction) est produite par les propagations unitaires. S'il arrive qu'après l'affectation d'une variable x_{n-1}, l'algorithme ne puisse trouver de valeur pour x_n qui satisfasse les clauses (quel que soit la valeur affectée à x_n, la propagation unitaire mène à une contradiction), nous parlons alors d'une situation d'échec. A ce stade, le CBT revient en arrière, à la dernière variable affectée soit x_{n-1}. Alors, il modifie la valeur de x_{n-1} tout en espérant qu'il pourra affecter la variable x_n. S'il n'y a toujours pas de solutions pour x_n, il recule et modifie la variable précédente x_{n-2}, et s'il n'existe toujours pas de solution, il reculera jusqu'à x_0, dans le cas contraire il trouve obligatoirement la solution.

Autrement dit, l'algorithme cherche à prolonger progressivement une solution partielle S en instanciant, à chaque pas, une nouvelle variable x_i à une valeur booléenne. Seules les contraintes portant uniquement sur des variables déjà instanciées sont testées. Si la nouvelle instanciation partielle ainsi obtenue n'est pas une solution partielle, nous effectuons un retour-arrière sur la variable x_i, c'est à dire nous procédons à un retour arrière vers la variable précédente de x_i, s'il y en a une. Dans le cas contraire, le problème est insatisfiable. Il s'agit donc d'une recherche arborescente où les sommets de l'arbre sont des instanciations partielles, la racine est l'instanciation vide et les feuilles sont soit une (des) solution(s) complète(s), soit des instanciations ne satisfaisant pas la totalité des contraintes [Ginsberg, 1993].

Néanmoins, l'insatifiabilité est souvent détectée tôt dans le cas où un sous-ensemble χ de variables est instancié, ce qui permet au CBT de le supprimer de l'espace de recherche.

Le CBT est la méthode de résolution de base la plus répandue pour la résolution de contrainte et elle peut être adaptée de différentes façons [Dechter, Frost, 2002].

2.4 Retour ponctuel (Backjumping) (BJ)

Comme nous venons de le voir dans la précédente section, lors de la rencontre d'échec, l'algorithme CBT remet en cause son dernier choix et essaie alors une nouvelle valeur pour la variable courante x. Lorsque toutes les valeurs ont été utilisées, il revient en arrière sur la variable précédente. Néanmoins, rien ne garantit que cette variable soit en cause dans les échecs rencontrés lors de l'affectation de x. Par exemple, si cette variable n'est pas liée à x par une chaîne de clauses, elle ne peut être la cause de ces échecs. Donc essayer de nouvelles valeurs pour cette variable nous mènera aux mêmes échecs au niveau de l'affectation de x. Pour contrer ce principal problème du CBT, une méthode de retour arrière non chronologique, le retour ponctuel, a été développée.

La méthode BJ se distingue du CBT par le fait que dans le cas où toutes les extensions d'affectation courante avec une valeur de x sont inconsistantes, l'algorithme BJ revient en arrière sur la dernière variable affectée dont la valeur est en conflit avec une valeur de x plutôt que d'effectuer un retour arrière vers la $i-1^{ieme}$ variable affectée. De ce fait, l'algorithme BJ effectue directement un retour arrière sur la variable dont l'instanciation responsable du conflit. Son objectif principal reste l'accélération du processus de retour arrière. En effet, l'aspect du retour arrière non chronologique consiste à analyser les causes des échecs avec la possibilité de conserver les raisons de l'inconsistance, sous la forme de nouvelles contraintes, [Dechter, Frost, 2002].

Autrement dit, chaque variable affectée par une propagation unitaire a une raison qui est la clause utilisée pour cette propagation. Lorsque nous avons un conflit, nous utilisons la résolution sur la clause fausse et la dernière raison. Nous recommenons ainsi avec la clause obtenue et la raison précédente jusqu'à arriver à une clause c ne contenant plus de variables affectées au niveau courant (c-à-d depuis le dernier choix). Comme cette clause c est obtenue par résolution c'est une conséquence de la FNC. De plus par construction, tous les littéraux de cette clause sont faux. Finalement aucune assignation ne pourra rendre cette clause vraie tant qu'aucun de ses littéraux ne changera de valeur. Donc si nous retournons au dernier choix apparaissant dans cette clause nous n'éviterons

aucune solution.

2.5 L'apprentissage (Learning)

Le BJ est plus efficace que le CBT, cependant il ne peut pas écarter la redondance de la recherche. Les approches basées sur ce principe sont regroupées sous la dénomination algorithme d'apprentissage.

Les méthodes d'apprentissage (learning) peuvent favoriser la recherche et permettent de trouver plus rapidement les solutions, tout en évitant de retomber sans cesse dans les mêmes situations d'inconsistance.

L'apprentissage dans ce cas vise à éliminer le plus possible de branches dans l'arbre de résolution. Pour ce faire, le but principal est d'identifier les ensembles de conflits minimaux et les enregistrer sous la forme de nouvelles contraintes. Lors de la détection d'une inconsistance, nous ajouterons une contrainte qui facilitera l'identication de l'inconsistance présente dans le graphe.

Le Learning agit en identifiant, à chaque échec rencontré, la cause de l'échec, comme c'est la cas du BJ. Cette cause de l'échec provient des variables dont dépend la dernière variable considérée. Nous pouvons ajouter la clause c, clause ne contenant plus de variables affectées au niveau courant à l'ensemble des clauses et que périodiquement il faut éliminer ces clauses pour ne pas trop en avoir.

Par ailleurs, il existe deux formes d'apprentissage soit en profondeur ou en surface. L'analyse d'une situation d'inconsistance pour identifier les ensembles de conflit minimaux demeure une tâche difficile. C'est pourquoi, nous préférerons parfois utiliser des méthodes d'apprentissage en surface comme le $value-based\ learning$, le $graph-based\ shallow\ learning$ ou encore le *retourponctuel learning*. Le principe de l'algorithme $value-based\ learning$ consiste à enregistrer, selon l'inconsistance détectée, l'ensemble de variables affectées ainsi que leurs valeurs respectives. Cet enregistrement d'information se fait dépendamment des contraintes du problème. En effet, nous enregistrons unique-

ment les couples (variable, valeur) qui entrent en conflit avec la feuille de la situation d'inconsistance, et ceci sous forme de nouvelles contraintes. Pour le *Graph – based shallow learning*, l'algorithme utilise la structure du graphe afin d'identifier les inconsistances et les enregistrer. Lors de la détection d'une feuille de situation d'inconsistance tel que l'instanciation des variables $x_1,\ldots,x_i$ est inconsistante avec la variable x_{i+1}, l'algorithme enregistre le tout sous forme de nouvelles contraintes pour éviter que la situation ne se reproduise. Quant à *retourponctuel learning*, il utilise, les informations ramassées pendant la phase de retour en arrière du BJ et ceci pour enregistrer de nouvelles contraintes. Ces méthodes d'apprentissage en surface permettent l'enregistrement d'ensembles de conflits non minimaux, [Frost, Dechter, 1994].

Les méthodes précédemment décrites sont utilisées dans différents solutionners, notamment ceux pour le problème SAT que nous abordons maintenant.

2.6 Quelques solutionners pour le problème SAT

Plusieurs algorithmes exacts ont été réalisés pour solutionner le problème SAT. Ils possèdent chacun des heuristiques ou des pré–traitements différents, ce qui leur donne la possibilité d'obtenir de bons résultats. Nous présenterons ceux qui, à ce jour, ont pu obtenir des résultats pertinents.

- *Satz* [Li et Anbulagan, 1997]: L'algorithme *Satz* s'appui sur l'algorithme DPLL avec UP comme heuristique.

- *EqSatz* [Li, 2000 a]: EqSatz est un algorithme similaire à Satz. Cependant, il inclut un mécanisme qui lui permet de traiter particulièrement les clauses induisant des équivalences logiques.

- *Chaff* [Moskewicz et al., 2001]: L'algorithme Chaff est très performant. Il a été primé dans plusieurs compétitions SAT. Ses deux principales caractéristiques restent sa propagation unitaire plus rapide et une nouvelle heuristique $VSIDS$ (Variable State Independent Decaying Sum) qu'il utilise. *Chaff* arrive a de très

bons résultats, de nombreuses implémentations se sont inspirés de lui telle que *ZChaff* [Moskewicz et al., 2001], *Satzoo* [Eén et Sorensson, 2003], etc.

- *SATO* (Satisfiability Testing Optimized) [ZHANG, 1997], [Zhang, Stickel, 2000]: il a été conçus aux fins de résoudre les problèmes complexes tel que le carré latin. Sa puissance lui donne l'avantage de résoudre beaucoup d'autres familles de problèmes SAT. Ses deux principaux atouts sont son utilisation d'une nouvelle heuristique et une nouvelle manière de gérer les clauses vides.

  Les auteurs se sont basés sur la structure de données à utiliser pour coder efficacement les clauses. Elles sont codées sous forme d'arbres. Chaque sommet de l'arbre peut être soit :

  1. l'arbre vide, noté par Nil.
  2. la marque d'obtention de clause vide, notée $\square$.
  3. un quadruplet $< v, P, N, R >$ où v représente l'index d'une variable, P un arbre représentant les clauses où v apparaît positivement, N un arbre représentant les clauses où v apparaît négativement et R un arbre représentant les clauses où v n'apparaît pas.

  L'exemple 3 montre une telle structure.

  Exemple 3

  Soit l'ensemble de clauses $\{(x_1 \vee x_2), (\neg x_1 \vee \neg x_2)\}$. L'arbre codant les clauses est explicité à la figure 2.6.

L'algorithme SATO dispose d'une procédure qui fusionne deux arbres et son association avec la structure de données utilisée pour coder efficacement les clauses fournit les avantages suivants :

- élimination des doubles clauses pendant la construction de l'arbre;

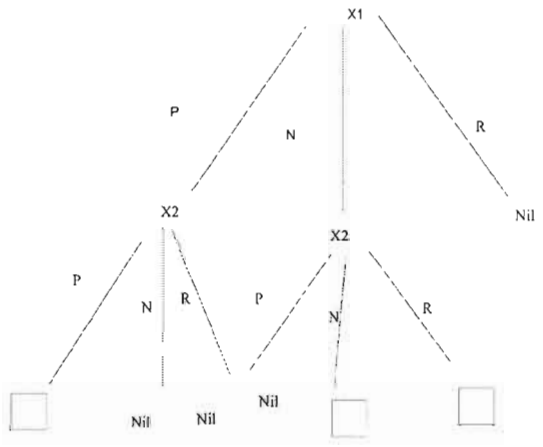


Figure 2.6 Exemple d'arbre de clauses avec SATO

- élimination des clauses sous-sommées pendant la construction de l'arbre (c.-à-d. des clauses qui contiennent une autre clause comme préfixe);
- rapidité de calcul de certaines résolvantes;
- propagation unitaire efficace, vue que les variables sont triées, un seul parcours de l'arbre est nécessaire pour toute la phase de la propagation. La suppression des clauses satisfaites par un littéral ℓ, se fait en mettant à Nil les sous-arbres correspondant à ℓ vrai et la recherche des nouveaux mono-littéraux se fait uniquement dans les sous-arbres correspondant au littéral $\neg\ell$.

De plus SATO intègre les TL-Clauses que nous définissons dans ce qui suit.

2.7 Les TL-Clauses (True Literal)

Pour une bonne compréhension des TL-clauses, nous mettons en évidence leurs avantages, nous décrivons également la particularité de SATO en utilisant ces dernières et enfin, nous montrons comment SATO résout les conflits aux TL-clauses.

Définition 24. *Une TL-clause sur un ensemble de littéraux est une contrainte sur le nombre de ses littéraux vrais. En fait, nous définissons pour c un ensemble de littéraux, TL(c) le nombre des littéraux de c qui sont vrais pour une certaine affectation. Donc si la clause c est formée de n littéraux booléens ℓ_i, $i = 1, .., n$, alors $TL(c) = \ell_1 + \ell_2 + \cdots + \ell_n$.*

Les TL-clauses sont les contraintes de la forme $TL(c) = k$, $TL(c) \leq k$ et $TL(c) \geq k$.

2.7.1 Motivation

Remarquons tout d'abord qu'une clause $\ell_1 \vee \ell_2 \vee \cdots \vee \ell_n$ est équivalente à la TL-clause $\ell_1 + \ell_2 + \cdots + \ell_n \geq 1$, donc les TL-clauses sont bien une généralisation des clauses. De plus, Zhang et al. [2004] confirment qu'un bon nombre de contraintes des problèmes relatifs à la planification sont exprimables par le par le biais des TL-clauses (*True-Literal*). En effet, les TL-clauses permettent une réduction significative du nombre de contraintes nécessaires pour représenter des contraintes arithmétiques. Finalement, elles permettent aussi la résolution rapide du problème par rapport à l'utilisation des clauses ordinaires.

Aloul et al. [2002] avaient déjà utilisé en même temps des contraintes causales et des contraintes arithmétiques, que nous avons appelé FNC/TL, ces derniérs auteurs avaient aussi combiné un solver de programmation linéaire, pour les variables booléennes avec un solver SAT, en opérant de grandes améliorations d'exécution pour plusieurs problèmes. Les contraintes dans leur Integer Linear Program (ILP) solutionners sont de la forme $\sum_{i=1}^{n} a_i x_i \leq b$, ou a_i, b sont des entiers et x_i des variables booléennes. Si pour tout i, $a_i = 1$, la même contrainte peut être exprimée comme $TL(x_1 \vee x_2 \vee ... \vee x_n) \leq b$. Tout compte fait, les TL-clauses sont considérées comme un cas spécial de ces contraintes.

2.7.2 Particularité de SATO

Parmi les solutionners de problemès SAT, SATO, qui a la particularité de pouvoir travailler directement avec des contraintes exprimées sous forme de TL-clauses, SATO considère pour chaque clause c, la fonction $TL(c)$.

De plus il accepte les clauses exprimées dans le format FNC de DIMACS [Zhang et al., 2004], en plus des TL-clauses. En effet, en format DIMACS un littéral est représenté par un nombre entier différent de zéro et une clause est représentée par une liste de nombre entiers suivis d'un zéro pour marquer sa fin. SATO accepte des TL-clauses est représentées également par une liste de nombre entiers suivis d'un symbole de l'ensemble $\{<, <=, >, >=\}$ et d'un nombre entier. Les TL-clauses et les clauses ordinaires peuvent être insérées dans n'importe quel ordre dans le fichier d'entrée. En outre, lors du prétraitement, la TL-clause est convertie en une ou deux TL-clauses de la forme $TL(c) \leq k$ et k est enregistré dans la structure de données représentant la clause. Nous illustrons ce format par l'exemple 4.

Exemple 4

Nous exprimons la FNC/TL suivante en format DIMACS étendu.

$x_1 \vee x_2$

$-x_1 \vee x_3 \vee x_4$

$x_2 \vee \neg x_3 \vee x_4$

$x_1 + x_2 + x_4 \leq 2$

Pour cela, nous devons préparer un fichier dont le contenu est comme suit :

c La ligne ci-dessous est l'en-tête du format DIMACS, qui exprime qu'il s'agit d'un problème avec 4 variables et 4 contraintes.

p FNC/TL 4 4

1 2 0

-1 3 4 0

2 -3 4 0

$1 + 2 + 4 \leq 2$

2.7.3 Résolution des conflits aux TL-clauses

Pour satisfaire une TL-clause comme $TL(c) \leq k$, le nombre de littéraux vrais dans c, où c est un ensemble de n littéraux booléens, ne doit pas dépasser la valeur k. Afin de vérifier cette condition, SATO ajoute un compteur associée à une TL-clause c, appelé $TLcount(c)$, initialisé à k. Cette valeur est diminuée de 1 dès qu'un littéral de la clause c devient vrai. Si $TLcount(c)$ devient égal à 0, la propagation est effectuée pour chaque littéral non affecté de c. En effet, un tel littéral doit être faux si nous voulons que cette TL-clause soit vraie. Par contre, si $TLcount(c)$ devient inférieur à 0, la propagation se termine et un conflit est déclaré. La résolution d'un tel conflit, consiste à remplacer la TL-clause touchée par ce conflit par la négation des littéraux ayant la valeur vraie et la négation du littéral responsable du conflit. Nous illustrons ce principe par l'exemple suivant.

Exemple 5

Soit la TL-clause suivante : $TL(x \vee y \vee z \vee w) \leq 2$ et deux clauses binaires : $z \vee u$ et $\neg u \vee w$. Durant la recherche, nous affectons à x et à y la valeur 1, donc z et w prennent forcément la valeur 0, du fait que $TL(x \vee y \vee z \vee w) \leq 2$. Si la clause $z \vee u$ est traitée en premier, nous aurons $u = 1$ et donc nous aurons un conflit au niveau de la clause binaire $\neg u \vee w$, comme il est montré à la figure 2.7. Pour régler ce conflit, nous traçons les ancêtres de chaque littéral dans la clause qui est responsable de ce conflit (u et w). Donc de u, nous aurons $z \vee u$, et de w, nous aurons $TL(x \vee y \vee z \vee w) \leq 2$.

Nous prenons uniquement les littéraux qui valent 1 (x et y). En d'autres termes, l'ancêtre de w est traité comme $(\neg x \vee \neg y \vee \neg w)$, de même, l'ancêtre de z est traité comme $(\neg x \vee \neg y \vee \neg z)$.

La solution, comme il a été montré pà la figure 2.8 est $x = 0$, $y = 0$, $w = 1$, $z = 1$ et $u = 1$.

Il est affirmé dans cette approche qu'une optimisation est faite et ceci pour éviter à réincrémenter k lorsque nous faisons un retour arrière. Pour être plus rapide, le calcul

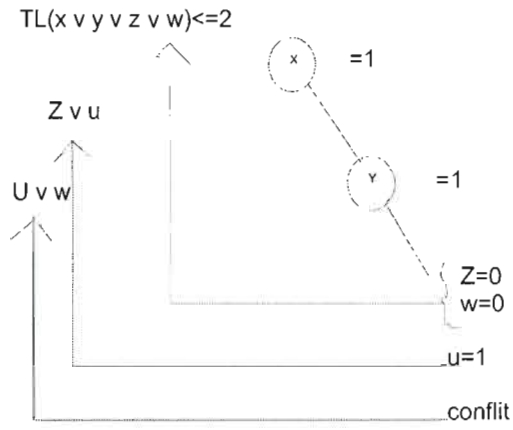


Figure 2.7 Exemple d'une TL-clause

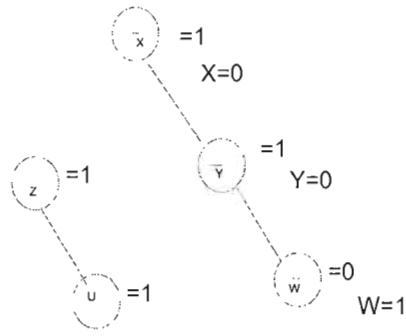


Figure 2.8 Solution de l'exemple

du nombre de littéraux vrais est plutôt fait lorsque $k = 0$. Dans le cas ou k est vraiment égal à 0 il y a propagation, sinon une correction de k est faite.

2.8 Conclusion

Nous avons étudié dans ce chapitre, les algorithmes complets pour la résolution pratique du problème SAT. Ces derniers sont basés sur la procédure de Davis et Putnam et plus précisément sur la version proposée par Davis, Logemann et Loveland connue sous le nom DPLL. En effet, la procédure DPLL est un algorithme énumératif dont le but est de générer un arbre de recherche. Deux points importants renforcent cet algorithme, qui sont la fonction de simplification de la formule et celle du choix de la prochaine variable à affecter. Donc c'est bien grâce à ces améliorations qu'il est possible d'éviter certaines redondances dans la recherche effectuée.

Nous avons décrit également l'algorithme DPLL étendue sur les TL-clauses, qui représente une nouvelle méthode capable de résoudre n'importe quelle contrainte exprimée sous forme de clause ordinaire ou contrainte arithmétique, appelée TL-clause. En effet, dans cette nouvelle approche une optimisation est faite. Elle permet une réduction considérable du nombre de contraintes, nécessaires par apport à la forme normale conjonctive. Ceci permet de traiter des instances de taille raisonnable et de souvent les résoudre dans un temps raisonnable.

Les méthodes exposées dans ce chapitre ont été utilisées pour résoudre de différents problèmes tel que l'affectation, planification ou encore l'ordonnancent. Dans le chapitre suivant, nous l'appliquons au *problème d'ordonnancement de véhicules*, sujet de notre recherche.

CHAPITRE III

PROBLÉMATIQUE

Dans ce chapitre, nous décrivons le *problème d'ordonnancement de véhicules* ayant fait l'objet du challenge ROADEF'2005[1] (Société Française de Recherche Opérationnelle et d'Aide à la Décision), suivie d'une mise en contexte et enfin, nous abordons les modalités pratique pour notre problème.

3.1 Description du problème d'ordonnancement de véhicules

L'optimisation a été en tout temps un souci majeur pour les industries, en particulier pour les chaînes de montage. La compagnie de véhicules Renault n'échappe pas à cette règle. C'est dans cette optique que la compagnie de véhicules Renault a lancé un challenge, qui a pour but l'ordonnancement des véhicules en montage en prenant en considération les différents aspects de réalisation (options, couleurs, etc.). Ceci donnera plus de flexibilité dans les procédures de gestion dans un environnement commercial plein de concurrence.

En examinant soigneusement les tâches menées dans l'atelier de montage, deux tâches quotidiennes sont identifiées. La première est la planification d'une journée de fabrication pour chaque véhicule commandé, en prenant en considération la capacité des lignes de fabrication, et les délais contractuels (fabricant-client). La deuxième tâche se

[1] http://www.prism.uvsq.fr/ vdc/ROADEF/CHALLENGES/2005/

rapporte à l'ordonnancement des véhicules lors de la journée planifiée. Ainsi, le grand défi reste, de ne pas dépasser la capacité des ateliers.

Nous porterons une attention particulière au processus de l'ordonnance-ment du *problème d'ordonnancement de véhicules*, notamment dans les deux ateliers de la peinture et de montage.

3.1.1 Processus de l'ordonnancement du problème d'ordonnancement de véhicules

Atelier peinture

Dans l'atelier de peinture, l'objectif recherché est de minimiser les dépenses reliées aux opérations de nettoyage des pistolets. Le nettoyage se fait par la génération de rafales de peintures les plus longues possibles. Regrouper tous les véhicules de même couleur, permettra de réduire le nombre de purges. En outre, nous avons une contrainte qui limite la longueur maximale de chaque de rafale de teintes. En effet, la purge est nécessaire périodiquement même s'il n'y a pas eu de changements de couleur.

Atelier montage

En ce qui concerne cet atelier, l'objectif visé est d'éparpiller les véhicules difficiles dans la séquence de production. Les véhicules dits difficiles, sont ceux qui comprennent des options, telles que le toit ouvrant, l'air climatisé ou autres, pour lesquelles une charge de travail additionnelle est nécessaire. Pour chaque option, la chaîne d'assemblage possède une capacité maximale indiquant le nombre de voitures N pouvant être produites sur une séquence de P voitures consécutives, que nous appelons contrainte de capacité ou de ratio.

Nous distinguons alors, deux types de contraintes de ratio, les prioritaires et les non prioritaires. Le premier type de contraintes correspond à des contraintes « lourdes »de

l'atelier de montage, quand au deuxième type il correspond à des opérations plus légères. Une attention particulière sera accordée aux contraintes du premier type c'est à dire aux contraintes de ratio prioritaires par rapport à celles du second type.

C'est pourquoi, pour chaque contrainte de ratio N/P, nous cherchons à obtenir la configuration idéale suivante : sur toute sous sequence de véhicules dans la ligne de production, que nous appelons fenêtre glissante de taille P sur le film, qui est en fait la séquence de véhicules qui sera envoyé à la fabrication, il ne doit pas y avoir plus de N véhicules concernés par la contrainte de ratio. Dans le cas d'une contrainte de ratio du type $1/P$, nous visons une séquence de 2 véhicules dans le film concernés par la contrainte de ratio distants d'au moins $P-1$ positions : $X----X$ pour un ratio de $1/5$.

Exemple

Pour une contrainte de ratio de type $1/5$, nous comptabilisons les violations avec des fenêtres glissantes de taille 5:

$---X---X$: 1 violation sur la fenêtre glissante $---[X---X]$

$---X--X-$: 2 violations sur les fenêtres glissantes $--[-X--X]-$ et $---[X--X-]$.

Enfin, le *problème d'ordonnancement de véhicules* doit tenir compte de l'état des derniers véhicules de la journée précédente, dont les positions sont figées et connues d'avance, pour le besoin du calcul d'ordonnancement de la journée courante et des violations des contraintes de ratio sur la journée courante.

Renault fournit des instances de problèmes avec des objectifs qui indiquent la priorité à savoir :

HPRC, minimiser les violations des contraintes de ratio prioritaires

LPRC, minimiser les violations des contraintes de ratio non prioritaires

PCC, minimiser le nombre de changements de couleurs de peinture

Par conséquent, il y a trois multi-objectifs, le premiers est HPRC-LPRC-PCC, qui donne la priorité au HPRC, en suite à LPRC, et finalement à PCC, le second HPRC-PCC-LPRC et enfin PCC-HPRC-LPRC. Cela signifie que n'importe quelle solution avec moins de violations de contraintes non prioritaire est considérée meilleure que toutes autres avec plus de violations de contraintes prioritaire, indépendant du nombre de violations des contraintes non prioritaire et du nombre de purges.

3.1.2 Problème à résoudre

Le *problème d'ordonnancement de véhicules* consiste à déterminer l'ordonnancement du passage des véhicules lors de la fabrication dans une chaîne de montage, et ceci en satisfaisant les besoins des ateliers de peinture et de montage. Le *problème d'ordonnancement de véhicules* se trouve être un problème d'optimisation multi-objectifs qui est NP-complet [Cook, 1971], où nous devons trouver une solution qui satisfait les contraintes de ratio et qui minimise le nombre de purges.

3.2 Modalités pratiques

Nous disposons de trois bases d'instances fournies par ROADEF, les bases A, B et X. Chaque base contient une liste d'instances. Chacune de ces instances correspond à un jeu de données d'une usine. La même usine peut être présente dans plusieurs instances de problèmes et un jeu de données d'une usine englobe quatre fichiers textes à savoir:

- **Fichier optimization_objectives** : Dans lequel l'ordre des objectifs d'optimisation (HPRC, LPRC et PCC) à satisfaire est fixé, de l'objectif le plus prioritaire vers le moins prioritaire.
- **Fichier véhicules** : Il décrit les véhicules à produire, ainsi que leurs caractéristiques (la journée de fabrication, la couleur, la valeur (0/1) par contrainte de ratio indi-

quant si le véhicule est visé par la contrainte de ratio et le numéro de séquence calculé par l'application opérationnelle de Renault).

- **Fichier ratio** : Ce fichier définit les contraintes de ratio prioritaires et non prioritaires. Pour chaque une, nous avons le ratio N/P, la valeur de priorité (1=prioritaire, 0=non prioritaire) et son identifiant.

- **Fichier paint_batch_limit**: Il indique la longueur maximale des rafales de peintures.

3.3 Conclusion

Le problème de car-sequencing est un problème d'ordonnancement de la production qui se trouve être un problème d'optimisation combinatoire. En effet, ce problème consiste à déterminer l'ordre dans lequel un ensemble de voitures seront fabriquées, en ayant certains objectifs à atteindre. Ce problème concerne deux ateliers, atelier de peinture et celui d'assemblage. Pour chaque atelier, nous avons exprimé l'objectif visé ainsi que les contraintes à satisfaire. Nous avons également décrit les modalités pratiques, qui sont en fait, les instances fournies par ROADEF et qui nous permettent de vérifier notre approche.

Nous formulerons le problème du car-séquencing en un problème SAT, résolu avec trois approches au chapitre suivant.

CHAPITRE IV

RÉSOLUTION DU PROBLÈME D'ORDONNANCEMENT DE VÉHICULES

Dans ce chapitre, nous présentons la méthodologie de résolution par un encodage en logique propositionnelle.

4.1 Méthodologie de résolution

La solution du *problème d'ordonnancement de véhicules* est une assignation de chaque véhicule à une position sur la chaîne de production, satisfaisant certaines contraintes. La recherche de solution nous amène à opter pour la stratégie qui consiste tout d'abord à transformer le *problème d'ordonnancement de véhicules* en un problème de satisfiabilité (SAT), puis à le résoudre à l'aide du solver SATO4.2. Pour ce faire, nous utilisons en plus des clauses, des contraintes arithmétiques, donc nos instances contiennent des clauses et des TL–clauses. Un tel modèle d'encodage booléen est décrit, par deux types d'ensembles de familles de contraintes : des familles de contraintes clausales, contenant des clauses et des familles de contraintes arithmétiques, contenant des TL–clauses.

Pour tout ce qui suit, nous utilisons les caractères majuscules pour les paramètres du problème (nombres de véhicules, nombre de couleurs, ensembles de véhicules etc ...) et les minuscules pour les variables booléennes qui apparaîtront dans les clauses et les TL-clauses. Les paramètres du problème sont au nombre de quatre :

NV : nombre de véhicules

NC : nombre de couleurs

NCR : nombre de contraintes de ratio

L : longueur maximale de la chaîne de rafales de peinture

Nous allons décrire dans ce qui suit les différents encodages booléens en FNC/TL représentant le *problème d'ordonnancement de véhicules.*

4.2 Encodage en FNC/TL

Afin d'encoder les instances du *problème d'ordonnancement de véhicules* du challenge ROADEF 2005 en FNC/TL, nous introduisons en premier lieu deux groupes de variables booléennes : p_i qui est vrai s'il y a une purge avant le traitement du i véhicule de la solution et x_{ij} pour indiquer quel véhicule est à quelle position dans la solution, où x_{ij} est vrai si le i véhicule de la solution est le j de la donnée.

Avec NV véhicules, nous avons NV variables du premier groupe : $p_i, i, j = 1, \ldots, NV$ et NV^2 variables $x_{ij}, i, j = 1..NV$ pour le second groupe.

Pour se rendre compte de ce que cela peut représenter en pratique, considérons les données réelles suivantes, venant des trois familles d'instances fournies par Renault, où le nombre de véhicules s'étend de 77 à 1542 (avec une moyenne de 766). Nous constatons que pour traiter plusieurs milliers de véhicules nous aurons plusieurs millions de variables et ceci dû à notre deuxième groupe de variables. Il est donc raisonnable d'essayer de réduire le nombre de variables de notre deuxième groupe.

Si nous regroupons maintenant les véhicules par des types définis par leurs dates (représentant le jour planifié par ROADEF, pour lequel le véhicule doit être traité), couleurs et options, alors pour les instances de RENAULT, nous obtenons entre 12 et 414 types (avec une moyenne de 160).

Puisque deux véhicules du même type peuvent être interchangés dans la solution sans

modifier le nombre de purges ni le nombre de violations de contraintes de ratio, nous pouvons réinterpréter les variables x_{ij} du deuxième groupe par x_{ij} est vrai si le i véhicule de la solution est du type j.

Par conséquent nous avons maintenant x_{ij}, $i = 1 \ldots NV, j = 1 \ldots NT$, où NT est le nombre de types. Ceci engendre au plus quelques centaines de milliers de variables. Notons que pour tout ce qui suit, nous travaillons avec les variables x_{ij} définies par les types.

Maintenant considérons les contraintes que nous avons divisées en trois groupes :

- Contraintes arithmétiques de base.
- Contraintes de purges.
- Contraintes de ratio.

4.2.1 Contraintes arithmétiques de base

Souvent en recherche opérationnelle, certaines contraintes sont mieux exprimées en termes d'opérations arithmétiques. La contrainte (4.1) par exemple, exprime le fait que le nombre de purges doit être minimisé, (4.2) exprime que le i véhicule de la solution est d'un seul type, finalement (4.3) exprime que le j type doit apparaître NV_j fois dans la solution, où NV_j est le nombre de véhicules du type j.

$$Min(\sum_{i=1}^{NV} p_i) \tag{4.1}$$

$$\sum_{j=1}^{NT} x_{ij} = 1, i = 1 \ldots NV \tag{4.2}$$

$$\sum_{i=1}^{NV} x_{ij} = NV_j, j = 1 \ldots NT \tag{4.3}$$

4.2.2 Contraintes de purges

Nous exprimons le fait qu'il doit y avoir une purge à tout les L+1 véhicules au minimum par la contrainte (4.4).

Pour s'assurer qu'en cas de changement de couleur entre le i et le $(i+1)$ véhicule, nous devons purger, nous pourrions utiliser les contraintes suivantes : $x_{ij} \wedge x_{i+1j'} \rightarrow p_{i+1}$, où $i = 1, \ldots, NV-1$ et $j, j' = 1, \ldots, NT$, j, j' étant deux types de différentes couleurs. Mais ceci produirait un nombre de clauses de l'ordre de $NV * NT^2$. Avec les instances de RENAULT, ceci mènera à plusieurs millions de clauses, ce qui ne semble pas raisonnable vu qu'il y a seulement entre 4 et 24 couleurs (avec une moyenne de 14). Il est donc naturel d'introduire des variables additionnelles afin de réduire le nombre de clauses produites.

Nous essayons de réduire le nombre important de variables et de clauses trouvées en encodant la couleur d'un véhicule en binaire. L'idée est de différentier directement les véhicules de couleurs différents dans la chaîne de rafales de la peinture. En fait, nous introduisons les variables c_{ik} qui désignent la couleur du i véhicule de la solution. La couleur du i véhicule de la solution (en binaire) est donc donnée par la suite de variables booléennes $c_{i1}, c_{i2}, \ldots, c_{ik}$, où $k = \lceil \log_2(NC) \rceil$ et $i = 1, .., NV$. Avec un maximum de 24 couleurs, nous obtenons un maximum de $5 * NV$ nouvelles variables, par conséquent quelques milliers au plus.

Nous exprimons par les contraintes (4.5) and (4.6) le fait que, si un bit de couleur du i véhicule est différent à celui du $(i+1)$ nous devons purger.

$$p_j \vee p_{j+1} \vee \cdots \vee p_{j+L}, j = 1 \ldots NV - L \quad (4.4)$$

$$c_{ik} \wedge \neg c_{i+1k} \rightarrow p_{i+1}, i = 1 \ldots NV - 1, k = 1, \ldots, \lceil log_2(NC) \rceil \quad (4.5)$$

$$\neg c_{ik} \wedge c_{i+1k} \rightarrow p_{i+1}, i = 1 \ldots NV - 1, k = 1, \ldots, \lceil log_2(NC) \rceil \quad (4.6)$$

Enfin, nous devons nous assurer que la couleur donnée par les c_{ik} est en effet la vraie

couleur, sachant que la couleur du type j est connue. Soit la valeur γ_{jk} désignant l'encodage binaire de la couleur du j véhicule. Cette propriété est exprimée par : $x_{ij} \rightarrow \bigwedge_{k=1}^{n}(c_{ik} \leftrightarrow \gamma_{jk})$,$i = 1 \ldots NV$, $j = 1 \ldots NV$ et $n = \lceil \log_2(NC) \rceil$. Cet encodage engendre $NV * NT * \lceil log_2(NC) \rceil$ clauses, ce qui représente quelques millions de clauses pour les intstances ayant un nombre de véhicules assez important.

4.2.3 Contraintes de ratio

Nous faisons face avec les contraintes de ratio à une situation semblable à celle des couleurs. Dans les instances ROADEF 2005, nous avons que le nombre de types varie de 12 à 414 types (au moyenne de 160), Par contre il y a uniquement de 1 à 26 contraintes de ratio (avec une moyenne de 12). Par conséquent nous introduisons de nouvelles variables booléennes r_{il} qui sont vraies si le i véhicule de la solution vérifie la l contrainte de ratio, où $i = 1 \ldots NV$ et $l = 1 \ldots NCR$. Ce qui ajoute quelques dizaines de milliers de nouvelles variables.

La contrainte (4.7) exprime le fait que, si l est la contrainte ayant le ratio N/P, il ne doit pas y avoir plus de N véhicules qui satisfont cette contrainte de ratio dans une séquence de P véhicules.

Finalement la contrainte (4.8) exprime le faite que si le i véhicule de la solution est du j type de la donnée, toutes les contraintes de ratio du j type sont satisfaites par le i véhicule de la solution. Il y a $NV * NRC$ contraintes de type (4.7), par conséquent pour nos instances, ceci donne environ 40 000 contraintes au maximum, d'autre part, la contrainte (4.8) donne au maximum $NV * NT * NRC$ contraintes.

$$\sum_{i'=i}^{Min(i+P-1,NV)} r_{i'l} \leq N, \qquad \begin{array}{l} i = 1 \ldots NV, l = 1 \ldots NRC, \\ \text{N/P le ratio de la contrainte l} \end{array} \tag{4.7}$$

$$x_{ij} \rightarrow r_{il}, \qquad \begin{array}{l} i = 1 \ldots NV, j = 1 \ldots NT, \\ \text{l est une option de contrainte de type j} \end{array} \tag{4.8}$$

Les calculs théoriques engendrés par cet encodage sont résumés dans les tableaux 4.2.3 et 4.2.3. Les contraintes clausales du tableau 4.2.3 peuvent être traduites en clauses, (voir la sous section 1.2.1).

Tableau 4.1 Complexité des TL-clauses

Contraintes TL-clauses	(4.1)	(4.2)	(4.3)	(4.7)	(4.9)
Nombre de contraintes dans la famille	1	NV	NT	NV*NCR	1
Longueur d'une TL-clause	NV	NT	NV	Min(i+P-1,NV)	NV
Taille totale	O(NV)	O $(NV * NT)$	O $(NV * NT)$	$O(NV * NCR)$	$O(NV)$

Tableau 4.2 Complexité des contraintes clausales

Contraintes clausales	(4.4)	(4.5)	(4.6)	(4.8)
Nombre de formules dans la famille	NV-L	$NV * \lceil \log_2(NC) \rceil$	$O(NCR * NV)$	$O(NV * NT)$
Longueur des clauses	L+1	3	N	2
Taille totale	O(NV)	O $(NV * \lceil \log_2(NC) \rceil)$	O (NCR*NV)	$O(NV * NT)$

4.3 Optimisation multi-objectifs et approches de résolution

Deux optimisations sont à considérer. Le premier objectif consiste à réduire au minimum le nombre de purges, quant au deuxième et dans le cas où toutes les contraintes de ratio ne peuvent pas être satisfaites, nous devons optimiser le nombre de violations. Afin d'optimiser le premier objectif, nous utilisons un seuil explicite sur le nombre de purges. Nous choisissons ce seuil comme suit. Il y a NV véhicules et il doit y avoir au moins une purge chaque L véhicules, le nombre minimal de purges est donc MP = $\lceil NV/L \rceil$. Mais, nous ne pouvons pas toujours trouver une solution avec seulement MP purges, donc, nous augmentons ce seuil par un coeficient de purge CP plus grand que 1.0. Ceci mène à la TL-clause suivante :

$$\sum_{i=1}^{NV} p_i \leq \lceil (NV/L) * CP \rceil \tag{4.9}$$

Chaque valeur de CP donne une instance différente. Afin d'optimiser le nombre de

purges, il est donc nécessaire de faire plusieurs essais sur les instances avec différentes valeurs de CP. Pour tous les essais, comme nous les verrons, le temps d'exécution reste raisonnable.

Pour le deuxième objectif d'optimisation, qui est de réduire au minimum le nombre de violation des contraintes de ratio, la situation n'est pas aussi simple. Dans ce cas-là, nous devons considérer non seulement chaque contrainte mais également pour une contrainte donnée de N/P, chaque fenêtre de P véhicules consécutifs.

Nous allons maintenant montrer, pour chaque multi-objectif, comment nous optimisons le nombre de violations des contraintes de ratio. Rappelons que ces multi-objectifs sont:

- HPRC-LPRC-PCC.
- HPRC-PCC-LPRC.
- PCC-HPRC-LPRC.

4.3.1 HPRC-LPRC-PCC

Pour atteindre le multi-objectif HPRC-LPRC-PCC, nous optons pour l'approche *Toutes Contraintes de Ratio* (TCR), où nous prenons en considération toutes les contraintes de ratio prioritaires et non prioritaires, tout en essayant de purger au minimum. Puisque c'est un objectif de taille, nous commençons par une grande valeur de CP (1.9 dans nos expériences) puis, nous l'augmentons jusqu'au point où nous obtenons une solution.

Le principe de l'approche TCR consiste à générer les types des véhicules en les regroupant selon les critères date, couleurs et contraintes de ratio, ensuite, nous cherchons à trouver un type pour chaque position dans la chaîne. La solution trouvée sera donc un ensemble de couple formés de la position du véhicule et du type correspondant (position i,type j) présentés par x_{ij}.

À partir de cette solution, nous plaçons tous les véhicules de chaque type sur la position correspondante.

4.3.2 HPRC-PCC-LPRC

Quand au multi-objectif HPRC-PCC-LPRC, l'approche *Contraintes de Ratio Prioritaires* (*CRP*) est appliquée. Nous tentons de satisfaire toutes les contraintes prioritaires, puis d'optimiser le nombre de purges, et en dernier lieu, de satisfaire au maximum les contraintes non prioritaires. Dans ce cas-ci, la valeur du CP choisie est inférieure à celle de la sous-section précédente.

En effet, cette approche de résolution consiste à décomposer le problème en deux étapes. Dans la première étape, nous cherchons à satisfaire toutes les contraintes prioritaires. Nous commençons par générer les types des véhicules en les regroupant selon ces trois critères seulement : date, couleurs et contraintes de ratio prioritaires, ensuite, nous résolvons le problème en cherchant un type pour chaque position dans la chaîne. La solution trouvée sera donc des couples de position du véhicule et le type correspondant comme ci-dessus x_{ij}, (position i,type j).

Nous aurons donc, autant de couple (type, véhicule) que le nombre de véhicules appartenant à ce type.

À partir de cette première solution générée, et pour chaque type, nous plaçons tous les véhicules appartenant à ce dernier un par un sur une position parmi celles où ce type apparaît. Autrement dit, à une position i donnée de la solution, nous prenons un véhicule quelconque de ce type et nous le plaçons à cette position i, nous répétons cette étape jusqu'a ce que tous les véhicules soient positionnés. Puis nous calculons le nombre de contraintes non prioritaire satisfaite. Cependant, si le nombre de cette dernière est assez important, nous passons à la deuxième étape.

Lors de la deuxième étape, nous essayons d'améliorer la solution en terme de satisfaction des contraintes non prioritaires, sans pour autant changer le nombre de violations des contraintes prioritaires. Pour atteindre cet objectif, nous interchangeons les véhicules du même types entre eux, vue qu'ils ont les mêmes contraintes prioritaires et mêmes couleurs, mais pas nécessairement les mêmes contraintes non prioritaires. Si le nombre

de contraintes non prioritaires satisfaites a augmenté, nous maintiendrons la nouvelle solution sinon elle sera rejetée.

4.3.3 PCC-HPRC-LPRC

Pour ce qui est du dernier multi-objectif PCC-HPRC-LPRC, nous optons pour l'approche *Types des Couleurs* (TC). Avec exactement le même principe que l'approche CRP, nous résolvons le *problème d'ordonnancement de véhicules.* Nous changeons uniquement les critères de la création des types. Dans la présente approche, le type n'est défini que par la date et la couleur.

En effet, la résolution par cette approche se fait en deux phases. Lors de la première, nous rassemblons tous les véhicules ayant les caractéristiques cités en commun (date et couleur). Nous procédons à la génération des types des véhicules en les regroupant selon ces deux critères seulement, puis nous résolvons le problème en cherchant un type pour chaque position de la chaîne. La solution sera donc des couples de position du véhicule et le type correspondant(position i,type j), x_{ij}.

À partir de cette solution, nous plaçons tous les véhicules de chaque type sur la position correspondante, par le même principe que celui présenté dans la précédente approche. Nous calculons alors, le nombre de contraintes prioritaire et non prioritaire satisfaite. Cependant, si le nombre de contraintes prioritaire satisfaite est assez important, alors nous garderons cette solution qui satisfait tout d'abord le minimum de purges, puis le maximum des contraintes prioritaires et enfin les non prioritaires. Néanmoins, si le nombre de contraintes prioritaire satisfaites est réduit, nous passons à la deuxième phase pour l'améliorer.

Lors de cette dernière, nous essayons d'améliorer la solution en terme de satisfaction maximale des contraintes prioritaires puis les non prioritaires sans pour autant toucher aux purges. On interchange les véhicules du même types entre eux vue qu'ils ont la même couleur mais pas nécessairement les mêmes contraintes. Si le nombre de contraintes satisfaites a augmenté, nous maintiendrons la nouvelle solution sinon elle sera rejetée.

Après avoir terminé avec les contraintes prioritaires, nous procédons à l'amélioration des non prioritaires de la même manière et ceci sans toucher ni aux purges ni aux contraintes prioritaires.

4.4 Conclusion

Dans ce chapitre, nous avons exposé la méthodologie de résolution du *problème d'ordonnancement de véhicules*, qui consiste en fait à trouver une affectation adéquate des véhicules aux différentes positions dans la chaîne de production. En effet, notre stratégie est basée sur la transformation de notre problème en un problème de satisfiabilité (SAT). Pour ce faire, nous avons décrit un encodage booléen FNC/TL, où nous avons défini deux types d'ensembles de familles de contraintes, à savoir, des familles de contraintes clausales ne contenant que des clauses, et des familles de contraintes arithmétiques contenant des TL-clauses.

Le modèle généré est résolu par suite, à l'aide du solver SATO4.2, un outil qui peut résoudre un modèle exprimant les clauses et les TL-clauses en même temps. L'usage de TL-clauses a permis d'ailleurs, d'obtenir une complexité raisonnable en terme de nombre de contraintes, que ce soit causales ou TL-clauses.

Ainsi, trois approches ont été développées, chacune atteignant un des objectifs fixés par ROADEF. Les trois approches ont été basées sur l'encodage en logique propositionnelle, qui traduit les différentes contraintes exigées par le challenge en clauses et particulièrement en TL–clauses, présentant parfaitement notre *problème d'ordonnancement de véhicules.*

En outre, le trois approches convergent, dans le sens où elles partagent la même structure d'organisation des données, en d'autres termes, pour chacune de ces approches, les véhicules sont regroupés par type et les mêmes clauses et TL–clauses, qui traduisent les exigences des deux ateliers, atelier de peinture et celui d'assemblage. Par contre, elles diffèrent par l'objectif à atteindre, fixé au préalable et aussi dans la manière dont le modèle d'encodage est résolu. Autrement dit, pour l'approche TCR, le problème est

résolu globalement en une seule étape. Quant aux approches CRP et TC, le problème est décomposé en deux sous problèmes, où chacun est résolu à part, suivi d'une phase d'amélioration de la solution obtenue.

En effet, du point de vue pratique, les trois approches restent réalisables, étant donné que leurs complexités en terme de nombre de variables, de clauses et de TL–clause est relativement raisonnables. Dans le prochain chapitre, nous présentons l'analyse des résultats obtenus avec les trois approches.

CHAPITRE V

RÉSULTATS EXPÉRIMENTAUX ET ANALYSE

Ce chapitre présente les résultats obtenus lors d'une série de tests effectuée sur les données du challenge ROADEF'2005, afin d'évaluer la qualité de la solution.

Aprés avoir décrit brièvement les étapes suivies permettant la résolution du problème pour en arriver concrètement à trouver un ordonnancement des véhicules satisfaisant au mieux l'objectif fixé au départ, nous présentons pour chacune des trois approches décrites au chapitre précédent, l'ensemble des résultats obtenus étayé par une analyse. À la fin de ce chapitre nous commentons les résultats de l'ensemble de ces trois approches.

5.1 Réalisation

L'ordonnancement des véhicules selon un critère d'optimisation choisi au préalable et donné par le challenge a été réalisé en trois étapes. Nous avons réalisé un logiciel (nommé Roadef) qui génère pour chacune des trois approches, les FNC/TL en format DIMACS pour les groupes d'instances A,B et X.

Dans une deuxième étape, nous avons utilisé l'outil SATO version 4.2 pour solutionner ces instances FNC/TL. L'outil SATO4.2 produit, en effet en sortie un fichier solution, qui représente tous les littéraux ayant la valeur True.

Finalement, notre logiciel *Roadef* a permis en utilisant l'option "Interprétation Résultat" de traduire les solutions de la FNC/TL dans le format de solution exigé par le challenge.

Ceci permet de vérifier nos solutions avec l'outil « checkers-exeCarSeq.exe »se trouvant sur le site de challenge ROADEF'2005. Cet outil calcule le nombre de violations des contraintes de ratio prioritaires et non prioritaires ainsi que le nombre de purges, donc permet de comparer nos solutions avec celles disponibles sur le site du challenge.

5.2 Analyse

Dans cette section, nous présentons une comparaison des résultats obtenus par notre méthode pour chaque multi-objectif avec les résultats du challenge ROADEF'2005. Le temps d'exécution du challenge ROADEF'2005 a été limité à 600 secondes en utilisant un PC Pentium4/1.6 Ghz/1 Go RAM. Quant à nos instances, nous les avons solutionnées sur une machine semblable.

5.2.1 HPRC-LPRC-PCC

Comme nous l'avons dit à la section 4.3.1, pour les instances du challenge ROADEF'2005 ayant un multi-objectif HPRC-LPRC-PCC, nous recherchons des solutions satisfaisant toutes les contraintes de ratio. Nous débutons avec un coefficient de purge 1.9, tableau 5.1.

Les instances du tableau 5.1 sont toutes celles du challenge ROADEF'2005 ayant comme multi-objectif HPRC-LPRC-PCC. La première colonne désigne le code d'instance, représenté par la première lettre désignant l'ensemble des instances à qui appartient (A,B ou X), suivi d'un numéro. Nous donnons en appendice A la correspondance entre les codes que nous introduisons et les noms des instances tels qu'ils apparaissent sur le site du challenge. NV est le nombre de voitures, MP le nombre minimal de purges. FNC/TL et SATO indiquent le temps écoulé en seconde pour produire la FNC/TL et pour les résoudre avec SATO. Quant aux HPRC, LPRC et PCC, ils désignent le nombre de violations de contraintes prioritaires, le nombre de violations de contraintes non prioritaires et le nombre de purges pour le gagnant du challenge ROADEF'2005 par rapport à notre méthode.

Tableau 5.1 Toutes Contraintes de Ratio, Coefficient de Purge 1.9

Données			Temps d'execution(sec)		ROADEF'2005/TCR-CP1.9			
INST	NV	MP	FNC/TL	SATO	HPRC	LPRC	PCC	CPE
A1	904	61	31	41	423/??	782/??	63/??	/
A2	618	62	15	40	00/00	61/00	290/86	1.38
A3	981	50	35	45	155/??	00/??	68/??	/
A4	1231	124	43	64	00/00	99/00	720/169	1.36
A5	499	2	7	26	00/??	1/??	31/??	/
A6	1274	128	53	99	4/00	00/00	302/180	1.40
A7	1329	133	54	105	4/00	34/00	309/189	1.42
A8	434	17	4	23	367/00	52/00	27/25	1.48
A9	904	25	33	43	367/00	52/00	27/25	1
B1	540	1	31	45	00/??	00/??	109/??	/
B2	595	60	10	26	03/00	00/00	337/69	1.15
B3	854	57	33	44	00/00	00/00	182/68	1.19
B4	902	17	32	48	00/00	00/00	20/19	1.14
B5	451	30	5	21	00/00	69/00	130/35	1.16
B6	1257	125	43	77	00/00	3912/00	479/139	1.11
B7	1130	46	35	46	48/00	00/00	316/55	1.19
B8	1319	132	47	91	1074/00	1068/00	298/148	1.12
B9	385	26	18	25	54/00	3/00	76/31	1.19
B10	93	4	2	13	00/00	70/00	6/4	1
B11	773	52	33	40	35/00	2170/00	167/59	1.13
B12	133	2	3	17	67/??	52/??	49/??	/
B13	293	2	5	25	385/??	341/??	205/??	/
B14	1263	51	42	81	00/00	29/00	117/59	1.15
B15	385	26	18	30	98/00	188/00	38/38	1.47
B16	1119	56	34	49	214/??	671/??	59/??	/
B17	540	1	30	32	22/00	148/00	13/1	1
B19	1257	126	36	46	122/00	5589/00	126/126	1
B20	93	4	3	8	00/00	71/00	4/4	1
B22	133	2	3	13	156/00	90/00	6/2	1
B25	1119	56	32	43	214/00	671/00	59/56	1
B28	451	30	5	19	52/00	178/00	31/30	1
B29	902	63	37	39	115/00	670/00	64/63	1
X1	78	1	1	5	2/00	3/00	12/1	1
X2	460	5	4	22	36/??	341/??	95/??	/
X3	1271	22	29	58	00/00	160/00	407/39	1.77
X6	377	1	14	30	56/00	00/00	6/1	1

Tableau 5.2 Toutes Contraintes de Ratio, Coefficient de Purge 1.3

Données			Temps d'execution(sec)		ROADEF'2005/TCR-CP1.3			
INST	NV	MP	FNC/TL	SATO	HPRC	LPRC	PCC	CPE
A2	618	62	15	38	00/00	61/00	290/79	1.28
A4	1231	124	43	65	00/00	99/00	720/162	1.30
A6	1274	128	53	99	4/00	00/00	302/167	1.30
A7	1329	133	54	104	4/00	34/00	309/172	1.29
B15	385	26	18	30	98/00	188/00	38/32	1.24
X3	1271	22	29	58	00/00	00/00	407/28	1.27

Comme le montre le tableau 5.1, notre méthode trouve une solution pour 28 cas sur 36 avec ce coefficient de purge. En outre, le temps total de la résolution est très court, il ne dépasse jamais les 3 minutes, ce qui reste nettement inférieur au maximum autorisé par le challenge (10 minutes). Enfin, dans tous les cas où nous obtenons une solution, elle se trouve être optimale pour les deux premiers objectifs (nous satisfaisons toutes les contraintes de ratio) et nous obtenons toujours de meilleurs résultats par rapport au gagnant du challenge ROADEF'2005, du point de vue troisième objectif (PCC).

Par ailleurs, il est possible de réduire le nombre de purges. La dernière colonne du tableau 5.1 représente le coefficient de purge effectif CPE. Ce dernier est le rapport du nombre de purges trouvé par notre méthode au nombre minimum de purges (PCC/MP).

Afin de réduire la valeur de PCC, nous générons de nouvelles instances en choisissant un CP inferieure au CPE. Le tableau 5.2 montre que pour un CP égale à 1.3, nous arrivons à abaisser le CPE pour toutes les instances du tableau 5.1 ayant un CPE au-dessus de 1.3 ceci dans un temps d'exécution raisonnable.

D'un autre côté pour, les 8 instances du tableau 5.1, pour lesquels nous n'avons pas obtenu de solution, l'augmentation du CP à 2.9 et puis à 3.9 nous a permis de trouver une solution dans tous les cas. Les tableaux 5.3 et 5.4, montrent les solutions obtenues. Remarquons que le temps de résolution reste toujours très court. Comme précédemment, la solution obtenue est toujours optimale pour les deux premiers objectifs et elle est aussi meilleure comparativement à celle trouvée par le gagnant du

Tableau 5.3 Toutes Contraintes de Ratio, Coefficient de Purge 2.9

Données			Temps d'execution(sec)		ROADEF'2005/TCR-CP2.9			
INST	NV	MP	FNC/TL	SATO	HPRC	LPRC	PCC	CPE
A3	981	50	36	47	155/00	00/00	68/106	2.12
A5	499	2	7	27	00/00	1/00	31/5	2.5
B1	540	1	32	47	00/00	00/00	109/2	2
B12	133	2	3	17	67/00	52/00	49/4	2
B13	293	2	5	26	385/00	341/00	205/5	2.5
X2	460	5	4	23	36/00	341/00	95/13	2.8

Tableau 5.4 Toutes Contraintes de Ratio, Coefficient de Purge 3.9

Données			Temps d'execution(sec)		ROADEF'2005/TCR-CP3.9			
INST	NV	MP	FNC/TL	SATO	HPRC	LPRC	PCC	CPE
A1	904	61	31	43	423/00	782/00	63/185	3.03
B16	1119	56	36	53	214/00	671/00	59/171	3.05

challenge, pour le dernier objectif.

5.2.2 HPRC-PCC-LPRC

Comme nous l'avons expliqué à la section 4.3.2, nous avons adopté l'approche CRP pour ce multi-objectif. Dans ce cas, une solution est obtenue pour tout les cas d'instance comme le montre le tableau 5.5. Nous avons utilisé un CP égal à 1.09, ce qui est inférieur à ceux de la précédente section, vue qu'ici, nous essayons de satisfaire moins de contraintes. Nos solutions sont encore optimales pour le premier objectif (HPRC), de plus, elles restent meilleure que celles trouvées par le gagnant du challenge ROADEF'2005 pour le deuxième objectif (PCC), sauf pour A1. Pour le dernier objectif (LPRC), nous présentons les résultats obtenus avant et après la permutation des véhicules. Il en résulte que la permutation n'améliore que très peu nos résultats.

Tableau 5.5 Contraintes de Ratio Prioritaire, Coefficient de Purge 1.09

Données			Temps d'execution(sec)			ROADEF'2005/CRP-CP1.09			
INST	NV	MP	FNC/TL	SATO	PERM	HPRC	LPRC	PCC	CPE
A1	904	61	28	39	25	00/00	51/93/89	63/66	1.08
A2	434	17	3	21	21	00/00	612/755/752	174/67	1.08
A3	981	50	34	41	28	13/00	00/35/32	129/54	1.08
A4	1231	124	51	51	30	00/00	3134/4755/4750	228/136	1.09
A5	499	4	5	24	21	00/00	1/26/21	31/4	1
A6	1274	128	41	72	32	4/00	00/86/83	249/136	1.06
A7	1329	133	44	76	33	4/00	79/95/91	280/143	1.07
A8	619	89	26	33	26	00/00	00/764/759	112/95	1.06
B1	540	4	29	38	23	00/00	144/185/179	19/4	1
B2	595	64	7	19	25	03/00	1029/1049/1043	93/67	1.04
B3	854	59	32	39	28	00/00	187/201/201	130/64	1.08
B4	902	94	29	37	28	00/00	378/386/385	161/102	1.08
B5	451	33	4	17	21	00/00	69/105/102	130/33	1
B6	1257	130	41	72	30	00/00	5180/4648/4648	167/139	1.06
B7	1130	49	33	42	29	48/00	8/81/75	310/52	1.06
B8	1319	135	40	80	32	1074/00	1068/960/956	298/145	1.07
B9	385	28	16	22	19	54/00	124/136/136	49/29	1.03
B10	93	4	1	7	15	00/00	71/77/75	4/4	1
B11	773	52	30	35	26	35/00	2150/2228/2225	167/57	1.09
B12	133	3	2	14	18	67/00	61/126/123	63/3	1
B13	293	3	4	24	20	385/00	351/544/542	187/3	1
B14	1263	53	39	76	32	00/00	89/95/95	78/57	1.07
B15	385	26	16	22	20	54/00	124/140/137	49/28	1.07
B16	1119	58	31	39	28	00/00	103/118/117	189/63	1.08
X1	234	23	1	11	21	153/00	00/104/102	34/25	1.08
X2	508	66	5	26	21	31/00	1116/1321/1321	76/70	1.06
X3	822	22	31	46	25	00/00	98/160/154	110/24	1.09
X4	792	66	27	43	24	00/00	1005/1005/1001	196/72	1.09
X5	252	27	1	15	22	08/00	35/48/46	87/28	1.03
X6	974	65	37	52	27	61/00	29/33/31	187/71	1.09
X7	1279	32	41	55	33	00/00	66/65/61	192/35	1.09
X8	278	28	1	17	20	00/00	30/45/45	30/29	1.03
X9	372	25	2	26	21	00/00	00/16/14	37/27	1.08
X10	1283	65	40	63	31	00/00	30/58/55	231/70	1.07
X11	985	13	39	56	29	00/00	794/921/915	55/13	1
X12	1543	61	50	75	33	00/00	239/292/289	69/65	1.07

Tableau 5.6 Types des Couleurs, Coefficient de Purge 1.05

Données			Temps d'execution(sec)			ROADEF'2005/TC-CP1.05			
INST	NV	MP	FNC/TL	SATO	PERM	HPRC	LPRC	PCC	CPE
A1	904	25	26	31	36	367/452/448	52/84/83	27/25	1
A2	981	50	31	37	39	155/302/289	00/61/54	68/51	1.02
A3	499	2	3	15	29	39/51/51	1/23/23	11/2	1
A4	904	61	28	33	37	423/456/451	782/901/896	63/65/63	1.03
B1	540	1	27	29	28	22/43/43	148/162/157	13/1	1
B2	1130	18	31	38	36	1327/1475/1459	31/150/150	50/18	1
B3	1257	126	33	41	31	122/250/250	5589/7235/7229	126/126	1
B4	385	26	14	25	22	98/175/173	188/296/293	38/27	1.03
B5	93	4	1	5	16	00/33/33	71/179/160	4/4	1
B6	773	23	28	31	29	709/965/953	2171/2284/2261	52/23	1
B7	133	2	1	9	20	156/248/239	90/287/285	6/2	1
B8	293	2	3	17	26	651/752/745	671/842/834	7/2	1
B9	1263	15	37	71	38	45/87/81	96/285/211	55/15	1
B10	1119	56	29	37	35	214/235/221	671/893/879	59/56	1
B11	902	63	26	35	31	115/279/235	670/930/915	64/63	1
B12	595	60	5	18	29	3/195/195	1029/1497/1488	93/63	1.05
B13	854	57	30	36	30	95/170/150	288/590/563	62/60	1.05
B14	451	30	2	14	28	52/104/98	178/306/279	31/30	1
X1	718	4	19	31	30	2/84/84	3/125/99	12/4	1
X2	92	00	1	4	18	10/32/31	00/45/34	5/0	/
X3	377	1	13	27	23	56/187/181	00/55/55	6/1	1

5.2.3 PCC-HPRC-LPRC

Dans ce multi-objectif, comme expliqué à la section 4.3.3,nous essayons d'abord d'optimiser le nombre de purges avec un coefficient de purge égale à 1.05. Comme le montre le tableau 5.6, pour le premier objectif (PCC), nous obtenons pour toutes les instances une valeur qui est meilleure que celle obtenue par le gagnant du challenge ROADEF'2005. En effet, la solution obtenue pour les 16 instances sur 21 est optimale car notre PCC est égal à MP. Concernant les 5 cas restant, notre solution reste très proche de l'optimum. Pour le deuxième et troisièmement objectif, nos résultats obtenus après amélioration restent moins bons, mais quand même assez proches à ceux obtenus par le gagnant. Dans ce cas aussi la permutation n'améliore pas beaucoup nos résultats.

5.3 Conclusion

Nous avons présenté l'application de notre méthode, pour les trois différents multi-objectifs, sur des données réelles établies par le challenge ROADEF'2005. En effet, notre méthode consiste à coder le problème d'ordonnancement des véhicules en FNC/TL. Les résultats obtenus par les trois approches sont prometteurs, dans la mesure où nous avons pu montré que le solver SATO trouve assez rapidement une solution, en particulier pour les multi-objectifs HPRC-PCC-LPRC et PCC-HPRC-LPRC.

En dépit du changement effectué sur les valeurs de CP dans le but de trouver une solution ou de l'améliorer, le temps d'exécution total reste toujours assez court.

CONCLUSION

Ce travail de recherche a permis de vérifier que les méthodes de résolution de problème SAT appliquées à un problème d'ordonnancement. Les algorithmes présentés au chapitre 2 ont servis de base au développement de l'algorithme spécifique au solver SATO4 (version 2). Plus précisément l'algorithme DPLL et en particulier les algorithmes retour-arrière (Backtracking) et retour ponctuel (Backjumping) étendus sur les TL-clauses ont permis d'utiliser SATO4.2 pour résoudre le *problème d'ordonnancement de véhicules* du challenge ROADEF'2005.

Les objectifs fixés pour notre travail de recherche ont été atteints et les approches présentées dans ce mémoire ont permis d'obtenir des résultas intéressants.

Dans un premier temps, l'objectif visait à transformer le *problème d'ordonnancement de véhicules* en un problème SAT. En effet, l'étude de ce problème a permis le développement d'un encodage en FNC/TL de toutes les contraintes du problème en se fixant l'objectif à optimiser au préalable, a été atteint avec une complexité raisonnable en nombre de contraintes. Nous avons d'ailleurs montré avec notre logiciel roadef que cet encodage pouvait être utilisé en pratique pour générer des instances FNC/TL qui pouvaient être résolues à l'aide du solver SATO4.2.

Trois approches fonctionnelles ont été étudiées dans le chapitre 4. La première la plus contraignante consistait à prendre en considération les trois objectifs à optimiser à la fois à savoir la minimisation des violations des deux types de contraintes de ratio tout en purgeant au minimum. Par contre dans les deux autres approches, le problème initial a été subdivisé en sous problèmes. Chaque sous–problème traitait un critère d'optimisation. Cette subdivision a beaucoup simplifié le problème, en fait,ceci a permis d'obtenir des solutions dans plusieurs cas.

Une analyse des résultats issus des trois approches développées, obtenus par les différents essais sur des instances réelles du chalenge ROADEF'2005, est décrite dans le chapitre 5. Cette analyse a bien confirmé la qualité de nos résultats.

Les méthodes décrites dans ce travail permettent de trouver une solution en solutionnant plusieurs instances, ne contenant pas les les mêmes contraintes de ratio, ni le même coefficient de purge. Lorsque nos instances ne contenaient pas toutes les contraintes de ratio, nous avons simplement permuté les véhécules pour tenter d'optimiser le nombre total de contraintes de ratio violées. Nous aimerions dan le future poursuivre la recherche amorcée en appliquant des heuristiques sophistiquées de la recherche opérationnelle permettant de résoudre le *problème d'ordonnancement de véhicules* vu comme étant un problème SAT, à fin d'améliorer la solution en temps et en qualité et principalement pour réduire le nombre d'essais à effectuer sur l'objectif de purge à optimiser, pour obtenir une solution.

APPENDICE A

CODES DES INSTANCES

Tableau A.1 Codes des instances de l'approche *Toutes Contraintes de Ratio* (TCR)

TCR Code	$ROADEF$ Code
X1	022_RAF_EP_ENP_$S49$_$J2$
X2	028_$CH1$_EP_ENP_RAF_$S50$_$J4$
X3	025_EP_ENP_RAF_$S49$_$J1$
X4	022_RAF_EP_ENP_$S49$_$J2$
X5	035_$CH1$_RAF_EP_$S50$_$J4$
X6	025_$CH2$_RAF_EP_$S50$_$J4$
A1	064_32_2_RAF_EP_ENP_$CH2$
A2	048_39_1_EP_ENP_RAF
A3	039_38_4_RAF_EP_$CH1$
A4	025_38_1_EP_ENP_RAF
A5	022_3_4_RAF_EP_ENP
A6	024_38_3_EP_ENP_RAF
A7	024_38_5_EP_ENP_RAF
A8	064_38_2_RAF_EP_ENP_$CH2$
A9	064_32_2_RAF_EP_ENP_$CH1$
B1	022_EP_ENP_RAF_$S22$_$J1$
B2	048_$CH2$_EP_ENP_RAF_$S22$_$J3$
B3	064_$CH1$_EP_ENP_RAF_$S22$_$J3$
B4	048_$CH1$_RAF_EP_ENP_$S22$_$J3$
B5	064_$CH2$_EP_ENP_RAF_$S22$_$J4$
B6	025_EP_ENP_RAF_$S22$_$J3$
B7	023_EP_ENP_RAF_$S23$_$J3$
B8	024_$V2$_RAF_EP_ENP_$S22$_$J1$
B9	028_$CH1$_EP_ENP_RAF_$S22$_$J2$
B10	028_$CH2$_EP_ENP_RAF_$S23$_$J3$
B11	029_EP_ENP_RAF_$S21$_$J6$
B12	035_$CH1$_EP_ENP_RAF_$S22$_$J3$
B13	035_$CH2$_EP_ENP_RAF_$S22$_$J3$
B14	035_$CH1$_EP_ENP_RAF_$S22$_$J4$
B15	028_$CH1$_RAF_EP_ENP_$S22$_$J2$
B16	028_$CH1$_REA_EP_ENP_$S22$_$J1$
B17	022_RAF_EP_ENP_$S22$_$J1$
B18	023_RAF_EP_ENP_$S23$_$J3$
B19	025_RAF_EP_ENP_$S22$_$J3$
B20	028_$CH2$_RAF_EP_ENP_$S23$_$J3$
B21	029_RAF_EP_ENP_$S21$_$J6$
B22	035_$CH1$_RAF_EP_ENP_$S22$_$J3$
B23	035_$CH2$_RAF_EP_ENP_$S22$_$J3$
B24	039_$CH1$_RAF_EP_ENP_$S22$_$J4$
B25	039_$CH3$_RAF_EP_ENP_$S22$_$J4$
B26	048_$CH2$_RAF_EP_ENP_$S22$_$J3$
B27	064_$CH1$_RAF_EP_ENP_$S22$_$J3$
B28	064_$CH2$_RAF_EP_ENP_$S22$_$J4$
B29	048_$CH3$_RAF_EP_ENP_$S22$_$J3$

Tableau A.2 Codes des instances de l'approche *Contraintes de Ratio Prioritaires* (*CRP*)

CRP Code	*ROADEF* Code
X1	0655_*CH*1_*EP*_*RAF*_*ENP*_*S*52_*J*1_*J*2_*S*01_*J*1
X2	048_*CH*2_*EP*_*RAF*_*ENP*_*S*49_*J*5
X3	029_*EP*_*RAF*_*ENP*_*S*49_*J*5
X4	042_*CH*1_*EP*_*RAF*_*ENP*_*S*50_*J*4
X5	034_*VU*_*EP*_*RAF*_*ENP*_*S*51_*J*1_*J*2_*J*3
X6	064_*CH*1_*EP*_*RAF*_*ENP*_*S*49_*J*1
X7	023_*EP*_*RAF*_*ENP*_*S*49_*J*2
X8	655_*CH*_*EP*_*RAF*_*ENP*_*S*51_*J*2_*J*3_*J*4
X9	064_*CH*2_*EP*_*RAF*_*ENP*_*S*49_*J*4
X10	039_*CH*3_*EP*_*RAF*_*ENP*_*S*49_*J*1
X11	034_*VP*_*EP*_*RAF*_*ENP*_*S*51_*J*1_*J*2_*J*3
X12	039_*CH*1_*EP*_*RAF*_*ENP*_*S*49_*J*1
A1	064_38_2_*CH*1_*EP*_*RAF*_*ENP*
A2	064_38_2_*EP*_*RAF*_*ENP*_*CH*2
A3	039_38_4_*EP*_*RAF*_*CH*1
A4	024_38_1_*EP*_*RAF*_*ENP*
A5	022_3_4_*EP*_*RAF*_*ENP*
A6	024_38_3_*EP*_*RAF*_*ENP*
A7	024_38_5_*EP*_*RAF*_*ENP*
A8	024_48_39_1_*EP*_*RAF*_*ENP*
B1	022_*EP*_*RAF*_*ENP*_*S*22_*J*1
B2	048_*CH*2_*EP*_*RAF*_*ENP*_*S*22_*J*3
B3	064_*CH*1_*EP*_*RAF*_*ENP*_*S*22_*J*3
B4	048_*CH*1_*EP*_*RAF*_*ENP*_*S*22_*J*3
B5	064_*CH*2_*EP*_*RAF*_*ENP*_*S*22_*J*4
B6	025_*EP*_*RAF*_*ENP*_*S*22_*J*3
B7	023_*EP*_*RAF*_*ENP*_*S*23_*J*3
B8	024_*V*2_*EP*_*RAF*_*ENP*_*S*22_*J*1
B9	028_*CH*1_*EP*_*RAF*_*ENP*_*S*22_*J*2
B10	028_*CH*2_*EP*_*RAF*_*ENP*_*S*23_*J*3
B11	029_*EP*_*RAF*_*ENP*_*S*21_*J*6
B12	035_*CH*1_*EP*_*RAF*_*ENP*_*S*22_*J*3
B13	035_*CH*2_*EP*_*RAF*_*ENP*_*S*22_*J*3
B14	039_*CH*1_*EP*_*RAF*_*ENP*_*S*22_*J*4
B15	028_*CH*1_*EP*_*RAF*_*ENP*_*S*22_*J*2
B16	039_*CH*3_*EP*_*RAF*_*ENP*_*S*22_*J*4

Tableau A.3 Codes des instances de l'approche *Types des Couleurs* (TC)

TC Code	$ROADEF$ Code
X1	$022\_RAF\_EP\_ENP\_S49\_J2$
X2	$035\_CH1\_RAF\_EP\_S50\_J4$
X3	$025\_CH2\_RAF\_EP\_S50\_J4$
A1	$064\_38\_2\_RAF\_EP\_ENP\_CH1$
A2	$039\_38\_4\_RAF\_EP\_ENP\_CH1$
A3	$022\_3\_4\_RAF\_EP\_ENP$
A4	$064\_32\_2\_RAF\_EP\_ENP\_CH2$
B1	$022\_RAF\_EP\_ENP\_S22\_J1$
B2	$023\_RAF\_EP\_ENP\_S23\_J3$
B3	$025\_RAF\_EP\_ENP\_S22\_J3$
B4	$028\_CH1\_RAF\_EP\_ENP\_S22\_J2$
B5	$028\_CH2\_RAF\_EP\_ENP\_S23\_J3$
B6	$029\_RAF\_EP\_ENP\_S21\_J6$
B7	$035\_CH1\_RAF\_EP\_ENP\_S22\_J3$
B8	$035\_CH2\_RAF\_EP\_ENP\_S22\_J3$
B9	$039\_CH1\_RAF\_EP\_ENP\_S22\_J4$
B10	$039\_CH3\_RAF\_EP\_ENP\_S22\_J4$
B11	$048\_CH3\_RAF\_EP\_ENP\_S22\_J3$
B12	$048\_CH2\_RAF\_EP\_ENP\_S22\_J3$
B13	$064\_CH1\_RAF\_EP\_ENP\_S22\_J3$
B14	$064\_CH2\_RAF\_EP\_ENP\_S22\_J4$

Références Bibliographiques

Achlioptas, D., Gomes, C., Kautz, H., Selman, B. 2000. « Generating satisfiable instances ». In *Proc. of the National Conference on Artificial Intelligence*, p. 256–261.

Aloul, F., Ramani, A., Markov, I., Sakallah, K. 2002. « Generic ILP versus Specialized 0-1 ILP: An update ». In *ACM/IEEE Intl. Conf. Comp.-Aided Design*, p. 450-457.

Aloul, F., Ramani, A., Markov, I., Sakallah, K. 2002. « Solving Difficult SAT instances in the Presence of Symmetry ». In *Proc. of the Design Automation Conference*, p. 731–736.

Aloul, F., Sakallah, K. 2000.« An experimental evaluation of conflict diagnosis and recursive learning in boolean satisfiability ». In *Proc. of the International Workshop on Logic Synthesis (IWLS)*, p. 117–122.

Ansotegui, C., Val, A., Dotu, I., Fernandez, C., Manya, F. 2004. « Modeling Choices in Quasigroup Completion: SAT vs. CSP ». In *Proc. of the National Conference on Artificial Intelligence*, p.137-142.

Audemard, G., Benhamou, B.,Siegel, P. 1999. « La méthode d'avalanche AVAL : une méthode énumérative pour SAT ». In *Journée nationale des problèmes NP Complets (JNPC 99)*, p. 17-25, INSA, Lyon.

Bailleux, O., Boufkhad, Y. 2004. « Full CNF Encoding: The Counting Constraints Case ». In *Seventh International Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, p. 263–268.

Bessiere, C., Brito, I., Maestre, A. ,Meseguer, P. 2005 « Asynchronous Backtracking without Adding Links : A New Member in the ABT Family ». *L'intelligence artificielle*, vol. 161, p. 7–24.

Boivin, S. 2005. *Résolution d'un Problème de Satisfaction de Contraintes pour l'Ordonnancement d'une Chaine d'Assemblage Automobile.* Mémoire de maîtrise, Université du Québec à Montréal, Montréal.

Boivin, S., Gravel, M., Krajecki, M., Gagné, C. 2005. « Résolution du problème de car-sequencing à l'aide d'une approche de type fc ». In *JFPC*, Lens, France, p. 11–20.

Boutevin, C., Gourgand, M., Norre, S. 2001. « Formalisation Simplifiée et Résolution Approchée du Problème de Véhicules ». 3^{eme} *Conférence Francophone de MOdélisation et SIMulation "Conception, Analyse et Gestion des Systèmes Industriels" MOSIM'01, Université de Technologie de Troyes (UTT), Troyes (France)*

Chai, D., Kuehlmann, A. 2005. « A Fast Pseudo-Boolean Constraint Solver ». *IEEE Transaction on Computer-Aided Design of Integrated Circuits and Sytems*, vol. 24, no. 3.

Cori, R., Lascar, D. 2003. Logique mathématique, tome 1. 3^{eme} éd. cor. Paris : Éditions Masson, 385 p.

Cook, S. A. 1971. « The complexity of theorem proving procedures ». In *3rd ACM symp. on Theory of Computing*, p. 151–158, Ohio.

Davenport, A., Tsang, E. 1999. « Solving constraint satisfaction sequencing problems by iterative repair ». In *Proc. of the First International Conference on the Practical Applications of Constraint Technologies and Logic Programming*, p. 345-357.

Davis, M., Putnam, H. 1960, « A computing procedure for quantification theory ». In *Journal of the ACM*, vol. 7, p. 201–215.

Davis, M., Logemann, G., Loveland, D. 1962, « A machine program for theorem proving. In *Journal of the ACM*, vol. 7, p. 394–397.

Dechter, R., Frost, d. 2002. « Backjump-based backtracking for constraint satisfaction problems ». *Artificial Intelligence 136*, p. 147–188.

Drake, L., Frisch, A., Walsh, T. 2002. « Adding resolution to the DPLL procedure for Boolean satisfiability ». In *Proc. of 5th SAT*, p. 122-129.

Eén, N., Sorenson, N. 2003. « An Extensible SAT-solver ». In *SAT 2003*, vol. 2919 of LNCS, p. 502-518.

Eén, N., Sorenson, N. 2004. « Temporal Induction by Incremental SAT Solving ». *Electronic Notes in Theoretical Computer Science 89 no. 4.*

Estellon, B., Gardi, F., Nouioua,K. 2005. « Ordonnancement de véhicules dans les usines Renault : une approche par recherche locale à voisinage réduit ». In *Actes du 6ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*,Tours, p. 3–4.

Estellon, B., Gardi, F., Nouioua,K. 2005. « Ordonnancement de véhicules dans les usines Renault : une approche par recherche locale à voisinage large ». In *Actes du 6ème Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, Tours.

Estellon, B., Gardi, F., Nouioua,K. 2005. « Real-life car sequencing: very large neighbor-

hood search vs very fast local search ». *Soumis à European Journal of Operational Research.*

Frost, D., Dechter, R. 1994 « Dead-end driven learning ». In *Proc. AAAI*, P. 294–300, Seattle, Washington.

Gagné, C., Gravel, M., Price, W. 2005. « Solving real car sequencing problems with ant colony optimization ». À paraître dans *European Journal of Operational Research.*

Gent, I. P. 1998. « Two results on car sequencing problem ». *APES Research Report 02-1998.* Department of Computer Science, University of Strathclyde, Glasgow, Scotland, UK.

Gent, I. P., Walsh, T. 1993. « Towards an understanding of hillclimbing procedures for SAT ». In *Proc. of the Eleventh Nat'l Conf. on Artificial Intelligence (AAAI-93)*, p. 28-33.

Ginsberg, M.L. 1993. « Dynamic BackTracking ». *Journal of Artificial Intelligence Research 1*, p. 2–45.

Goldberg, E., Novikov, Y. 2002. « BerkMin: a fast and robust SAT-solver ». In *Proc. of the Design and Test in Europe Conference*, p. 142–149.

Gottlieb, J., Puchta, M., Solnon, C. 2003. « A Study of Greedy, Local Search, and Ant Colony Optimization Approaches for Car Sequencing Problems ». *Applications of evolutionary computing (EvoCOP 2003)*, p. 246–257.

Gravel, M., Gagné, C., Price, W. 2005. « Review and comparison of three methods for the solution of the car sequencing problem ». *Journal of the Operational Research Society*, vol. 56, no. 11, p. 1287–1295.

Hooker, J. N., Vinay, V. 1995. « Branching rules for satisfiability ». *Journal of Automated Reasoning, vol. 15, no 3*, p. 359-383.

Li, C. M., 2000. « Integrating equivalency reasoning into Davis-Putnam procedure ». In *Proc. of the AAAI 00*, p. 291–296.

Li, C. M., 2000. « La propagation de contraintes et la procédure DPLL pour le raisonnement propositionnel ». *Compte rendu d'habilitation, Technique et Science Informatique*, vol. 20-N6/2001, p. 821–823.

Li, C. M., Anbulagan, A. 1997. « Heuristics based on unit propagation for satisfiability problems ». In *Proc. of the IJCAI'97*, p. 366–371.

Li, C. M., Anbulagan, A. 1997. « Why a random 3-SAT problem is harder than another? ». In *Proc. of the JNPC'97*, p. 49–54.

Moskewicz, M., Madigan, C., Zhao, Y., Zhang, L., Malik, S. 2001. « Chaff : Engineering an efficient sat solver ». À paraître dans *39th Design Automation Conference, Las Vegas.*

Nguyen, A., Cung, V. D. 2005. « Le problème du Car Sequencing Renault et le Challenge ROADEF'2005 ». In *Actes des 1eres Journées Francophones de Programmation par Contraintes*, p. 3–10. Centre de Recherche en Informatique de Lens Université d'Artois, Lens, France.

Renault.
http://www.renault.com/renault_com/fr/main/index.aspx

Roadef challenge 2005.
http://www.prism.uvsq.fr/vdc/ROADEF/CHALLENGES/2005/challenge2005

Roadef'2005.
http://www.ocea.li.univ-tours.fr/roadef05/

Ryan, L. 2004. *Efficient Algorithms for Clause-Learning SAT Solvers.* Mémoire de maîtrise, Simon Fraser University , Canada.

SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings, 2004.

SATO: A Solver for Propositional Satisfiability.
http://www.cs.uiowa.edu/ hzhang/sato/

Tompkins, A., Hoos, H. 2004. « An Implementation and Experimentation Environment for SLS Algorithms for SAT and MAX-SAT ». In *Seventh Intl Conf. on Theory and Applications of Satisfiability Testing (SAT2004)*, Vancouver, p. 37–46.

Tsang, E. 1999. « A Glimpse of Constraint Satisfaction ». *Artificial Intelligence Review 13*, p. 215–227.

Zhang, H. 2002. « Generating college conference basketball schedules by a SAT Solver », *Proc. of 5th Int. Symposium on the Theory and Applications of Satisfiability Testing*, p. 281–291.

Zhang H. 1997. « SATO: An Efficient Propositional Prover ». in *proc Int. Conference on Automated Deduction (CADE'97), CADE*, vol. 1249, p. 272–275.

Zhang, H., Li, D., Shen H. 2004. « A Sat Based Scheduler for Tournament Schedules ». In *The Seventh Int. Conference on Theory and Applications of Satisfiability Testing (SAT 2004)*, Vancouver, BC, Canada, p. 191–196.

Zhang, L., Madigan, C., Moskewicz, M., Malik S. 2001. « Efficient Conflict Driven Learning in a Boolean Satisfiability Solver ». In *Proc. of the Int. Conference on Computer- Aided Design (ICCAD'01)*, p. 279–285.

Zhang, H., Stickel, M. 1996. « SATO: An Efficient Algorithm for Unit Propagation ». In *Proc. of the Fourth Int. Symposium on Artficial Intelligence and Mathematics*

(AI-MATH'96), (Fort Lauderdale (Florida USA)), p. 166–169.

Zhang, H., Stickel, M. 2000. « Implementing the Davis-Putnam Method ». *J. Autom. Reasoning, 24(1/2)*, p. 277–296.