Titre: Title:	Algorithmes pour la prise de décision distribuée en contexte hiérarchique
Auteur: Author:	Jonathan Gaudreault
Date:	2009
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Gaudreault, J. (2009). Algorithmes pour la prise de décision distribuée en contexte hiérarchique [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/132/

UNIVERSITÉ DE MONTRÉAL

ALGORITHMES
POUR LA PRISE DE DÉCISION DISTRIBUÉE
EN CONTEXTE HIÉRARCHIQUE

JONATHAN GAUDREAULT
DÉPARTEMENT DE GÉNIE INFORMATIQUE ET GÉNIE LOGICIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIAE DOCTOR
(GÉNIE INFORMATIQUE)
MAI 2009

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

ALGORITHMES POUR LA PRISE DE DÉCISION DISTRIBUÉE EN CONTEXTE HIÉRARCHIQUE

présentée par : GAUDREAULT Jonathan

en vue de l'obtention du diplôme de: Philosophiae Doctor

a été dûment acceptée par le jury d'examen constitué de :

M. DESMARAIS Michel, Ph.D., président

M. PESANT Gilles, Ph.D., membre et directeur de recherche

M. FRAYRET Jean-Marc, Ph.D., membre et codirecteur de recherche

M. ROUSSEAU Louis-Martin, Ph.D., membre

M. SHEN Weiming, Ph.D., membre

à Nathalie, Jérémie et Jean-Victor
mon grand amour, mes trésors

Remerciements

Je tiens d'abord à remercier mes directeur et codirecteur de thèse, messieurs Pesant et Frayret. Monsieur Pesant a accepté avec enthousiasme de diriger cette thèse; j'ai bénéficié et appris énormément de cette collaboration. Quant à monsieur Frayret, je n'arrive pas à me rappeler l'avoir réellement *choisi* comme codirecteur : à l'époque, je lui ai présenté mon projet et la relation s'est développée naturellement, progressivement. Ses encouragements sans cesse renouvelés ont constitué un instrument des plus précieux.

Merci également à madame Sophie D'Amours qui m'a accueilli au Consortium de recherche FORAC, il y a sept ans déjà. Elle dispose d'une faculté unique : celle de permettre aux gens de s'épanouir et de réaliser leurs rêves, en acceptant d'aller au-delà des conventions, hors des sentiers battus. Tout ceci aurait été impossible sans elle. Je lui en suis très reconnaissant.

Également, je ne pourrais passer sous silence la contribution de monsieur Alain Rousseau. Il fut mon mentor – mon maître – à mon arrivée au consortium. Merci aussi à Constance Van Horne – présidente de mon « fan club » – qui m'a tant encouragé à entreprendre un doctorat. Merci également à Claude-Guy Quimper pour ses conseils judicieux tout au long de ce projet. Ton amitié m'est précieuse.

Et surtout, merci aux membres de ma famille pour leur soutien indéfectible, plus particulièrement à Nathalie, Éric, et ma mère. Par le jeu des vases communicants, Nathalie s'est investie énormément pour que je puisse réaliser ce travail; je lui dois beaucoup.

En terminant, merci à mon fils Jérémie, qui du haut de ses quatre ans, m'a prodigué le conseil suivant : « Papa, quand ça ne va pas, il faut garder son calme, ne pas se décourager et continuer. Comme Maurice Richard ».

Résumé

Cette thèse a pour objet la coordination entre entités autonomes. De manière plus précise, nous nous intéressons à la coordination dans un contexte hiérarchique. Les problèmes étudiés montrent les caractéristiques suivantes : (1) il s'agit de problèmes d'optimisation distribués, (2) le problème est naturellement décomposé en sous-problèmes, (3) il existe *a priori* une séquence selon laquelle les sous-problèmes doivent être résolus, (4) les sous-problèmes sont sous la responsabilité de différentes entités et (5) chaque sous-problème est défini en fonction des solutions retenues pour les sous-problèmes précédents.

Parmi les principaux domaines d'application, on trouve les systèmes d'aide à la décision organisationnels et les problèmes de synchronisation dans les chaînes logistiques industrielles. Ce dernier domaine sert de fil conducteur dans cette thèse : le travail de plusieurs unités de production est nécessaire pour fabriquer et livrer les commandes des clients. Différentes alternatives sont possibles en ce qui a trait aux pièces à utiliser, au choix des processus de fabrication, à l'ordonnancement des opérations et au transport. Chaque partenaire désire établir son plan de production (quoi faire, où et quand le faire), mais il est nécessaire pour eux de coordonner leurs activités.

Les méthodes utilisées en pratique industrielle peuvent être qualifiées d'*heuristiques de coordination*. À l'opposé, il existe des algorithmes d'optimisation distribués et *exacts*, notamment les techniques de raisonnement sur contraintes distribuées (*Distributed Constraint Optimization Problems*, ou DCOP). Cependant, ces derniers algorithmes s'accommodent mal de la nature hiérarchique des problèmes étudiés et pourraient difficilement être utilisés en pratique. Les forces et les faiblesses des méthodes *heuristiques* et *exactes* nous ont donc amené à proposer de nouvelles approches.

Nous proposons d'abord un formalisme appelé HDCOP (pour *Hierarchical* DCOP). Il permet de représenter formellement le problème, l'attribution des responsabilités entre les agents, la séquence de résolution et l'espace des solutions accessibles aux agents. L'espace de solutions est représenté par un arbre, ce qui permet aux agents d'utiliser un algorithme de recherche distribué de base en tant que mécanisme de coordination (e.g. SyncBB).

Cet espace de coordination montre certaines caractéristiques particulières. Il s'agit d'un arbre non-binaire de profondeur fixe ayant un facteur de branchement très grand et variable d'un nœud à l'autre. Également, l'arbre est généré dynamiquement pendant la résolution. Dans ce contexte, même pour de très grands temps de calcul, un algorithme réalisant une recherche en profondeur (tel SyncBB) visite uniquement des solutions très semblables les unes des autres (puisque contenues dans la même zone de l'arbre).

Pour remédier à ce problème, nous avons adapté au contexte multi-agent des stratégies de recherche réputées efficaces en environnement centralisé, à savoir les méthodes basées sur l'analyse des *déviations* (e.g. LDS). Ces méthodes sont très utilisées en programmation par contraintes classique. Nous proposons deux adaptations distribuées. Le premier algorithme (SyncLDS) est très simple d'implémentation mais permet le travail d'un seul agent à la fois. Le second (MacDS) permet aux agents de travailler simultanément. De plus, il montre certaines propriétés intéressantes pour un algorithme distribué : tolérance aux pannes de communication et à l'inversion de l'ordre des messages. Les deux méthodes permettent aux agents de découvrir de bien meilleures solutions qu'avec les méthodes de base. Cependant, MacDS réduit davantage le temps de calcul nécessaire pour l'atteinte d'une solution de qualité donnée.

Finalement, nous avons proposé une stratégie de recherche *adaptative* appelée ADS. Les agents utilisent une forme d'apprentissage pour établir dynamiquement et collectivement quelles zones de l'arbre devraient être explorées en priorité. La méthode prend appui sur le fait que les arbres sont non-binaires; un modèle permet d'anticiper l'impact associé à la réalisation d'un retour-arrière vers un nœud donné, en se basant sur la qualité des solutions obtenues auparavant. L'efficacité du processus de coordination s'en voit grandement améliorée.

Toutes ces méthodes ont été évaluées pour un cas réel de coordination dans une chaîne logistique de l'industrie forestière. Nous les avons également évaluées pour un large éventail de problèmes synthétiques, de manière à illustrer différentes propriétés des algorithmes.

Abstract

This thesis concerns multiagent coordination in hierarchical settings. These are distributed optimization problems showing the following characteristics: (1) the global problem is naturally decomposed into subproblems, (2) a sequence, defined *a priori*, exists in which the subproblems must be solved, (3) various agents are responsible for the subproblems, and (4) each subproblem is defined according to the solutions adopted for the preceding subproblems.

Organizational distributed decision making and *Supply chain coordination* are among the main application domains. The latter case is more thoroughly studied in this thesis. In this kind of problem, the cooperation of several facilities is needed to produce and deliver the products ordered by external customers. However, different alternatives are possible regarding the parts to use, the manufacturing processes to follow, the scheduling of operations and the choice of transportation. Therefore, supply chain partners must coordinate their local decisions (e.g. what to do, where and when), with the common objective of delivering the ordered products with the least possible delay.

The most commonly used coordination mechanisms can be described as *heuristics*. In contrast, some generic and *complete* distributed algorithms exist – researchers in *Distributed Artificial Intelligence* (DAI) have proposed a generic framework called *Distributed Constraint Optimization Problem* (DCOP). However, there are certain difficulties in mapping the actual business context (which is highly hierarchical) into the DCOP framework. Thus, based on the strengths and weaknesses of both the complete and heuristic approaches, we propose new approaches.

We first introduce a framework called HDCOP (for *Hierarchical* DCOP). It allows formal modeling of the global problem, agents' responsibilities, the solving sequence and the solution space accessible to the agents. Within HDCOP, the coordination space

is modelled as a tree. This calls for the use of distributed tree search algorithm (e.g. SyncBB) as the coordination mechanism.

However, the coordination space shows particular characteristics. It is a non-binary tree with a fixed depth, with a huge branching factor that varies from one node to another. Moreover, the tree is not fully defined *a priori*; it is 'revealed' progressively during the search process. In such a context, even for huge computation times, algorithms applying depth-first search (like SyncBB) persist in exploring only minor variations of the first global solution (they differ only from the solution for the last subproblems).

To overcome this issue, we have adapted centralized search strategies based on the computation of *discrepancies* (e.g. LDS) for their use in multi-agent contexts. These methods allow systematic searching of the solution space (thus look for the optimal solution) but aim at producing good solutions in a short period of time. We have proposed two distributed adaptations of this idea. The first algorithm (SyncLDS) is easier to implement, but allows the computation of only one agent at a time. The second (MacDS) allows agents to work concurrently so as to speed up the search process. It is also more robust, as it uses asynchronous communication and tolerates random message transmission delays.

Finally, we have proposed ADS, an adaptive backtracking strategy based on the analysis of discrepancies. It enables the agents to collectively and dynamically *learn* which areas of the tree are most promising in order to visit them first. The strategy exploits the fact that the search tree is not binary. Based on the quality of previous solutions, the model seeks to extrapolate for which node it would be more profitable to visit alternative arcs first. This significantly speeds up the search process.

These methods were evaluated using a real industrial supply chain problem in the forest products industry. In order to generalize results and study different properties of the algorithms, we also evaluated them for a wider range of problems using generated data.

Table des matières

DÉDICACE III

REMERCIEMENTS IV

RÉSUMÉ V

ABSTRACT VIII

TABLE DES MATIÈRES X

LISTE DES TABLEAUX XIV

LISTE DES FIGURES XIX

LISTE DES SIGLES ET ABRÉVIATIONS XVII

LISTE DES ANNEXES XVIII

AVANT-PROPOS XIX

CHAPITRE 1 INTRODUCTION 1

1.1 Réseaux de création de valeur 1

1.2 Coordination des opérations 2

1.3 Principaux obstacles à l'utilisation d'une approche centralisée 5

1.4 Contraintes de nature organisationnelle régissant les interactions 5

1.5 Organisation de la thèse 7

CHAPITRE 2 UN CAS INDUSTRIEL: COORDINATION DANS UN RÉSEAU DE PRODUCTION DE BOIS D'ŒUVRE 9

2.1 Unité de sciage 10

2.2 Unité de séchage 12

2.3 Unité de finition 14

2.4 Prise de décision locale 15

2.5 Conclusion 17

CHAPITRE 3 APPROCHES POUR LA COORDINATION D'ENTITÉS AUTONOMES 18

3.1 Aperçu 18
3.2 Méthodes basées sur la communication et la négociation 21
3.2.1 Heuristiques de coordination 22
3.2.2 Le modèle de Schneeweiss pour la formulation des problèmes de décision distribués 24
3.3 Méthodes basées sur l'algorithmie et le calcul 25
3.3.1 Distributed Constraint Satisfaction Problem (DisCSP) 26
3.3.2 Distributed Constraint Optimization Problem (DCOP) 32
3.4 Conclusion 39

CHAPITRE 4 HIERARCHICAL DISTRIBUTED CONSTRAINT OPTIMIZATION PROBLEM (HDCOP) : UN CADRE POUR LA PRISE DE DÉCISION DISTRIBUÉE EN CONTEXTE HIÉRARCHIQUE 41

4.1 Modéliser le problème de coordination en contexte hiérarchique sous la forme d'un DCOP 42
4.1.1 Situations avec plusieurs variables par agent 43
4.1.2 Respect du caractère privé des informations 44
4.2 Obstacles à l'utilisation du cadre DCOP dans le contexte étudié 44
4.2.1 Autonomie limitée des agents 45
4.2.2 Évacuation de la notion de hiérarchie 45
4.2.3 Capacité d'anticipation 46
4.2.4 Omnipotence 47
4.2.5 Synthèse des objections 48
4.3 Introduction du cadre proposé (HDCOP) 48
4.4 Recherche distribuée dans un arbre en tant que mécanisme de coordination 51

Algorithme de base 51
4.5 Discussion 52
4.5.1 La notion d'optimalité 52
4.5.2 Coordination dans des réseaux de création de valeur non linéaires ... 53
4.5.3 Exploitation de l'approche proposée au sein d'un système industriel de prise de décision 55
4.6 Conclusion 57

CHAPITRE 5 MULTI-AGENT CONCURRENT DISCREPANCY SEARCH (MACDS) : RETOUR-ARRIÈRE BASÉ SUR L'ANALYSE DES DÉVIATIONS EN CONTEXTE MULTI-AGENT 59

5.1 Limited Discrepancy Search (LDS) 60
5.2 Synchronous LDS (SyncLDS) 61
5.2.1 Pseudocode 62
5.3 Multi-agent Concurrent Discrepancy Search (MacDS) 65
5.3.1 Présentation générale 65
5.3.2 Pseudocode 67
5.3.3 Caractéristiques de l'algorithme 69
5.3.4 Gestion de la mémoire 70
5.4 Expérimentations 72
5.4.1 Évaluation pour le cas industriel 73
5.4.2 Évaluation avec des données synthétiques 76
5.5 Discussion 78
5.6 Conclusion 79

CHAPITRE 6 ADAPTIVE DISCREPANCY SEARCH (ADS) : UNE STRATÉGIE DE RECHERCHE ADAPTATIVE 81

6.1 Idée générale 82
6.2 Anticiper la contribution associée à la réalisation d'une déviation supplémentaire 84

6.3 Stratégie de retour-arrière exploitant le modèle 88
6.3.1 Mise à jour du modèle 89
6.3.2 Choix du candidat pour le retour-arrière 90
6.4 Synchronous Adaptive Discrepancy-based Search (SyncADS) 91
6.5 Expérimentations 94
6.5.1 Évaluation avec les données industrielles 94
6.5.2 Généralisation – Évaluation avec des données synthétiques 99
6.6 Travaux apparentés 104
6.6.1 Approche par entraînement 105
6.6.2 Approche adaptative 106
6.7 Conclusion 108
CHAPITRE 7 CONCLUSION 109
7.1 HDCOP 109
7.2 Introduction de nouveaux algorithmes d'optimisation distribués 110
7.3 Utilisation d'un mécanisme d'apprentissage pour l'optimisation distribuée 111
7.4 Limites 112
7.5 Potentiel d'application en contexte centralisé 113
7.6 Applications industrielles 115
RÉFÉRENCES 116
ANNEXES 126

Liste des tableaux

Tableau 1.1 : Éléments constitutifs d'un cadre logistique. 6

Tableau 2.1 : Planification des opérations d'une scierie. 16

Tableau 3.1. Approches exploitées par les méthodes d'optimisation distribuées. 21

Tableau 3.2. Contraintes du problème représenté à la figure 3.4. 28

Tableau 5.1 : Caractéristiques des algorithmes comparés dans le cadre de l'évaluation de MacDS. 72

Tableau 6.1 : ADS – Réduction (en %) du temps de calcul requis pour l'obtention de la meilleure solution pour les cas industriels étudiés 96

Tableau 6.2 : ADS – Réduction moyenne du temps de calcul (en %) pour l'obtention de solutions de qualité égale ou supérieure à celles obtenues avec la stratégie de retour-arrière LDS. 97

Liste des figures

Figure 1.1 : Exemple de réseau de création de valeur. 2
Figure 1.2 : Flux de produits entre deux agents. 3
Figure 2.1 : Production de bois d'œuvre. 9
Figure 2.2 : Représentation d'une matrice de production. 11
Figure 2.3 : Exemples de patrons de chargement. 12
Figure 2.4 : Exemples de processus de séchage constitués de plusieurs activités. 13
Figure 2.5 : Plan de production pour une ligne de rabotage. 15
Figure 2.6 : Flux de produits entre les unités de sciage, de séchage et de finition 17
Figure 3.1 : Planification de l'amont vers l'aval, en flux poussé. 23
Figure 3.2 : Planification de l'aval vers l'amont, en flux tiré. 23
Figure 3.3 : Planification en deux phases. 24
Figure 3.4 : Exemple de DisCSP. 27
Figure 3.5 : Résolution d'un DisCSP avec SyncBT. 30
Figure 3.6 : Résolution d'un DisCSP avec ABT. 31
Figure 3.7 : Exemple de DCOP sous ADOPT. 36
Figure 3.8 : Résolution d'un DCOP avec ADOPT. 36
Figure 4.1 : Modélisation d'un DCOP avec plusieurs variables par agent. 42
Figure 4.2 : Reformulation d'un DCOP avec une seule variable par agent. 44
Figure 4.3 : Application de la planification en deux phases par un collectif de trois agents. 49
Figure 4.4 : Espace de coordination sous HDCOP. 50
Figure 4.5 : Application de SyncBB pour la résolution d'un HDCOP. 52
Figure 4.6 : Flux de produits dans un RCV convergent. 53
Figure 4.7 : Séquence de sous-problèmes pour la coordination dans un RCV convergent. 54
Figure 4.8 : Scénario alternatif pour la coordination dans un réseau RCV convergent. 55
Figure 5.1: Arbre binaire et nombre de déviations associées à chaque feuille. 60
Figure 5.2 : SyncLDS – Pseudocode. 64
Figure 5.3 : MacDS – Trace d'exécution partielle. 66

Figure 5.4 : MacDS – Pseudocode du fil de contrôle de l'agent. 68
Figure 5.5 : MacDS – Pseudocode pour le fil d'exécution associé à un nœud. 69
Figure 5.6 : MacDS – Réduction de la fonction objectif en fonction du temps de calcul pour les cas industriels étudiés. 75
Figure 5.7: MacDS – Temps de calcul espéré pour l'atteinte de la meilleure solution en fonction du niveau de corrélation entre la qualité d'une feuille et le nombre de déviations. 77
Figure 5.8: MacDS – Temps de calcul espéré pour l'atteinte de la meilleure solution (SyncBB utilisé en comme référence) 77
Figure 5.9: MacDS – Temps de calcul espéré pour l'atteinte de la meilleure solution en fonction de la valeur de différents paramètres environnementaux. 80
Figure 6.1 : ADS – Exemple d'arbre de recherche. 83
Figure 6.2 : ADS – Relation entre `ArcValue ( )` et `bestToDate [ ]`. 85
Figure 6.3 : ADS – Fonction d'extrapolation `F (i)` proposée. 87
Figure 6.4 : ADS – Pseudocode pour la mise à jour de `bestToDate [ ]` et `F ( )` 90
Figure 6.5 : ADS – Pseudocode pour la sélection du candidat au retour-arrière. 91
Figure 6.6 : SyncADS – Pseudocode. 93
Figure 6.7 : Distribution des valeurs β dans les nœuds des arbres correspondant aux problèmes industriels étudiés. 99
Figure 6.8 : Comparaison des stratégies ADS et LDS pour un arbre de petite taille. 101
Figure 6.9 : Impact du nombre de sous-problèmes sur les performances des stratégies LDS et ADS. 102
Figure 6.10 : Impact du paramètre γ sur les performances des stratégies LDS et ADS. 103
Figure 6.11 : Impact du paramètre δ sur les performances des stratégies LDS et ADS. 104
Figure A2.1 : Illustration intuitive du modèle de planification des opérations de séchage. 131

Liste des sigles et abréviations

ADOPT ..... Asynchronous Distributed Optimization
ADS ..... Adaptive Discrepancy Search
APS ..... Advanced Planning and Scheduling
COP ..... Constraint Optimization Problem
CSA ..... Concurrent Search Algorithm
CSP ..... Constraint Satisfaction Problem
DCOP ..... Distributed Constraint Optimization Problem
DCR ..... Distributed Constraint Reasoning
DDS ..... Depth-bounded Discrepancy Search
DFS ..... Depth-First Search
DisCSP ..... Distributed Constraint Satisfaction Problem
DPOP ..... Dynamic Programming Optimization
HDCOP ..... Hierarchical Distributed Constraint Optimization Problem
IDB ..... Iterative Distributed Breakout
LDS ..... Limited Discrepancy Search
MacDS ..... Multi-agent concurrent Discrepancy Search
MAP ..... Multiagent Agreement Problem
MIP ..... Mixed Integer Programming
RCV ..... Réseau de création de valeur
SyncADS ..... Synchronous Adaptive Discrepancy Search
SyncBB ..... Synchronous Branch and Bound
SyncBT ..... Synchronous Backtracking
SyncLDS ..... Synchronous Limited Discrepancy Search

Liste des annexes

Annexe 1 : Planification des opérations de sciage..... 126

Annexe 2 : Planification des opérations de séchage..... 130

Annexe 3 : Planification des opérations de finition..... 136

Avant-propos

Ce projet est né alors que j'assistais à la conférence ICAPS[1] au printemps 2005. Je suis revenu de Californie avec l'envie irrésistible d'étudier comment les méthodes d'intelligence artificielle distribuée pouvaient s'appliquer aux problèmes de gestion des opérations abordés au Consortium de recherche FORAC. J'avais le sentiment profond qu'il existait deux solitudes: deux communautés (intelligence artificielle et gestion des opérations) utilisant des vocabulaires différents et assistant à des conférences distinctes, possédant chacune des richesses de savoir lui permettant de contribuer à l'univers de l'autre.

Le projet se présentait au départ comme un simple exercice de transfert de connaissances d'une communauté à l'autre, mais il s'est rapidement montré sous un autre jour. L'application des modèles et des algorithmes aux problèmes étudiés était loin d'être triviale. Ce fut donc l'occasion d'entreprendre une grande aventure à laquelle j'aspirais depuis longtemps : le doctorat. C'était le projet rêvé pour moi : une problématique concrète, qui servirait de source de motivation et d'inspiration pour la création de nouveaux concepts et algorithmes, lesquels seraient considérés comme des avancées à la fois dans les domaines de l'intelligence artificielle distribuée et du génie industriel.

Cette thèse réunit les principales idées proposées durant cette période. Le cadre proposé, de même que les algorithmes pour la coordination d'entités autonomes ont donné lieu à deux articles de journaux (acceptés ou en révision)[2,3], six articles de conférences, d'ateliers ou de colloques[4-9]et deux communications[10,11]. En parallèle, des travaux sur le problème spécifique de production du bois d'œuvre ont donné lieu à trois articles de journaux et de conférences (acceptés ou révision)[12-14]. Enfin, pendant cette période, des collaborations sur le thème des chaînes logistiques ont donné lieu à cinq autres articles dont je suis coauteur.

---

[1] International Conference on Automated Planning and Scheduling.

[2] Gaudreault J., Frayret J. M. et Pesant G., "Distributed Search for Supply Chain Coordination", *Computers in Industry, Special Issue on Collaborative Engineering: from Concurrent Engineering to Enterprise Collaboration*, vol 60, no 6, 2009.

[3] Gaudreault J., Pesant G., Frayret J. M. et D'Amours S., "Efficient Distributed Decision Making in Supply Chains Using an Adaptive Search Strategy". *Soumis en 2008 et présentement en cours de révision. Une version préliminaire peut être consultée sous la forme d'un document de travail (CIRRELT-2008-49).*

[4] Gaudreault J., Frayret J. M. et Pesant G., "Retour-arrière basé sur les divergences pour l'optimisation distribuée", *Troisièmes journées francophones de programmation par contraintes (JFPC)*, Rocquencourt, France, 2007, pp. 237-244.

[5] Gaudreault J., Frayret J. M. et Pesant G., "Discrepancy-based Method for Distributed Supply Chain Operations Planning", *Doctoral Consortium of the International Conference on Automated Planning and Scheduling (ICAPS)*, Providence, RI, 2007.

[6] Gaudreault J., Frayret J. M. et Pesant G., "Discrepancy-based Optimization for Distributed Supply Chain Operations Planning", *Proceedings of the Ninth International Workshop on Distributed Constraint Reasoning (DCR)*, Providence, RI, 2007.

[7] Gaudreault J., Frayret J. M. et Pesant G., "Discrepancy-based Method for Hierarchical Distributed Optimization", *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, Patras, Grèce, 2007, pp. 75-81.

[8] Gaudreault J., Frayret J.-M., Pesant G., Forget P. et D'Amours S., "Operations Coordination in the Lumber Industry: from Heuristics to Machine Learning", *Proceedings of the 13th Annual International Conference on Industrial Engineering Theory, Applications & Practice*, Las Vegas, NV, 2008.

[9] Gaudreault J., Pesant G., Frayret J. M. et D'Amours S., "ADS: An Adaptive Search Strategy for Efficient Distributed Decision Making", *Learning and Intelligent Optimization Conference (LION)*, Trente, Italie, 2009.

[10] Gaudreault J., Frayret J. M. et Pesant G., "Discrepancy-based Search for Multi-Agent Optimization", *Optimization Days Conference (JOPT)*, Montréal, QC, 2007.

[11] Gaudreault J., Frayret J. M. et Pesant G., "Adaptive Discrepancy Search", *Optimization Days Conference (JOPT)*, Montréal, QC, 2009.

[12] Gaudreault J., Frayret J. M., Rousseau A. et D'Amours S., "Combined planning and scheduling in a divergent production system with co-production". *Soumis en 2008 et présentement en cours de révision. Une version préliminaire peut être consultée sous la forme d'un document de travail (CENTOR DT-2006-JMF-1).*

[13] Gaudreault J., Forget P., Rousseau A., Frayret J. M. et D'Amours S., "Distributed Operations Planning in the Lumber Supply Chain: Models and Coordination". *Soumis en 2008 et présentement en cours de révision. Une version préliminaire peut être consultée sous la forme d'un document de travail (CIRRELT-2009-07).*

[14] Gaudreault J., Forget P., Frayret J.-M., Rousseau A. et D'Amours S., "A Multi-Agent and OR-based Approach to Operations Planning in the Lumber Industry", *Proceedings of the 13th Annual International Conference on Industrial Engineering Theory, Applications & Practice*, Las Vegas, NV, 2008.

Chapitre 1 Introduction

Cette thèse a pour objet la coordination entre entités autonomes. Dans le contexte étudié, ces entités ont un but commun. Elles doivent collaborer pour atteindre ce but, mais chacune a ses préférences quant à la façon d'y parvenir. Nous désirons permettre aux entités d'arriver à un compromis le plus rapidement possible.

De manière plus précise, nous nous intéressons à la coordination dans un contexte hiérarchique. Les problèmes étudiés montrent les caractéristiques suivantes : (1) il s'agit de problèmes d'optimisation, (2) le problème global est naturellement décomposé en sous-problèmes, (3) il existe *a priori* une séquence selon laquelle les sous-problèmes doivent être résolus, (4) les sous-problèmes sont sous la responsabilité de différentes entités et (5) chaque sous-problème est défini en fonction des solutions retenues pour les sous-problèmes précédents.

Parmi les principaux domaines d'application, on trouve les systèmes d'aide à la décision organisationnels et la coordination dans les chaînes logistiques industrielles (également appelées *réseaux de création de valeur*) [1]. C'est ce dernier domaine qui nous servira de fil conducteur pour la présentation et l'étude de la problématique.

1.1 Réseaux de création de valeur

Pour mener à bien sa mission, l'entreprise d'aujourd'hui doit collaborer avec de nombreux partenaires. Ces unités d'affaires ont des expertises variées et complémentaires. Elles forment un réseau: le réseau de création de valeur (RCV). La figure 1.1 présente un exemple de RCV. Dans cet exemple, les arcs représentent les flux de produits entre les différentes unités d'affaires constituant le réseau.

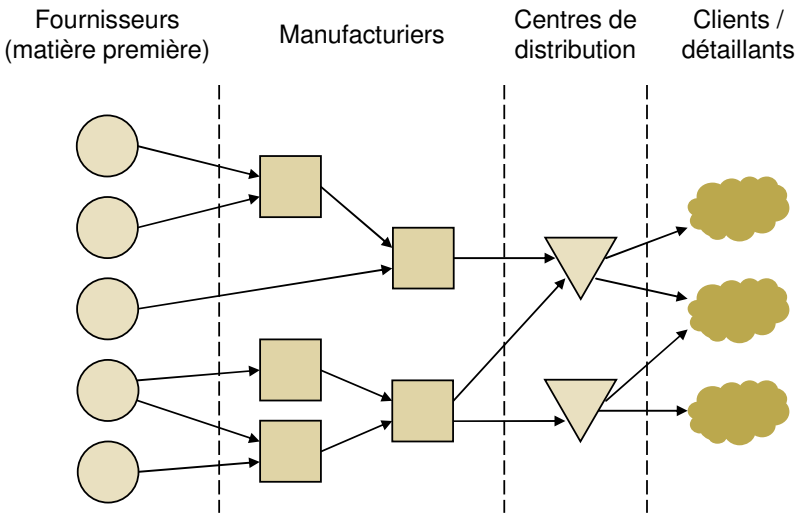


Figure 1.1 : Exemple de réseau de création de valeur.
Adapté de [2].

Le domaine de la gestion des chaînes logistiques (en anglais, *supply chain management*, ou SCM) concerne la coordination des activités manufacturières et logistiques entre unités décisionnelles autonomes. Plusieurs aspects sont abondamment abordés dans la littérature, notamment la problématique du design de réseaux de production et de distribution, la gestion des inventaires, le design de contrats dans le cadre de relations client-fournisseur de même que le développement de systèmes de planification centralisés pour les chaînes logistiques. Pour notre part, nous nous intéresserons à la coordination des opérations à court terme.

1.2 Coordination des opérations

Il arrive que les unités du réseau doivent travailler ensemble de manière à atteindre un but commun. Voyons l'exemple d'un groupe d'unités ayant des expertises variées. Un client commande un produit et le travail de chaque unité est nécessaire pour fabriquer et livrer le produit au client. Différentes alternatives sont possibles, tant à ce qui a trait aux pièces à utiliser, au choix des processus de fabrication, à l'ordonnancement des opérations et au transport. Les partenaires désirent donc établir un plan de production commun (quoi faire, où et quand le faire). En anglais, on réfère souvent à ce problème

en utilisant les termes *collaborative production planning* [3] ou bien *supply chain coordination problem* [4].

Chaque unité est responsable de l'établissement de son plan de production local, mais des contraintes lient ces plans entre eux. Par exemple, une unité ne peut pas consommer un produit qui ne lui a pas été livré. Pour chaque paire d'entités, celles-ci doivent donc s'entendre sur les flux de produits qui seront échangés entre eux. La figure 1.2 présente deux agents (A_1 et A_2). Ils doivent s'entendre sur ce flux de produit ($L_{p,t}^{i,j}$ représente la quantité de produit p qui sera livré par l'unité i à l'unité j au temps t).

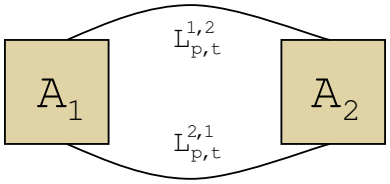


Figure 1.2 : Flux de produits entre deux agents.

Niveau d'autonomie des unités

Dans certains cas, chaque unité adhère entièrement à l'objectif global et n'a aucune préférence quant au plan de production à utiliser localement. Les unités sont alors dites « bénévoles » [5]. Idéalement, chaque unité peut anticiper comment son plan de production local peut contribuer au but commun, de manière à prendre les bonnes décisions.

Il est également possible que chaque unité soit pleinement autonome. L'unité sait planifier sa production en fonction de la demande de *son* client (qui est généralement une autre unité, pas nécessairement le client final). L'unité planifie sa production selon sa propre fonction d'utilité, mais peut proposer des plans alternatifs lorsqu'on lui indique que c'est nécessaire pour atteindre le but commun.

Il existe également des situations intermédiaires : une unité habituellement autonome est intégrée à un consortium (permanent ou non). Les unités désirent qu'un but global soit

atteint, mais conservent leurs préférences quant à leur plan local et leurs méthodes de planification.

D'un point de vue extérieur, toutes ces situations peuvent être décrites de la manière suivante. Nous disposons d'un groupe d'unités, chacune capable de planifier ses activités. Chacune le fait selon une méthode qui lui est propre (il peut s'agit d'un modèle d'optimisation[15] ou encore d'une heuristique). Certaines unités sont en mesure de proposer des plans alternatifs de manière à favoriser l'atteinte du but commun.

Optimisation

Dans l'exemple introduit précédemment, le but était de livrer à temps le produit commandé par le client final. Dans d'autres cas, le but prend la forme d'une fonction objectif à minimiser ou à maximiser: on cherchera, par exemple, à minimiser le retard pour le client, à réduire les coûts (en supposant une formule de partage des profits), ou autre.

Contraintes de temps

Le groupe d'unités ne dispose probablement pas d'un temps infini pour produire un plan. Le client final ne patientera pas éternellement avant de décider si l'offre du consortium lui convient.

Le temps de calcul accordé par le client peut être connu de manière exacte par le consortium, ou bien être seulement estimé. Le consortium cherchera donc à obtenir rapidement une solution. Une fois cette solution obtenue, et s'il reste du temps, alors il sera possible de chercher une solution alternative de meilleure qualité. Avec un temps infini, on s'attend à trouver la solution optimale. On parle alors d'algorithmes *à tout moment* (en anglais, *anytime algorithm*) [6,7].

---

[15] Ce modèle, appliqué localement, peut chercher à optimiser la fonction objectif globale ou une fonction d'utilité locale.

1.3 Principaux obstacles à l'utilisation d'une approche centralisée

Peut-on résoudre le problème de coordination décrit précédemment de manière centralisée? Il y a différents empêchements à une telle approche. Tout d'abord, mentionnons que le problème lui-même appelle une certaine forme de décomposition. Les différentes unités étant parfois très spécialisées (nous dirons qu'elles forment un réseau *hétérogène*), il serait difficile de concevoir un algorithme unique prenant en charge les spécificités de chaque sous-problème. De plus, en supposant que les unités forment des alliances d'une manière dynamique, il est utopique de penser qu'un nouvel algorithme centralisé puisse être développé à chaque fois.

Également, il existe souvent des contraintes de nature organisationnelle empêchant une approche centralisée. La nature même du problème local de planification de chacun peut être de nature privée. Les données peuvent être distribuées physiquement et certaines peuvent être de nature privée. Les unités constituant le collectif peuvent être des entreprises différentes ne désirant pas partager toutes leurs données. Par exemple, une unité peut ne pas être prête à partager l'information concernant sa capacité de production, ou encore l'information rendant possible la modélisation de ses préférences.

En conséquence, on fait face à un problème distribué par nature et il doit être abordé comme tel.

1.4 Contraintes de nature organisationnelle régissant les interactions

En contexte industriel, il existe la plupart du temps une certaine forme de *hiérarchie* dans le réseau. Par exemple, dans une relation client-fournisseur, le rôle de client est dans une certaine mesure assimilable à celui d'un donneur d'ordres dans une hiérarchie militaire.

Les relations entre les partenaires sont souvent régies par un *cadre logistique* défini au préalable. Ce cadre fait généralement l'objet d'une entente à long terme liant les organisations; il peut être formel ou informel, négocié ou *ad hoc*. Le mécanisme utilisé

par les partenaires pour coordonner leurs activités (*mécanisme de coordination*) n'est qu'un des éléments de ce cadre. Cependant, il doit être compatible avec les autres éléments.

Ce cadre logistique peut notamment comprendre les éléments suivants (tableau 1.1): (i) l'objet de la relation d'affaires (e.g. produits échangés, délais d'approvisionnement, quantités minimum ou maximum, pénalités, partage de profits, etc.); (ii) l'information que les entreprises acceptent d'échanger (e.g. niveaux de stocks, prévisions de ventes, bons de commandes, etc.); (iii) le mécanisme de coordination en tant que tel (e.g. règles et protocoles pour l'échange d'information; règles spécifiant quelle décisions doivent être prises, par qui, et quand, etc.).

La négociation de l'entente et son éventuelle acceptation par les partenaires dépend de considérations stratégiques, lesquelles ne font pas l'objet des présents travaux. Cependant, mentionnons qu'une entreprise acceptera cette entente uniquement s'il est rationnel pour elle de le faire (i.e. si cela peut s'inscrire dans sa logique d'affaires).

Tableau 1.1 : Éléments constitutifs d'un cadre logistique.

Niveau stratégique ➢ ***Définition d'une entente à long terme***	**Définition de la relation d'affaires (i)** ☐ Délais d'approvisionnement ☐ Quantités min/max ☐ Pénalités / Partage de profits
	Définition de l'information échangée (ii) ☐ Prévisions de ventes? ☐ Niveaux des stocks? ☐ Bons de commandes? ☐ Promesses de livraison?
	Définition des mécanismes de coordination utilisés (iii) ☐ Règles et protocoles pour l'échange d'informations ☐ Quelles décisions doivent être prises, par qui et quand
Niveau opérationnel (au jour le jour) ➢ ***Gestion des opérations conformément à l'entente***	**Application des mécanismes de coordination (iv)** **Utilisation d'algorithmes locaux pour la prise de décision (v)** (ex: planification de la production)

De plus, il ne faut oublier que l'entreprise devra gérer ses opérations conformément à l'entente sur une base quotidienne. Elle doit pouvoir appliquer le mécanisme de coordination (iv) et disposer d'algorithmes ou modèles de décisions (v) locaux pour réaliser les prises de décisions sous sa juridiction (e.g. planifier sa production). Il y donc là une contrainte implicite qui lie l'entente à long terme aux mécanismes opérationnels : l'agent doit disposer des algorithmes nécessaires pour prendre les décisions sous sa juridiction. Tout cela est à considérer lorsque vient le temps de proposer un mécanisme de coordination.

1.5 Organisation de la thèse

Tout au long de cette thèse, nous considérerons un cas précis de coordination des opérations dans un réseau industriel: celui de la production du bois de sciage. Ce cas sera présenté en détail au chapitre Chapitre 2.

Au chapitre Chapitre 3, nous présenterons les principaux mécanismes de coordination existants dans la pratique industrielle et dans la littérature scientifique. Nous verrons que les principales méthodes utilisées en pratique peuvent être qualifiées d'*heuristiques de coordination*. À l'opposé, il existe des algorithmes d'optimisation distribués *exacts*. Cependant, ces derniers s'accommodent mal de la nature hiérarchique des problèmes étudiés.

Les forces et les faiblesses des méthodes heuristiques et exactes nous ont amenés à proposer de nouvelles méthodes devant s'avérer plus performantes. Le chapitre Chapitre 4 introduit un formalisme appelé HDCOP. Il permet de modéliser le problème de coordination pour la prise de décision distribuée en contexte hiérarchique. L'espace de coordination est alors représenté par un arbre. Alors que les heuristiques utilisées par les industriels permettent de visiter uniquement la première feuille de cet arbre, nous montrerons comment il est possible de réaliser une recherche distribuée dans cet arbre tout en permettant aux agents de conserver les mêmes relations d'affaires et la même

attribution de responsabilités entre les agents. Le processus de coordination entre les agents est alors assimilable à une recherche distribuée dans un arbre.

Dans un tel contexte, la capacité des agents à découvrir rapidement de bonnes solutions dépendra de la stratégie de recherche dans l'arbre qui sera employée. Ainsi, au chapitre Chapitre 5, nous proposons d'adapter au contexte multi-agent des stratégies de recherche réputées efficaces en environnement centralisé, à savoir les méthodes basées sur l'analyse des *déviations*. Nous proposons deux adaptations. La première (SyncLDS) est très simple d'implémentation mais ne permet le travail que d'un seul agent à la fois. La seconde (MacDS) permet aux agents de travailler en simultané. Ces méthodes seront évaluées avec des données industrielles provenant du cas d'étude de même que pour des données synthétiques.

Au chapitre Chapitre 6, nous proposons une nouvelle stratégie de retour-arrière qui est dite *adaptative*. Les agents utilisent une forme d'apprentissage pour établir dynamiquement et collectivement quelles zones de l'arbre devraient être explorées en priorité. L'efficacité du processus de coordination s'en voit grandement améliorée.

Finalement, le chapitre Chapitre 7 synthétise les principales contributions de cette thèse.

Chapitre 2 Un cas industriel: coordination dans un réseau de production de bois d'œuvre

Dans ce chapitre, nous décrirons un cas industriel qui servira de support pour la présentation et l'évaluation des mécanismes de coordination présentés dans cette thèse. Ce cas concerne trois unités d'affaires impliquées dans la production de bois d'œuvre. Chaque unité est responsable d'une étape de transformation : sciage, séchage ou finition.

(1) L'unité de sciage transforme des billots en pièces de bois appelées *sciages*.

(2) L'unité de séchage retire une certaine quantité d'humidité des pièces de bois.

(3) L'unité de finition donne un fini lisse aux pièces et les classe selon leurs qualités et défauts. Les pièces peuvent être redécoupées de manière à obtenir des pièces d'une valeur combinée supérieure à celle de la pièce originale.

Pour chaque unité, les processus de transformation impliqués sont complexes. À chaque étape, pour un produit en entrée, plusieurs types de produits différents sont fabriqués (flux divergents), et cela de manière simultanée (coproduction). De plus, pour un produit en entrée, plusieurs processus de transformation alternatifs sont possibles à chaque étape. Le choix du processus influence les proportions de chaque produit qui sera obtenu.

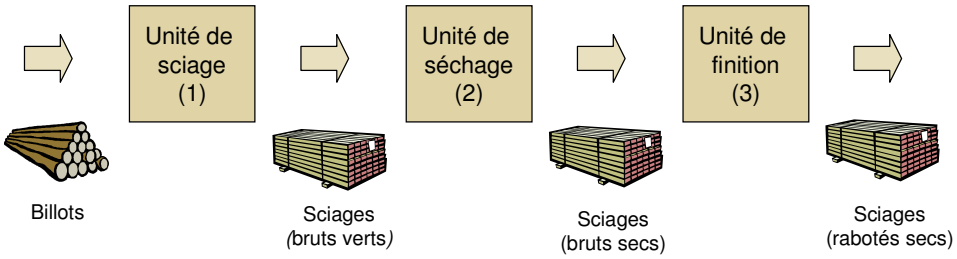


Figure 2.1 : Production de bois d'œuvre.

2.1 Unité de sciage

En pratique, le bois est récolté en forêt par des entrepreneurs indépendants selon un plan de travail préétabli. Dans les présents travaux nous ne considérons pas cette problématique de l'approvisionnement en matière première. Nous supposerons que l'inventaire initial est connu et que les arrivages subséquents le sont également, ce qui est une pratique courante en industrie.

Les billots[16] sont généralement stockés dans la cour à bois pendant une longue période avant d'être transformés en usine. Ils sont regroupés sous forme de piles, en fonction de caractéristiques physiques qui varient d'une entreprise à l'autre (e.g. essence, longueur, diamètre, provenance, etc.). Chaque pile se voit donc associée à une *classe* particulière. Il est à noter que les entreprises ne trient pas les billots individuellement à leur arrivée à l'usine; la totalité d'un chargement de camion se voit attribuer la même classe[17].

À l'usine, les billots sont découpés de manière à obtenir des pièces de bois qu'on appelle *sciages* (en anglais, *lumber*). À partir d'un lot de billots de même classe (et même à partir d'un seul billot) on obtiendra plusieurs sciages de dimensions différentes.

Les scieries conservent des historiques de production qui permettent d'estimer les quantités de chaque type de sciage devant être obtenues pour chaque classe de bille. Cette information définit une matrice de production. La figure 2.2 illustre l'information comprise dans une telle matrice. Les arcs montrent la quantité de chaque type de sciage qui est attendue, par volume de matière d'une classe donnée en entrée. Conformément à l'usage, les sciages sont identifiés par leur *dimension* (épaisseur et largeur, en pouces).

---

[16] Certaines usines sont plutôt en provisionnées en *tiges*; il s'agit d'arbres complets qui ont été ébranchés. Les billots (ou *billes*) sont obtenus lorsque l'opération de tronçonnage est réalisée en forêt, ce qui est la norme dans l'est canadien.

[17] Ainsi, les billes à l'entrée de l'usine de sciage peuvent être séparées par essence (par exemple) uniquement si ce tri a été réalisé en forêt.

Dans la plupart des scieries, la ligne de production peut être configurée selon différents *modes*. Pour chacun de ces modes nous aurons une matrice de production différente. Cela donne un certain contrôle sur le panier de produit en sortie. Certaines classes de billots sont incompatibles avec certains modes. Par exemple, dans la plupart des scieries le sapin et l'épinette ne peuvent être transformés pendant le même quart de travail; elles seront donc associées à des modes différents.

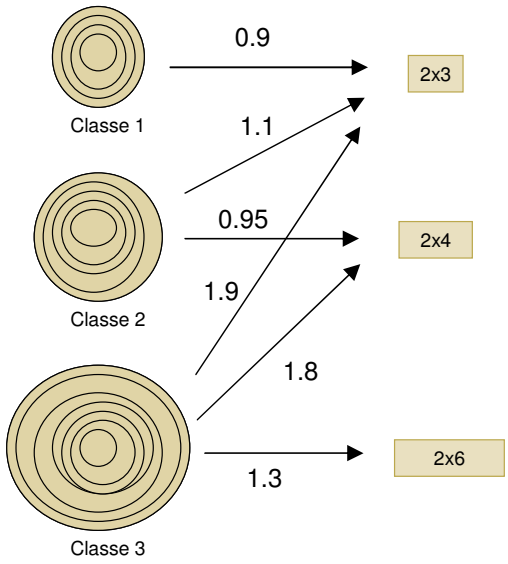


Figure 2.2 : Représentation d'une matrice de production.

Pour résumer, les décisions de planification de la production qui doivent être prises sont les suivantes : pour chaque quart de travail, décider (1) selon quel mode l'usine devrait être configurée, et (2) quelles quantités de chaque classe de billot devraient être consommées. On identifie par le fait même la production escomptée de l'usine.

À la sortie de l'usine, les sciages *verts* sont assemblés en *paquets* en attendant d'être attachés. Dans un même paquet on trouvera des sciages qui sont de la même *dimension*

(2”x3”, 2”x4”, etc.), de la même longueur (8 pieds, 12 pieds, etc.) et la plupart du temps de la même essence (sapin, épinette, etc.)[18].

2.2 Unité de séchage

Le séchage du bois est une opération qui vise à réduire sa teneur en humidité, de manière à rencontrer certaines normes industrielles et les besoins de certains clients spécifiques[19].

Le séchage du bois d’œuvre est un processus complexe qui s’étale sur plusieurs jours. Le séchage est réalisé en *lot* dans d’énormes séchoirs. Des paquets de différentes longueurs peuvent être séchés ensemble, mais toutes les pièces doivent être de même dimension (e.g. 2"x3"). Les paquets doivent être assemblés de manière à former un prisme rectangulaire remplissant le séchoir de manière quasi-complète. Il existe différentes contraintes reliées à la stabilité de cet empilement. Pour ces raisons, chaque scierie définit ses propres patrons de chargement pouvant être utilisés. La figure 2.3 montre deux exemples de patrons de chargements.

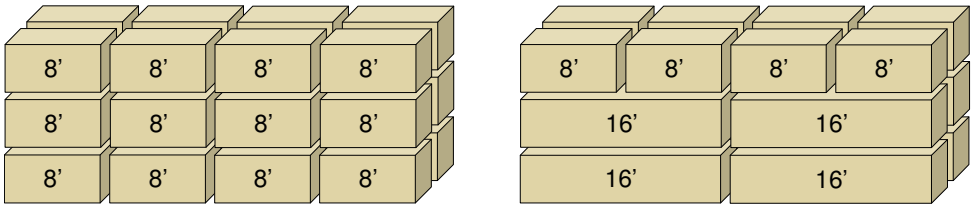


Figure 2.3 : Exemples de patrons de chargement.
Il s’agit d’exemples fictifs; un patron réel comporterait quelques centaines de paquets.

Dans certaines circonstances, une section spéciale de la cour à bois peut être utilisée pour réaliser un *pré-séchage* à l’air libre. Le pré-séchage dure quelques semaines et permet de réduire et équilibrer l’humidité dans les pièces. Cela réduit de manière

[18] Certains industriels regroupent plutôt les pièces en fonction de leur teneur en humidité, plutôt qu’en fonction de l’essence.

[19] Par exemple pour les pièces devant être utilisées pour le jointage et le collage.

importante le temps à passer au séchoir. Le pré-séchage joue également un rôle dans l'amélioration de la qualité du produit fini.

Pour un lot donné de sciages verts, il y a différentes combinaisons d'activités de séchage et de rabotage qui peuvent être utilisées. La figure 2.4 présente un exemple comportant deux activités différentes de pré-séchage et cinq activités de séchage. Les activités de pré-séchage diffèrent essentiellement par leur durée. Pour le séchage, elles diffèrent en fonction de paramètres comme la température de l'air, de contrôle d'humidité et de durée. Chaque chemin dans cette figure définit une séquence d'activités permettant de sécher le lot en entrée. Chaque chemin constitue un *processus* alternatif.

En résumé, les décisions de planification concernant le séchage sont donc les suivantes : (1) quels produits doivent être séchés, (3) selon quels patrons de chargement, (4) avec quels processus, (5) avec quels séchoirs et (6) à quel moment.

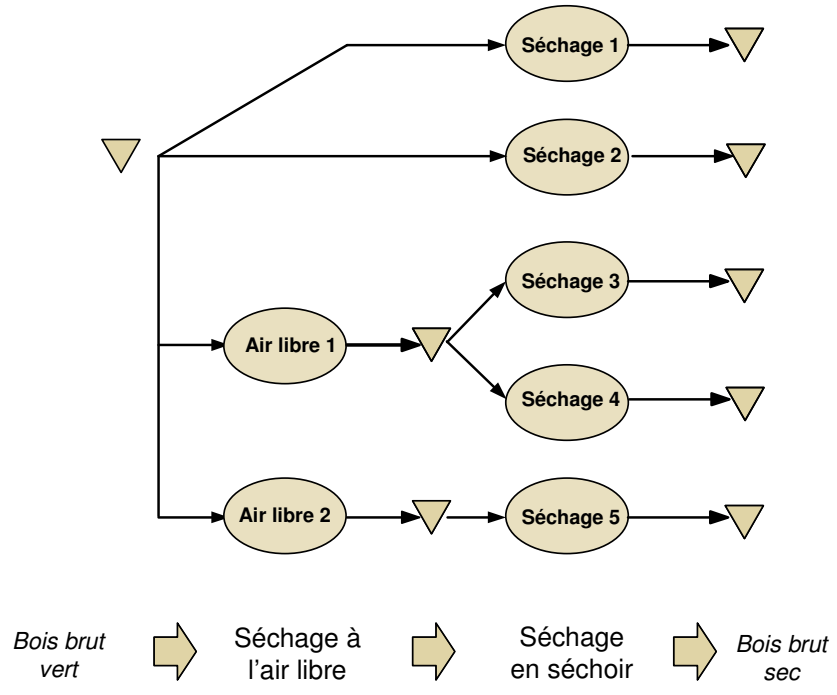


Figure 2.4 : Exemples de processus de séchage constitués de plusieurs activités.

2.3 Unité de finition

Au sein de l'unité de finition, les sciages sont d'abord *rabotés*. Cette opération permet de donner aux pièces la forme désirée[20], d'obtenir un fini lisse et d'arrondir les arêtes. Les pièces sont ensuite classées en fonction de leur qualité; on leur attribue un *grade* en fonction de la teneur résiduelle en eau et des défauts physiques de la pièce (qui n'étaient pas tous apparents avant le rabotage). À cette étape, une pièce peut être *éboutée* de manière à produire une pièce plus courte d'un grade supérieur. Un très long sciage peut même être subdivisé en plusieurs pièces. Ce processus est généralement automatisé et optimisé de manière à produire la plus grande valeur possible sur le marché, mais sans considérer les commandes actuelles de l'usine.

Comme au sciage, le processus génère plusieurs types de produits différents (coproduction) à partir d'un même produit en entrée (divergence). Il est important de noter que la coproduction ne peut être éliminée du point de vue de la planification; elle est liée au processus de transformation lui-même. Il est commun d'obtenir plus de 20 types de produits différents en transformant un lot de produits identiques en entrée.

La quantité de chaque produit qui est obtenue à la sortie de l'unité de finition dépend du processus de séchage utilisé au préalable. En conséquence, dans le modèle de planification qui sera introduit plus tard, nous considérerons la sortie de chaque processus de séchage (i.e. chemin dans la figure 2.4) comme un « produit » différent en entrée pour le processus de finition.

Un temps de mise en course important est nécessaire lorsqu'on passe d'une dimension à l'autre en entrée (e.g. passer du 2"x3" au 2"x6"). Pour cette raison, la plupart des entreprises choisissent de ne permettre ce changement qu'entre des quarts de travail et favorisent les campagnes dans une même dimension de plus d'un quart de travail. Les

---

[20] Après le séchage, les pièces sont déformées; elles peuvent présenter torsion, courbure et autres défauts. En enlevant une couche de bois on vise à redonner à la pièce la forme d'un prisme rectangulaire.

sciages de même dimension et de même longueur doivent être transformés ensemble (généralement, on doit procéder en ordre croissant de longueur).

En résumé, les décisions de planification de production pour la finition sont les suivantes : (1) quelles campagnes réaliser (i.e. quelles dimensions), (2) quand et pour combien de temps et (3) pour chaque campagne, quelle quantité de chaque longueur devons-nous transformer. On identifie par le fait même le panier de produit qui devrait être obtenu en sortie. La figure 2.5 présente un exemple simple de plan de production, comprenant trois campagnes (2"x3", 2"x6" et 2"x4") et montrant le temps passé sur chaque longueur.

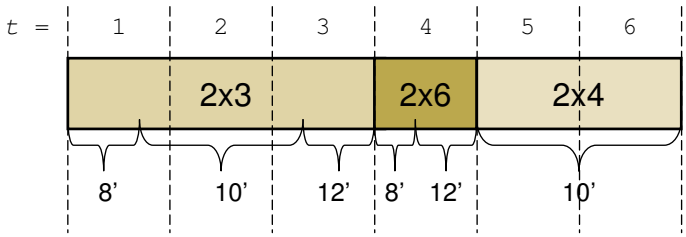


Figure 2.5 : Plan de production pour une ligne de rabotage.
Le plan concerne six quarts de travail et trois campagnes réalisées.

2.4 Prise de décision locale

Dans la suite de cette thèse, nous ferons l'hypothèse que chacune de ces unités a la capacité de planifier ses propres activités. À cet effet, les annexes 1 à 3 présentent des modèles d'optimisation pour chacun des sous-problèmes.

Chaque unité peut utiliser son algorithme pour produire un plan de production visant à livrer les produits commandés par son client[21] avec le moins de retard possible. Cependant, chacun ne peut le faire qu'avec une vision locale. Par exemple, puisque le client de l'unité de sciage est l'unité de séchage, l'unité de sciage cherche à livrer à temps ce que l'unité de séchage lui a commandé. Par ailleurs, pour chaque algorithme,

[21] Il s'agit du client externe ou encore d'une autre unité à l'intérieur du réseau.

les livraisons de matière première pour chaque unité peuvent être considérées (au choix) comme une variable de décision ou un paramètre.

Le tableau 2.1 présente les caractéristiques des problèmes de planification et des méthodes de résolution présentées en annexe.

Tableau 2.1 : Planification des opérations d'une scierie.

	Sciage	Séchage	Finition
Caractéristiques des processus	Flux produits divergents Coproduction Processus alternatifs Seuls les processus compatibles entre eux peuvent être réalisés dans le même quart de travail	Flux produits divergents Coproduction Processus alternatifs	Flux produits divergents Coproduction Processus alternatifs Deux niveaux de setups (famille de produits; longueur)
Modélisation des processus	<Types de billots + Patrons de coupe → Distribution de produits en sortie> Matrice de compatibilité des processus	Définition des activités de séchage. Les activités sont "assemblées" par l'algorithme pour construire dynamiquement les processus de séchage	<Entrée → Distribution de produits en sortie> Familles de produits Taille minimum des lots par famille / longueur
Paramètres	Calendrier disponibilité des ressources Décisions antérieures à respecter Coûts d'inventaires Coûts des matières premières	Calendrier disponibilité des ressources Décisions antérieures à respecter Coûts des opérations	Calendrier disponibilité des ressources Décisions antérieures à respecter Coûts d'inventaires Coûts des matières premières Calendrier d'expédition Coûts de setup
Méthode de résolution	Programmation mathématique (MIP)	Algorithme *à tout moment* utilisant la programmation par contraintes	Algorithme *à tout moment* utilisant la programmation par contraintes

2.5 Conclusion

Chacune des unités présentées peut être vue comme une entité autonome. La figure 2.6 présente les flux de produits entre ces trois unités, selon le formalisme introduit à la section 1.2.

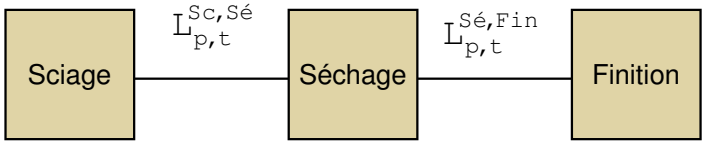


Figure 2.6 : Flux de produits entre les unités de sciage, de séchage et de finition

En résumé, nous sommes en présence d'un réseau constitué d'unités de production, chacune capable de planifier sa production en vertu d'un critère et d'un algorithme local.

Nous verrons dans les chapitres suivants comment les amener à se concerter dans le but de synchroniser leurs activités.

Chapitre 3 Approches pour la coordination d'entités autonomes

Dans ce chapitre, nous présenterons d'abord un aperçu des méthodes pouvant s'appliquer à la coordinaton d'entités autonomes en contexte hiérarchique (section 3.1). Nous présenterons deux grandes catégories de méthodes; celles-ci seront décrites avec davantage de détails dans les sections 3.2 et 3.3. Finalement, nous conclurons en résumant les principales forces et faiblesses des méthodes présentées, au regard de la problématique étudiée.

3.1 Aperçu

Tel que mentionné en introduction, la littérature générale concernant la gestion des chaînes logistiques (en anglais, *Supply chain management*, ou SCM) est vaste, mais beaucoup plus réduite en ce qui concerne la coordination des opérations manufacturières. Nous reprendrons ici les principaux éléments constitutifs du *cadre logistique* introduit au chapitre précédent (tableau 1.1) afin de présenter les principales avenues de recherche. C'est volontairement que nous ne couvrirons pas les approches centralisées ou partiellement centralisées. L'accent est mis sur les méthodes pouvant être mises à profit pour la coordination d'entités autonomes ou semi-autonomes.

Tout d'abord, au niveau opérationnel (v), la littérature propose de nombreux algorithmes pour la prise de décision spécialisés (ex : ordonnancement), souvent sans considération pour les autres aspects du problème. Ces algorithmes ont généralement fait leur chemin jusqu'aux industriels, étant intégrés aux progiciels de gestions (e.g. *Enterprise Ressource Planning*, *Advanced Planning and Scheduling*, etc.).

À l'autre extréminé (i), plusieurs chercheurs étudient les caractéristiques d'une bonne relation d'affaires, c'est-à-dire celles pouvant mener à une bonne coordination de manière naturelle [8]. La littérature dans le domaine de la gestion des inventaires

regorge d'exemples concernant l'utilisation de mécanismes tels que les pénalités [9,10], le partage des revenus [11-13], les escomptes au volume [14,15] et autres ententes contractuelles rattachées aux mécanismes de commande [8,16,17].

D'autres chercheurs étudient l'impact d'un meilleur partage d'informations (ii) sur les performances du réseau de création de valeur. Par exemple, Schneeweiss étudie dans [18] les performances du réseau en fonction du niveau d'exactitude de l'information accessible aux entités. Qui plus est, on trouve de nombreuses études concernant les mécanismes de partage d'information concernant les niveaux d'inventaires de manière à faciliter le calcul des prévisions et des stocks de sécurité.

Concernant les mécanismes de coordination en tant que tel (iii; iv) la littérature récente propose des cadres pour la coordination basés sur la négociation et le partage d'informations (e.g. [4,19]). Généralement, ces approches s'appliquent dans des contextes où les partenaires ont des objectifs conflictuels mais doivent s'entendre sur un plan. Elles sont habituellement définies pour des relations à deux partenaires ou bien la participation d'un coordonnateur est requise. Finalement, les technologies *multi-agents* sont de plus en plus utilisées pour la création de systèmes collaboratifs (voir [20] et [21]).

Le RCV en tant que système multi-agent

Un réseau de création de valeur (RCV) peut naturellement être décrit comme un système multiagent [2] : il est constitué d'entités autonomes (les agents), cherchant à atteindre des buts locaux et/ou globaux, interagissant entre elles et devant respecter les contraintes de leur environnement.

Dans une revue de littérature célèbre [22] et sa version révisée [23], Shen a répertorié plus d'une centaine de travaux utilisant les systèmes multiagents dans un contexte manufacturier. Les applications vont du design de produits jusqu'au suivi de l'exécution des opérations sur le plancher d'une usine. Plus spécifiquement, on trouve dans la

littérature de multiples méthodes spécifiquement destinées à la planification de la production. Dans une autre revue de littérature [24], Shen répertorie les principales applications des systèmes multiagents pour la planification des opérations et l'ordonnancement. Dans la plupart de ces travaux, la décomposition du système en agents est réalisée par le concepteur. Le concept de multi-agent est utilisé par le concepteur en tant que technique de décomposition et de conception de systèmes informatiques; c'est une variante de l'approche dite *diviser pour régner*. Dans le contexte qui nous intéresse, nous avons affaire à un réseau d'agents existant *a priori* et devant se coordonner.

Ce problème de coordination n'est pas uniquement l'apanage des réseaux de création de valeur. Certains auteurs ont proposé des méthodes génériques pour la résolution de problèmes d'optimisation distribués; certaines pourraient-elles s'appliquer à notre problème de coordination en contexte hiérarchique? On retrouve plusieurs de ces méthodes dans la littérature en intelligence artificielle distribuée, regroupées sous le terme *Distributed Constraint Optimization Problem* (DCOP).

En résumé, qu'elles se veuillent génériques ou qu'elles aient été développées pour un problème spécifique, il nous semble clair qu'il existe deux grandes familles d'approches : il y a d'abord (1) les approches basées sur la communication et la négociation et (2) celles basées sur l'algorithmie et le calcul. Au tableau 3.1, nous présentons les principales caractéristiques de ces deux familles. La plupart des méthodes peuvent être classées selon l'une de ces familles, bien que certaines possèdent des caractéristiques de chacune d'elles.

Nous verrons plus loin que la plupart des méthodes utilisées en pratique pour le problème de coordination de la production s'inscrivent dans le cadre de la première approche. Par contre, la seconde catégorie a l'avantage de promettre des solutions de meilleures qualités. Cependant il y a une difficulté apparente avec ces méthodes à

capturer la nature hiérarchique du problème. Les prochaines sections décrivent ces deux approches avec davantage de détails.

Tableau 3.1. Approches exploitées par les méthodes d'optimisation distribuées.

	(1) Approches basées sur la communication / négociation	(2) Approches basées sur l'algorithmie et le calcul
Source d'inspiration pour la création d'une nouvelle méthode	Inspirée des réseaux d'entreprises et/ou des réseaux d'humains, de la façon qu'ils ont de se coordonner.	Souvent basée sur un algorithme d'optimisation centralisé existant.
Conception d'une nouvelle méthode	Identification des objets échangés par les partenaires; Rôle des agents; Formalisation du protocole d'interaction.	Spécification de l'algorithme et d'un formalisme pour la description du problème; Distribution des fonctionnalités entre les agents; Identification des objets échangés.
Nature des objets échangés entre les agents	Les objets échangés ont un sens du point de vue des affaires (ex : passage d'une commande).	Les objets échangés ont un sens du point de vue algorithmique / mathématique, mais pas nécessairement du point de vue des affaires (ex : échange de coefficients lagrangiens).
Domaine d'application de la nouvelle méthode	Généralement spécifique à un domaine d'application.	Générique, pour peu que le problème puisse être modélisé à l'aide du formalisme décrit par la méthode.
Exactitude de la méthode	Généralement une heuristique.	Généralement exacte.

3.2 Méthodes basées sur la communication et la négociation

Les méthodes basées sur la communication et la négociation s'inspirent des réseaux d'entreprises et des réseaux d'humains (ci-après *agents*). Elles s'inspirent de la façon qu'ils ont de résoudre le problème dans le monde réel : informations échangées entre les agents, responsabilités des agents, règles d'interactions entre les agents, etc. Il en résulte un système où les objets échangés entre les agents ont un sens dans le monde réel : bons

de commande, acceptations/refus, etc. Un protocole d'interaction spécifie (explicitement ou implicitement) quel agent peut transmettre quelle information et/ou demande, et à quel moment il peut le faire. Certains ont proposé des langages formels permettant de spécifier des protocoles. Par exemple, Barbuceanu et Fox [25] ont proposé le langage COOL, qu'ils utilisent pour spécifier des mécanismes de coordination dans un réseau de création de valeur.

Certains protocoles sont très répandus. Le protocole *Contract Net* [26] en est un exemple. Il formalise le processus d'appel d'offres entre un agent et ses fournisseurs potentiels. Il est utilisé dans de nombreuses méthodes [23]. Les mécanismes à base d'enchères sont également beaucoup utilisés (voir, entre autre exemple [27]).

3.2.1 Heuristiques de coordination

Les méthodes les plus couramment utilisées peuvent être qualifiées *d'heuristiques de coordination hiérarchiques*. Selon ces approches, il existe une séquence selon laquelle les partenaires doivent planifier leurs opérations.

L'attribution générale des responsabilités aux agents et la décomposition du problème global en sous-problèmes sont souvent induites par le contexte d'affaires ou par des ententes à long terme en vigueur dans l'industrie (voir section 1.4).

Nous verrons ici quelques-unes de ces méthodes en utilisant pour exemple le problème de coordination présenté au chapitre chapitre 2.

(1) L'approche la plus simple consiste à planifier la production des unités de l'amont vers l'aval, les décisions d'une unité devenant des contraintes pour les successeurs. Pour le réseau décrit à la figure 2.6, la planification est réalisée de « gauche à droite » en commençant par l'unité la plus éloignée du client. Cela garantit de trouver une solution réalisable (du point de vue de la production), mais elle peut être de fort mauvaise qualité. Dans le le pire cas, l'unité la plus à gauche peut décider de ne rien fournir à son client (parce

qu'elle anticipe mal son besoin). Ce client ne fournira lui-même rien à son client, et ainsi de suite.

Bref, la qualité du résultat dépend de la capacité de chaque unité à anticiper ce que ses successeurs peuvent faire avec ce qu'elle leur livre, de même que la connaissance qu'a chaque agent des commandes du client final. En pratique, cette approche est donc utilisée pour la production sur stock (production en *flux poussé*), plutôt que dans un contexte dit *flux tiré* (où l'on cherche à satisfaire à la demande exprimée par les clients).

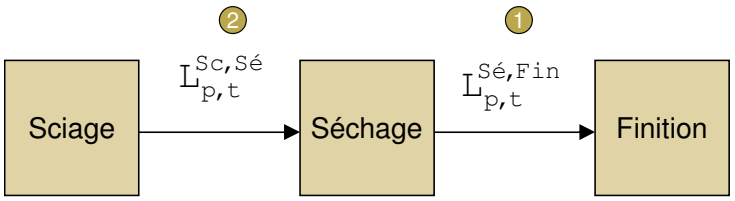


Figure 3.1 : Planification de l'amont vers l'aval, en flux poussé.

(2) Et si, contrairement au cas précédent, on planifiait « de la droite vers la gauche » de manière à mieux prendre en compte le besoin du client? Il est alors fort probable qu'un des agents demanderait à son fournisseur une livraison impossible à satisfaire. Planifier « de la droite vers la gauche » ne peut donc être fait sans prévoir un mécanisme de retour-arrière ou encore un mécanisme de résolution des conflits. Une autre option (plus généralement retenue) est de supposer qu'une entente à long terme spécifie ce qui constitue une commande « acceptable » que le fournisseur est réputé pouvoir satisfaire.

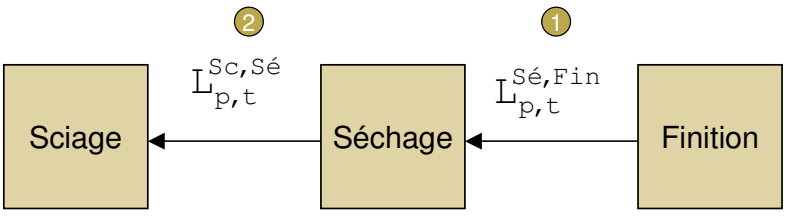


Figure 3.2 : Planification de l'aval vers l'amont, en flux tiré.

(3) Le problème a également été adressé en considérant deux niveaux de décision pour chaque unité [21]. Chaque unité est responsable de calculer deux choses : (1) une demande pour son fournisseur, calculée en supposant que le fournisseur dispose d'une capacité de production infinie et (2) le vrai plan de production, élaboré lorsque la « réponse » du fournisseur est connue. La séquence selon laquelle les agents interviennent est représentée à la figure 3.3.

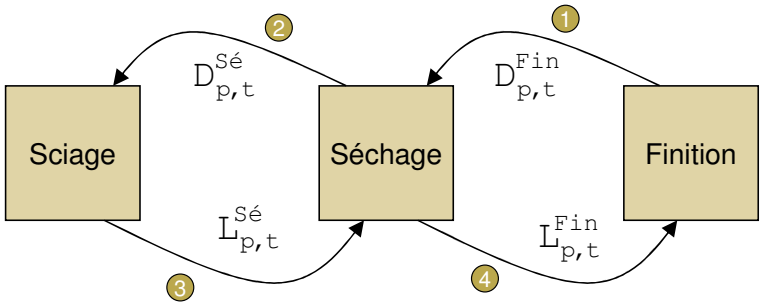


Figure 3.3 : Planification en deux phases.

3.2.2 Le modèle de Schneeweiss pour la formulation des problèmes de décision distribués

Schneeweiss a présenté un cadre général pour décrire ce genre de hiérarchies de problèmes [1]. À titre d'exemple, considérons un collectif d'agents appliquant la méthode de planification en deux phases illustrée à la figure 3.3. Nous avons alors une hiérarchie de cinq sous-problèmes de décision (un sous-problème pour chaque type d'information reçue ou transmise en vertu du flot exprimé à la figure 3.3).

Selon ce modèle, la décision prise à un niveau donné (*top*) est interprétée par le niveau suivant (*bottom*) comme un paramètre du problème à résoudre au niveau *bottom*. Autrement dit, le sous-problème de b*ottom* est défini en fonction de la décision de *top*.

Pour obtenir une solution au problème global, les sous-problèmes seront résolus de manière séquentielle du haut vers le bas (une approche appelée an anglais *upstream*

planning [19,28]). Cette approche implique qu'on dispose pour chaque niveau d'un modèle/algorithme permettant de résoudre le sous-problème qui lui est associé.

Cependant, en agissant ainsi, le collectif d'agent ne considère qu'une et une seule solution globale qui risque fort de ne pas être optimale. C'est la raison pour laquelle nous qualifions cette approche (et les précédentes) d'*heuristiques de coordination hiérarchiques*. Bien sûr, donner à l'agent *top* la capacité d'anticiper le comportement de *bottom* permet une amélioration des résultats (voir [18]). Cependant, des mécanismes plus généraux peuvent être envisagés.

3.3 Méthodes basées sur l'algorithmie et le calcul

Tout à l'opposé des *heuristiques de coordination*, on trouve des méthodes visant l'obtention d'une solution optimale, malgré le fait que le problème et la recherche d'une solution soient distribués. La plupart des méthodes de ce type sont inspirées d'un algorithme centralisé qu'on cherche à distribuer. On vise généralement à permettre la résolution d'un large éventail de problèmes, pour peu qu'ils puissent être modélisés à l'aide du formalisme proposé par la méthode.

Dans ces méthodes, les objets échangés entre les agents ont parfois peu de sens dans le monde réel. Par exemple, certains ont proposé des méthodes où la coordination est réalisée par l'échange de coefficients lagrangiens [29,30].

Plusieurs méthodes sont issues de ce qu'on appelle le traitement distribué des contraintes (*Distributed Constraint Reasoning*, DCR). Certaines méthodes sont développées pour la résolution de problèmes de satisfaction de contraintes (*Distributed Constraint Satisfaction Problem*, DisCSP), d'autres pour des problèmes d'optimisation (*Distributed Constraint Optimization Problem*, DCOP) [31,32].

3.3.1 Distributed Constraint Satisfaction Problem (DisCSP)

Le formalisme DisCSP a été introduits par Yokoo dans [33]. Les DisCSP ont été introduits comme une manière de formaliser certains problèmes en intelligence artificielle distribuée. Auparavant, la communauté utilisait les termes *Cooperative Distributed Problem Solving*, et *Distributed Problem Solving* [34], mais ceux-ci [35,36] n'étaient pas définis formellement[22].

Les DisCSP sont décrits en détail dans le livre de Yokoo [37]. Le formalisme proposé par Yokoo est une extension au formalisme largement utilisé pour représenter un problème de satisfaction de contraintes non distribué (*Constraint Satisfaction Problem*, CSP). On définit ci-après formellement les CSP et DisCSP.

Définition 3.1 : Constraint Satisfaction Problem (CSP). Un CSP est défini de la manière suivante. Soit un ensemble de variables $X = \{X_1, \ldots, X_n\}$. Chaque variable X_i peut prendre une valeur parmi un ensemble D_i. Cet ensemble constitue le domaine de la variable. Formellement, on a $D = \{D_1, \ldots, D_n\}$. On a également un ensemble de contraintes $C = \{C_1, \ldots, C_m\}$. Chaque contrainte C_i s'exprime par un prédicat $p_i(y_1, \ldots, y_j)$ défini pour un sous-ensemble de variables $\{y_1, \ldots, y_j\} \subseteq X$. On cherche à donner une valeur à chaque variable de manière à ce que chaque contrainte soit satisfaite.

Définition 3.2 : Distributed Constraint Satisfaction Problem (DisCSP). Un DisCSP [37] est un CSP devant être résolu par une équipe d'agents $A = \{A_1, \ldots, A_k\}$. Chaque variable appartient à un agent $\mathcal{A}(X_i) \in A$. Un agent peut affecter une valeur uniquement aux variables qu'il possède.

---

[22] Ces termes embrassent toutefois une gamme de problèmes plus large que les DisCSP et les DCOP. Ils ne sont donc pas révolus.

Mentionnons également qu'un agent peut connaître la valeur affectée à une variable d'un autre agent s'il est lié à cette variable par une contrainte. C'est l'algorithme de résolution utilisé qui spécifie quand et comment cette information est transmise.

La figure 3.4 présente un graphe de contraintes pour un DisCSP fictif. On y trouve trois agents (A, B et C). Les agents A et B représentent des usines et l'agent C est une compagnie de transport. Les variables sont représentées en majuscules et les paramètres en minuscules. La zone ombragée montre à quel agent appartient chaque variable. Les arcs représentent les contraintes.

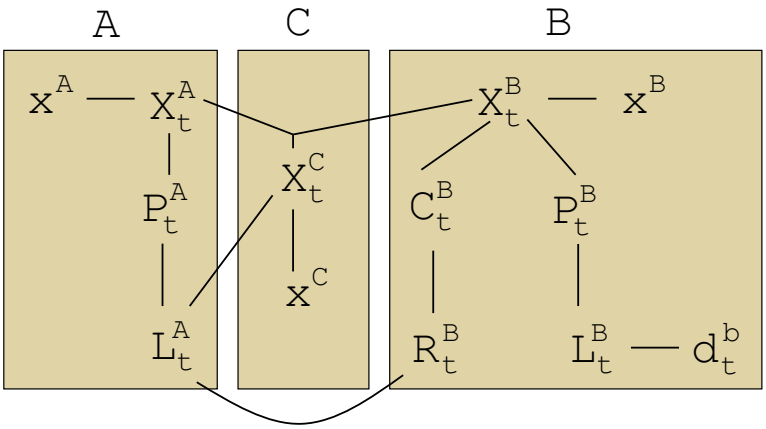


Figure 3.4 : Exemple de DisCSP.

Les variables X_t^A et X_t^B représentent la décision de production que prendra un agent pour la période t; elle peut être interprétée comme le temps pendant lequel l'agent produira lors de cette période. La variable X_t^C est la quantité de produit transportée par l'agent C à la période t. Les paramètres x imposent une limite à la capacité de production. Les variables P_t représentent les quantités de produits fabriqués, L_t les quantités livrées, R_t les quantités reçues et C_t les quantités consommées. Finalement, le paramètre d_t représente la demande du client externe.

Certaines contraintes sont internes aux agents alors que d'autres sont des contraintes interagents. On trouve une contrainte ternaire reliant les variables X_t^A, X_t^B et X_t^C. Elle

oblige à respecter une borne supérieure au niveau du coût de la solution globale. Le tableau 3.2 décrit formellement ces contraintes.

Tableau 3.2. Contraintes du problème représenté à la figure 3.4.

Agent A	Agent B	Agent C
$X_t^A \leq x^A$; $P_t^A = 3X_t^A$; $L_t^A \leq P_t^A$.	$X_t^B \leq x^B$; $C_t^B = 4X_t^B$; $P_t^B = 2X_t^B$; $\sum_t C_t^B \leq R_t^B$; $L_t^B \leq P_t^B$; $L_t^B = d_t$.	$X_t^C \leq x^C$.
Contraintes interagents		
$\sum_t 2X_t^A + 3X_t^B + 4X_t^C \leq 72$; $L_t^A = R_t^B$; $L_t^A = X_t^C$		

Résolution d'un DisCSP

Yokoo - et plusieurs autres ensuite - ont proposé différents algorithmes pour résoudre les DisCSP. Les mesures de performance utilisées pour comparer les algorithmes sont généralement: (1) le temps nécessaire pour résoudre le problème et (2) le nombre de messages échangés [38]. Dans ce qui suit, on présente les principales familles d'algorithmes.

Algorithmes dits *synchrones*

L'algorithme de base qui sert souvent de référence est le *Synchronous Bactracking* (SyncBT) [33]. Pour la description de cet algorithme, nous ferons l'hypothèse que nous avons une seule variable par agent.

Dans SyncBT, les agents/variables sont ordonnés *a priori*. Les agents prennent leur décision locale à tour de rôle. En cas d'impasse, on réalise un retour-arrière chronologique; lorsqu'un agent détecte la violation d'une contrainte, il envoie un message à son prédécesseur lui demandant de changer la valeur de sa variable. Ce faisant, le collectif d'agents réalise une recherche en profondeur (*Depth-first Search*, DFS) dans un arbre représentant l'espace des solutions. L'arbre est construit dynamiquement par le collectif. C'est ce qu'on appelle, en environnement centralisé, le retour-arrière chronologique, ou tout simplement *backtracking* en anglais [39].

Avec SyncBT, il y a un seul agent à la fois qui travaille. Le terme *synchrone* fait référence au fait qu'un agent ne peut pas choisir/modifier la valeur retenue pour sa variable de manière asynchrone (i.e. à tout moment); il doit attendre son tour.

La figure 3.5 présente un exemple de résolution avec SyncBT pour un problème de coordination entre trois agents (`X`, `Y` et `Z`). L'agent `X` résout d'abord son problème local. Il transmet sa décision à l'agent `Y` (message #1, sur la figure). L'agent `Y` cherche ensuite à solutionner son problème, mais il réalise que ça lui est impossible compte tenu de la décision de `X`. L'agent `Y` en informe `X` (message #2), qui produit une décision alternative et en informe `Y` (message #3). L'agent `Y` prend alors une décision locale pour ce « nouveau » problème, il en informe `Z` (message #4) qui rencontre une impossibilité (message #5). L'agent `Y` revoit alors sa décision, la transmet à `Z` (message #6). Finalement, l'agent `Z` arrive à trouver une solution réalisable pour son plan (message #7), laquelle tient compte des décisions des prédécesseurs.

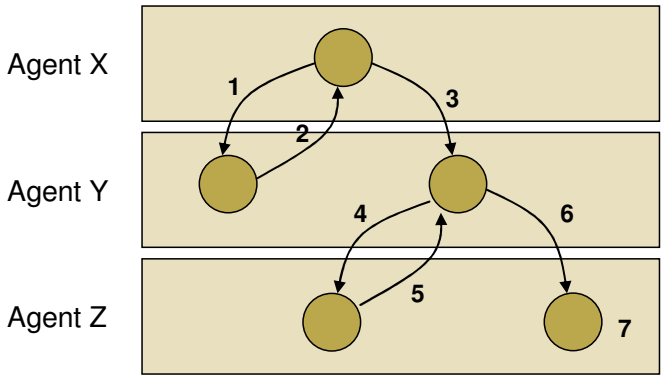


Figure 3.5 : Résolution d'un DisCSP avec SyncBT.

Algorithmes dits *asynchrones*

L'algorithme *Asynchronous Bactracking* (ABT) [33] permet aux agents de travailler de manière simultanée. La façon d'y arriver est de permettre à chaque agent de changer à tout moment la valeur de sa variable (d'où son asynchronisme). Lorsqu'un agent change la valeur de sa variable, il avise les agents de priorité inférieure (comme pour SyncBT, on suppose que les agents sont ordonnés *a priori*). Lorsqu'un agent reçoit la valeur d'un agent de priorité supérieure, il choisit pour sa variable une valeur compatible en vertu des contraintes. Si ce n'est pas possible, un message est envoyé aux agents impliqués dans ce « blocage ». Ces agents définiront une nouvelle contrainte (*nogood*) qui les force à ce qu'au moins l'un d'entre eux change de valeur.

L'exemple suivant (figure 3.6) est tiré de [37]. Dans cet exemple, trois agents `X`, `Y` et `Z` doivent s'entendre. Encore une fois, chaque agent a une seule variable. Les variables `X` et `Z` ont pour valeur possible 1 et 2. La variable `Y` peut seulement prendre 2 comme valeur. On a pour contrainte que `X` doit être différent de `Z`. Également, `Y` doit être différent de `Z`. L'ordre de priorité des agents est le suivant : `X`, `Y` et `Z`. Au départ, chaque agent choisit une valeur et envoie celle-ci aux agents de niveau inférieur avec lequel il est lié par une contrainte (message #1, sur la figure). L'agent `Z` détecte ensuite qu'il lui est impossible de choisir une valeur compte tenu de celles prises par `X` et `Y`. Il publie alors une nouvelle contrainte (*nogood*) spécifiant que `X` ne peut prendre 1 pour valeur en

même temps que Y prend 2 pour valeur. L'agent Z envoie ce *nogood* à l'agent impliqué dans ce *nogood* qui a la plus basse priorité, c'est-à-dire Y (message #2). Sur la réception de ce message, Y cherche à assigner à sa variable une valeur consistante avec le *nogood* et les autres contraintes. Puisqu'il n'y en a pas, il transmet un *nogood* à l'agent X (message #3). L'agent X change alors sa valeur et la transmet à Z (message #4). L'agent Z choisit alors une valeur satisfaisant toutes les contraintes.

ABT a évolué dans le temps: mentionnons les versions de Yokoo en 1998 [40] et 2000 [41]. Les différentes versions de ABT sont répertoriées dans [42]. En 2005, a été proposé l'algorithme ABT_not [43]. L'avantage de ce dernier est d'éviter que des agents non initialement connectés par une contrainte aient à se parler directement pour gérer le *nogood*. L'algorithme *Distributed Dynamic Backtracking* (DisDB) [44] est également considéré comme un dérivé de ABT [38].

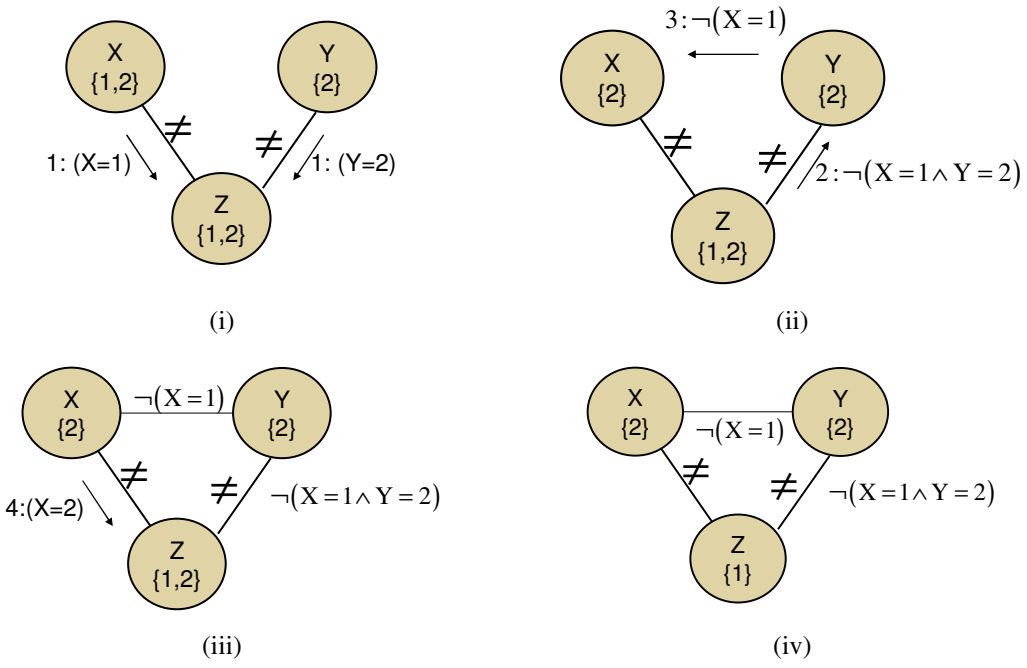


Figure 3.6 : Résolution d'un DisCSP avec ABT.
Adapté de [37].

Les performances de SyncBT et ABT ont été comparées par Yokoo [33,37]. D'autres travaux [45] ont montré que la bonne performance de ABT n'était pas uniquement due à son asynchronisme, mais aussi en grande partie à la stratégie de retour arrière utilisée par ABT. Zivan a proposé [45] une version améliorée de SyncBT utilisant une stratégie de retour arrière différente et un nouvel algorithme synchrone nommé *Synchronous Conflict Based Backjumping*. Dans ces circonstances, il a montré que les performances des algorithmes synchrone et asynchrone étaient comparables (en termes de temps de calcul) mais que moins de messages étaient échangés pour l'algorithme synchrone.

Concurrent Search Algorithms (CSA)

Zivan a introduit une façon différente de réaliser la concurrence. Il a proposé le terme *Concurrent Search Algorithm* [46-49]. Comme pour SyncBT, les agents travaillent séquentiellement (et de manière synchrone) à la construction d'une solution. Par contre, le collectif travaille simultanément sur plusieurs solutions. L'algorithme spécifie quand et comment les nouveaux fils (*threads*) sont créés. Un nouveau fil de recherche est créé en divisant en deux le domaine d'une variable.

Zivan a proposé les algorithmes ConcBT [47,49] et ConcDB [48,49]. Ceux-ci se distinguent entre eux au niveau de la stratégie de retour arrière employée par chacun. ConcBT réalise un retour arrière chronologique alors que ConcDB applique une variante de *Dynamic Backtracking* [49,50].

3.3.2 Distributed Constraint Optimization Problem (DCOP)

Différents efforts ont été accomplis afin d'étendre la notion de DisCSP aux problèmes d'optimisation. On ne trouve pas dans la littérature de définition faisant l'unanimité. Qui plus est, les définitions et les algorithmes pour les DCOP ne sont pas aussi généraux que ceux admis pour un COP centralisé. Nous présentons ici la définition donnée par Modi [51] qui est la plus généralement admise.

Définition 3.3 : Distributed Constraint Optimization Problem (DCOP). Un DCOP est défini par un ensemble de variables $X = \{X_1, \ldots, X_n\}$. Chaque variable X_i est sous la responsabilité d'un et un seul agent $A_i \in A = \{A_1, \ldots, A_n\}$ qui contrôle la valeur de la variable. Chaque variable X_i doit prendre une valeur parmi un domaine D_i. Formellement, on a $D = \{D_1, \ldots, D_n\}$. On a également un ensemble de contraintes binaires $C = \{C_1, \ldots, C_m\}$. Chaque contrainte C_j s'exprime par une fonction $f_j(y_\alpha, y_\beta) : D_\alpha \times D_\beta \rightarrow \mathbb{R}^+$ définie pour un couple de variables $\{y_\alpha, y_\beta\} \subseteq X$ avec $\alpha \neq \beta$. On cherche à donner une valeur à chaque variable de manière à minimiser la fonction $F(\,) := \sum_{j=1,\ldots m} f_j$.

Résolution d'un DCOP : Synchronous Branch and Bound (SyncBB)

La façon la plus simple de résoudre un DCOP est d'appliquer ce que Hirayama et Yokoo appellent *Synchronous Branch and Bound* (SyncBB) [51,52]. Il s'agit d'une adaptation de SyncBT. L'algorithme est modifié de manière à ce que la recherche se poursuive une fois qu'une première solution est obtenue. La meilleure solution rencontrée lors de l'exploration complète de l'arbre sera celle qui sera finalement retenue. Comme pour SyncBT, lorsqu'une impasse est détectée ou qu'une feuille (solution globale) est atteinte, l'agent qui a détecté cette condition envoie à l'agent précédent le message lui demandant une proposition alternative. Dans SyncBB, ce message contiendra la valeur de la meilleure solution globale trouvée jusqu'à maintenant. Elle peut être utilisée pour réaliser des coupes dans l'arbre lors du reste de la recherche. La méthode est *complète*, puisqu'elle retourne la solution optimale si on dispose d'assez de temps.

D'autres algorithmes ont été proposés. Dans ce qui suit, nous décrirons les principaux.

Résoudre une séquence de DisCSP

Dans son document de travail de 1991, Yokoo [53] propose trois algorithmes asynchrones basés sur les DisCSP: (1) *Distributed Depth-first Branch & Bound* (DDBB), (2) *Distributed Iterative Deepening* (DID) et (3) une combinaison des deux premiers.

Ces algorithmes proposent de résoudre séquentiellement plusieurs DisCSP. On spécifie d'abord une valeur pour la borne supérieure de la fonction objectif (dans le cas de DDBB) ou une borne inférieure (dans le cas de DID). On cherche ensuite à résoudre le DisCSP correspondant. S'il n'y a pas de solution, la valeur de la borne est modifiée et on résout le nouveau DisCSP. Le document de travail de Yokoo n'a jamais donné lieu à une publication.

Problèmes surcontraints

D'autres travaux ont été réalisés pour d'autres formulations moins générales des DCOP. Par exemple, dans [52] on cherche à minimiser le nombre de contraintes violées dans un DisCSP surcontraint. On y propose *Iterative Distributed Breakout* (IDB). IDB réalise l'équivalent d'une « recherche locale centralisée», mais de manière distribuée. Il est basé sur l'algorithme *Distributed Breakout* (DB) [54] développé par les mêmes auteurs pour résoudre les DisCSP.

ADOPT : une méthode asynchrone

Lors de son introduction, l'algorithme appelé *Asynchronous Distributed Optimisation* (ADOPT) [32,51,55,56] était le seul algorithme à la fois asynchrone et complet pour les DCOP [56]. Il permet d'obtenir une garantie sur la qualité d'une solution trouvée, lorsque l'exécution de l'algorithme est interrompue avant la fin. La garantie prend la forme d'un écart relatif maximal entre la qualité de la solution et l'optimum. Cela est

rendu possible grâce au fait que les agents calculent en tout temps une borne optimiste sur la qualité de la solution[23].

Dans la description qui suit, on suppose qu'il n'y a qu'une seule variable par agent et que le collectif cherche à minimiser une fonction objectif. Les variables/agents doivent d'abord être ordonnés selon leur priorité. L'ordre n'a pas à être absolu : deux agents peuvent avoir la même priorité. ADOPT oblige seulement à ce que deux agents liés par une contrainte aient des niveaux de priorité différents. De plus, un des agents doit être prioritaire sur tous les autres. Dans ce contexte, les agents sont organisés à la manière d'un arbre.

L'idée principale sous-tendant ADOPT est de permettre à l'agent de changer la valeur de sa variable aussitôt qu'il existe une possibilité que la nouvelle valeur mène à une meilleure solution que l'ancienne valeur. Tous les agents réalisent ce qui suit de manière asynchrone. Pour chaque valeur possible du domaine de sa variable, l'agent stocke une borne inférieure sur la fonction objectif. L'agent sélectionne pour sa variable la valeur ayant la plus petite borne inférieure et transmet la valeur à tous ses descendants (selon l'arbre de priorités) avec lesquels il est lié par une contrainte. Les destinataires choisiront pour leur variable la valeur ayant la plus petite borne inférieure, compte tenu du choix de l'expéditeur. La nouvelle borne inférieure est communiquée au père afin qu'il puisse mettre à jour son tableau de bornes inférieures. Lorsque ce tableau est mis à jour, il se peut que l'agent réalise que son choix antérieur ne correspond plus à la plus petite des bornes inférieures. L'agent change alors la valeur de sa variable.

La figure 3.7 présente un exemple de problème adapté de [51]. Chaque agent peut choisir une couleur (foncé ou pâle). Un arc liant deux agents représente une contrainte (sous-figure i). La valeur de chaque contrainte (i.e. le coût à assumer en fonction de la couleur choisie par deux agents liés) est présentée dans la sous-figure (ii). On cherche à minimiser la somme de ces coûts.

---

[23] Dans un contexte de minimisation d'une fonction objectif, il s'agira d'une borne inférieure.

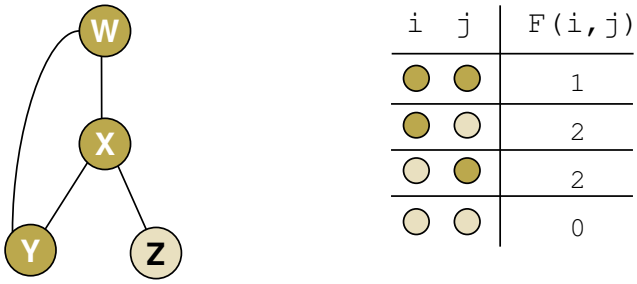


Figure 3.7 : Exemple de DCOP sous ADOPT.
Adapté de [51].

La figure 3.8 présente un exemple de résolution. Dans cet exemple, on suppose d'abord que chaque agent choisit la couleur « foncé ».

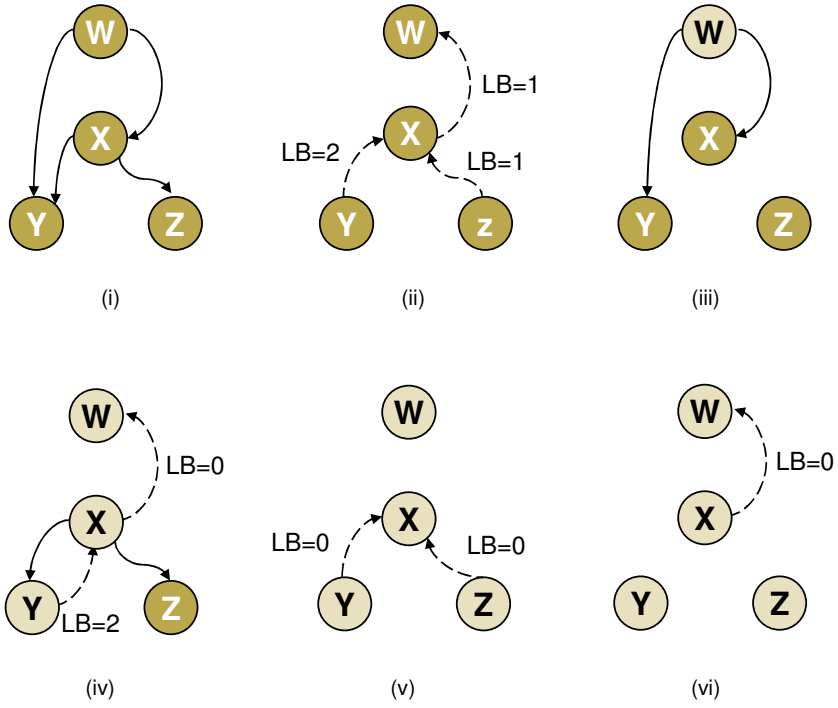


Figure 3.8 : Résolution d'un DCOP avec ADOPT.
Adapté de [51].

À l'étape (i), chaque agent envoie sa valeur à tous ses descendants avec lesquels il est lié par une contrainte en vertu du graphe de la figure 3.7. Ensuite, ces agents calculent une borne inférieure sur la base des valeurs de leurs ancêtres et envoient cette valeur à leur père (ii). Pour la calculer, l'agent établit la meilleure valeur qu'il peut prendre compte tenu de la valeur des ancêtres avec lesquels il est lié, de manière à réduire le coût total de toutes les contraintes dans lesquelles il est impliqué.

De cette manière, l'agent W apprend qu'avec sa couleur « foncé », le système global ne pourra faire mieux qu'avoir un coût de 1 (c'est une borne optimiste, car nous ne sommes pas certains de pouvoir le faire). L'agent W change donc sa couleur pour « pâle » étant donné qu'il lui semble possible d'obtenir une meilleure solution de cette façon (sa borne optimiste pour « pâle » lui laisse penser que c'est possible). Il en informe ses descendants (iii). À l'étape (iv), les agents X et Y réalisent que si W est pâle, il leur est préférable d'être pâles également. Ils changent donc leurs couleurs, en informent leurs descendants, calculent une nouvelle borne optimiste et la communiquent à leur père.

Lorsque X recevra le message de Y, l'agent X aura déjà viré au pâle, alors il ignorera ce message périmé. À l'étape (v), Y et Z ont reçu la nouvelle valeur de X (pâle). L'agent Y décide de rester inchangé et envoie une nouvelle borne inférieure à X. L'agent Z décide de virer au pâle et envoie une nouvelle borne inférieure à Y. Finalement, en (vi) X envoie une borne à W. L'optimalité de la solution est ainsi établie.

En résumé, les agents réalisent collectivement un *Iterative Deepening Search* (IDS) [39,57] mais de manière distribuée et asynchrone. Dans [51] et [56], les auteurs étudient quelle partie de la bonne performance de ADOPT est due à l'asynchronisme des agents et quelle partie est due à IDS. Pour ce faire, les auteurs proposent une version distribuée de IDS qu'ils nomment SynchID : « SynchID simulates iterative deepening search in a distributed environment by requiring agents to execute sequentially and synchronously (…) The central difference is that SynchID is sequential while ADOPT is concurrent» [56]. ADOPT a également été comparé avec SyncBB.

Sous ADOPT, il n'y a pas de différence entre une contrainte « dure » et une contrainte « molle ». Chaque contrainte est représentée par une fonction de deux variables qui retourne un nombre réel. ADOPT cherche à minimiser la somme de ces fonctions. Une contrainte « dure » est modélisée par une fonction retournant une valeur infinie lorsqu'elle n'est pas satisfaite.

Pour pouvoir utiliser ADOPT, un DCOP doit respecter ceci : (1) les contraintes sont binaires et (2) la fonction objectif s'exprime comme la somme des « valeurs » des contraintes. De plus, (3) La fonction objectif doit être monotone[24]. La façon la plus simple d'y arriver est de supposer que chaque contrainte ne peut prendre qu'une valeur positive.

Pour ces raisons, il existe des problèmes distribués qui ne peuvent pas être modélisés sous ADOPT. Par exemple dans [58], Yokoo lui-même cite le cas d'un collectif d'agents cherchant à optimiser une fonction d'utilité, sous la contrainte que la consommation totale d'une ressource (ex : de l'essence ou une enveloppe budgétaire) ne dépasse pas une certaine quantité.

Méthodes basées sur la programmation dynamique

Récemment, différentes méthodes inspirées de la programmation dynamique ont été proposées. Ces méthodes visent à réduire le nombre de messages échangés comparativement aux méthodes exploitant le retour-arrière [59].

Petcu et Faltings décrivent dans [59] un algorithme naïf basé sur la programmation dynamique: à tour de rôle, chaque agent génère la liste de toutes les solutions partielles possibles compte tenu des décisions des prédécesseurs. Le dernier agent conserve la meilleure solution. Ainsi le nombre de messages échangés est linéaire par rapport au nombre d'agents, mais les messages sont énormes et la concurrence inexistante.

---

[24] Une fonction $F(x)$ est dite monotone *croissante* si et seulement si $\forall x > 0$ nous avons $F(x) \geq F(x-1)$. Elle est dite monotone *décroissante* si $\forall x > 0$ nous avons $F(x) \leq F(x-1)$.

Toujours dans [59], les auteurs rappellent qu'il existe un algorithme (*sum-product algorithm*) proposé dans [60], permettant de résoudre les problèmes de ce type en utilisant des messages de taille fixe. Cependant, cet algorithme exige que le graphe des contraintes prenne la forme d'un arbre. Dans [61], Petcu et Faltings expliquent comment passer outre à cette limitation. Sur la base de ces idées, Petcu et Faltings ont proposé l'algorithme *Dynamic Programming Optimization* (DPOP) [59,61]. L'algorithme est comparé avec ADOPT, mais uniquement en termes de nombre de message échangés.

L'objectif de ces méthodes basées sur la programmation dynamique est de permettre une réduction du nombre de message échangés. De plus, l'efficacité de ces méthodes repose sur une certaine aptitude des agents à calculer une borne sur la fonction d'utilité. Leur application à un problème purement combinatoire serait problématique.

3.4 Conclusion

Les méthodes basées sur les communications et la négociation offrent au concepteur l'avantage de pouvoir s'inspirer du monde réel. La grande majorité des méthodes proposées pour résoudre le problème de la coordination des agents dans un réseau de création de valeur s'inscrivent dans le cadre de cette approche. Les méthodes issues de cette approche peuvent généralement être décrites comme des heuristiques de coordination : on fait le pari (et parfois on montre empiriquement) que des agents exploitant la méthode généreront des solutions de qualité acceptable. Elles permettent généralement d'obtenir une seule solution; elles ne permettent pas de profiter du temps de calcul encore disponible pour générer des solutions alternatives potentiellement meilleures. Ces méthodes ne peuvent souvent s'appliquer qu'à un type de problème en particulier.

À l'opposé, les méthodes basées sur l'algorithmie et le calcul se veulent très génériques. Paradoxalement, ces algorithmes exigent que le problème soit modélisé selon certaines formulations particulières et contraignantes. Elles supposent notamment l'absence de hiérarchie entre les agents et imposent une forme particulière pour la fonction objectif. Il y a donc des difficultés apparentes pour qui désire utiliser ces méthodes dans des contextes d'affaires réels.

Chapitre 4 Hierarchical Distributed Constraint Optimization Problem (HDCOP) : un cadre pour la prise de décision distribuée en contexte hiérarchique

Nous avons vu au chapitre précédent que les méthodes basées sur l'algorithmie et le calcul se veulent très génériques. Paradoxalement, les méthodes existantes exigent une modélisation du problème selon certaines formulations particulières et contraignantes.

Dans ce chapitre, nous verrons d'abord comment le problème de prise de décision distribuée en contexte hiérarchique peut en apparence être modélisé en tant que DCOP (section 4.1). Cependant, des difficultés majeures se dressent devant l'utilisation de ces approches dans des contextes d'affaires réels comme celui qui est étudié. Notamment, (1) les approches DCOP ignorent la question de la hiérarchie entre les agents (le sous-problème de chacun doit être défini *a priori*), (2) elles imposent une forme particulière pour la fonction objectif, et, finalement, (3) elles imposent également la méthode et le critère devant être utilisés pour la prise de décision locale de chaque agent. Nous étudierons ces difficultés à la section 4.2.

Par la suite, à la section 4.3, nous introduirons un nouveau formalisme appelé HDCOP. Il permet de modéliser le problème de prise de décision distribuée en contexte hiérarchique. L'espace de coordination est représenté par un arbre; cela rend possible l'utilisation de certains algorithmes classiques de recherche dans un arbre en tant que mécanismes de coordination (section 4.4). On permet alors aux agents d'explorer systématiquement l'espace de coordination, tout en conservant les mêmes relations d'affaires, responsabilités et algorithmes locaux de décision.

Finalement, nous discuterons de certaines caractéristiques du cadre et nous présenterons différents avenues d'utilisation (section 4.5).

4.1 Modéliser le problème de coordination en contexte hiérarchique sous la forme d'un DCOP

Nous avons introduit aux chapitres Chapitre 1 et Chapitre 2 le problème de coordination entre entités autonomes en contexte hiérarchique. Pour le cas précis de la coordination dans un réseau de création de valeur, nous avons utilisé une modélisation conceptuelle selon laquelle chaque paire d'agents devait s'entendre sur le flux de produit liant ces deux agents (voir figure 1.2). Selon cette conception des choses, une variable partagée entre les deux agents représente le flux (voir à cet effet la figure 2.6).

Une formulation alternative est de considérer que les agents ont chacun une « copie » de la variable et qu'une contrainte d'égalité lie ces deux variables. La figure 4.1 représente le cas du bois d'œuvre (originellement présenté à la figure 2.6), modélisé de cette façon. Le « problème local » de chaque agent comporte plusieurs variables : celles correspondant au flux de produits échangés avec les autres agents, de même que celles correspondant au plan de production. Bien sûr, il s'agit d'un exemple simplifié, puisque le problème de chaque agent comporte en réalité des milliers de variables.

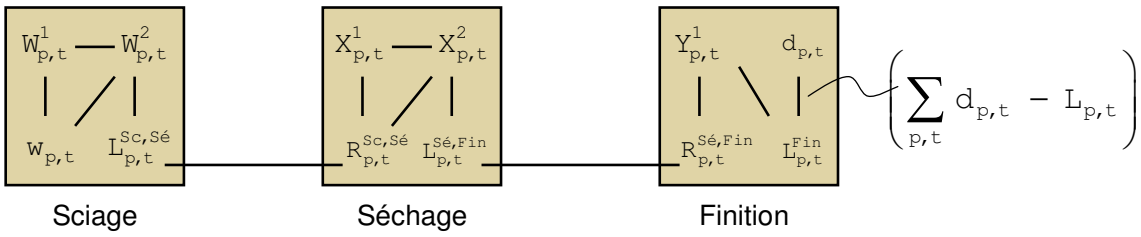


Figure 4.1 : Modélisation d'un DCOP avec plusieurs variables par agent.

Chaque contrainte interagent est une contrainte d'égalité. Rappelons que, selon le formalisme DCOP, les contraintes dures sont représentées par des fonctions prenant une valeur nulle lorsque la contrainte est respectée, et une valeur infinie sinon.

Les contraintes internes de chaque agent sont également des contraintes dures qui prendront une valeur selon cette méthode. La seule exception est la contrainte interne

liant les livraisons de l'agent Finition $\left(L_{p,t}^{Fin}\right)$ et la demande (représentée par le paramètre $d_{p,t}$); c'est que l'objectif commun est de minimiser l'écart entre la quantité demandée et la quantité livrée, alors on doit le formuler sous la forme d'une contrainte.

4.1.1 Situations avec plusieurs variables par agent

Dans la littérature DCOP, la pratique généralisée consiste à ne considérer que les problèmes avec une seule variable par agent. Notamment, Modi ne présente aucune étude du comportement d'ADOPT pour les cas avec plusieurs variables par agent, que ce soit dans sa thèse [51] ou dans ses articles [32,55,56]. Par contre, il mentionne que ces cas pourraient être considérés de la manière suivante : (1) on associe un agent virtuel à chaque variable et (2) on permet à chaque agent « physique » de jouer le rôle des agents virtuels qu'il représente. Par contre, cette avenue ne peut être utilisée dans notre cas puisque cela supposerait que chaque agent « physique » est capable de résoudre son problème une variable à la fois, ce qui n'a pas de sens pour un agent utilisant un modèle d'optimisation pour prendre sa décision (et donc donnant une valeur à plusieurs variables simultanément).

Une autre technique proposée par Modi consiste à remplacer les multiples variables d'un agent par une seule « grosse » variable. Modi décrit très bien la technique : il suffit de « remplacer toutes les variables d'un agent par une nouvelle variable dont le domaine est le produit croisé des domaines de chaque variable originale » [51].

Le cas du bois d'œuvre présenté à la section Chapitre 2 (et à la figure 4.1) peut être reformulé de cette façon. On présente le résultat à la figure 4.2. Dans cette figure, W, X et Y représentent les variables de nos trois agents et C les contraintes. Nous ajouterons une quatrième variable « bidon » (Z). Il s'agit plutôt d'une constante servant à représenter la demande du client externe.

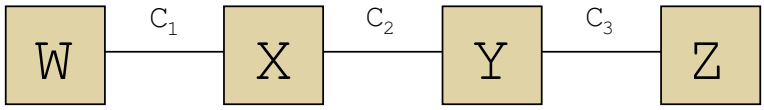


Figure 4.2 : Reformulation d'un DCOP avec une seule variable par agent.

Bien sûr, les contraintes doivent être reformulées elles aussi. Pour être conforme à ADOPT, la contrainte C_1 prend une valeur infinie lorsque W et X sont des plans incompatibles. La contrainte C_2 prend une valeur infinie si les plans X et Y sont incompatibles. La contrainte C_3 est égale à la pénalité que le réseau doit assumer pour ne pas livrer à temps les produits aux clients, ou bien correspond à une mesure quelconque de l'insatisfaction qu'engendrerait le plan pour le client externe. Dans notre exemple, la valeur de la contrainte ne dépend que de la variable Y, puisque Z est constant).

4.1.2 Respect du caractère privé des informations

Dans les algorithmes pour DCOP comme ADOPT, un agent transmet à ses voisins la valeur des variables pour lesquelles il est relié via une contrainte. Cependant, aucune information n'est transférée à propos du domaine des variables. Lorsque le problème est modélisé avec une seule variable par agent, en transmettant la valeur de la variable on transmet inévitablement de l'information de nature privée. Nous considérons toutefois qu'une solution simple à ce problème consisterait à cacher la valeur des bits qui ne sont pas nécessaires au calcul de la valeur de la contrainte, avant de transmettre la valeur de la variable.

4.2 Obstacles à l'utilisation du cadre DCOP dans le contexte étudié

Des difficultés majeures se dressent devant l'utilisation de ces approches dans des contextes d'affaires réels comme celui que nous étudions. Notamment, ces approches ignorent la question de la hiérarchie entre les agents (le sous-problème de chacun doit être défini *a priori*), elles imposent une forme particulière pour la fonction objectif, et

finalement, elles imposent également la méthode et le critère qui doivent être utilisés pour la prise de décision locale de chaque agent.

Les sections suivantes (4.2.1 à 4.2.4) développent quatre arguments contre l'utilisation du cadre DCOP dans le contexte étudié. Elles sont suivies d'une discussion sur la portée de ces arguments (4.2.5).

4.2.1 Autonomie limitée des agents

Dans un cadre DCOP, le concepteur a le loisir de spécifier un algorithme de son choix qui sera exécuté par chacun des agents. Cet algorithme spécifie la méthode et le critère devant être appliqués localement par chaque agent pour choisir sa valeur locale. Dans notre contexte, la méthode utilisée par un agent pour prendre sa décision fait plutôt partie de la définition du problème. Notre objectif est de permettre à des agents autonomes existant *a priori* de se coordonner, et non pas de concevoir des agents permettant de résoudre un problème de manière distribuée. Selon cette vision des choses, en DCOP le focus est sur le design des *agents*, alors que pour notre part nous faisons face à un problème de design de *mécanisme de coordination* pour un système multi-agent.

4.2.2 Évacuation de la notion de hiérarchie

Dans un cadre DCOP, la notion de hiérarchie entre les sous-problèmes est évacuée. On suppose que le sous-problème de chaque agent est défini *a priori*. Les agents auront à résoudre simultanément ces sous-problèmes en sachant qu'il existe des dépendances entre ceux-ci.

À l'opposé, en contexte hiérarchique, un sous-problème sera défini en fonction des solutions retenues pour les sous-problèmes précédents. Cela est induit par le contexte d'affaires (nous avons utilisé le terme *contraintes organisationnelles* à la section 1.4) de même que par la définition même des agents impliqués. À cet effet, on comprend qu'un agent peut très bien être en mesure de résoudre une instance spécifique de son sous-

problème (e.g. planifier sa production en tenant compte de la demande de son client) sans pour autant savoir résoudre une version ‘générique’ de ce problème (e.g. planifier sa production en ayant aucune idée des produits demandés, ni aucune idée des processus de fabrication utilisés par les autres agents).

Alors que des modèles comme celui de Schneeweiss (section 3.2.2) permettent de bien capturer cette notion de hiérarchie induite par le contexte d’affaires (décomposition en sous-problèmes dépendants et responsabilité de chaque agent), cette notion est évacuée en DCOP.

4.2.3 Capacité d’anticipation

Les algorithmes pour DCOP, tels que ADOPT, supposent que chaque agent est capable de bien évaluer l’impact de ses décisions locales sur l’objectif global; leur bon fonctionnement en dépend. Cela se traduit mathématiquement par l’obligation pour l’agent de pouvoir calculer localement une borne optimiste de bonne qualité pour la fonction objectif globale, de manière à fixer de manière informée la valeur de sa variable. C’est pour cette raison qu’en DCOP la fonction objectif est encodée comme la somme monotone des fonctions objectifs locales. Mais est-ce que formuler l’objectif global de cette façon est une condition suffisante pour garantir que les agents agiront de manière informée?

Pour répondre à cette question, étudions notre problème représenté par le graphe de contraintes à la figure 4.2. Observons la contrainte C_1. Elle spécifie que tout écart entre les livraisons de l’un et les réceptions de l’autre engendre une valeur infinie pour la contrainte. Par ailleurs, la visibilité de la fonction objectif globale qu’ont les deux agents impliqués se limite à la valeur des contraintes dans lesquelles ils sont impliqués. Leur visibilité est donc limitée à la composante de la fonction objectif constituée des contraintes dures. Or, celles-ci sont présentes dans la fonction objectif uniquement parce que nous utilisons une formulation DCOP, notre objectif réel étant de minimiser l’insatisfaction du client externe (contrainte C_3).

La contrainte C_1 incite donc uniquement les deux agents à satisfaire la contrainte dure (sans aucun égard à la satisfaction du client externe). Puisque tout écart entre les livraisons de l'un et les réceptions de l'autre engendre une valeur infinie pour la contrainte, la façon la plus simple pour eux de s'entendre est de décider de ne s'échanger aucun produit.

Autrement dit, tout se déroule comme si les agents en aval ne sont aucunement conscients qu'il existe une demande d'un client externe quelque part; ou à tout le moins tout se déroule comme s'ils supposaient que les agents en amont sont capables de desservir le client final à partir de leurs stocks courants.

Le résultat est que nos agents disposent d'une capacité d'anticipation suffisante pour appliquer l'algorithme, mais insuffisante pour guider de manière éclairé le processus de recherche.

4.2.4 Omnipotence

Les algorithmes pour DCOP comme ADOPT supposent un certain niveau d'omnipotence[25] des agents. Voyons l'exemple suivant dans lequel nous passerons outre aux objections précédentes (l'agent calcule la borne et est d'accord pour prendre sa décision locale sur cette base). Les algorithmes comme ADOPT et DPOP supposent que l'agent est capable, en tout temps, d'énumérer toutes les valeurs possibles pour sa variable (pour le cas où on a une seule valeur par agent) et de choisir celle montrant la meilleure borne. Cela a deux conséquences pratiques : (1) on impose à l'agent l'utilisation d'un algorithme exact pour prendre sa décision (nous avons discuté ce point en 4.2.1), et (2) cette décision doit pouvoir être prise en un temps raisonnable pour que l'algorithme soit utilisable. En contexte industriel, cela suppose donc que le choix du « meilleur plan local en regard de l'objectif global» peut alors être fait seulement si l'agent utilise un modèle d'optimisation (et non une heuristique) capable de fournir la

[25] Au sens suivant défini par le dictionnaire Petit Robert : « puissance absolue, sans limitation. »

solution optimale en un temps raisonnable. En général, pour des problèmes de nature industrielle et de taille réelle, la complexité du problème local est telle que ce n'est pas envisageable. À titre spécifique, les modèles développés pour la planification du rabotage et du séchage n'ont pas permis d'obtenir les solutions optimales, même au terme de plusieurs jours de calcul.

4.2.5 Synthèse des objections

Les deux premiers arguments (4.2.1 et 4.2.2) présentent des objections de nature organisationnelle à l'utilisation du cadre DCOP. On explique pourquoi il est incompatible avec le *cadre logistique* (voir section 1.4).

Les troisièmes et quatrièmes (4.2.3 et 4.2.4) sont des objections de nature algorithmique. Le quatrième constitue un empêchement pratique à l'utilisation d'algorithmes basés sur le calcul des bornes (comme ADOPT et DPOP). Le troisième est plutôt annonciateur de la piètre performance escomptée de ces algorithmes. Toutefois, ces deux arguments ne constituent pas une preuve - loin de là - que tout algorithme résolvant un DCOP ne pourrait s'avérer utile en contexte industriel. En effet, plusieurs problèmes dans le domaine de la gestion des chaînes logistiques pourraient bénéficier du cadre des DCOP et de ses algorithmes. Par exemple, mentionnons Parunak *et al.* qui discutent de l'adaptation des DCOP pour le design collaboratif de produits [62].

4.3 Introduction du cadre proposé (HDCOP)

Cette section introduit un formalisme permettant de modéliser le problème de coordination d'entités autonomes pour la prise de décision distribuée en contexte hiérarchique. L'objectif poursuivi est de permettre aux agents d'explorer systématiquement l'espace de coordination, tout en conservant les mêmes relations d'affaires, responsabilités et algorithmes locaux de décision (c'est donc dire, en respect avec le cadre logistique établi).

Le contexte de base auquel s'applique le cadre peut être décrit de la manière suivante : (1) un problème d'optimisation est naturellement décomposé en sous-problèmes, (2) il existe *a priori* une séquence selon laquelle les sous-problèmes doivent être résolus, (3) les sous-problèmes sont sous la responsabilité de différents agents et (4) chaque sous-problème est défini en fonction des solutions retenues pour les sous-problèmes précédents.

Nous avons vu au chapitre précédent que la séquence de sous-problème peut être représentée à l'aide du formalisme de Schneeweiss (section 3.2.2). Pour obtenir une solution globale, les problèmes sont généralement résolus de manière séquentielle du haut vers le bas. On n'obtient ainsi une et une seule solution.

En contexte industriel, les algorithmes utilisés pour résoudre chacun des problèmes locaux permettent généralement de produire plusieurs solutions alternatives [3]. En considérant chacune d'elles comme une proposition alternative à l'agent suivant, on peut décrire l'espace de coordination comme un arbre non-binaire de profondeur fixe.

Par exemple, considérons le cas suivant où trois agents (A_1, A_2 et A_3) appliquent la planification en deux phases (figure 4.3).

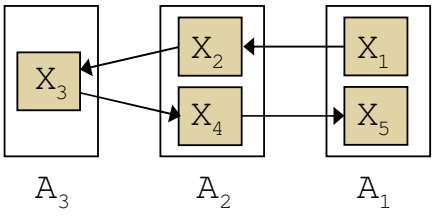


Figure 4.3 : Application de la planification en deux phases par un collectif de trois agents.

L'arbre correspondant (figure 4.4) comportera un niveau par type de sous-problème. Chaque niveau de l'arbre correspondra à une des boîtes de la figure 4.3. Chaque nœud sur un niveau donné est une instance de ce sous-problème (défini par les décisions retenues pour les sous-problèmes précédents). Chaque arc correspond à une solution réalisable pour le sous-problème. Le nombre et la séquence de ces arcs dépend de

l'algorithme local utilisé par l'agent. Chaque feuille de l'arbre (i.e. chaque solution pour le dernier sous-problème) est une solution au problème global.

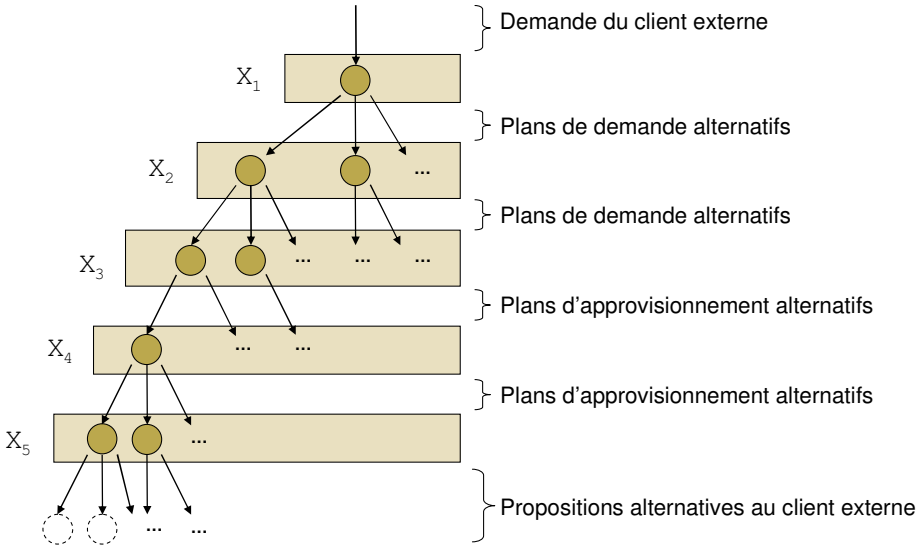


Figure 4.4 : Espace de coordination sous HDCOP.

Plus formellement, nous proposons la définition suivante :

Définition 4.1 : Hierarchical Distributed Constraint Optimisation Problem (HDCOP). Le problème global est défini par un vecteur de sous-problèmes $X = [X_1, \ldots, X_M]$. Chaque sous-problème X_i est sous la responsabilité d'un agent $\mathcal{A}(X_i) \in A = \{A_1, \ldots, A_N\}$. Pour chaque sous-problème X_i, l'agent $\mathcal{A}(X_i)$ dispose d'un solveur $\mathcal{S}^i$ produisant le vecteur S^i de solutions alternatives: $\mathcal{S}^i(X_i) \rightarrow S^i$ où $S^i = \left[S_1^i, \ldots, S_{|S^i|}^i\right]$. Ces solutions locales ne sont pas connues *a priori*; elles seront révélées l'une après l'autre par le solveur. Éventuellement, l'une sera sélectionnée. Nous la dénoterons par S_*^i. Les agents sont à la recherche de la solution globale représentée par le vecteur $\left[S_*^1, \ldots, S_*^M\right]$ pour laquelle la fonction $\mathcal{F}\left(\left[S_*^1, \ldots, S_*^M\right]\right)$ est minimisée.

Chaque sous-problème X_i est défini par les solutions retenues pour les sous-problèmes précédents: $X_i = \mathcal{G}^i\left(\left[S_*^j \middle| 1 \leq j < i\right]\right)$.

4.4 Recherche distribuée dans un arbre en tant que mécanisme de coordination

La précédente reformulation du problème rend possible l'utilisation d'algorithmes classiques de recherche dans un arbre en tant que mécanismes de coordination. Cependant, on doit respecter certaines conditions. Premièrement, on doit respecter la nature distribuée du système. Deuxièmement, on doit respecter les relations d'affaires entre les partenaires. Finalement, on doit tenir compte du fait que l'arbre n'est pas défini *a priori*. En effet, (1) les solutions alternatives (arcs) pour un sous-problème donné (nœud) ne sont pas connues avant d'être produites par le solveur local. Ces arcs sont donc produits/explorés selon un ordre préétabli qui dépend du solveur local utilisé. De plus, (2) les instances de chaque sous-problème ne sont pas connues d'avance puisqu'elles sont définies par les solutions retenues pour les sous-problèmes précédents. L'arbre est donc construit dynamiquement pendant le processus de recherche.

Pour résumer, on peut établir un parallèle avec le domaine plus classique de la résolution centralisée de problèmes combinatoires à l'aide des méthodes de recherche dans un arbre (voir définitions pour CSP et COP au chapitre Chapitre 3). Pour ces problèmes, c'est l'algorithme de résolution qui construit l'arbre de recherche (en choisissant l'ordre d'instanciation des variables et l'ordre des valeurs essayées). À l'opposé, dans le contexte particulier des problèmes hiérarchiques, la séquence de sous-problèmes (équivalent à la séquence de variables) et l'ordre dans lequel les propositions sont produites par les solveurs locaux (équivalent au choix de variable) sont déterminés à l'avance.

Algorithme de base

La méthode la plus simple pour l'exploration de cet arbre par un collectif d'agents consiste à utiliser SyncBB (section 3.3.2). Rappelons que les agents prennent leur

décision locale à tour de rôle. Lorsqu'une feuille est atteinte, l'agent qui a détecté cette condition envoie un message à l'agent précédent lui demandant une proposition alternative. Ce message contient la valeur de la meilleure solution globale trouvée jusqu'à maintenant. Elle peut être utilisée pour réaliser des coupes dans l'arbre pendant le reste de la recherche. Cependant, en contexte industriel, les possibilités de coupes sont très limitées puisque les agents ne connaissent pas les alternatives qui s'offrent aux autres agents (processus de production alternatifs, plans de travail alternatifs, etc).

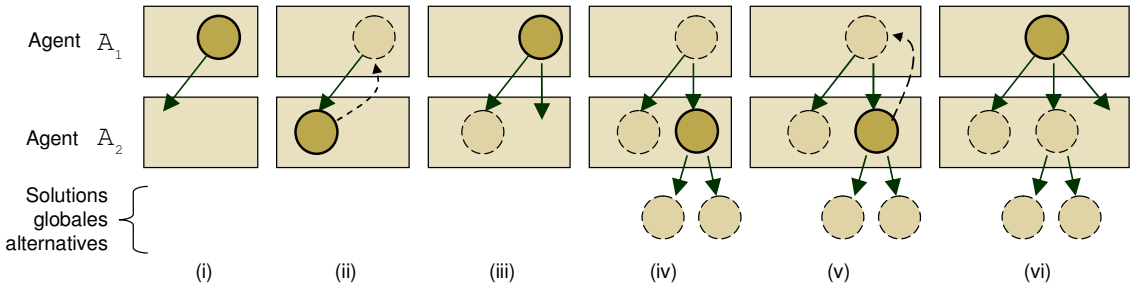


Figure 4.5 : Application de SyncBB pour la résolution d'un HDCOP.

4.5 Discussion

La présente section vise à discuter des limites de l'approche proposée. La section 4.5.1 rappelle que l'*espace de coordination* défini par le cadre logistique et considéré par notre méthode est un sous-ensemble d'un *espace des solutions* plus vaste mais ignorant de nombreuses contraintes de nature organisationnelle. Dans la section 4.5.2 nous expliquerons comment la méthode peut être utilisée comme mécanisme de coordination dans les réseaux de création de valeur non-linéaires. Finalement, la section 4.5.3 présente quelques scénarios pour l'exploitation de notre approche au sein de systèmes de prise de décision inter/intra-organisationnels.

4.5.1 La notion d'optimalité

Le formalisme et la méthode de résolution proposés permettent aux agents de réaliser une recherche distribuée dans *l'espace de coordination*, tout en conservant les mêmes relations d'affaires, la même attribution des responsabilités et les mêmes algorithmes

locaux de prise de décision. Cet espace de coordination est défini par les relations établies entre les agents (la séquence de sous-problèmes, notamment) et par les propositions que chaque agent juge acceptable de transmettre aux autres agents (révélées par l'algorithme local de l'agent).

Cependant, si on fait abstraction du cadre logistique dans lequel s'inscrivent les relations d'affaires entre les agents, il existe probablement de meilleures solutions au problème global que celles de l'espace de coordination. De ce point de vue, on peut considérer l'espace de coordination comme un sous-ensemble d'un ensemble plus vaste : l'*espace des solutions*.

4.5.2 Coordination dans des réseaux de création de valeur non linéaires

La méthode proposée s'applique pour peu qu'il existe une séquence identifiée de sous-problèmes. Est-ce à dire que pour un réseau de création de valeur, nous ne pouvons considérer que le cas spécifique des chaînes linéaires? Ce n'est pas le cas. Nous verrons ici l'exemple d'un réseau de production convergent où un agent est approvisionné par deux fournisseurs (figure 4.6).

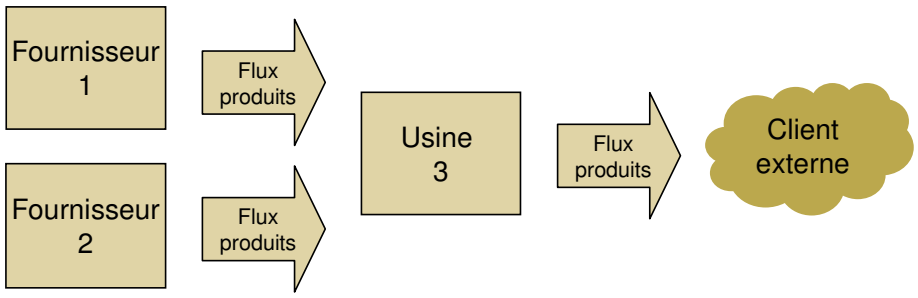


Figure 4.6 : Flux de produits dans un RCV convergent.

On présente à la figure 4.7 un exemple de protocole de coordination qui pourrait être utilisé dans ce contexte. Selon ce scénario, l'usine prépare d'abord un plan temporaire (1). Elle calcule ainsi ses besoins en matière première qu'elle transmet au fournisseur 1. Le fournisseur cherche à satisfaire à cette demande (2) et répond par un plan

d'approvisionnement qui ne rencontre pas nécessairement toute la demande (e.g. il peut être prévu que certaines quantités soient en retard). Sur la réception de ce plan, l'usine révise son plan de production en tenant compte des nouvelles contraintes d'approvisionnement (3). À partir de ce nouveau plan de production, elle calcule ses besoins en matière pour le fournisseur 2. En retour, le fournisseur 2 cherche à satisfaire cette demande (4). Finalement, l'usine révise son plan de production encore une fois (5). Parce que ce protocole définit une séquence de sous-problèmes; il peut être généralisé de manière à représenter l'espace de coordination sous la forme d'un HDCOP.

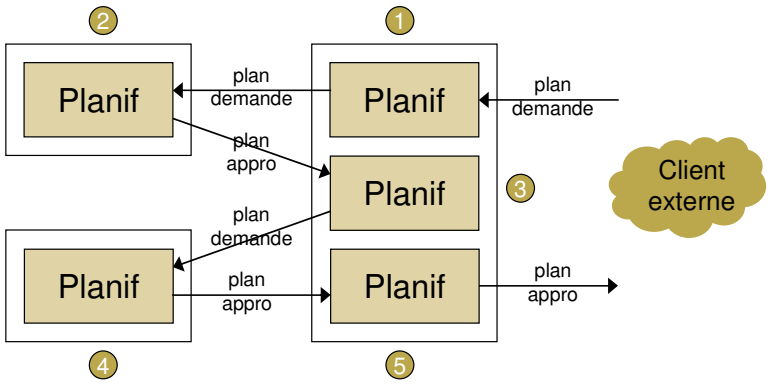


Figure 4.7 : Séquence de sous-problèmes pour la coordination dans un RCV convergent.

Dans le scénario précédent, le premier fournisseur dispose d'un avantage concurrentiel sur le deuxième. Bien qu'il soit raisonnable d'imaginer un tel contexte d'affaires, nous présentons un second scénario qui place les deux fournisseurs sur le même pied (figure 4.8). En vertu de ce scénario, l'usine décide à l'étape 1 de la répartition de sa demande entre ses deux fournisseurs (à l'aide d'un critère de son choix). Ensuite, les deux fournisseurs doivent répondre par un plan de demande. L'usine replanifie ses activités seulement après voir obtenu la réponse des deux fournisseurs.

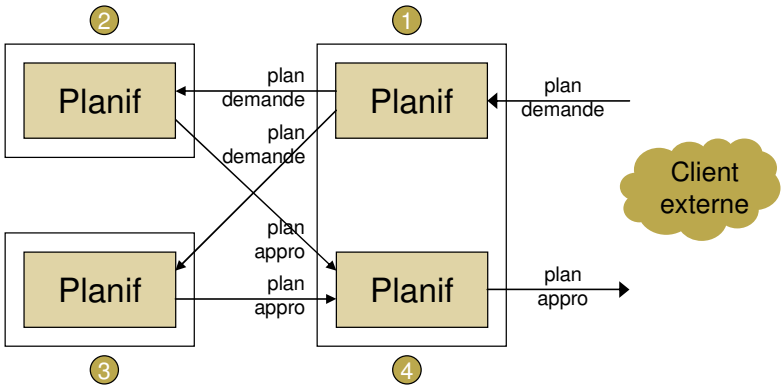


Figure 4.8 : Scénario alternatif pour la coordination dans un réseau RCV convergent.

D'autres scénarios correspondant à d'autres contextes d'affaires pourraient être analysés. L'élément clé à retenir est que la méthode présentée est applicable pour peu qu'il existe une séquence de résolution identifiable *a priori*. Cependant, il n'est pas clair que cette condition puisse être rencontrée pour des réseaux de création de valeur divergents; il est difficile d'imaginer pour ces situations une séquence de résolution naturelle et identifiée *a priori*. Nous allons alors au-delà des limites couvertes par l'approche telle que présentée.

4.5.3 Exploitation de l'approche proposée au sein d'un système industriel de prise de décision

L'approche proposée dans ce chapitre pourrait être exploitée de différentes manières au sein d'un système de prise de décision industriel. Dans cette section, nous désirons illustrer quelques scénarios d'utilisation. Notre objectif n'est pas de répertorier de manière systématique tous les schémas possibles; nous présenterons les trois principaux.

Les deux premiers scénarios concernent l'utilisation de la méthode au sein d'un système avancé de planification et d'ordonnancement distribué (en anglais, *Distributed Advanced Planning and Scheduling*, ou d-APS). Le troisième est un peu plus général; il exploite l'approche au sein d'un système non distribué.

Scénario 1. Dans le cadre de ce scénario, nous supposons que chaque unité décisionnelle dispose d'un système informatique lui permettant de planifier ses opérations locales. Il prend la forme d'un agent logiciel capable de communiquer avec les agents des autres unités. Lorsque l'unité en contact avec les clients externes reçoit une nouvelle commande (ou demande), le processus de planification collaboratif est déclenché. Les agents travaillent alors collectivement à établir le meilleur plan possible (à l'aide des techniques présentées dans ce chapitre) jusqu'à ce que la limite de temps alloué à la coordination soit rencontrée. Différents cas de figures alternatifs peuvent être considérés :

(1) Le nouveau plan est bâti à partir de zéro (on fait table rase du plan antérieur);

(2) Les opérations qui étaient déjà prévues (pour satisfaire des commandes reçues dans le passé) sont maintenues. On ne peut qu'en ajouter des nouvelles;

(3) Le plan antérieur est modifié sous réserve de certaines contraintes préétablies. Selon cette vision des choses, les agents peuvent s'attaquer à leurs sous-problèmes en utilisant un algorithme de *recherche locale* (en anglais, *local search*); le plan antérieur constitue la solution servant à initialiser l'algorithme de *recherche locale*.

Scénario 2. Dans ce scénario, on suppose que les décisions d'acceptation/refus de commandes et les décisions de planification des entreprises sont d'abord prises en utilisant un mécanisme quelconque (différent de ce qui est présenté dans ce chapitre). Certaines commandes peuvent même être acceptées de manière tentative. Le mécanisme de planification collaboratif que nous avons introduit n'est utilisé que de façon périodique (e.g. pendant la nuit, une fois par semaine, ou autre) dans le but de trouver un meilleur plan.

Scénario 3. Le dernier scénario concerne une situation de prise de décision pouvant être décrite dans un cadre hiérarchique (en utilisant le formalisme de Schneeweiss, par exemple), mais pour lequel le calcul/résolution n'a pas obligatoirement à être réalisé de manière distribuée.

Pensons à un système de décision à l'intérieur d'une organisation. Supposons qu'un problème global est naturellement décomposé en sous-problèmes et qu'on dispose de bons algorithmes pour chacun des sous-problèmes (e.g. planification de la production, planification du transport et planification des horaires de travail). Dans cette situation, l'approche classique consisterait à appliquer un algorithme glouton, i.e. résoudre la séquence des sous-problèmes l'un après l'autre. L'alternative que nous proposons est d'utiliser le formalisme HDCOP de manière à généraliser l'heuristique. On réalisera ensuite une recherche dans l'arbre correspondant, de manière à découvrir de bonnes solutions globales alternatives. C'est, par exemple, ce que pourrait faire une entreprise intégrée qui souhaiterait planifier de manière « centralisée » ses opérations de sciage, séchage et finition - puisqu'il n'existe pas d'algorithme/modèle unifié utilisable en pratique qui permet de planifier ces trois types d'opérations en une seule étape.

Dans ce cas l'approche HDCOP est une alternative intéressante permettant l'intégration d'algorithmes utilisant des technologies différentes les unes des autres (modèles mathématiques, heuristiques, recherche locale, etc.)

4.6 Conclusion

Nous avons vu dans ce chapitre pourquoi l'utilisation du cadre standard DCOP est difficilement envisageable pour la coordination en contexte industriel hiérarchique.

En conséquence, nous avons proposé un nouveau cadre appelé HDCOP. Le formalisme HDCOP permet de modéliser le problème tout en tenant compte des contraintes organisationnelles. L'espace de coordination est représenté sous la forme d'un arbre, ce qui permet d'utiliser un algorithme de recherche distribué en tant que mécanisme de

coordination. De ce point de vue, la méthode proposée partage à la fois les caractéristiques des approches heuristiques basées sur la communication, et celles des approches basées sur l'algorithmie et le calcul (voir tableau 3.1).

Par ailleurs, on peut avancer sans se tromper que la capacité du collectif d'agents à découvrir rapidement de bonnes solutions dépendra de la stratégie d'exploration de l'arbre HDCOP. Les deux prochains chapitres portent sur cette question.

Chapitre 5 Multi-Agent Concurrent Discrepancy Search (MacDS) : retour-arrière basé sur l'analyse des déviations en contexte multi-agent

Au chapitre précédent, nous avons proposé l'utilisation de l'algorithme *Synchronous Branch and Bound* (SyncBB) dans le but de résoudre un HDCOP. SyncBB présente deux lacunes principales. Premièrement, un seul agent à la fois est au travail. Deuxièmement, il applique le retour-arrière chronologique. Or, en environnement centralisé, le retour-arrière chronologique est souvent surpassé par des méthodes basées sur l'analyse des *déviations*[26], comme par exemple *Limited Discrepancy Search* (LDS) [63,64].

Ces méthodes utilisent le nombre de *déviations* comme un indicateur de la qualité d'un nœud ou d'une solution, de manière à guider la recherche dans l'arbre. Ce concept des déviations pourrait-il être mis à profit dans notre contexte?

Ce chapitre propose deux méthodes permettant d'exploiter cette même idée en contexte multi-agent. La première méthode (SyncLDS) est un algorithme de recherche distribué dit *synchrone* (comme pour SyncBB un seul des agents à la fois est au travail). La seconde (MacDS) permet aux agents de travailler simultanément. Il montre également certaines propriétés intéressantes en termes de robustesse.

---

[26] En anglais : *discrepancies*. Nous définirons le terme formellement à la section 5.1. Certains auteurs francophones préfèrent utiliser le terme *divergence*. Pour notre part, nous préférons l'éviter car en contexte multi-agent, on a tendance à interpréter le terme dans le sens de « divergence d'opinion ».

5.1 Limited Discrepancy Search (LDS)

La première méthode de recherche centralisée basée sur l'analyse des déviations (*Limited Discrepancy Search*, LDS) a été introduite par Harvey et Ginsberg dans [63,64].

L'idée principale derrière LDS est que les feuilles d'un l'arbre (solutions) n'ont pas toutes *a priori* la même qualité espérée ; que celle-ci décroît en fonction du nombre de fois qu'il faut brancher à droite pour aller de la racine jusqu'à cette feuille (i.e. le nombre total de déviations, voir figure 5.1).

L'explication est la suivante : les arcs sous un nœud (les solutions alternatives pour un sous-problème, dans notre cas) sont ordonnés en vertu d'un critère (l'algorithme local utilisé, dans notre cas). En conséquence, chaque branchement vers la droite est un branchement qui va à l'encontre de ce critère. LDS vise donc à visiter prioritairement les feuilles associées à un nombre de déviations faible.

Cette technique a également pour effet que les solutions visitées en un temps très court seront davantage différentes les unes des autres que dans le cas d'un retour-arrière chronologique (car dans ce cas, elles diffèrent uniquement de par la solution retenue pour le dernier sous-problème).

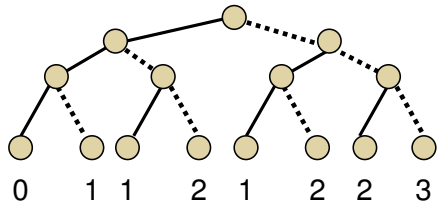


Figure 5.1: Arbre binaire et nombre de déviations associées à chaque feuille.
Il correspond au nombre de fois qu'il faut brancher à droite pour passer de la racine jusqu'à la feuille (arcs pointillés sur la figure).

LDS a d'abord été introduit pour la résolution de problèmes de satisfaction de contrainte. Le concept a ensuite été appliqué avec succès pour l'optimisation [65]. Pour

les arbres non-binaires[27], il a été proposé de compter les déviations comme suit : le `i`-ème arc suivi à un nœud donné compte pour `i-1` déviations [65].

Dans la publication originale, LDS était décrite de manière procédurale mais l'idée peut également être exploitée de manière à définir un « sélecteur de nœud » utilisable dans un engin de recherche standard [66] : lorsque les conditions pour un retour-arrière sont rencontrées, l'engin de recherche active le nœud déjà visité pour lequel le prochain fils non visité a le nombre total de déviations le plus faible. C'est pourquoi LDS peut être considéré comme une politique (ou stratégie) de retour-arrière.

D'autres variantes ont été proposées par la suite (e.g. [66,67]). Plusieurs ont été intégrées dans des solveurs commerciaux, notamment dans ILOG SOLVER.

5.2 Synchronous LDS (SyncLDS)

Cette section propose un protocole simple (SyncLDS) permettant à une équipe d'agents d'appliquer une stratégie de retour-arrière de type LDS. Dans la méthode proposée, l'arbre n'existe nulle part à proprement parler; chaque agent ne connaît que les nœuds qui sont sous sa juridiction mais les solutions globales seront visitées dans le même ordre que si on réalisait un LDS centralisé dans l'arbre centralisé équivalent.

Comme pour SyncBB un seul agent à la fois est actif et une solution est obtenue en résolvant la séquence de sous-problème. La différence avec SyncBB se situe au niveau de la stratégie de retour-arrière. La transition d'un agent à l'autre se fait par l'échange d'un message qui peut être interprété comme la transmission d'un privilège (ou jeton). Nous détaillerons ce mécanisme dans les prochains paragraphes.

Chaque fois qu'un agent transmet une proposition à son successeur, il inscrit dans le message quel serait le chemin menant à cet arc dans l'arbre global (pour ce faire, l'agent

---

[27] À la différence d'un arbre n-aire, pour lequel chaque nœud a le même nombre de fils (exactement `n`), chaque nœud d'un arbre non-binaire peut avoir un nombre quelconque de fils.

n'a qu'à compter le nombre de solutions locales produites jusqu'à maintenant pour le nœud courant, soustraire un, et concaténer ce nombre au chemin reçu avec la proposition de l'agent précédent).

Lorsque qu'une solution globale est trouvée, l'agent qui détecte cette condition envoie un message à tous les agents leur demandant d'identifier le nœud sous leur juridiction pour lequel le prochain fils aurait le plus petit nombre de déviations. On peut interpréter cela comme un appel d'offres; l'agent qui a détecté la condition de retour-arrière donne le contrôle à celui qui fournira la plus petite réponse. La prochaine solution globale est obtenue à partir de ce point par la résolution séquentielle des sous-problèmes restants.

5.2.1 Pseudocode

Chaque agent exécute le pseudocode présenté à la figure 5.2. La fonction principale (`MsgProposition`) est déclenchée par la réception d'une proposition (solution locale en provenance de l'agent précédent). Cette proposition est dénotée par $\langle$`d, p[ ]`$\rangle$. L'élément `d` représente les décisions pour les sous-problèmes précédents et `p[ ]` est le vecteur d'entiers représentant le chemin correspondant à ce nœud dans l'arbre global.

Sur réception du message, l'agent crée un nœud correspondant à ce sous-problème. Le nœud est dénoté par le tuple $\langle$`d, p[ ], n`$\rangle$. L'élément `n` indique le nombre de solutions locales déjà obtenues pour ce nœud; il est donc initialement à zéro. L'agent ajoute ce nœud à sa liste de nœuds locale (`nodeList`) et s'attaque à ce sous-problème (`Work(node)`). Il trouve une première solution à ce sous-problème qu'il envoie en tant que proposition à l'agent suivant (`send MsgProposition`). S'il n'y a pas d'agent suivant, on a trouvé une solution au problème global.

On doit ensuite retraiter collectivement vers le nœud qui est davantage prioritaire (`CooperativeBacktrackingLDS`). Chaque agent doit identifier localement le nœud de sa liste qui est prioritaire. Les nœuds sont comparés sur la base du nombre total de

déviations dans leur chemin `p[ ]` à l'aide de la fonction `CompareLDS()`[28]. En cas d'égalité, on applique une politique de retour-arrière chronologique pour départager les nœuds (`CompareBT`). Les soumissions de chaque agent sont ensuite comparées de la même façon. Cette approche rappelle l'implémentation d'une stratégie de retour-arrière en environnement centralisé en utilisant le principe d'un sélecteur de nœud (en anglais, *node selector*), à la différence qu'ici le principe est d'abord appliqué localement par chaque agent.

Tout comme dans SyncBB, les agents sont informés de la qualité de la meilleure solution trouvée jusqu'à maintenant. Cela n'est cependant pas illustré dans le pseudocode.

[28] En pratique, chaque nœud peut maintenir sa liste triée en tout temps; retourner l'élément le plus prioritaire consiste alors à simplement retourner celui en tête (ou en fin) de liste.

```
WhenReceive MsgProposition(<d,p[]>) do
 node ← <d, p[], n=0>
 nodeList.add(node)
 Work(node)

Procedure Work(node)
 proposition ← NextSolution(node);
 if (proposition ≠ ∅)
    node.n ← node.n+1
    if (Successor(node) ≠ ∅)
       send MsgProposition(<proposition, node.p[]+[node.n-1]>)
       to Successor(node)
    else
       CooperativeBacktrackingLDS()
 else
    nodeList.remove(node)
    CooperativeBacktrackingLDS()

Procedure CooperativeBacktrackingLDS()
 send MsgAskBestLocalNode() to EveryBody
 answers[] ← all answer from EveryBody : (answer ≠ ∅)
 candidate ← best node in answers[] according to function CompareLDS()
 send MsgBacktrack(node) to Agent(answer)

WhenReceive MsgAskBestLocalNode() do
 if (nodeList.count = 0) return ∅
 else return best node in nodeList according to function CompareLDS()

WhenReceive MsgBacktrack(node) do
 Work(node)

Function CompareLDS(p1, p2)
 t1 ← Σ(j=0..Card(p1)-1) p1[j]
 t2 ← Σ(j=0..Card(p2)-1) p2[j]
 if (t1 < t2) return p1
 else if (t2 < t1) return p2
 else return CompareBT(p1, p2)

Function CompareBT(p1, p2)
 depth ← Min(Card(p1), Card(p2))
 j ← 0
 while (p1[j] = p2[j] and j < depth) j ← j+1
 if (j < depth)
     if (p1[j] ≤ p2[j]) return p1 else return p2
 else
     if (Card(p1) ≥ Card(p2)) return p1 else return p2
```

Figure 5.2 : SyncLDS – Pseudocode.

5.3 Multi-agent Concurrent Discrepancy Search (MacDS)

Cette section présente un autre algorithme permettant aux agents d'appliquer une stratégie de retour-arrière basée sur l'analyse des déviations. Il permet le travail simultané des agents, utilise des communications asynchrones [68], est tolérant aux délais aléatoires de transmission de messages. Il permet aussi de tirer avantage des situations où les temps de résolution des sous-problèmes sont différents les uns des autres.

Rappelons qu'il existe dans la littérature deux approches pour permettre le travail simultané (voir section 3.3). Nous utiliserons une approche inspirée des *Concurrent search algorithms* (CSA).

5.3.1 Présentation générale

Nous décrirons d'abord l'algorithme de manière informelle à l'aide d'un exemple impliquant trois agents (figure 5.3). À la manière des CSA, toute solution sera obtenue par la résolution séquentielle des sous-problèmes. La première feuille est atteinte en résolvant la séquence de sous-problèmes (sous-figure i). Comme pour SyncLDS, les agents attachent à chaque message l'information concernant le chemin qui irait de la racine au nœud dans l'arbre global correspondant (sur la figure, ces chemins sont affichés sur les arcs).

La différence principale avec SyncLDS est la suivante : aussitôt qu'un agent a transmis sa proposition, il commence immédiatement à travailler sur une proposition alternative qu'il transmettra également aussitôt qu'elle sera prête. Dans la sous-figure (i), le système est dans un état où une première solution globale a été obtenue; chaque agent travaille à produire une solution locale alternative. Éventuellement, un des agents va disposer d'une proposition alternative. Supposons qu'il s'agit de l'agent `C` (sous-figure ii); nous avons alors notre seconde solution globale. L'agent `C` commence alors à travailler sur une troisième proposition. Supposons que l'agent `B` est le prochain à produire et transmettre

une proposition (sous-figure iii) : l'agent C doit alors déterminer s'il est préférable de produire une troisième solution basée sur le nœud sur lequel il travaille déjà (à gauche), ou bien une première solution basée sur le nouveau nœud (à droite). Cette décision sera prise de manière locale base sur l'analyse des déviations.

Dans notre exemple (sous-figure iii), l'agent C doit choisir entre produire une solution qui engendrerait un total de 2 déviations (0+0+2) ou une solution ayant 1 déviation (0+1+0). Pour appliquer une politique de type LDS (plus petit nombre de déviation en premier), l'agent choisira la seconde option (sous-figure iv). Dans la sous-figure (v), l'agent a reçu un nouveau message de l'agent A et vient de s'attaquer à ce nouveau nœud.

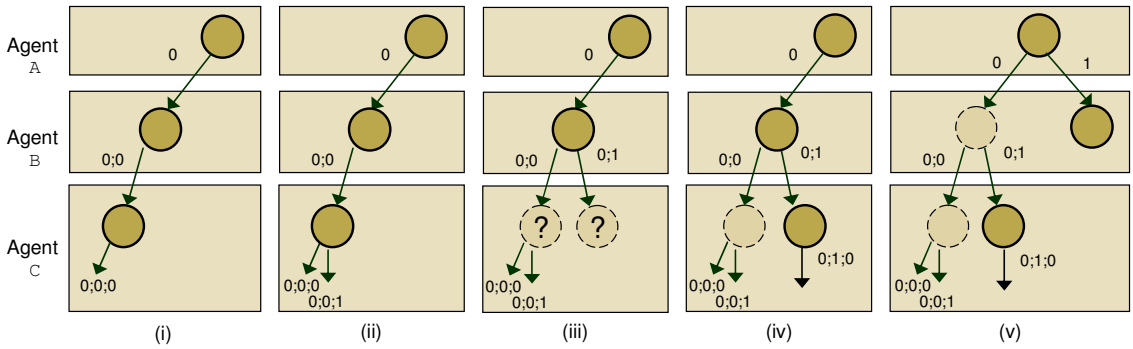


Figure 5.3 : MacDS – Trace d'exécution partielle.

Pour résumer, chaque agent gère une liste de nœuds (correspondant aux solutions alternatives reçues de l'agent précédent). Avec chaque proposition transmise, l'agent attache l'information concernant le chemin qui irait de la racine au nœud correspondant dans l'arbre correspondant au problème global. À tout moment, l'agent travaille sur le nœud/sous-problème de sa liste ayant le plus haut niveau de priorité. Cette priorité est établie en fonction du chemin menant au nœud et du nombre de solutions locales déjà produites pour ce sous-problème (e.g. le nombre total de déviation associées à la prochaine solution locale). En changeant cette fonction de sélection, on pourrait

implémenter d'autre stratégies que LDS, comme *Depth-bounded Discrepancy Search* (DDS) [67], ou même un retour-arrière chronologique[29].

Tout comme pour SyncBB et SyncLDS, la valeur de la meilleure solution trouvée jusqu'à maintenant est propagée aux agents précédents (bien que ce ne soit pas illustré par le pseudocode qui suit).

Dans le cas extrême ou un seul agent serait propriétaire de tous les sous-problèmes, MacDS visiterait alors les nœuds dans le même ordre que le ferait un algorithme centralisé appliquant la même stratégie de retour-arrière. Dans un contexte distribué ou chaque agent gère sa liste de nœuds, chaque solution au problème global est obtenue en un temps égal ou inférieur à ce qui serait nécessaire en environnement centralisé (en faisant abstraction des délais de communication). Chaque agent se met au travail aussitôt qu'il a un nœud dans sa liste, mais suspend ce travail lorsqu'un nœud de priorité plus grande est ajouté dans sa liste.

5.3.2 Pseudocode

Dans l'implémentation proposée, chaque agent exécute plusieurs fils d'exécution (en anglais, *thread*). Il y a d'abord un fil qui sert au contrôle de l'agent, de même qu'un fil supplémentaire pour chaque nœud/sous-problème à résoudre. Pour un agent, le fil d'un seul nœud peut être actif à tout moment.

Le fil de contrôle (figure 5.4) est activé lorsque l'agent reçoit un message (`MsgProposition`) et lorsque l'agent vient tout juste de produire une nouvelle solution pour un nœud (`WhenNewSolution`). L'agent met alors à jour sa liste de nœuds et active le fil d'exécution correspondant au nœud avec le plus haut niveau de priorité (`ActivateANode`). Dans le pseudocode présenté, on suppose que la liste des nœuds est

[29] Dans ce dernier cas, MacDS réalise en quelque sorte l'équivalent d'un SyncBB, mais avec travail simultané des agents.

triée en ordre décroissant de priorité. Les fonctions `CompareLDS()` et `CompareBT()` présentées plus tôt à la figure 5.2 sont des exemples de fonction pouvant être utilisées pour maintenir cette liste triée, dépendamment de la stratégie que l'on désire appliquer.

```
WhenReceive MsgProposition(<d,p[]>) do
 if (running ≠ ∅) running.Sleep()
 nodeList.insert(<d, p[], 0, false>)
 ActivateANode()

WhenNewSolution(node, solution)
 node.Sleep()
 node.n ← node.n+1
 proposition ← PrepareProposition(node.d, solution)
 send MsgProposition(<proposition, node.p[]+[node.n-1]>)
 to Successor(node)

 if (node.noMoreSol)
     nodeList.remove(node)
     running ← ∅
 ActivateANode()

Procedure ActivateANode()
 if (nodeList.count() > 0)
     running ← nodeList[0]
     running.WakeUp()
 else
     running ← ∅
```

Figure 5.4 : MacDS – Pseudocode du fil de contrôle de l'agent.

Le pseudocode correspondant aux fils d'exécution des nœuds est présenté à la figure 5.5. Lorsque le nœud est créé, son fil d'exécution est inactif. Il doit d'abord être activé par le fil de contrôle. Ensuite, lorsqu'il produit une nouvelle solution pour le sous-problème, il le signale au fil de contrôle (`SignalNewSolution`) et tombe en dormance (`sleep`). Le fil de contrôle transmet alors un message à l'agent propriétaire du prochain type de sous-problème.

```
Procedure Run(node)
  node.noMoreSolution ← false;
  node.solution ← NextSolution(node);

  while (node.solution ≠ ∅)
      SignalNewSolution(node, solution);
      Sleep();
      node.solution ← NextSolution(node);

  node.noMoreSolution ← true;
  SignalNewSolution(node, solution);
```

Figure 5.5 : MacDS – Pseudocode pour le fil d'exécution associé à un nœud.

5.3.3 Caractéristiques de l'algorithme

Mentionnons les principales caractéristiques de l'algorithme :

- L'algorithme est complet (puisqu'il explore le même espace de solutions que SyncBB) et sa terminaison est garantie : on explore le même espace que SyncBB et SyncLDS, le mécanisme du sélecteur de nœud ne fait que modifier l'ordre de production des solutions. La totalité de l'espace des solutions sera donc exploré et la solution optimale retournée.

- Puisque nous imitons le comportement d'une stratégie basée sur les déviations, nous pouvons espérer capturer une partie des gains que ces approches rendent possibles en environnement centralisé.

- Les agents travaillent simultanément. Le mécanisme utilisé permet que chaque agent soit actif pour peu qu'il existe une tâche dans sa liste. Il n'est pas garanti que le système réalise à chaque fois l'équivalent d'un LDS car il en déviera (le moins possible) lorsque cela permet d'éviter qu'un agent se retrouve en situation d'attente.

- D'autres fonctions de priorisation des tâches que celle correspondant à la politique LDS peuvent être utilisées.

- En cas de panne de communication isolant deux sous-réseaux d'agents, chaque sous-réseau isolé continue à travailler de manière autonome. Si la fonction de priorisation des tâches est de type LDS, chaque sous-réseau réalise ce qui ressemble le plus à un LDS compte tenu des tâches disponibles. Quand la communication revient, un arrivage important de messages est à prévoir; chaque agent continue alors le travail en traitant d'abord les tâches prioritaires, convergeant progressivement vers un comportement DDS.

- Si les délais de communication sont variables (l'ordre d'arrivée de deux messages peut être inversé par rapport à l'ordre d'expédition), une tâche pourra donc être traitée par un agent avant une tâche davantage prioritaire qui aurait dû se trouver dans la liste à ce moment. En traitant immédiatement la tâche la plus prioritaire qui se trouve dans la liste, on évite que notre agent soit bloqué à cause d'un message qui n'arrive pas. De cette manière, il n'est pas obligatoire que les messages arrivent dans le même ordre qu'ils ont été expédiés (alors que cette garantie est obligatoire pour les algorithmes synchrones et certains algorithmes asynchrones, comme par exemple ADOPT).

5.3.4 Gestion de la mémoire

Le principal désavantage de l'algorithme est la quantité de mémoire nécessaire pour chaque agent. Dans le pire cas, le dernier agent aura dans sa liste autant de tâches qu'il existe de nœuds au niveau `NbAgents-1` de l'arbre représentant l'espace des solutions. Si chaque agent peut produire en moyenne `m` solutions pour chaque sous problème, alors le dernier agent pourra avoir $m^{NbAgents-1}$ tâches dans sa liste. Toutefois, il est possible de contrôler la consommation de mémoire en procédant selon l'une des approches suivantes:

- Supposons un agent pouvant estimer le temps t_τ nécessaire pour réaliser une tâche τ. Supposons également `PR(t)` la fonction de densité pour le temps alloué

à la coordination. Aucune tâche ne pourra être réalisée lorsque le temps présent $\left(t^{now}\right)$ sera égal à t^{lim}, où t^{lim} est défini comme suit :

$$t^{lim} = \min z : \int_{z}^{\infty} PR(t)\, dt = 0.$$

Ainsi, en tout temps, on pourra conserver dans la liste uniquement l'ensemble des tâches les plus prioritaires pouvant être réalisées avant l'atteinte de t^{lim}. Après l'insertion d'une nouvelle tâche, on supprimera donc les tâches en fin de liste qu'il serait impossible de réaliser de toute façon.

- On peut également imposer une limite supérieure à la quantité de mémoire allouée à la liste de tâches. Après l'insertion d'une nouvelle tâche, les tâches les moins prioritaires causant un dépassement de quantité de mémoire sont supprimées. Contrairement à l'approche précédente, on modifie ainsi le comportement du système en termes de nombre de solutions obtenues.

La première approche borne donc la liste en fonction du temps de calcul disponible, la seconde en fonction de la mémoire disponible. Pour un problème en particulier pour lequel le temps moyen d'exécution d'une tâche est estimé, et pour lequel la quantité de mémoire moyenne associée à une tâche est estimée également, on pourrait calculer la quantité de mémoire nécessaire pour que la liste soit tronquée à cause d'un manque de temps plutôt qu'un manque de mémoire.

Par ailleurs, puisque l'agent ne réalise qu'une tâche à la fois, la mémoire secondaire de l'ordinateur peut être utilisée pour stocker les tâches inactives sans causer de dégradation dramatique des performances que cause normalement l'utilisation de la mémoire virtuelle. Une approche simple consiste à implémenter les fils d'exécution sous la forme de vrais processus indépendants (du point de vue du système d'exploitation). Cela permet de prendre avantage d'automatismes du système d'exploitation : la mémoire des processus inactifs est transférée vers la mémoire secondaire lorsque la mémoire

primaire vient à manquer. C'est l'approche que nous avons utilisé dans notre implémentation dont il sera discuté à la section suivante. Dans les expérimentations dont il sera question, c'est la limite de temps qui constituait la contrainte (plutôt que la limite de mémoire) puisque les sous-problèmes sont longs à résoudre.

5.4 Expérimentations

MacDS a d'abord été évalué pour un cas réel de coordination dans un réseau de création de valeur de l'industrie des produits forestiers. Il s'agit du cas décrit au chapitre Chapitre 2. Les résultats sont rapportés à la section 5.4.1. Nous l'avons ensuite évalué pour un large éventail de problèmes synthétiques montrant différentes caractéristiques (section 5.4.2).

Dans ces expériences, nous comparons MacDS configuré de manière à appliquer une politique de type LDS (ci-après, MacDS_LDS) avec l'algorithme Synchronous Branch and Bound (SyncBB) qui réalise un retour-arrière chronologique (en anglais, *chronological bactracking*, ou BT). Pour mesurer quelle partie du gain est reliée au travail simultané des agents et quelle partie est reliée à la stratégie de retour-arrière, nous évaluons également SyncLDS de même qu'une version de MacDS appliquant le retour-arrière chronologique (MacDS_BT).

Le tableau 5.1 classifie les algorithmes évalués en fonction de leurs principales caractéristiques.

Tableau 5.1 : Caractéristiques des algorithmes comparés dans le cadre de l'évaluation de MacDS.

	Retour-arrière chronologique (BT)	Limited Discrepancy Search (LDS)
Travail synchrone	SyncBB	SyncLDS
Travail simultané	MacDS_BT	MacDS_LDS

5.4.1 Évaluation pour le cas industriel

Les données industrielles ont été extraites des systèmes informatiques industriels de type ERP (*Enterprise Ressource Planning*) à différents moments de l'année 2005. Pour chaque usine, on dispose d'un solveur permettant de planifier ses opérations. Ceux-ci permettent de produire des solutions alternatives. Le mécanisme de coordination par défaut qui est utilisé est la planification en deux phases (voir section 3.2.1 et figure 3.3). Tous les éléments permettant de formuler le problème de coordination à la manière d'un HDCOP sont donc réunis.

Les expérimentations ont été réalisées dans un environnement distribué où chaque agent fonctionne sur un ordinateur différent. Cela permet de mesurer l'impact réel du travail simultané des agents dans un contexte ou les sous-problèmes sont complexes (le problème centralisé équivalent comprendrait des centaines de milliers de variables et des millions de solutions).

La première solution obtenue avec chacun des algorithmes évalué est toujours la même : elle correspond à la première feuille de l'arbre; c'est aussi l'unique solution qui serait obtenue avec le mécanisme de coordination de base (*two-phase planning*). Conséquemment, nous avons comparé les algorithmes sur la base du gain supplémentaire qu'ils permettent d'obtenir (pourcentage de réduction de la fonction objectif) en fonction du temps de calcul supplémentaire alloué à l'algorithme (jusqu'à une heure). Dans notre cas, la fonction objectif correspond au *coût* des retards (volumes livrés en retard au client multiplié par la durée du retard). Le graphique montre donc le pourcentage de réduction de ces retards permis par l'algorithme.

La figure 5.6 présente les résultats pour les quatre cas industriels étudiés (i, ii, iii, iv). SyncBB et MacDS_LDS donnent des résultats comparables dans les premières secondes. Cela s'explique par le fait qu'ils produisent les mêmes solutions tant que le dernier agent ne reçoit pas une deuxième tâche dans sa liste. Par la suite, MacDS_LDS prend généralement le dessus de manière importante : alors que SyncBB persiste à

n'explorer que des variations mineures des premières solutions, MacDS permet l'exploration de zones différentes de l'arbre de recherché. L'exception est le cas (ii) pour lequel SyncBB arrive gagnant avec un écart de **0.5%**.

On peut voir l'impact de la stratégie de retour-arrière seule en comparant SyncBB et SyncLDS. Ce dernier surpasse SyncBB, excepté pour des temps de calculs très courts.

Concernant l'impact du travail simultané des agents, on peut le voir en comparant MacDS_LDS et SyncLDS. Pour toute solution atteinte par SyncLDS, MacDS_LDS produit une solution de qualité égale ou supérieure en un temps de calcul égal ou inférieur. La réduction moyenne du temps de calcul pour chacun des cas sont les suivantes : **18.5%**, **88.7%**, **54.5%** et **64.0%**.

Deux facteurs expliquent ces résultats. Le travail simultané des agents fait que chaque feuille est visitée en un temps égal ou inférieur à ce qui est nécessaire pour le cas synchrone, mais donne également la possibilité d'explorer plus de solutions pour un même temps de calcul.

MacDS_BT a donné exactement les mêmes résultats que SyncBB. Pour cette raison, il n'apparaît pas sur les figures. L'explication est la suivante. Une recherche en profondeur implique de se concentrer uniquement sur le dernier sous-problème tant que ses solutions n'ont pas toutes été énumérées. Pour nos problèmes industriels, cela demande un temps considérable. Dans MacDS_BT, les agents précédents produisent pendant ce temps des solutions partielles alternatives mais elles n'en viennent jamais à être exploitées par le dernier agent.

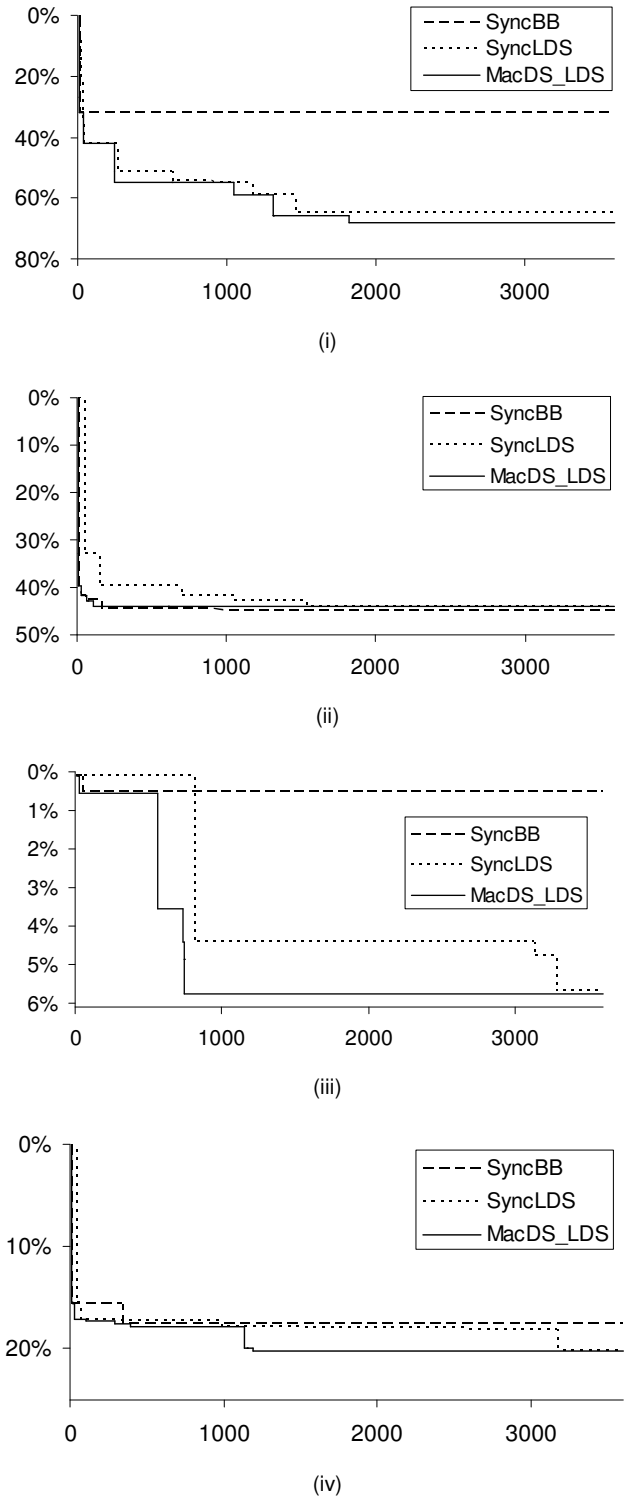


Figure 5.6 : MacDS – Réduction de la fonction objectif en fonction du temps de calcul pour les cas industriels étudiés.

5.4.2 Évaluation avec des données synthétiques

Nous avons ensuite évalué MacDS pour un large éventail de problèmes générés.

La littérature nous apprend que la performance de la stratégie LDS varie en fonction du niveau de justesse de l'hypothèse qui sous-tend LDS (i.e. qu'il existe une corrélation entre la qualité de la feuille et le nombre de déviations). Conséquemment, nous avons évalué MacDS en utilisant des arbres générés de manière à ce qu'ils rencontrent plus ou moins bien cette propriété; certains étaient complètement aléatoires alors que d'autres offraient une bonne corrélation.

Le modèle utilisé pour générer ces arbres est le suivant. Nous avons défini un paramètre δ qui définit le niveau de corrélation désiré. Il peut prendre une valeur dans l'intervalle $]0.0, \ldots, 1.0]$. Pour un arbre donné, la probabilité qu'une feuille spécifique soit la solution optimale est proportionnelle à $\delta^{(p)}$, où p est égale au nombre de déviations associé à la feuille. Lorsque δ est maximal (1.0) l'arbre est complètement aléatoire; chaque feuille a la même probabilité de renfermer la solution optimale. Plus sa valeur est petite, meilleure est la corrélation entre le nombre de déviations et la qualité de la solution. Cela permet donc d'évaluer l'algorithme sur un continuum de problèmes représentant les pires et les meilleures conditions pour l'algorithme.

Nous avons également évalué la performance de l'algorithme en fonction des paramètres suivants: le nombre de sous-problèmes (`Card(X)`), le délai nécessaire à la transmission d'un message d'un agent à l'autre (τ), le temps de résolution d'un sous-problème (α), et le nombre de solutions pour chaque sous-problème (`n`). La mesure de performance utilisée est le temps de calcul espéré pour l'atteinte de la meilleure solution.

La figure 5.7 et la figure 5.8 montrent la performance de l'algorithme en fonction de la valeur du paramètre δ. MacDS_LDS est toujours le meilleur des quatre algorithmes, à égalité avec MacDS_BT lorsque δ=1.0 (i.e. pour les arbres complètement aléatoires).

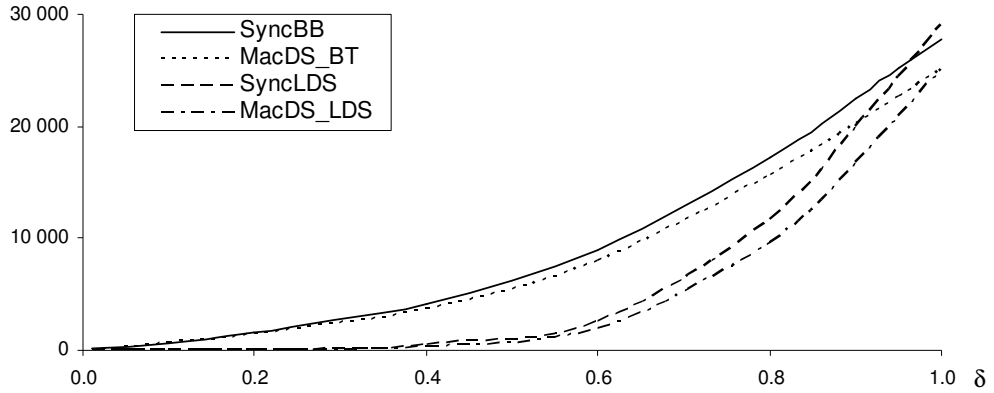


Figure 5.7: MacDS – Temps de calcul espéré pour l'atteinte de la meilleure solution en fonction du niveau de corrélation entre la qualité d'une feuille et le nombre de déviations. Résultats (en secondes) en fonction de δ [pour `Card(X)=4`; `n=10`; α=1; τ=0].

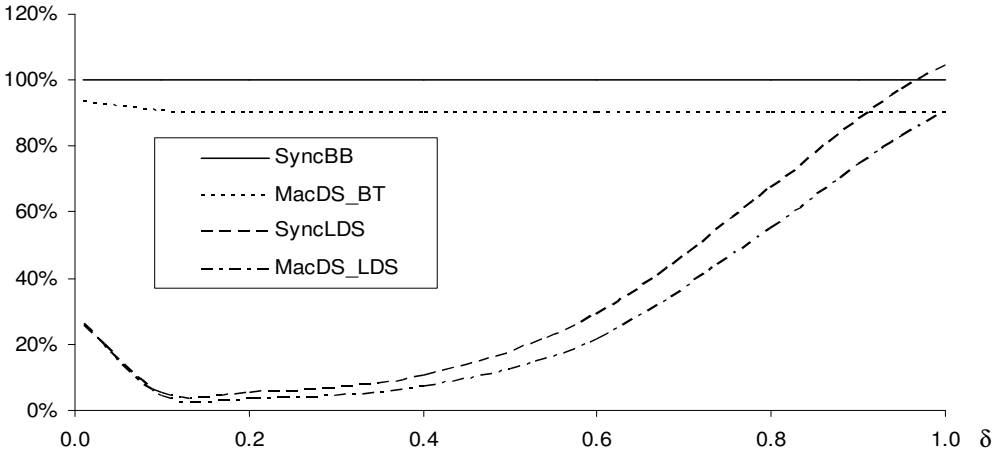


Figure 5.8: MacDS – Temps de calcul espéré pour l'atteinte de la meilleure solution (SyncBB utilisé en comme référence)

Pour une stratégie de retour-arrière donnée (BT ou LDS), MacDS bat toujours sa contrepartie synchrone. Si on compare entre eux les deux algorithmes synchrones, on remarque (comme d'autres l'ont montré avant nous) que la stratégie LDS perd de son

avantage sur le retour arrière chronologique BT au fur et à mesure que les arbres sont aléatoires. Pour des arbres très aléatoires, BT est même préférable à LDS (voir figure 5.8). Cependant, en permettant le travail simultané des agents, MacDS permet à la stratégie LDS de dominer la stratégie BT peu importe le niveau de stochasticité (i.e. peu importe la valeur de δ).

On observe également (figure 5.7 et figure 5.8) que pour de très petites valeurs de δ tous les algorithmes convergent. Cela s'explique par le fait que la probabilité que la première feuille de l'arbre soit la meilleure solution ($\delta^{(p=0)}$) tend vers `1.0` lorsque δ tend vers `0.0`; alors que pour les autres feuilles $\delta^{(p)}$ tend vers `0.0` puisque `p>0`.

Finalement, la figure 5.9 illustre l'impact des autres paramètres. La sous-figure (i) montre que le délai de transmission des messages a un impact linéaire pour chacun des quatre algorithmes, mais un impact beaucoup plus petit pour les deux versions de MacDS. Pour ceux-ci, le temps de calcul espéré pour atteindre la meilleure solution est linéaire par rapport à `(Card(X)-1)`τ. La sous-figure (ii) illustre le fait que l'impact du temps de résolution des sous-problèmes (α) est également linéaire, mais est encore une fois plus petit pour MacDS. La sous-figure (iii) montre l'impact exponentiel du nombre de sous-problèmes (`Card(X)`).

Ces résultats sont présentés pour δ=`0.7`; les résultats avec cette valeur sont représentatifs du comportement général observé avec d'autres valeurs.

5.5 Discussion

Nous avons évalué les algorithmes proposés dans deux contextes expérimentaux différents. Le cas industriel visait d'abord à montrer l'applicabilité de la méthode en contexte réel. Elle a permis une réduction significative des retards dans les livraisons aux clients. Bien sûr, pour un autre contexte industriel, les résultats pourraient être différents. C'est la raison pour laquelle à notre avis les résultats les plus pertinents sont ceux de la section 5.4.2. Plutôt que générer des données pour un problème spécifique,

nous avons généré directement les arbres correspondant à des espaces de coordination. Certains étaient théoriquement favorables aux approches de type LDS, d'autres beaucoup moins. Cela a permis d'évaluer MacDS et SyncLDS dans des conditions variées. Ce modèle théorique nous a permis de montrer que MacDS_LDS surpassait les autres approches même pour les cas les moins favorables *a priori*. Les bons résultats obtenus avec le cas industriel ne semblent donc pas accidentels et nous pouvons nous attendre à mesurer des gains pour d'autres problèmes industriels.

5.6 Conclusion

L'algorithme MacDS peut être configuré de manière à reproduire différentes stratégies de retour-arrière basées sur l'analyse des déviations. Dans ce chapitre, nous avons appliqué des stratégies couramment utilisées en environnement centralisé. Cependant, il existe peut-être des stratégies inédites qui seraient particulièrement indiquées en contexte distribué? En effet, peut-être certains sous-problèmes revêtent-ils plus d'importance que d'autres? Conviendrait-il alors d'en tenir compte dans le calcul des déviations? Le prochain chapitre aborde cette question.

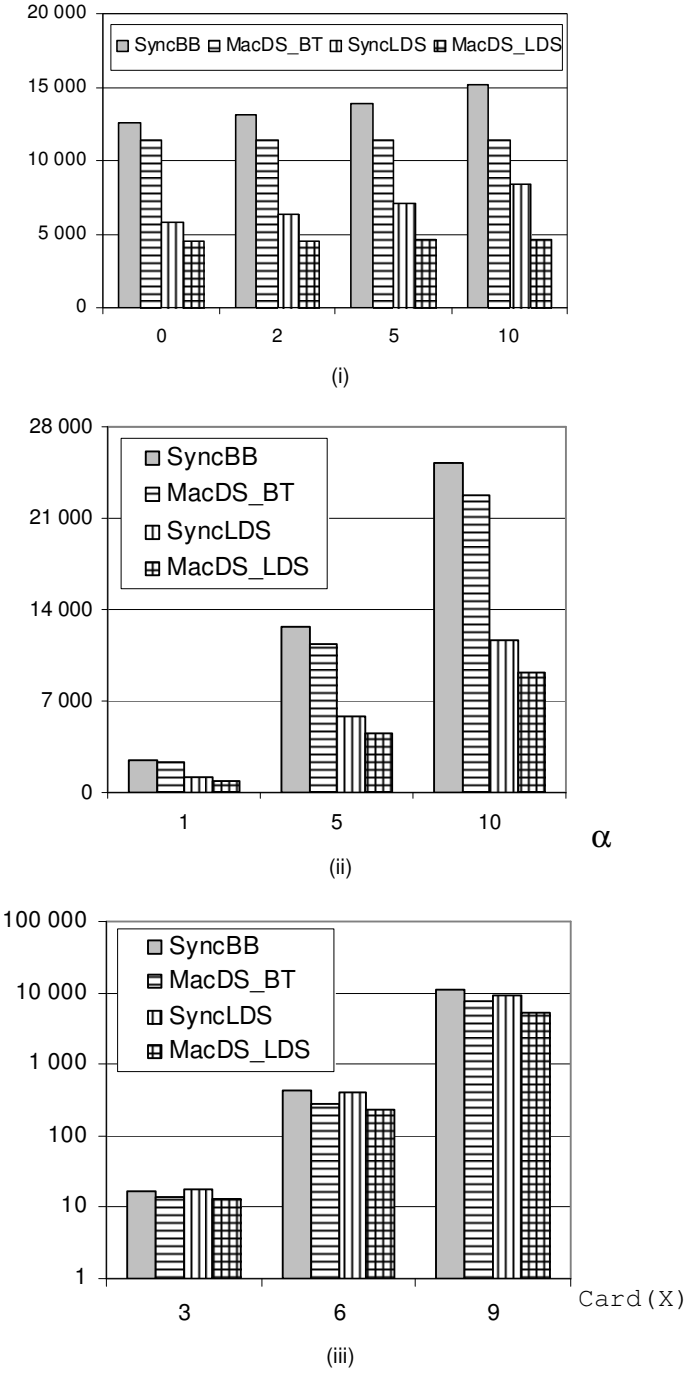


Figure 5.9: MacDS – Temps de calcul espéré pour l'atteinte de la meilleure solution en fonction de la valeur de différents paramètres environnementaux.

Résultats en fonction de: (i) délai de transmission des messages [avec `Card(X)=4`; `n=10`; α=1; δ=0.7], (ii) temps de résolution des sous-problèmes [avec `Card(X)=4`; `n=10`; τ=0; δ=0.7] et (iii) nombre total de sous-problèmes [avec `n=3`; α=1; τ=0; δ=0.7].

Chapitre 6 Adaptive Discrepancy Search (ADS) : une stratégie de recherche adaptative

Au chapitre précédent, nous avons vu que l'utilisation d'une stratégie de retour-arrière de type LDS (plutôt qu'un retour-arrière chronologique) donne de bons résultats notamment parce que cela permet de visiter rapidement différentes zones de l'espace de coordination.

Cela nous amène à poser l'hypothèse suivante : s'il était possible d'identifier dynamiquement les zones les plus intéressantes de l'arbre et d'y concentrer nos efforts, l'efficacité du processus de recherche s'en trouverait améliorée. Ce serait donc dire (1) que le nombre de déviations seul n'explique pas la répartition dans l'arbre des solutions de qualité, (2) qu'il y a dans chaque instance de problème une autre structure qui l'explique en partie, et (3) qu'il est possible de découvrir cette structure pendant la recherche.

Dans ce chapitre, nous proposons donc une stratégie de retour-arrière dite *adaptative* (ADS) : lors du processus de recherche, les agents identifient collectivement et dynamiquement les zones de l'arbre les plus prometteuses de manière à les explorer de manière prioritaire. L'objectif est d'obtenir de bonnes solutions en un temps très court tout en permettant l'exploration complète de l'espace de coordination (i.e. en maintenant l'espoir de trouver la solution optimale).

La stratégie prend appui sur le fait que notre arbre est non-binaire. Puisque chaque nœud (instance de sous-problème) comporte plusieurs arcs sortant (solution locale), nous aurons l'occasion de retraiter à plusieurs reprises vers chacun des nœuds. Chaque fois, on mesurera la conséquence d'avoir généré une énième solution locale pour son sous-problème (i.e. la conséquence d'avoir permis une énième *déviation* supplémentaire). La stratégie proposée cherche à établir dynamiquement pour quel nœud il est le plus payant de produire des solutions locales supplémentaires et combien on devrait en générer avant

de passer à un autre nœud. De plus, on cherche à établir un équilibre entre le fait de canaliser nos efforts sur certains nœuds, versus découvrir de nouvelles opportunités ailleurs dans l'arbre.

Les concepts de base sont d'abord introduits de manière informelle à la section 6.1. On présente ensuite un modèle qui permet d'anticiper la contribution associée à la réalisation d'une *déviation* supplémentaire dans un nœud donné (section 6.2). Ensuite, on présente une stratégie de retour-arrière exploitant cette information (section 6.3) et un protocole permettant de l'implémenter de manière décentralisée (section 6.4). Des expérimentations seront réalisées à la section 6.5. Finalement la section 6.6 situe notre approche par rapport à d'autres qui utilisent l'apprentissage, mais dans le cadre de la résolution de problèmes combinatoires classiques.

6.1 Idée générale

Soit un problème fictif de minimisation représenté par un arbre non-binaire. La figure 6.1(i) montre la portion visitée de cet arbre après l'obtention d'une première solution globale. Nous supposerons que la qualité de cette solution est de 1.0. À ce moment, trois nœuds sont candidats au retour-arrière (**a**, **b** et **c**). Supposons que nous appliquions une politique de retour-arrière de type LDS pour choisir notre candidat. Pour chacun de ces nœuds, la prochaine feuille (solution globale) issue de ce nœud aurait un nombre total de déviations égal à 1. Si nous désirions appliquer une politique de retour-arrière de type LDS, nous n'aurions de préférence pour aucun de ces nœuds. Nous allons donc réaliser tour à tour un retour-arrière vers chacun d'eux. On obtiendra alors trois nouvelles solutions globales, que nous supposerons de qualité respective 1.2, 0.6 et 0.9 (sous-figure ii).

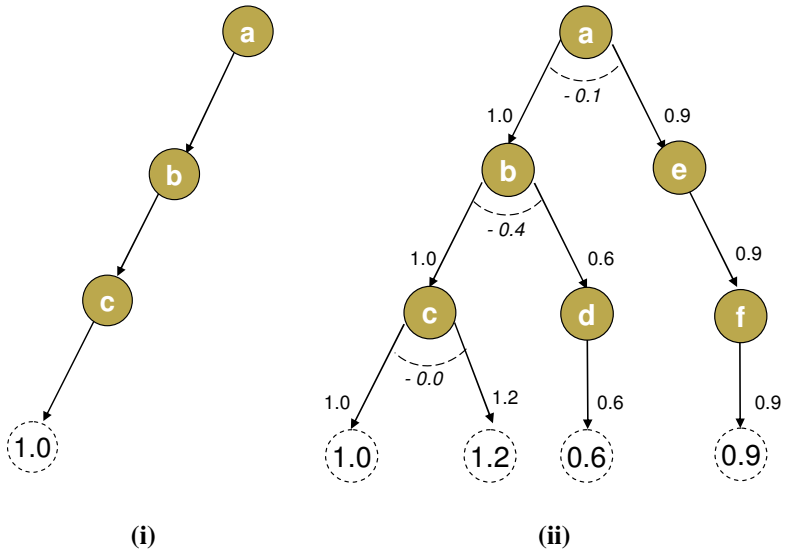


Figure 6.1 : ADS – Exemple d'arbre de recherche.

Nous avons maintenant 6 nœuds (**a** à **f**) qui sont candidats au retour-arrière[30]. Encore une fois, une politique de type LDS mettrait ces nœuds sur le même pied (chacun engendrerait une feuille ayant un nombre total de déviations égal à 2). Cependant, on remarque que pour le nœud **c**, son deuxième arc a engendré une feuille moins bonne que la première (1.2 > 1.0). Pour le nœud **b**, son deuxième arc a engendré une amélioration de 0.4 (0.6 – 1.0 = 0.4). Pour le nœud **a** l'amélioration a été de 0.1 (1.0 – 0.9 = 0.1).

Bref, alors que LDS met les nœuds **a**, **b** et **c** sur le même pied, il nous semble plus intéressant de générer d'abord une troisième solution alternative pour **b**. Quant aux nœuds **d**, **e** et **f**, générer une solution alternative pour un de ces nœuds pourrait-il se révéler intéressant? Nous n'en savons rien pour l'instant. Les prochaines sections généralisent cette idée.

[30] Les nœuds **a**, **b** et **c** sont toujours candidats puisque l'arbre est non-binaire (il y a plus de deux solutions pour chaque sous-problème).

6.2 Anticiper la contribution associée à la réalisation d'une déviation supplémentaire

Supposons qu'une recherche dans un arbre est en cours. Une énième solution globale vient d'être trouvée et on désire procéder à un retour-arrière. Soit `nodeList` la liste des nœuds `node` qui sont candidats au retour-arrière. Pour un `node` ∈ `nodeList`, nous dénoterons par `n` le nombre le solutions locales déjà générées pour son sous-problème (i.e. le nombre d'arcs émergents de ce nœud qui ont été explorés jusqu'à maintenant). Toujours dans le contexte de ce nœud `node`, nous définirons les éléments suivants :

Définition 6.1 : `Child ( )`. Nous dénoterons par `Child (i)` le nœud pointé par l'`i`-ème arc émergent du nœud `node` (défini pour `i = 0...n – 1`). `Child (i)` peut être une feuille ou un nœud interne. À titre d'exemple, pour le nœud **b** de la figure 6.1ii, nous avons `n = 2`, `Child (0)` correspond au nœud **c** et `Child (1)` au nœud **d**.

Définition 6.2 : `Next Leaf ( )`. Nous dénoterons par `NextLeaf (i)` la feuille correspondant à la première solution globale qui serait obtenue si on réalisant une descente en profondeur dans le sous-arbre dont la racine est `Child (i)`. Si `Child (i)` est une feuille, alors `NextLeaf (i)` retourne cette feuille. Sinon, nous avons :

`Leaf (i) := Child (i) . NextLeaf (0).`

Définition 6.3 : `ArcValue ( )`. Nous définirons `ArcValue (i)` comme étant égal à la qualité de la première solution globale qui serait obtenue si on réalisait une descente en profondeur dans le sous-arbre : `ArcValue (i) := NextLeaf (i) . score`. À titre d'exemple, sur la figure 6.1ii chaque arc *i* issu d'un nœud est étiqueté de sa valeur `ArcValue (i)`.

Certains pourraient proposer d'autres définitions pour `ArcValue(i)`, par exemple « la qualité de la meilleure solution dans le sous-arbre correspondant trouvée jusqu'à maintenant ». Cependant, la définition que nous proposons a l'avantage suivant. Chaque fois que l'on retraite vers un nœud, on peut mesurer la qualité de l'arc suivi aussitôt qu'on atteint la prochaine feuille (i.e. avant le prochain retour-arrière) et cette valeur demeure inchangée pour le reste de la recherche, ce qui limite les mises à jour dans le modèle.

Définition 6.4 : `bestToDate[]`. Il s'agit d'un vecteur de taille `n`, tel que $\texttt{bestToDate}[i] = \min_{j=0}^{i} \texttt{ArcValue}(i)$.

La figure 6.2 illustre la relation entre `ArcValue( )` et `bestToDate[ ]`. Cet exemple présente un nœud pour lequel `n = 4` (sous-figure i). Chaque arc `i = 0...3` a pour étiquette sa valeur `ArcValue(i)`. La sous-figure ii illustre la difficulté apparente d'extrapoler une valeur pour `ArcValue(4)`. À l'opposé, `bestToDate[ ]`est monotone et décroissant en fonction de `i` (sous-figure iii). On peut plus facilement imaginer utiliser les points existants afin d'extrapoler une valeur pour `bestToDate[4]`.

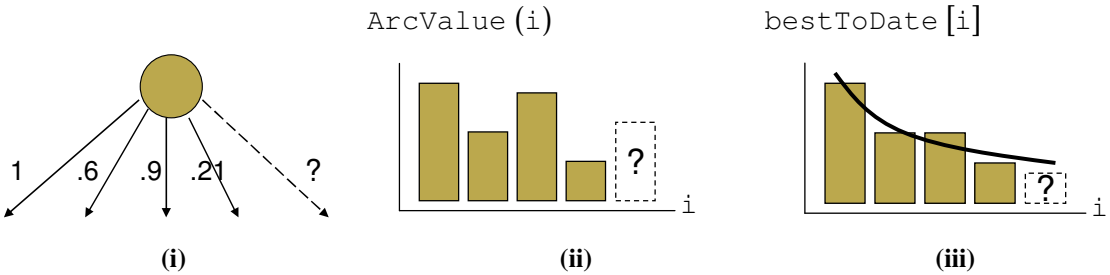


Figure 6.2 : ADS – Relation entre `ArcValue( )` et `bestToDate[ ]`.

Définition 6.5 : `F ( )`**.** Nous supposerons l'existence d'une fonction continue $\texttt{F}\left(\mathbb{R}^{+}\right) \rightarrow \mathbb{R}^{+}$ qui approxime `bestToDate [ ]`. Elle sera générée par un algorithme `A ( )`, à partir de l'ensemble des valeurs déjà connues pour `bestToDate [ ]`. Nous avons donc :

$$\texttt{A}\left(\texttt{bestToDate}\left[0\ldots n-1\right]\right) \rightarrow \left(\texttt{F}\left(\mathbb{R}^{+}\right) \rightarrow \mathbb{R}^{+}\right).$$

Notre principale préoccupation n'est pas de trouver une fonction qui explique bien les points déjà connus, mais plutôt d'extrapoler un nouveau point en dehors de la plage déjà connue. Ce problème (extrapolation) présente un défi supplémentaire par rapport à celui de l'interpolation (générer des points intérieurs) [69]. Compte tenu de la définition donnée précédemment pour `bestToDate [ ]`, l'algorithme `A ( )` peut se limiter à considérer des fonctions monotones décroissantes.

Par exemple, si on pose l'hypothèse que `bestToDate [i]` décroit entre sa valeur initiale $\left(\texttt{ArcValue}\left(0\right)\right)$ selon un taux constant β jusqu'à une limite inférieure α, l'algorithme `A ( )` doit alors produire une fonction de la forme suivante :

$$\texttt{F}\left(\texttt{i}\right) := \left(\texttt{ArcValue}\left(0\right) - \alpha\right) e^{-\beta \texttt{i}} + \alpha.$$

La figure 6.3 illustre cette situation. La tâche de l'algorithme `A ( )` est alors d'estimer la valeur des paramètres α et β. Cela pourrait être fait, par exemple, en réalisant un ajustement de courbe avec l'algorithme Gauss-Newton ou l'algorithme Levenberg-Marquardt [70,71].

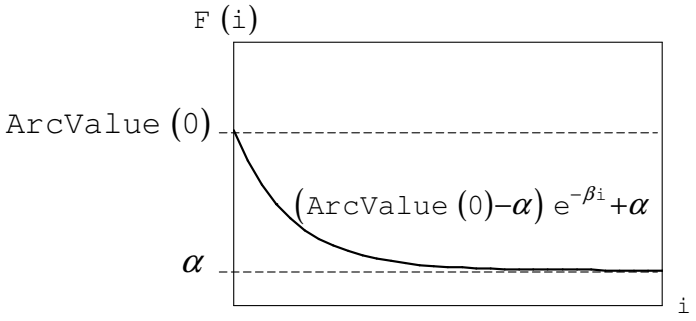


Figure 6.3 : ADS – Fonction d'extrapolation `F (i)` proposée.

Définition 6.6 : `improvement [ ]`**.** Soit un nœud ayant engendré deux solutions locales consécutives `i – 1` et `i`. Ces deux arcs ont engendré des feuilles de qualité `arcValue [i – 1]` et `arcValue [i]`. La valeur de `improvement [i]` correspond à l'amélioration engendrée au niveau de `bestToDate [ ]` :

$$\texttt{improvement}\,[i] = \texttt{bestToDate}\,[i-1] - \texttt{bestToDate}\,[i]$$

Définition 6.7 : `Improvement ( )`. De manière similaire, nous définirons la fonction `Improvement ( )`, qui est l'amélioration attendue associée à la génération d'une énième solution locale pour un sous-problème donné, en vertu de notre estimateur `F ( )` pour `bestToDate [ ]` :

$$\texttt{Improvement}\,(i) := \texttt{F}\,(i-1) - \texttt{F}\,(i)$$

Pour un nœud *node* pour lequel *n* solutions locales ont déjà été générées, on s'intéressera particulièrement à la valeur `Improvement (n)`. Elle nous indique si une solution locale supplémentaire (arc) devrait engendrer une solution globale (i.e. `NextLeaf (n)`) meilleure que celles associées aux arcs précédents, et à quel point elle devrait l'être. Afin de simplifier l'écriture nous écrirons `Improvement ( )` en lieu et place de `Improvement (n)` pour la suite de ce texte.

6.3 Stratégie de retour-arrière exploitant le modèle

La stratégie proposée peut être décrite comme suit. La première solution globale est obtenue comme pour SyncBB ou SyncLDS, c'est-à-dire en résolvant séquentiellement les sous-problèmes. Pour les premiers retour-arrières, on procède au choix du candidat en vertu d'une politique LDS. Après avoir retraité au moins une fois à chacun des niveaux de l'arbre, on peut commencer à utiliser le modèle pour choisir le prochain candidat au retour-arrière. Les nœuds seront comparés sur la base de leurs valeurs `Improvement ( )` respectives[31].

On rappelle que pour pouvoir calculer la valeur de `Improvement ( )` d'un nœud donné, au moins 2 de ses arcs doivent avoir été explorés. En conséquence, seuls les nœuds pour lesquels cette condition est rencontrée sont considérés. On suppose également l'existence d'un seuil ε au-delà duquel l'amélioration attendue est jugée significative dans le contexte du domaine d'application. En deçà de ε, on suppose qu'il est préférable d'explorer d'autres nœuds. En pratique on pourra simplement utiliser ε=0.

Lorsque trop peu de nœuds (moins de 2) se qualifient en vertu du paragraphe précédent, on utilise alors le même critère que LDS pour choisir le candidat au retour-arrière. Ce faisant, on augmente le nombre de nœuds qui se qualifient et on ouvre la porte à la découverte de nouveaux nœuds intéressants.

Aucun nœud (même s'il est jugé inintéressant par notre modèle) n'est abandonné à tout jamais. L'exploration de son prochain arc est seulement reportée tant que d'autres nœuds semblent plus intéressants. L'algorithme permet donc l'exploration complète de l'arbre, tout en visant l'obtention de bonnes solutions pour des temps très courts.

---

[31] Nous utilisons `Improvement ( )` plutôt que `F ( )` comme critère de poids pour la raison suivante. Si nous utilisions `F ( )`, un nœud ayant engendré une bonne solution dans le passé serait indéfiniment préféré pour la suite, même s'il se mettait à engendrer de mauvaises solutions.

Chaque fois qu'une nouvelle solution au problème global est obtenue, le modèle est mis à jour. Cela modifie dynamiquement la priorité associée aux nœuds, d'où le caractère adaptatif de la stratégie. La section 6.3.1 présente le pseudocode nécessaire à cette mise à jour. La section 6.3.2 présente le pseudocode permettant de choisir le candidat au retour-arrière en vertu de la stratégie proposée.

6.3.1 Mise à jour du modèle

Supposons qu'on vienne d'obtenir une solution globale (feuille). On doit alors mettre à jour le vecteur `bestToDate[ ]` et la fonction `F( )` pour certains des ancêtres de cette feuille.

La figure 6.4 présente le pseudocode réalisant cette mise à jour. La fonction `UpdateModel()` reçoit en paramètre la qualité de la nouvelle feuille (`score`) et son identifiant unique `p[]`. Il s'agit d'un vecteur d'entiers représentant le chemin dans l'arbre allant de la racine jusqu'à cette feuille. L'élément `p[j]` indique, pour un niveau `j`, quel arc devrait alors être suivi. La fonction `Card()` retourne la taille de ce vecteur. Finalement, on rappelle que `nodeList` contient tous les nœuds candidats au retour-arrière. Chaque nœud `node` est défini par un tuple `<p[], bestToDate[], F>`.

La boucle principale permet de naviguer le long de `p[]`, de la feuille jusqu'à la racine de l'arbre. Pour le nœud qui est le parent immédiat de la feuille, la mise à jour est toujours réalisée. Pour les nœuds en amont, la mise à jour n'est pas toujours nécessaire.

Considérons le noeud `node` et son `i`-ème fils `child`, tels qu'ils sont tous deux sur le chemin menant à la feuille `leaf`. Si `leaf` n'est pas la première feuille dans le sous-arbre ayant sa racine à `child`, alors `leaf ≠ node`. `NextLeaf(i)` et le score de cette feuille n'intervient pas dans le calcul de `ArcValue(i)` and `bestToDate[i]` pour `node` (selon la définition 6.3 et la définition 6.4). Dans ce cas, les nœuds sur le chemin qui sont en amont de `node` n'ont pas non plus à être mis à jour.

```
Procedure UpdateModel(p[], score)
 do
 {
    i ← p[Card(p)];
    delete last element from p; // p est maintenant le chemin du parent
    node ← select node in nodeList : (node.p = p); // node est maint. le parent
    if (i = 0) node.bestToDate[0] ← score;
    else node.bestToDate[i] ← Min(node.bestToDate[i-1], score);
    node.F ← A(node.bestToDate);
 }
 while ((Card(p) > 0) and (i = 0))
```

Figure 6.4 : ADS – Pseudocode pour la mise à jour de `bestToDate [ ]` et `F ( )`.
Elle sera exécutée lors de l'atteinte d'une feuille.

6.3.2 Choix du candidat pour le retour-arrière

La figure 6.5 présente le pseudocode (voir fonction `SelectNode`). Elle permet de sélectionner le nœud pour lequel `Improvement()` est maximal. Rappelons que puisque le modèle n'est utilisable qu'après qu'un minimum d'informations aient été accumulées, les premiers retour-arrières sont réalisés en vertu d'une politique LDS. Cela est fait en donnant priorité aux nœuds pour lesquels la prochaine solution globale engendrée aurait un nombre total de déviations inférieur ou égal à 1. On applique également la politique LDS si trop peu de nœuds rencontrent le critère de sélection ADS (appliqué par `FilterADS`). Les nœuds pour lesquels `n` est égal à zéro ont également priorité puisque cela correspond à une descente normale dans l'arbre (aucun retour-arrière n'est nécessaire). Dans tous ces cas, on applique la politique LDS.

La fonction `CompareLDS` permet d'appliquer la politique LDS. Elle compare deux nœuds et retourne celui avec la plus grande priorité. Les paramètres de la fonction correspondent au chemin dans l'arbre de la prochaine solution globale que le nœud permettrait d'engendrer (i.e. la concaténation de `p[]` et `n` en tant que nouveau vecteur). En cas d'égalité, on applique l'équivalent d'un retour-arrière pour briser l'égalité (`CompareBT`).

```
Function SelectNode(nodeList)
  nodeListADS ← FilterADS(nodeList)
  if (Card(nodeListADS) < 2)
  or (∃ node in nodeList) : ((SumOfDisc(node)+node.n ≤ 1) or (node.n = 0))
     candidate ← select node in nodeList according to function CompareLDS()
  else
     candidate ← select node in nodeListADS : node.Improvement() is maximal
  return candidate

Function FilterADS(nodeList)
  return all node in nodeList : (node.n ≥ 2) and (node.Improvement() > ε)

Function SumOfDisc(node)
  return Σ(j=0..Card(node.p[])-1) node.p[j]

Function CompareLDS(p1, p2)
  t1 ← Σ(j=0..Card(p1)-1) p1[j]
  t2 ← Σ(j=0..Card(p2)-1) p2[j]
  if (t1 < t2) return p1
  else if (t2 < t1) return p2
  else return CompareBT(p1, p2)

Function CompareBT(p1, p2)
  depth ← Min(Card(p1), Card(p2))
  j ← 0
  while (p1[j] = p2[j] and j < depth) j ← j+1
  if (j < depth)
      if (p1[j] ≤ p2[j]) return p1 else return p2
  else
      if (Card(p1) ≥ Card(p2)) return p1 else return p2
```

Figure 6.5 : ADS – Pseudocode pour la sélection du candidat au retour-arrière.

6.4 Synchronous Adaptive Discrepancy-based Search (SyncADS)

Cette section présente un protocole (SyncADS) permettant d'implémenter de manière décentralisée la stratégie de retour-arrière adaptative (ADS) décrite précédemment.

Chaque agent exécute le pseudocode présenté à la figure 6.6. La fonction principale (`MsgProposition`) est déclenchée par la réception d'une proposition (solution locale en provenance de l'agent précédent). Cette proposition est dénotée par `<d, p[]>`. L'élément `d` représente les décisions pour les sous-problèmes précédents et `p[]` est le vecteur d'entiers représentant le chemin correspondant à ce nœud dans l'arbre global.

Sur réception du message, l'agent crée un nœud correspondant à ce sous-problème et l'ajoute à sa liste de nœuds locale (`nodeList`). L'agent s'attaque alors à ce sous-

problème (`Work(node)`). Il trouve une première solution à ce sous-problème, qu'il envoie en tant que proposition à l'agent suivant (`send MsgProposition`). S'il n'y a pas d'agent suivant, on a en main une solution au problème global. L'agent met alors à jour son modèle (`UpdateBestToDate`), sa fonction `F`, informe son prédécesseur de la qualité de cette solution (`send MsgQuality`) et déclenche la procédure distribuée pour le choix du nœud qui sera le candidat au retour-arrière (`Cooperative BacktrackingADS`).

Dans la fonction `CooperativeBacktrackingADS`, on demande à chaque agent d'identifier le nœud sous son autorité qu'il choisirait localement (chaque agent le sélectionne en utilisant la fonction `SelectNode` de la figure 6.5). On demande également aux agents combien de nœuds se qualifient localement en vertu du critère de filtrage ADS. Connaissant cette information, l'agent appelant peut identifier le nœud/sous-problème le plus prioritaire (le reste de cette fonction est très similaire à celui de la fonction `SelectNode` de la figure 6.5). On donne ensuite le contrôle à l'agent responsable du nœud identifié (`send MsgBacktrack(node)`). Il est cependant important de noter que les agents n'ont pas réellement à transmettre les nœuds comme nous le faisons dans le pseudocode; ils n'ont qu'à transmettre le vecteur `p[]` et la valeur `n`. Seules ces informations sont nécessaires pour identifier quel nœud est prioritaire.

```
WhenReceive MsgProposition(<d,p[]>) do
 node ← <d, p[], n=0>
 nodeList.add(node)
 Work(node)

Procedure Work(node)
 proposition ← NextSolution(node);
 if (proposition ≠ ∅)
    node.n ← node.n+1
    if (Successor(node) ≠ ∅)
       send MsgProposition(<proposition, node.p[]+[node.n-1]>)
       to Successor(node)
    else
       UpdateBestToDate(node, node.n-1, proposition.score);
       node.F ← A(node.bestToDate[])
       if (node.n = 1)
          send MsgQuality(score, node.p[]+node.n-1) to Predecessor(node)
       CooperativeBacktrackingADS()
 else
    nodeList.remove(node)
    CooperativeBacktrackingADS()

Procedure UpdateBestToDate(node, i, score)
 if (i = 0) node.bestToDate[0] ← score
 else node.bestToDate[i] ← Min(node.bestToDate[i-1], score)

WhenReceive MsgQuality(score, p[]) do
 node ← select node in nodeList : p[] begins with node.p[]
 i ← p[Card(node.p)-1]
 UpdateBestToDate(node, i, score)
 if (i = 0) and (Predecessor(node) ≠ ∅)
     send MsgQuality(score, p[]) to Predecessor(node)

Procedure CooperativeBacktrackingADS()
 send MsgAskNbADSQualifiedNodes to Everybody       // including itself
 nbQualifiedNodesADS ← Σ answer from Everybody
 send MsgAskBestLocalNode() to EveryBody
 answers ← all answer from EveryBody : (answer ≠ ∅)
 if (nbQualifiedNodesADS < 2)
 or (∃ node in answers) : ((SumOfDisc(node)+node.n ≤ 1) or (node.n = 0))
    candidate ← select node in answers[] according to function CompareLDS()
 else
    candidate ← select node in answers[] : node.Improvement() is maximal
 send MsgBacktrack(node) to Agent(answer)

WhenReceive MsgAskNbADSQualifiedNodes() do return Card(FilterADS(nodeList))

WhenReceive MsgAskBestLocalNode() do return SelectNode(nodeList)

WhenReceive MsgBacktrack(node) do Work(node)
```

Figure 6.6 : SyncADS – Pseudocode.

6.5 Expérimentations

Nous évaluerons d'abord l'approche proposée avec les données du cas industriel. Nous évaluerons les gains associés à l'utilisation de la stratégie proposée, de même que la qualité du modèle prédictif. Par la suite, nous évaluerons l'algorithme pour des problèmes générés de manière à généraliser les résultats obtenus.

6.5.1 Évaluation avec les données industrielles

Nous avons vu au chapitre précédent que même pour de très grand temps de calcul, SyncBB ne fait qu'explorer des variations mineures de la première solution globale obtenue. La qualité des solutions trouvées cesse rapidement de s'améliorer. À l'opposé une stratégie de type LDS permet de mieux « échantillonner » l'espace des solutions. La qualité des solutions trouvées continue de s'améliorer (avant de finalement atteindre un plateau un peu avant une heure). Dans ce qui suit, nous supposerons donc que la partie de l'arbre explorée pendant cette période est un échantillon représentatif de l'espace des solutions; nous évaluerons notre nouvel algorithme en utilisant ces mêmes arbres.

Contrairement aux expériences du chapitre précédent, nous travaillons donc avec des arbres issus d'un prétraitement. Ce faisant, nous ramenons artificiellement à zéro le temps nécessaire à la production de chaque solution locale (et donc, au développement d'un arc). Cela nous a permis de réaliser une série d'expériences qui auraient demandé des milliers d'heures de temps de calcul dans la configuration expérimentale du chapitre précédent[32]. De plus, cela permet de comparer l'impact des stratégies de retour-arrière ADS et LDS en utilisant SyncADS et SyncLDS, mais en simulant en quelque sorte que les agents travaillent simultanément: chaque fois qu'en vertu de notre politique de retour-arrière nous devons passer à la prochaine solution locale d'un nœud donné pour un autre agent, celle-ci est déjà prête.

---

[32] On fait référence aux résultats rapportés dans ce chapitre, de même qu'à d'autres pistes expérimentales explorées en vainc.

Dans les expérience qui suivent, nous évaluons également l'impact de différents modèles prédictifs $\mathtt{F}(\,)$ pouvant être utilisés pour l'extrapolation de $\mathtt{bestToDate}[\,]$. D'abord (1), le modèle introduit à la section 6.2 et qui demande l'apprentissage de deux paramètres α et β. Rappelons qu'il suppose que $\mathtt{bestToDate}[\,]$ décroît selon un taux constant β jusqu'à l'atteinte d'un plateau α. On a alors :

$$\mathtt{F}(\mathtt{i}) := (\mathtt{ArcValue}(0) - \alpha)\ e^{-\beta \mathtt{i}} + \alpha.$$

Ensuite (2), un modèle simplifié avec un seul paramètre où l'on suppose que $\mathtt{bestToDate}[\,]$ décroît jusqu'à 0. On a donc :

$$\mathtt{F}(\mathtt{i}) := \mathtt{ArcValue}(0)\ e^{-\beta \mathtt{i}}.$$

À titre de contre-exemple, nous avons également testé (3) une fonction polynomiale :

$$\mathtt{F}(\mathtt{i}) := \gamma \mathtt{i}^2 + \lambda \mathtt{i} + \varphi.$$

Bien que ce modèle devrait offrir une bonne capacité d'interpolation (i.e. il devrait bien modéliser la courbe des points déjà connus), il est attendu que modèle ne devrait pas donner de bons résultats en ce qui a trait à sa capacité d'extrapolation pour les points suivants (la fonction sur laquelle s'appuie le modèle n'est pas strictement descendante).

Quant au deuxième modèle, son avantage sur le premier (advenant qu'il permette d'obtenir de bons résultats) serait que l'algorithme d'apprentissage n'aurait qu'un seul paramètre à estimer et qu'il pourrait le faire à l'aide d'une simple régression linéaire. Cependant, dans les expérimentations qui suivent, nous réaliserons les mises à jour des paramètres pour tous les modèles avec l'algorithme Levenberg-Marquardt[33]. Celui-ci est réputé avoir l'avantage suivant sur l'algorithme Gauss-Newton : bien que pour ces deux

[33] Nous utilisons l'implémentation « Levenberg-Marquardt.NET » réalisée par Kris Kniaz. Voir http://kniaz.net

algorithmes on doive fournir des valeurs d'initialisation pour les paramètres à estimer, la performance du premier est réputée moins dépendante des valeurs choisies. Dans nos expérimentations, nous avons utilisé les valeurs d'initialisations suivantes : $\beta = 0.5$ et $\alpha = 0$.

Résultats

Le tableau 6.1 compare les temps nécessaires pour obtenir la meilleure solution. On présente la réduction de temps de calcul (en %) permise par SyncADS (par rapport à SyncLDS) pour les quatre cas industriels étudiés. Les résultats sont présentés pour chaque configuration de SyncADS.

Le premier modèle surpasse le second pour chacun des cas étudiés. Il permet une réduction moyenne de **48.3%** alors qu'elle est de **44.5%** pour le deuxième modèle. Quant au troisième modèle, nous obtenons sans surprise de piètres résultats.

Tableau 6.1 : ADS – Réduction (en %) du temps de calcul requis pour l'obtention de la meilleure solution pour les cas industriels étudiés.

Modèle	Cas #1	Cas #2	Cas #3	Cas #4	Moyenne
1. $F(i) := (\text{ArcValue}(0) - \alpha)\ e^{-\beta i} + \alpha$	47.6%	36.2%	55.2%	54.2%	48.3%
2. $F(i) := \text{ArcValue}(0)\ e^{-\beta i}$	42.9%	30.9%	50.2%	54.2%	44.5%
3. $F(i) := \gamma i^2 + \lambda i + \varphi$	-17.5%	-3.9%	-7.8%	-4.2%	-8.4%

Le tableau 6.2 présente la réduction moyenne du temps nécessaire à l'obtention de solutions de qualité intermédiaire. Cette métrique vise à vérifier que SyncADS permet d'obtenir des solutions de qualité intermédiaire en un temps égal ou inférieur à celui de

SyncLDS[34]. Pour les modèles 1 et 2, les résultats sont moins bons que pour l'indicateur précédent (i.e. temps pour obtenir la solution optimale). En effet, moins on cherche une solution de qualité, plus l'écart entre SyncADS et SyncLDS rétrécit. L'explication est la suivante : moins on désire une bonne solution, plus vite elle peut être obtenue par chacun des algorithmes, et moins SyncADS aura le temps d'apprendre et de se démarquer. Nous avons même un résultat (cas #2 avec le modèle 1) pour lequel SyncADS a été moins rapide en moyenne de **7.7%** (alors qu'il est plus rapide de **36.2%** pour trouver la solution optimale). Nous étudierons avec plus de détail cette relation entre la performance d'ADS et le temps de calcul à la section 6.5.2.

Pour le modèle 3, nous ne remarquons aucune tendance claire. Il ne semble y avoir aucune corrélation entre ses résultats pour le premier indicateur et pour le deuxième. Notons seulement encore une fois sa piètre performance globale. À notre avis cela s'explique par la mauvaise capacité prédictive du modèle.

Tableau 6.2 : ADS – Réduction moyenne du temps de calcul (en %)
pour l'obtention de solutions de qualité égale ou supérieure
à celles obtenues avec la stratégie de retour-arrière LDS.

Modèle	Cas #1	Cas #2	Cas #3	Cas #4	Moyenne
1. $F(i) := (\mathrm{ArcValue}(0) - \alpha)\ e^{-\beta i} + \alpha$	28.7%	-7.7%	46.6%	44.3%	28.0%
2. $F(i) := \mathrm{ArcValue}(0)\ e^{-\beta i}$	24.0%	12.7%	42.9%	43.6%	30.8%
3. $F(i) := \gamma i^2 + \lambda i + \varphi$	26.6%	-6.3%	23.4%	-5.4%	9.6%

[34] Cette métrique est calculée comme suit. Considérons l'exécution de SyncLDS. La qualité de la « meilleure solution au problème global trouvée jusqu'à maintenant » évolue dans le temps. Pour tout niveau de qualité atteint par SyncLDS, nous évaluerons le temps qui est nécessaire à SyncADS pour atteindre une solution de qualité égale ou meilleure.

Qualité des modèles

Nous avons vu à la section précédente que SyncADS a permis de meilleurs résultats que SyncLDS pour le problème industriel. À notre avis, cela s'expliquerait de deux façons. D'abord (1), les modèles proposés pour `F ( )` modélisent bien le vecteur `bestToDate [ ]` et permettent une bonne extrapolation des prochaines valeurs. Ensuite (2), la courbe/profil du vecteur `bestToDate [ ]` est relativement différente d'un nœud à l'autre. En effet, si elle était identique pour tous les nœuds, choisir le candidat au retour-arrière sur la base de celui-ci serait futile; une stratégie LDS donnerait alors des résultats équivalents.

Pour vérifier l'hypothèse 1, nous avons mesuré les écarts entre les prévisions des modèles (i.e. tous les appels à `F (i)` réalisés lors de l'exécution de l'algorithme) et les valeurs `bestToDate [i]` mesurées *a posteriori*. Nous obtenons les résultats suivants : le modèle 1 produit une erreur moyenne[35] de **1.6 %,** le modèle 2 de **3.6 %** et le modèle 3 de **9.6 %**. Notons que le modèle fournissant la meilleure anticipation est aussi celui qui a fourni les meilleures performances.

Pour vérifier l'hypothèse 2, nous avons procédé comme suit. Nous avons cherché à caractériser chaque nœud à l'aide d'une valeur β. Pour chaque nœud, nous avons donc pris les valeurs `bestToDate [ ]` (connues après la recherche) et calculé la meilleure valeur β épousant ces points.

La figure 6.7 présente la distribution des β parmi les nœuds des arbres. La courbe de tendance nous montre que cette distribution se rapproche d'une exponentielle $\left(e^{-12.718x}\right)$ définie sur l'intervalle $[0, \ldots, 0.5]$. Rappelons que les nœuds pour lequels β se

---

[35] L'erreur moyenne est calculée comme suit : $\sum_i \frac{\left|\texttt{bestToDate}[i] - \texttt{F}(i)\right|}{\texttt{bestToDate}[i]}$

rapproche de zéro sont ceux pour lesquels il s'est avéré non payant de générer des solutions alternatives (ceux-ci sont donc très nombreux dans notre problème industriel) alors que ceux avec un β élevés sont les nœuds les plus prometteurs (ceux sur lesquels la stratégie cherche à s'attarder en priorité).

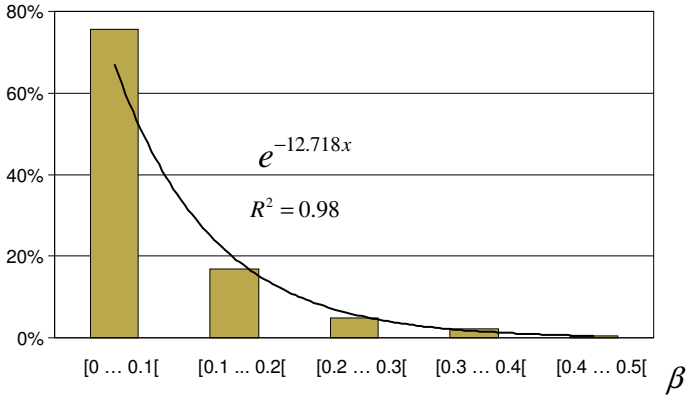


Figure 6.7 : Distribution des valeurs β dans les nœuds des arbres correspondant aux problèmes industriels étudiés.

De même, pour chaque nœud, nous avons observé la dernière valeur de `bestToDate[ ]` de manière à estimer α. Dans notre population de nœuds, α est distribué plus ou moins uniformément entre `ArcValue(0)` et `0.5 × ArcValue(0)`.

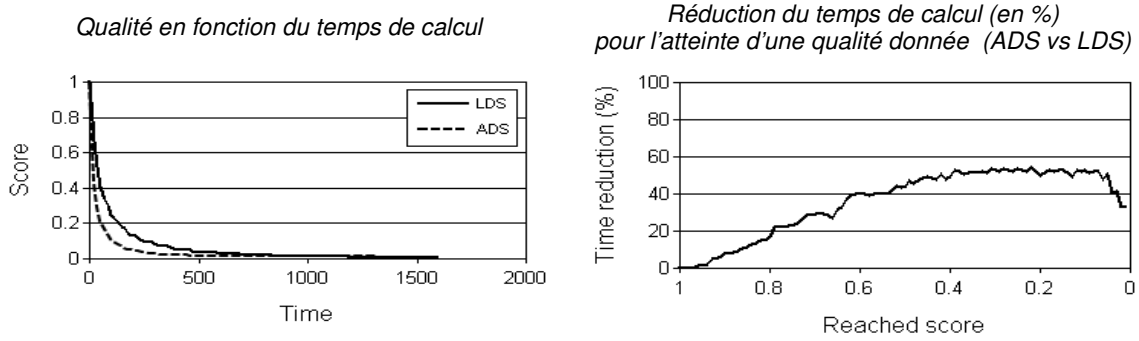
6.5.2 Généralisation – Évaluation avec des données synthétiques

Nous avons vu à la section précédente que, pour les arbres correspondants à nos problèmes industriels, il y a certains nœuds sur lesquels il vaut davantage la peine de s'attarder. Ce sont les nœuds pour lesquels nous avons mesuré de grands β. On peut supposer que la performance de notre approche est reliée à la distribution de ces β dans l'arbre. Dans cette section, nous générerons d'autres ensembles de données, de manière à étudier cette caractéristique de l'algorithme. Nous étudierons également l'impact du paramètre δ et l'impact de celui du nombre de sous-problèmes (i.e. la profondeur de l'arbre).

Nous supposerons des problèmes de minimisation tels que la première feuille de l'arbre correspond à une solution de qualité égale à 1. Pour chaque nœud, nous choisirons une valeur aléatoire pour le paramètre β à partir d'une distribution $\mathtt{PR}(\beta = \mathtt{x}) \coloneqq \mathtt{e}^{-\gamma_{\mathtt{x}}}$. Pour nos problèmes industriels, nous avions mesuré $\gamma = -12.718$ mais nous essaierons également d'autres valeurs. De la même manière, les valeurs α seront tirées d'une distribution uniforme entre 0 et $\delta \times \mathtt{arcValue}[0]$. Connaissant les valeurs de β et α de chaque nœud, nous établirons les valeurs des autres feuilles en utilisant le modèle 1. Les arbres seront générés dynamiquement pendant le processus de recherche. Le nombre total de nœud dépend donc du temps alloué à la recherche.

Pour introduire notre cadre d'évaluation, considérons d'abord le cas suivant. Il s'agit d'arbres de profondeur 4, générés avec $\gamma = 10$ (en supposant $0.0 \leq \beta \leq 0.5$) et $\delta = 0$. Nous avons choisi ces arbres peu profonds car ils nous permettent d'atteindre de très bonnes solutions en un temps raisonnable. La figure 6.8(i) présente la qualité de la meilleure solution obtenue en fonction du temps de calcul (nombre de nœuds visités) pour SyncLDS et SyncADS. On remarque que pour des temps très grands, les deux permettent de très bonnes solutions. Cependant, SyncADS atteint plus rapidement des solutions de bonne qualité. La sous-figure (ii) présente la réduction de temps de calcul permise par SyncADS, pour une qualité de solution égale à SyncLDS. Par exemple, l'obtention d'une solution de score 0.4 (i.e. une valeur pour la fonction objectif **60%** moindre que la première solution obtenue[36]) prend environ **50%** moins de temps à obtenir avec SyncADS. On remarque également une réduction relative des performances de SyncADS pour l'obtention de solutions avec un score très près de 0. C'est que, pour les deux méthodes, le score tend vers zéro pour des temps très grands (sous-figure i) et l'écart relatif tend à s'amenuiser.

[36] Est-il nécessaire de rappeler que la première solution de LDS et ADS est toujours la même?



(i) (ii)

Figure 6.8 : Comparaison des stratégies ADS et LDS pour un arbre de petite taille.
Résultats pour un arbre de profondeur 4 avec [`0.0` $\leq \beta \leq$ `0.5`; γ=10; δ=0].

L'expérience suivante (figure 6.9) montre l'impact de la profondeur de l'arbre (dénoté par `Card(X)`, par soucis d'unité avec la notation des chapitres précédents). Les sous-figures (i) et (ii) montrent la qualité de la solution en fonction du temps pour SyncLDS et SyncADS, et cela pour les profondeurs `50`, `100` et `150`. Deux détails attirent notre attention. Premièrement, avec ADS l'amélioration de la qualité de la solution en fonction du temps est davantage graduelle et continue. Deuxièmement, avec ADS on remarque qu'au début de la recherche il y a apparence de plateau (particulièrement évident pour la profondeur `150`). Cela correspond à la phase d'initialisation de notre algorithme où le retour-arrière s'effectue comme le ferait LDS. Plus la profondeur de l'arbre est importante, plus cette période est longue. Finalement, la sous-figure (iii) montre la réduction du temps de calcul permis par ADS. Par exemple, l'obtention d'une solution de score 0.4 prendra environ **80%** moins de temps en utilisant la stratégie ADS (arbre de profondeur `50`).

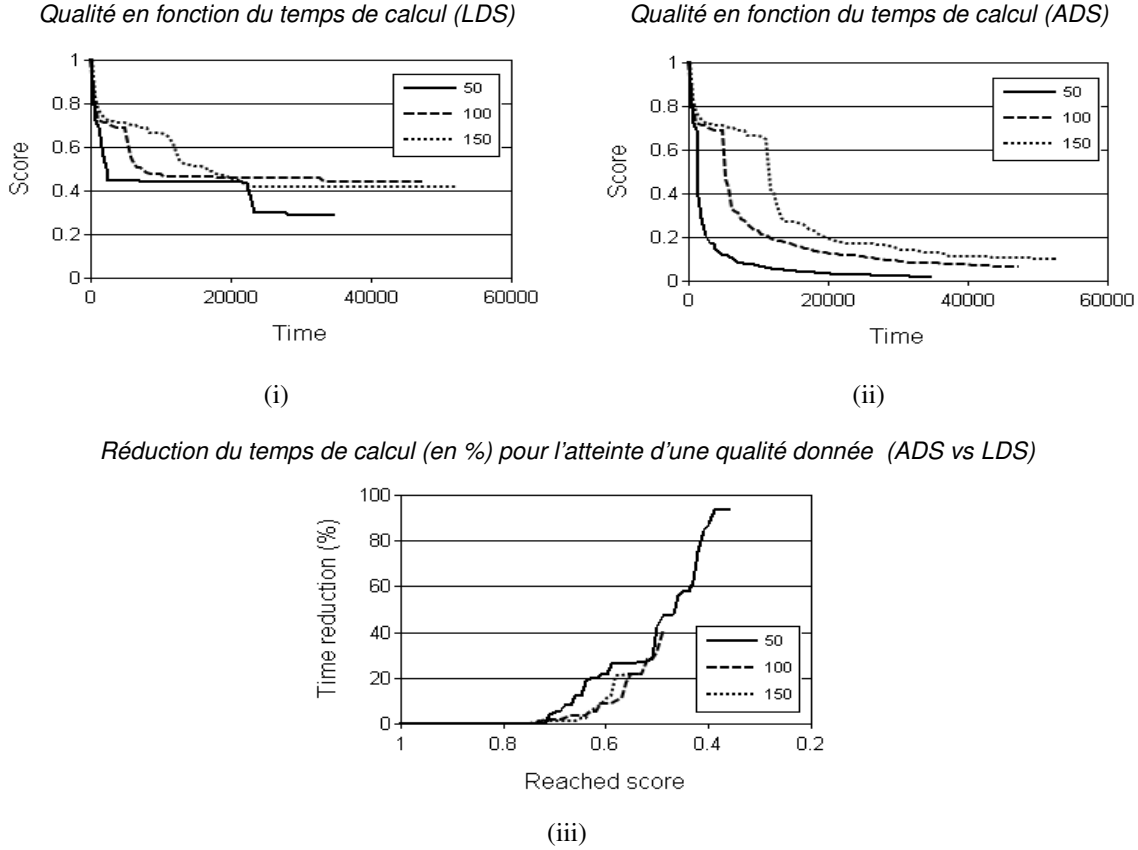


Figure 6.9 : Impact du nombre de sous-problèmes sur les performances des stratégies LDS et ADS. Résultats pour 50, 100 et 150 sous-problèmes avec [$0.0 \leq \beta \leq 0.5$; γ=10; δ=0].

La figure 6.10 montre l'impact du paramètre γ. Encore une fois, nous avons supposé des valeurs de β entre 0.0 et 0.5 selon une distribution exponentielle $e^{-\gamma x}$. Avec $\gamma = 0$, les nœuds des arbres prennent des valeurs de β issues d'une distribution uniforme. Plus γ est élevée, moins il y a de nœuds avec des valeurs de β élevées. Autrement dit, plus grand est le γ, moins il y a de nœuds pour lequel il est payant de réaliser un grand nombre de déviations. Il devrait donc être plus difficile de trouver de bonnes solutions (hypothèse 1) et la recherche et détection des nœuds « payants » devrait rapporter (hypothèse 2). Les sous-figures (i) et (ii) confirment l'hypothèse 1; pour une même stratégie, les courbes de la qualité en fonction du temps sont de moins en moins bonnes à mesure que γ croît. La sous-figure (iii) confirme l'hypothèse 2; plus

grand est le γ, plus la stratégie ADS montre un avantage important sur la stratégie LDS. Ces résultats illustrent empiriquement l'idée intuitive suivante : plus les nœuds « prometteurs » sont rares[37], plus il vaut la peine de les rechercher et de s'y attarder. Toutefois, même pour $\gamma = 0$, on remarque un avantage de ADS sur LDS. C'est qu'il y a une variabilité dans l'arbre (et donc des nœuds plus intéressants que les autres) même si les valeurs β sont tirées d'une distribution uniforme.

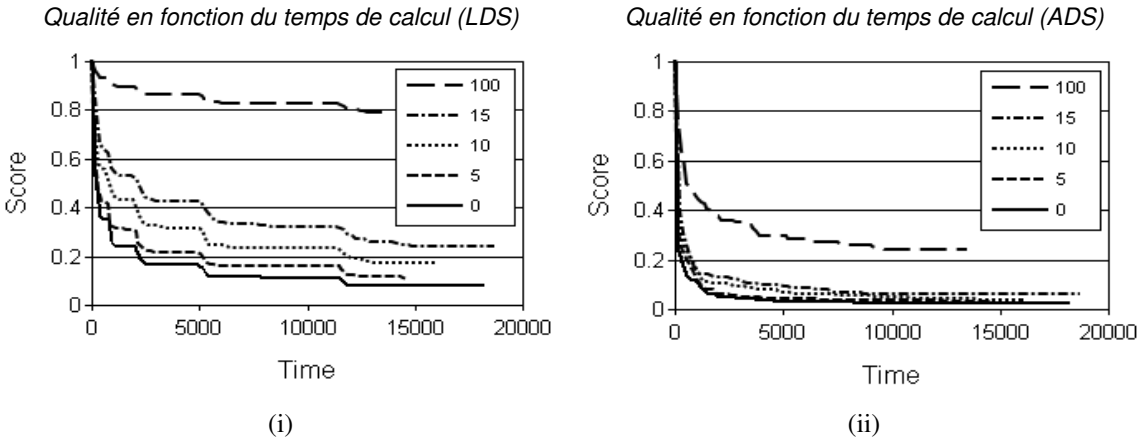


Figure 6.10 : Impact du paramètre γ sur les performances des stratégies LDS et ADS. Résultats pour γ=0; 1; 10; 15 et 100 avec [0.0 ≤ β ≤ 0.5; Card(X)=10; δ=0].

[37] Un nœud est « prometteur » s'il est payant à court terme de générer des solutions alternatives supplémentaires pour son sous-problème.

Finalement, la dernière expérience illustre l'impact du paramètre δ. Rappelons que les valeurs α des nœuds (la valeur vers laquelle `bestToDate[i]` tend pour de grandes valeurs `i`) sont pigées entre `arcValue[0]` et $\delta \times$ `arcValue[0]` selon une distribution uniforme. Lorsque nous avons δ = `0`, alors pour tout nœud la valeur `NextLeaf(i)` tend vers `0` pour un `i` élevé. Plus δ est élevé, plus il y a de variabilité dans l'arbre et plus l'apprentissage s'avère payant. Les résultats présentés à la figure 6.11 l'illustrent.

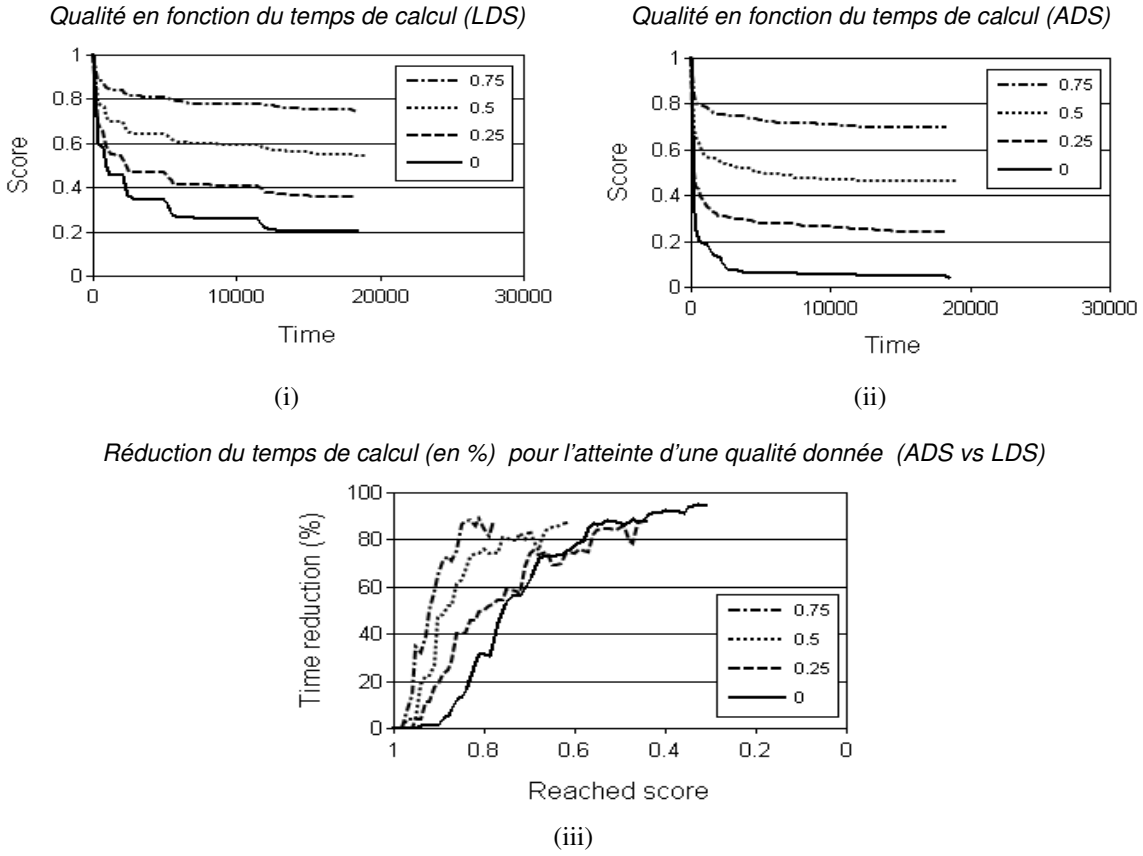


Figure 6.11 : Impact du paramètre δ sur les performances des stratégies LDS et ADS.
Résultats pour δ=`0`; `0.25`; `0.50` et `0.75` avec [`0.0` ≤ β ≤ `0.5`; γ=10; `Card(X)=10`].

6.6 Travaux apparentés

Cette section situe notre approche par rapport à d'autres qui utilisent l'apprentissage, mais dans le cadre de la résolution de problèmes combinatoires classiques. Mais tout

d'abord, il convient de rappeler la différence majeure suivante entre ces domaines (en plus du fait que nos problèmes soient distribués, bien sûr).

Pour les problèmes combinatoires classiques, c'est l'algorithme de résolution qui construit l'arbre de recherche (en choisissant l'ordre d'instanciation des variables et l'ordre des valeurs essayées). Des techniques d'apprentissage peuvent donc être utilisées pour établir une stratégie de choix de variables et de choix de valeur. Dans le contexte particulier des problèmes hiérarchiques, la séquence de sous-problèmes (équivalent à la séquence de variables) et l'ordre dans lequel les propositions sont produites par les solveurs locaux (équivalent au choix de variable) sont déterminés à l'avance (voir définition 4.1). Notre marge de manœuvre se limite donc à la stratégie de retour-arrière. Il est toutefois intéressant d'établir un parallèle entre les approches centralisées.

Nous distinguerons deux grands courants : (1) l'approche par entraînement et (2) l'approche dite *adaptative*[38]. Notre méthode s'inscrit dans le cadre de cette dernière.

6.6.1 Approche par entraînement

Cette approche consiste à entraîner un système pour la résolution d'une famille particulière de problèmes en lui fournissant une série de problèmes d'entraînement. Le système réalise alors de manière autonome ce que ferait le praticien cherchant à configurer un solveur pour un contexte particulier [72]. Par exemple, le système pourrait évaluer quelles heuristiques de choix de variables et de choix de valeur sont les meilleures pour une famille de problèmes donnée.

Le système ACE [73] utilise une approche semblable mais plus avancée. Aux termes de la phase d'entraînement, il attribue des poids aux différentes heuristiques en fonction de leur pertinence. Une fois en production, le système fera « voter » les différentes heuristiques (e.g. pour déterminer la prochaine variable sur laquelle brancher) en tenant

[38] Aucune appellation ne fait l'unanimité. Nous utilisons celle-ci sous toutes réserves.

compte de leurs poids. La performance obtenue peut être meilleure que celle des heuristiques individuelles.

Toujours dans la voie de l'entraînement, le système proposé par [74] étudie un ensemble d'arbres afin d'identifier des coupes pouvant être réalisées sur tous ces arbres tout en s'assurant qu'on puisse tout de même trouver de bonnes solutions. En production (i.e. après la phase d'entraînement, quand on aura à résoudre de nouveaux problèmes), on appliquera ces coupes aux nouveaux arbres.

6.6.2 Approche adaptative

L'approche dite *adaptative* concerne le développement de systèmes qui réagissent et s'ajustent dynamiquement pendant la résolution d'une instance particulière d'un problème. Cette approche est souvent mise à profit pour la résolution de problèmes de satisfaction de contraintes (CSP) centralisés.

Plusieurs méthodes appliquent ce qu'on appelle en anglais le *learning from failure*. Lors d'un échec pendant la descente de l'arbre, on analyse les conditions ayant mené à cet échec en vue de mettre à profit cette connaissance dans le reste de la recherche.

Par exemple, les techniques de *nogood recording* ou de *clause learning* visent à éviter de refaire plusieurs fois des combinaisons d'affectations qui sont inconsistantes entre elles. L'information cumulée lors de la recherche peut également servir à établir quelles variables sont les plus difficiles à instancier de manière à changer dynamiquement l'ordre d'instanciation des variables. L'algorithme YIELDS s'inscrit dans cette voie [75]. Également, dans *Impact Based Search* (IBS), on mesure l'impact des variables en observant de quelle façon leur instanciation réduit la taille de l'espace de recherche [76]. Dans [77] et [78], à chaque fois qu'une contrainte cause un échec, on augmente la priorité des variables impliquées dans cette contrainte. Dans [79], on propose une approche adaptative pour le choix de variables mais dans le contexte des *Weighted CSP*.

D'autres approches réalisent une certaine forme d'apprentissage dans une phase de prétraitement. Par exemple, dans [80], on réalise d'abord une recherche locale dans le but de trouver des solutions intéressantes. Celles-ci servent ensuite à initialiser les traces de phéromones utilisées par l'algorithme de type *colonie de fourmis* utilisé pour la suite.

Dans le contexte des stratégies de retour-arrière, les approches qui apprennent à évaluer la qualité d'un nœud/arc nous sont d'un intérêt particulier. Ruml a proposé une initiative intéressante à ce point de vue. Alors qu'un LDS de base accorde le même poids à toutes les déviations, BLFS [81] attribue dynamiquement des poids différents aux déviations selon leur profondeur. Pour chaque niveau d'un arbre binaire, il définit deux paramètres. L'un représente le coût de brancher à gauche à partir d'un nœud à ce niveau dans l'arbre, l'autre le coût de brancher à droite. La valeur d'une feuille est réputée être égale à la somme des coûts sur le chemin allant de la racine à cette feuille. En connaissant la valeur d'un certain nombre de feuilles, il réalise une régression linéaire afin d'établir la valeur des paramètres.

Dans les travaux de Ruml, le modèle n'est pas utilisé afin de définir une stratégie de retour-arrière. L'algorithme proposé réalise plutôt une série de descentes successives dans l'arbre. À chaque descente on tente de descendre par un chemin minimisant les coûts. À chaque niveau de profondeur, le choix est fait de manière stochastique afin de ne pas toujours emprunter le même chemin.

Ruml a obtenu de très bons résultats avec cet algorithme (voir [82]). La limite de cette approche est que le facteur de branchement doit être le même pour chaque nœud d'un même niveau; et on ne pourra rien dire sur une déviation supplémentaire à un nœud donné tant qu'on n'en a pas réalisé au moins autant dans un autre nœud sur le même niveau. De plus, l'impact de réaliser une énième déviation à une profondeur donnée est réputé être le même pour tous les nœuds de ce niveau. Or, cette condition n'était pas rencontrée pour notre problème industriel.

6.7 Conclusion

Nous avons proposé une stratégie de recherche adaptative (ADS) destinée à permettre une prise de décision plus efficace en contexte hiérarchique. Elle permet aux agents d'identifier collectivement et dynamiquement quelles zones de l'arbre sont les plus prometteuses et de s'y attarder en priorité.

Concrètement, les agents y arrivent en évaluant l'opportunité de réaliser des déviations supplémentaires; un modèle leur permet d'apprendre à évaluer cet impact pour chaque nœud et en fonction du nombre de déviations déjà réalisées à ce nœud. Le mécanisme permet aux agents de chercher de manière systématique dans un arbre (et ainsi espérer trouver la solution optimale) tout en visant à obtenir de bonnes solutions en un temps très court.

Nous avons évalué la méthode pour notre cas industriel; elle a permis de réduire les temps de calcul de près de la moitié. De plus nous avons évalué la méthode avec des problèmes synthétiques. Cela a permis une évaluation pour un large éventail de problèmes qui différaient les uns des autres en fonction du degré de facilité avec lequel on pouvait espérer trouver des nœuds/sous-problèmes menant à de bonnes solutions.

Chapitre 7 Conclusion

Cette thèse portait sur la coordination d'entités autonomes en contexte hiérarchique. On rencontre plusieurs de ces problèmes en génie industriel; qu'il s'agisse de problèmes de coordination dans les chaînes logistiques, ou plus simplement de prise de décision au sein d'une organisation.

Les contributions scientifiques de la thèse se situent à la fois dans le domaine d'application (gestion des opérations) et dans le domaine plus fondamental des algorithmes d'optimisation distribués (une sous-branche de l'intelligence artificielle distribuée).

7.1 HDCOP

Au chapitre Chapitre 4, nous avons discuté de la possibilité de modéliser et résoudre le problème étudié à l'aide de techniques de raisonnement sur contraintes distribuées (en anglais, *Distributed Constraint Reasoning*, ou DCR) proposées dans la littérature. Certains aspects de nature organisationnelle ne sont pas pris en compte par les formalismes classiques. Nous avons donc proposé un nouveau formalisme appelé HDCOP. Il permet de représenter formellement le problème, l'attribution des responsabilités entre les agents, la séquence de résolution et l'espace de coordination accessible aux agents.

L'introduction de ce formalisme visait deux objectifs. D'une part, nous avons montré comment une grande quantité de problèmes pratiques de coordination peuvent bénéficier des approches systématiques de recherche distribuée (l'alternative étant l'utilisation de simples heuristiques). Une fois le problème modélisé en tant que HDCOP, un algorithme distribué de base (e.g. SyncBB) peut être utilisé par les agents en tant que mécanisme de coordination. Pour le cas industriel étudié, cela permettait une nette amélioration par rapport aux heuristiques de coordination.

D'autre part, l'introduction de ce formalisme enrichit le cadre de réflexion de la communauté DCR. En établissant cette distinction entre DCOP et HDCOP, nous proposons un système de classification pour les problèmes, mais aussi pour les algorithmes. Par exemple, SyncBB ne s'applique qu'aux DCOP et aux HDCOP, mais ADOPT est destiné aux DCOP seulement. Le fait d'avoir proposé une définition formelle pour les HDCOP ouvre donc en même temps la porte à la création de nouveaux algorithmes.

7.2 Introduction de nouveaux algorithmes d'optimisation distribués

L'espace de coordination représenté par un HDCOP montre certaines caractéristiques particulières. Il s'agit d'un arbre non-binaire de profondeur fixe avec un facteur de branchement très grand et variable d'un nœud à l'autre. Également, l'arbre est généré dynamiquement pendant la résolution. Dans ces circonstances, SyncBB ne visite que des solutions très semblables les unes des autres (puisque contenues dans la même zone de l'arbre) même pour un temps de calcul très grand. Pour remédier à ce problème, nous avons adapté au contexte multi-agent le principe des stratégies de recherche centralisées basées sur l'analyse des déviations. Ces méthodes sont très utilisées en programmation par contraintes classique.

Nous avons proposé SyncLDS, un protocole qui permet aux agents de visiter les solutions globales dans le même ordre que le ferait un algorithme centralisé appliquant la politique LDS. Il s'agit à notre connaissance du premier algorithme de recherche distribué exploitant la notion de calcul des déviations. Suite à l'introduction de notre méthode auprès de la communauté scientifique, au moins un autre groupe de chercheurs[39] étudie comment le concept pourrait être utilisé pour la résolution de DCOP classiques.

---

[39] W. Yeoh, étudiant au doctorat du professeur S. Koenig à l'University of Southern California (communication personnelle, janvier 2009).

Nous avons également proposé l'algorithme MacDS. Il permet aux agents de travailler simultanément, ce qui accélère le processus de résolution. En proposant cet algorithme, nous avons montré comment cette caractéristique pouvait être introduite en optimisation distribuée sans pour autant être obligé d'utiliser la notion d'*asynchronisme*, au sens de ADOPT. MacDS montre également des propriétés intéressantes en tant qu'algorithme distribué : tolérance aux pannes de communication, à l'inversion de l'ordre des messages, etc.

Ces deux algorithmes ont été évalués pour le cas d'étude industriel. Ils ont permis d'améliorer la qualité des solutions et les temps de calcul nécessaires à leur obtention.

Ils ont également été évalués avec des données synthétiques. L'objectif était de montrer le potentiel des algorithmes en fonction du niveau d'exactitude de l'hypothèse qui sous-tend LDS (i.e. qu'il existe une relation inverse entre la qualité d'une solution et le nombre de déviations associées à une feuille). Ce modèle théorique est une contribution supplémentaire de cette thèse. En utilisant ce modèle nous avons pu montrer ceci : le travail simultané des agents permet à la stratégie de retour-arrière LDS de dominer le retour-arrière chronologique, peu importe le niveau de validité de l'hypothèse qui sous-tend LDS (ce qui n'est pas le cas lorsque ces stratégies sont utilisées par un algorithme synchrone, ou encore en environnement centralisé).

7.3 Utilisation d'un mécanisme d'apprentissage pour l'optimisation distribuée

Enfin, nous avons proposé une méthode nommée ADS. Plutôt qu'appliquer une stratégie de retour-arrière définie à l'avance (telle que LDS) on permet aux agents d'établir dynamiquement la stratégie à utiliser. Sous ADS, les agents découvrent collectivement et dynamiquement quelles zones de l'arbre semblent les plus prometteuses dans le but de s'y attarder en priorité. La méthode prend appui sur le fait que les arbres sont non-binaires. Nous avons proposé un modèle qui permet d'anticiper l'impact associé à la réalisation d'une énième déviation à partir d'un nœud donné, en se basant sur la qualité des feuilles visitées précédemment.

Cette méthode a été évaluée pour les problèmes industriels, pour lesquels elle a permis d'obtenir de bonnes solutions beaucoup plus rapidement qu'avec une stratégie basée sur LDS. Il y a donc un élément de structure dans ces problèmes distribués qui n'est pas capturé par LDS (lequel attribue la même valeur à chaque déviation) mais que notre modèle arrive à exploiter.

Le modèle qui sert à réaliser l'apprentissage pendant la recherche peut également être utilisé pour caractériser un arbre de recherche *a posteriori*, de même que pour générer des problèmes montrant différentes caractéristiques. Sur cette base, nous avons pu étudier la relation entre la performance de notre algorithme et le degré d'uniformité qu'on retrouve au niveau de la distribution des bonnes solutions dans l'arbre. Dans le futur, le modèle pourrait également être utilisé pour étudier la structure d'autres problèmes.

7.4 Limites

Le formalisme que nous avons introduit (HDCOP) et les algorithmes qui en découlent sont destinés à la coordination entre agents autonomes évoluant dans un contexte hiérarchique. Ces agents sont *autonomes* au sens qu'ils existent *a priori* (ils prennent leurs décisions locales avec un algorithme de leur choix). Le contexte est dit *hiérarchique* au sens que des contraintes organisationnelles régissent l'interaction entre ces agents.

Nous avons discuté au chapitre Chapitre 4 de certaines limites de l'approche. Tout d'abord, il est clair que la prise en compte de ces contraintes organisationnelles nous force à limiter l'espace des solutions considéré (c'est la raison pour laquelle nous avons parlé d'*espace de coordination*, plutôt que d'*espace des solutions*). Cependant, ces contraintes organisationnelles sont bien réelles. Nous devons en tenir compte au même titre que le créateur d'un algorithme d'ordonnancement doit tenir compte de contraintes interdisant la préemption sur une machine : pour abolir cette contrainte ce créateur devrait modifier la machine. Or, dans notre contexte, modifier la « machine

organisationnelle » (par une fusion d'entreprise, ou autre) est une question qui relève du management et qui n'a pas été abordée dans cette thèse.

Quant au domaine d'application spécifique des réseaux de création de valeur, il existe certaines topologies de réseaux que nous ne pouvons coordonner avec l'approche HDCOP. Dans cette thèse nous avons appliqué notre méthode à des réseaux de production linéaires. Au chapitre Chapitre 4, nous avons également expliqué comment la méthode peut être appliquée à des réseaux convergents. Cependant nous ne pourrions pas utiliser la méthode proposée pour des réseaux de production divergents. C'est qu'il serait alors difficile d'imaginer une séquence de résolution des sous-problèmes (nécessaire pour modéliser le problème en tant que DCOP) conforme au cadre logistique pouvant régir les relations dans un tel réseau. Ces cas ne sont donc pas couverts par notre méthode.

7.5 Potentiel d'application en contexte centralisé

Le formalisme introduit (HDCOP) pourrait être utilisé pour modéliser un problème centralisé pouvant être décomposé en une séquence de décisions pour lesquelles on dispose d'algorithmes spécialisés performants. Ces algorithmes pourraient exister *a priori* et reposer sur des technologies d'optimisation différentes les unes des autres. Le formalisme HDCOP permettrait alors de décrire un espace de solution plus vaste que celui contenant l'unique solution qui serait obtenue en résolvant simplement la séquence de sous-problème. Par contre, cet espace de solution serait beaucoup plus petit que celui qui serait décrit par un modèle centralisé standard. L'intérêt de HDCOP pour les problèmes centralisés serait donc limité aux situations où un modèle ou algorithme centralisé est inexistant ou non envisageable pour des questions de performance.

La recherche dans cet espace HDCOP pourrait être réalisée en utilisant une stratégie standard basée sur la valeur des bornes calculées à chaque nœud. Cependant, si celles-ci s'avéraient mauvaises, on pourrait alors envisager l'utilisation d'une stratégie basée sur l'analyse des déviations (SyncLDS ou MacDS). Dans le cas où MacDS serait utilisé,

chaque type de sous-problème pourrait être sous la responsabilité d'un processeur spécialisé différent.

La stratégie adaptative ADS pourrait également être considérée. En effet, l'arbre HDCOP proposé montrerait des caractéristiques topologiques similaires au cas distribué (il serait très large et peu profond). Pour connaître la pertinence réelle d'utiliser ADS, il faudrait cependant évaluer à quel point les bonnes solutions sont distribuées non-uniformément dans l'arbre, en utilisant le modèle que nous avons proposé à cet effet.

Finalement, il existe à notre avis d'autres catégories de problèmes centralisés pour lesquels la méthode adaptative pourrait être intéressante. Il s'agit des problèmes pour lesquels un planificateur humain expérimenté arrive à proposer des heuristiques efficaces. C'est notamment le cas du problème de planification des opérations de finition (un des sous-problèmes du cas industriel étudié dans cette thèse - voir le chapitre chapitre 2 et l'annexe 3). Bien que ce problème puisse être modélisé sous la forme d'un problème linéaire mixte en nombre entiers, nous n'avons pas obtenu de bonnes solutions dans des temps raisonnables avec un solveur standard. C'est la raison pour laquelle nous avons proposé une heuristique inspirée de la pratique industrielle, que nous avons ensuite généralisée pour en faire une stratégie de recherche (annexe 3). L'arbre qui est implicitement défini par cette stratégie a certaines caractéristiques particulières : différents nœuds correspondent à différents types de décision (choix d'une famille de produit à traiter, choix de quantités, choix d'une machine, etc.). Certaines de ces décisions sont-elles plus importantes que les autres? Peut-être serait-il intéressant de définir dynamiquement la stratégie de retour-arrière, de manière à privilégier les nœuds renfermant un bon potentiel d'amélioration? ADS pourrait-il être d'un bon usage dans un tel contexte? Cette question n'a pas été étudiée dans cette thèse, le focus ayant été maintenu sur la question de la coordination d'entités autonomes. Elle pourrait cependant l'être dans le cadre de travaux futurs.

7.6 Applications industrielles

La problématique de la coordination est une réalité pour la plupart des grandes entreprises manufacturières. Nous espérons que nos travaux contribueront à l'émergence d'une nouvelle génération d'outils permettant de rendre nos entreprises et organisations davantage compétitives. À cet égard, les concepteurs de logiciels pourront s'inspirer des scénarios d'intégration que nous avons définis au chapitre Chapitre 4.

Les méthodes proposées pourraient également être appliquées dans d'autres secteurs, notamment pour la coordination au sein d'entreprises offrant des services de logistique (transport, notamment). Tel qu'abordé dans la section précédente, nous imaginons également l'utilisation de certains éléments proposés de manière à faciliter la coordination au sein de grandes organisations – pensons notamment au secteur de la santé.

Finalement, concernant le problème industriel spécifique qui a servi de cas d'étude dans cette thèse, les membres industriels du Consortium de recherche FORAC espèrent voir la création d'une entreprise destinée à commercialiser un outil de planification de la production pour l'industrie du bois d'œuvre. Nous espérons que cette entreprise saura s'inspirer des travaux de cette thèse et des articles scientifiques qui en sont issus.

Références

[1] Schneeweiss C. *Distributed Decision Making*, New York: Springer, 2003.

[2] Moyaux T., Chaib-draa B. and D'Amours S. "Supply Chain Management and Multiagent Systems: An Overview" in: *Multiagent-Based Supply Chain Management*, Chaib-draa B. and Müller J. P. Eds. New York: Springer, 2006.

[3] Kilger C. and Reuter B. "Collaborative Planning" in: *Supply Chain Management and Advanced Planning*, Stadtler H. and Kilger C. Eds. New York: Springer, 2005. pp. 259-278.

[4] Fink A. "Supply Chain Coordination by Means of Automated Negotiations Between Autonomous Agents" in: *Multiagent-Based Supply Chain Management*, Chaib-draa B. and Müller J. P. Eds. New York: Springer, 2006. pp. 450

[5] Rosenschein J. S. and Genesereth M. R., "Deals Among Rational Agents", *Proceedings of the Ninth International Joint Conference on Artificial Intelligence (IJCAI)*, 1985, pp. 91-99.

[6] Beaumont P., "Multi-Platform Coordination and Resource Management in Command and Control". M.Sc. thesis, Université Laval, Québec, 2004.

[7] Dean T. and Boddy M., "An analysis of time-dependant planning", *The Seventh National Conference on Artificial Intelligence (AAAI)*, 1988, pp. 49-54.

[8] Cachon G. P. "Supply Chain Coordination with Contracts" in: *Supply Chain Management: Design, Coordination and Operation*, de Kok A. G. and Graves S. G. Eds. Amsterdam: Elsevier, 2003.

[9] Lee H.L. and Whang S., "Decentralized multi-echelon supply chains: incentives and information", *Management Science*, vol 45, no 5, 1999, pp. 633-640.

[10] Gurnani H. Y., "Coordination in decentralized assembly systems with uncertain component yields", *European Journal of Operational Research*, vol 176, no 3, 2007, pp. 1559-1576.

[11] Anupindi R., Bassok Y. and Zemel E., "A General Framework for the Study of Decentralized Distribution Systems", *Manufacturing and Service Operations Management*, vol 3, no 4, 2001, pp. 349-368.

[12] Karaesmen F., Buzacott J. A. and Dallery Y., "Integrating advance order information in make-to-stock production systems", *IIE Transactions*, vol 34, no 8, 2002, pp. 649-662.

[13] Cachon G. P. and Lariviere M. A., "Supply chain coordination with revenue-sharing contracts: Strengths and limitations", *Management Science*, vol 51, no 1, 2005, pp. 30-44.

[14] Shin H. and Benton W. C., "A quantity discount approach to supply chain coordination", *European Journal of Operational Research*, vol 180, no 2, 2007, pp. 601-616.

[15] Sarmah S. P., Acharya D. and Goyal S. K., "Buyer vendor coordination models in supply chain management", *European Journal of Operational Research*, vol 175, no 1, 2006, pp. 1-15.

[16] Tsay A. A., "The quantity flexibility contract and supplier-customer incentives", *Management Science*, vol 45, no 10, 1999, pp. 1339-1358.

[17] Gullu R., Van Houtum G. J., Sargut F. Z. and Erkip N., "Analysis of a decentralized supply chain under partial cooperation", *Manufacturing and Service Operations Management*, vol 7, no 3, 2005, pp. 229-247.

[18] Schneeweiss C. and Zimmer K., "Hierarchical coordination mechanisms within the supply chain", *European Journal of Operational Research*, vol 153, no 3, 2004, pp. 687-703.

[19] Dudek G. and Stadtler H., "Negotiation-based collaborative planning between supply chains partners", *European Journal of Operational Research*, vol 163 , no 3, 2005, pp. 668-687.

[20] Monostori L., Vancza J. and Kumara S. R. T., "Agent-Based Systems for Manufacturing", *CIRP Annals - Manufacturing Technology*, vol 55, no 2, 2006, pp. 697-720.

[21] Frayret J. M., D'Amours S., Rousseau A., Harvey S. and Gaudreault J., "Agent-based Supply Chain Planning in the Forest Products Industry", *International Journal of Flexible Manufacturing Systems*, vol 19, no 4, 2007.

[22] Shen W. and Norrie D. H., "Agent-based systems for intelligent manufacturing: a state-of-the-art survey", *Knowledge and Information Systems*, vol 1, no 2, 1999, pp. 129-156.

[23] Shen W., Hao Q., Yoon H. J. and Norrie D. H., "Applications of agent-based systems in intelligent manufacturing: An updated Review", *Advanced Engineering Informatics*, vol 20, no 4, 2006, pp. 415-431.

[24] Shen W., Wang L. and Hao Q., "Agent-based distributed manufacturing process planning and scheduling: a state-of-the-art survey", *IEEE Transactions on Systems, Man and Cybernetics, Part C*, vol 36, no 4, 2006, pp. 563-77.

[25] Barbuceanu M. and Fox M. S., "COOL: a language for describing coordination in multi-agent systems", *Proceedings of First International Conference on Multiagent Systems*, 1995, pp. 17-24.

[26] Smith R. G., "The contract net protocol: high level communication and control in a distributed problem solver", *IEEE Transactions on Computers*, vol C-29, no 12, 1980, pp. 1104-1113.

[27] Ertogral K. and Wu S. D., "Auction-theoretic coordination of production planning in the supply chain", *IIE Transactions*, vol 32, no 10, 2000, pp. 931-940.

[28] Bhatnagar R., Chandra P. and Goyal S. K., "Models for multi-plant coordination", *European Journal of Operational Research*, vol 67, no 2, 1993, pp. 141-160.

[29] Luh P. B., Ming Ni, Haoxun Chen and Thakur L. S., "Price-based approach for activity coordination in a supply network", *IEEE Transactions on Robotics and Automation*, vol 19, no 2, Apr, 2003- , pp. 335-346.

[30] Nishi T., Konishi M. and Shinozaki R., "An augmented Lagrangian approach for decentralized supply chain planning for multiple companies", *The International Conference on System, Man and Cybernetics*, 2005, pp. 1168-1173.

[31] Maheswaran R. T., Pearce J. P., Bowring E., Varakantham P. and Tambe M., "Privacy loss in distributed constraint reasoning: a quantitative framework for analysis and its applications", *Autonomous Agents and Multi-Agent Systems*, vol 13, no 1, 2006, pp. 27-60.

[32] Modi P. J., Shen W. M., Tambe M. and Yokoo M., "An asynchronous complete method for general distributed constraint optimization", *Proceedings of the Third International Workshop on Distributed Constraint Reasoning (DCR)*, 2002.

[33] Yokoo M., Ishida T., Durfee E. H. and Kuwabara K., "Distributed constraint satisfaction for formalizing distributed problem solving", *International Conference on Distributed Computing Systems*, 1992, pp. 614-621.

[34] Lesser V. R. and Corkill D. D., "Functionally accurate, cooperative distributed systems", *IEEE Transactions on Systems, Man and Cybernetics*, vol 11, no 1, 1981, pp. 81-96.

[35] Durfee E. H., Lesser V. R. and Corkill D. D., "Trends in cooperative distributed problem solving", *Knowledge and Data Engineering, IEEE Transactions on*, vol 1, no 1, 1989, pp. 63-83.

[36] Durfee E. H. and Rosenschein J., "Distributed problem solving and multi-agent systems: Comparisons and examples", *Proceedings of the 13th International Workshop on Distributed Artificial Intelligence*, 1994, pp. 94-104.

[37] Yokoo M. *Distributed constraint satisfaction: foundations of cooperation in multi-agent systems*, Berlin, New York: Springer, 2001.

[38] Meisels A., "Distributed Constraints Satisfaction Algorithms, Performance, Communication", *Tutorial of the International Conference on Principles and Practice of Constraint Programming Conference (CP)*, 2004.

[39] Russell S. J. and Norvig P. *Artificial intelligence a modern approach*, Upper Saddle River, N.J: Prentice Hall/Pearson Education, 2003.

[40] Yokoo M., Durfee E. H., Ishida T. and Kuwabara K., "The distributed constraint satisfaction problem: formalization and algorithms", *IEEE Transactions on Knowledge and Data Engineering*, vol 10, no 5, 1998, pp. 673-685.

[41] Yokoo M. and Hirayama K., "Algorithms for distributed constraint satisfaction: a review", *Autonomous Agents and Multi-Agent Systems*, vol 3, no 2, 2000, pp. 185-207.

[42] Bessiere C., Brito I., Maestre A. and Meseguer P., "The Asynchronous Backtracking Family", LIRMM-CNRS, Montpellier, France, Report 03139, 2003.

[43] Bessiere C., Maestre A., Brito I. and Meseguer P., "Asynchronous backtracking without adding links: a new member in the ABT family", *Artificial Intelligence*, vol 161, no 1-2, 2005, pp. 7-24.

[44] Bessiere C., Maestre A. and Meseguer P., "Distributed dynamic backtracking", *Proceedings of 7th International Conference on Principles and Practice of Constraint Programming (CP), LNCS #2239*, 2001.

[45] Zivan R. and Meisels A., "Synchronous vs Asynchronous search on DisCSPs", *Proceedings of the European Workshop on Multi-Agent Systems (EUMAS)*, 2003.

[46] Zivan R. and Meisels A., "Concurrent backtrack search on DisCSPs", *Proceedings of the 17th International Florida Artificial Intelligence Research Society Conference (FLAIRS)*, 2004, pp. 776-781.

[47] Zivan R. and Meisels A., "Concurrent dynamic backtracking for distributed CSPs", *Proceedings of the 10th International Conference on Principles and Practice of Constraint Programming (CP), LNCS #3258*, 2004, pp. 782-787.

[48] Zivan R. and Meisels A., "Dynamic ordering for asynchronous backtracking on DisCSPs", *Proceedings of the 11th International Conference on Principles and Practice of Constraint Programming (CP), LNCS #3709*, 2005, pp. 32-46.

[49] Zivan R. and Meisels A., "Concurrent search for distributed CSPs", *Artificial Intelligence*, vol 170, no 4-5, 2006, pp. 440-61.

[50] Ginsberg M. L., "Dynamic backtracking", *Journal of Artificial Intelligence Research*, vol 1, 1999, pp. 25-46.

[51] Modi P. J., "Distributed constraint optimization for multiagent systems". 3133310, University of Southern California, United States – California, 2003.

[52] Hirayama K. and Yokoo M., "Distributed partial constraint satisfaction problem", *Proceedings of the Third International Conference on Principles and Practice of Constraint Programming (CP), LNCS #1330*, 1997, pp. 222-236.

[53] Yokoo M. and Durfee E. H., University of Michigan, Ann Arbor, Michigan, CSE-TR-101-91, 1991.

[54] Yokoo M. and Hirayama K., "Distributed breakout algorithm for solving distributed constraint satisfaction problems", *International Conference on Multiagent Systems*, 1996, pp. 401-408.

[55] Modi P. J., Shen W. M., Tambe M. and Yokoo M., "An asynchronous complete method for distributed constraint optimization", *Proceedings of the Second International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS)*, 2003.

[56] Modi P. J., Shen W. M., Tambe M. and Yokoo M., "Adopt: asynchronous distributed constraint optimization with quality guarantees", *Artificial Intelligence*, vol 161, no 1-2, 2005, pp. 149-180.

[57] Korf R. E., "Depth-first iterative-deepening: an optimal admissible tree search", *Artificial Intelligence*, vol 27, no 1, 1985, pp. 97-109.

[58] Bowring E., Tambe A. M. and Yokoo M., "Multiply-constrained distributed constraint optimization", *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems (AAMAS)*, 2006, pp. 1413-1420.

[59] Petcu A. and Faltings B., "DPOP: A Scalable Method for Multiagent Constraint Optimization", *International Joint Conference on Artificial Intelligence (IJCAI)*, 2005.

[60] Kschischang F. R., Frey B. J. and Loeliger H.-A., "Factor graphs and the sum-product algorithm", *IEEE Transactions on Information Theory*, vol 47, no 2, 2001, pp. 498-519.

[61] Petcu A. and Faltings B., "A Distributed, Complete Method for Multi-Agent Constraint Optimization", *Fifth International Workshop on Distributed Constraint Reasoning (DCR)*, 2004.

[62] Parunak H. V. D., Ward A., Fleischer M., Sauter J. and Chang T.-C., "Distributed component-centered design as agent based distributed constraint optimization", *Proceedings of the AAAI Workshop on Constraints and Agents*, 1997.

[63] Harvey W. D., "Nonsystematic backtracking search". Ph.D. thesis, Stanford University, California, 1995.

[64] Harvey W. D. and Ginsberg M. L., "Limited discrepancy search", *International Joint Conference on Artificial Intelligence (JCAI)*, 1995, pp. 607-613.

[65] Le Pape C. and Baptiste P., "Heuristic control of a constraint-based algorithm for the preemptive job-shop scheduling problem", *Journal of Heuristics* , vol 5, no 3, 1999, pp. 305-325.

[66] Beck J. C. and Perron L., "Discrepancy-Bounded Depth First Search", *Workshop on Integration of AI and OR Technologies for Combinatorial Optimization Problems (CPAIOR)*, 2000, pp. 7-17.

[67] Walsh T., "Depth-bounded discrepancy search", *International Joint Conference on Artificial Intelligence (JCAI)*, 1997, pp. 1388-1393.

[68] Lynch N. A. *Distributed algorithms*, San Francisco: Morgan Kaufmann, 1996.

[69] Press W. H. *Numerical recipes the art of scientific computing*, Cambridge: Cambridge University Press, 2007.

[70] Marquardt D. W., "An algorithm for least-squares estimation of nonlinear parameters", *Journal of the Society for Industrial and Applied Mathematics*, vol 11, no 2, 1963, pp. 431-441.

[71] Press W. H. *Numerical recipes in C the art of scientific computing*, Cambridge, UK: Cambridge University Press, 1992.

[72] Hutter F., Babic D., Hoos H. H. and Hu A. J., "Boosting verification by automatic tuning of decision procedures", *Formal Methods in Computer Aided Design*, 2007, pp. 27-34.

[73] Epstein S. L., Freuder E. C. and Wallace R. J., "Learning to support constraint programmers", *Computational Intelligence*, vol 21, no 4, 2005, pp. 336-371.

[74] Breimer E., Goldberg M., Hollinger D. and Lim D., "Discovering optimization algorithms through automated learning", *Graphs and Discovery. DIMACS Working Group Computer-Generated Conjectures from Graph Theoretic and Chemical Databases, 12-16 Nov. 2001*, 2005, pp. 7-25.

[75] Karoui W., Huguet M.-J., Lopez P. and Naanaa W., "YIELDS: a yet improved limited discrepancy search for CSPs", *Proceedings of the 4th International Conference on the Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR), LNCS #4510*, 2007, pp. 99-111.

[76] Refalo P., "Impact-based search strategies for constraint programming", *International Conference on Principles and Practice of Constraint Programming (CP), LNCS #3258*, 2004, pp. 557-571.

[77] Boussemart F., Hemery F., Lecoutre C. and Sais L., "Boosting systematic search by weighting constraints", *Proceedings of the 16th European Conference on Artificial Intelligence (ECAI)*, 2004, pp. 146-150.

[78] Grimes D. and Wallace R. J., "Learning from failure in constraint satisfaction search", *AAAI Workshop*, 2006, pp. 7-14.

[79] Levasseur N., Boizumault P. and Loudni S., "A value ordering heuristic for weighted CSP", *Proceedings of the 19th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2007, pp. 259-62.

[80] Solnon C., "Boosting ACO with a preprocessing step", *EvoWorkshops - Applications of Evolutionary Computing, LNCS #2279*, 2002, pp. 163-172.

[81] Ruml W., "Adaptive Tree Search". Ph.D. thesis, Harvard University, 2002.

[82] Ruml W., "Heuristic Search in Bounded-depth Trees: Best-Leaf-First Search", *Working Notes of the AAAI-02 Workshop on Probabilistic Approaches in Search*, 2002.

[83] Bartak R., "Conceptual Models for Combined Planning and Scheduling", *Proceedings of CP99 Workshop on Large Scale Combinatorial Optimisation and Constraints*, 1999, pp. 2-14.

[84] Bartak R., "On the Boundary of Planning and Scheduling: A Study", *Proceedings of the 18th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG)*, 1999, pp. 28-39.

Annexe 1 Planification des opérations de sciage

1. Le modèle

Nous présentons ici le modèle utilisé pour la planification et l'ordonnancement des activités de sciage. Chaque processus est modélisé sous la forme d'une association entre des quantités de billes en entrée, le panier de produit attendus et les temps d'utilisation des machines. Plusieurs processus peuvent être utilisés pendant le même quart de travail, sous réserve de certaines contraintes de configuration (voir section 2.1).

Ensembles

T	nombre de périodes dans l'horizon de planification. Dans ce qui suit, l'index t réfère aux périodes $t = 1,\ldots,T$;
$\mathbf{P}$	ensemble des produits p;
$\mathbf{P}^{consumed}$	sous-ensemble des produits p pouvant être consommés (i.e. les billes). $\mathbf{P}^{consumed} \subseteq \mathbf{P}$;
$\mathbf{P}^{produced}$	sous-ensemble des produits p qui peuvent être fabriqués (i.e. sciages bruts-verts). $\mathbf{P}^{produced} \subseteq \mathbf{P}$. Les sous-ensembles $\mathbf{P}^{consumed}$ et $\mathbf{P}^{consumed}$ sont disjoints;
$\mathbf{M}$	ensemble des machines m;
$\mathbf{A}$	ensemble des processus (activités) de transformation a utilisables au sein de l'unité;
$\mathbf{F}$	chaque élément $f \in \mathbf{F}$ défini un mode opérationnel selon lequel l'usine peut être configurée;
$\mathbf{F}^a$	défini les modes $f \in \mathbf{F}^a \subseteq \mathbf{F}$ permettant à l'usine d'exécuter le processus a. Un processus peut être compatible avec plusieurs modes et plusieurs processus peuvent être compatibles avec le même mode;

Paramètres

$i_{p,0}$	quantité du produit $p \in \mathbf{P}$ en inventaire au début de l'horizon de planification;
i_p	coût d'inventaire (par période) pour une unité de volume du produit p;
$s_{p,t}$	approvisionnement en produit $p \in \mathbf{P}^{consumed}$ livré à l'usine et disponible à la consommation au début de la période t;

$d_{p,t}$ demande pour le produit $p \in \mathbf{P}^{produced}$. Pour être livrée à temps, cette quantité doit être disponible pour expédition à la fin de la période t;

w_p coût à assumer lorsque qu'une unité de volume du produit $p \in \mathbf{P}^{produced}$ est en retard pour une période;

v_a coût variable associé à la réalisation du processus a, incluant les coûts des matières premières si applicables;

$\phi_{a,p}$ volume de matière première $p \in \mathbf{P}^{consumed}$ consommé chaque fois que le processus a est réalisé. Un processus a peut consommer plusieurs types de produits simultanément;

$\rho_{a,p}$ volume de produit $p \in \mathbf{P}^{produced}$ obtenu à chaque fois que le processus a est réalisé. Un processus peut produire plusieurs produits simultanément;

$\delta_{a,m}$ capacité de la machine $m \in \mathbf{M}$ (en unités de temps) utilisée à chaque fois que le processus a est réalisé;

$c_{m,t}$ Capacité de la machine disponible à chaque période t (en unités de temps).

Variables

$QC_{p,t}$ volume total de produit $p \in \mathbf{P}^{consumed}$ consommé pendant la période t;

$QP_{p,t}$ volume total de produit $p \in \mathbf{P}^{produced}$ fabriqué durant la période t;

$I^{+}_{p,t}$ volume de produit $p \in \mathbf{P}$ en stock à la fin de la période t;

$I^{-}_{p,t}$ demande cumulée pour le produit $p \in \mathbf{P}$ qui n'est pas satisfaite à la fin de la période t. C'est donc le volume de commandes en retard à un moment donné;

$I_{p,t}$ volume de produit $p \in \mathbf{P}$ qui serait en stock si la demande cumulée était satisfaite. Cette variable prend une valeur négative lorsque $I^{-}_{p,t}$ est positive. La variable est introduite afin de simplifier l'expression des contraintes de flux. Voir la contrainte (1.7) pour la relation entre $I_{p,t}$, $I^{+}_{p,t}$ et $I^{-}_{p,t}$;

$Y_{f,t}$ Variable binaire prenant la valeur 1 lorsque l'usine est configure selon le mode f à la période t; 0 sinon;

$X_{a,t}$ Utilisation du processus a à la période t. Il s'agit d'une variable continue;

Fonction objectif

Dans le contexte de l'application industrielle étudiée, on ne peut pas considérer la demande comme une contrainte dure. En effet, une usine peut recevoir une demande impossible à satisfaire; les retards sont alors inévitables et notre rôle consiste à minimiser ces retards. Conséquemment la fonction objectif proposée cherche à minimiser le coût de ces retards. Nous prenons également en compte le coût d'inventaire et les coûts variables de production.

$$\text{Min} \quad \sum_{\forall p \in \mathbf{P}^{\text{produced}}} \left(w_p \sum_{t=1}^{T} I_{p,t}^{-} \right) + \sum_{\forall p \in \mathbf{P}} \left(i_p \sum_{t=1}^{T} I_{p,t}^{+} \right) + \sum_{\forall a \in \mathbf{A}} \left(v_a \sum_{t=1}^{T} X_{a,t} \right) \tag{1.1}$$

Contraintes opérationnelles

La consommation et la production quotidienne de l'usine sont directement reliées au nombre de fois où chaque processus est réalisé lors de la même période.

$$QC_{p,t} = \sum_{a \in \mathbf{A} \mid \phi_{a,p} > 0} \left(X_{a,t} \times \phi_{a,p} \right) \qquad \forall p \in \mathbf{P}^{\text{consumed}}, \quad t = 1, \ldots, \mathbf{T} \tag{1.2}$$

$$QP_{p,t} = \sum_{a \in \mathbf{A} \mid \rho_{a,p} > 0} \left(X_{a,t} \times \rho_{a,p} \right) \qquad \forall p \in \mathbf{P}^{\text{produced}}, \quad t = 1, \ldots, \mathbf{T} \tag{1.3}$$

Pour chaque période t de l'horizon de planification, l'usine peut être configurée selon un et un seul mode opérationnel (1.4), et ne peut utiliser que les processus compatibles avec ce mode (1.5).

$$\sum_{f \in \mathbf{F}} Y_{f,t} \leq 1 \qquad \forall t = 1, \ldots, \mathbf{T} \tag{1.4}$$

$$0 \leq X_{a,t} \leq \left(\infty \times \sum_{f \in \mathbf{F}^a} Y_{f,t} \right) \qquad \forall a \in \mathbf{A}, \quad t = 1, \ldots, \mathbf{T} \tag{1.5}$$

où ∞ est nombre très grand

Le nombre de fois où chaque processus peut être réalisé est contraint par la capacité de chacune des machines (1.6).

$$\sum_{a \in \mathbf{A}} \left(X_{a,t} \times \delta_{a,m} \right) \leq c_{m,t} \qquad \forall m \in \mathbf{M}, \quad t = 1, \ldots, \mathbf{T} \tag{1.6}$$

Contraintes d'équilibre de flux

Les contraintes (1.7) et (1.8), conjointement avec la fonction objectif, permettent le calcul de la demande cumulée non satisfaite. Évidemment, aucun retard n'est permis pour la matière première car cela reviendrait à permettre la production d'un plan non réalisable (1.9).

$$I_{p,t} = I^{+}_{p,t} - I^{-}_{p,t} \qquad \forall p \in \mathbf{P},\ t = 1,\ldots,T \qquad (1.7)$$

$$I^{+}_{p,t} \geq 0; \quad I^{-}_{p,t} \geq 0 \qquad \forall p \in \mathbf{P},\ t = 1,\ldots,T \qquad (1.8)$$

$$I^{-}_{p,t} = 0; \qquad \forall p \in \mathbf{P}^{consumed},\ t = 1,\ldots,T \qquad (1.9)$$

Les contraintes (1.10) et (1.11) permettent d'établir la relation entre les inventaires, l'approvisionnement et la consommation de billes.

$$I_{p,1} = i_{p,0} + s_{p,1} - QC_{p,1} \qquad \forall p \in \mathbf{P}^{consumed} \qquad (1.10)$$

$$I_{p,t} = I_{p,t-1} + s_{p,t} - QC_{p,t} \qquad \forall p \in \mathbf{P}^{consumed},\ t = 2,\ldots,T \qquad (1.11)$$

Les contraintes (1.12) et (1.13) permettent d'établir la relation entre l'inventaire, la demande et la production des produits fabriqués.

$$I_{p,1} = i_{p,0} - d_{p,1} \qquad \forall p \in \mathbf{P}^{produced} \qquad (1.12)$$

$$I_{p,t} = I_{p,t-1} + QP_{p,t-1} - d_{p,t} \qquad \forall p \in \mathbf{P}^{produced},\ t = 2,\ldots,T \qquad (1.13)$$

2. Implémentation et résolution

Pour les problèmes industriels de taille réelle, ce modèle a pu être résolu en utilisant le solveur ILOG CPLEX 9.1. Des solutions quasi-optimales sont trouvées en quelques secondes.

Annexe 2 Planification des opérations de séchage

1. Le modèle

Le processus de séchage du bois est constitué de plusieurs activités séquentielles (voir section 2.2). Dans le modèle proposé, les différentes versions du processus (i.e. les différents chemins dans la figure 2.4) ne sont pas modélisées explicitement. On représente plutôt chaque activité individuellement, lesquelles peuvent être combinées dynamiquement par le modèle de manière à former les différents processus alternatifs. Les règles de connexion entre les activités (i.e. ce qui fait qu'une activité est un successeur valide pour une autre activité) sont définies par des contraintes au niveau des stocks de produits intermédiaires. Ainsi, un produit intermédiaire consommé par une activité doit d'abord être produit une activité précédente.

La Figure A2.1 illustre le fonctionnement du modèle. Nous avons différents types d'activités $a \in \mathbf{A}$. Chaque type d'activité peut être réalisée sur les machines $m \in \mathbf{M}_a \subseteq \mathbf{M}$ et possède une durée spécifique δ_a. Les paramètres $\phi_{a,p}$ et $\rho_{a,p}$ spécifient la consommation et la production de cette activité pour les produits $p \in \mathbf{P}$. Construire un plan de production consiste à déterminer quelles activités seront réalisées, quand les réaliser et avec quelles machines. Une solution peut être représentée par un diagramme de Gantt (bloc **M** sur la Figure A2.1). Chaque type d'activité peut être inséré dans le plan autant de fois que nécessaire. Une activité insérée a un impact sur les inventaires de produits $(I_{p,t})$ via sa consommation $(\phi_{a,p})$ et sa production $(\rho_{a,p})$. La demande en provenance de l'usine de rabotage $(d_{p,t})$ a également une influence sur les niveaux d'inventaires. En anglais, on refère à ce genre de modèles en utilisant les termes *timetable models* ou *time-line models* [83,84].

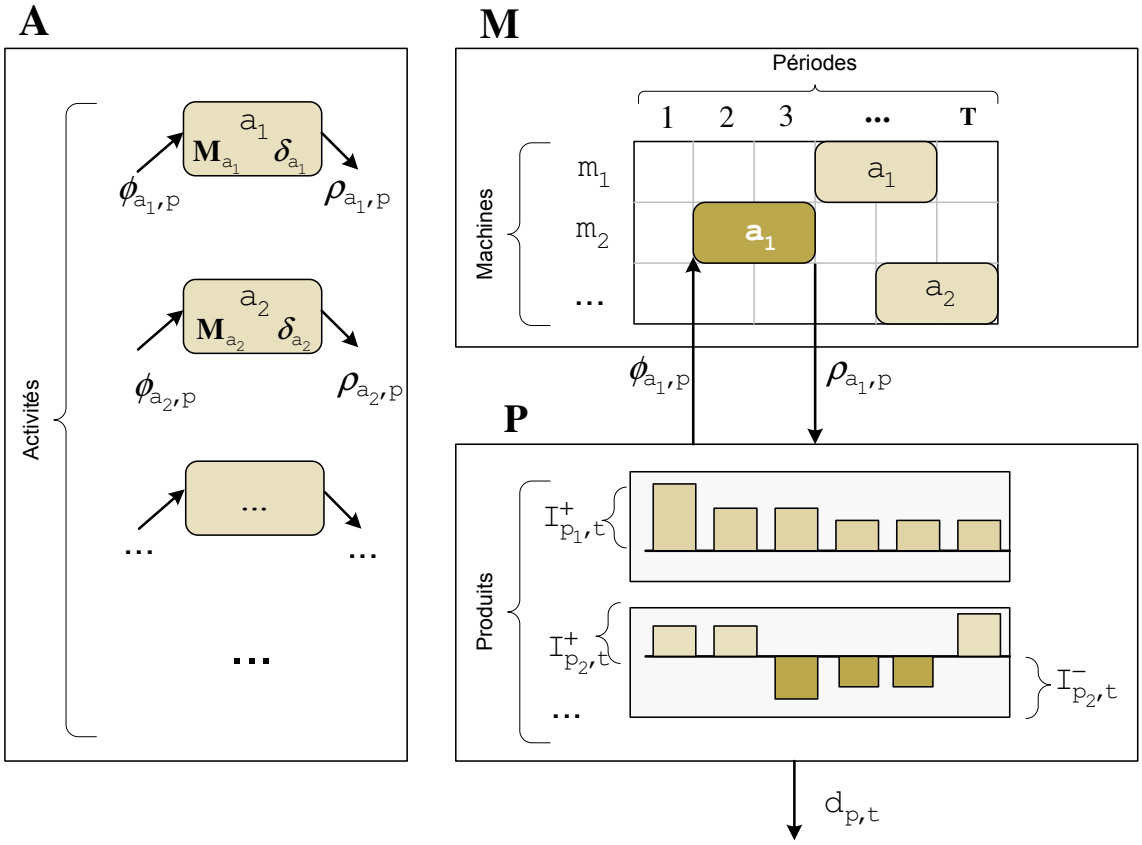


Figure A2.1 : Illustration intuitive du modèle de planification des opérations de séchage.

Ensembles

Ce modèle pour le séchage utilise une notation semblable à celle du modèle de sciage. Les ensembles suivants ont la même signification que dans le modèle précédent : périodes $(\mathtt{T})$, produits $\left(\mathbf{P},\ \mathbf{P}^{\mathtt{consumed}},\ \mathbf{P}^{\mathtt{produced}}\right)$ et machines $(\mathbf{M})$. Parce que les processus de séchage impliquent des produits intermédiaires qui sont à la fois consommés et fabriqués, les ensembles $\mathbf{P}^{\mathtt{consumed}}$ et $\mathbf{P}^{\mathtt{produced}}$ ne sont plus disjoints. Conséquemment, de manière à simplifier la présentation des contraintes d'équilibre de flux, nous considérerons chaque produit $\mathtt{p} \in \mathbf{P}$ comme une ressource pouvant être à la fois consommée, fabriquée, reçue (en approvisionnement externe) et expédiée.

Également, ces ensembles sont définis pour le modèle de séchage:

$\mathbf{A}$ ensemble des types d'activités de séchage a;

$\mathbf{A}_p^{consume}$ sous-ensemble des activités pouvant consommer le produit p. $\mathbf{A}_p^{consume} \subseteq \mathbf{A}$;

$\mathbf{A}_p^{produce}$ sous-ensemble des activités pouvant produire le produit p. $\mathbf{A}_p^{produce} \subseteq \mathbf{A}$;

$\mathbf{A}_m$ sous-ensemble des activités pouvant être réalisées sur la machine $m \in \mathbf{M}$. $\mathbf{A}_m \subseteq \mathbf{A}$;

$\mathbf{M}_a$ sous-ensemble des machines pouvant réaliser l'activité a. $\mathbf{M}_a = \{m \in \mathbf{M} \mid a \in \mathbf{A}_m\}$.

Paramètres

Les paramètres suivants ont la même signification que dans le modèle de sciage, bien qu'ils soient maintenant définis pour tous les produits $p \in \mathbf{P}$: approvisionnement $(s_{p,t})$, demande $(d_{p,t})$, inventaire initial et coûts $(i_{p,0},\ i_p,\ w_p)$, consommation et production des activités $(\phi_{a,p},\ \rho_{a,p})$. Cependant, nous supposerons que la demande $d_{p,t}$ est égal à zéro pour les produits qui peuvent être consommés. Le paramètre v_a a également la même signification que dans le modèle de sciage. Finalement, nous définissons les paramètres supplémentaires suivants :

$c_{m,t}$ égal à 1 si la machine m peut être utilisée durant la période t, 0 sinon;

δ_a nombre de périodes consécutives nécessaires pour réaliser l'activité a.

Variables

Les variables suivantes ont la même signification que dans le modèle de l'unité de sciage bien qu'elles soient maintenant définies pour tous les produits $p \in \mathbf{P}$: $QC_{p,t}$, $QP_{p,t}$, $I^+_{p,t}$, $I^-_{p,t}$, $I_{p,t}$. De plus, nous définissions les variables de décisions suivantes :

$X_{(a,m),t}$ Variable de décision binaire qui prend la valeur 1 lorsqu'une activité de type a débute sur la machine m à la période t, 0 sinon. Elle est définie pour chaque couple $(a, m) \mid a \in \mathbf{A}_m$.

Fonction objectif

La fonction objectif est semblable à celle du modèle de l'unité de sciage :

$$\text{Min} \quad \sum_{\forall p \in \mathbf{P}} \left(w_p \sum_{t=1}^{T} I^{-}_{p,t} + i_p \sum_{t=1}^{T} I^{+}_{p,t} \right) + \sum_{\forall (a,m) | a \in \mathbf{A}_m} \left(v_a \sum_{t=1}^{T} X_{(a,m),t} \right) \tag{1.14}$$

Contraintes opérationnelles

La contrainte de consommation (1.15) définie $QC_{p,t}$ comme étant égale à la consommation totale des activités débutant à la période t et consommant le produit p. La somme est définie uniquement pour les couples d'activités a et de machines m telles que m peut réaliser a, et a consomme p.

$$QC_{p,t} = \sum_{(a,m) \,\middle|\, \substack{a \in \mathbf{A}_m \\ a \in \mathbf{A}_p^{consume}}} X_{(a,m),t} \times \phi_{a,p} \qquad \forall p \in \mathbf{P}, \quad t = 1, \ldots, T \tag{1.15}$$

La contrainte de production (1.16) est la contrepartie de la contrainte précédente. Elle fixe la production totale comme étant la somme de la production des activités a se terminant pendant la période t (i.e. celles ayant débutées à la période $t - \delta_a + 1$) et fabricant le produit p.

$$QP_{p,t} = \sum_{(a,m) \,\middle|\, \substack{a \in \mathbf{A}_m \\ a \in \mathbf{A}_p^{produce} \\ t - \delta_a + 1 \geq 1}} X_{(a,m),t-\delta_a+1} \times \rho_{a,p} \qquad \forall p \in \mathbf{P}, \quad t = 1, \ldots, T \tag{1.16}$$

La contrainte de capacité (1.17) spécifie que le nombre d'activités en cours sur une machine m à la période t doit être inférieur ou égale à 1. Pour chaque type d'activité a, une instance est en cours à la période t si l'une de celle-ci a démarré dans l'intervalle $[t - \delta_a + 1, \ldots, t]$.

$$\sum_{a \in \mathbf{A}_m} \sum_{\tau = \max[t-\delta_a+1,\, 1]}^{t} X_{(a,m),\tau} \leq C_{m,t} \qquad \forall m \in \mathbf{M}, \quad t = 1, \ldots, T \tag{1.17}$$

Contraintes d'équilibre de flux

Les contraintes d'équilibre de flux sont similaires à celles présentées pour le modèle de l'unité de sciage, à l'exception des produits considérés comme des ressources pouvant être consommées et produites.

$$I_{p,t} = I^{+}_{p,t} - I^{-}_{p,t} \qquad \forall p \in \mathbf{P},\ t = 1, \ldots, T \qquad (1.18)$$

$$I^{+}_{p,t} \geq 0; \quad I^{-}_{p,t} \geq 0 \qquad \forall p \in \mathbf{P},\ t = 1, \ldots, T \qquad (1.19)$$

$$I^{-}_{p,t} = 0; \qquad \forall p \in \mathbf{P}^{consumed},\ t = 1, \ldots, T \qquad (1.20)$$

$$I_{p,1} = i_{p,0} + s_{p,1} - QC_{p,1} - d_{p,1} \qquad \forall p \in \mathbf{P} \qquad (1.21)$$

$$I_{p,t} = I_{p,t-1} + s_{p,t} + QP_{p,t-1} - QC_{p,t} - d_{p,t} \qquad \forall p \in \mathbf{P},\ t = 2, \ldots, T \qquad (1.22)$$

2. Implémentation et résolution

Pour des problèmes industriels de taille réelle, de bonnes solutions ne peuvent être obtenues en un temps raisonnables. L'heuristique glouton suivante peut être utilisée pour résoudre ce problème.

Premièrement, une liste des processus de séchage est établie a priori (i.e. chaque chemin sur la figure 2.4). Ensuite, le plan est obtenu de manière incrémentale (en partant d'un plan vide) en réalisant les étapes suivantes :

1. Calculer la valeur de la fonction objectif pour le plan courant.
2. Insérer dans le plan les activités du processus permettant de réduire au maximum la valeur de la fonction objectif. Chaque processus potentiel est évalué comme suit :
    a. Calculer la date la plus tôt où l'un des produits fabriqués par ce processus est en pénurie en vertu du plan actuel.
    b. En considérant celle-ci comme la date cible, céduler les activités de la dernière à la première en appliquant une politique juste en temps.
    c. Calculer l'amélioration engendrée par le processus en observant la nouvelle valeur de la fonction objectif.

3. Retourner à l'étape 1 (l'algorithme termine lorsqu'il est impossible d'insérer un processus tout en réduisant la valeur de la fonction objectif).

On peut générer des solutions alternatives en procédant de la manière suivante : à l'étape 2, l'évaluation des processus est destinée à choisir le meilleur processus à insérer; on peut alors trier les processus ayant des contributions positives en ordre décroissant de contribution.

La décision d'insertion d'un processus dans le plan correspond alors à un point de choix : insérer le meilleur, ou le deuxième meilleur, ou … etc. En considérant un tel point de choix pour toutes les itérations de la procédure, on se trouve à définir un arbre de recherche. La première feuille de cet arbre correspond à la solution qui serait obtenue avec l'heuristique glouton. En choisissant une stratégie de retour arrière (DFS ou LDS, par exemple) on peut explorer des solutions alternatives.

Annexe 3 Planification des opérations de finition

1. Le modèle

En pratique, les trois opérations de finition (rabotage, classement et éboutage) sont réalisées de manière séquentielle sur une seule et même ligne de production. Du point de vue de la planification, cette ligne peut être considérée comme une seule et même machine. La productivité de cette machine virtuelle sera déterminée par l'opération qui constitue le goulot d'étranglement.

Tel que mentionné à la section 2.3, un plan de production pour cette unité est constitué d'une série de campagnes (figure 2.5). Chacune est associée à une famille de produits (e.g. 2"x4") qui nécessite la configuration de l'usine selon un certain mode. Les changements de campagne doivent survenir entre les périodes de travail. Pendant une campagne, des produits bruts de longueurs différentes peuvent être transformés (e.g. 2"x4"-8', 2"x4"-10') mais les longueurs doivent être traitées selon un ordre spécifique.

Dans le modèle suivant, des variables de décisions binaires $(Y_{f,t})$ spécifient comment l'usine doit être configurée à chaque période. Un coût de mise en course (ζ) doit être considéré lorsqu'une nouvelle campagne débute. D'autres variables de décisions spécifient les quantités de produits de chaque longueur (e.g. 8', 10') devant être traitées lors de chaque période – plutôt que par campagne tel qu'exigé par l'énoncé original du problème. Pour compenser cette apparente relaxation des contraintes du problème, des contraintes supplémentaires sont introduites de manière à considérer que toute la consommation survient au début de la campagne, et que toute la production survient à la fin. Le modèle doit donc tenir la trace de deux niveaux d'inventaires par produit : l'un dans la cour (comme pour les unités de sciage et de séchage) et l'autre dans l'usine.

Ensembles

Le modèle pour l'unité de finition partage des éléments de notation avec les modèles précédents. Les ensembles suivants ont la même signification : $\mathtt{T}$, $\mathbf{P}$, $\mathbf{P}^{\mathtt{consumed}}$, and $\mathbf{P}^{\mathtt{produced}}$. Similairement au modèle du sciage, chaque $\mathtt{f} \in \mathbf{F}$ définit un mode dans lequel l'usine de rabotage peut être configurée. Chaque mode correspond à une famille de produits (e.g. 2″x4″). De plus, les ensembles suivants sont définis :

$\mathbf{P}_{\mathtt{f}}^{\mathtt{consumed}}$ produits bruts $\mathtt{p}$ pouvant être consommés lorsque l'usine est configurée selon le mode $\mathtt{f}$. $\mathtt{p} \in \mathbf{P}_{\mathtt{f}}^{\mathtt{consumed}} \subseteq \mathbf{P}^{\mathtt{consumed}}$;

$\mathbf{P}_{\mathtt{f}}^{\mathtt{produced}}$ produits finis $\mathtt{p}$ pouvant être fabriqués lorsque l'usine est configurée selon le mode $\mathtt{f}$. $\mathtt{p} \in \mathbf{P}_{\mathtt{f}}^{\mathtt{produced}} \subseteq \mathbf{P}^{\mathtt{produced}}$;

$\mathbf{FR}$ ensemble des couples $(\mathtt{f}, \mathtt{r}) \,\big|\, (\mathtt{f} \in \mathbf{F}) \wedge \left(\mathtt{r} \in \mathbf{P}_{\mathtt{f}}^{\mathtt{consumed}}\right)$.

Paramètres

Les paramètres suivants ont la même signification que dans le modèle de l'unité de sciage : $\mathtt{s}_{\mathtt{p},\mathtt{t}}$, $\mathtt{d}_{\mathtt{p},\mathtt{t}}$, $\mathtt{i}_{\mathtt{p},0}$, $\mathtt{i}_{\mathtt{p}}$, et $\mathtt{w}_{\mathtt{p}}$. Les paramètres additionnels suivants sont définis :

$\rho_{(\mathtt{f},\mathtt{r}),\mathtt{p}}$ volume du produit $\mathtt{p} \in \mathbf{P}^{\mathtt{produced}}$ obtenu lorsqu'une unité de volume du produit $\mathtt{r} \in \mathbf{P}_{\mathtt{f}}^{\mathtt{consumed}}$ est consommé lorsque l'usine est configure dans le mode $\mathtt{f}$. Défini pour les couples $(\mathtt{f}, \mathtt{r}) \in \mathbf{FR}$;

$\delta_{(\mathtt{f},\mathtt{r})}$ temps nécessaire à la consommation d'une unité de volume du produit $\mathtt{r} \in \mathbf{P}^{\mathtt{consumed}}$ lorsque l'usine est configurée selon le mode $\mathtt{f}$. Défini pour les couples $(\mathtt{f}, \mathtt{r}) \in \mathbf{FR}$;

$\mathtt{v}_{(\mathtt{f},\mathtt{r})}$ coût associé au traitement d'une unité de volume du produit $\mathtt{r} \in \mathbf{P}^{\mathtt{consumed}}$. Défini pour les couples $(\mathtt{f}, \mathtt{r}) \in \mathbf{FR}$;

$\mathtt{c}_{\mathtt{t}}$ capacité de l'usine (en unîtes de temps) pour la période $\mathtt{t}$;

ζ coût de mise en course associé à un changement de mode.

Variables

Les variables qui suivent ont la même signification que dans le modèle de l'unité de sciage:. La variable $Y_{f,t}$ identifie dans quel mode l'usine est configurée. Les variables $I^{+}_{p,t}$, $I^{-}_{p,t}$ et $I_{p,t}$ correspondent aux inventaires dans la cour. La variable $QC_{p,t}$ indique la quantité transférée de la cour à bois vers l'usine au début d'une campagne. La variable $QP_{p,t}$ à la quantité transférée de l'usine à la cour à la fin de la campagne.

De plus, les variables suivantes sont définies :

$BS_{f,t}$ égale à 1, si l'usine est configurée dans le mode f pour la période t et que ce n'était pas le cas pour la période t-1. Égale à 0 sinon.

BS_{t} égale à 1 si une campagne (pour n'importe quel mode) débute à la période t. Égale à 0 sinon;

BE_{t} égale à 1 si une campagne (pour n'importe quel mode) termine à la période t. Égale à 0 sinon;

$UC_{(f,r),t}$ volume de produit r à transformer pendant la période t alors que l'usine est configurée selon le mode f. Le produit doit déjà se trouver dans l'usine. Défini pour les couples $(f, r) \in \mathbf{FR}$.

$UP_{p,t}$ volume de produit p fabriqué à la période t. Nous considérons que le produit demeure dans l'usine jusqu'à la fin de la campagne. Définit pour les produits $p \in \mathbf{P}^{produced}$;

$UI_{p,t}$ volume de produit p dans l'usine à la fin de la période t. Défini pour tous les produits $p \in \mathbf{P}$.

Fonction objectif

La fonction objectif est semblable à celle définie pour le modèle de l'unité de sciage. Les coûts de production variables dépendent de la matière première consommée mais également des coûts de mise en course.

$$\text{Min} \quad \sum_{\forall p \in \mathbf{P}^{\text{produced}}} \left(w_p \sum_{t=1}^{T} I_{p,t}^{-} \right) + \sum_{\forall p \in \mathbf{P}} \left(i_p \sum_{t=1}^{T} I_{p,t}^{+} \right) + \zeta \sum_{t=1}^{T} BS_t + \sum_{\forall (f,r) \in \mathbf{FR}} \left(v_{(f,r)} \sum_{t=1}^{T} UC_{(f,r),t} \right) \tag{1.23}$$

Contraintes opérationnelles

Premièrement, l'usine peut être configurée dans un seul mode à la fois (1.24) et un produit ne peut être traité (consommé) que si l'usine est configurée dans un mode compatible (1.25).

$$\sum_{f \in \mathbf{F}} Y_{f,t} \leq 1 \qquad \forall t = 1, \ldots, T \tag{1.24}$$

$$0 \leq UC_{(f,r),t} \leq (\infty \; Y_{f,t}) \qquad \forall (f, r) \in \mathbf{FR}, \; t = 1, \ldots, T \tag{1.25}$$

where ∞ is a significantly large number.

Les contraintes (1.26) à (1.28) spécifient qu'une campagne ne peut débuter ou terminer que si l'usine est configurée dans un mode compatible. La contrainte (1.29) nous assure que la campagne se déroule sans en interruption entre le moment de son début et sa fin.

$$BS_{f,1} = Y_{f,1} \qquad \forall f \in \mathbf{F} \tag{1.26}$$

$$BS_{f,t} \leq Y_{f,t} \qquad \forall f \in \mathbf{F}, \; t = 1, \ldots, T \tag{1.27}$$

$$BE_{f,t} \leq Y_{f,t} \qquad \forall f \in \mathbf{F}, \; t = 1, \ldots, T \tag{1.28}$$

$$Y_{f,t} = Y_{f,t-1} - BE_{f,t-1} + BS_{f,t} \qquad \forall f \in \mathbf{F}, \; t = 2, \ldots, T \tag{1.29}$$

Les variables BS_t et BE_t prennent respectivement la valeur 1 si et seulement si une campagne débute (1.30) ou se termine (1.31) à la période t.

$$BS_t = \sum_{f \in \mathbf{F}} BS_{f,t} \qquad \forall t = 1, \ldots, T \tag{1.30}$$

$$BE_t = \sum_{f \in \mathbf{F}} BE_{f,t} \qquad \forall t = 1, \ldots, T \tag{1.31}$$

Les produits bruts peuvent entrer dans l'usine uniquement au début d'une campagne compatible (1.32). Les produits finis sortent de l'usine uniquement à la fin de la campagne (1.33). Aucun produit ne peut être laissé dans l'usine à la fin d'une campagne (1.34).

$$QC_{p,t} \leq \left(\infty \sum_{f \in \mathbf{F} \mid p \in \mathbf{P}_f^{consumed}} BS_{f,t} \right) \qquad \forall p \in \mathbf{P},\ t = 1, \ldots, T \qquad (1.32)$$

$$QP_{p,t} \leq \left(\infty \sum_{f \in \mathbf{F} \mid p \in \mathbf{P}_f^{produced}} BE_{f,t} \right) \qquad \forall p \in \mathbf{P},\ t = 1, \ldots, T \qquad (1.33)$$

$$UI_{p,t} \leq \infty \left(1 - BE_t\right) \qquad \forall p \in \mathbf{P},\ t = 1, \ldots, T \qquad (1.34)$$

Les contraintes suivantes assurent l'équilibre des flux à l'intérieur de l'usine pour les produits bruts :

$$UI_{p,1} = QC_{p,1} - \sum_{f \in F \mid (f,p) \in \mathbf{FR}} \left(UC_{(f,p),1} \right) \qquad \forall p \in \mathbf{P}^{consumed} \qquad (1.35)$$

$$UI_{p,t} = UI_{p,t-1} + QC_{r,t} - \sum_{f \in F \mid (f,p) \in \mathbf{FR}} \left(UC_{(f,p),t} \right) \qquad \forall p \in \mathbf{P}^{consumed},\ t = 1, \ldots, T \qquad (1.36)$$

$$UI_{p,t} \geq 0 \qquad \forall p \in \mathbf{P}^{consumed},\ t = 1, \ldots, T \qquad (1.37)$$

Les contraintes suivantes assurent l'équilibre des flux à l'intérieur de l'usine pour les produits finis :

$$UI_{p,t} = UI_{p,t-1} + UP_{p,t} - QP_{p,t} \qquad \forall p \in \mathbf{P}^{produced},\ t = 2, \ldots, T \qquad (1.38)$$

$$UI_{p,t} \geq 0 \qquad \forall p \in \mathbf{P}^{consumed},\ t = 1, \ldots, T \qquad (1.39)$$

Cette contrainte établit la relation entre la consommation et la production à l'intérieur de l'usine :

$$UP_{p,t} = \sum_{(f,r) \in \mathbf{FR}} UC_{(f,r),t} \times \rho_{(f,r),p} \qquad \forall p \in \mathbf{P},\ t = 1, \ldots, T \qquad (1.40)$$

Finalement, la capacité de l'usine doit être respectée:

$$\sum_{(f,r) \in \mathbf{FR}} \left(UC_{(f,r),t} \times \delta_{(f,r)} \right) \leq c_t \qquad \forall t = 1, \ldots, T \qquad (1.41)$$

Contraintes d'équilibre de flux (cour à bois)

Les contraintes d'équilibre de flux pour la cours sont les mêmes que celles définies pour l'unité de sciage. Nous réutilisons donc les contraintes (1.7) à (1.13) pour établir la relation entre $i_{p,0}$, $I_{p,t}$, $I^+_{p,t}$, $I^-_{p,t}$, $s_{p,1}$, $d_{p,t}$, $QC_{p,1}$ et $QP_{p,t}$.

2. Implémentation et résolution

Tout comme pour le problème de planification de l'unité de séchage, un solveur MIP ne permettait pas d'obtenir de bonnes solutions en un temps raisonnable pour les instances de taille industrielles. L'heuristique glouton suivante permet d'obtenir de bonnes solutions en un temps raisonnable.

Le plan est bâti de manière incrémentale en allant de la première période à la dernière. Lorsque la ligne de finition devient disponible, on démarre une campagne pour la famille de produits $f \in \mathbf{F}$ pour laquelle il est le plus urgent de démarrer la production (compte tenu des dates de livraisons et des délais de production). Conséquence des coûts de mise en course, il est souhaitable que cette campagne ait la plus longue durée possible, ce qui nous incite à satisfaire le plus grand nombre de commandes futures possibles). Cependant, la campagne doit être terminée lorsque la prochaine date de livraison est rencontrée. Nous devons aussi laisser du temps libre pour la production associée à d'autres familles. Voici le pseudocode de la procédure proposée :

1. Soit t la première période pour laquelle la ligne de finition est disponible (aucune campagne n'est cédulée pour cette période) et pour laquelle des produits bruts sont disponibles à la transformation.
2. Pour chaque mode $f \in \mathbf{F}$:
    a. Soit t_f la première période où une commande pour l'un des produits $p \in \mathbf{P}_f^{produced}$ n'est pas satisfaite compte tenu du plan de production actuel.
    b. Supposons une campagne a, terminant à $e_t = t_f - 1$ et permettant de satisfaire toute la demande pour tous les produits $p \in \mathbf{P}_f^{produced}$ à la période t_f. Soit s_f la période de début de a compte tenu du temps de production nécessaire (pour l'instant, nous ne considérons pas la question de la disponibilité de la matière première).

c. S'il n'y a pas de matière première $p \in \mathbf{P}_f^{consumed}$ disponible à la période s_f, incrémenter s_f jusqu'à ce qu'une quantité quelconque le doit (e_f demeure inchangé).

3. Trier les modes f en ordre croissant de s_f.
4. En considérant les modes $f \in \mathbf{F}$ pour lesquels de la matière première $p \in \mathbf{P}_f^{consumed}$ est disponible à la période t, choisir celui avec le plus petit s_f. Insérer dans le plan une campagne pour ce mode:
   a. Elle débute à la période t.
   b. Elle a la plus longue durée possible (en tenant compate de la matière disponible) mais sans dépasser t_f (prochaine livraison pour la famille courante) ni le prochain s_f dans le vecteur défini à l'étape 3 (date de début de la prochaine campagne pour la seconde famille qu'il est le plus urgent de produire).
   c. Le choix des produits à consommer de même que les quantités sont établis de la manière suivante :
      i. On prévoir d'abord les quantités de produit $p \in \mathbf{P}_f^{produced}$ nécessaires à la satisfaction de la prochaine commande non satisfaite pout cette famille.
      ii. Si de l'espace est disponible, on consomme les quantités pour la prochaine commande de cette famille, etc.
5. Retourner à l'étape 1.

Comme pour le problème de planification de l'unité de séchage, nous pouvons utiliser une stratégie de recherche pour générer des solutions alternatives. L'étape 4 de la procédure ci-haut peut être considérée comme un point de choix : des campagnes pour différentes familles peuvent être insérées dans le plan. En considérant ces alternatives à chaque itération de la procédure, nous définissons un arbre de recherche. Une stratégie de type DFS ou LDS peut être utilisée pour chercher dans cet arbre.