Ordonnancement d'atelier avec contraintes temporelles entre opérations

Freddy Deppner

Institut National
Polytechnique de Lorraine

Département de formation doctorale en informatique

École doctorale IAEM Lorraine

Ordonnancement d'atelier avec contraintes temporelles entre opérations

THÈSE

présentée et soutenue publiquement le 6 mai 2004

pour l'obtention du

Doctorat de l'Institut National Polytechnique de Lorraine

(spécialité informatique)

par

Freddy DEPPNER

Composition du jury

Rapporteurs :	Lionel Dupont	Professeur à l'Ecole des Mines d'Albi-Carmaux
	Michel Gourgand	Professeur à l'ISIMA de Clermont-Ferrand
Examinateurs :	Alain Guinet	Professeur à l'INSA de Lyon
	Alexander Bockmayr	Professeur à l'Université Henri Poincaré, Nancy I
Directeur de Thèse :	Marie-Claude Portmann	Professeur à l'Institut National Polytechnique de Lorraine

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Remerciements

Comme il est d'usage, je me dois de débuter ce manuscrit par les habituels remerciements aux personnes et aux entités qui, de près ou de loin, ont permis l'aboutissement d'un si long travail. Même si cette thèse est avant tout le fruit d'un investissement personnel, je me rends compte rétrospectivement que ce résultat n'aurait pas été possible sans l'intervention de nombreuses personnes qu'il m'est impossible de toutes citer ici et je les prie de m'en excuser.

Merci tout d'abord au rectorat de Strasbourg et plus particulièrement à M. Bohn, Directeur Adjoint des Ressources Humaines, qui me libéra de mes obligations professorales pour me permettre de débuter le plus rapidement possible mon travail au sein de la société Incotec. Merci ensuite à M. Hentzler, PDG de la société Incotec, pour qui cette première convention CIFRE fut une découverte et une aventure pas toujours facile. Merci également aux nombreux collègues au sein d'Incotec dont je salue les compétences et dont les conseils me furent bien utiles tout au long de mon travail. Je me dois de remercier la Direction de la Formation Doctorale Informatique du LORIA, et plus particulièrement M. Karl Trombe et Mme. Nadine Beurné, pour la compréhension dont ils ont su faire preuve au cours de ces quatre années. Mes remerciements s'adressent également à messieurs Dupont et Gourgand, pour avoir accepté de rapporter sur ce travail, et messieurs Bockmayr et Guinet, pour avoir accepté d'examiner ce travail et de faire partie du jury, malgré leurs emplois du temps que j'imagine bien remplis.

Je me permets de débuter un nouveau paragraphe pour souligner la place prépondérante qu'est réservée à Mlle. Portmann qui n'hésita pas à sacrifier nombre de ses week-ends afin que nous puissions travailler ensemble. Je la remercie pour sa très grande disponibilité, son aide et ses encouragements dans les moments de doutes. Sans son soutien, cette thèse n'aurait jamais pu arriver à son terme. Merci également à toute son équipe, en particulier à Julien Fondrevelle pour son aide précieuse.

Un énorme merci à mes amis qui m'ont soutenu durant toutes ses années, à Fabienne pour sa relecture attentive. Merci enfin à mes proches et plus particulièrement à ma femme Sylviane et ma fille Floriane qui ont beaucoup souffert de l'indisponibilité de leur mari et père. Un dernier merci à Aurélien qui a eu la délicatesse d'attendre le 2 janvier 2004 pour venir au monde et permettre à son papa de terminer sereinement la rédaction de sa thèse.

Table des matières

Introduction générale **1**

Chapitre 1 Motivations, Définitions et Notations **3**

1.1 Motivations 4
1.1.1 L'ordonnancement et ses domaines d'application 4
1.1.2 Les contraintes considérées 5
1.1.3 Contexte de la thèse 5
1.2 Exemples industriels 6
1.2.1 Successions sans attente 6
1.2.2 Successions en un laps de temps défini 7
1.2.3 Contrôle 8
1.2.4 EducRoute 9
1.2.5 Hoist Scheduling 11
1.2.6 Ordonnancement de tâches informatiques 12
1.3 Etude de cas - PharmaCorp 13
1.3.1 Gamme pharmaceutique et système de production associé 13
1.3.2 Objectifs de PharmaCorp 14
1.4 Bilan des études de cas 15
1.5 Etat de l'art 16
1.6 Notations 17
1.6.1 Cas particulier des gammes linéaires 17
1.6.2 Gammes quelconques 18
1.7 Modélisation des ressources et des calendriers 18
1.8 Graphe disjonctif 20
1.9 Les différents types d'ordonnancement 21
1.10 Méthodes de résolution 23
1.10.1 Mesure de la qualité des méthodes de résolution 24

1.10.2 Processus de conception de méthodes de résolution 24
1.10.3 Littérature sur les méthodes de résolution 27
1.11 Contexte des expérimentations 28
1.12 Contenu et structure de la thèse 29

Chapitre 2 Construction d'une solution faisable **33**
2.1 Introduction 34
2.2 Algorithmes de construction 36
2.2.1 Problème sans écarts maximaux 36
2.2.2 Problème avec écarts maximaux 38
2.2.3 Phénomène du «saute-mouton» et non convergence 40
2.2.4 Décomposition par grappes 42
2.3 Convergence des algorithmes sous certaines hypothèses 44
2.4 Expériences sur des gammes linéaires 50
2.4.1 Génération des exemples et méthodes utilisées 50
2.4.2 Analyse des résultats 53
2.4.3 Décalage 57
2.5 PharmaCorp 58
2.5.1 Ressources 58
2.5.2 Résultats 59
2.6 Conclusion 61

Chapitre 3 Problèmes d'atelier de type flowshop **63**
3.1 Introduction 64
3.2 Dominance des flowshops de permutation 65
3.3 Algorithmes génétiques 69
3.3.1 Principes généraux 69
3.3.2 Codage et évaluation des individus 70
3.3.3 Opérateurs de croisement 71
3.3.4 Qualité des Opérateurs de croisement 76
3.4 Première série d'expériences 78
3.4.1 Génération des jeux d'essai 78
3.4.2 Analyse des résultats 79
3.5 Autodétermination 81
3.5.1 Principe 81

3.5.2 Evolution des compteurs N_i 83
3.5.3 Autodétermination vs méthode classique 84
3.6 Mutation 90
3.7 Conclusion 91

Chapitre 4 Algorithmes génétiques et amélioration par voisinage **93**
4.1 Introduction 94
4.2 Etude sur les principaux voisinages 94
4.2.1 Sans écarts maximaux 94
4.2.2 Avec écarts maximaux 100
4.3 Algorithme Génétique et Recherche Locale 102
4.4 Expériences 102
4.4.1 Approche retenue 102
4.4.2 Méthode exacte de Fondrevelle ([Fon03]) 104
4.4.3 Expériences sur les exemples de Taillard 5x20 et 10x20 108
4.5 Conclusion 109

Conclusion et perspectives **111**
1 Conclusion 111
2 Perspectives 112

Annexes **115**
1 Annexe du chapitre 1 115
2 Annexe du chapitre 2 118
3 Annexe du chapitre 3 125
3.1 Qualité des opérateurs de croisement - jeux d'essai 10x30 125
3.2 Qualité des opérateurs de croisement - jeux d'essai 5x20 133
3.3 Autodétermination vs 1X - Jeux d'essai de taille 10x30 140
3.4 Autodétermination vs 1X - Jeux d'essai de taille 5x20 142
3.5 Autodétermination vs TSXD - Jeux d'essai de taille 10x30 143
3.6 Autodétermination vs TSXD - Jeux d'essai de taille 5x20 144
3.7 Autodétermination vs 2X - Jeux d'essai de taille 10x30 145
3.8 Autodétermination vs 2X - Jeux d'essai de taille 5x20 146
3.9 Autodétermination vs 1X-TSXD - Jeux d'essai de taille 10x30 . . . 147
3.10 Autodétermination vs 1X-TSXD - Jeux d'essai de taille 5x20 148

3.11 Autodétermination vs Autodétermination avec mutation 149
3.12 1X vs 1X avec mutation 151
3.13 TSXD vs TSXD avec mutation 152
4 Annexe du chapitre 4 153

Bibliographie **157**

Index **165**

Table des figures

1.1 Succession immédiate entre la préparation et l'exécution 7
1.2 Couchage du papier thermique 7
1.3 Opérations de contrôle 9
1.4 Contrôles réguliers au cours d'une opération de tournage 9
1.5 Educroute - opérations synchrones 10
1.6 Educroute - enchaînement des sous-modules 11
1.7 La Poste - Job de sauvegarde 12
1.8 Profil de capacité d'une ressource 19
1.9 Profil de capacité d'une ressource multiple 19
1.10 Graphe disjonctif 21
1.11 Graphe disjonctif arbitré ; graphe potentiels-tâches 22
1.12 Solution valide sans attente 22
1.13 Etat de l'art - vue synthétique 31

2.1 Solution non valide obtenue par AOS-MD en l'absence de H0 41
2.2 Solution dégradée de l'algorithme ARP-MD 41
2.3 Déroulement du phénomène de saute-mouton 42
2.4 Nombre moyen de calculs de dates d'exécution m=10, n=20, $\varepsilon = 5.36$ . . . 54
2.5 Makespan moyen m=10, n=20, $\varepsilon = 5.36$ (ensemble des instances) 54
2.6 Nombre de solutions valides obtenues m=10, n=20, $\varepsilon = 5.36$ 54
2.7 Makespan moyen m=10, n=20, $\varepsilon = 5.36$ (solutions valides uniquement) . . 55
2.8 Nombre moyen de calculs de dates d'exécution m=10, n=20, $\varepsilon = 13.1$ . . . 55
2.9 Nombre de solutions valides obtenues m=10, n=20, $\varepsilon = 13.1$ 55
2.10 ARP-MD : Evolution du makespan moyen pour différents décalages m=10, n=20 58
2.11 AOS-MD : Evolution du makespan moyen pour différents décalages m=10, n=20, $\varepsilon = 3.9$ 58
2.12 Gantt jeu d'essai PharmaCorp obtenu avec ARP-G 61
2.13 Gantt jeu d'essai PharmaCorp obtenu avec ARP-MD 61

3.1 Ordonnancement optimal 66
3.2 Ordonnancement optimal pour l'ensemble des flowshops 66

3.3 Ordonnancement optimal sur l'ensemble des flowshop de permutation . . . 67
3.4 Solution minimisant le C_{max} 68
3.5 Solution avec flowshop de permutation 68
3.6 Compteurs 77
3.7 Evolution des compteurs N_i - jeux d'essai 10x30 sans écarts maximaux . . 85
3.8 Evolution des compteurs N_i - jeux d'essai 10x30 avec écarts maximaux . . 85
3.9 Evolution des compteurs N_i - jeux d'essai 10x30 sans attente 85
3.10 Evolution des compteurs N_i - jeux d'essai 5x20 sans écarts maximaux . . . 86
3.11 Evolution des compteurs N_i - jeux d'essai 5x20 avec écarts maximaux . . 86
3.12 Evolution des compteurs N_i - jeux d'essai 5x20 sans attente 86
3.13 Auto vs 1X - jeux d'essai 10x30 sans écarts maximaux - résultats en tenant compte du meilleur individu 87
3.14 Auto vs 1X - jeux d'essai 10x30 sans écarts maximaux - résultats en tenant compte des 10 meilleurs individus 88

4.1 Ordonnancement valides S et S' 98
4.2 Graphe disjonctif arbitré $G(S)$ 99

A.1.1 Gamme du processus de lyophilisation PharmaCorp 116
A.1.2 Système de production PharmaCorp 117
A.2.1 Résultats numériques Jobshop m=10 n=30 $\varepsilon = 5.34$ 119
A.2.2 Résultats numériques Jobshop m=10 n=20 $\varepsilon = 5.36$ 120
A.2.3 Résultats numériques Jobshop m=10 n=10 $\varepsilon = 3.79$ 121
A.2.4 Résultats numériques Jobshop m=5 n=20 $\varepsilon = 4.35$ 122
A.2.5 Résultats numériques Jobshop m=5 n=10 $\varepsilon = 5.77$ 123
A.2.6 Résultats numériques Jobshop m=5 n=5 $\varepsilon = 2.20$ 124
A.3.1 Evolution du compteur E1 - jeux d'essai 10x30 sans écarts maximaux . . 129
A.3.2 Evolution du compteur E1 - jeux d'essai 10x30 avec écarts maximaux . . 129
A.3.3 Evolution du compteur E1 - jeux d'essai 10x30 sans attente 130
A.3.4 Evolution du compteur E1 en fonction de la tension sur les écarts maximaux 130
A.3.5 Evolution du makespan moyen au cours des cycles pour les jeux d'essai sans écarts maximaux de taille 10x30 131
A.3.6 Evolution du makespan moyen au cours des cycles pour les jeux d'essai avec écarts maximaux de taille 10x30 131
A.3.7 Evolution du makespan moyen au cours des cycles pour les jeux d'essai sans attente de taille 10x30 132
A.3.8 Evolution du compteur E1 - jeux d'essai 5x20 sans écarts maximaux . . . 136
A.3.9 Evolution du compteur E1 - jeux d'essai 5x20 avec écarts maximaux . . . 136
A.3.10 Evolution du compteur E1 - jeux d'essai 5x20 sans attente 137

A.3.11 Evolution du compteur E1 en fonction de la tension sur les écarts maximaux . . . 137
A.3.12 Evolution du makespan moyen au cours des cycles pour les jeux d'essai sans écarts maximaux de taille 5x20 . . . 138
A.3.13 Evolution du makespan moyen au cours des cycles pour les jeux d'essai avec écarts maximaux de taille 5x20 . . . 138
A.3.14 Evolution du makespan moyen au cours des cycles pour les jeux d'essai sans attente de taille 5x20 . . . 139
A.3.15 Autodétermination vs 1X - 10x30 sans écarts maximaux . . . 141
A.3.16 Autodétermination vs 1X - 10x30 avec écarts maximaux . . . 141
A.3.17 Autodétermination vs 1X - 10x30 sans attente . . . 141
A.3.18 Autodétermination vs 1X - 5x20 sans écarts maximaux . . . 142
A.3.19 Autodétermination vs 1X - 5x20 avec écarts maximaux . . . 142
A.3.20 Autodétermination vs 1X - 5x20 sans attente . . . 142
A.3.21 Autodétermination vs TSXD - 10x30 sans écarts maximaux . . . 143
A.3.22 Autodétermination vs TSXD - 10x30 avec écarts maximaux . . . 143
A.3.23 Autodétermination vs TSXD - 10x30 sans attente . . . 143
A.3.24 Autodétermination vs TSXD - 5x20 sans écarts maximaux . . . 144
A.3.25 Autodétermination vs TSXD - 5x20 avec écarts maximaux . . . 144
A.3.26 Autodétermination vs TSXD - 5x20 sans attente . . . 144
A.3.27 Autodétermination vs 2X - 10x30 sans écarts maximaux . . . 145
A.3.28 Autodétermination vs 2X - 10x30 avec écarts maximaux . . . 145
A.3.29 Autodétermination vs 2X - 10x30 sans attente . . . 145
A.3.30 Autodétermination vs 2X - 5x20 sans écarts maximaux . . . 146
A.3.31 Autodétermination vs 2X - 5x20 avec écarts maximaux . . . 146
A.3.32 Autodétermination vs 2X - 5x20 sans attente . . . 146
A.3.33 Autodétermination vs 1X-TSXD - 10x30 sans écarts maximaux . . . 147
A.3.34 Autodétermination vs 1X-TSXD - 10x30 avec écarts maximaux . . . 147
A.3.35 Autodétermination vs 1X-TSXD - 10x30 sans attente . . . 147
A.3.36 Autodétermination vs 1X-TSXD - 5x20 sans écarts maximaux . . . 148
A.3.37 Autodétermination vs 1X-TSXD - 5x20 avec écarts maximaux . . . 148
A.3.38 Autodétermination vs 1X-TSXD - 5x20 sans attente . . . 148
A.3.39 Autodétermination avec et sans mutation - 5x20 sans écarts maximaux . 149
A.3.40 Autodétermination avec et sans mutation - 5x20 avec écarts maximaux . 149
A.3.41 Autodétermination avec et sans mutation - 5x20 sans attente . . . 149
A.3.42 Autodétermination avec et sans mutation - 10x30 sans écarts maximaux 150
A.3.43 Autodétermination avec et sans mutation - 10x30 avec écarts maximaux 150
A.3.44 Autodétermination avec et sans mutation - 10x30 sans attente . . . 150
A.3.45 1X avec et sans mutation - 5x20 sans écarts maximaux . . . 151
A.3.46 1X avec et sans mutation - 5x20 avec écarts maximaux . . . 151

A.3.47 1X avec et sans mutation - 5x20 sans attente 151
A.3.48 TSXD avec et sans mutation - 5x20 sans écarts maximaux 152
A.3.49 TSXD avec et sans mutation - 5x20 avec écarts maximaux 152
A.3.50 TSXD avec et sans mutation - 5x20 sans attente 152

Liste des Algorithmes

2.1 ARP - Algorithme à base de règles de priorité 36
2.2 AOS - Algorithme d'ordre strict 38
2.3 ARP-MD - ARP avec gestion des écarts maximaux 39
2.4 AOS-MD - AOS avec gestion des écarts maximaux 40
2.5 ARP-G - Algorithme de placement par grappes à base de règles de priorité 43
2.6 AOS-G - Algorithme de placement par grappes d'ordre strict 44
2.7 Algorithme de placement d'une grappe sous H0, H1, H2, H3 48
3.1 Algorithme génétique «classique» 70
3.2 Processus de sélection d'un opérateur de croisement 83
3.3 Algorithme génétique avec méthode d'autodétermination 84
4.1 Algorithme mémétique «classique» 103
4.2 Algorithme mémétique modifié 103
4.3 Algorithme de plus forte pente 104

Liste des tableaux

1.1 Temps opératoires problème de jobshop 3x3 21

2.1 Temps opératoires et affectations 41
2.2 Ressources du processus de lyophilisation 60
2.3 Résultats numériques PharmaCorp 60

3.1 Temps opératoires 66

4.1 Temps opératoires 98
4.2 Classes d'instances Fondrevelle 2003 105
4.3 Résultats détaillés pour les 10 instances de la famille Fondrevelle 2003 N°1 106
4.4 Résultats sur jeux d'essai Fondrevelle 2003 107
4.5 Résultats sur jeux d'essai Taillard 5x20 108
4.6 Résultats sur jeux d'essai Taillard 10x20 109

A.3.1 Jeux d'essai 10x30 sans écarts maximaux 126
A.3.2 Jeux d'essai 10x30 avec écarts maximaux 127
A.3.3 Jeux d'essai 10x30 sans attente 128
A.3.4 Jeux d'essai 5x20 sans écarts maximaux 133
A.3.5 Jeux d'essai 5x20 avec écarts maximaux 134
A.3.6 Jeux d'essai 5x20 sans attente 135
A.4.1 Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°1 . . 153
A.4.2 Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°2 . . 153
A.4.3 Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°3 . . 154
A.4.4 Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°4 . . 154
A.4.5 Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°5 . . 154
A.4.6 Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°6 . . 155
A.4.7 Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°7 . . 155
A.4.8 Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°8 . . 155
A.4.9 Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°9 . . 156
A.4.10 Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°10 . 156

Introduction générale

Il n'est pas une thèse en ordonnancement qui n'évoque un contexte industriel de plus en plus compétitif, une concurrence acharnée, une recherche permanente de rendements pour justifier le développement d'outils de gestion de production de plus en plus performants. Si ces différentes raisons peuvent apparaître comme des clichés, ils sont cependant le reflet d'une réalité. Au cours d'une de mes missions de consultant auprès d'un grand groupe pharmaceutique, le directeur de la production m'a confié «nous ne pouvons plus nous satisfaire d'un plan de production type, nous devons nous adapter rapidement aux demandes fluctuantes de nos clients. Un manque de réactivité se traduit inévitablement pour nous par des pertes de parts de marché». Cette citation résume à elle seule les difficultés auxquelles doivent faire face bon nombre de société qui subissent la politique des « flux tendus » et doivent réagir rapidement à des commandes annulées ou suspendues à la dernière minute, à des livraisons urgentes... Il est donc vital pour elles de se doter d'outils de gestion de production performants répondant à des exigences parfois contradictoires comme le souhait d'avoir un ordonnanceur générique et capable en même temps de gérer efficacement les contraintes particulières du système de production. Ce besoin est perçu depuis longtemps par les éditeurs d'ERP qui se doivent de compléter leur offre par des modules d'ordonnancement capables de fournir un planning opérationnel et non plus uniquement prévisionnel.

Le présent manuscrit est le fruit d'un travail réalisé dans le cadre d'une convention CIFRE entre l'INRIA Lorraine et l'équipe MACSI du LORIA dirigée par Marie-Claude Portmann d'une part, et la société Incotec spécialisée dans l'édition de progiciels pour la gestion de production d'autre part. Il nous a été confié la mission de compléter le logiciel d'ordonnancement de cette société afin d'y intégrer la gestion de contraintes temporelles entre opérations, en particulier des contraintes d'écart maximal qui stipulent que des opérations doivent enchaîner leur exécution dans des délais impartis. Ce type de contrainte est particulièrement fréquent dans les secteurs pharmaco-chimiques et agro-alimentaires avec la présence de produits périssables ou instables dans le temps.

Le premier chapitre constitue une introduction détaillée. Nous exposons des généralités sur les problèmes d'ordonnancement, les notations, les modèles utilisés ainsi que les méthodes de résolution classiques. Nous présentons les problèmes d'ordonnancement avec contraintes temporelles entre opérations. En particulier, nous décrivons dans ce chapitre plusieurs

cas industriels réels comportant de telles contraintes afin de sensibiliser le lecteur à la problématique et démontrer que ces contraintes sont plus répandues qu'on ne le pense. Nous dressons un panorama de l'état de l'art sur le sujet.

Le second chapitre est consacré à la construction d'une solution valide dans le cas général. Nous utilisons à cet effet une approche par construction et présentons plusieurs algorithmes à base de règles de priorités et d'ordre strict que nous avons modifiés pour permettre une gestion efficace des contraintes d'écart maximal. Nous nous intéressons également dans ce chapitre aux conditions suffisantes de convergence de nos algorithmes et essayons d'en évaluer la complexité.

Le troisième chapitre est consacré aux problèmes d'atelier de type flowshop. Nous nous intéressons plus particulièrement à l'optimisation du makespan (durée totale de fabrication) en utilisant un algorithme génétique. Nous développons une méthode de sélection des opérateurs permettant de s'adapter plus facilement à n'importe quel type de problématique.

Dans le quatrième chapitre, nous complétons notre algorithme génétique avec une recherche locale. A cet effet, nous effectuons un récapitulatif des principaux voisinages utilisés dans les procédures d'optimisation par recherche locale. Nous complétons ainsi notre approche génétique par une recherche locale et présentons les résultats numériques obtenus pour plusieurs types de jeux d'essai.

Comme il est d'usage, nous terminons cette thèse par un chapitre de conclusion exposant les principales perspectives de notre étude.

1

Motivations, Définitions et Notations

Dans ce premier chapitre nous présentons les généralités sur les problèmes d'ordonnancement, les notations et les principales méthodes de résolution. Nous exposons les motivations qui nous ont incités à étudier les contraintes temporelles entre opérations et plus particulièrement les contraintes temporelles d'écart maximal. Nous dressons un éventail de différents exemples industriels réels comportant de telles contraintes, ainsi qu'une étude de cas d'un processus de lyophilisation dans une entreprise pharmaceutique soumise à des contraintes très strictes d'écart maximal. Nous faisons un rapide tour d'horizon de l'état de l'art.

Certains travaux sur les études de cas de ce chapitre ont fait l'objet de présentations et de publications antérieures à cette thèse dans Proceedings of the IEEE International Conference on Multimedia Computing and Systems (Hiroshima, juin 1996) [DGH96] et SPIE Proceedings [BDK+98] [KDHJ98].

1.1 Motivations

1.1.1 L'ordonnancement et ses domaines d'application

D'après la définition générale des problèmes d'ordonnancement de Roy [Roy70], il y a problème d'ordonnancement

- quand un ensemble de travaux est à réaliser,
- que cette réalisation est décomposable en tâches,
- que le problème consiste à définir la localisation temporelle des tâches et/ou la manière de leur affecter les moyens nécessaires.

Résoudre un problème d'ordonnancement revient donc à répondre aux questions *«quand»* et *«avec quels moyens»* exécuter chacune des tâches.

De manière synthétique, on peut dire qu'on a un problème d'ordonnancement dès qu'on doit planifier un ensemble de tâches. On se rend compte que les domaines d'application de l'ordonnancement sont immenses. On peut cependant distinguer principalement trois grandes familles :
- l'ordonnancement de projet,
- l'ordonnancement de la fabrication de produits en atelier ou de services,
- l'ordonnancement dans les systèmes informatiques et les systèmes embarqués.

Tous les problèmes d'ordonnancement utilisent les mêmes modèles généraux. Pourquoi dans ces conditions effectuer le distinguo entre ces différentes familles ? Outre le fait que les spécialistes de chaque domaine se rencontrent assez rarement, la principale différence se situe au niveau des contraintes régissant chaque domaine d'application. En ordonnancement de projet, on trouvera rarement des gammes linéaires liées aux produits et les contraintes sur les ressources sont généralement moins fortes qu'en ordonnancement d'atelier où les machines sont en nombre et à capacité limités. La maille de temps n'est pas non plus identique. Les calendriers sont plus finement modélisés en ordonnancement d'atelier où on tient compte des postes 2x8 et 3x8, des pauses... alors qu'en ordonnancement de projet on raisonne en terme de semaines et que dans les systèmes informatiques cette notion est inexistante. Dans les systèmes informatiques, les contraintes de ressource sont également très fortes, les tâches peuvent être interruptibles ou dupliquées alors qu'en ordonnancement d'atelier ceci n'est pas en général le cas. Les objectifs que l'on cherche à optimiser en priorité ne sont pas non plus identiques (délais, somme des retards, coûts, utilisation des ressources...).

Grossièrement, les modèles sont identiques mais les contraintes sont plus ou moins «serrées» selon le domaine d'application considéré.

1.1.2 Les contraintes considérées

Dans cette thèse, nous nous intéressons tout particulièrement aux contraintes temporelles entre les opérations. Nous considérons les écarts minimaux (ou temps d'attente minimaux) entre les couples d'opérations, mais surtout les écarts maximaux (ou temps d'attente maximaux).

Une contrainte d'écart minimal définie entre deux opérations stipule que la seconde opération ne peut débuter son exécution avant qu'un laps de temps minimal se soit écoulé depuis le début ou la fin de la première opération. Ce laps de temps peut correspondre à du chevauchement, un temps de transfert ne mobilisant aucune ressource, un temps de séchage ou de stockage, un temps de communication entre process d'un système informatique...
Une contrainte d'écart maximal définie entre deux opérations stipule au contraire que la seconde opération doit débuter son exécution impérativement dans un laps de temps donné après le début ou la fin d'exécution de la première. En chimie par exemple, la filtration d'un produit doit intervenir rapidement après la formulation pour éviter la précipitation du produit. Pour les systèmes informatiques, un process doit répondre dans un délai imparti...

Même si nos résultats se veulent le plus générique possible, nos algorithmes ont été développés à partir d'exemples et d'observations de problèmes d'ordonnancement d'atelier ou de services. Nous situons donc l'ensemble de nos travaux dans ce domaine.

1.1.3 Contexte de la thèse

Incotec est une SSII d'une soixantaine de personnes réparties pour moitié en France à Strasbourg et pour moitié en Inde à Poune. Spécialisée dans l'édition de progiciels, la société développe et commercialise en particulier Incoplan, logiciel d'ordonnancement. Essentiellement issu d'un savoir faire dans le domaine mécanique, Incotec souhaitait ouvrir Incoplan vers d'autres domaines d'activités et plus particulièrement les industries chimiques, pharmaceutiques et agroalimentaires. Dans ces secteurs, les contraintes temporelles entre opérations sont des contraintes très fortes. Incotec s'est mis en contact avec le LORIA (Laboratoire Lorrain de Recherche en Informatique et ses Applications) et plus particulièrement l'équipe MACSI (Modélisation, Analyse et Conduite de Systèmes Industriels), de l'INRIA Lorraine, dirigée par Marie-Claude Portmann et installée à l'Ecole des Mines de Nancy, pour tenter d'améliorer Incoplan et gérer plus efficacement ce genre de contraintes. Un dossier CIFRE a été monté, un appel à candidature a été lancé et j'ai eu l'honneur d'être retenu pour le poste de Doctorant.

Les exemples exposés dans cette thèse sont des cas concrets que j'ai eu l'opportunité d'étudier au cours de mes travaux de consulting et de maquetage pour le compte d'Incotec, ou au cours de mes expériences professionnelles antérieures. Pour des raisons de

confidentialité, nous avons cependant dû «édulcorer» les différents exemples et sommes restés suffisamment génériques pour ne pas porter préjudice à Incotec et ses clients, tout en conservant les aspects les plus intéressants de chaque cas. Tous les noms d'entreprises, à l'exception d'Incotec, ont été modifiés.

1.2 Exemples industriels

Depuis le début des années 90, un intérêt croissant pour les contraintes d'écart maximal est en train de voir le jour, particulièrement en ordonnancement de projets. Les publications restent cependant encore marginales. La simple construction d'une solution valide peut s'avérer être NP-complet dès une machine pour certains problèmes [BMR88] et peut expliquer ce manque d'attention.

A première vue, cette relative marginalisation pourrait également tenir au fait que ce type de contrainte occupe une place secondaire dans la hiérarchie des contraintes. Ceci est pourtant loin d'être le cas. En y regardant de plus près, on se rend compte que de très nombreux processus industriels sont soumis à ce genre de contraintes. Si on imagine aisément la place prépondérante que peuvent occuper les écarts maximaux au sein de processus chimiques faisant intervenir des produits instables dans le temps, ou dans le secteur agroalimentaire en présence de produits périssables, des exemples plus courants de la simple mécanique y font aussi appel.

Dans cette section, les figures illustrant les différents exemples comportent deux types de liens. Les traits pleins symbolisent les liens de précédence avec contrainte d'écart minimal. Ces derniers sont valués par le laps de temps minimal devant s'écouler entre des événements liés aux opérations (début ou fin selon la position des flèches). En absence de valuation, il s'agit d'une succession simple de deux opérations. Les traits pointillés symbolisent au contraire les contraintes d'écart maximal entre opérations et sont valués par le temps maximal pouvant s'écouler entre des événements liés aux opérations.

1.2.1 Successions sans attente

De nombreuses gammes réelles comportent des opérations successives qui s'exécutent sur des ressources différentes mais doivent se succéder sans temps morts. Ceci est le cas pour des opérations de préparation (figure 1.1) ou encore en l'absence de place de stockage avec impossibilité de conserver la pièce sur la ressource (blocage de ressource interdit).

Il est à noter que nous considérons, dans notre modélisation et nos outils, que la demande en ressources est constante durant toute la durée d'une opération ce qui induit également des opérations successives sans attente lorsqu'il faut modéliser une opération demandant des variations sur sa demande en ressources. Ceci est le cas, par exemple, pour des opérations de réglage avec des niveaux variables des demandes de ressources.

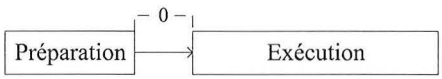


FIG. 1.1 – Succession immédiate entre la préparation et l'exécution

1.2.2 Successions en un laps de temps défini

Certaines opérations doivent se succéder en un laps de temps défini. En 1985, Hodson et al [HMP85] ont présenté le cas d'une société spécialisée dans la fabrication de plats surgelés. La congélation doit intervenir au plus tard 30 minutes après la cuisson sinon le plat est déclaré impropre à la consommation.

Une entreprise de renommée internationale implantée en Alsace dispose d'une unité de fabrication de papier thermique (support d'étiquettes destinées à l'alimentaire, de tickets de jeux en temps réel...). La fabrication de ce papier se décompose en 3 grandes phases :
- Préparation des bains de produits chimiques (mixing)
- Couchage (coating)
- Coupe (cutting)

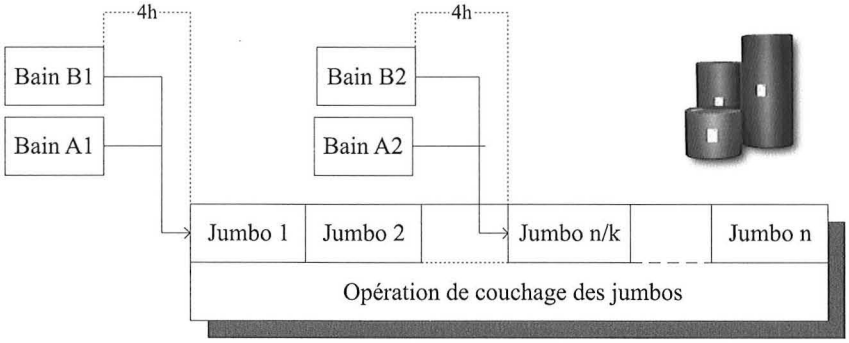


FIG. 1.2 – Couchage du papier thermique

L'unité de base de fabrication est le jumbo, énorme rouleau de papier caractérisé par une laize et une longueur. Pour fabriquer un lot de jumbos, un certain nombre de bains de produits chimiques sont nécessaires. Ces bains sont consommés au fur et à mesure au cours de l'opération de couchage consistant à recouvrir les surfaces recto-verso du papier d'une sauce de couchage (terme officiel) pour lui faire acquérir les propriétés voulues (brillance, fluorescence, pouvoir d'absorption, de dispersion...). Pour un lot de n jumbos, le MRP «Materiel Requirement Planning» calcule à partir de la nomenclature le nombre k de bains nécessaires. Le planning est ensuite établi en sachant qu'un bain est consommé tous les n/k jumbos et que les bains doivent être disponibles au plus tôt 4h avant d'être consommés pour ne pas perdre leurs propriétés chimiques (figure 1.2). Le couchage s'effectuant à un rythme de 300 à 500 mètres/minute via 4 stations d'enduction couplées à 4 sécheurs, tout

retard dans la disponibilité des bains interrompt la chaîne entraînant perte de temps et de papier. La reprise du process nécessite une période plus ou moins longue jusqu'à atteindre un état stable garantissant une uniformité de la qualité du couchage.
Une opération de couchage peut utiliser plusieurs bains de nature différente selon les propriétés que l'on souhaite faire acquérir au papier. A l'issue des opérations de couchage, les jumbos sont envoyés en chambre de vieillissement pour y être stockés de 4 à 7 jours.

Au milieu des années 90, avec l'explosion des réseaux et d'internet, de nouveaux services étaient à l'étude. On prédisait en particulier la fin des magasins de location vidéo et le développement des serveurs «video on demand». Des utilisateurs se connectent à un site pour visionner à la demande des flux vidéos contre paiement. Un travail est dans ce cas de figure constitué d'un ensemble de séquences vidéos à transmettre à un client donné. Entre l'envoi de chaque séquence, un délai minimal doit être respecté pour éviter de surcharger le buffer du client. En même temps, un délai maximal correspondant au temps de visionnage de la séquence doit être respecté pour éviter les coupures. A chaque nouveau client se connectant, un re-schedule des envois est nécessaire pour déterminer le nouveau planning des envois des trames et déterminer si le nouvel arrivant peut être accepté sans dégrader la qualité de service. A l'époque, bon nombre d'études avaient été menées pour dimensionner les buffers, les trames vidéos, planifier les accès disques (voir par exemple Deppner et al [DGH96])...

1.2.3 Contrôle

Des opérations ayant des temps opératoires très longs nécessitent, en cours d'exécution, des opérations de contrôle qui permettent, le cas échéant, de rectifier des réglages. Il ne s'agit pas d'effectuer ces contrôles trop tôt (le système n'étant pas encore dans une phase stable) ni trop tard (production perdue en cas de contrôle négatif).
La figure 1.3 expose un tel cas de figure. Au cours de l'opération d'une durée p, deux contrôles sont nécessaires. Le premier, «Contrôle1», doit intervenir entre le 1/4 et le 1/3 de l'opération. Ceci est modélisé par une contrainte d'écart minimal de $1/4.p$ unités de temps devant s'écouler depuis le début de l'opération, plus une contrainte d'écart maximal de $1/3.p$ unités de temps entre le début de l'opération et le début du contrôle; idem pour «Contrôle2» avec une contrainte d'écart minimal de $2/3.p$ unités de temps et une contrainte d'écart maximal de $3/4.p$ unités de temps.
Ce cas de figure a été rencontré fréquemment dans de nombreux ateliers de mécanique comportant des opérations de tournage. L'opérateur lance la commande et tout s'exécute de manière automatisée mais des contrôles, des modifications d'outils ou de pièces sont planifiés à intervalles réguliers au cours de l'opération (figure 1.4).

Une des sociétés majeures dans le monde pour la conception et la fabrication des appareils à engrenage, principalement à grande vitesse, est située en Alsace. Ses produits de grande précision sont destinés à des centrales électriques hydrauliques, thermiques ou

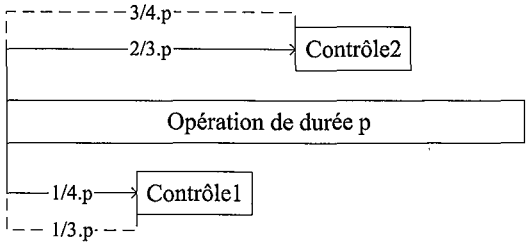


FIG. 1.3 – Opérations de contrôle

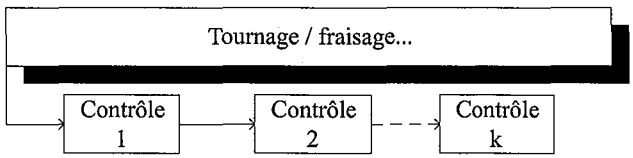


FIG. 1.4 – Contrôles réguliers au cours d'une opération de tournage

nucléaires, voire de certains navires. Ces produits comportent des roues dentées de 2m à 2.50m de diamètre et 50cm d'épaisseur. Les opérations de dentelure nécessitent 12, 24, 48 heures ou plus avec de fréquents contrôles (vérification du niveau de lubrifiant...) voire le remplacement des outils de coupe.

La société WIFAG à Berne (Suisse) commercialise des presses offset pour l'impression journal. Une chaîne d'impression journal est principalement constituée de 4 énormes presses pour imprimer successivement les 4 couleurs noir, jaune, magenta et cyan. D'énormes rouleaux de papier défilent à plusieurs mètres/seconde au travers de ces énormes presses. Ces dernières sont soumises à des contraintes mécaniques fortes et la qualité d'impression n'est pas uniforme au cours du process. Même si actuellement il est possible d'automatiser les mesures colorimétriques via une caméra CCD montée sur la presse [BDK+98] [KDHJ98] et de rectifier en temps réel les paramètrages, la majorité des entreprises ne disposent pas de ces systèmes et il est nécessaire, à intervalle régulier, de procéder à un échantillonnage et de mesurer les différents paramètres (densité colorimétrique, accroissement du point de trame, repérage, doublage et glissement...) pour corriger les réglages.
Le procédé de couchage décrit précédemment est soumis à la même problématique. Cependant, un système de contrôle en ligne a été mis en place pour détecter et localiser tout défaut éventuel.

1.2.4 EducRoute

EducRoute est un centre d'éducation routière et de formation continue. Chaque année, le Conseil Général et l'ANPE lui adressent des demandes de formation pour des chômeurs ou des contrats de reconversion... Outre les contraintes sur les ressources (disponibilité

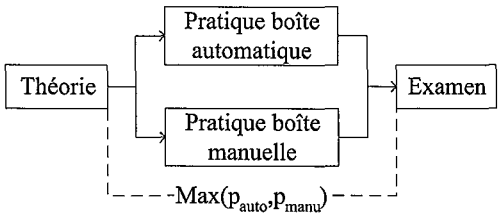


FIG. 1.5 – Educroute - opérations synchrones

des camions et des grues, adéquation entre le formateur et la matière enseignée), l'une des contraintes porte sur l'enchaînement des modules de la formation. La formation doit s'effectuer d'affilée, sans temps morts entre les différentes phases (à l'exception des week-ends et jours fériés) car les participants ne perçoivent qu'une somme forfaitaire proportionnelle à la durée de la formation (ex 13 semaines). Il ne s'agit donc pas de planifier une formation sur 4 mois !

Si les stagiaires chez EducRoute suivent le même cursus, certains modules comportent cependant des sous-modules différenciés. A l'issue de la formation théorique par exemple, les stagiaires se séparent en deux groupes et poursuivent par la conduite de semi-remorque à boîte automatique ou manuelle. Les deux groupes se retrouvent ensuite de nouveau ensemble pour un examen. Tout en ayant un tronc commun et un parcours «synchrone», certains modules sont différenciés. Ceci est modélisé sur la figure 1.5 par deux opérations parallèles devant s'exécuter entre la théorie et l'examen dans un intervalle de temps de longueur égale à la durée maximale de la pratique manuelle ou automatique. Si ce cas de figure peut faire penser à du batch, nous sommes dans une problématique différente dans la mesure où les ressources ne sont pas identiques. De plus, les stagiaires peuvent être libérés dès la fin de leur module contrairement aux problèmes de batch où les produits ont une date de début et de fin d'exécution commune (voir par exemple [DDF02]).

Une formation est découpée en modules, eux-même partagés en sous-modules. Tous les modules et les sous-modules doivent s'enchaîner sans temps mort. Le découpage de certains modules est cependant fait de telle sorte qu'à l'intérieur de ces modules, l'ordre des sous-modules n'a pas d'importance. Ceux-ci peuvent s'enchaîner dans n'importe quel ordre mais dans une durée globale définie (figure 1.6). Ces sous-modules (SM2, SM3, SM4) utilisent cependant la même ressource (salle, formateur ou type de véhicule). Si ce cas de figure est très proche du «sum-batch», il n'est pas équivalent. Il s'agit là de groupe permutable sans temps mort.

Ce cas a également été modélisé dans une aciérie pour le process de coulée continue. Les différents lots caractérisés par la nuance (concentration en fer, nickel...) sont soumis à des contraintes de succession très fortes ; par exemple : après une coulée de telle nuance il faut enchaîner avec des coulées de telle autre nuance ; pas plus de 20 coulées de telle nuance

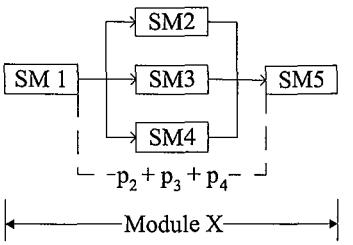


FIG. 1.6 – Educroute - enchaînement des sous-modules

d'affilée; au delà de 40 coulées, appliquer une coulée de nettoyage... Un prétraitement est effectué pour grouper les coulées de sorte à respecter les contraintes de succession des nuances ou tout du moins à minimiser le nombre des enchaînements interdits. Une fois le groupage effectué, les différentes coulées s'effectuent en continu et s'enchaînent sans temps mort. Dans la réalité, on peut se donner un peu de mou en jouant sur les débits pour rattraper légèrement un retard ou essayer de raccrocher à la coulée précédente mais ce degré de liberté est faible. Dans ce cas de figure, les contraintes temporelles ne se situent plus entre opérations d'un même travail mais sont définies au niveau d'une ou plusieurs ressources qui doivent enchaîner sans temps mort les différentes opérations. On pourra par exemple consulter l'article de Saadani et al [SGM03] pour un problème de flowshop à 3 machines.

1.2.5 Hoist Scheduling

Le hoist scheduling (traitement de surface) se situe à la périphérie des problèmes avec contraintes temporelles entre opérations. Pour une présentation de la problématique on pourra se reporter à [BBV01], pour un état de l'art on pourra se référer à [MB03]. Schématiquement, les produits faisant l'objet d'un traitement de surface doivent transiter dans plusieurs cuves de solutions chimiques et/ou électrolytiques. Le temps de séjour dans chaque cuve doit être compris entre une durée minimale et une durée maximale. Le convoyage des produits est effectué par des palans ou des robots en capacité limitée. Ce type de problématique peut très bien être modélisé par des contraintes temporelles entre opérations. Un travail est constitué d'une alternance d'opérations de trempe et de convoyage. Le temps opératoire de chaque opération de trempe correspond au temps de présence minimal du produit dans la cuve. On impose de plus une contrainte d'écart maximal entre le début de l'opération de trempe et l'opération de convoyage, correspondant au temps de trempe maximal autorisé dans la cuve. Le convoyage de la pièce dans la cuve suivante débutera donc impérativement avant que la durée de séjour maximale autorisée dans la cuve ne se soit écoulée depuis le début de l'opération de trempe. La succession entre l'opération de convoyage et l'opération de trempe est immédiate. En général, les cuves sont à capacité limitée et servent d'emplacement de stockage pour les produits. Le modèle doit donc également inclure des blocages de ressources.

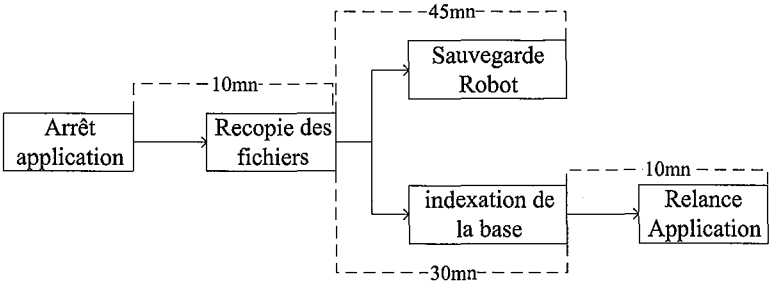


FIG. 1.7 – La Poste - Job de sauvegarde

1.2.6 Ordonnancement de tâches informatiques

Je travaille actuellement en tant que Responsable d'Exploitation Informatique pour le compte du groupe La Poste. Chaque nuit, des campagnes de sauvegarde sont effectuées. Les différents serveurs accèdent aux «têtes d'écriture» d'un robot pour sauvegarder leurs données applicatives sur bandes magnétiques. Un robot est constitué de trois lecteurs DAT et d'un bras articulé qui se charge d'introduire et de retirer les bandes du lecteur puis de les ranger. Chaque «tête d'écriture» correspond à un des lecteurs DAT.

Nous disposons d'un ordonnanceur qui permet d'ordonnancer les opérations en définissant des dépendances entre les différents jobs. Un job est constitué d'une succession d'opérations du type arrêt de l'application, recopie des fichiers, ré-indexation de la base, sauvegarde sur le robot... Cet ordonnanceur ne tient cependant pas compte des capacités des ressources. A l'issue d'une opération, l'ordre est donné à la suivante de démarrer son exécution. Si cette opération ne s'est pas effectuée dans les délais impartis, le job tombe en «abend», bloquant ainsi tous ceux qui ont des liens de précédence avec ce dernier. Le pilote de ressource est contraint de reprendre manuellement l'enchaînement des opérations.

En temps normal, ce type d'incident est rare. Cependant, des campagnes de sauvegarde exceptionnelles peuvent être programmées. Lors de ces campagnes, un nombre plus élevé d'opérations de sauvegarde demandent l'accès simultané aux «têtes d'écriture» du robot. Celles-ci étant en nombre limité, les jobs sont mis en attente et s'ils ne démarrent pas dans les délais impartis, ils tombent en «abend».

La figure 1.7 représente un job modélisé par une succession d'opérations qui possèdent entre elles des contraintes de temps d'attente maximale. Ces contraintes sont fortes. Le non-respect de l'une d'entre elles bloque l'exécution du reste du job ainsi que les jobs ayant une dépendance avec ce dernier. Contrairement aux exemples précédents, ces contraintes sont définies de la fin d'une opération à la fin d'une autre.

1.3 Etude de cas - PharmaCorp

Mon travail au sein de la société Incotec m'a permis d'effectuer bon nombre d'études de cas et de maquettes de problématiques rencontrées auprès de clients potentiels. Nous présentons ci-dessous le cas d'une entreprise pharmaco-chimique. Par mesure de confidentialité, il nous est impossible de citer cette entreprise, nous la désignons sous le pseudonyme de PharmaCorp. Ce cas revêt une importance toute particulière car c'est cette entreprise qui a historiquement poussé Incotec à entamer des opérations de recherche et développement sur les contraintes temporelles. Son étude a constitué pour moi une étape importante et décisive pour ma thèse puisque j'ai développé mes algorithmes sur la base de l'étude du processus de lyophilisation des secteurs «Oncologie» et «Hormones de croissance».

1.3.1 Gamme pharmaceutique et système de production associé

La gamme présentée en annexe sur la figure A.1.1 est une version simplifiée du processus réel qui comporte des opérations supplémentaires. Les flèches pleines représentent les contraintes de précédence entre opérations. Les traits en pointillés représentent les contraintes d'écart maximal entre opérations et sont libellés par le délai maximal à respecter entre la fin d'une opération et le début d'une autre. Le process conduisant à l'élaboration des produits à partir de matières premières comporte grossièrement trois étapes successives :

- Préparation du matériel et des matières premières avec des opérations qui peuvent être effectuées en parallèle sur des ressources pouvant exister en plusieurs exemplaires (gamme de type assemblage)
- Lyophilisation (plusieurs opérations en série qui doivent passer sur l'une des lignes de lyophilisation qui ne sont pas identiques en ce qui concerne leurs durées opératoires et les contraintes de précédence)
- Conditionnement unitaire (plusieurs opérations en série sur des machines en exemplaire unique)

La préparation du matériel et des matières premières consiste essentiellement à doser les matières premières (pesée, formulation) pour ensuite les mélanger et les filtrer. Durant cette phase, l'intervalle de temps maximal toléré entre la pesée des matières premières et la formulation est de 24h. La filtration doit ensuite succéder immédiatement à la formulation pour éviter la précipitation des produits. Parallèlement à la préparation des matières premières, le matériel nécessaire est stérilisé (flacons, bouchons...). Cette stérilisation doit être terminée au plus tôt 48h avant le montage, opération qui consiste à placer les flacons dans une immense cuve (stérilisée elle aussi), sorte de grosse marmite sur roulette qui va transiter avec son chargement le long de la ligne de lyophilisation.

Une fois les matières premières et le matériel préparés, on effectue le remplissage des flacons. Cette opération est la première sur la ligne de lyophilisation. Elle se déroule en

chambre stérile et consiste à remplir chacun des petits flacons avec la dose adéquate de produit. Cette opération doit intervenir 24h au plus tard après le montage.
La phase de lyophilisation s'effectue indifféremment sur l'une des trois lignes de lyophilisation, deux lignes classiques et une ligne «UsiFroid». La particularité de la ligne «UsiFroid» est de permettre le stockage en chambre froide des produits lyophilisés en attendant la libération des ressources de sertissage. Ce stockage ne doit cependant pas excéder les 128h. Entre l'opération de lyophilisation proprement dite et la formulation, un écart maximal de 70h doit être respecté. Cet écart dépend du type de produit et peut varier de 48 à 96h selon les cas.
A noter que les lignes de lyophilisation sont partagées entre plusieurs produits qui n'utilisent pas les mêmes ressources. La gamme présentée est celle valable pour une famille de produits représentant 70% de la charge des lignes. La ligne de lyophilisation est localisée dans une chambre stérile. Cette dernière est utilisée à partir de l'opération de remplissage jusqu'aux opérations de déchargement et nettoyage. Pendant ce temps, elle ne peut être utilisée par aucun autre produit, nous sommes donc également soumis à des contraintes de type blocage de ressources.

Le déchargement clôt la phase de lyophilisation. Entre le déchargement et le sertissage un écart maximal de 40h est imposé. Cette contrainte n'est cependant pas systématique. Il ne s'agit pas d'une contrainte technique mais d'une contrainte réglementaire. Celle-ci dépend des normes en vigueur dans le pays de destination (pays anglo-saxon ou non) du lot fabriqué.

Au niveau du système de production, les principales ressources à disposition sont des ressources humaines (laborantins, «conducteurs» de lyophilisateur) soumis à des calendriers 1x8 ou 2x8, des salles techniques (laboratoires, chambres stériles...), des filtres, des autoclaves au nombre de 2, un stérilisateur, une machine pour visser les bouchons sur les flacons, et bien sûr les lyophilisateurs au nombre de 3 répartis sur les 3 lignes de lyophilisation. Chaque opération utilise plusieurs ressources, en particulier, une salle et une ressource humaine sont nécessaires. L'un des objectifs de la planification est également de fournir un planning précis de l'occupation des salles et des ressources humaines. Une même ressource peut être utilisée par plusieurs opérations. Ceci est par exemple le cas sur la ligne de lyophilisation où les différentes opérations utilisent le lyophilisateur. La figure A.1.2 qui se trouve en annexe, offre une vue synthétique de l'utilisation des ressources par les différentes opérations de la gamme (cadres en pointillés). Sur cette figure ne sont présentées que les deux lignes classiques de «lyophilisation».

1.3.2 Objectifs de PharmaCorp

Ce n'est pas tant l'optimisation d'un critère donné qui intéresse PharmaCorp mais plutôt la recherche d'une solution valide. Jusqu'à peu, PharmaCorp disposait d'un planning type sur deux semaines qui lui permettait de produire sur ses lignes de lyophilisation

suffisamment de lots de chaque produit pour répondre à la demande. Il a fallu deux années à PharmaCorp pour élaborer et optimiser ce planning. A l'heure actuelle, il ne permet plus de répondre efficacement à la demande en forte augmentation. Il devient impératif pour répondre aux commandes de produire par exemple 6 lots du produit A alors que le planning type n'en fournit que 3.
L'exigence de la société s'exprime en terme de réactivité. Elle souhaite pouvoir explorer et mettre au point plusieurs plannings types permettant de répondre plus efficacement aux demandes fluctuantes. La principale exigence est que ces plannings respectent les contraintes réglementaires. Il est indispensable de fournir un outil permettant d'obtenir des solutions respectant les contraintes; un outil permettant de fournir des solutions minimisant le nombre de contraintes insatisfaites n'est pas acceptable. L'optimisation du planning en terme de makespan (durée totale de l'ordonnancement) ou de délai constitue la cerise sur le gâteau mais n'est nullement l'objectif principal.

1.4 Bilan des études de cas

Quels enseignements peut-on tirer des différents exemples industriels et de l'étude de cas de la société PharmaCorp ? Tout d'abord, les études menées prouvent que les exemples mettant en oeuvre des contraintes temporelles entre opérations sont nombreux et variés. Si la plupart du temps ces contraintes sont «masquées» pour ne pas alourdir et complexifier les modèles, dès qu'on désire modéliser finement un processus, les contraintes temporelles entre opérations (et d'autres encore) deviennent incontournables. On se retrouve très rapidement devant des problèmes hautement complexes pour lesquels les modèles classiques d'ordonnancement d'atelier apparaissent trop grossiers, les résultats généraux deviennent caducs. Cette nécessité de modéliser plus finement les process devient pourtant une exigence de moins en moins incontournable dans le cadre des objectifs ISO 9001 et autres plans de qualité.

Excepté pour EducRoute ou pour les serveurs «video on demand», les contraintes temporelles sont des véritables contraintes qui doivent être respectées sous peine de lourdes conséquences (production perdue). Il ne s'agit pas de «souhaits» du client ou d'objectifs avec pénalités. Une solution pour être valide doit satisfaire ces contraintes.

Dans les exemples que nous avons étudiés, les contraintes temporelles sont soit localisées au niveau de quelques groupes d'opérations (préparation, contrôle...), soit généralisées à l'ensemble de la gamme. Dans ce dernier cas de figure, on est en présence la plupart du temps d'une organisation de l'atelier apparentée au flowshop. Les contraintes temporelles sont essentiellement situées au sein d'un même travail. PharmaCorp est l'exemple de gamme le plus complexe que nous ayons eu à traiter. Nous avons été amenés à étudier des cas plus complexes, comme le problème de coulée continue, mais dans ces cas, un prétraitement est effectué en amont par le bureau des méthodes pour en réduire la

complexité. Celui-ci détermine par exemple l'enchaînement des coulées de sorte que la construction d'une solution valide soit aisée.

PharmaCorp fait apparaître que les écarts maximaux entre opérations ne sont pas systématiques pour tous les travaux. Ces écarts ne sont pas uniquement imposés par les ressources mais également par le type de produit. Selon le travail, ces écarts n'ont pas les mêmes valeurs. Enfin, les contraintes temporelles entre opérations ne se situent pas toujours entre opérations successives.

1.5 Etat de l'art

Comme nous l'avons déjà souligné, l'ordonnancement de projet et l'ordonnancement d'atelier utilisent les mêmes modèles et sont soumis à des contraintes identiques. L'importance de ces contraintes n'est cependant pas la même dans les deux cas.

En ordonnancement de projet, les contraintes portent essentiellement sur les successions et les précédences entre les tâches, très peu sur les ressources comme en ordonnancement d'atelier. Ainsi, en ordonnancement de projet, les gammes sont rarement linéaires. Les intervenants d'un projet sont nombreux et variés. Leurs interventions nécessitent d'être synchronisées et de se succéder en des laps de temps définis. La richesse des contraintes temporelles est donc plus importante en ordonnancement de projet. On trouve trace de ces contraintes dans la littérature sous la dénomination «precedence delay», «time lag», «transportation time», «time window» selon qu'il s'agisse d'ordonnancement de projet ou d'atelier, selon que les contraintes sont définies entre la fin d'une opération et le début d'une autre, entre les débuts de chaque opération...

En ordonnancement d'atelier ont surtout été étudiées les contraintes d'écart minimal entre opérations qui correspondent à des temps de séchage, de chevauchement limité ou de transfert d'une position de travail à une autre (présence d'un robot par exemple) sans utilisation de ressource goulet. Dès 1958, Mitten [Mit58] a étudié le problème de flowshop à 2 machines avec chevauchement entre les opérations successives de chaque travail. Il a ainsi proposé une extension de l'algorithme de Johnson [Joh54] pour résoudre polynomialement le problème du C_{max} (minimisation du temps total de réalisation de l'ensemble des travaux) pour le flowshop de permutation à 2 machines. Plus récemment, Szwarc [Szw83], Dell'Amico [Del96], Rebaine et Strusevich [RS99], Strusevich [Str99] ont également étudié les problèmes d'ordonnancement d'atelier avec écarts minimaux. Les différents auteurs se restreignent cependant tous à des cas particuliers comme le flowshop de permutation, 1 ou 2 machines, temps de transfert fixes etc... car, en dehors de ces cas particuliers pouvant se résoudre polynomialement, le problème consistant à trouver une solution minimisant le makespan est NP-complet (voir Brucker et al [BCKS01] pour un panorama de la complexité des problèmes d'atelier avec temps de transfert, ou le

site http ://www.mathematik.uni-osnabrueck.de/research/OR/ pour une vue générale de résultats de complexité). Les problèmes de transport avec utilisation d'une ressource à capacité limitée dépassent le cadre de cette thèse. On pourra consulter les articles de Hurink et Knust [HK01b] ou Lee et Chen [LC01] à ce sujet.

En ce qui concerne les problèmes d'atelier avec contraintes d'écart maximal, très peu de références ont été trouvées excepté pour le cas extrême du no-wait (sans attente) [GG64] [Wie72] [SC79] [Röc84] [Kra98] [Svi03]. La présence d'écarts maximaux induit en effet un problème supplémentaire : décider si oui ou non il existe une solution valide est en soi un problème NP-difficile [BMR88]. Hurink et Keuchel [HK01a], Brucker et al [BHH99] ont étudié les problèmes à 1 machine. Leurs méthodes de résolution combinent recherche de l'optimum et recherche d'une solution valide. Plus récemment, Fondrevelle [Fon03] a développé une méthode de résolution exacte pour les problèmes de flowshop de permutation en présence de contraintes d'écart maximal entre les opérations successives d'un même job. La figure 1.13 à la fin de ce chapitre présente une vue synthétique de l'état de l'art en ordonnancement d'atelier.

La littérature en ordonnancement de projet comporte un nombre bien plus important de références portant sur des contraintes de précédence généralisées avec notamment des temps d'attente maximaux entre tâches. Pour un panorama de l'état de l'art, on pourra se référer à l'article de Neumann et al [NSZ02]. Les nombreuses méthodes développées, essentiellement à base de Procédures par Séparation et Evaluation (PSE), combinent également construction d'une solution valide et recherche de l'optimum [Sch98] [RH98] [Hei03]. Une méthode à base de propagation de contraintes pour la recherche d'une solution valide, le tout encapsulé dans un algorithme d'optimisation, a été proposée par Cesta et al [COS02]. Des méthodes à base d'heuristiques et de règles de priorité ont été développées par Brinkmann et Neumann [BN96], Franck et al [FNS01].

1.6 Notations

1.6.1 Cas particulier des gammes linéaires

Nous présentons d'abord les problèmes d'atelier de type Jobshop ou Flowshop à m machines et n travaux, déterministes sans préemption. $J = \{J_1, .., J_n\}$ désigne l'ensemble des n travaux, $M = \{M_1, .., M_m\}$ celui des m machines. Nous pouvons travailler avec des calendriers permanents mais aussi en supposant que les machines ne sont pas toujours disponibles (calendriers imposés par ressource). Les algorithmes que nous exposerons par la suite n'excluent pas qu'une ressource puisse traiter plus d'une opération à la fois. Pour exposer nos résultats théoriques, nous supposerons cependant que tel n'est pas le cas et qu'une ressource ne peut traiter qu'une seule opération à la fois. Il ne peut donc pas y avoir de chevauchement entre deux opérations sur la même ressource. Par contre, un chevau-

chement entre deux opérations liées par une précédence et s'exécutant sur des ressources différentes est possible. Les temps opératoires sont supposés constants.

Les m opérations ou tâches du travail j sont notées $O_{1,j}, O_{2,j}, .., O_{m,j}$. Pour toute opération $O_{i,j}$, notons
- $r_{i,j}$ sa date de disponibilité (l'opération ne peut débuter avant cette date)
- $p_{i,j}$ son temps opératoire
- $s_{i,j}$ sa date de début
- $c_{i,j}$ sa date de fin
- $M_{i,j}$ la machine qui exécute l'opération $O_{i,j}$ ($M_{i,j} = j$ dans le cas du flowshop)

Le système doit satisfaire un ensemble d'inégalités de potentiels généralisées entre les opérations de la forme :

$$s_{i,j} + d(O_{i,j}, O_{k,l}) \leq s_{k,l} \qquad d(O_{i,j}, O_{k,l}) \in I\!R$$

- $d(O_{i,j}, O_{k,l}) = p_{i,j}$ traduit une précédence simple entre les deux opérations.
- $0 \leq d(O_{i,j}, O_{k,l}) < p_{i,j}$ traduit une précédence entre $O_{i,j}$ et $O_{k,l}$ avec chevauchement (sous réserve que les deux opérations utilisent des ressources différentes)
- $d(O_{i,j}, O_{k,l}) > p_{i,j}$ traduit une précédence entre $O_{i,j}$ et $O_{k,l}$ avec écart minimal entre la fin de $O_{i,j}$ et le début de $O_{k,l}$ (temps de transfert, de séchage...)
- $d(O_{i,j}, O_{k,l}) < 0$ traduit une contrainte d'écart maximal entre le début de $O_{k,l}$ et le début de $O_{i,j}$ ($O_{i,j}$ doit débuter au plus tard à $s_{k,l} - d(O_{i,j}, O_{k,l})$)

Il est possible de définir certaines contraintes entre la fin d'une opération et le début d'une autre (temps de transfert ou temps d'attente maximal). Le fait d'avoir supposé les temps opératoires constants nous permet cependant de modéliser toutes les contraintes temporelles par des inégalités de potentiels $s_{i,j} + d(O_{i,j}, O_{k,l}) \leq s_{k,l}$ [BMR88].

1.6.2 Gammes quelconques

Nous traitons également des gammes quelconques, non linéaires où des opérations peuvent s'exécuter en parallèle (PharmaCorp, opérations de contrôle...). Dans ce cas, les notations précédentes restent valables mais la numérotation des opérations ne traduit plus systématiquement un ordre de précédence entre les opérations d'un travail.

1.7 Modélisation des ressources et des calendriers

Les ressources renouvelables et les calendriers sont modélisés sous forme d'«un profil de capacité». Ces profils de capacité sont des courbes discrètes qui traduisent les disponibilités des ressources au cours du temps. La figure 1.8 est un exemple de profil de capacité

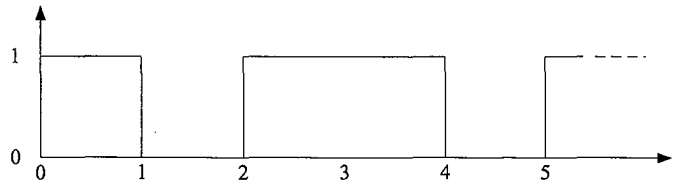


FIG. 1.8 – Profil de capacité d'une ressource

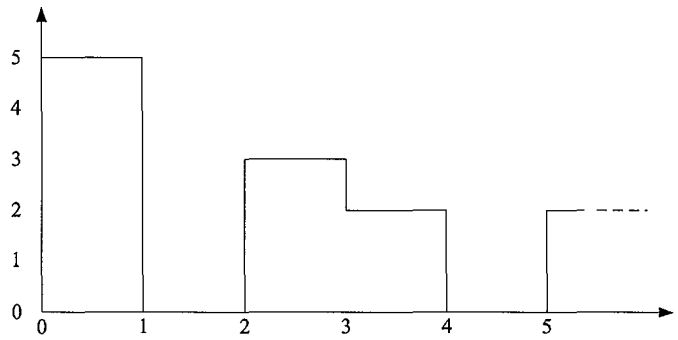


FIG. 1.9 – Profil de capacité d'une ressource multiple

d'une ressource ne pouvant traiter qu'une seule opération à la fois : en abscisse l'axe du temps, en ordonnée la quantité de ressource disponible. Entre $t = 0$ et $t = 1$ la ressource est disponible. Entre $t = 1$ et $t = 2$, la ressource n'est plus disponible. Cette indisponibilité peut être due à un calendrier (période chômée), à une maintenance programmée ou à une opération planifiée traitée par la ressource. La figure 1.9 est un autre exemple de profil de capacité d'une ressource multiple en 5 exemplaires. Le profil de capacité initial d'une ressource tient compte des calendriers. Placer une opération revient simplement à mettre à jour ce profil de capacité.
Le profil de capacité C^j de la ressource M_j est codé sous forme d'une liste de triplets

$$C^j = \{(a_1^j, b_1^j, c_1^j), .., (a_k^j, b_k^j, c_k^j), ..\}$$

Chaque triplet (a_k^j, b_k^j, c_k^j) représente un intervalle de temps $[a_k^j, b_k^j]$ pendant lequel la quantité de ressource c_k^j est disponible. La liste est triée chronologiquement.

Placer une opération de durée p sur la ressource M_j revient donc à :

a «Elaguer» une copie du profil de capacité C^j pour supprimer les triplets (a_k^j, b_k^j, c_k^j) pour lesquels c_k^j est inférieur à la quantité de ressource nécessaire pour traiter l'opération considérée. Après élagage, tous les triplets restants représentent des intervalles de temps où la disponibilité de la ressource est suffisante pour traiter l'opération.
b Parmi les triplets restants, fusionner tous ceux qui sont contigus.
c Parcourir la liste résiduelle pour trouver un intervalle de longueur supérieure à p et postérieur à la date de disponibilité de l'opération.

d Modifier le profil de capacité initial pour tenir compte de l'opération placée.

Le placement d'une opération s'effectue en $O(k)$ où k est le nombre de triplets de C^j. Si une opération doit utiliser plusieurs ressources de natures différentes, une pré-opération d'intersection des profils de capacité sera nécessaire pour déterminer les périodes communes. On appliquera ensuite la même procédure de placement que précédemment.

1.8 Graphe disjonctif

Pour la représentation d'un problème de jobshop ou flowshop, nous utilisons un graphe disjonctif (disjunctive graph) $G = (V, C \cup D)$ dû à Roy et Sussman [RS64]. A chaque opération correspond un sommet de l'ensemble V. Additionnellement, V possède deux sommets fictifs, α et ω représentant respectivement le début et la fin d'un ordonnancement. Ceux-ci sont communément nommés racine et antiracine (ou puits). L'ensemble des arcs C représente les contraintes de précédence entre les opérations, encore appelées contraintes conjonctives. Si les opérations $O_{i,j}$ et $O_{k,l}$ sont soumises à une contrainte de précédence $s_{i,j} + d(O_{i,j}, O_{k,l}) \leq s_{k,l}$, un arc orienté de $O_{i,j}$ vers $O_{k,l}$, valué par $d(O_{i,j}, O_{k,l})$ relie les deux sommets correspondants. D est l'ensemble des arcs disjonctifs entre les sommets. Si $O_{i,j}$ et $O_{k,l}$ sont deux opérations monopolisant la même ressource, ces deux opérations ne peuvent s'exécuter simultanément et l'une doit précéder l'autre. De ce fait, l'une des deux inégalités $s_{i,j} + p_{i,j} \leq s_{k,l}$ ou $s_{k,l} + p_{k,l} \leq s_{i,j}$ devra être satisfaite. D contiendra donc un arc doublement orienté, de $O_{i,j}$ vers $O_{k,l}$, valué par $p_{i,j}$ (ou plus en cas de temps de refroidissement de la machine), et de $O_{k,l}$ vers $O_{i,j}$ valué par $p_{k,l}$ (ou plus).
En résumé, un arc orienté de C entre deux sommets traduit une précédence entre deux opérations alors qu'une paire d'arcs de D, inversement orientés entre deux sommets, est une disjonction traduisant le fait qu'une ressource ne peut traiter qu'une seule opération à la fois.

Ordonnancer un problème de jobshop consiste à trouver un ordre de passage des opérations sur les différentes ressources, autrement dit à arbitrer l'ensemble des arcs doubles de D pour ne conserver pour chaque paire qu'un seul arc. Le graphe résultant est un graphe pôtentiels-tâches. Pour être valide, il ne doit pas comporter de circuit de longueur strictement positive. On parlera alors de sélection complète et on notera pour toute sélection S, $G(S)$ le graphe disjonctif arbitré correspondant. Dans la suite de cette thèse, on réserve le terme de graphe conjonctif au graphe (V, C) .
Pour une sélection S, la date de début au plus tôt d'une opération est obtenue en considérant dans $G(S)$ le plus long chemin valué de la racine à son sommet ; le makespan, noté $C_{max}(S)$, est égal à la longueur du plus long chemin valué de la racine au puits. S'il n'y a pas d'ambiguïté sur la sélection S, dans le contexte particulier des paragraphes de cette thèse, on note le makespan C_{max}. Pour l'ensemble des sélections complètes, on note C^*_{max} la plus petite valeur du makespan.

Le tableau 1.1 contient les 9 temps opératoires d'un problème de jobshop 3x3. Le graphe disjonctif est représenté par la figure 1.10, la figure 1.11 en est une sélection complète.

Opération	$O_{1,1}$	$O_{2,1}$	$O_{3,1}$	$O_{1,2}$	$O_{2,2}$	$O_{3,2}$	$O_{1,3}$	$O_{2,3}$	$O_{3,3}$
Ressource	M_1	M_2	M_3	M_2	M_1	M_3	M_1	M_3	M_2
Durée	4	5	4	7	5	5	4	6	5

TAB. 1.1 – Temps opératoires problème de jobshop 3x3

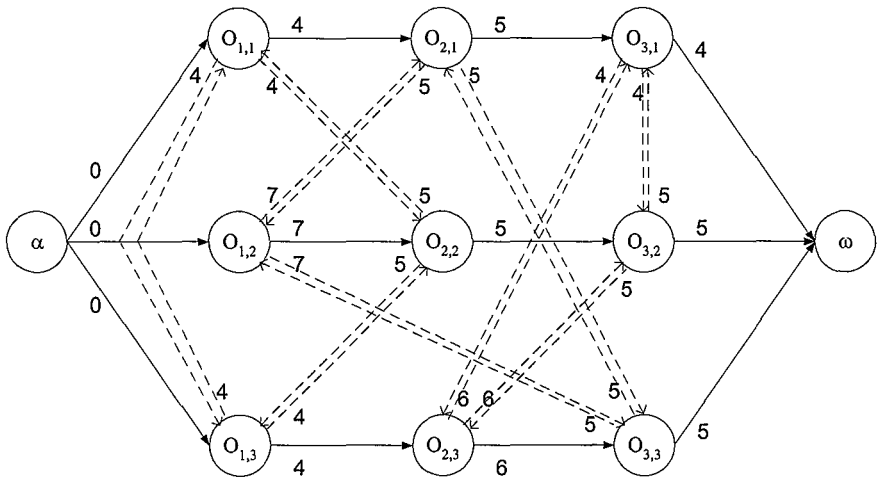


FIG. 1.10 – Graphe disjonctif

Construire une solution valide est NP-complet dans le cas général [BMR88]. Pour s'en convaincre, il suffit de considérer un problème de jobshop sans écarts maximaux et de rajouter une contrainte d'écart maximal entre la racine et le puits égale au C^*_{max}. La recherche d'une solution valide est donc équivalente à la recherche d'une solution optimale.

1.9 Les différents types d'ordonnancement

Cette partie présente les définitions des différents types d'ordonnancement, d'après le livre de Baker [Bak74] et l'article de Sprecher et al [SKD95].

Valide : Un ordonnancement est dit valide si toutes les contraintes de précédence entre les opérations et toutes les contraintes de ressources sont satisfaites.

Semi-actif : Un ordonnancement est dit semi-actif s'il est valide et s'il est impossible de translater une opération vers la gauche pour obtenir un nouvel ordonnancement valide tout en conservant le même ordre de succession des opérations sur les différentes ressources.

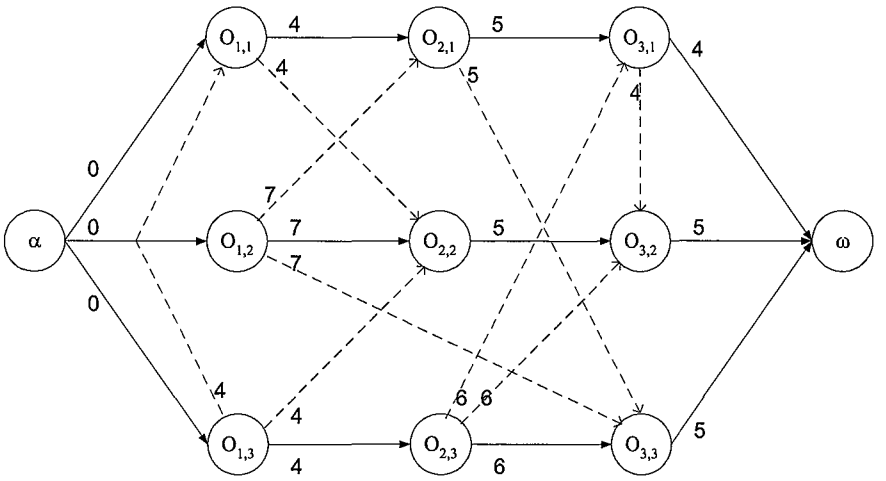


FIG. 1.11 – Graphe disjonctif arbitré ; graphe potentiels-tâches

En présence d'écarts maximaux, cette définition n'est plus tout à fait exacte. Toute l'ambiguïté repose sur la phrase «s'il est impossible de translater une opération vers la gauche».

Considérons l'exemple suivant :
- deux ressources (M_1,M_2) et un travail comportant deux opérations ($O_{1,1}$, $O_{2,1}$) ;
- $O_{1,1}$ s'exécute sur M_1, $O_{2,1}$ sur M_2 ;
- $p_{1,1} = p_{2,1} = 1$;
- $d(O_{2,1}, O_{1,1}) = -1$ (pas de temps mort autorisé entre la fin de $O_{1,1}$ et le début de $O_{2,1}$, symbolisé par le lien vertical gras sur la figure 1.12) ;
- $r_{1,1} = 0$.

La solution valide de la figure 1.12 n'est cependant pas optimale, il est possible de translater les deux opérations vers la gauche pour caler $O_{1,1}$ à l'origine. Pourtant, individuellement, il est impossible de translater les opérations, $O_{1,1}$ étant liée à $O_{2,1}$ par une contrainte d'écart maximal. La solution proposée est semi-active au sens de la définition précédente. Il convient donc de modifier cette définition et de la reformuler de la manière suivante :

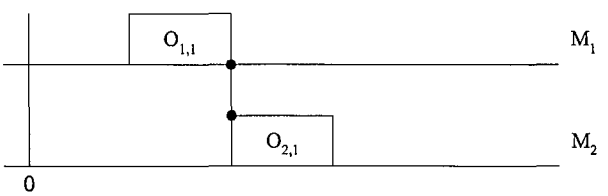


FIG. 1.12 – Solution valide sans attente

Semi-actif (généralisé) : Un ordonnancement est dit semi-actif s'il est valide et s'il est impossible de translater un sous-ensemble d'opérations vers la gauche pour obtenir

un nouvel ordonnancement valide tout en conservant le même ordre de succession des opérations sur les différentes ressources.

De la même manière, la définition d'un ordonnancement actif :

Actif : Un ordonnancement valide est dit actif lorsqu'il est impossible de translater une opération vers la gauche, pour obtenir un nouvel ordonnancement valide, sans en retarder une autre.

est généralisée :

Actif (généralisé) : Un ordonnancement valide est dit actif lorsqu'il est impossible de translater un sous-ensemble d'opérations vers la gauche, pour obtenir un nouvel ordonnancement valide, sans en retarder un autre.

Sans délai (non-delay) : Un ordonnancement valide est dit sans délai si aucune ressource ne reste inoccupée alors qu'elle pourrait débuter une opération.

Nous avons les inclusions suivantes : Sans délai $\subseteq$ Actif $\subseteq$ Semi-actif $\subseteq$ Valide.

Critère régulier : Un critère $f(C_1, C_2, ..., C_n)$ est régulier si

$$C_1' \leq C_1, C_2' \leq C_2, ..., C_n' \leq C_n \Longrightarrow f(C_1', C_2', ..., C_n') \leq f(C_1, C_2, ..., C_n)$$

Un ensemble est dominant s'il contient au moins une solution optimale pour le critère considéré. Pour les versions non-généralisées des définitions, l'ensemble des ordonnancements actifs est dominant pour les critères réguliers dépendant des dates de fin d'exécution des tâches, alors que l'ensemble des ordonnancements sans délai ne l'est pas [Bak74]. Ceci reste donc également vrai pour les versions généralisées utilisées dans cette thèse. Les ordonnancements actifs revêtent donc une importance toute particulière.

1.10 Méthodes de résolution

Un problème de décision est un problème pour lequel on cherche, à partir des données, à répondre à une question par oui ou non. Un problème d'optimisation consiste à trouver parmi un ensemble de solutions celle qui optimise, c'est-à-dire minimise ou maximise, un critère donné. Sauf mention contraire, on traite, dans cette thèse, de problèmes de minimisation et plus particulièrement du makespan.
Les méthodes de résolution sont nombreuses et variées. La recherche très active ces dernières décennies a permis de mettre au point de nouvelles méthodes toujours plus performantes et souvent issues d'hybridations de plusieurs approches, au point qu'il devient difficile de les classifier. Notre ambition n'est pas ici de classifier de manière exhaustive toutes les méthodes mais d'en citer les grandes familles.

1.10.1 Mesure de la qualité des méthodes de résolution

Comme pour beaucoup de concepts, la qualité d'une méthode est un critère subjectif. Une méthode sera évidemment appréciée si elle donne la solution optimale dans le cas d'un problème d'optimisation, ou une solution réalisable dans le cas d'un problème de décision. Ceci est le cas des Procédures par Séparation et Evaluation (PSE) ou de la programmation dynamique pour peu qu'on leur en laisse le temps. Cependant, ces méthodes sont pour la plupart exponentielles et les temps d'exécution sont excessivement longs, ce qui les rend inadaptées pour des problèmes de taille industrielle.
On apprécie dans ce cas des méthodes approchées pour leur rapidité à trouver une solution considérée comme «bonne». Cependant, si ces méthodes peuvent donner «accidentellement» une solution optimale, la plupart du temps elles ne permettent pas d'obtenir l'optimum. De même, si dans certains cas, on peut déterminer grâce à des propriétés ou à des bornes calculées si la solution obtenue est optimale ou non, dans le cas général ceci est impossible. Dans ce cas, les méthodes les plus intéressantes sont celles qui offrent une garantie de performance, on sait que l'on est à moins de ϵ de la solution optimale.
Enfin, une méthode de résolution exacte ou approchée est également appréciée à sa complexité algorithmique, capacité à trouver rapidement une bonne solution. On distingue les méthodes polynomiales (la durée est polynomiale par rapport au nombre de données d'entrée), les pseudo polynomiales (la durée est polynomiale par rapport au nombre de données d'entrée et à la taille maximale des données d'entrée) et le cas quelconque où la durée n'est pas bornée par un polynôme mais peut éventuellement être contrôlée par l'utilisateur.

1.10.2 Processus de conception de méthodes de résolution

Cette partie a été fortement inspirée par le chapitre I de la thèse d'état de ma directrice de thèse [Por87].

1.10.2.1 Méthodes par construction

Les méthodes par construction sont des méthodes itératives consistant en partant d'une solution vide à la compléter à chaque étape en prenant une ou plusieurs décisions jusqu'à aboutir à une solution complète. Parmi les méthodes de construction, on distingue les méthodes gloutonnes, polynomiales en nombre d'étapes, pour lesquelles on ne revient jamais sur une décision prise (NEH [NEH83] , méthodes sérielles [CC88]...), et les méthodes non gloutonnes avec retour arrière possible lorsqu'on ne peut plus compléter la solution par des choix valides. Les méthodes non gloutonnes se décomposent elles-mêmes en méthode d'exploration totale et en méthode d'exploration échantillonnaire (on les répète avec des choix aléatoires) ou partielle (on limite volontairement les choix possibles). Les PSE, où la technique de séparation consiste, par exemple, à construire progressivement une solution

en suivant l'axe du temps ou en choisissant des arcs à arbitrer, font partie des méthodes de construction non gloutonnes.

1.10.2.2 Méthodes par décomposition

Les méthodes par décomposition consistent, comme leur nom l'indique, à décomposer le problème initial en plusieurs sous-problèmes diminuant ainsi d'autant la complexité. Si le principe s'énonce facilement, la mise en oeuvre n'est pas toujours évidente. La décomposition ne peut pas toujours s'effectuer en sous-problèmes indépendants ce qui entraîne des problèmes de fusion des solutions partielles pour obtenir une solution globale.

Parmi les techniques de décomposition on note :

La décomposition hiérarchique
L'exemple le plus courant d'un tel découpage est celui d'ordonnancement à moyen terme, court terme et très court terme. Pour ce type de découpage, les décisions prises à un niveau deviennent des contraintes pour le niveau inférieur et la «qualité» de la solution trouvée à un niveau peut remettre en cause les décisions prises au niveau supérieur. Les modèles des niveaux supérieurs sont agrégés et simplifiés (macro-gamme, sous-système de production remplaçant les machines...). Des techniques de plans glissants sont associées à ces approches. Les aller-retours entre les niveaux sont absolument nécessaires. Le chapitre 2, «Structures hiérarchisées des systèmes de production» de l'ouvrage de Pujo et Kieffer [PK02] permet d'avoir une description détaillée des principes de la décomposition hiérarchique appliquée au pilotage des systèmes de production.

La décomposition structurelle
Pour résoudre, par exemple, un problème d'ordonnancement où les inconnues sont la localisation temporelle et les moyens affectés aux tâches, on suppose dans un premier temps que les moyens sont illimités et on résoud le problème de localisation temporelle. Une fois le problème de localisation temporelle résolu, on s'attache à résoudre celui des affectations. S'il n'y a pas de solution réalisable, on résout à nouveau le problème de localisation temporelle mais avec des contraintes supplémentaires...

La décomposition spatiale
Un exemple de décomposition spatiale consiste à regrouper les travaux utilisant les mêmes ressources et à ordonnancer séparément chaque groupe. Ce découpage sera d'autant plus performant que les groupes sont distincts et utilisent des îlots de production bien séparés. Dans le cas contraire, il faudra rajouter des contraintes pour rendre ces groupes indépendants (par exemple, une gamme est artificiellement découpée en sous-gammes utilisant chacune les ressources d'un îlot de production), ordonnancer chaque groupe et modifier les contraintes si la solution obtenue n'est pas réalisable.

La méthode du «shifting bottleneck» d'Adams et al [ABZ88] peut également être considérée comme une méthode de décomposition spatiale pour le problème de jobshop à m machines.

La décomposition temporelle
C'est sans doute la méthode de décomposition la plus intuitive. On découpe le problème en créant des groupes de travaux que l'on ordonnance successivement en progressant le long de l'axe du temps. Ce découpage se fait par exemple selon les dates d'échéance de chaque travail. Cette méthode peut encore être améliorée par l'utilisation de plans glissants.

La décomposition de l'ensemble des solutions
C'est la méthode de décomposition la plus générale. L'ensemble des solutions est décomposé en sous-ensembles plus faciles à explorer. Ce type de décomposition est notamment à la base des PSE et des méthodes arborescentes à base de propagation de contraintes utilisées dans les solvers commercialisés.

Les différentes techniques de décomposition peuvent être croisées. Par exemple, on peut utiliser des propagations de contraintes pour améliorer une PSE ou utiliser des évaluations obtenues par relaxation pour améliorer les méthodes arborescentes à base de propagation de contraintes ou encore un petit nombre de travaux placés localement de manière optimale par une PSE dans le cas d'une décomposition temporelle.

1.10.2.3 Méthodes par amélioration et processus évolutifs

Recherche locale ou recherche par voisinage
Pour les méthodes d'amélioration par recherche locale, on part d'une solution courante initiale puis on définit un ensemble de solutions voisines de la solution courante qui définissent le voisinage. Une solution de ce voisinage est retenue pour devenir la nouvelle solution courante et servir de point de départ à une nouvelle recherche jusqu'à ce qu'une condition d'arrêt soit rencontrée. Les différentes méthodes se distinguent essentiellement par le voisinage utilisé et la stratégie de parcours de ce voisinage. La méthode de la plus forte pente consiste à explorer toutes les solutions du voisinage et retenir celle qui optimise la valeur de la fonction objectif. La méthode de descente explore le voisinage jusqu'à rencontrer une solution strictement meilleure que la solution courante, et qui va servir de point de départ à une nouvelle recherche. Dans les deux cas, la méthode s'arrête lorsqu'il n'y a plus de solution voisine strictement meilleure que la solution courante. Le fait de retenir une solution strictement meilleure que celle de départ assure une terminaison de l'algorithme, évite de boucler et permet de toujours retourner un optimum local. Ces méthodes sont simples et rapides mais le fait de toujours descendre nous emprisonne systématiquement dans un optimum local souvent de piètre qualité par rapport à l'optimum global, surtout lorsque la topographie de l'espace des solutions est constituée d'une alternance de pics et de vallées. Pour tenter de remédier à ce problème, des variantes ont été

développées consistant à lancer en un autre point de l'espace des solutions une nouvelle recherche une fois atteint un optimum local, à retenir des solutions de qualité égale que la solution courante pour autoriser les déplacements sur un plateau, voire à accepter comme nouvelle solution courante des voisins de qualité inférieure pour «sauter» par dessus les pics et atteindre les vallées voisines. A la base de cette dernière amélioration on trouve essentiellement la méthode tabou de Glover [Glo90] et le recuit simulé de Kirkpatrick et al [KGV83]. Comme nous n'utilisons pas ces méthodes dans cette thèse nous ne les détaillons pas.

Processus évolutifs
Les processus évolutifs sont également des méthodes par amélioration basées sur le principe d'évolution naturelle. Ils sont constitués de trois grandes étapes :
- génération d'une population initiale ;
- évaluation de l'adaptation des individus au milieu ;
- création de nouveaux individus et élimination des individus les moins adaptés.

Les deux dernières étapes sont répétées jusqu'à ce qu'une condition d'arrêt soit rencontrée. Parmi ces méthodes figurent notamment les algorithmes génétiques. Une population d'individus représentant des solutions au problème évolue au gré des croisements, des mutations et des sélections naturelles. Au fil des générations, les meilleurs gènes, ou combinaisons de gènes, vont se propager en s'échangeant si possible les meilleures caractéristiques et en espérant que la concaténation de groupes de bons gènes donne de bons individus. La population va évoluer vers une population de «bons» individus qui constituent autant de solutions intéressantes du problème à soumettre au décideur qui pourra tenir compte éventuellement de contraintes non modélisées.

1.10.2.4 Hybridations et approches diverses

Comme nous l'avons déjà signalé, la recherche très active ces dernières décennies a permis de mettre au point de nouvelles méthodes toujours plus performantes et issues d'hybridation des approches présentées plus haut. On trouve de plus en plus des hybridations entre algorithmes génétiques et méthodes d'amélioration par voisinage sous la dénomination d'algorithmes mémétiques.

1.10.3 Littérature sur les méthodes de résolution

Il existe de nombreux ouvrages traitant des différentes méthodes de résolution. Outre les grands classiques comme le livre de Carlier et Chrétienne [CC88] ou celui de Blazewicz et al [BEP+96] qui permettent d'avoir un panorama des problèmes d'ordonnancement, on peut citer des ouvrages plus récents comme celui de Lopez et Roubellat [LR01], «Ordonnancement de la Production». Le chapitre 3, «les métaheuristiques» de Widmer, Hertz et Costa, présente les approches par construction, par recherche locale (recuit simulé, tabou) et évolutives, ainsi que des hybridations entre algorithme génétique et recherche

locale. On y trouve également quelques applications en ordonnancement. Le chapitre 4, «algorithmes génétiques et ordonnancement» de Portmann et Vignier, est une présentation intéressante des algorithmes génétiques et de leurs applications aux problèmes d'ordonnancement. Dans ce chapitre sont traités les problèmes à une machine, le flowshop et le jobshop. On y trouve également esquissée une hybridation entre algorithme génétique et PSE. Dans le chapitre 5, «propagation de contraintes en ordonnancement» d'Esquirol, Lopez et Huguet, sont abordées les méthodes de propagation de contraintes en ordonnancement et plus particulièrement la propagation de contraintes temporelles et de contraintes de ressource. Il y est notamment mis en lumière les difficultés liées aux traitements de contraintes temporelles complexes comme la présence de calendrier sur les ressources induisant une disjonction des intervalles constituant les domaines de définition des variables.

Dans le livre «Métaheuristiques pour l'optimisation difficile» de Dréo, Petrowski, Siarry et Taillard [DPST03] sont présentées les principales métaheuristiques : recuit simulé, recherche avec tabous, algorithmes évolutionnaires, colonies de fourmis... Le chapitre 8 propose une application des algorithmes génétiques à l'optimisation des réseaux mobiles UMTS. Le chapitre 9 décrit une autre application des algorithmes génétiques à la gestion du trafic aérien. Y est également abordée une hybridation entre algorithme génétique et PSE. Le chapitre 10 est l'occasion d'introduire la programmation par contraintes ainsi qu'une intégration des algorithmes à base de colonies de fourmis dans un environnement de programmation par contraintes.

L'ouvrage de Pirlot et Teghem, «Résolution de problèmes de RO par les métaheuristiques» [PT03], constitue une étude comparative des différentes métaheuristiques et diverses hybridations possibles sur des problèmes d'optimisation comme la coloration des sommets d'un graphe (chapitre 1, Hertz), les tournées de véhicules (chapitre 2, Gendreau, Laporte, Potvin, Semet), l'affectation quadratique (chapitre 3, Mautor)... Le chapitre 4 de Tuyttens, Pirlot et Teghem, présente également un problème d'ordonnancement d'atelier avec la présence de contraintes sans attentes entre diverses opérations successives. Le chapitre 6 de Bloch, Portmann et Vignier, décrit une application des algorithmes génétiques à l'ordonnancement de la production, en particulier le jobshop, le flowshop hybride et le hoist scheduling.

Citons encore le livre de Charon, Germa, Hudry [CGH96] dont le chapitre 11 traite des méthodes PSE, le chapitre 14 des méthodes par voisinage, recuit simulé, méthode tabou, et le chapitre 15 des algorithmes génétiques.

1.11 Contexte des expérimentations

Nous avons effectué plusieurs séries d'expérimentations pour comparer différentes méthodes de résolution qui seront présentées dans les chapitres suivants. Tous les essais

ont été effectués sur un PC Pentium III 500Mhz équipé de Windows 98. Nos différents algorithmes ont été développés sous C++.
Pour obtenir une base de comparaison, nous n'avons pas retenu comme critère le temps CPU. Nous considérons celui-ci comme trop subjectif et soumis à des paramètres non contrôlés, surtout si on compare des travaux programmés par des personnes différentes et dans des langages différents. La qualité du compilateur peut rendre un programme plus performant. Certains des algorithmes que nous utilisons pour les comparaisons, en particulier la PSE développée dans notre équipe par Julien Fondrevelle [Fon03], ne sont pas écrits en C++. Par ailleurs, nous faisons référence à des résultats obtenus par des tiers qui ont utilisé des plate-formes machines différentes de la nôtre et cela aurait pris trop de temps de reprogrammer à nouveau les méthodes des autres sous les mêmes conditions que nos propres travaux avec le risque de ne pas programmer de manière suffisamment efficace les approches des autres. Dans nos propres travaux, les algorithmes utilisent sporadiquement des tris et selon la méthode de tri employée, les performances peuvent varier. Nous n'avons pas souhaité ni eu le temps d'optimiser la durée de nos algorithmes en jouant également sur le choix des procédures de tri ou autres procédures internes.
Toutes ces considérations nous ont incités, dans la mesure du possible, à adopter d'autres critères de comparaison comme le nombre de calculs de dates d'exécution, le nombre de placements d'opérations ou le nombre de croisements et d'évaluations dans le cas des algorithmes génétiques...

1.12 Contenu et structure de la thèse

Dans la suite de cette thèse, nous imposons les deux hypothèses suivantes :

Hypothèse 1 : (notée H0 dans le chapitre 2) si dans le graphe conjonctif (V, C) il existe un arc négatif orienté de $O_{i,j}$ vers $O_{k,l}$, alors il existe également un chemin de $O_{k,l}$ à $O_{i,j}$ constitué exclusivement d'arcs positifs.

Hypothèse 2 : les opérations sont insécables.

L'hypothèse 1 nous semble raisonnable et a été vérifiée dans les cas industriels étudiés. Elle stipule en effet que les contraintes d'écart maximal existent entre couples d'opérations ayant déjà des liens de précédence directs ou indirects au sein de la gamme. L'hypothèse 2 n'est généralement pas restrictive dans le cas des problèmes industriels car on peut rarement couper les opérations n'importe où et en cas de découpe possible, ce sont les bureaux des méthodes qui séparent éventuellement l'opération en plusieurs opérations individuelles.

Dans cette thèse nous nous intéressons aux contraintes d'écart minimal entre opérations correspondants aux temps de transfert, de séchage ou de stockage. Même si les algorithmes et les modélisations que nous présentons permettent de gérer les chevauchements,

ceux-ci ne bénéficient pas d'une étude particulière. Les contraintes d'écart maximal sont considérés avec attention, notre souci principal étant de construire une solution valide. A cet effet, nous développons dans le second chapitre plusieurs algorithmes de liste et de règle de priorité modifiés pour tenir compte des écarts maximaux. L'étude de cas industriels ayant fait ressortir que les contraintes temporelles affectent le plus souvent un sous-groupe d'opérations au sein d'un même travail, nous exposons un algorithme par construction basé sur la décomposition en composantes fortement connexes du graphe conjonctif (V, C). Cette décomposition naturelle avait déjà été suggérée en ordonnancement de projet par Brinkmann et Neumann [BN95]. Nous nous intéressons également dans ce chapitre aux conditions suffisantes de convergence de nos algorithmes et essayons d'en évaluer la complexité.

Les exemples industriels ayant également mis en évidence des organisations d'atelier principalement de type flowshop, ceux-ci bénéficient d'une attention particulière dans le troisième chapitre. Nous nous intéressons à l'optimisation du C_{max} en utilisant un algorithme génétique. Nous étudions pour l'occasion le comportement de plusieurs opérateurs génétiques et développons une méthode de sélection des opérateurs permettant de s'adapter à n'importe quel type de problématique.

Dans le quatrième chapitre, nous effectuons un récapitulatif des principaux voisinages utilisés dans les procédures d'optimisation par recherche locale. Nous complétons ainsi notre approche génétique par une recherche locale et présentons les résultats numériques obtenus pour plusieurs types de jeux d'essai. Nous terminons cette thèse par un chapitre de conclusion exposant les principales perspectives de notre étude.

	JobShop	FlowShop		machines parallèles	1 machine
		m=2	m>2		
Min	Strusevich [Str99] Rebaine et Strusevich [RS99]	Mitten [Mit58] Dell'Amico [Del96] Finke et al [FEJ02]	Szwarc [Szw83]	Munier et al [MQS98]	Balas et al [BLV95] Gupta [Gup96]
Max	Cheng [CS97]	Yang et Chern [YC95]	Fondrevelle [Fon03]		
Min et Max	Deppner [Dep03]				Brucker et al [BHH99] Hurink et Keuchel [HK01]
No Wait	Kravchenko[Kra98]	Gilmore et Gomory [GG64]	Wiesmer[Wie72] Röck[Röc84] Sviridenko[Svi03]		

FIG. 1.13 – Etat de l'art - vue synthétique

2

Construction d'une solution faisable

Pour obtenir une solution faisable, nous avons choisi d'axer nos efforts sur les algorithmes de construction. Dans ce second chapitre, nous expliquons les raisons d'un tel choix. Nous présentons une première adaptation d'un algorithme à base de règles de priorité et d'un algorithme d'ordre strict afin d'y intégrer la gestion des contraintes temporelles d'écart maximal. Nous mettons en évidence le fait que ces approches engendrent de nombreux conflits de priorités entre les opérations et génèrent des solutions fortement dégradées en terme de makespan. Nous améliorons notre première approche en utilisant une décomposition de l'ensemble des opérations en grappes de sorte à minimiser au maximum les conflits. Nous terminons ce chapitre par des résultats numériques.

Les résultats développés dans ce chapitre ont été présentés au 4^e congrès de la société française de recherche opérationnelle et d'aide à la décision (ROADEF 2002) [Dep02] et à la 4^e conférence francophone de modélisation et simulation (MOSIM 2003) [Dep03].

2.1 Introduction

Dans ce chapitre, nous cherchons à développer un algorithme permettant d'obtenir une solution faisable. Notre souci n'est pas l'optimisation d'un critère tel que le makespan mais bel et bien de pouvoir déjà construire une solution valide.

Comme nous l'avons déjà souligné, la construction d'une solution valide dans le cas général est un problème NP-complet [BMR88]. Hurink et Keuchel [HK01a] ont étudié le cas d'une ressource. Les auteurs ont développé une méthode d'amélioration par voisinage basée sur la notion de blocs critiques [BJS94]. Ce voisinage leur permet, partant d'une solution non valide, d'aboutir à une solution valide tout en cherchant à optimiser le C_{max}. En 1994, Brucker et al [BJS94] ont développé une PSE pour résoudre le problème de job shop sans écarts maximaux. En 1999, Brucker et al [BHH99] ont étendu leur approche de 1994 et ont proposé une PSE pour construire une solution valide avec des contraintes d'écart maximal mais pour une machine. Si leurs résultats peuvent s'étendre à m machines, les deux approches butent sur les temps de calcul ainsi que sur la modélisation du problème à l'aide du graphe potentiels-tâches et de la permutation d'arcs disjonctifs. Ce type d'approche est difficilement généralisable à des problèmes avec contraintes industrielles de type calendaire. De plus, même si les performances des ordinateurs ont progressé, les utilisateurs exigent également des temps de réponse quasi immédiats. Toujours pour le C_{max}, De Reyck et Herroelen [RH98], Schwindt [Sch98], Heilmann [Hei03] ont proposé des PSE pour résoudre optimalement des problèmes d'ordonnancement de projets soumis à des contraintes d'écart minimal et maximal. Si leurs méthodes semblent se montrer efficaces dans la majorité des cas, ils combinent dans leur PSE recherche de l'optimum et recherche d'une solution valide au risque de ne pas trouver de solution valide, même non optimale. Citons encore la méthode de Despontin et Briand [DB03] à base d'une heuristique de construction complétée par une méthode d'exploration de voisinage. Dans leur méthode, les écarts maximaux ne sont pas des contraintes mais des objectifs affectés de «poids». La méthode d'exploration par voisinage permet d'offrir à l'utilisateur une solution minimisant le nombre d'écarts maximaux non satisfaits. Comme nous l'avons souligné lors de l'étude de cas PharmaCorp, ce type d'approche n'est pas envisageable.

Beaucoup de méthodes combinent à la fois recherche d'une solution valide et optimisation d'un critère au risque de ne pouvoir fournir au final de solution faisable. Il nous semble judicieux de séparer construction et recherche de l'optimum car, sous certaines hypothèses (cas du flowshop sans attente par exemple), la construction d'une solution valide est relativement simple. En outre, cette séparation permet également de s'adapter plus facilement à l'optimisation de critères différents avec pour objectif prioritaire l'obtention d'une solution faisable.

Des méthodes à base de propagation de contraintes ont également été proposées par Cheng et Smith [CS97] et par Cesta et al [COS02]. Si une telle méthode se montre efficace,

une expérience menée par la société Incotec pour intégrer un logiciel de propagation de contraintes du marché a démontré que ce dernier manquait de souplesse et ne permettait pas d'intégrer facilement des contraintes de type calendaire. De plus, traduire toutes les contraintes pour qu'elles soient exploitables par un algorithme de propagation de contraintes n'est pas une mince affaire ; or pour être accepté par un client, l'outil proposé doit être simple et compréhensible. Même si ce type d'argument n'a que peu de valeur pour les académiciens, il est fondamental pour un futur client.

Dans la suite de ce chapitre, nous allons présenter une première adaptation d'un algorithme à base de règles de priorité et d'un algorithme d'ordre strict afin d'intégrer la gestion des contraintes d'écart maximal entre les opérations.

Pourquoi avoir choisi d'adapter des algorithmes de construction ?
Au vu des contraintes complexes considérées dans cette thèse, les méthodes par construction semblent inadaptées et peu efficaces. On serait tenté de privilégier des méthodes de propagation par contraintes. La raison principale qui a motivé ce choix tient dans la structure même du logiciel d'ordonnancement de la société Incotec. Lorsque j'ai débuté ma thèse, le seul outil d'ordonnancement de cette société était un algorithme par construction à base de règles de priorité. L'objectif obligatoire qui m'avait été fixé était d'améliorer l'outil afin de gérer plus efficacement les contraintes d'écart temporel entre les opérations, et non pas de concevoir ex nihilo un nouvel outil. Ceci permettait en particulier d'utiliser toutes les interfaces existantes et de proposer rapidement une solution aux clients de la société Incotec.

Outre ces contraintes «structurelles», les algorithmes par construction présentent également de nombreux avantages :

- ils sont simples à mettre en oeuvre ;
- ils sont intuitifs (un humain ne procèderait pas différemment s'il devait effectuer un ordonnancement manuellement) ;
- ils sont rapides ;
- ils sont souples et permettent d'intégrer facilement de nombreuses contraintes (calendriers, ressources multiples par opérations...) ;
- jusqu'à peu de temps, ces algorithmes étaient très répandus et constituaient le moteur principal de la quasi-totalité des outils sur le marché ;
- les algorithmes à base de règles de priorité ou d'ordre strict peuvent être complétés par des algorithmes de recherche par voisinage (recuit simulé, tabou..) pour optimiser ou corriger une solution afin de tenir compte de contraintes plus complexes.

2.2 Algorithmes de construction

L'importance des ordonnancements actifs a déjà été soulignée au chapitre précédent. Parmi les algorithmes de construction permettant d'obtenir des ordonnancements actifs, on distingue principalement deux familles, les algorithmes à base de règles de priorité et les algorithmes d'ordre strict. **Sauf mention contraire, nous ne considérerons implicitement que des ordonnancements actifs.**

2.2.1 Problème sans écarts maximaux

2.2.1.1 Algorithme à base de règles de priorité - ARP

Les algorithmes à base de règles de priorité consistent à compléter au fur et à mesure une solution partielle en faisant se dérouler le temps et en plaçant les opérations dès que possible. Parmi toutes les tâches dont les prédécesseurs sont déjà placés, on détermine celle dont la date de fin d'exécution c est la plus petite. Si aucune autre tâche ne peut débuter avant c, on place l'opération sinon on sélectionne parmi les opérations pouvant débuter strictement avant c celle qui est la plus prioritaire. Parmi ces algorithmes, on compte celui de Giffler et Thompson [GT60], et celui de Baker [Bak74]. L'algorithme 2.1 en présente le principe, nous le noterons **ARP** pour **A**lgorithme à base de **R**ègles de **P**riorité.

Algorithme 2.1 ARP - Algorithme à base de règles de priorité

1. L est l'ensemble de toutes les opérations.
2. **Tant que** L n'est pas vide :
   (a) Soit T l'ensemble de toutes les opérations de L qui peuvent être exécutées, c'est-à-dire les opérations dont tous les prédécesseurs sont déjà placés.
   (b) Pour chaque opération de T, calculer les dates de début et de fin d'exécution.
   (c) Dans T trouver l'opération qui aura la plus petite date de fin. On note O cette opération, c sa date de fin et M la machine correspondante.
   (d) Parmi les opérations de T utilisant la machine M et dont la date de début d'exécution est strictement inférieure à c, choisir la plus prioritaire (selon les règles de choix intégrées au générateur). Soit O' cette opération.
   (e) Placer O' sur M, enlever O' de L.

   Fin Tant que.

L'ordre de priorité de chaque tâche peut être statique auquel cas les tâches sont rangées dans une liste et triées selon un ordre de priorité. La priorité de chaque opération est alors donnée par son indice dans la liste, l'opération d'indice 1 étant la plus prioritaire. L'ordre de priorité peut également être dynamique et utiliser des heuristiques pour départager les opérations ordonnançables à un instant donné. Il existe un très grand nombre de règles de priorité plus ou moins complexes. Parmi ces règles, les plus fréquemment utilisées sont :
- choix au hasard,
- choix de l'opération ayant la plus petite (resp. la plus grande) durée opératoire,
- choix de l'opération ayant la plus petite marge libre,
- choix de l'opération ayant la plus petite date échue,
- choix de l'opération ayant le plus de successeurs,
- choix de l'opération ayant la plus petite date de disponibilité,
- ...

Remarque : dans tous les algorithmes, le calcul des dates de début et de fin s'effectue en calant à gauche de sorte à débuter l'exécution de l'opération considérée le plus tôt possible (ici à la fin de l'ordonnancement en cours de construction).

2.2.1.2 Algorithme d'ordre strict - AOS

A la différence de l'algorithme ARP, l'algorithme d'ordre strict défini par Carlier [Car84] place les opérations une à une en respectant scrupuleusement l'ordre d'une liste L. Les opérations sont lues selon l'ordre de cette liste et placées le plus tôt possible en respectant les contraintes de précédence et de ressource. Pour un tel algorithme, le prérequis essentiel est le suivant : si l'opération O_i précède l'opération O_j selon le graphe de précédence alors l'indice de O_i dans L est inférieur à celui de O_j. Contrairement à un algorithme à base de règles de priorité où le temps s'écoule de manière croissante et où les opérations sont placées sur les ressources sans laisser de «trou» réutilisable pour une opération, le placement d'une opération dans un algorithme d'ordre strict peut occasionner un «trou» qui sera éventuellement récupéré par le placement des opérations suivantes. L'algorithme 2.2 décrit le principe d'un algorithme d'ordre strict, nous le noterons **AOS** pour **A**lgorithme d'**O**rdre **S**trict.

AOS présente l'avantage d'effectuer nettement moins de calculs de dates d'exécution des opérations que ARP qui à chaque cycle recalcule ces dates (étape 2.b de l'algorithme 2.1). En contre-partie, AOS n'intègre pas d'heuristique dynamique, l'ordre de placement des opérations étant prédéfini. Du coup, la solution peut être de moins bonne qualité que celle de ARP. Les deux algorithmes construisent tous deux des ordonnancement actifs.
Avec des ressources à calendriers permanents, ARP permet de calculer aisément les dates de début et de fin d'exécution à partir des dates de disponibilité. AOS nécessite de parcourir le profil de capacité de chaque ressource pour trouver un créneau horaire. En présence de calendriers, il sera également nécessaire pour ARP de parcourir le profil de capacité

Algorithme 2.2 AOS - Algorithme d'ordre strict

1. L est l'ensemble de toutes les opérations triées selon l'ordre de précédence et selon les règles de priorité intégrées au générateur. i=1.
2. **Tant que** i $\leq$ |L|
   (a) Soit O la ième opération de L. Placer O le plus tôt possible sur la ressource capable de l'exécuter.
   (b) i = i + 1.
   Fin Tant que

de chaque ressource pour trouver un créneau horaire. Une fois les dates calculées, le placement sur la ressource consistera simplement à mettre à jour le profil de capacité de la ressource.

L'espace des ordonnancements engendrés par les deux algorithmes est identique. Si S est un ordonnancement actif obtenu par ARP, en considérant la liste L des opérations triées selon leur date de début d'exécution dans S et en appliquant AOS à L on retrouvera la même solution S. Inversement, si la priorité des opérations pour ARP est calculée à partir des indices des opérations d'une liste L de AOS, la solution générée par ARP sera identique à celle de AOS. L'espace des solutions est identique mais il est à noter toutefois que la même règle de priorité statique ne donne pas le même ordonnancement selon qu'elle est intégrée dans un algorithme d'ordre strict ou dans un algorithme à base de règles de priorité.

2.2.2 Problème avec écarts maximaux

2.2.2.1 Algorithme à base de règles de priorité modifié - ARP-MD

Les algorithmes ARP et AOS intègrent déjà les écarts minimaux puisque ces derniers n'affectent que les dates de disponibilités. Pour intégrer la gestion des contraintes d'écart maximal, une idée naïve consiste à placer les opérations les unes après les autres comme décrit précédemment mais sans tenir compte dans un 1er temps des écarts maximaux. Lorsqu'on constate qu'en plaçant une opération $O_{i,j}$, une contrainte d'écart maximal n'est plus respectée avec une opération $O_{k,l}$ déjà placée i.e. $s_{i,j} + d(O_{i,j}, O_{k,l}) > s_{k,l}$, alors on dépile toutes les opérations placées depuis $O_{k,l}$ ($O_{k,l}$ incluse), on positionne la date de disponibilité $r_{k,l}$ de $O_{k,l}$ à $s_{i,j} + d(O_{i,j}, O_{k,l})$ et on recommence à ordonnancer les opérations. L'algorithme 2.3 décrit ce principe, nous le désignerons par **ARP-MD** pour **A**lgorithme à base de **R**ègles de **P**riorité **Mo**Difié. Les ordonnancement générés par ARP-MD n'ont aucune garantie d'être actifs, ni même semi-actifs.

Algorithme 2.3 ARP-MD - ARP avec gestion des écarts maximaux

1. L est l'ensemble de toutes les opérations ; R est la pile des opérations déjà placées (initialement R=$\emptyset$).
2. **Tant que** L n'est pas vide :
   (a) Soit T l'ensemble de toutes les opérations de L dont les prédécesseurs (en ne considérant que les arcs positifs) sont placés.
   (b) Pour chaque opération de T, calculer les dates de début et de fin d'exécution.
   (c) Dans T trouver l'opération qui aura la plus petite date de fin. On note O cette opération, c sa date de fin et M la machine correspondante.
   (d) Parmi les opérations de T utilisant la machine M et dont la date de début est strictement inférieure à c, choisir la plus prioritaire (selon les règles de choix intégrées au générateur). Soit O' cette opération.
   (e) Parmi les opérations de R, dresser la liste P des opérations devant respecter une contrainte d'écart maximal avec O'.
   (f) Si toutes les contraintes sont respectées alors placer O' sur M, enlever O' de L, ajouter O' à R.
   (g) Sinon, pour chaque O"$\in$P tel que $s_{O'} + d(O', O'') > s_{O''}$:
      i. dépiler de R toutes les opérations placées depuis O" (O" incluse), les remettre dans L et les retirer des ressources.
      ii. corriger la date de disponibilité $r_{O''}$ de O" à $s_{O'} + d(O', O'')$

   Fin Tant que.

2.2.2.2 Algorithme d'ordre strict modifié - AOS-MD

Pour intégrer la gestion des écarts maximaux dans AOS, on peut procéder de la même manière que pour ARP-MD. Les opérations sont placées les unes après les autres en ne tenant compte que des écarts minimaux (qui n'influent que sur les dates de disponibilités). Lorsqu'on constate qu'en plaçant une opération $O_{i,j}$, une contrainte d'écart maximal n'est plus respectée avec une opération $O_{k,l}$ déjà placée i.e. $s_{i,j} + d(O_{i,j}, O_{k,l}) > s_{k,l}$, alors on dépile toutes les opérations placées depuis $O_{k,l}$ ($O_{k,l}$ incluse), on positionne la date de disponibilité $r_{k,l}$ de $O_{k,l}$ à $s_{i,j} + d(O_{i,j}, O_{k,l})$ et on recommence le placement depuis l'opération $O_{k,l}$. L'algorithme 2.4 intègre cette modification, nous le noterons AOS-MD pour **A**lgorithme d'**O**rdre **S**trict **Mo**Difié. Cet algorithme ne garantit pas non plus que les ordonnancements construits soient actifs, ni même semi-actifs.

Le fait d'avoir imposé l'hypothèse **H0** nous permet d'affirmer qu'en cas de terminaison de l'algorithme, la solution obtenue est valide. Dans le cas contraire, on peut très bien

Algorithme 2.4 AOS-MD - AOS avec gestion des écarts maximaux

1. L est l'ensemble de toutes les opérations triées selon l'ordre de précédence et selon les règles de priorité intégrées au générateur. i=1.
2. **Tant que** i $\leq$ |L|
   (a) Soit O la ième opération de L. Calculer la date de début de O tel que celle-ci soit la plus petite possible. Soit M la ressource de O.
   (b) Parmi les opérations de L, d'indice inférieur à i, dresser la liste P des opérations devant respecter une contrainte d'écart maximal avec O.
   (c) Si toutes les contraintes sont respectées alors placer O sur M, i=i+1.
   (d) Sinon, pour chaque O'$\in$P tel que $s_O + d(O, O') > s_{O'}$:
      i. retirer des ressources toutes les opérations placées depuis O' (O' incluse)
      ii. corriger la date de disponibilité $r_{O'}$ de O' à $s_O + d(O, O')$

   Positionner i au plus petit indice des opérations de P.

Fin Tant que.

obtenir des solutions non valides. Considérons l'exemple suivant :
- 3 opérations $\{O_1, O_2, O_3\}$ et 2 ressources $\{M_1, M_2\}$;
- O_1 et O_2 s'exécutent sur M_1, O_3 sur M_2 ;
- $p_1 = 2$, $p_2 = 1$ $p_3 = 1$;
- O_3 et O_2 sont soumis à la contrainte de potentiels $d(O_3, O_2) = -1$ (O_3 doit débuter son exécution au plus tard 1 unité de temps après la date de début d'exécution de O_2) ;
- la liste d'ordre strict est $L = \{O_1, O_2, O_3\}$;

Comme nous n'avons pas imposé de chemin constitué d'arcs positifs entre O_2 et O_3, la liste d'ordre strict L est valide. L'algorithme procédera au placement de O_1 puis de O_2 sur M_1. La date de début d'exécution s_3 calculée par l'algorithme sera 0 et violera la contrainte d'écart maximal avec O_2. L'algorithme dépilera O_2 et O_3, recalculera la date de disponibilité de O_2 alors que c'est celle de O_3 qui devrait être repositionnée. Indéfiniment l'algorithme AOS-MD fournira la solution de la figure 2.1.

2.2.3 Phénomène du «saute-mouton» et non convergence

Si les algorithmes ARP-MD et AOS-MD se mettent facilement en oeuvre et permettent d'adapter des algorithmes à base de règles de priorité ou d'ordre strict existants sans trop d'efforts, ils présentent cependant un très gros inconvénient. Ils peuvent tous deux occasionner un nombre inacceptable de dépilement d'opérations. Comme à chaque dépi-

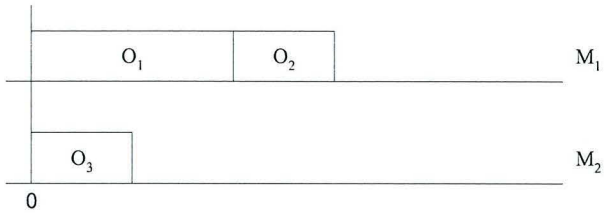


FIG. 2.1 – Solution non valide obtenue par AOS-MD en l'absence de H0

lement les dates de disponibilité sont incrémentées, les opérations peuvent être retardées plusieurs fois sans justification. Considérons $O_{i,j}$ et $O_{k,l}$ liées par un arc négatif. Après le placement de $O_{i,j}$, l'algorithme peut procéder au placement d'autres opérations utilisant la même ressource que $O_{k,l}$. Avec le placement tardif de $O_{k,l}$, l'inégalité de potentiel $s_{k,l}+d(O_{k,l}, O_{i,j}) \leq s_{i,j}$ n'est plus satisfaite. L'algorithme dépile les opérations et décale $r_{i,j}$ dans le futur. Les opérations mises ainsi en concurrence peuvent se décaler l'une l'autre, se violer mutuellement les inégalités de potentiels et dans certains cas engendrer un phénomène particulier que nous avons appelé *«saute-mouton»* lorsque deux couples d'opérations en concurrence se violent mutuellement les inégalités de potentiels et se «sautent» l'une par-dessus l'autre. De tels phénomènes produisent des solutions dégradées avec de nombreux «trous» dans le Gantt dans le sens où la solution obtenue n'est pas calée gauche et est très mauvaise en terme de makespan (exemple figure 2.2).

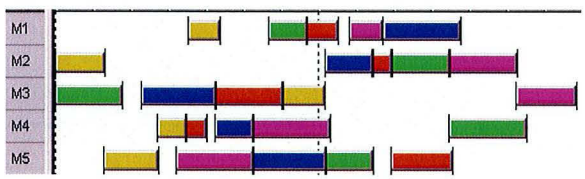


FIG. 2.2 – Solution dégradée de l'algorithme ARP-MD

Exemple de phénomène du «saute-mouton» avec ARP-MD. Considérons 2 ressources, 2 travaux dont les temps opératoires et les affectations sont données dans le tableau 2.1.

opération	$O_{1,1}$	$O_{2,1}$	$O_{1,2}$	$O_{2,2}$
ressource	M_1	M_2	M_1	M_2
durée	1	3	1	3

TAB. 2.1 – Temps opératoires et affectations

On considère les inégalités de potentiels $s_{2,1} - 4 \leq s_{1,1}$ et $s_{2,2} - 4 \leq s_{1,2}$. Entre les deux opérations successives de chaque travail il ne doit pas y avoir de temps d'attente supérieur à 3 unités de temps. La règle de priorité est le choix aléatoire de l'opération. La figure 2.3 illustre les 3 étapes du phénomène.

1ère étape : nous supposons ici que le choix aléatoire place $O_{1,1}$ puis $O_{1,2}$ sur M_1 et $O_{2,2}$ puis $O_{2,1}$ sur M_2. L'inégalité de potentiel entre $O_{1,1}$ et $O_{2,1}$ n'étant pas satisfaite, on décale la date de disponibilité de $O_{1,1}$ d'une unité de temps (quantité de temps manquante pour satisfaire la contrainte d'écart maximal), on dépile toutes les opérations et on recommence.
2ème étape : on débute obligatoirement le placement par $O_{1,2}$ puis $O_{1,1}$ sur M_1. Nous supposons maintenant que le choix aléatoire place $O_{2,1}$ puis $O_{2,2}$ sur M_2. L'inégalité de potentiel entre $O_{2,1}$ et $O_{2,2}$ n'étant pas satisfaite, on décale la date de disponibilité de $O_{1,2}$ d'une unité de temps, on dépile toutes les opérations et on recommence.
3ème étape : on se retrouve dans la situation initiale mais les dates de disponibilité des opérations $O_{1,1}$ et $O_{1,2}$ ont toutes deux été décalées d'une unité de temps. Le cycle recommence et on peut effectuer le même placement que lors de la 1ère étape et ainsi de suite. Le résultat est un ordonnancement non calé gauche et il n'y a aucune garantie que l'on ne va pas boucler.

Pour éviter un tel phénomène, il aurait suffi après le placement de $O_{1,1}$ de procéder immédiatement à celui de $O_{2,1}$ sans se soucier des règles de priorité pouvant être violées.

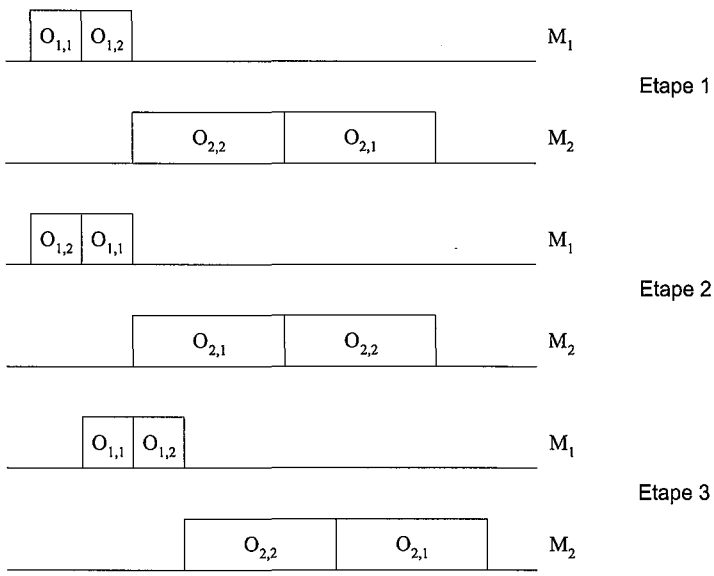


FIG. 2.3 – Déroulement du phénomène de saute-mouton

2.2.4 Décomposition par grappes

Afin de limiter la dégradation de la solution, nous proposons l'algorithme 2.5 qui repose sur un placement par grappes. On ne place plus individuellement une opération après une autre mais des grappes d'opérations les unes après les autres. Ces grappes sont simplement les composantes fortement connexes du graphe orienté (V, C) (graphe conjonctif).

Celles-ci partitionnent l'ensemble des opérations et sont liées par une relation de précédence induite par celle sur les opérations : *la grappe G précède la grappe G' si et seulement s'il existe une opération $O \in G$, une opération $O' \in G'$ et un chemin à arcs positifs de O à O'*. Ceci est équivalent à l'existence d'un chemin entre G et G' dans le graphe réduit.

Les éléments allogènes pouvant violer les écarts maximaux sont rejetés à l'extérieur d'une grappe. Un tel algorithme minimise les conflits au niveau des opérations et diminue la combinatoire. La construction d'une solution valide reste NP-complet mais réduite au niveau d'une grappe. De plus, il existe une solution valide si et seulement s'il en existe une pour chaque grappe. Bien que les travaux de cette thèse ait été effectués de manière indépendante, l'idée d'une telle décomposition n'est pas nouvelle et a déjà été évoquée en 1995 par Brinkmann et Neumann [BN95] dans un rapport technique de l'université de Karlsruhe. Plus récemment, Franck et al [FNS01] ont publié un panorama de leurs méthodes incluant en particulier une approche de décomposition par cycles (composantes fortement connexes) et l'ordonnancement de chaque cycle par un algorithme à base de règles de priorité.

Algorithme 2.5 ARP-G - Algorithme de placement par grappes à base de règles de priorité

1. L est l'ensemble de toutes les grappes.
2. **Tant que** L n'est pas vide :
   (a) Soit T l'ensemble de toutes les grappes de L qui peuvent être exécutées, c'est-à-dire les grappes dont les grappes prédécesseurs sont toutes placées.
   (b) Pour chaque grappe de T, calculer les dates de début et de fin d'exécution.
   (c) Dans T trouver la grappe qui aura la plus petite date de fin. On note G cette grappe, c sa date de fin.
   (d) Parmi les grappes de T dont la date de début est strictement inférieure à c, choisir la plus prioritaire. Soit G' cette grappe.
   (e) Placer la grappe G', enlever G' de L.

   Fin Tant que.

Remarques :

- La date de début (resp. fin) d'exécution d'une grappe est la plus petite date de début (resp. fin) d'exécution des opérations de la grappe.
- La priorité d'une grappe peut être la priorité moyenne de ses opérations, la plus grande (resp. petite) priorité des opérations de la grappe, le nombre d'opérations de la grappe etc..

- En l'absence de contraintes d'écart maximal, chaque grappe est réduite à une opération et on retrouve les algorithmes ARP et AOS classiques.

Algorithme 2.6 AOS-G - Algorithme de placement par grappes d'ordre strict

1. L est l'ensemble de toutes les grappes triées selon l'ordre de précédence et selon les règles de priorité intégrées au générateur. i=1.
2. **Tant que** i $\leq$ |L|
   (a) Soit G la ième grappe de L. Placer la grappe G.
   (b) i = i + 1.
   Fin Tant que

Il reste à expliciter le processus d'ordonnancement d'une grappe utilisé pour le calcul des dates d'exécution d'une grappe en 2.b pour l'algorithme ARP-G 2.5 et 2.a pour l'algorithme AOS-G 2.6. Pour ordonnancer une grappe, on peut réutiliser l'algorithme ARP-MD 2.3 ou l'algorithme AOS-MD 2.4. Les algorithmes obtenus sont notés :

ARP-G-RP algorithme par grappes à base de règles de priorité utilisant un algorithme à base de règles de priorité pour le placement de chaque grappe ;

ARP-G-OS algorithme par grappes à base de règles de priorité utilisant un algorithme d'ordre strict pour le placement de chaque grappe ;

AOS-G-RP algorithme par grappes d'ordre strict utilisant un algorithme à base de règles de priorité modifié pour le placement de chaque grappe ;

AOS-G-OS algorithme par grappes d'ordre strict utilisant un algorithme d'ordre strict modifié pour le placement de chaque grappe ;

2.3 Convergence des algorithmes sous certaines hypothèses

Dans le cas général, les algorithmes ARP-G-RP, ARP-G-OS, AOS-G-RP et AOS-G-OS n'ont aucune garantie de converger. Même si les conflits entre opérations sont limités au sein des opérations de la grappe, ceux-ci sont toujours présents et nous restons soumis à des phénomènes du type saute-mouton. Pour assurer une terminaison à l'algorithme, on impose un horizon de planification $[h; H]$ (sans perte de généralité, on peut prendre $h = 0$). Dans l'intervalle de planification, toutes les contraintes doivent être respectées. Au-delà, on ignore les contraintes et on procède à un placement simple sans se soucier des

contraintes éventuelles pouvant être violées (que ce soit des contraintes d'écart maximal ou des contraintes de ressource).

Si cette solution peut paraître artificielle, elle se justifie par les méthodes industrielles de planification sur des périodes glissantes. Comme pour les phénomènes météorologiques, les prévisions sont correctes sur une période donnée mais vont diverger très rapidement en fonction des aléas de l'atelier. Il est dans ce cas illusoire d'effectuer une planification fine à long terme et espérer s'en tenir à ces prévisions. Planifier en respectant toutes les contraintes sur une période déterminée puis relâcher les contraintes pour obtenir une approximation de la charge prévisionnelle nous semble être une approche plus pragmatique. Cet horizon de planification reste un paramètre à la discrétion de l'utilisateur.

Néanmoins, sous certaines hypothèses, on peut démontrer la convergence des algorithmes ARP-MD et AOS-MD.

Considérons les hypothèses suivantes :

H0 : Si dans le graphe conjonctif (V, C) il existe un arc négatif orienté de O à O' alors il existe également un chemin de O' à O constitué exclusivement d'arcs positifs.

H1 : Il existe dans le graphe conjonctif (V, C) un ordre total entre les opérations qui utilisent la même ressource.

H2 : Il existe une solution valide.

Remarque : l'hypothèse H1 stipule que l'ensemble D des arcs disjonctifs est vide, ou encore que le graphe conjonctif (V, C) est un graphe disjonctif arbitré (voir chapitre 1, paragraphe 1.8).

Théorème 2.1 *Sous les hypothèses H0, H1 et H2, les algorithmes AOS-MD et ARP-MD sont convergents et construisent un ordonnancement actif.*

Démonstration
Considérons l'algorithme AOS-MD, $L = (O_1, ..O_n)$ est la liste d'ordre strict des opérations. Notons s_i la date de début d'exécution de O_i et r_i sa date de disponibilité.
Les hypothèses H1 et H2 permettent d'affirmer qu'il existe une solution valide minimale $S^* = (s_1^*, .., s_n^*)$ dans le sens où, pour toute autre solution valide $S' = (s_1', .., s_n')$ on a $s_1^* \leq s_1', .., s_n^* \leq s_n'$.

Pour démontrer le théorème, nous allons démontrer le résultat intermédiaire suivant : *«les dates de début d'exécution* s_i $(i = 1, .., n)$ *calculées par l'algorithme AOS-MD sont inférieures à celles de* S^**»*, ce qui se traduit par

$$\forall i = 1, .., n \ : \ s_i \leq s_i^*$$

Sans perte de généralité, on peut considérer que les dates de début d'exécution sont initialisées à 0.
Pour démontrer le résultat précédent nous allons raisonner par récurrence sur le nombre de placements d'opérations. Considérons l'hypothèse de récurrence au rang k
HR(k)= «lors du kème placement, la date de début d'exécution calculée de l'opération considérée est inférieure ou égale à sa date de début d'exécution dans S^**»*.

L'hypothèse de récurrence est clairement vérifiée lors du placement de la 1ère opération. On suppose l'hypothèse de récurrence vraie jusqu'au rang k.

Considérons le $k+1$ ème placement d'opération. Soit O_i l'opération considérée et M la ressource associée. Soit l'ensemble $E_i = \{O_j; j \neq i \text{ et } d(O_j, O_i) \geq 0)\}$. L'ensemble E_i contient en particulier l'opération qui précède O_i sur M d'après H1. Soit

$$s = \max_{E_i}(s_j + d(O_j, O_i))$$

Si $E_i = \emptyset$, posons $s = 0$. L'opération O_i est placée au plus tôt sur M à partir de la date $max(s, r_i)$. La date de début s_i ainsi calculée est la plus petite date de début d'exécution possible de l'opération O_i sur M. En effet, d'après H1, il n'existe aucune autre opération déjà placée sur M qui pourrait retarder O_i.

1er cas : $r_i \leq s$
Soit $O_x \in E_i$ tel que $s = s_x + d(O_x, O_i) = max_{E_i}(s_j + d(O_j, O_i))$. Le placement de O_x ayant déjà eu lieu, d'après l'hypothèse de récurrence, $s_x \leq s_x^*$. L'inégalité $s_x^* + d(O_x, O_i) \geq s_x + d(O_x, O_i)$ est donc vérifiée. s_i étant la plus petite date de début d'exécution possible de l'opération O_i sur M à partir de la date $s = s_x + d(O_x, O_i)$, nous avons effectivement $s_i^* \geq s_i$. L'hypothèse de récurrence est bien vérifiée au rang $k+1$.

2ème cas : $r_i > s$
Si r_i est encore égal à sa valeur initiale i.e. r_i n'a pas encore été modifié suite à la violation d'une contrainte d'écart maximal, on a trivialement $s_i \geq s_i^*$.
Dans le cas contraire, il existe une opération O_y placée précédemment qui a occasionné le dépilement de O_i et modifié la valeur de r_i. Dans ce cas, $r_i = s_y + d(O_y, O_i)$ avec $d(O_y, O_i) < 0$. D'après l'hypothèse de récurrence, $s_y + d(O_y, O_i) \leq s_y^* + d(O_y, O_i)$. s_i étant la plus petite date de début d'exécution possible de l'opération O_i sur M à partir de la date $s_y + d(O_y, O_i)$, nous avons effectivement $s_i^* \geq s_i$. L'hypothèse de récurrence est également vérifiée au rang $k+1$.

Dans les deux cas, l'hypothèse de récurrence est vérifiée au rang $k+1$ donc pour toute opération O_i, $s_i \leq s_i^*$.

Ce résultat nous permet d'en déduire le théorème. La suite $(s_1, .., s_n)$ des dates de début d'exécution des opérations est une suite strictement croissante et majorée par $(s_1^*, .., s_n^*)$ donc convergente. La suite $(s_1, .., s_n)$ étant de plus discrète, la convergence s'effectue en un nombre fini de placements d'opérations. La solution finale ne pouvant qu'être valide, la limite de chaque suite (s_i) est égale à s_i^*. La solution obtenue est semi-active et active par défaut d'après H1.

La démonstration pour ARP-MD est identique. La seule différence réside dans l'ordre de placement des opérations qui n'est pas identique à la liste d'ordre strict L. L'hypothèse H1 nous garantit cependant que cette différence n'affecte que les opérations qui s'exécutent sur des ressources différentes. Les dates calculées sont donc rigoureusement identiques. □

L'hypothèse H1 est trop restrictive pour être réaliste, un ordre total pour l'ensemble des opérations n'est pas une hypothèse raisonnable. Cependant, au sein d'une grappe, cette hypothèse est tout à fait valable. Le placement des opérations d'une grappe ne remet nullement en cause les opérations des autres grappes déjà placées. Les opérations des grappes déjà placées jouent le rôle de période d'indisponibilité sur le calendrier de la ressource. De ce fait, l'hypothèse H1 se trouve automatiquement satisfaite pour le placement de chaque grappe. Si de plus, l'hypothèse H2 est satisfaite lors du placement de chaque grappe, ARP-G et AOS-G construisent un ordonnancement actif.

Les algorithmes ARP-G et AOS-G peuvent également être utilisés pour des problèmes de flowshop avec contraintes de succession sans attente au niveau des ressources [SGM03], c'est-à-dire lorsque les différentes ressources doivent enchaîner l'exécution des opérations sans temps mort. Si on fixe un ordre de succession des opérations sur les machines, un ordre identique dans le cas du flowshop de permutation, les m grappes sont alors constituées par les opérations devant s'exécuter sur chaque ressource. Les hypothèses H0, H1 et H2 sont satisfaites pour chaque grappe. Les algorithmes ARP-G et AOS-G permettent donc de construire un ordonnancement actif. Le déroulement des algorithmes ARP-G et AOS-G va consister à placer les opérations sur la première ressource puis sur la seconde et ainsi de suite.

Le coût des algorithmes ARP-MD et AOS-MD en terme de nombre de calculs de dates d'exécution est difficilement quantifiable. Il est cependant possible d'avoir un ordre de grandeur. Considérons en sus des hypothèses H0, H1 et H2, l'hypothèse H3 :

H3 : Les ressources sont disponibles à partir de t=0 (les calendriers des ressources sont permanents).

Algorithme 2.7 Algorithme de placement d'une grappe sous H0, H1, H2, H3

1. Soit L la liste d'ordre strict des opérations de la grappe G. O_i désigne la ième opération de L, s_i sa date de début d'exécution.
2. Ordonnancer les opérations de L par un algorithme d'ordre strict en négligeant les contraintes d'écart maximal.
3. Flag1=faux
4. Tant que Flag1=faux :
   (a) Flag2=vrai.
   (b) Pour i variant de 1 à $|G|$:
      i. $s = max_j(s_j + d(O_j, O_i); j \neq i)$
      ii. Si $s > s_i$ alors $s_i = s$ et Flag2=faux.
   (c) Flag1=Flag2.

   Fin Tant que.

Théorème 2.2 *Sous les hypothèses H0 ,H1, H2 et H3, l'algorithme 2.7 permet de construire une solution valide pour une grappe G en $O(|G|^2)$ calculs de dates d'exécution.*

Démonstration
Les valeurs s_i calculées lors de l'étape 2 de l'algorithme 2.7 correspondent à la longueur d'un chemin P_i du graphe conjonctif de la racine à O_i.

Nous allons démontrer le lemme suivant :
« Si la date de début s_i de la ième opération O_i de L est incrémentée lors du xème passage dans la boucle 4.b de l'algorithme 2.7 alors il existe, après incrémentation, un chemin P_i de la racine à O_i de longueur s_i possédant exactement x arcs négatifs».

Pour démontrer le lemme on procède par récurrence sur x.

Pour $x = 1$.
Avant le passage dans 4.b, tous les chemins P_i ne contiennent que des arcs positifs puisque les premières dates d'exécution ont été calculées lors de l'étape 2 de l'algorithme 2.7 en négligeant les arcs négatifs. Considérons une opération O_i pour laquelle le s_i vient juste d'être incrémenté lors du passage dans 4.b. Soit O_k une des opérations telle que $s_k + d(O_k, O_i) = max_j(s_j + d(O_j, O_i); j \neq i)$.

1er cas : $k > i$ (selon la numérotation retenue pour les opérations)
La valeur de $d(O_k, O_i)$ ne peut être que négative, le cas contraire est impossible et contredirait le fait que $k > i$. En considérant le chemin P_k de la racine à O_k plus l'arc négatif de O_k à O_i on obtient un chemin de longueur s_i contenant exactement 1 arc négatif.

2ème cas : $k < i$
La valeur de $d(O_k, O_i)$ ne peut être que positive. Dans le cas contraire, il existerait un chemin constitué d'arcs positifs de O_i vers O_k (hypothèse H0) ce qui contredirait le fait que $k < i$. La date s_k vient d'être modifiée car dans le cas contraire on n'aurait pas à incrémenter s_i puisque le calcul initial de s_i lors de l'étape 2 a tenu compte des écarts minimaux. On peut appliquer le 1er cas à s_k : il existe un chemin depuis la racine jusqu'à O_k possédant 1 arc négatif. En considérant le chemin P_k de la racine à O_k plus l'arc positif de O_k à O_i on obtient un chemin de longueur s_i contenant exactement 1 arc négatif.

On suppose l'hypothèse de récurrence vraie au rang $x > 1$. Considérons s_i incrémenté lors du $x + 1$ ème passage dans 4.b.
Soit O_k une des opérations tel que $s_k + d(O_k, O_i) = max_j(s_j + d(O_j, O_i); j \neq i)$.

1er cas : $k > i$
O_k est successeur de O_i, $d(O_k, O_i)$ est négatif et s_k a été modifié au tour x. En effet, si s_k n'avait pas été modifié au tour x, la valeur de s_i calculée au tour x serait toujours valide. En appliquant l'hypothèse de récurrence à s_k, il existe un chemin P_k de la racine à O_k de longueur s_k et possédant exactement x arcs négatifs. En considérant le chemin P_k plus l'arc négatif de O_k à O_i on obtient un chemin de longueur s_i contenant exactement $x + 1$ arcs négatifs.

2ème cas : $k < i$
O_k est prédécesseur de O_i, $d(O_k, O_i)$ est positif et s_k vient d'être modifié au tour $x + 1$. En effet, au tour précédent, le recalcul de s_i intervient après celui de s_k. Si au tour $x + 1$, s_i doit de nouveau être incrémenté cela signifie que s_k vient également d'être modifié lors du tour $x + 1$. On peut appliquer le 1er cas à s_k : il existe un chemin P_k de la racine à O_k de longueur s_k possédant $x + 1$ arcs négatifs. En considérant le chemin P_k plus l'arc positif de O_k à O_i on obtient un chemin de longueur s_i contenant exactement $x + 1$ arcs négatifs.

Ceci achève la démonstration du lemme.

La terminaison de l'algorithme sous H0, H1, H2 et H3 découle du lemme. Le graphe est sans circuit de longueur strictement positive (H2), tout chemin élémentaire de la racine à un sommet possède au plus $|G|$ arcs. Il y aura donc au plus $|G|$ passages dans la boucle 4 ($|G| + 1$ pour la vérification finale). Chaque passage dans la boucle correspondant au calcul d'au plus $|G|$ dates d'exécution d'opérations, nous obtenons bien une complexité en $O(|G|^2)$. □

Remarque : le résultat précédent peut encore être affiné si on impose la «transitivité» sur les écarts maximaux : si le couple d'opérations (O_i, O_j) est soumis à une contrainte d'écart maximal $d(O_i, O_j) < 0$ et si le couple d'opérations (O_j, O_k) est également soumis

à une contrainte d'écart maximal $d(O_j, O_k) < 0$ alors le couple d'opérations (O_i, O_k) est soumis à une contrainte d'écart maximal telle que $0 > d(O_i, O_k) \geq d(O_i, O_j) + d(O_j, O_k)$. Dans ces conditions les chemins utilisés pour incrémenter les dates d'exécution ne comportent pas deux arcs négatifs consécutifs et il y aura au plus $\frac{1}{2}|G|$ passages dans la boucle 4.

Comme le placement de la première grappe fait disparaître l'hypothèse H3 car elle crée des «trous» dans le calendrier, le résultat précédent nous permet d'obtenir seulement un estimateur pour le nombre de calculs de dates d'exécution de l'algorithme AOS-G. Pour les problèmes d'atelier de type job shop ou flowshop, si les contraintes temporelles entre opérations sont définies au sein d'opérations du même travail, $n.m^2$ est une estimation du nombre de calculs des dates d'exécution de AOS-G pour les problèmes à n travaux et m ressources. L'algorithme ARG-G nécessite quant à lui beaucoup plus de calculs de dates d'exécution puisque nous sommes contraints, après chaque grappe placée, de recalculer les dates d'exécution de chaque grappe ordonnançable pour trouver celle avec la plus petite date de fin d'exécution.
Si l'on se restreint exclusivement aux flowshops de permutation, il n'y a plus de «trous» dans les calendriers à partir de la date de fin de la dernière opération sur chaque ressource. L'hypothèse H3 est vérifiée pour le placement de chaque grappe et $n.m^2$ n'est plus simplement un estimateur mais l'ordre de grandeur du nombre de calculs de dates d'exécution de AOS-G.

2.4 Expériences sur des gammes linéaires

2.4.1 Génération des exemples et méthodes utilisées

Les versions modifiées et les versions par grappes des algorithmes à base de règles de priorité et d'ordre strict ont été comparées sur plusieurs familles de problèmes de type jobshop de taille 5x5, 5x10, 5x20, 10x10, 10x20, 10x30.
Les jeux d'essai considérés ne comportent que des gammes linéaires, les grappes ne sont constituées que d'opérations successives au sein d'un même travail. L'algorithme de placement d'une grappe, qu'il soit à base de règle de priorité ou d'ordre strict, donnera le même résultat, puisqu'on place les opérations dans l'ordre de la gamme, et nécessitera le même nombre de calculs de dates d'exécution dans les deux cas. ARP-G-RP et ARP-G-OS (resp. AOS-G-RP et AOS-G-OS) sont équivalents, nous les désignerons par ARP-G (resp. AOS-G).
Les algorithmes qui ont été comparés sont :
- ARP-MD ;
- AOS-MD ;
- ARP-G ;
- AOS-G.

2.4.1.1 Temps opératoires

Les temps opératoires $p_{i,j}$ ont été générés aléatoirement entre deux bornes $p_{min} = 10$ et $p_{max} = 50$. Nous n'avons pas considéré d'autres écarts minimaux que les précédences classiques. Comme nous nous intéressons à la construction d'une solution faisable en présence d'écarts maximaux et comme les temps d'attente minimaux n'ont pas d'impact sur la faisabilité d'une solution, nos jeux d'essai n'en comportent pas. L'horizon de planification a été fixé à $[0; 2.n.p_{max}]$.

2.4.1.2 Ecarts maximaux

Pour chaque travail j, 1, 2 ou 3 couples d'opérations $(O_{i,j}, O_{k,j})$ $(i < k)$ ont été choisis aléatoirement avec un écart maximal :

$$d(O_{i,j}, O_{k,j}) = -C \sum_{a=i}^{k-1} p_{a,j}$$

C est une constante. La valeur initiale de C a été fixée à 3. Les performances des algorithmes ont été mesurées pour des valeurs de C régulièrement espacées entre 3 et 1. Nous avons ainsi pu mesurer le comportement des algorithmes en fonction de la «tension» sur les écarts maximaux, de la plus faible $(C = 3)$ à la plus forte $(C = 1)$ qui correspond à un problème sans attente.

2.4.1.3 Critères de comparaison

Les deux critères de comparaison ont été l'évolution du nombre moyen de calculs de dates d'exécution et l'évolution de la moyenne des makespan en fonction de la tension C.

2.4.1.4 Règle de priorité pour ARP-MD

Pour ARP-MD, la règle de priorité utilisée est la priorité à l'opération ayant la plus petite «marge» disponible par rapport aux écarts maximaux. Soit $O_{i,j}$ une opération dont on souhaite connaître la priorité et P le sous-ensemble des opérations déjà placées qui doivent respecter une contrainte d'écart maximal avec $O_{i,j}$. La priorité de $O_{i,j}$ est définie par

$$Max\{s_{k,l} - d(O_{i,j}, O_{k,l}); O_{k,l} \in P \text{ et } d(O_{i,j}, O_{k,l}) < 0\}$$

2.4.1.5 Liste d'ordre strict pour AOS-MD

Pour AOS-MD, une liste d'ordre strict a été générée. Si la liste est triée selon les ordres de précédence des opérations tel que toutes les opérations d'un même travail soient également successives dans la liste, l'algorithme d'ordre strict modifié donne des résultats similaires à un algorithme par grappes. Nous avons donc légèrement «entrelacé» les opérations des travaux de sorte à avoir une liste qui ne consiste pas simplement à placer les travaux les uns après les autres.

Pour mesurer cet entrelacement nous avons créé la mesure ε. Si $(O_{i,j}, O_{k,j})$ est un couple d'opérations du travail j liées par une contrainte d'écart maximal $d(O_{i,j}, O_{k,j}) < 0$, on comptabilise pour ce couple le nombre $N(O_{i,j}, O_{k,j})$ d'opérations n'appartenant pas au travail j et positionnées dans la liste d'ordre strict entre $O_{i,j}$ et $O_{k,j}$. ε est la somme sur l'ensemble de ces couples des $N(O_{i,j}, O_{k,j})$, le tout divisé par le nombre total de couples considérés.
ε représente le nombre moyen d'opérations se situant dans la liste d'ordre strict entre deux opérations liées par une contrainte d'écart maximal. Pour le couple $(O_{i,j}, O_{k,j})$ par exemple, les opérations situées dans la liste d'ordre strict entre $O_{i,j}$ et $O_{k,l}$ peuvent, si elles utilisent la même ressource que $O_{k,l}$, retarder le placement de cette dernière opération et violer la contrainte d'écart maximal. Plus le nombre d'opérations intermédiaires est élevé et plus le risque de violer la contrainte est important.
La mesure ε n'est significative que si on compare des jeux d'essai de taille identique. Cette mesure n'est pas parfaite, elle est intuitive mais il existe une forte corrélation entre ε et le phénomène du «saute-mouton». Plus ε est petit et moins la liste sera entrelacée avec pour conséquence une minimisation du nombre de conflits générant le phénomène de «saute-moton». Plus ε est grand et plus les conflits risquent d'être importants et le «saute-mouton» marqué. Nous avons tenté de réfléchir à une mesure plus précise. Pour être tout à fait rigoureux, il ne faudrait comptabiliser que les opérations utilisant la même ressource que $O_{k,l}$. Encore faudrait-il que le placement de ces opérations affecte effectivement celui de $O_{k,l}$ c'est-à-dire que ces opérations aient des dates de fin d'exécution après la date de fin d'exécution de $O_{i,j}$ ce qui ne peut être connu qu'a posteriori. Il est donc difficile d'obtenir une mesure précise a priori, c'est pourquoi nous avons préféré conserver la mesure ε que nous avons jugée simple, intuitive et suffisante pour nos jeux d'essai.

Pour générer une liste d'ordre strict, nous avons affecté aléatoirement une priorité entre 1 et pr_{max} à chaque opération. La liste des opérations est ensuite triée selon l'ordre de précédence et la priorité des opérations. La valeur de ε est mesurée a posteriori. Plus pr_{max} est grand et plus la liste sera entrelacée. Les valeurs indiquées sur les figures (exemple $\varepsilon = 5.36$ sur la figure 2.4) correspondent à la mesure ε moyenne sur l'ensemble des jeux d'essai.

2.4.1.6 Priorité d'une grappe pour ARP-G

Nous avons affecté aléatoirement une priorité fixe entre 1 et 100 à chaque opération. Pour les algorithmes par grappes, la priorité d'une grappe est la priorité maximale des opérations qui constituent la grappe.

2.4.1.7 Liste d'ordre strict des grappes

La liste d'ordre strict utilisée pour les grappes par AOS-G est triée selon l'ordre de précédence définie sur les grappes et selon l'ordre de priorité définie par la liste d'ordre strict L

de AOS-MD. La priorité d'une grappe est égale au plus petit indice dans L des opérations de la grappe. Plus cet indice est petit et plus prioritaire sera la grappe.

2.4.1.8 Lecture des figures

Toutes les figures des tests correspondants se trouvent en annexe et l'analyse des résultats au paragraphe suivant. Nous avons fait figurer ci-dessous les courbes pour les jeux d'essai 10x20 que nous avons jugées représentatives des comportements des algorithmes.

La figure 2.4 représente le nombre moyen de calculs de dates d'exécution sur l'ensemble des 30 jeux d'essai de taille 10x20 pour chaque valeur de C :
abscisse : tension C sur les écarts maximaux ;
ordonnée : logarithme népérien du nombre moyen de calculs des dates d'exécution ;
droite horizontale rouge (repérable par une flèche à droite du dessin pour les reproductions noir et blanc) : estimateur du nombre de calculs de dates d'exécution $ln(n.m^2)$.

La figure 2.5 permet de visualiser, pour chaque algorithme, l'évolution du makespan moyen sur l'ensemble des 30 jeux d'essai de taille 10x20 pour chaque valeur de C :
abscisse : tension C sur les écarts maximaux ;
ordonnée : makespan moyen sur les 30 jeux d'essai de taille 10x20.

La figure 2.6 représente pour chaque valeur de C le nombre de solutions valides obtenues par les algorithmes ARP-MD et AOS-MD sur l'ensemble des 30 jeux d'essai de taille 10x20 :
abscisse : tension C sur les écarts maximaux ;
ordonnée : nombre de solutions valides obtenues par chacun des algorithmes.
La troisième courbe (*succès cumulés*) dénombre le nombre total de constructions valides ayant été obtenues par l'un ou l'autre algorithme.

La figure 2.7 reprend les mêmes informations que la figure 2.5 mais en ne tenant compte que des constructions valides.

2.4.2 Analyse des résultats

2.4.2.1 Nombre moyen de calculs de dates d'exécution

Comme on pouvait s'y attendre, le nombre de calculs de dates d'exécution des opérations est bien plus élevé pour l'algorithme à base de règles de priorité que pour celui d'ordre strict, que ce soit en comparant les versions par grappes ou les versions modifiées. Si ARP-G et AOS-G connaissent une progression normale du nombre moyen de calculs lorsque la tension augmente, leurs homologues ARP-MD et AOS-MD voient le nombre moyen de calculs augmenter de manière significative et exploser pour les fortes tensions.

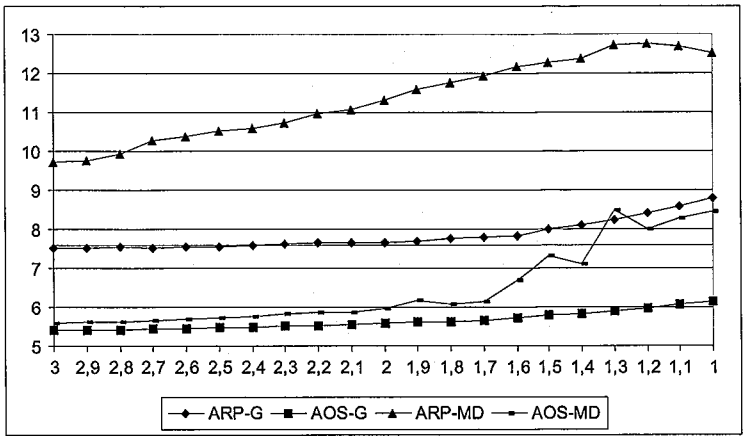


FIG. 2.4 – Nombre moyen de calculs de dates d'exécution m=10, n=20, $\varepsilon = 5.36$

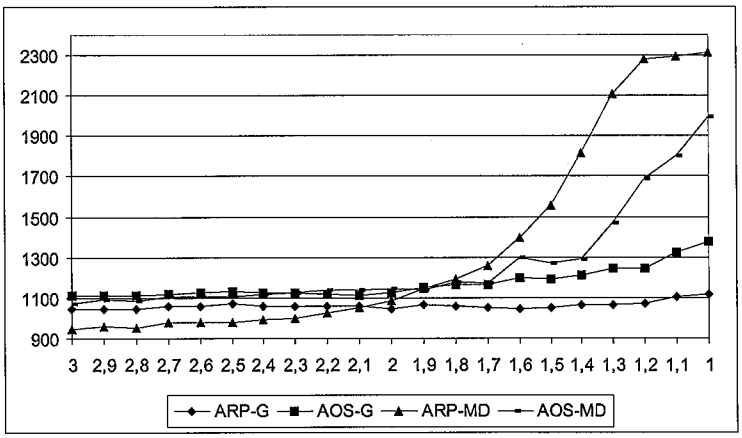


FIG. 2.5 – Makespan moyen m=10, n=20, $\varepsilon = 5.36$ (ensemble des instances)

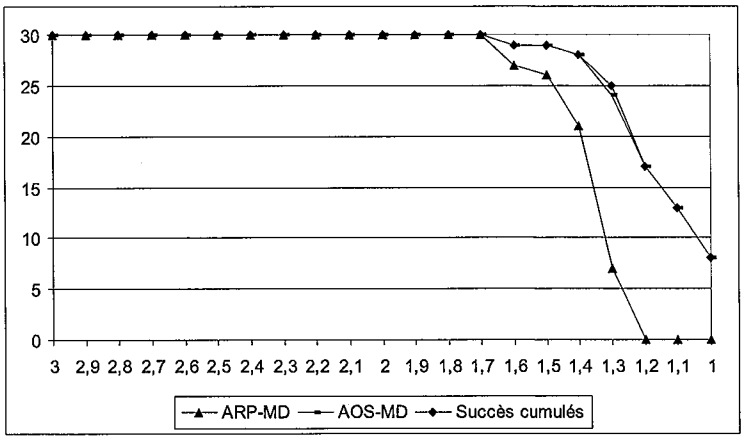


FIG. 2.6 – Nombre de solutions valides obtenues m=10, n=20, $\varepsilon = 5.36$

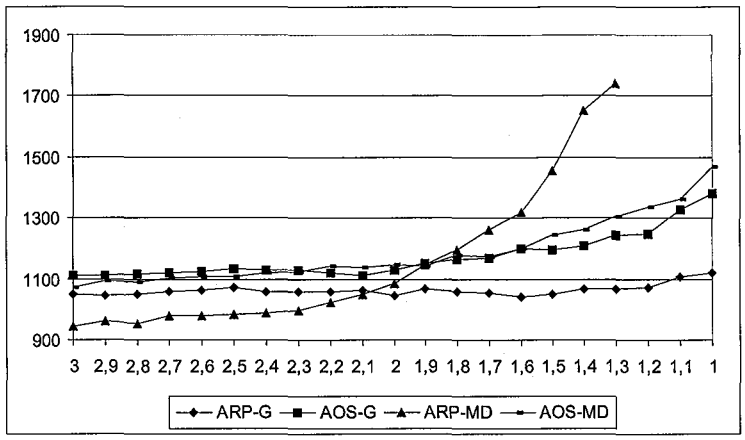


FIG. 2.7 – Makespan moyen m=10, n=20, $\varepsilon = 5.36$ (solutions valides uniquement)

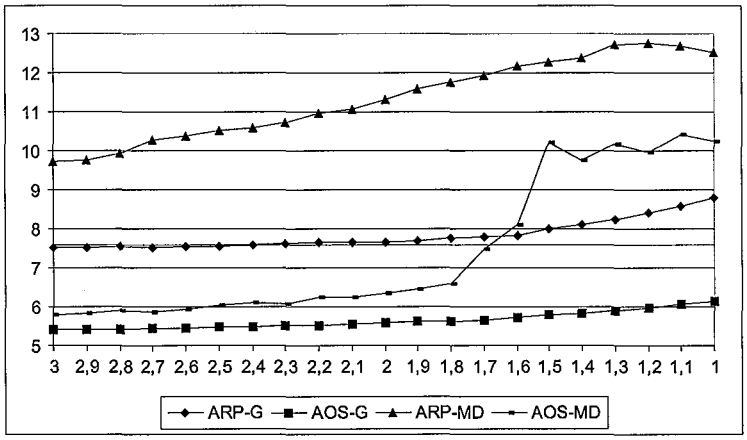


FIG. 2.8 – Nombre moyen de calculs de dates d'exécution m=10, n=20, $\varepsilon = 13.1$

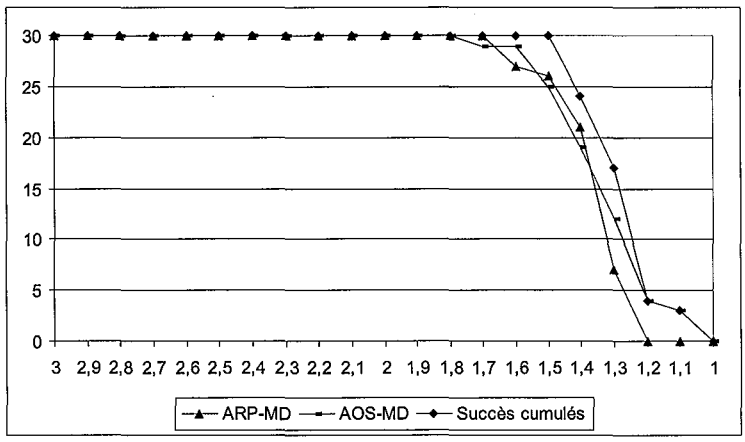


FIG. 2.9 – Nombre de solutions valides obtenues m=10, n=20, $\varepsilon = 13.1$

La courbe du nombre moyen de calculs de dates d'exécution pour les algorithmes ARP-MD et AOS-MD croît de manière significative jusqu'à culminer à un maximum avant de diminuer. Cette diminution est occasionnée par l'horizon de planification. La tension sur les écarts maximaux devenant trop forte, ARP-MD et AOS-MD ne parviennent plus à trouver de solutions valides. A force de dépiler et décaler les dates de disponibilité des opérations, l'horizon de fin de planification H est atteint. Plus la tension sur les écarts maximaux augmente et plus le décalage des dates de disponibilité $s_{O'} + d(O', O'')$ devient important. L'horizon de fin de planification est atteint plus vite d'où une baisse du nombre moyen de calculs qui ne traduit que l'impossibilité de construire une solution valide avant la fin de l'horizon de planification.

La figure 2.6 présente, pour chaque valeur de C, le nombre total de solutions valides obtenues sur les 30 instances du jeu d'essai. Ce nombre chute de manière significative pour les fortes tensions. En particulier, ARP-MD n'est pas parvenu à construire une seule solution valide pour des tensions $C \leq 1.2$. Les versions par grappe ont permis d'obtenir systématiquement une solution valide.

En comparant les versions par grappes et les versions modifiées, on s'aperçoit que AOS-MD parvient à se maintenir à un niveau très bas par rapport à ARP-G pour les faibles tensions, le nombre de calculs augmentant de manière significative dès que la tension augmente.

Les constatations précédentes nécessitent d'être modulées. Les performances des algorithmes ARP-MD et AOS-MD sont très dépendantes du choix des règles de priorité et du tri de la liste d'ordre strict. Lorsque la liste de priorité de AOS-MD est trop entrelacée, le nombre de calculs explose (figure 2.8 avec une mesure d'entrelacement de $\varepsilon = 13.1$) et l'algorithme ne parvient plus à construire de solutions valides pour les fortes tensions (figure 2.9).

Pour les algorithmes par grappes ARP-G et AOS-G, l'influence des règles de priorité ou l'ordre de tri des grappes n'a que peu d'influence sur le nombre moyen de calculs de date d'exécution des opérations. Lorsque le nombre de travaux ou de ressources diminue, les courbes présentent la même allure mais le pic est atténué ou absent. Le nombre d'opérations en conflit étant diminué, les algorithmes modifiés parviennent à trouver une solution, même dégradée, pour les fortes tensions.

2.4.2.2 Makespan moyen

L'évolution du makespan moyen (figure 2.5) traduit le même phénomène. Si le makespan suit une progression normale avec les algorithmes par grappes, celui-ci augmente de manière significative avec les algorithmes par liste ou d'ordre strict modifiés jusqu'à culminer un peu au-delà de l'horizon de planification $2.n.p_{max}$ unités de temps que nous

avions fixé. On constate que sur les familles de jeu d'essai considérées, les algorithmes par grappes obtiennent de meilleures performances en terme de makespan moyen que les algorithmes d'ordre strict ou de liste modifiés pour les fortes tensions. Pour les faibles tensions au contraire, les versions modifiées parviennent à faire mieux que leurs homologues par grappes.

2.4.3 Décalage

Jusqu'à présent, en cas de dépilement lorsqu'une contrainte d'écart maximal $d(O_i, O_j) < 0$ n'était pas satisfaite, nous avons toujours incrémenté la date de disponibilité r_j de O_j en lui affectant la nouvelle valeur $r_j = s_i + d(O_i, O_j)$. Nous avons démontré que ce décalage était suffisant et permettait de construire sous certaines conditions des algorithmes actifs (théorème 2.1). Dans le cas général, on peut se poser la question de savoir si d'autres décalages ne seraient pas plus intéressants. En effet, quel serait l'impact sur la qualité de la solution si on incrémentait simplement r_j d'une unité de temps ou de p_{min}. Incotec a été confronté à la même problématique et a opté pour un décalage d'une unité de temps. Ce décalage est cependant trop consommateur de CPU. Le nombre de calculs augmente de manière drastique au point qu'il est parfois impossible d'obtenir une solution en temps raisonnable. Pour contourner cet inconvénient, Incotec a opté pour un pas paramétrable à la discrétion de l'utilisateur.

Nous avons effectué des tests sur la famille de jeux d'essai 10x20. La figure 2.10 présente l'évolution moyenne du makespan pour l'algorithme ARP-MD avec le décalage classique (courbe ARP-MD), un décalage d'une unité de temps (courbe ARP-MD') et un décalage correspondant à p_{min} unités de temps (courbe ARP-MD"). Outre le fait que le nombre de calculs est astronomique pour les décalages d'une unité de temps (3370134 contre 38178 pour ARP-MD et 269174 pour ARP-MD" pour la plus faible des tensions), on constate que la qualité de la solution est moins bonne pour des décalages d'une unité de temps et de p_{min} unités de temps. Des résultats identiques ont été constatés pour l'algorithme AOS-MD (figure 2.11).

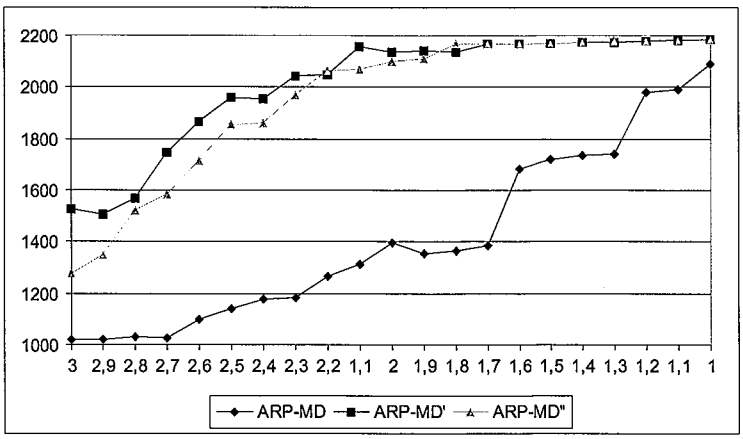


FIG. 2.10 – ARP-MD : Evolution du makespan moyen pour différents décalages m=10, n=20

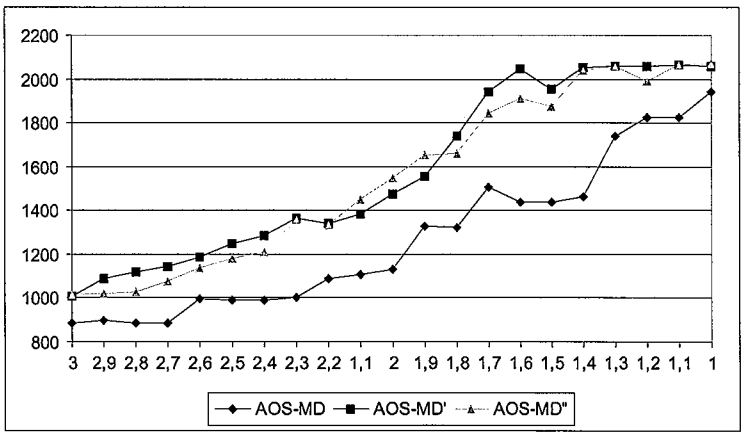


FIG. 2.11 – AOS-MD : Evolution du makespan moyen pour différents décalages m=10, n=20, $\varepsilon = 3.9$

2.5 PharmaCorp

Nous terminons par des expériences menées à partir de la version simplifiée du processus de Lyophilisation de la société PharmaCorp, gamme de la figure A.1.1.

2.5.1 Ressources

La figure A.1.2 permet de visualiser les affectations des ressources (cadres en pointillés) à chaque opération. Les opérations utilisent de 1 à 3 ressources à capacités limitées. Le tableau 2.2 résume les différentes ressources considérées, leur nature, les quantités

disponibles et les calendriers. Le calendrier 3x8 est un calendrier permanent, le calendrier 2x8 permet de travailler du lundi au vendredi de 6h à 21h.

Nous avons modélisé les deux lignes de lyophilisation «classiques». A l'issue des opérations de montage et de filtration, l'une ou l'autre ligne est choisie. Ces deux lignes ont des caractéristiques identiques mais comportent des ressources différentes (lyophilisateurs 1 et 2 ; chambres stériles R08 et R18). Le choix de la ligne est effectué lors du placement de l'opération de remplissage. L'algorithme effectue le calcul des dates d'exécution des deux opérations possibles et conserve la plus avantageuse, celle dont la date de fin d'exécution est la plus petite, ce qui conditionne le choix de la ligne. Ce choix est évidement remis en cause en cas de dépilement.

2.5.2 Résultats

Les tests ont été effectués sur un jeu d'essai comportant 12 travaux, 3 courts, 3 moyens et 6 longs, soit 216 opérations au total. Les travaux se différencient par les temps d'exécution de l'opération de lyophilisation. Les lots courts, moyens et longs ont des temps opératoires respectifs de 35, 45 et 75 heures. Ce découpage en lots de différentes tailles est représentatif de la production fournie par le planning type de la société PharmaCorp.

Dans un premier temps, nous avons exécuté l'algorithme ARP-MD en utilisant la règle de priorité de la plus petite marge libre par rapport aux écarts maximaux (voir section 2.4.1.4 de ce chapitre). Nous avons obtenu un makespan de 741 pour un total de 60094 calculs de dates d'exécution.

Dans un second temps, nous avons voulu tester la stabilité des algorithmes vis à vis du phénomène du «saute-mouton» en modifiant aléatoirement les priorités mais tout en conservant les lots courts moyens et longs. Pour ce faire, nous avons affecté à chaque opération une priorité fixe choisie aléatoirement entre 1 et 10. L'algorithme ARP-MD a été appliqué en considérant cette priorité statique, ARP-G a été appliqué en considérant la priorité d'une grappe comme étant égale à la plus grande priorité des opérations la constituant. Les algorithmes AOS-MD et AOS-G ont utilisé une liste d'ordre stricte entrelacée comme pour les jeux d'essai avec gammes linéaires (paragraphe 2.4.1.5 et 2.4.1.6).

Nous avons généré ainsi 50 jeux d'essais. Les résultats apparaissent dans le tableau 2.3. On constate que les algorithmes par grappes obtiennent des résultats de meilleure qualité que les versions modifiées qui pâtissent du phénomène de «saute-mouton». Les figures 2.12 et 2.13 présentent les ordonnancements obtenus respectivement par ARP-G et ARP-MD. Le Gantt de la figure 2.13 est plus étalé, présente de nombreux temps morts par rapport à celui de la figure 2.12.

Ressource	Type	Quantité	Calendrier
Laborantin E1	Compétence	6	2x8
Laborantin E2	Compétence	2	2x8
Laborantin E3	Compétence	2	2x8
Mireur	Compétence	4	2x8
Conducteur Lyo	Compétence	2	3x8
Stérilisateur	Machine	1	-
Autoclave	Machine	2	-
Machine pour Peser	Machine	1	-
Machine pour formuler	Machine	1	-
Filtre	Machine	2	-
MAR	Machine	1	-
R06	chambre stérile	1	-
R08	chambre stérile	1	-
R18	chambre stérile	1	-
R09	chambre stérile	1	-
R222	chambre stérile	1	-
R232	salle	1	-
Lyophilisateur 1	Machine	1	-
Lyophilisateur 2	Machine	1	-

TAB. 2.2 – Ressources du processus de lyophilisation

Algorithme	C_{max}		Nombre de Calculs		ε
	moyenne	écart type	moyenne	écart type	
ARP-G-RP	659	46	6897	877	-
ARP-G-OS	670	43	6977	671	-
AOS-G-RP	809	54	472	34	-
AOS-G-OS	833	51	498	18	-
ARP-MD	1045	82	121211	11218	-
AOS-MD	1046	163	842	171	3.4

TAB. 2.3 – Résultats numériques PharmaCorp

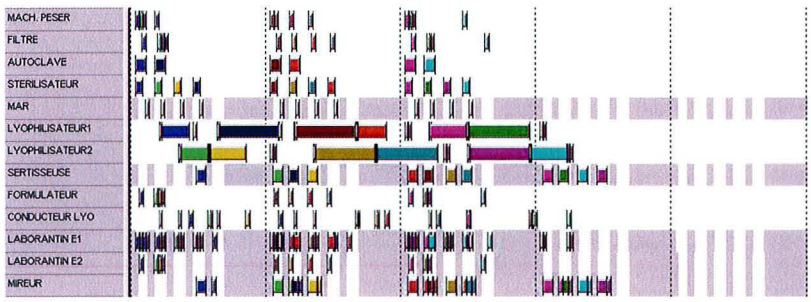


FIG. 2.12 – Gantt jeu d'essai PharmaCorp obtenu avec ARP-G

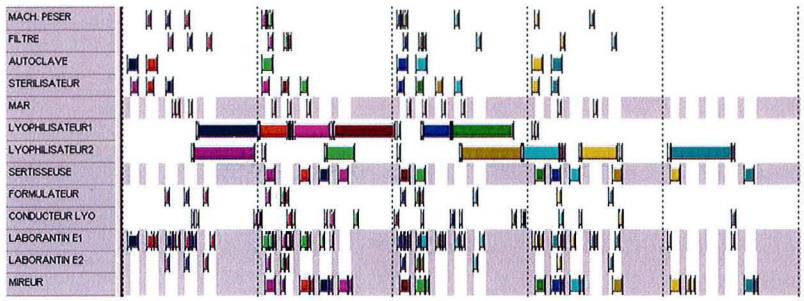


FIG. 2.13 – Gantt jeu d'essai PharmaCorp obtenu avec ARP-MD

2.6 Conclusion

L'algorithme de placement par grappes est une généralisation des algorithmes à base de règle de priorité ou d'ordre strict. Cela nous semble être une bonne approche pour construire une solution valide. Elle ne nécessite que des modifications mineures d'un algorithme de liste ou d'ordre strict déjà existant. La construction d'une solution valide est très rapide. De plus, même si la terminaison de l'algorithme est assurée de manière artificielle par un horizon de planification, l'algorithme parviendra à construire une solution valide dans la majorité des cas industriels. En effet, les cas de figure que nous avons été amenés à étudier nous ont prouvé que les grappes étaient réduites essentiellement à quelques opérations. A l'intérieur d'une grappe, les phénomènes de «saute-mouton» apparaissent principalement lorsque plusieurs opérations se partagent la même ressource. Dans ce cas, un bureau des méthodes en amont minimise déjà considérablement les conflits en imposant des ordres de succession sur ces opérations. Dans ce cas, les algorithmes par grappes permettent également de construire des ordonnancements actifs pour des problèmes sans temps mort autorisé sur les ressources. A moins de considérer des jeux d'essai «académiques», la décomposition par grappes permet donc de réduire considérablement la combinatoire. C'est une approche simple, efficace et pragmatique.

Il faut tout de même apporter plus de souplesse dans la définition des grappes. Si pour les fortes tensions sur les écarts maximaux il n'y a pas le choix et il faut placer consécutivement les opérations d'une grappe, les tests mettent en évidence de meilleures performances des versions modifiées de l'algorithme de liste ou d'ordre strict pour les faibles tensions sur les écarts maximaux. Une amélioration possible consisterait certainement à n'intégrer dans une grappe que les opérations liées par des écarts maximaux «forts», du type sans attente, ce qui permettrait de couper les grappes en grappes plus petites, plus facilement plaçables, et autoriserait un entrelacement des grappes réduites plus importants augmentant les performances de l'algorithme en terme de makespan.

3

Problèmes d'atelier de type flowshop

Dans ce chapitre, nous nous intéressons aux problèmes d'atelier de type flowshop avec contraintes d'écart minimal et maximal entre les opérations d'un même travail. Notre objectif est de minimiser le C_{max}. Pour ce faire, nous utilisons les algorithmes génétiques. Après avoir énoncé quelques résultats sur la dominance des flowshop de permutation, nous procédons à un rappel sur le fonctionnement des algorithmes génétiques. Pour rendre ces derniers plus efficaces, nous effectuons une analyse comparative de la qualité de plusieurs opérateurs de croisement. Nous constatons qu'aucun des opérateurs génétiques utilisés n'est efficace quel que soit le type d'instance de problème considéré. Nous proposons une méthode d'autodétermination qui sélectionne automatiquement l'opérateur de croisement le mieux adapté au problème considéré et à l'état courant de la population.

3.1 Introduction

Les problèmes d'atelier de type flowshop revêtent une importance particulière en ordonnancement. Le flowshop et ses variantes (flowshop hybride..) sont le reflet d'un nombre important d'organisations d'ateliers que l'on rencontre dans l'industrie. Dans ce type d'organisation, les travaux sont indépendants les uns des autres et l'ordre de passage des tâches d'un même travail sur les machines est identique pour tous les travaux. Un nombre important d'études a été consacré à ces problèmes et il est impossible de les présenter de manière exhaustive. Pour une description de la problématique et des algorithmes existants, on pourra se reporter aux livres de Lopez et Roubellat [LR01] (chapitre 2, «concepts et méthodes de base») ou Blazewicz et al [BEP+96].

Dans le chapitre précédent, nous avons étudié un algorithme de construction par grappes, polynomial sous certaines hypothèses. Les problèmes de type flowshop se prêtent naturellement à cette approche puisque les grappes sont au plus égales aux travaux. En présence de calendriers permanents, nous satisfaisons les conditions de terminaison en temps polynomial et pouvons affirmer que la construction d'une solution valide, en se restreignant aux flowshops de permutation, se fera en $O(n.m^2)$. Cependant, si les contraintes sont définies exclusivement entre opérations successives d'un même travail, on peut trouver des algorithmes de construction d'une solution valide en $O(n.m)$ comme celui proposé par Fondrevelle dans sa PSE [Fon03].

Notre souci n'est plus tant la construction d'une solution valide que la recherche d'une solution optimale en terme de C_{max}. Sans contraintes temporelles entre opérations, la recherche d'une solution optimale en terme de C_{max} est polynomiale pour 2 ressources mais NP-difficile pour 3 ressources et plus [Kan76].

En ce qui concerne les problèmes de flowshop avec contraintes d'écart minimal et maximal, peu de travaux ont été consacrés à cette problématique. La majorité des publications concerne soit des problèmes avec temps minimaux [Mit58] [Szw83] [Del96], soit des problèmes sans attente car ces derniers sont équivalents à des problèmes du voyageur de commerce asymétriques [Wie72] et bénéficient des méthodes développées pour ces derniers. Gilmore et Gomory [GG64] ont ainsi démontré que le problème de flowshop sans attente à 2 machines était polynomial. Pour plus de 3 machines, la recherche de l'optimum dans le cas du flowshop sans attente devient NP-difficile au sens fort [SC79]. Récemment, Fondrevelle [Fon03] a présenté une méthode de résolution exacte pour les problèmes de flowshop avec écarts maximaux entre opérations. Sa méthode, basée sur une Procédure par Séparation et Evaluation, se limite cependant aux problèmes de flowshop de permutation et au cas où les contraintes sont définies exclusivement entre opérations successives d'un même travail.

La complexité de la recherche de l'optimum nécessite l'utilisation de méthodes approchées pour pouvoir espérer trouver en temps raisonnable une bonne solution en terme de C_{max}

pour des problèmes de taille industrielle. Dans cette partie, nous présentons une approche par algorithmes génétiques. Pourquoi un tel choix ? Les algorithmes génétiques sont assez souples à mettre en oeuvre et permettent de s'adapter très facilement à tout type d'objectif. Ils ne sont en effet pas spécifiques à une problématique donnée, contrairement aux méthodes par séparation et évaluation qui nécessitent des estimateurs précis et adaptés. Cependant, pour un gain de performance, les opérateurs de croisement doivent être choisis avec soin. Nous étudions dans la présente partie plusieurs types d'opérateurs et analysons leurs qualités intrinsèques.

3.2 Dominance des flowshops de permutation

Beaucoup d'études ont porté sur une sous-classe des flowshops, celle des flowshops de permutation. Dans cette sous-classe, l'ordre de passage des travaux est identique sur toutes les ressources. Cette restriction permet de simplifier non seulement la difficulté du problème mais également l'organisation au niveau de l'atelier. Il convient de se poser la question de la dominance de cet ensemble et de mesurer l'écart maximum à l'optimum global.

Notons C^*_{max} la meilleure durée totale pour l'ensemble des ordonnancements et C^0_{max} la meilleure durée totale pour les ordonnancements de permutation. En l'absence d'écarts maximaux, Potts et al [PSW91] ont exhibé une famille de problèmes de type flowshop tel que

$$\frac{C^*_{max}}{C^0_{max}} \geq \frac{1}{2}(\sqrt{2n} + \frac{1}{2})$$

Cette famille de problèmes est cependant trop académique. En effet, le nombre des ressources m est une variable égale au double de celle des travaux n. De plus, chaque travail ne comporte que deux opérations de durée non nulle. En revanche, Portmann et al [PAK02], ont démontré que sous des hypothèses plus réalistes :

- le nombre m de ressources est une constante ;
- les durées d'exécution des tâches sont bornées par p_{min} et p_{max} ;
- $p_{min} > 0$ ou il existe au moins une ressource dont la charge tend vers l'infini lorsque le nombre des travaux n tend vers l'infini ;

le rapport C^*_{max}/C^0_{max} tend vers 1 lorsque n tend vers l'infini.

Qu'en est-il en présence d'écarts minimaux et maximaux ? Pour le critère du makespan, il est bien connu qu'en l'absence de temps d'attente minimaux et maximaux, l'ensemble des flowshop de permutation est dominant lorsque $m = 2, 3$. De même, dans le cas de m machines, l'ordre de passage identique des travaux sur les deux premières machines d'une part et sur les deux dernières machines d'autre part définit un ensemble dominant. Ceci n'est plus le cas à partir de $m = 2$ en présence de contraintes d'écart maximal et minimal.

Exemple : soient 2 ressources, 2 travaux tels que
- $p_{i,j} = 1 \ \forall i = 1,2 \, ; \forall j = 1,2$
- $s_{1,1} + 3 \leq s_{2,1}$

Pour tout flowshop de permutation, le makespan sera de 5. Sur la figure 3.1, un makespan de 4 est atteint en permutant $O_{2,1}$ et $O_{2,2}$ sur M_2.

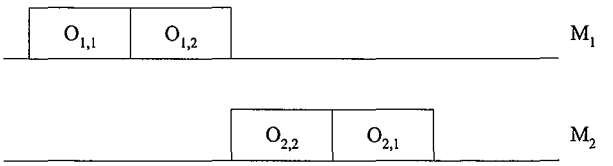


FIG. 3.1 – Ordonnancement optimal

Voici un autre contre-exemple avec écart maximal. Soient 4 travaux avec des temps opératoires figurant dans le tableau 3.1. On impose les inégalités de potentiels $s_{2,3} - 2 \leq s_{1,3}$ et $s_{2,4} - 2 \leq s_{1,4}$ (les opérations $O_{1,3}$ et $O_{2,3}$ ainsi que les opérations $O_{1,4}$ et $O_{2,4}$ doivent se succéder sans temps mort, contrainte symbolisée par un trait gras vertical sur les figures 3.2 et 3.3). Un ordonnancement optimal est donné sur la figure 3.2. Le lecteur pourra se convaincre aisément (en listant par exemple l'ensemble des ordonnancements de permutation) que tout flowshop de permutation conduit à une solution de moins bonne qualité (figure 3.3).

opération	$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{1,4}$	$O_{2,1}$	$O_{2,2}$	$O_{2,3}$	$O_{2,4}$
durée	1	2	2	2	4	1	1	1

TAB. 3.1 – Temps opératoires

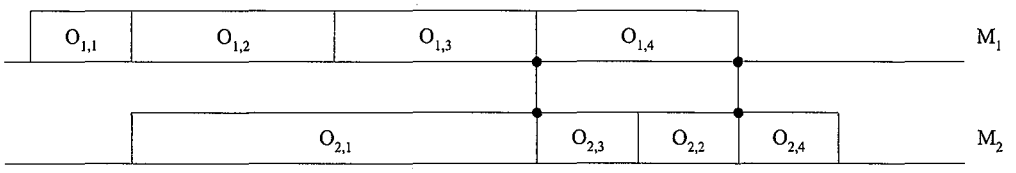


FIG. 3.2 – Ordonnancement optimal pour l'ensemble des flowshops

Ces deux contre-exemples incitent donc à considérer l'ensemble des flowshops et non pas uniquement ceux de permutation. C'est cependant faire fi de nombreux cas particuliers. En effet, les solutions valides pour les problèmes de flowshop sans attente sont exclusivement des flowshops de permutation. De même, s'il existe entre tout couple d'opérations successives $O_{i,j}$ et $O_{i+1,j}$ au sein d'un même travail, une contrainte d'écart maximal

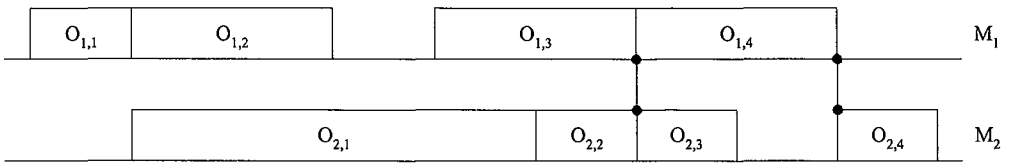


FIG. 3.3 – Ordonnancement optimal sur l'ensemble des flowshop de permutation

$d(O_{i+1,j}, O_{i,j})$ compris entre $-3p_{min}$ et 0, les ordonnancements de permutation sont les seuls valides.

Démonstration
Supposons l'ordre de succession des tâches j et $j+1$ inversé sur la ressource $i+1$, c'est-à-dire, sans perte de généralité, $O_{i,j}$ précède $O_{i,j+1}$ sur i mais $O_{i+1,j+1}$ précède $O_{i+1,j}$ sur $i+1$. Pour que la solution soit valide, nous devons avoir l'inégalité :

$$s_{i,j} + p_{i,j} + p_{i,j+1} + p_{i+1,j+1} \leq s_{i+1,j} \leq s_{i,j} - d(O_{i+1,j}, O_{i,j})$$

Autrement dit, nous devons avoir

$$d(O_{i+1,j}, O_{i,j}) \leq -(p_{i,j} + p_{i,j+1} + p_{i+1,j+1})$$

Si pour tout i, j nous avons $0 > d(O_{i+1,j}, O_{i,j}) > -3p_{min}$, les flowshops de permutation sont les seuls valides. □

Le résultat précédent peut encore être affiné mais il fait apparaître que les faibles écarts maximaux favorisent les flowshops de permutation. Ceux-ci ont en effet tendance à minimiser les temps d'attente entre les opérations et donc à minimiser les possibilités de permutation de l'ordre de passage des travaux d'une ressource à l'autre. Qu'en est-il des contraintes d'écart minimal ? Celles-ci au contraire imposent plus d'attente entre les opérations et risquent d'accroître les possibilités de permutation.

Considérons l'exemple suivant :
- n tâches, 2 ressources
- $p_{i,j} = 1 \ \forall i = 1, .., n, \ \forall j = 1, 2$
- $0 < d(O_{1,n}, O_{2,n}) = D \leq n$

Le meilleur C_{max} est obtenu pour l'ordre de passage $O_{1,n}, O_{1,1}, O_{1,2}, .., O_{1,n-1}$ des opérations sur la première ressource et $O_{2,1}, O_{2,2}, .., O_{2,n-1}, O_{2,n}$ sur la deuxième ressource (figure 3.4). Dans ce cas $C_{max} = n + 2$.

Pour tout ordonnancement de permutation, notons k la position de $O_{1,n}$ sur la première ressource. Sans perte de généralité, on peut considérer que l'ordre de passage des opéra-

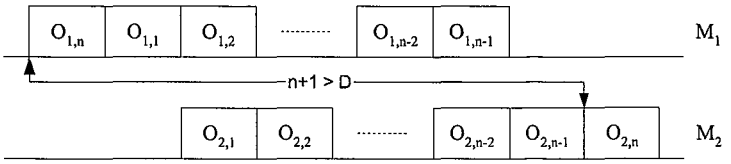


FIG. 3.4 – Solution minimisant le C_{max}

tions sur la première ressource est

$$O_{1,1}, O_{1,2}, .., O_{1,k-1}, O_{1,n}, O_{1,k}, O_{1,k+1}, .., O_{1,n-1}$$

Dans ce cas, $C^*_{max} = n + D$ (figure 3.5). En prenant $D = n$, le rapport $C_{max}/C^*_{max} = 1/2$ et ne tend pas vers 1 lorsque n tend vers l'infini.

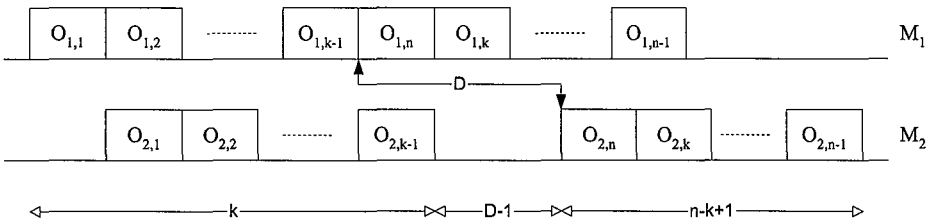


FIG. 3.5 – Solution avec flowshop de permutation

Cependant, dans l'exemple précédent l'écart minimal $d(O_{1,n}, O_{2,n})$ n'est pas borné. Il semble raisonnable de considérer que, tout comme les temps opératoires bornés par p_{min} et p_{max}, les écarts minimaux le sont également par d_{max}. Dans ce cas le rapport C^*_{max}/C^0_{max} tend vers 1 lorsque n tend vers l'infini.

Démonstration :
Soit P un problème de flowshop. Soit $\tilde{C}^*_{max}$ la meilleure durée totale pour l'ensemble des ordonnancements sans tenir compte des écarts minimaux. Soit $\tilde{C}^0_{max}$ la meilleure durée totale des ordonnancements de permutation sans tenir compte des écarts minimaux.
On a l'inégalité suivante :

$$\tilde{C}^*_{max} \leq C^*_{max} \leq C^0_{max} \leq \tilde{C}^0_{max} + (m-1).d_{max}$$

La dernière inégalité est obtenue de la manière suivante :
On considère l'ordonnancement de permutation donnant le meilleur $\tilde{C}^0_{max}$. Celui-ci ne respecte pas forcément les temps d'attente minimaux. Sans perte de généralité, on peut considérer que l'ordre de passage des tâches sur la première ressource est $j = 1, .., n$. En posant $S'(O_{i,j}) = S(O_{i,j}) + (i-1)d_{max}$ ($S(O_{i,j})$ est la date de début de l'opération

$O_{i,j}$) pour $i > 1$, on obtient une nouvelle solution valide respectant les contraintes d'écart minimal. Son makespan est $\tilde{C}^0_{max} + (m-1).d_{max}$.

$$1 \leq C^*_{max}/C^0_{max} \leq (\tilde{C}^0_{max} + (m-1).d_{max})/\tilde{C}^*_{max}$$

Or lorsque n tend vers l'infini, $(\tilde{C}^0_{max} + (m-1).d_{max})/\tilde{C}^*_{max} \sim \tilde{C}^0_{max}/\tilde{C}^*_{max}$ et comme $\tilde{C}^0_{max}/\tilde{C}^*_{max}$ tend vers 1 lorsque n tend vers l'infini, il en est de même pour le rapport C^*_{max}/C^0_{max}. □

Grâce aux résultats précédents, nous pouvons conclure que les flowshops de permutation restent une bonne approximation et un bon ensemble de recherche de l'optimum en terme de C_{max}. En outre, il nous semble raisonnable de considérer des flowshops de permutation afin de ne pas ajouter encore des difficultés organisationnelles aux difficultés déjà inhérentes aux problématiques avec contraintes temporelles.

3.3 Algorithmes génétiques

3.3.1 Principes généraux

Les algorithmes génétiques constituent des méthodes d'optimisation approchées nées de l'observation du monde du vivant. Ils tirent leur nom de l'analogie entre la représentation d'une solution sous forme de chaîne de caractères '0' ou '1', et la structure génétique d'un chromosome. Ils sont basés sur le processus d'évolution génétique des organismes biologiques à travers des générations selon la théorie de l'évolution de Darwin.
Une population d'individus représentant des solutions au problème évolue au gré des croisements, des mutations et des sélections naturelles. Ces individus vivent dans le même milieu, se disputent les ressources naturelles et doivent faire face à des prédateurs. Les meilleurs individus, les mieux adaptés à leur environnement, les plus «forts», seront favorisés et auront les plus grandes probabilités de survivre et de s'accoupler. Les plus faibles et les moins adaptés seront éliminés. Au fil des générations, les meilleurs gènes, ou combinaisons de gènes, vont donc se propager en s'échangeant les meilleurs caractéristiques. La population va évoluer vers une population de «bons» individus qui constitueront autant de solutions «optimisées» du problème.
En soi, ce genre de méthode ne comporte aucune garantie de trouver la solution optimale. Tout repose sur le brassage efficace des gènes et la qualité de l'opérateur de croisement censé transmettre à l'enfant les caractéristiques les plus intéressantes des parents pour produire des individus de mieux en mieux adaptés.

Ce type d'algorithme a donné naissance à une littérature riche et variée qu'il est impossible de présenter de manière exhaustive. Holland [Hol75], Goldberg [Gol89] et Davis [Dav85] sont les principaux chercheurs qui ont les premiers adaptés les algorithmes génétiques à

la recherche de solutions de problèmes d'optimisation. On notera également les travaux de Reeves [Ree95] pour une application au problème de flowshop.

L'algorithme 3.1 est un exemple d'algorithme génétique classique.

Algorithme 3.1 Algorithme génétique «classique»

1. Générer la population initiale P_0 de Q individus
2. Evaluer la "force" des individus de la population P_0
3. Tant que la condition d'arrêt n'est pas satisfaite
   (a) Sélectionner $Q/2$ couples d'individus dans la population P_{k-1}
   (b) Pour chaque couple :
      i. Appliquer l'opérateur de croisement afin d'obtenir les enfants
      ii. Avec une probabilité p_{mut}, appliquer l'opérateur de mutation à chaque enfant
      iii. Evaluer la "force" des enfants
      iv. Ajouter les nouveaux individus à la population P_{k-1}
   (c) Epurer la population pour supprimer les doublons et ne conserver que les Q "meilleurs" individus qui constituent la nouvelle population P_k

Il existe cependant de nombreuses variantes d'algorithmes génétiques selon le type de codage des individus, le mode de sélection des couples, le mode d'épuration ou de renouvellement de la population etc.. Certains algorithmes se différencient également par l'opérateur de mutation. Dans le cas de l'algorithme considéré ici, l'opérateur de mutation est appliqué aux enfants avec une certaine probabilité. Chaque individu n'a donc qu'une seule fois l'occasion de subir une mutation, au moment de sa conception. Dans d'autres types d'approches, lors de chaque cycle, un individu peut subir une mutation avec une probabilité donnée.

3.3.2 Codage et évaluation des individus

Les résultats de dominance des flowshops de permutation, même s'ils ne sont corrects que pour des cas particuliers ou de manière asymptotique, nous encouragent à adopter des approches qui construisent presque toujours des ordonnancements de permutation, mais saisissent également les opportunités qui se présentent en comblant des trous devant des opérations déjà placées.

Tout ordonnancement d'un flowshop de permutation est entièrement défini par l'ordre de passage des n opérations $O_{1,j}$ sur la première ressource. Sa définition nécessite donc de coder des permutations. Parmi les codages de permutation (voir l'article de Portmann

[Por96]), nous avons choisi le codage des individus sous forme de vecteur de permutation qui contient la liste des numéros des travaux dans l'ordre de leur passage sur la première machine (et sur les machines suivantes dans le cas du flowshop de permutation classique). Par exemple, dans le cas de 6 travaux, le vecteur $(5, 1, 4, 6, 3, 2)$ représente la solution unique où l'ordre des travaux sur la première ressource est 5, 1, 4, 6, 3 puis 2.

Nous utilisons un codage de permutation qui représente non pas l'ordre de passage sur les ressources, mais l'ordre de placement des travaux en tant que grappes. Ainsi, le vecteur $(5, 1, 4, 6, 3, 2)$ représente la solution unique obtenue en plaçant les opérations du travail 5 puis du travail 1.., le placement des opérations de chaque travail s'effectuant en utilisant l'algorithme AOS-MD ou ARP-MD définis au chapitre précédent. La construction de la solution représentée par le vecteur $(5, 1, 4, 6, 3, 2)$ est donc obtenue en appliquant l'algorithme AOS-MD avec la liste d'ordre strict :

$$L = \{\underbrace{O_{1,5}, O_{2,5}, .., O_{m,5}}_{\text{travail 5}}, \underbrace{O_{1,1}, O_{2,1}, .., O_{m,1}}_{\text{travail 1}}, ..., \underbrace{O_{1,2}, O_{2,2}, .., O_{m,2}}_{\text{travail 2}}\}$$

Le résultat obtenu n'est donc pas forcément un flowshop de permutation dans la mesure où le placement des opérations d'un travail génère des trous qui seront exploités par les placements des opérations des travaux ultérieurs.
Pour forcer les algorithmes à construire des flowshops de permutation, il suffit de calculer la date de disponibilité de chaque opération en tenant compte de la plus grande date de fin d'exécution des opérations déjà placées sur la ressource. Nous obtenons ainsi deux variantes de nos algorithmes de résolution, l'une où on accepte les flowshops de non permutation et l'autre où on n'accepte que la construction de flowshops de permutation.

L'évaluation d'un individu, encore appelée force ou fitness, est le C_{max} de la solution obtenue. Plus le makespan est petit et meilleure sera la solution.

3.3.3 Opérateurs de croisement

Dans les versions non hybridées des algorithmes génétiques, le choix d'un opérateur de croisement conditionne fortement la qualité de l'algorithme génétique. Celui-ci joue un rôle fondamental alors que l'opérateur de mutation n'a qu'un rôle secondaire, celui d'assurer la diversité de la population et d'éviter une convergence trop rapide.

Plusieurs critères sont à prendre en compte dans le choix de l'opérateur de croisement.

Convergence prématurée

L'opérateur doit être choisi de sorte à converger assez rapidement vers une population de bons individus, tout en évitant une convergence prématurée vers une population «stable» qui n'évoluera plus et aura tendance à s'auto-reproduire. De ce fait, un opérateur qui aura

tendance à cloner les parents est à éviter. Des mécanismes de contrôle de la diversité de la population peuvent contribuer à empêcher une convergence prématurée.

Nature du problème

Comme nous le verrons plus tard sur les résultats numériques, il n'existe pas d'opérateur de croisement expérimentalement dominant. De la nature du problème dépendra le choix de tel ou tel opérateur. Certains opérateurs se montrent mieux adaptés aux problèmes sans attente mais obtiendront de piètres résultats pour les problèmes sans écarts maximaux, et réciproquement.

Taux d'amélioration

Pour être «bon», un opérateur de croisement doit présenter une forte corrélation entre la qualité des parents et celle des enfants. Si les parents sont «bons», on s'attend implicitement à ce que les enfants le soient également. Il doit en outre permettre d'améliorer globalement la population et les enfants doivent donc présenter des caractéristiques plus intéressantes que leurs parents. Cependant, le fait d'engendrer de bons individus n'est pas à lui seul un critère suffisant. Certains opérateurs peuvent être excellents en début d'évolution mais se dégrader rapidement au cours des cycles lorsque la population aura déjà convergé.

Un de nos objectifs est de mesurer la qualité des différents opérateurs de croisement et d'évaluer le comportement de ces opérateurs tout au long de l'évolution afin de déterminer s'ils conservent leurs qualités au cours des cycles. Au fur et à mesure de l'évolution, la population va converger et il deviendra difficile d'améliorer encore les individus. Les opérateurs subiront donc forcément une perte de performance. Il convient de mesurer pour chaque opérateur si une dégradation «anormale» peut se faire ressentir.

Il existe une pléthore d'opérateurs de croisement. Nous en avons sélectionné principalement 4 que nous avons décliné sous plusieurs variantes. Ces opérateurs ont retenu notre attention soit parce qu'ils sont énormément utilisés, soit parce que leur mode opératoire me semblait particulièrement indiqué dans le cadre des contraintes d'écart minimal et maximal.

3.3.3.1 Opérateur 1X

Il s'agit de l'opérateur le plus simple et le plus classique à 1 point de coupure appliqué aux problèmes d'ordonnancement par Davis en 1985 [Dav85]. Son principe repose sur l'opérateur de croisement naturel consistant à «échanger» des portions de chromosomes entre les parents pour produire les enfants.

Exemple. Considérons 9 travaux et les deux parents P1 et P2 ci-dessous :

P1 : 2 1 4 5 3 8 6 7 9
P2 : 6 3 8 4 9 5 7 2 1

Pour croiser P1 et P2, on choisit aléatoirement un point de coupure entre 1 et 9, mettons 5. On ne peut cependant pas naïvement intervertir simplement les 5 premiers gènes de P1 et les 5 derniers gènes de P2 comme dans un croisement génétique classique. Ceci est valide dans le cas d'un codage sous forme de chaînes de caractères 0 et 1 mais pas dans notre cas. Ceci conduirait aux deux enfants E1 et E2 suivants :

E1 : 2 1 4 5 3 5 7 2 1
E2 : 6 3 8 4 9 8 6 7 9

Les deux enfants E1 et E2 ne représentent plus une solution valide. Il existe en effet des doublons et certains travaux ont disparu. Pour y remédier, les 5 premiers gènes de P1 sont conservés et on complète avec les gènes restants mais pris dans l'ordre de P2 :

E1 : 2 1 4 5 3 6 8 9 7

En inversant le rôle de P1 et P2 on obtient un second enfant :

E2 : 6 3 8 4 9 2 1 5 7

P1	2	1	4	5	3	8	6	7	9
P2	6	3	8	4	9	5	7	2	1
E1	2	1	4	5	3	6	8	9	7
E2	6	3	8	4	9	2	1	5	7

3.3.3.2 Opérateurs 2XC et 2XL

L'opérateur 2X est une extension de 1X mais avec 2 points de coupure. De la même manière qu'avec 1X, la partie centrale de P1 est conservée, et les gènes des parties latérales sont réordonnés en tenant compte de l'ordre de P2. On obtient ainsi le premier enfant E1. En inversant les rôles de P1 et P2 on obtient un second enfant E2. Ce croisement est noté 2XC (C pour central).

P1	2	1	4	5	3	8	6	7	9
P2	6	3	8	4	9	5	7	2	1
E1	2	1	4	5	3	8	6	9	7
E2	3	6	8	4	9	5	2	1	7

Inversement, on peut conserver les parties latérales de P1 et compléter la partie centrale avec les gènes manquants placés dans l'ordre de P2. Ce croisement sera noté 2XL (L pour latéral). En inversant les rôles de P1 et P2, on obtient également deux enfants E1' et E2'

P1	2	1	4	5	3	8	6	7	9
P2	6	3	8	4	9	5	7	2	1
E1'	2	1	3	8	4	5	6	7	9
E2'	6	3	4	5	8	9	7	2	1

3.3.3.3 Opérateur ERX (Edge Recombination Crossover)

Lorsqu'aucun temps d'attente n'est toléré entre les opérations d'un travail, le problème de flowshop peut être vu comme un problème de voyageur de commerce [Wie72]. ERX est un opérateur de croisement conçu par Whitley [WSF89] pour le problème du voyageur de commerce symétrique. Nous utilisons ici son extension pour la conservation de couples dans le cas de problèmes non symétriques. Cette extension est parfois dénommée ARX pour Arc Recombination Crossover (voir Portmann et Vignier [PV00] pour une description de l'opérateur ERX et ARX). Par abus de langage nous désignerons également cette extension par ERX.

Pour constituer la séquence de l'enfant E, la racine α est le travail courant et le premier travail de la séquence de l'enfant E.
Tant qu'il reste des travaux à placer :

1. Soit a le successeur du travail courant chez P1.
2. Soit b le successeur du travail courant chez P2.
3. Si a et b figurent déjà chez E alors choisir aléatoirement un travail non encore traité et le rajouter à E.
4. Sinon, si b figure déjà chez E, alors rajouter a.
5. Sinon, si a figure déjà chez E, alors rajouter b.
6. Sinon, si a possède moins de successeurs non encore placés que b, alors rajouter a à E.
7. Sinon, si b possède moins de successeurs non encore placés que a, alors rajouter b à E.
8. Sinon choisir aléatoirement un travail entre a et b et le rajouter à E.
9. Le travail rajouté à E devient le nouveau travail courant.

Voici un exemple d'enfant obtenu à partir des parents P1 et P2 (en gras les choix aléatoires de l'étape 3 et en italique ceux de l'étape 8) :

P1	α	2	1	4	5	3	8	6	7	9	ω
P2	α	6	3	8	4	9	5	7	2	1	ω

Travaux	α	1	2	3	4	5	6	7	8	9
Liste des successeurs	2	4	1	8	5	3	3	2	4	5
dans P1 et P2	6	ω			9	7	7	9	6	ω

P1	α	2	1	4	5	3	8	6	7	9	ω
P2	α	6	3	8	4	9	5	7	2	1	ω
E	α	2	1	4	9	5	7	**3**	8	6	ω

3.3.3.4 Opérateurs TSX et TSXW (Traveling Salesman Crossover)

Un autre opérateur utilisé pour résoudre le problème du voyageur de commerce a été développé par Grefenstette [GGRG85]. Pour constituer la séquence de l'enfant, on choisit aléatoirement la première ville de l'un des deux parents qui devient la ville de départ. On compare ensuite les 2 villes suivant la première dans la tournée des parents et on complète la séquence par celle qui minimise la distance à la ville de départ. Si l'une des villes a déjà été choisie, on complète par celle restante. Si les deux villes ont été choisies, on choisit aléatoirement une ville non encore sélectionnée pour compléter la tournée. Ce type d'opérateur est «orienté donnée» dans la mesure où l'opérateur de croisement tient compte de la valeur des données du problème et pas uniquement du contenu du chromosome (voir Portmann et Aloulou [PA01]).

Pour constituer la séquence de l'enfant E, on choisit aléatoirement le premier travail de P1 ou de P2, celui-ci devient le travail courant et le premier travail de E.
Tant qu'il reste des travaux à placer :

1. Soit a le successeur du travail courant chez P1.
2. Soit b le successeur du travail courant chez P2.
3. Si ni a ni b ne figurent encore chez E, choisir celui qui minimise la distance avec le travail courant et le rajouter à E, choix aléatoire en cas d'égalité.
4. Sinon, si a figure déjà chez E, alors rajouter b.
5. Sinon, si b figure déjà chez E, alors rajouter a.
6. Sinon, si a et b figurent déjà chez E, alors choisir aléatoirement un travail non encore traité et le rajouter à E.
7. Le Travail rajouté à E devient le nouveau travail courant.

Pour calculer la distance entre deux travaux i et j, on peut imaginer plusieurs méthodes. Dans le cas des problèmes sans attente, la date de fin C_i du travail i est uniquement fonction de la date de début de la première opération $O_{1,i}$ du travail i. Sur ce constat, Wiesmer [Wie72] a considéré la distance entre deux travaux i et j comme étant le «retard» induit sur le démarrage du travail j par le placement des opérations du travail i. La distance de i à j est donc $s_{1,j} - s_{1,i}$. L'opérateur de croisement basé sur cette distance sera noté TSXW (W pour Wiesmer).

Dans le cas présent, nous n'avons pas exclusivement affaire à des problèmes sans attente. Comme en outre on cherche à minimiser le C_{max}, il nous a semblé naturel de définir la

distance en faisant apparaître C_j et C_i. A partir d'un ordonnancement vide, nous plaçons les opérations du travail i puis celles du travail j et nous définissons la distance entre les deux travaux comme étant $C_j - C_i$. L'opérateur de croisement basé sur cette distance est noté TSX.

Une autre approche aurait été de calculer de manière dynamique, lors de la construction d'une solution, la distance séparant le dernier travail placé des travaux restant encore à ordonnancer ; le choix du travail étant celui qui minimise cette distance. Si cette approche devrait permettre de générer de bien meilleures solutions, elle n'a pas retenu notre attention dans la mesure où elle est trop forte consommatrice de CPU. Une variante consiste à n'appliquer cette nouvelle approche que pour les 3 ou 4 derniers travaux restant à placer de sorte à avoir une optimisation locale.

3.3.3.5 Opérateur TSXD (TSX Déterministe)

Le principe est identique à celui de TSX et utilise la même distance, excepté qu'on choisit systématiquement le premier travail du parent ayant la plus grande force (makespan le plus petit) pour débuter la séquence du fils. Si les deux travaux suivant le travail courant figurent déjà dans la séquence du fils, on choisit parmi les travaux restants celui qui minimise la distance au travail courant. Ce croisement est déterministe, on a supprimé toute la partie aléatoire de TSX.

Pour constituer la séquence de l'enfant E, on choisit le premier travail du parent ayant la meilleure évaluation, celui-ci devient le travail courant et le premier travail de E.
Tant qu'il reste des travaux à placer :

1. Soit a le successeur du travail courant chez P1.
2. Soit b le successeur du travail courant chez P2.
3. Si ni a ni b ne figurent encore chez E, choisir celui qui minimise la distance avec le travail courant et le rajouter à E, choix aléatoire en cas d'égalité.
4. Sinon, si a figure déjà chez E, alors rajouter b.
5. Sinon, si b figure déjà chez E, alors rajouter a.
6. Sinon, si a et b figurent déjà chez E, alors choisir parmi les travaux non encore traités celui qui minimise la distance au travail courant et le rajouter à E.
7. Le travail rajouté à E devient le nouveau travail courant.

3.3.4 Qualité des Opérateurs de croisement

3.3.4.1 Mode Opératoire

Nous avons implémenté l'algorithme génétique 3.1 avec les paramètres suivants :
- $Q = 100$;

- $p_{mut} = 0$ (aucun opérateur de mutation n'est appliqué) ;
- l'algorithme stoppe au bout de 50 générations ;

3.3.4.2 Compteurs

Pour mesurer la qualité des opérateurs, nous avons créé plusieurs compteurs et comptabilisé, lors de chaque cycle, le nombre de fois où un individu est meilleur ou moins bon que ses parents. Pour chaque couple d'individus, le père est celui qui possède la meilleure évaluation et la mère celui qui possède la moins bonne. Nous souhaitons comptabiliser :
- le nombre de fois où un enfant est meilleur que son père
- le nombre de fois où un enfant est meilleur que sa mère mais mois bon que son père
- le nombre de fois où un individu est moins bon que ses deux parents
- le nombre de fois où un enfant est la copie conforme de son père ou sa mère

Au fur et à mesure de l'évolution, la population va converger vers des individus dont l'évaluation va être relativement uniforme. Si en début d'évolution, la plupart des individus ont une évaluation différente, en fin d'évolution (au bout des 50 cycles), la probabilité de trouver des parents avec une évaluation identique augmente fortement. Se pose alors le problème de la classification des enfants qui résultent du croisement de deux parents avec la même évaluation. De même, un opérateur qui reproduit son père ou sa mère ne peut pas être considéré comme un bon opérateur. Pour lever toute ambiguïté, nous avons décidé de comptabiliser les individus de la manière suivante :

E1 Nombre de croisements où l'évaluation de l'enfant est strictement inférieure à celle de son père (l'enfant est meilleur que ses deux parents)

E2 Nombre de croisements où l'évaluation de l'enfant est supérieure ou égale à celle de son père et inférieure ou égale à celle de sa mère (comptabilise également les enfants clônes d'un des parents)

E3 Nombre de croisements où l'évaluation de l'enfant est strictement supérieure à celle de sa mère (l'enfant est moins bon que ses deux parents)

R Nombre de croisements où l'enfant est un clône de son père ou de sa mère

Q Nombre de croisements où les parents possèdent la même évaluation. Ce compteur est représentatif de l'uniformité de la population.

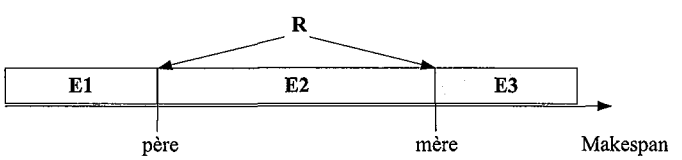


FIG. 3.6 – Compteurs

3.4 Première série d'expériences

3.4.1 Génération des jeux d'essai

Cette première série d'expériences a pour objectif de comparer entre eux les opérateurs génétiques. Nous voulons essentiellement savoir s'il y a une corrélation entre les performances des opérateurs et les différents types d'inégalités de potentiels (induisant des contraintes lâches, moyennes ou très dures, c'est-à-dire sans attente). En fait, nous avons construit 3 familles : une sans aucune contrainte d'écart maximal, une avec contraintes d'écart maximal moyennement serrées entre tout couple d'opérations successives, une avec contraintes d'écart maximal de type sans attente (opérations couplées séparées par une durée imposée) entre tout couple d'opérations successives.
Nous sommes conscients du fait que d'autres corrélations pourraient exister, comme par exemple un opérateur meilleur avec une machine fortement goulet, mais on ne peut tout tester à la fois et c'est pourquoi nous essayons de rester le plus neutre possible en générant de manière homogène les durées opératoires associées à chaque produit et chaque machine. Nous avons généré 2 familles de problèmes de taille 10x30 et 5x20 avec 50 instances pour chaque famille.

Les temps opératoires ont été choisis aléatoirement entre deux bornes $p_{min} = 50$ et $p_{max} = 100$. Pour chaque couple d'opérations successives au sein du même travail, le temps d'attente minimal entre la fin de la première opération et le début de la seconde est choisi aléatoirement entre 0 et $d_{max} = 50$; le temps d'attente maximal entre la fin de la première opération et le début de la seconde a été choisi aléatoirement entre le temps d'attente minimal et $2p_{min}$, de sorte que les contraintes d'écart maximal aient un effet moyen sur l'ensemble de la gamme (il est à noter que par rapport au flowshop sans attente, on crée du «mou» entre chaque couple d'opérations successives ce qui conduit à une marge importante entre la première et la dernière opération d'un travail). En conséquence, suite aux résultats de dominance du paragraphe 3.2, les seuls ordonnancements valides sont pour cette famille des ordonnancements de permutation.

Pour chaque jeu d'essai, l'algorithme a été appliqué 3 fois avec les différents opérateurs de croisement :

- une première fois sur les jeux d'essai générés selon le principe décrit précédemment.

- une seconde fois sur les jeux d'essai précédents où on a conservé les temps opératoires et les temps d'attente minimaux mais où les temps d'attente maximaux correspondent exactement aux temps d'attente minimaux. Ces jeux d'essai sont «sans attente» dans la mesure où à l'issue du temps d'attente minimal (correspondant à un temps de transfert), l'exécution de l'opération suivante doit débuter immédiatement.

- une troisième fois sur les jeux d'essai précédents où on a conservé les temps opératoires et les temps d'attente minimaux mais supprimé toute contrainte d'écart maximal.

Les résultats doivent nous permettre de mesurer :
- la qualité des opérateurs sur des jeux d'essai sans écarts maximaux
- la qualité des opérateurs sur des jeux d'essai avec écarts maximaux
- la qualité des opérateurs sur des jeux d'essai sans attente
- l'évolution de la qualité des opérateurs au cours des cycles
- l'évolution de la qualité des opérateurs en fonction de la tension sur les écarts maximaux

3.4.1.1 Lecture des figures

Tous les résultats et toutes les figures des tests correspondants se trouvent en annexe section 3.1 et 3.2.

Les figures A.3.1, A.3.2 et A.3.3 représentent l'évolution du compteur E1 pour chaque opérateur au cours des cycles pour les jeux d'essai de taille 10x30 sans contraintes d'écart maximal (figure A.3.1), avec contraintes d'écart maximal (figure A.3.2) et sans attente (figure A.3.3) :
abscisse : cycles pour lesquels les mesures ont été effectuées ;
ordonnée : pourcentage moyen du compteur E1 sur l'ensemble des 50 jeux d'essai.

La figure A.3.4 présente l'évolution du compteur E1 pour chaque opérateur lors du cycle initial 0 en fonction de la tension sur les écarts maximaux :
abscisse : 3 types de jeux d'essai (sans contraintes d'écart maximal (SPN), avec contraintes d'écart maximal (PN) et sans attente (SA)) ;
ordonnée : pourcentage moyen du compteur E1 sur l'ensemble des 50 jeux d'essai.

Les figures A.3.5, A.3.6 et A.3.7 représentent l'évolution du makespan moyen pour chaque opérateur au cours des cycles pour les jeux d'essai sans contraintes d'écart maximal (figure A.3.5), avec contraintes d'écart maximal (figure A.3.6) et sans attente (figure A.3.7) :
abscisse : cycles pour lesquels les mesures ont été effectuées ;
ordonnée : makespan moyen sur l'ensemble des 50 jeux d'essai.

Les mêmes types de figures se trouvent en annexe pour les jeux d'essai de taille 5x20.

3.4.2 Analyse des résultats

Les résultats obtenus n'ont pas de valeur absolue. Ils ne reflètent que le comportement des opérateurs sur les jeux d'essai considérés. Les mêmes tests sur d'autres jeux d'essai générés avec d'autres contraintes ont donné des résultats tout à fait différents. En outre, les moyennes obtenues sur les jeux d'essais retenus ont une forte variance. On ne peut donc pas en tirer de règle absolue. Ils nous permettent tout au plus de mettre en évidence des tendances. Les mêmes tendances ont été observées pour les deux familles de jeux d'essai.

3.4.2.1 Evolution des compteurs au cours des cycles

Les deux opérateurs 2X et 2XC ont un comportement similaire (ce qui était attendu). Ils obtiennent de bons résultats pour l'ensemble des jeux d'essai et présentent une dégradation «normale» au cours des cycles. Leur taux de reproductibilité reste à des niveaux raisonnables au cours des cycles.

L'opérateur ERX s'est montré décevant. En début de cycle, la répartition des compteurs E1, E2 et E3 est relativement homogène de l'ordre de 1/3 pour chacun, ce qui laisse supposer des résultats aléatoires pour les enfants, non corrélés avec les parents. L'opérateur ERX se dégrade très rapidement au cours des cycles et ne parvient que rarement à améliorer l'évaluation d'un des deux parents, le compteur E3 culminant à plus de 90% dès le 20ème cycle et quel que soit le type de jeux d'essai. A noter cependant qu'il possède le plus faible taux de reproduction des parents, à de rares exceptions, les enfants ne sont jamais les clônes des parents.

TSXW, TSX et TSXD se sont montrés assez décevants sur les jeux d'essai sans contraintes d'écart maximal et avec contraintes d'écart maximal. Par contre, sur les jeux d'essai sans attente, TSXD et TSX se sont avérés être de très bonne qualité. Lors du cycle initial, TSX et TSXD parviennent à améliorer les évaluations des deux parents dans respectivement 38% et 44% des croisements. L'opérateur TSXW continue à se montrer assez quelconque. Conformément à ce qui était attendu, le taux de reproductibilité des opérateurs TSX, TSXW et TSXD est élevé. On notera cependant que le taux de reproductibilité de TSXD n'est pas excessivement plus élevé que celui de TSX et TSXW contrairement à nos craintes.

L'opérateur 1X apparaît comme un opérateur moyen si l'on se réfère uniquement à l'évolution du compteur E1. De plus, il possède l'un des taux de reproductibilité le plus élevé ce qui peut paraître paradoxal pour un opérateur a priori «aléatoire». Par contre, le compteur E2 est prépondérant, preuve que 1X améliore globalement l'évaluation de la population.

3.4.2.2 Evolution du compteur E1 en fonction de la tension sur les écarts maximaux

TSX et TSXD sont les seuls à améliorer notablement leurs performances lorsque la tension sur les écarts maximaux croît. On note une corrélation certaine entre la qualité de ces opérateurs et la tension sur écarts maximaux. Les autres opérateurs se montrent assez stables sur l'ensemble des jeux d'essai sans contraintes d'écart maximal, avec contraintes d'écart maximal et sans attente. On peut noter toutefois une légère amélioration des performances pour les jeux d'essai avec contraintes d'écart maximal puis une diminution de ces performances pour les problèmes sans attente. Il semblerait que la présence de contraintes «moyennement fortes» permet aux opérateurs de mieux différencier les individus que l'absence de contraintes ou des contraintes trop fortes.

3.4.2.3 Evolution du makespan

L'opérateur 1X est l'opérateur de tous les paradoxes. Malgré un niveau moyen du compteur E1, 1X obtient d'excellents résultats en terme de makespan pour les jeux d'essai sans contraintes d'écart maximal et avec contraintes d'écart maximal (figures A.3.5 et A.3.6). Il n'y a que pour les jeux d'essai sans attente qu'il est supplanté par les opérateurs TSX, TSXW et TSXD (figure A.3.7).

3.5 Autodétermination

L'étude précédente fait ressortir qu'il n'y a pas d'opérateur expérimentalement dominant. Chaque opérateur est plus ou moins adapté selon le type de problème ou de période au cours de l'évolution. Il est du coup difficile d'opter pour tel ou tel opérateur et espérer obtenir systématiquement de bons résultats. Pour pallier cet inconvénient, nous avons implémenté une solution où l'algorithme s'autodétermine en fonction d'un historique.

3.5.1 Principe

Au début du processus, tous les opérateurs implémentés disposent d'une équiprobabilité d'être sélectionné pour le croisement. Au fur et à mesure des croisements, si un opérateur améliore l'évaluation des 2 parents, sa probabilité d'être choisi au coup suivant augmente.

Considérons k opérateurs notés $C_1, .., C_k$. On associe à chaque opérateur un compteur N_i et deux valeurs N_i^{ini} et N_i^{mini}. Les compteurs sont initialisés à $N_i = N_i^{ini}$. Ces valeurs peuvent être identiques, mais on peut très bien imaginer des niveaux d'initialisation différents si on désire privilégier dès le départ un opérateur au détriment des autres. La valeur N_i^{mini} est la valeur minimale que peut atteindre le compteur N_i : $1 \leq N_i^{mini} \leq N_i$.

Pour tout couple d'individus (père, mère), on choisit aléatoirement un nombre n entre 0 et $\sum_{i=1}^{k} N_i$. L'opérateur de croisement à appliquer est l'opérateur C_j tel que

$$\sum_{i=1}^{j-1} N_i \leq n < \sum_{i=1}^{j} N_i$$

Si l'opérateur de croisement permet d'obtenir un fils dont l'évaluation est supérieure à celle des parents, on incrémente son compteur, ce qui a pour effet d'accroître sa probabilité d'être sélectionné au prochain croisement.

Faut-il également décrémenter les compteurs lorsque le fils engendré par un croisement est moins bon que son père ?

Si on ne décrémente pas les compteurs, on risque de ne jamais remettre en cause au cours des cycles la hiérarchie des opérateurs de croisement. Nous avons vu que certains opérateurs sont excellents au début d'évolution mais se dégradent rapidement. Le fait que les compteurs ne fassent que croître induit un phénomène d'attraction. Les opérateurs bons en début d'évolution vont en effet prendre énormément de poids et être sélectionnés souvent. Le nombre des sélections va compenser leur perte de performance et leur permettre de se maintenir à un niveau élevé. De plus, en fin d'évolution, la population a convergé vers de bons individus. Le choix de tel ou tel opérateur ne conduit plus à des améliorations flagrantes. Il y a un nivellement de la qualité et on s'attend forcément à une équiprobabilité des opérateurs ce qui risque de ne pas être le cas sans décrémentation.

Il faut donc décrémenter les compteurs en cas de résultat négatif sinon on ne remet jamais en cause les choix de tel ou tel opérateur en cours d'évolution. Cependant, on ne peut pas augmenter le compteur si l'opérateur permet d'obtenir un fils meilleur que son père et le diminuer systématiquement dans le cas contraire. Les décrémentations étant plus fréquentes que les incrémentations (aucun opérateur ne dépasse le seuil de 50% pour le compteur E1), tous les compteurs se retrouveraient rapidement à leur niveau minimal N_i^{mini} et aucun opérateur ne serait en mesure de s'affirmer.
Au lieu de décrémenter systématiquement le compteur N_i de l'opérateur C_i, nous choisissons un nombre n entre 0 et $\sum_{j=1}^{k} N_j$. Si n est inférieur à αN_i, le compteur est décrémenté, sinon il ne l'est pas. Le paramètre α est un nombre positif dont le choix arbitraire est à la discrétion de l'utilisateur, permettant d'augmenter ou diminuer la probabilité de décrémenter le compteur. Par défaut $\alpha = 1$.
De cette manière, nous mettons en place un système de «ressort» qui aura tendance à ramener vers des niveaux plus raisonnables un compteur qui prend trop d'importance. Plus un opérateur prendra un poids relatif important par rapport aux autres opérateurs et plus il aura de chance d'être décrémenté si ses résultats se dégradent.
Les opérateurs retenus pour la méthode d'autodétermination sont :

- 2XC
- 2XL
- TSXD
- 1X

L'idée de travailler avec plusieurs opérateurs de croisement pour accroître les capacités d'exploration de l'espace des solutions et avoir une méthode moins «problem specific» n'est pas non plus révolutionnaire. Wang et Zheng [WZ03] ont travaillé avec un pool de 4 opérateurs. A chaque cycle la population est divisée en 4 sous population pour chacun des opérateurs. Les auteurs n'ont cependant pas différencié les opérateurs de croisement qui ont tous le même poids. Pour une toute autre problématique, Gacôgne [Gac99] applique également différents opérateurs au cours des cycles en fonction de leur aptitude à améliorer la population. Les opérateurs considérés ne sont cependant pas uniquement des opérateurs

Algorithme 3.2 Processus de sélection d'un opérateur de croisement

Soit (père, mère) un couple d'individus ; père est celui qui a la meilleur évaluation, mère celui qui a la moins bonne.

1. $N = \sum_{i=1}^{k} N_i$
2. On choisit aléatoirement un nombre n entre 0 et N.
3. L'opérateur sélectionné pour la reproduction du (père, mère) est l'opérateur C_j tel que
$$\sum_{i=1}^{j-1} N_i \leq n < \sum_{i=1}^{j} N_i$$
4. On engendre le fils du couple (père, mère) et on l'évalue.
5. Si l'évaluation du fils est meilleure que celle de son père, on incrémente N_j : $N_j = N_j + 1$.
6. Si l'évaluation du fils est inférieure ou égale à celle du père
   (a) on choisit aléatoirement un nombre p entre 0 et N.
   (b) Si $p < \alpha N_j$ alors $N_j = Max(N_j^{mini}, N_j - 1)$

de croisement mais également de mutation. Un pool d'opérateur initial est sélectionné dans un ensemble d'opérateurs (croisement, mutation..). Le score de chaque opérateur est incrémenté au cours des cycles selon sa capacité à accroître les performances des individus. Les opérateurs sont appliqués successivement dans l'ordre décroissant de leur score. Si au cours d'un cycle aucune évolution n'est constatée, un nouveau pool d'opérateurs est créé.

3.5.2 Evolution des compteurs N_i

Afin de mesurer l'évolution des compteurs et mettre en évidence, selon le type de problème, la dominance de tel ou tel opérateur, nous avons inséré le mode de sélection 3.2 de l'opérateur de croisement dans l'algorithme génétique 3.1. Le nouvel algorithme est présenté sur la figure 3.3. Les paramètres de l'algorithme 3.3 sont :
- $Q = 100$;
- $p_{mut} = 0$ (aucun opérateur de mutation n'est appliqué) ;
- l'algorithme stoppe au bout de 100 générations ;
- pour chaque opérateur, $N^{ini} = 10$ et $N^{mini} = 5$.

Les figures 3.7, 3.8 et 3.9 montrent l'évolution (en %) des compteurs au cours des cycles pour les jeux d'essai 10x30. Initialement (cycle 0), tous les opérateurs ont la même probabilité d'être sélectionnés, tous les compteurs sont à 25%. Très rapidement certains opérateurs deviennent dominants, 2XL pour les jeux d'essai avec et sans écarts maximaux, TSXD pour les jeux d'essai sans attente. Au fur et à mesure de l'évolution, la population converge, le choix de tel ou tel opérateur ne conduit plus à une amélioration et les comp-

Algorithme 3.3 Algorithme génétique avec méthode d'autodétermination

1. Générer la population initiale P_0 de Q individus
2. Evaluer la "force" des individus de la population P_0
3. Tant que la condition d'arrêt n'est pas satisfaite
   (a) Sélectionner $Q/2$ couples d'individus dans la population P_{k-1}
   (b) Pour chaque couple :
      i. Sélectionner l'opérateur de croisement en utilisant la méthode d'autodétermination
      ii. Appliquer l'opérateur de croisement sélectionné afin d'obtenir les enfants
      iii. Avec une probabilité p_{mut}, appliquer l'opérateur de mutation à chaque enfant
      iv. Evaluer la "force" des enfants
      v. Ajouter les nouveaux individus à la population P_{k-1}
   (c) Epurer la population pour supprimer les doublons et ne conserver que les Q "meilleurs" individus qui constituent la nouvelle population P_k

teurs s'équilibrent autour du niveau minimal et donnent un pourcentage de sélection de 25% car tous les niveaux minimum sont identiques et fixés à 5.

Les figures 3.10, 3.11 et 3.12 montrent l'évolution (en %) des compteurs au cours des cycles pour les jeux d'essai 5x20. Le comportement des compteurs est identique à celui pour les jeux d'essai 10x30.

3.5.3 Autodétermination vs méthode classique

Nous avons comparé la méthode d'autodétermination avec la méthode classique où un opérateur donné est systématiquement choisi. Pour ce faire, nous avons exécuté concurrentiellement chacune des deux méthodes sur les 50 jeux d'essai de chaque type. Pour chaque jeux d'essai, au cours de l'évolution, nous avons mesuré toutes les 500 constructions l'évaluation du meilleur individu et l'évaluation moyenne des 10 meilleurs individus obtenus par chaque méthode. Nous avons ainsi comptabilisé au cours des cycles :

AD le pourcentage des jeux d'essai pour lesquels la méthode d'autodétermination obtient de meilleurs résultats ;

CL le pourcentage des jeux d'essai pour lesquels la méthode classique obtient de meilleurs résultats ;

EQ le pourcentage des jeux d'essai pour lesquels les deux méthodes obtiennent des résultats identiques.

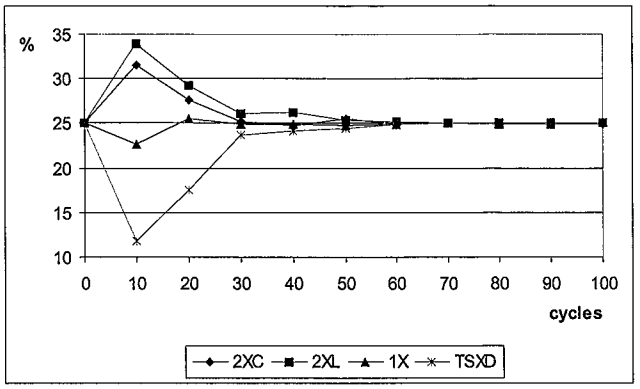


FIG. 3.7 – Evolution des compteurs N_i - jeux d'essai 10x30 sans écarts maximaux

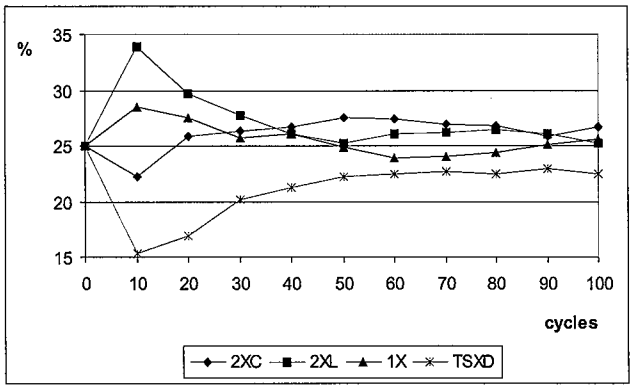


FIG. 3.8 – Evolution des compteurs N_i - jeux d'essai 10x30 avec écarts maximaux

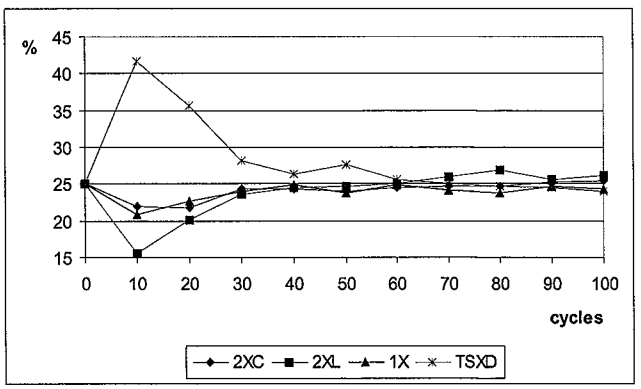


FIG. 3.9 – Evolution des compteurs N_i - jeux d'essai 10x30 sans attente

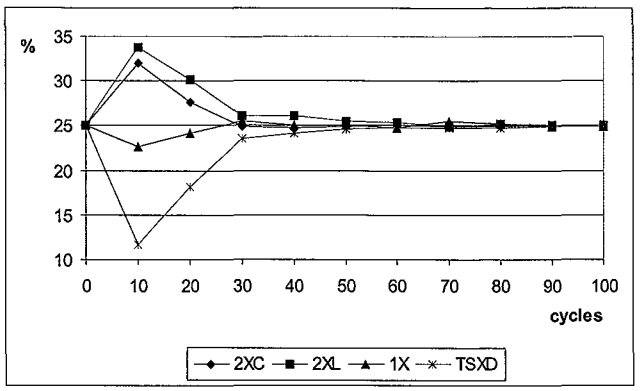


FIG. 3.10 – Evolution des compteurs N_i - jeux d'essai 5x20 sans écarts maximaux

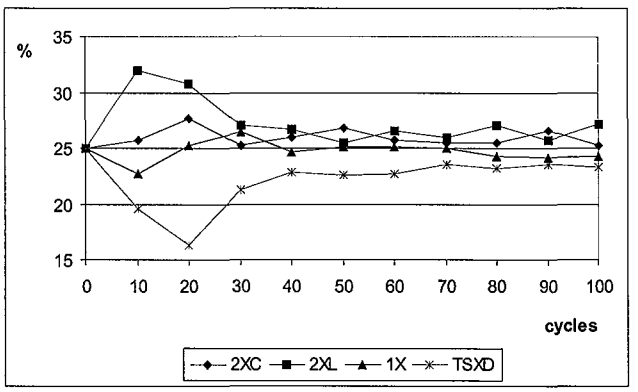


FIG. 3.11 – Evolution des compteurs N_i - jeux d'essai 5x20 avec écarts maximaux

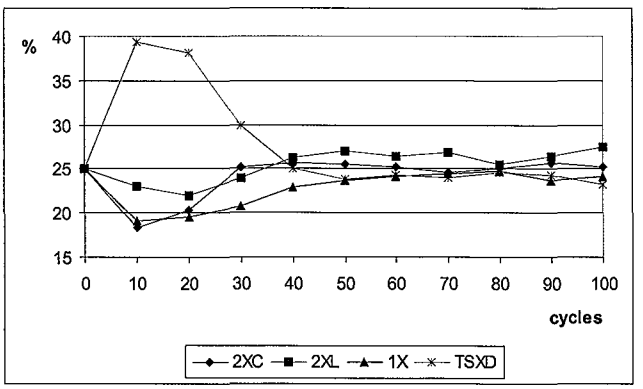


FIG. 3.12 – Evolution des compteurs N_i - jeux d'essai 5x20 sans attente

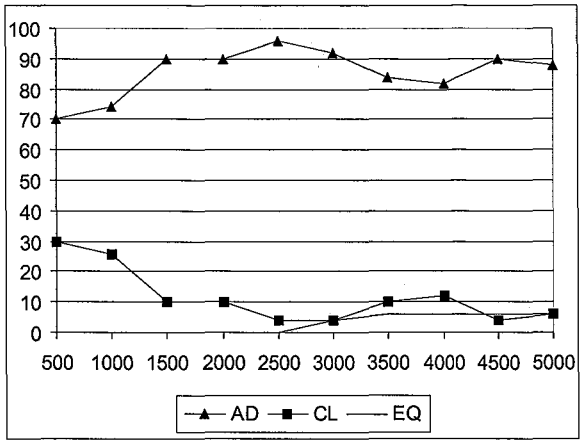


FIG. 3.13 – Auto vs 1X - jeux d'essai 10x30 sans écarts maximaux - résultats en tenant compte du meilleur individu

Toutes les figures sont en annexes. La figure 3.13 en est un exemple. Elle représente, pour les jeux d'essai de taille 10x30, les trois courbes AD, CL et EQ en mesurant l'évaluation du meilleur individu. En abscisse figure le nombre de croisements effectués et en ordonnée les différents pourcentages pour les deux méthodes ainsi que le pourcentage des jeux d'essai pour lesquels les deux méthodes obtiennent des résultats identiques.
On constate qu'au bout de 2500 croisements et évaluations, le meilleur individu obtenu par la méthode d'autodétermination avait un makespan inférieur à son homologue généré par la méthode classique dans 86% des 50 jeux d'essai.
La figure 3.14 présente les mêmes informations mais on a mesuré, pour les deux méthodes, la moyenne des évaluations des 10 meilleurs individus.

3.5.3.1 Autodétermination vs 1X

Nous avons d'abord comparé la méthode d'autodétermination avec la méthode classique où l'opérateur 1X est systématiquement sélectionné. L'opérateur 1X présentait en effet les meilleurs résultats pour les jeux d'essai avec et sans écarts maximaux mais était surclassé par l'opérateur TSXD pour les jeux d'essai sans attente. Face à la méthode d'autodétermination, la méthode classique et l'opérateur 1X sont largement surclassés pour les jeux d'essai de taille 10x30 et ce quel que soit leur nature (figures A.3.15 pour les jeux sans écarts maximaux, A.3.16 pour les jeux avec écarts maximaux, A.3.17 pour les jeux sans attente). Même si en début d'évolution, la méthode d'autodétermination et la méthode classique parviennent à faire jeu égal pour les jeux avec écarts maximaux, très rapidement la méthode d'autodétermination se montre plus performante que son homologue et obtient de meilleurs résultats pour plus de 80% des jeux d'essai au bout des 100 cycles. Pour les jeux sans-attente, à part quelques rares exceptions, la méthode d'autodétermination

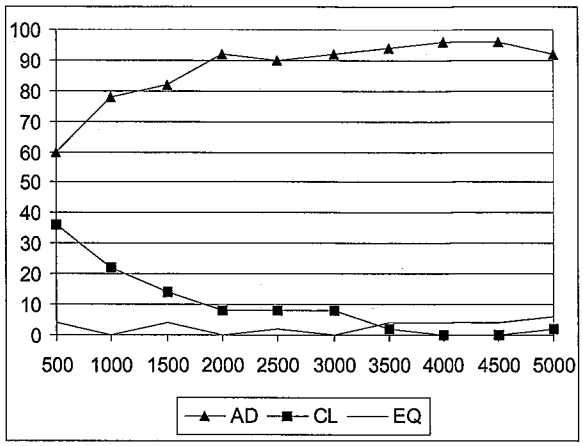


FIG. 3.14 – Auto vs 1X - jeux d'essai 10x30 sans écarts maximaux - résultats en tenant compte des 10 meilleurs individus

se montre toujours la plus performante. Nous avons également mesuré au bout des 100 cycles, l'écart moyen entre les résultats obtenus par la méthode d'autodétermination et la méthode classique en terme de makespan :
- 1.16% (1.22% sur les 10 meilleurs individus) pour les jeux sans écarts maximaux
- 1.10% (1.18% sur les 10 meilleurs individus) pour les jeux avec écarts maximaux
- 5.80% (6.33% sur les 10 meilleurs individus) pour les jeux sans attente

Pour les jeux d'essai de taille 5x20, la méthode d'autodétermination est toujours dominante bien que sa suprématie soit moins marquée (voir figures A.3.18 pour les jeux sans écarts maximaux, A.3.19 pour les jeux avec écarts maximaux, A.3.20 pour les jeux sans attente).

3.5.3.2 Autodétermination vs TSXD

Nous avons ensuite comparé la méthode d'autodétermination avec la méthode classique où l'opérateur TSXD est systématiquement sélectionné. L'opérateur TSXD offrait les meilleurs résultats pour les jeux d'essai sans attente. Face à la méthode d'autodétermination, la méthode classique et l'opérateur TSXD sont largement surclassés pour les jeux d'essai de taille 10x30 (figures A.3.21 pour les jeux sans écarts maximaux, A.3.22 pour les jeux avec écarts maximaux, A.3.23 pour les jeux sans attente) et de taille 5x20 (figures A.3.24 pour les jeux sans écarts maximaux, A.3.25 pour les jeux avec écarts maximaux, A.3.26 pour les jeux sans attente).
Même si en début d'évolution, l'opérateur TSXD parvient à de meilleurs performances que la méthode d'autodétermination pour les jeux sans attente, au cours des cycles cette dernière parvient à faire mieux que son homologue et se montre plus performante pour plus de 80% des jeux d'essai au bout des 100 cycles.

3.5.3.3 Autodétermination vs 2X

Face à un opérateur individuel, la méthode d'autodétermination se montre toujours plus performante quel que soit l'opérateur sélectionné (1X, TSXD, 2XC..). Nous avons donc décidé de tester la méthode d'autodétermination face à la méthode classique mais en utilisant les opérateurs 2XC et 2XL au lieu d'un seul opérateur. On construit systématiquement les deux fils d'un couple en prenant les opérateurs 2XC et 2XL et en les rajoutant à la population (l'épuration ne retenant que les 100 meilleurs individus sur l'ensemble de la population).

Cette fois-ci, les résultats sont plus nuancés pour les jeux d'essai 10x30 (figures A.3.27 pour les jeux sans écarts maximaux, A.3.28 pour les jeux avec écarts maximaux, A.3.29 pour les jeux sans attente) et 5x20 (figures A.3.30 pour les jeux sans écarts maximaux, A.3.31 pour les jeux avec écarts maximaux, A.3.32 pour les jeux sans attente). Si la méthode d'autodétermination se montre toujours dominante pour les jeux d'essai sans attente, le couple d'opérateurs 2XC et 2XL se montre plus performant pour les autres jeux d'essai, du moins au début. En effet, comme nous construisons systématiquement les deux fils de chaque couple de parents, l'espace des solutions balayé par la méthode classique est plus important que celui de la méthode d'autodétermination. Par contre, au cours de l'évolution, la population de la méthode classique va converger plus rapidement et le fait de construire systématiquement les deux fils va conduire à de nombreux «déchets» dans la mesure où on aura peu d'améliorations pour beaucoup d'individus engendrés. La méthode d'autodétermination va ainsi profiter de cette situation pour combler petit à petit son retard et parvenir au bout du compte à dominer la méthode classique ou du moins à faire jeu égal. On note ainsi sur l'ensemble des figures une croissance régulière de la courbe de la méthode d'autodétermination au détriment de celle de la méthode classique.

3.5.3.4 Autodétermination vs 1X-TSXD

En utilisant la même approche que précédemment, nous avons testé la méthode d'autodétermination face au couple d'opérateurs 1X et TSXD. En sélectionnant les deux opérateurs qui se sont montrés les plus efficaces sur les différents jeux d'essai, nous pensions avoir ainsi une méthode capable de mettre en difficulté la méthode d'autodétermination. Au vu des résultats pour les jeux d'essai de taille 10x30 (figures A.3.33 pour les jeux sans écarts maximaux, A.3.34 pour les jeux avec écarts maximaux, A.3.35 pour les jeux sans attente) et ceux de taille 5x20 (figures A.3.36 pour les jeux sans écarts maximaux, A.3.37 pour les jeux avec écarts maximaux, A.3.38 pour les jeux sans attente) on se rend compte qu'excepté pour les jeux sans attente, la supériorité de la méthode d'autodétermination est toujours manifeste. On note cependant une légère dégradation des performances de la méthode d'autodétermination au cours des cycles, particulièrement pour les jeux d'essai sans écarts maximaux de taille 5x20.

3.6 Mutation

Pour l'instant nous n'avons pas considéré d'opérateur de mutation. Il convient d'analyser l'impact d'un tel opérateur sur notre méthode. Nous sommes en effet partis du postulat que l'utilisation de plusieurs opérateurs de croisement suffisait à assurer une diversité de la population. Nous avons effectué une série d'expériences sur les jeux d'essai de taille 5x20 et 10x30 pour comparer les méthodes avec et sans mutation. L'opérateur de mutation que nous utilisons, consiste simplement à transposer deux travaux choisis aléatoirement. Si $(5, 1, 4, 6, 3, 2)$ représente une solution, $(5, 6, 4, 1, 3, 2)$ en est une autre obtenue par mutation en transposant l'ordre de placement des travaux 1 et 6.
L'algorithme 3.3 a été exécuté sur les jeux d'essai 5x20 et 10x30, une fois avec $p_{mut} = 0$ et une fois avec $p_{mut} = 0.1$. Les figures A.3.39, A.3.40 et A.3.41 synthétisent les résultats obtenus par les deux algorithmes sur les jeux d'essai de taille 5x20. Les trois courbes de chaque figure comptabilisent au cours des cycles :

Auto le pourcentage des jeux d'essai pour lesquels la méthode d'autodétermination sans mutation obtient de meilleurs résultats ;

AutoMut le pourcentage des jeux d'essai pour lesquels la méthode d'autodétermination avec mutation obtient de meilleurs résultats ;

EQ le pourcentage des jeux d'essai pour lesquels les deux méthodes obtiennent des résultats identiques.

L'influence d'un tel opérateur n'est pas négligeable mais est dépendant du type et de la taille des jeux d'essai. Sur les jeux d'essai de taille 5x20, l'opérateur de mutation n'apporte rien pour les jeux avec écarts maximaux. Pour ceux sans attente, son apport est même légèrement négatif. En contre partie, pour les jeux d'essai sans écarts maximaux l'ajout d'un opérateur de mutation apporte un petit plus. En augmentant la taille des jeux d'essai (figures A.3.42, A.3.43 et A.3.44) cette influence est plus marquée sans pour autant devenir écrasante. En moyenne, la différence entre la meilleure solution obtenue par la méthode d'autodétermination sans mutation et celle avec mutation n'est que de 0.6% au bout des 100 cycles.
En comparaison, d'autres opérateurs de croisement comme l'opérateur 1X subissent, dès les jeux d'essai de taille 5x20 (figures A.3.45, A.3.46 et A.3.47) des influences plus importantes. L'opérateur TSXD quant à lui bénéficie également de l'ajout d'un opérateur de mutation dans le cas des jeux d'essai de taille 5x20 avec écarts maximaux (figure A.3.49) mais cet apport est négligeable pour les autres types de jeux d'essai (figure A.3.48 et figure A.3.50). Est-ce parce que TSXD est particulièrement adapté aux jeux d'essai sans attente et inadapté aux jeux sans écarts maximaux ?

On ne peut nier qu'un opérateur de mutation peut avoir un apport bénéfique sur la méthode d'autodétermination, particulièrement en fin d'évolution. Cette méthode, de par la diversité des opérateurs de croisement utilisés, se montre cependant moins sensible que la méthode classique à base d'un seul opérateur de croisement et permet de faire l'impasse

sur l'ajustement très fin des différents paramètres. Nous avons implémenté un algorithme où le paramètre p_{mut} est une variable. Ce dernier est incrémenté si aucune amélioration du C_{max} n'est constatée au cours d'un cycle et décrémenté dans le cas contraire. Par faute de temps, nous n'avons pas eu la possibilité de faire figurer les tests dans cette thèse.

3.7 Conclusion

Les études menées sur un panel d'opérateurs de croisement ont mis en évidence, une fois de plus, la non-existence d'opérateurs expérimentalement dominants et adaptés à tout type de jeu d'essai. Sur les instances de problèmes sans contraintes d'écart maximal ou moyennement contraints, les opérateurs classiques 1X, 2XC et 2XL se sont montrés efficaces. En contrepartie, sur des instances de problèmes sans attente ou fortement contraints, des opérateurs adaptés du problème du voyageur de commerce, TSX et TSXD, ont démontré leur supériorité.
Pour plus d'efficacité et offrir une adaptabilité accrue à tout type de problème, nous avons développé la méthode d'autodétermination qui permet de sélectionner l'opérateur de croisement le mieux adapté à l'instance du problème et à l'état courant de la population. Les expériences menées ont prouvé la supériorité de cette méthode de sélection sur la méthode classique. Le fait de travailler avec un panel d'opérateurs de croisement permet non seulement de couvrir efficacement un spectre plus large de problèmes mais également de permettre à un opérateur à bout de souffle d'être relayé par un autre opérateur. Cette méthode permet ainsi d'assurer une diversité plus importante de la population et faire l'économie de l'ajustement des paramètres correspondant à l'utilisation d'un opérateur de mutation.
Notre méthode d'autodétermination peut également être appliquée à d'autres problématiques comme les problèmes de flowshop de permutation sans temps mort autorisé sur les ressources. En effet, nous avons vu au second chapitre que si l'ordre de passage des n opérations $O_{1,j}$ sur la première ressource est défini, les algorithmes ARP-G et AOS-G permettent de construire des ordonnancement actif.

4

Algorithmes génétiques et amélioration par voisinage

Les algorithmes génétiques permettent de balayer rapidement l'espace des solutions mais sont inadaptés pour une recherche fine et précise. Dans le but de compléter notre approche génétique par une recherche locale, nous effectuons dans ce chapitre un rappel des différents voisinages utilisés dans les méthodes d'amélioration du type tabou ou recuit simulé. Nous étudions la validité de ces voisinages en présence de contraintes d'écart maximal. Nous améliorons notre approche génétique par une recherche locale basée sur ces voisinages et terminons par quelques résultats numériques.

4.1 Introduction

Comme nous l'avons déjà souligné, les algorithmes génétiques permettent de parcourir rapidement l'espace des solutions. Par contre, ils sont inadaptés à une recherche locale précise. De nombreux auteurs ont mis en évidence une topographie particulière de l'espace des solutions induite par le type de voisinage utilisé. Ainsi, l'espace des solutions peut présenter une structure avec de nombreuses vallées et une alternance importante de pics. Les optimaux locaux peuvent se succéder rapidement et être concentrés autour d'un optimum global [BKM94] [Ree98] [WBWH02]. L'algorithme génétique peut ainsi très bien approcher une solution optimale sans pouvoir l'atteindre. Une telle topographie de l'espace des solutions explique en partie la supériorité des algorithmes de recherche locale (tabou, recuit simulé..). Il est donc nécessaire de compléter l'algorithme génétique par une recherche locale. Ce type d'hybridation est communément appelé approche mémétique.

Dans cette partie, nous présentons plusieurs résultats sur les blocs critiques à la base de la définition de nombreux voisinages. Nous étudierons également la validité des différents théorèmes et voisinages utilisés dans le cadre de notre problématique avec contraintes d'écart maximal.

4.2 Etude sur les principaux voisinages

Traditionnellement, le voisinage $N(S)$ d'une solution S est un ensemble de solutions qui peuvent être obtenues à partir de S par des transformations prédéfinies que l'on appelle communément transitions ou mouvements. Ces mouvements sont nombreux et variés. Une fois définis le ou les opérateurs de transition, une méthode d'amélioration par voisinage consiste à partir d'une solution initiale et à explorer tout ou partie de son voisinage. Une solution de ce voisinage, en général celle donnant la meilleure valeur au critère considéré, va servir de point de départ pour une nouvelle recherche et ainsi de suite jusqu'à satisfaire une condition d'arrêt. L'objectif d'une telle stratégie est de corriger progressivement la solution initiale par une succession de transitions pour converger vers un optimum local qu'on espère global. De la nature des opérateurs de transition et du voisinage dépendra l'efficacité de la méthode de recherche locale.

4.2.1 Sans écarts maximaux

4.2.1.1 Chemin critique

Oublions pour l'instant les contraintes d'écart maximal. Considérons un graphe disjonctif $G = (V, C \cup D)$, avec racine α et puits ω. Une première étape consiste à arbitrer l'ensemble des arcs de D afin d'obtenir un graphe sans circuit de longueur strictement positive. Cette étape conduit à un ordonnancement respectant les contraintes de précédence et de ressource. La date de début d'une opération est obtenue en considérant le plus long

chemin valué de la racine à son sommet ; le makespan est égal à la longueur du plus long chemin valué de la racine α au puits ω. Un tel chemin est dit critique. Les opérations situées sur un chemin critique sont dites critiques et revêtent une importance particulière. En effet, si on retarde une opération critique d'un délai θ, l'ordonnancement sera retardé d'autant.

4.2.1.2 Voisinage de Van Laarhoven et al

Le voisinage le plus simple et le plus générique est défini par l'opérateur de transition qui consiste à permuter deux opérations adjacentes sur la même ressource. Deux opérations $O_{i,j}$ et $O_{k,l}$ sont consécutives si elles se suivent sans temps mort i.e. $s_{k,l} = s_{i,j} + p_{i,j}$ (ou $s_{i,j} = s_{k,l} + p_{k,l}$). Ce premier voisinage est très important de par sa taille et nécessite d'être réduit pour plus d'efficacité. Il convient en effet de supprimer les mouvements ne permettant pas d'améliorer le makespan.

Une amélioration notable a été apportée en 1992 par Van Laarhoven et al [LAL92] qui ont proposé un algorithme de recuit simulé utilisant le voisinage défini par l'opérateur de transition consistant à transposer deux opérations critiques consécutives situées sur le même chemin critique et la même ressource. Ils ont ainsi supprimé du voisinage générique les solutions obtenues en permutant deux opérations non critiques, transition qui ne permet jamais d'améliorer le critère C_{max}. Ils ont également démontré que ce voisinage rendait l'espace des solutions connexe.

4.2.1.3 Théorème des blocs critiques

Une nouvelle amélioration dans la définition de voisinage a été apportée par Grabowski et al [GNZ86] [GNS88] en définissant la notion de bloc critique :

Un bloc est un ensemble maximal d'opérations consécutives (au moins deux) sur la même ressource. Un bloc est dit critique s'il se trouve sur un chemin critique.

Les auteurs ont démontré le théorème des blocs critiques suivant :

Théorème 4.1 *Soit S un ordonnancement valide. S'il existe un autre ordonnancement valide S' de makespan strictement inférieur à celui de S, alors au moins une opération d'un bloc critique de S différente de la première (resp. dernière) opération du bloc doit être traitée avant (resp. après) toutes les autres opérations du bloc.*

La démonstration est issue de l'article de Brucker et al [BJS94]. Bien qu'initialement les auteurs ne considèrent que des contraintes de succession simple $d(O_{i,j}, O_{k,l}) = p_{i,j}$, d'autres contraintes de potentiels $0 \leq d(O_{i,j}, O_{k,l}) < p_{i,j}$ et $d(O_{i,j}, O_{k,l}) > p_{i,j}$, n'affectent en rien la démonstration.

Démonstration
Nous allons procéder par l'absurde. Supposons que pour obtenir S' aucune opération d'aucun bloc critique de S ne soit exécutée avant (resp. après) la première (resp. dernière) opération de ce bloc.

Soit

$$P = (\alpha, u_1^1, u_2^1, ..., u_{m_1}^1, ..., u_1^k, u_2^k, ..., u_{m_k}^k, \omega)$$

un chemin critique de S où $u_1^j, u_2^j, ..., u_{m_j}^j$ désigne le j^{me} bloc de P. Le graphe disjonctif arbitré de S' contient de ce fait les arcs
$u_1^j \longrightarrow u_i^j \quad (j = 1, .., k; i = 2, ..., m_j)$
$u_i^j \longrightarrow u_{m_j}^j \quad (j = 1, .., k; i = 1, ..., m_j - 1)$
Le graphe disjonctif arbitré de S' contient donc un chemin

$$P' = (\alpha, v_1^1, v_2^1, ..., v_{m_1}^1, ..., v_1^k, v_2^k, ..., v_{m_k}^k, \omega)$$

où $v_1^j = u_1^j$, $v_1^{m_j} = u_1^{m_j}$ et les opérations $v_2^j, ..., v_{m_j-1}^j$ sont une permutation des opérations $u_2^j, ..., u_{m_j-1}^j$. La longueur d'un chemin critique de S' ne peut être inférieure à la longueur du chemin P'. En notant $L(S')$ le makespan de S' nous obtenons

$$\begin{aligned} L(S') &\geq \sum_{j=1}^{k}(\sum_{i=1}^{m_j} p_{v_i^j}) \\ &= \sum_{j=1}^{k}(\sum_{i=1}^{m_j} p_{u_i^j}) \\ &= L(S) \end{aligned}$$

ce qui est une contradiction. □

Ce théorème stipule que pour espérer améliorer le makespan d'un ordonnancement valide S il faut que l'une des deux conditions suivantes soit vérifiée :

1. une opération d'un bloc critique de S différente de la première opération de ce bloc est traitée avant toutes les autres opérations de ce bloc

2. une opération d'un bloc critique de S différente de la dernière opération de ce bloc est traitée après toutes les autres opérations de ce bloc

Attention ! Ce théorème ne stipule pas qu'il suffise uniquement qu'une opération d'un bloc critique de S différente de la première opération de ce bloc soit traitée avant toutes les autres opérations de ce bloc, ou qu'une opération d'un bloc critique de S différente de la dernière opération de ce bloc soit traitée après toutes les autres opérations de ce bloc, pour que le makespan de S s'améliore. Il s'agit d'une condition nécessaire mais non suffisante. Pour améliorer le makespan de la solution, il peut être nécessaire d'appliquer plusieurs fois les opérateurs :

- O1 : une opération d'un bloc critique est exécutée immédiatement avant la première opération de ce bloc
- O2 : une opération d'un bloc critique est exécutée immédiatement après la dernière opération de ce bloc

Une seule application de ces opérateurs ne conduit pas forcément à une amélioration de la solution. Par contre, on peut démontrer que l'utilisation de O1 et O2 rend effectivement l'espace de recherche connexe.

Démonstration
Soit S_{op} un ordonnancement optimal et S un ordonnancement non optimal. D'après le théorème 4.1, il existe une opération O d'un bloc critique qui doit être ordonnancée dans S_{op} avant ou après toutes les autres opérations de ce bloc. On déplace cette opération à sa nouvelle position qu'elle va conserver. Si le nouvel ordonnancement ainsi obtenu n'est pas optimal, on continue en sachant que O va toujours conserver la même position relativement aux autres opérations du bloc. Les opérations et les ressources étant limitées, le nombre d'applications des deux opérateurs ne peut pas être infini et après un nombre fini d'itérations, on aboutira à la solution optimale. □

Brucker et al [BJS94] ont utilisé le théorème 4.1 et les deux opérateurs O1 et O2 pour élaborer une PSE de recherche de l'optimum. On peut encore citer Yamada et Nakano [YN96] qui utilisent une méthode de recuit simulé dont les voisinages sont définis grâce aux opérateurs de transition O1 et O2. Ils ont ainsi trouvé avec leur algorithme de voisinage, la solution optimale de l'un des problèmes de jobshop 10x10 formulé par Muth et Thompson en 1963.

La définition de bloc critique impose que les opérations constituant les blocs soient consécutives i.e. $s_{k,l} = s_{i,j} + p_{i,j}$ et non pas $s_{k,l} = s_{i,j} + d(O_{i,j}, O_{k,l})$. Cette condition est nécessaire comme le prouve l'exemple suivant. Considérons les deux ordonnancements valides S et S' de la figure 4.1. Nous avons 4 travaux soit 8 opérations dont les durées opératoires sont données dans le tableau 4.1. Le graphe potentiels-tâches correspondant est donné sur la figure 4.2. On constate que les opérations $O_{1,1}$ et $O_{1,2}$ ne sont pas consécutives car $d(O_{1,1}, O_{1,2}) = 8 > p_{1,1} = 6$. De même, les opérations $O_{1,3}$ et $O_{1,4}$ ne sont pas consécutives car $d(O_{1,3}, O_{1,4}) = 6 > p_{3,1} = 4$. En n'imposant pas la consécutivité des opérations d'un bloc, les seuls blocs critiques de cet exemple sont $\{O_{1,1}, O_{1,2}, O_{1,3}, O_{1,4}\}$ et $\{O_{2,1}, O_{2,2}, O_{2,3}, O_{2,4}\}$. Pourtant, une meilleure solution peut être obtenue en permutant simplement les deux opérations internes à ces blocs.

4.2.1.4 Voisinage de Nowicki et Smutnicki

En 1996, Nowicki et Smutnicki [NS96a] ont proposé un algorithme tabou utilisant un voisinage quasi identique à celui obtenu avec les deux opérateurs O1 et O2. Cependant, ils

opération	$O_{1,1}$	$O_{1,2}$	$O_{1,3}$	$O_{1,4}$	$O_{2,1}$	$O_{2,2}$	$O_{3,2}$	$O_{4,2}$
durée	6	6	4	6	6	4	5	6

TAB. 4.1 – Temps opératoires

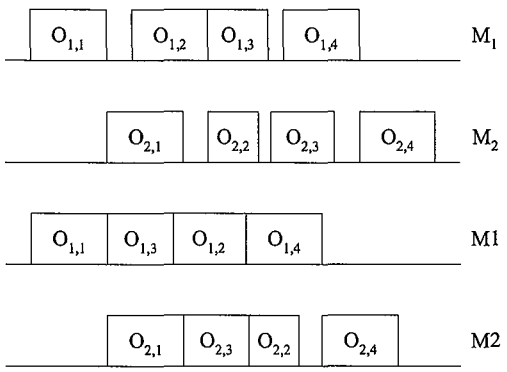


FIG. 4.1 – Ordonnancement valides S et S'

n'ont considéré que des transpositions des deux premières (resp. deux dernières) opérations de chaque bloc critique. Plus surprenant, pour définir leur voisinage ils ont sélectionné arbitrairement un chemin critique et démontré que toute transposition d'opérations critiques autre que celles des deux premières (resp. deux dernières) opérations de chaque bloc critique du chemin critique sélectionné, ne permettait pas d'améliorer le makespan de la solution. Même si leurs opérateurs de transition ne rendent pas l'espace de recherche connexe, l'efficacité de leur méthode a été mainte fois soulignée, notamment du fait de la taille réduite du voisinage et de l'efficacité de ce voisinage (seules les transitions permettant une amélioration immédiate de la solution ont été retenues).

Voisinage $N_{ns}(S)$ d'un ordonnancement S d'après Nowicki et Smutnicki :
Soit P un chemin critique de S constitué des blocs critiques $B_1, .., B_r$. $N_{ns}(S)$ est obtenu en
- transposant les deux premières opérations et les deux dernières opérations de chaque bloc B_i (1< i < r),
- transposant les deux dernières opérations de B_1
- transposant les deux premières opérations de B_r

Théorème 4.2 *Notons $N_{vl}(S)$ le voisinage d'un ordonnancement S d'après Van Laarhoven et al et $N_{ns}(S)$ celui de Nowicki et Smutnicki. Si S' est un ordonnancement de $N_{vl}(S) \backslash N_{ns}(S)$ alors $C_{max}(S') \geq C_{max}(S)$.*

Démonstration (issue de [NS96a])

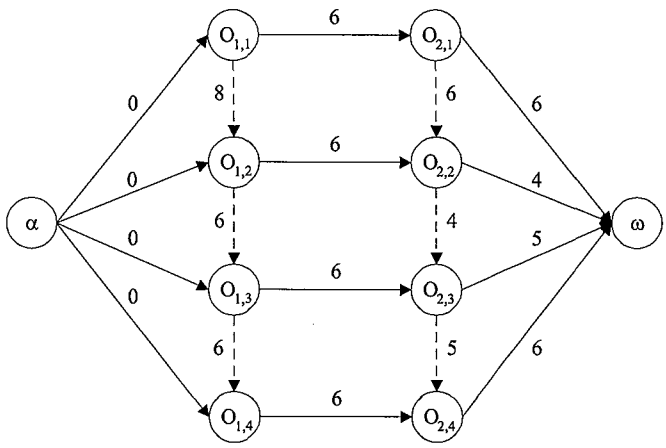


FIG. 4.2 – Graphe disjonctif arbitré $G(S)$

Soit $P = (\alpha, u_1, u_2, ..., u_w, \omega)$ le chemin critique de $G(S) = (V, C \cup D)$ qui génère les blocs $B_1, .., B_r$ et le voisinage $N_{ns}(S)$. Considérons $S' \in N_{vl}(S) \backslash N_{ns}(S)$. Notons (x, y) le couple d'opérations successives sur la ressource M transposées pour obtenir S'.

1er cas : $x \notin P$; $y \notin P$
La transposition de x et y n'affecte pas les arcs du chemin P qui continue à être un chemin de $G(S')$ mais plus nécessairement critique. Nous avons donc $C_{max}(S') \geq C_{max}(S)$.

2ème cas : $x \in P$; $y \notin P$
Soit $x = u_k$ $(1 \leq k \leq w)$. Si $k = w$, $G(S)$ contient donc l'arc positif (x, y), le chemin P allant de la racine à x puis encore de x à y a une longueur strictement supérieure à P ce qui contredit le fait que P soit un chemin critique, donc nécessairement $k < w$.
Comme $y \notin P$ et que y est consécutive à x sur la ressource M, u_{k+1} ne se trouve pas sur M et nous avons forcément l'arc (u_k, u_{k+1}) qui est un arc de C.

Si $k = 1$, (u_k, u_{k+1}) étant un arc de C, $G(S')$ contient un chemin constitué de l'arc (y, x) plus le chemin P allant de x au puits et qui a une longueur strictement supérieure à celle de P. Nous avons donc $C_{max}(S') \geq C_{max}(S)$.
Si $1 < k < w$ et si la ressource de u_{k-1} est M, l'arc (u_{k-1}, y) de $G(S')$ est un arc disjonctif et G(S') contient le chemin $P' = (\alpha, u_1, u_2, .., u_{k-1}, y, u_k, .., u_w, \omega)$ de longueur strictement supérieure à celle de P. Nous avons donc également $C_{max}(S') \geq C_{max}(S)$.
Si $1 < k < w$ et si la ressource de u_{k-1} est différente de M, l'arc (u_{k-1}, u_k) est un arc conjonctif de C et $G(S')$ contient le chemin P. Nous avons donc $C_{max}(S') \geq C_{max}(S)$.

3ème cas : $x \notin P$; $y \in P$
Cas symétrique au 2ème cas.

4ème cas : $x \in P$; $y \in P$

Soit $x = u_k$ $(1 \leq k < w)$. Nécessairement $y = u_{k+1}$. Les opérations x et y appartiennent au bloc B_e $(1 \leq e \leq r)$.
Si $1 < e < r$ et si S' est de makespan meilleur que S, en vertu du théorème 4.1, nécessairement une opération d'un bloc critique de S doit être exécutée avant (resp. après) la première (resp. dernière) opération du bloc. Comme la seule transposition effectuée est celle de x et y, cela suppose que y est exécutée avant la première opération du bloc qui est forcément x. Or nous avons $S' \in N_{vl}(S) \backslash N_{ns}(S)$ ce qui exclu que x et y soient les deux premières opérations du bloc critique B_e. Nous avons une contradiction ce qui implique que soit $C_{max}(S') \geq C_{max}(S)$, soit $C_{max}(S') < C_{max}(S)$ et $e = 1$ (le cas $e = r$ est impossible puisque $S' \in N_{vl}(S) \backslash N_{ns}(S)$).
Si $e = 1$, d'après ce qui a été dit précédemment, pour améliorer la solution S il faut que x et y soient les deux premières opérations de B_1. Dans ce cas $G(S')$ contient le chemin $P' = (\alpha, y = u_2, x = u_1, ..., u_w, \omega)$ de longueur au moins égale à celle de P. □

L'algorithme tabou de Nowicki et Smutnicki [NS96a] fait partie de l'état de l'art et son efficacité pour les problèmes de jobshop a été mainte fois reconnue [JM99] [JM00] [BA03]. Même face à des algorithmes mémétiques, celui-ci se révèle être un concurrent redoutable [GMR02]. Un autre voisinage, plus spécifique aux problèmes de flowshop de permutation, a également été proposé par Nowicki et Smutnicki [NS96b]. De manière simplifiée, l'opérateur de transition consiste à translater un travail d'un bloc critique de sa position originelle à une nouvelle position dans le bloc critique suivant ou le bloc critique précédent. La méthode tabou qui en résulte a également été soulignée pour son efficacité [WBWH02], et le voisinage a été utilisé dans d'autres méthodes [Ree98] [YR98]. Ce dernier voisinage n'a cependant pas retenu notre attention. Nous l'avons jugé trop spécifique et surtout de taille trop importante.

4.2.2 Avec écarts maximaux

Au vu des résultats obtenus, on comprend donc aisément l'importance des théorèmes précédents. Il convient donc de vérifier si ces théorèmes sur les blocs critiques restent valables en présence de contraintes d'écart maximal et, le cas échéant, d'adapter les théorèmes.

Jusqu'à présent nous ne nous sommes intéressés dans ce chapitre qu'aux écarts minimaux et non aux écarts maximaux. Ceux-ci impactent directement la validité d'une solution. En effet, une solution respectant toutes les contraintes d'écart minimal n'est pas forcément une solution valide au regard des contraintes d'écart maximal. Le théorème 4.1 reste valide en présence d'écarts maximaux. Hurink et Keuchel [HK01a] étudient le cas d'une seule ressource. Les auteurs énoncent et démontrent un théorème analogue à 4.1 mais pour le passage d'une solution non valide à une solution valide. Pour cela ils utilisent une notion de bloc légèrement différente en postulant qu'un bloc est une succession d'opérations d'un cycle du graphe tel que les arcs joignant les opérations de ce bloc ne soient pondérés que

par des temps opératoires et seuls les deux arcs reliant le bloc au reste du cycle soient affectés d'écarts maximaux ou d'écarts minimaux mais supérieurs aux temps opératoires. Ils démontrent ainsi qu'il est possible de passer d'une solution non valide à une solution valide à condition qu'au moins une opération d'un bloc soit exécutée avant (resp. après) toutes les autres opérations de ce bloc. Nous pouvons ainsi énoncer un théorème analogue pour le jobshop tout en conservant la notion de bloc critique classique car on peut considérer qu'un circuit de longueur strictement positive appartient à un chemin critique de longueur arbitrairement grande, et le bloc de Hurink et Keuchel devient un bloc critique.

Théorème 4.3 *Soit S un ordonnancement invalide. S'il existe un ordonnancement valide S' alors une opération d'un bloc critique de S différente de la première (resp. dernière) doit être traitée avant (resp. après) toutes les autres opérations de ce bloc.*

Démonstration
La démonstration est rigoureusement analogue à celle du théorème 4.1. □

Les deux théorèmes 4.1 et 4.3 nous offrent donc un voisinage unique permettant à la fois la recherche d'un ordonnancement valide et optimal. Il nous faut encore nous persuader qu'en utilisant un tel voisinage il est effectivement possible d'atteindre l'optimum i.e. que le voisinage utilisé rend l'espace des solutions connexe. Toujours dans [HK01a], il est prouvé que l'utilisation des opérateurs O1 et O2 rend effectivement l'espace de recherche connexe.

Démonstration (adaptée de [HK01a])
Soit S_{op} un ordonnancement optimal. Sans perte de généralité on peut introduire une contrainte d'écart maximal entre α et ω telle que $S_\omega - L(S_{op}) \leq S_\alpha$. La recherche de l'optimum revient à rechercher une solution valide en appliquant les deux opérateurs de recherche (ce qui prouve bien que la recherche d'un optimum et d'une solution valide utilise le même voisinage). Remarquons que tant qu'une solution n'est pas valide, elle contient des blocs critiques. Soit S un ordonnancement non valide. D'après le théorème 4.3, il existe une opération j d'un bloc critique qui doit être ordonnancée dans S_{op} avant ou après toutes les autres opérations de ce bloc. On déplace cette opération à sa nouvelle position. Si le nouvel ordonnancement ainsi obtenu n'est pas valide on continue en sachant que j va toujours conserver la même position relativement aux autres opérations du bloc. Les opérations et les ressources étant limitées, le nombre d'application des deux opérateurs ne peut pas être infini et après un nombre fini d'itérations, on aboutira à une solution valide donc à l'optimum. □

Le résultat de Nowicki et Smutnicki [NS96a] reste également valable à condition de modifier l'énoncé. En effet, en l'absence d'écarts maximaux, permuter un arc disjonctif conduit

encore à une solution valide, ce qui n'est plus le cas en présence de contraintes d'écart maximal. Ainsi, pour les problèmes de flowshop sans attente, le voisinage est vide.

Théorème 4.4 *Notons $N_{vl}(S)$ le voisinage d'un ordonnancement S d'après Van Laarhoven et al et $N_{ns}(S)$ celui de Nowicki et Smutnicki.*
Si S' est un ordonnancement valide de $N_{vl}(S) \backslash N_{ns}(S)$ alors $C_{max}(S') \geq C_{max}(S)$.

4.3 Algorithme Génétique et Recherche Locale

Les algorithmes mémétiques constituent une hybridation entre les algorithmes génétiques classiques et les algorithmes de recherche locale. L'idée consiste à rajouter un opérateur de recherche locale permettant de remédier quelque peu aux insuffisances des algorithmes génétiques en la matière. On en retrouve trace dès 1989 dans le rapport technique de Moscato [Mos89]. De telles approches sont encore dénommées «hybrid genetic algorithms», «genetic local searchers», «baldwinian algorithms», «lamarkian algorithms»... Pour une introduction, on pourra se référer à l'article de Moscato [Mos99].
L'algorithme 4.1 présente une telle hybridation. A l'issue de la phase d'épuration, une procédure d'amélioration locale est appliquée aux individus de la population avec une certaine probabilité. En effet, même si on utilise le voisinage réduit de Nowicki et Smutnicki, celui reste encore conséquent et appliquer la recherche locale à tous les individus requiert trop de temps CPU. De plus, la méthode d'optimisation locale accroît fortement la convergence de la population, ne l'appliquer qu'avec une probabilité donnée permet de réduire cette convergence prématurée et l'uniformité de la population.

De nombreuses variantes existent selon qu'on applique la phase de recherche locale à l'issue du croisement et de la mutation [HM99], après croisement et avant mutation [FM96]... De même, il est possible d'appliquer la recherche locale à la population entière comme le suggère Krasnogor et Smith [KS00], ou plutôt aux enfants qui viennent d'être créés comme le font França et al [FMM01].

4.4 Expériences

4.4.1 Approche retenue

Nous allons calculer un chemin critique, déterminer l'ensemble des blocs critiques et utiliser le voisinage de Nowicki et Smutnicki, Les résultats de dominance des flowshops de permutation nous ont incités, dans l'algorithme génétique, à utiliser des chromosomes qui donnent l'ordre des travaux, sachant que le générateur de placement, qui transforme les ordres des travaux en solution, a la possibilité de concevoir des solutions qui ne sont pas

Algorithme 4.1 Algorithme mémétique «classique»

1. Générer la population initiale P_0 de Q individus
2. Evaluer la "force" des individus de la population P_0
3. Tant que la condition d'arrêt n'est pas satisfaite
   (a) Sélectionner $Q/2$ couples d'individus dans la population P_{k-1}
   (b) Pour chaque couple :
      i. Appliquer l'opérateur de croisement afin d'obtenir les enfants
      ii. Avec une probabilité p_{mut}, appliquer l'opérateur de mutation à chaque enfant
      iii. Evaluer la "force" des enfants
      iv. Ajouter les nouveaux individus à la population P_{k-1}
   (c) Epurer la population pour ne conserver que les Q "meilleurs" individus qui constituent la nouvelle population P_k
   (d) Avec une probabilité p_{loc}, appliquer à chaque individu de la population P_k une procédure d'amélioration par recherche locale

Algorithme 4.2 Algorithme mémétique modifié

1. Générer la population initiale P_0 de Q individus
2. Evaluer la "force" des individus de la population P_0
3. Tant que la condition d'arrêt n'est pas satisfaite
   (a) Sélectionner $Q/2$ couples d'individus dans la population P_{k-1}
   (b) Pour chaque couple :
      i. Appliquer l'opérateur de croisement afin d'obtenir les enfants
      ii. Avec une probabilité p_{mut}, appliquer l'opérateur de mutation à chaque enfant
      iii. Evaluer la "force" des enfants
      iv. Avec une probabilité p_{loc}, appliquer à chaque enfant une procédure d'amélioration par recherche locale.
      v. Ajouter les nouveaux individus à la population P_{k-1}
   (c) Epurer la population pour ne conserver que les Q "meilleurs" individus qui constituent la nouvelle population P_k

toujours des ordonnancements de permutation. En conséquence, lorsque le voisinage sur les blocs critiques nous conduit à inverser deux opérations, par exemple, $O_{i,4}$ et $O_{i,6}$ sur la ressource M_i, nous permutons en fait dans le chromosome les travaux 4 et 6 avant d'appeler le générateur de solutions. Dans notre exemple, si le chromosome est $(5,1,4,6,3,2)$ et si l'un des mouvements consiste à permuter $O_{i,4}$ et $O_{i,6}$ sur la ressource M_i, nous considérons le chromosome $(5,1,6,4,3,2)$ comme appartenant au voisinage du chromosome $(5,1,4,6,3,2)$.

Nous avons retenu l'approche de França et al [FMM01] exposée dans l'algorithme 4.2. Celle-ci diffère de l'approche classique dans le sens où la recherche locale est appliquée à l'issue du croisement et de la mutation, l'idée étant de remplacer, avec une certaine probabilité, chaque enfant par un optimum local.

L'algorithme d'optimisation que nous avons retenu est une méthode de plus forte pente (algorithme 4.3). Cette approche se justifie du fait du voisinage réduit utilisé. De plus, nous ne cherchons pas à développer un algorithme performant qui va balayer l'espace des solutions à la recherche de l'optimum global mais obtenir un effet de loupe autour de chaque solution. Pour être efficace, un algorithme de recuit simulé ou tabou doit intégrer de nombreux artifices et nécessite le réglage de nombreux paramètres pour pouvoir s'extraire d'optimaux locaux ou de plateaux. Cette efficacité est au détriment du coût. Il nous semble plus intéressant d'appliquer un tel algorithme lors d'une phase finale de recherche, sur la ou les meilleures solutions trouvées.

Algorithme 4.3 Algorithme de plus forte pente

1. Soit S la solution courante. Flag=Vrai.
2. Tant que Flag=Vrai
   (a) Construire le voisinage $N_{ns}(S)$ de S
   (b) Flag=Faux ; $S_{opt} = S$.
   (c) Pour chaque $S' \in N_{ns}(S)$
      i. Evaluer la "force" de S'
      ii. Si $C_{max}(S') < C_{max}(S_{opt})$ Alors Flag=Vrai et $S_{opt} = S'$
   (d) $S = S_{opt}$

4.4.2 Méthode exacte de Fondrevelle ([Fon03])

4.4.2.1 Principe

Fondrevelle a présenté en 2003 une Procédure par Séparation et Evaluation (PSE) permettant de résoudre de manière optimale le problème de flowshop de permutation en présence

N°	mxn	Durées hétéro.	Charges hétéro.	λ (%)	Ecarts mach.	Ecarts jobs
1	5x12	Non	Non	10	Tous	Tous
2	5x12	Non	1-3-5	10	Tous	Tous
3	5x12	Non	Non	10	Tous	1-2-3-4
4	5x12	Non	Non		Aucun	Aucun
5	5x12	Non	1-3-5	10	2-4	Tous
6	5x12	Non	1-3-5	10	1-3	Tous
7	5x12	Oui	Non	10	Tous	Tous
8	5x12	Non	Non	5	Tous	Tous
9	10x12	Non	Non	10	Tous	Tous
10	5x12	Oui	1-3-5	10	2-3	1-2-3-7-8-9

TAB. 4.2 – Classes d'instances Fondrevelle 2003

de contraintes d'écart maximal entre les opérations successives d'un même travail. Sa PSE construit progressivement des solutions en rajoutant à chaque étape un travail à une séquence partielle. Les noeuds de l'arborescence correspondent à une séquence partielle de travaux. A partir d'un noeud, on construit les fils en rajoutant des travaux ne figurant pas encore dans la séquence partielle. Les feuilles correspondent quant à elles aux permutations complètes. Le cheminement choisi par la PSE est une exploration en profondeur d'abord. Fondrevelle a développé plusieurs bornes inférieures pour couper les branches de l'arbre. Pour de plus amples détails, nous vous invitons à consulter l'article [Fon03].

4.4.2.2 Jeux d'essai

Pour tester sa méthode de résolution, Fondrevelle [Fon03] a construit plusieurs familles de jeux d'essai pour tenir compte de multiples situations (durées homogènes ou non, ressources goulots..). Le tableau 4.2 résume les caractéristiques des 10 familles de jeux d'essai. Chaque famille comporte 10 jeux d'essai.
La colonne «Durées hétéro.» indique de quelle manière ont été générés les temps opératoires : Non signifie que les durées ont été choisies aléatoirement dans l'intervalle $[20; 50]$; Oui signifie que la moitié des jobs ont des temps opératoires choisis aléatoirement dans l'intervalle $[20; 50]$ et l'autre moitié des temps opératoires choisi aléatoirement dans l'intervalle $[20; 100]$.
La colonne «Charges hétéro.» correspond aux charges machine. Non signifie que celles ci sont homogènes. Dans le cas contraire, une partie des machines est moins chargée et les temps opératoires sur les machines dont le numéro figure dans cette colonne sont augmentés de 75%.
La colonne «λ (%)» permet de calculer les temps d'attente maximaux autorisés. Ceux-ci sont choisis aléatoirement dans l'intervalle $[0; 70\lambda]$. Pour $\lambda = 5$ par exemple, l'intervalle considéré est $[0; 70 \times 0.05]$.

Jeu N°	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	576	87697	7831	590	587	586
2	553	83245	7531	558	554	553
3	576	11700	7753	587	577	576
4	622	95542	8294	626	623	622
5	570	40327	7570	576	572	570
6	562	13146	7297	563	562	562
7	599	3328	7577	601	599	599
8	586	116968	7868	591	589	588
9	599	24178	6832	602	599	599
10	594	18776	7082	598	596	594

TAB. 4.3 – Résultats détaillés pour les 10 instances de la famille Fondrevelle 2003 N°1

Les colonnes «Ecarts mach.» et «Ecarts jobs» indiquent pour quelles machines et quels travaux ces contraintes d'écart maximal sont définies. Dans la colonne «Ecarts mach.», 2-4 signifie qu'il existe une contrainte d'écart maximal à la sortie des machines 2 et 4, c'est à dire entre les opérations succesives utilisant les resources 2 et 3 et entre les opérations succesives utilisant les resource 4 et 5. A noter que contrairement au reste de cette thèse, Fondrevelle a défini ces temps d'attentes maximales entre la fin d'une opération et le début de la suivante et non de début à début comme nous.

4.4.2.3 Résultats numériques

Nous avons exécuté l'algorithme 4.2 avec les paramètres suivants :
- $p_{mut} = p_{loc} = 0.1$;
- une population de 100 individus ;
- une évolution de 50 cycles.

Cet algorithme a été exécuté 10 fois sur chaque instance de chacun des 10 jeux d'essai de chaque famille. Nous avons retenue pour chaque instance de chaque famille le meilleur résultat, le moins bon et le score moyen. Le tableau A.4.1 détaille les résultats obtenus pour la première famille. La colonne C_{max} est l'optimum obtenu par la méthode de Fondrevelle sur le sous-ensemble des flowshop de permutation. La colonne «Noeuds» représente le nombre de noeuds explorés par sa méthode pour atteindre l'optimum. La colonne «Calculs» est le nombre moyen d'évaluations effectuées par notre méthode au cours des 10 tests sur chaque jeu d'essai. Une permutation de deux travaux ne nécessite pas de reconstruire toute la solution, cependant nous avons comptabilisé comme évaluation même les reconstructions partielles. Les colonnes «Pire», «Moyen» et «Meilleur» sont respectivement le pire score, le score moyen et le meilleur score obtenus par notre méthode au cours des 10 tests.

L'ensemble des résultats détaillés figurent en annexe (tableaux A.4.1, A.4.2, A.4.3, A.4.4, A.4.5, A.4.6, A.4.7, A.4.8, A.4.9, A.4.10). Le tableau 4.4 synthétise les résultats pour l'ensemble des jeux d'essai. La colonne «Calculs» représente toujours le nombre moyens de calculs effectués par notre méthode pour chaque famille. L'«écart moyen à l'optimum» exprime, comme son nom l'indique, l'écart moyen en % du makespan à l'optimum sur l'ensemble des essais. La colonne «Optimum» indique le nombre de jeux d'essai par famille pour lesquels notre méthode a trouvé ou amélioré l'optimum (notre méthode ne se bornant pas uniquement aux flowshops de permutation). Le nombre de jeux d'essai où notre méthode a permis d'améliorer le résultat de Fondrevelle est cependant restreint compte tenu des résultats de dominance sur les flowshops de permutation. Ceci est le cas pour la famille n°10 où nous avons trouvé l'optimum pour 7 jeux sur 10 et amélioré le C_{max} pour les 3 autres jeux. Ceci est lié à la nature du jeu d'essai dont les temps opératoires et les charges sont hétérogènes et les écarts maximaux ne sont définis que pour les ressources 2 et 3 et les travaux 1-2-3-7-8-9. Dans l'ensemble, les résultats sont très encourageants. Non seulement nous avons trouvé l'optimum sur le sous-ensemble des flowshops de permutation dans 93 des cas sur 100 au cours de nos tests, mais de plus, nous nous approchons de cet optimum à moins de 1% en moyenne.

Famille N°	Calculs	écart moyen à l'optimum	Optimum
1	7564	0,36%	8
2	6360	0,47%	10
3	6246	0,28%	7
4	4689	0,07%	10
5	4215	0,02%	10
6	4938	0,06%	10
7	8053	0,56%	9
8	7985	0,36%	10
9	8614	0,27%	9
10	8025	0,35%	10

TAB. 4.4 – Résultats sur jeux d'essai Fondrevelle 2003

Nous avons tenté de comparer notre méthode avec celle de Fondrevelle sur les jeux d'essai utilisés au cours du chapitre 3. Malheureusement, la PSE n'a pas pu générer d'optimum en temps raisonnable. Fondrevelle a développé une heuristique NEH-1, dérivée de la méthode NEH, mais n'incorporant que les temps d'attente maximaux. Cette heuristique s'est montrée particulièrement performante sur ses jeux d'essais puisque en moyenne, NEH-1 permettait d'obtenir une solution approchée à moins de 1.5% en moyenne de l'optimum. Pour les besoins de nos jeux d'essais, Fondrevelle a généralisé NEH-1 pour y incorporer également les temps d'attente minimaux. Nous avons pu comparer notre méthode avec

Jeux	Borne inf.	Borne sup.	Calculs	Meilleur res.	Moyen
1	1232	1278	5729	1290 (0,93%)	1295 (1,31%)
2	1290	1359	5835	1365 (0,44%)	1368 (0,66%)
3	1073	1081	6297	1089 (0,73%)	1099 (1,64%)
4	1268	1293	8139	1309 (1,22%)	1319 (1,97%)
5	1198	1236	5265	1250 (1,12%)	1261 (1,98%)
6	1180	1195	6394	1195 (0,00%)	1216 (1,73%)
7	1226	1239	6993	1251 (0,96%)	1252 (1,04%)
8	1170	1206	7866	1206 (0,00%)	1220 (1,15%)
9	1206	1230	7437	1241 (0,89%)	1254 (1,91%)
10	1082	1108	7836	1111 (0,27%)	1131 (2,03%)

TAB. 4.5 – Résultats sur jeux d'essai Taillard 5x20

ses deux heuristiques sur les jeux d'essai du chapitre 3. Ce coup-ci, les heuristiques se sont montrées particulièrement décevantes puisque notre méthode a permis d'obtenir des solutions meilleures que les deux heuristiques de plus de 24% en moyenne sur les 50 jeux d'essai de taille 5x20.

4.4.3 Expériences sur les exemples de Taillard 5x20 et 10x20

Notre méthode se voulant la plus générique possible, nous avons également appliqué l'algorithme 4.2 avec les mêmes paramètres que précédemment, sur les jeux d'essai de Taillard de taille 5x20 et 10x20, ces derniers ne comportant pas d'écarts maximaux. De plus, Taillard n'ayant considéré que des flowshops de permutation, nous «bridons» volontairement notre algorithme pour ne travailler que sur cet espace. Les résultats figurent dans les tableaux 4.5 et 4.6. La colonne «Borne inf.» constitue la borne inférieure théorique de l'optimum sur le sous-ensemble des flowshops de permutation. La colonne «Borne sup.» indique quant à elle le meilleur résultat trouvé jusqu'à présent sur les flowshops de permutation (données actualisées en septembre 2003). La colonne «Calculs» nous renseigne toujours sur le nombre moyen d'évaluations que nous avons effectuées. Les colonnes «Meilleur res.» et «Moyen» sont respectivement le meilleur résultat trouvé par notre méthode et le résultats moyen au cours des 10 essais pour chaque jeu (entre parenthèse l'écart en % à la borne supérieure). Là aussi notre méthode se montre à la hauteur puisque nous parvenons en moyenne à moins de 2% du meilleur résultat connu pour les jeux de taille 5x20, et à moins de 3% pour ceux de taille 10x20.

Jeux	Borne inf.	Borne sup.	Calculs	Meilleur res.	Moyen
1	1448	1582	12950	1613 (1,92%)	1625 (2,65%)
2	1479	1659	12784	1687 (1,66%)	1718 (3,43%)
3	1407	1496	13277	1514 (1,19%)	1537 (2,67%)
4	1308	1378	11842	1385 (0,51%)	1412 (2,41%)
5	1325	1419	12631	1426 (0,49%)	1454 (2,41%)
6	1290	1397	12366	1424 (1,90%)	1432 (2,44%)
7	1388	1484	10365	1503 (1,26%)	1524 (2,62%)
8	1363	1538	12221	1566 (1,79%)	1577 (2,47%)
9	1472	1593	11019	1621 (1,73%)	1640 (2,87%)
10	1356	1591	12440	1621 (1,85%)	1635 (2,69%)

TAB. 4.6 – Résultats sur jeux d'essai Taillard 10x20

4.5 Conclusion

Les premiers résultats concernant l'hybridation de notre algorithme génétique avec une méthode de recherche locale sont très encourageants et nous laissent bon espoir d'améliorer encore les performances. La population initiale est pour l'instant générée aléatoirement, une amélioration notable peut être obtenue en y incorporant des heuristiques performantes issues du flowshop classique ou adaptées à notre problématique comme celles développées par Fondrevelle. De même, l'algorithme de recherche locale est un simple algorithme de descente qui peut lui aussi gagner en performances en implémentant un algorithme de recuit-simulé ou tabou. Ce type d'algorithme étant cependant plus coûteux, il ne sera appliqué que lors d'une phase finale de recherche sur la ou les meilleures solutions.

Conclusion et perspectives

1 Conclusion

Dans le premier chapitre nous avons étudié divers cas industriels et mis en évidence le fait que les contraintes temporelles entre opérations, et plus particulièrement les contraintes d'écart maximal, ne se cantonnent pas exclusivement aux secteurs pharmaco-chimiques ou agro-alimentaires mais sont largement répandues dans des secteurs aussi divers que la mécanique ou la formation continue. L'étude bibliographique a fait ressortir un intérêt croissant pour ce type de problématique depuis le début des années 90, essentiellement en ordonnancement de projet, mais les publications restent encore marginales. En particulier, les problèmes d'atelier de type jobshop ou flowshop n'ont pas connu de grande attention. La principale raison pouvant être avancée pour ce manque d'intérêt réside dans le fait que construire une solution valide dans le cas général est en soi un problème NP-complet. Ce n'est qu'avec les progrès technologiques de l'informatique et des ordinateurs qu'une attention plus soutenue a pu être portée à ces contraintes.

La plupart des méthodes de résolution d'un problème comportant des écarts maximaux combinent à la fois recherche de l'optimum et recherche d'une solution valide, avec le risque de ne pas trouver de solution du tout. Ces méthodes se justifient dans des cas extrêmes, avec la présence de nombreuses contraintes de potentiels entre travaux différents. Les études de cas menées au chapitre 1 ont cependant fait ressortir que dans les situations «standards», les écarts maximaux sont localisés au niveau de quelques opérations et le plus souvent au sein du même travail. Ces observations nous ont incités à étudier, dans le second chapitre, des généralisations d'algorithme de construction à base de règle de priorité ou de liste d'ordre strict, algorithmes moins lourds à mettre en oeuvre que des PSE par exemple. Après avoir mis en évidence des phénomènes de type «saute-mouton» qui dégradent fortement les solutions, nous avons étudié une amélioration des algorithmes en utilisant la décomposition naturelle des opérations selon les composantes fortement connexes du graphe potentiel-tâche. Les placements ne s'effectuent plus opérations par opérations mais paquets d'opérations par paquets d'opérations que nous avons baptisés grappes. Nous avons démontré que sous des hypothèses assez réalistes, nos algorithmes généralisés permettent de construire des ordonnancements actifs.

Les études de cas ayant fait ressortir essentiellement des organisations de type flowshop,

nous nous sommes particulièrement attachés à étudier ce type d'atelier dans le chapitre 3. Notre objectif a été d'optimiser le critère du C_{max} en présence de contraintes d'écart maximal. Pour ce faire nous avons utilisé une approche à base d'algorithmes génétiques. Nous avons étudié plusieurs opérateurs de croisement pour déterminer lequel était le plus adapté à notre problématique. Cette étude ayant fait ressortir l'absence d'un tel opérateur, nous avons développé une méthode d'autodétermination qui sélectionne automatiquement, dans un pool d'opérateurs de référence, celui qui est le plus adapté à l'instance du problème et à l'état courant de la population de l'algorithme génétique.

Pour rendre notre approche plus performante et accroître la finesse de recherche de notre algorithme génétique, nous avons étudié la possibilité de compléter l'approche génétique par l'insertion d'un algorithme de recherche locale. Le chapitre 4 a consisté à exposer les principes des algorithmes mémétiques. Nous avons analysé plusieurs types de voisinages classiques ainsi que leurs extensions en présence d'écarts maximaux. Nous avons ensuite implémenté un algorithme mémétique dont la méthode de recherche locale à base d'un algorithme de plus forte pente utilise le voisinage de Nowicki et Smutniki. Cette approche a permis d'obtenir d'excellents résultats sur les benchmarks développés par Fondrevelle et des résultats prometteurs sur ceux de Taillard.

Nous ne pouvons terminer cette conclusion sans évoquer les retombées pour l'entreprise d'une telle thèse et nous poser la question de l'atteinte des objectifs. L'objectif principal qui nous avait été fixé était l'amélioration du moteur d'Incoplan pour gérer plus efficacement les contraintes d'écart maximal. A ce niveau, je considère que l'objectif a été atteint. En quittant la société Incotec, notre moteur était totalement intégré au logiciel. J'ai eu l'opportunité d'effectuer plusieurs maquettes en utilisant l'algorithme ARP-G. En particulier, ARP-G-RP a été présenté à la société PharmaCorp après avoir intégré des contraintes supplémentaires comme le blocage de ressources, contraintes qui n'ont pas fait l'objet d'une étude particulière dans cette thèse. Couplé à notre algorithme génétique, nous avons pu obtenir le planning type élaboré en deux ans par la société PharmaCorp. Complété par un algorithme de recherche locale non présenté dans cette thèse, la solution a pu encore être optimisée de 3.6%. Nous avons ainsi pu faire la preuve que l'approche retenue était valide.

2 Perspectives

Même si nous avons fait la preuve de la validité de notre approche, il nous reste encore de nombreuses améliorations à apporter en terme de performances. Pour la construction d'une solution faisable, la définition des grappes nécessite plus de souplesse, surtout en présence de faibles tensions sur les écarts maximaux. Nous envisageons de profiter de la rapidité de nos algorithmes pour construire plusieurs solutions en utilisant des tailles de grappes différentes et en faisant croître ces dernières dynamiquement. Initialement

les grappes ne sont constituées que d'opérations élémentaires ou de couples d'opérations soumis à une contrainte de type sans attente. Ces grappes sont placées une à une en utilisant l'algorithme ARP-G ou AOS-G. Après le placement d'une grappe, on teste la validité des contraintes d'écart maximal entre les opérations de la grappe et les opérations de grappes précédemment placées. Si une telle contrainte est violée, on dépile les grappes et on regroupe en une nouvelle grappe les grappes qui ne respectaient pas les contraintes d'écart maximal entre elles. De cette manière, les grappes vont croître petit à petit jusqu'à atteindre une taille «critique» qui permettra d'obtenir une solution valide. En présence de faibles tensions sur les écarts maximaux, la taille des grappes sera réduite alors qu'avec des contraintes très fortes, les grappes atteindront leur taille maximale.

Si les premiers résultats numériques que nous avons obtenus sont plus qu'encourageants, nous pouvons et devons optimiser notre approche génétique en incorporant dans la population initiale des heuristiques plus performantes en adaptant celle du flowshop classique. De même, la sélection aléatoire des parents mérite d'être revue pour plus d'efficacité. Pour l'instant, la méthode de plus forte pente qui complète notre approche génétique n'est pas des plus performantes. Nous envisageons un second algorithme de type tabou ou de recuit simulé qui serait appliqué lors de la phase finale sur la ou les meilleures solutions.

Notre algorithme génétique a été développé et adapté aux problèmes de type flowshop. Il nous reste à le généraliser au jobshop. Nous envisageons un codage sous la forme d'une liste de priorité, non pas pour les opérations mais pour les grappes, l'algorithme ARP-G se chargeant ensuite de construire la solution correspondant à ces priorités. Notre algorithme mémétique pourra être appliqué directement sans modifications au problème du jobshop puisque nous avons utilisé le voisinage générique de Nowicki et Smutnicki et que nous travaillons sur le codage des solutions et non sur la solution elle-même. Pour plus d'efficacité, nous songeons cependant à développer un algorithme de recherche locale, toujours basé sur le voisinage de Nowicki et Smutnicki, mais qui permutera les deux opérations critiques et non l'ensemble des deux grappes. Les premiers tests effectués avec un algorithme de recuit simulé ont prouvé que les capacités d'optimisation étaient bien meilleures en partant d'une solution générée par un algorithme par grappes que d'une solution quelconque. Pour une solution par grappes, les opérations liées par une contrainte d'écart maximal ont été placées quasi consécutivement, de ce fait la marge libre par rapport aux écarts maximaux est importante et la probabilté de violer une contrainte d'écart maximal en permutant deux opérations est relativement faible. L'algorithme de recuit simulé perdra ainsi moins de temps en tâtonnements et en passant par des solutions non valides.

Annexes

1 Annexe du chapitre 1

La gamme PharmaCorp est présentée sur la figure A.1.1. Les traits pleins représentent les liens de précédence entre les opérations. Les traits pointillés représentent les contraintes d'attente maximale entre opérations et sont valués par le temps maximal pouvant s'écouler entre la fin d'une opération et le début d'une autre.

La figure A.1.2 représente le système de production de PharmaCorp. A chaque opération sont associées une ou plusieurs ressources dont les libellés figurent dans le rectangle en pointillé relié à l'opération (rectangle en traits pleins).

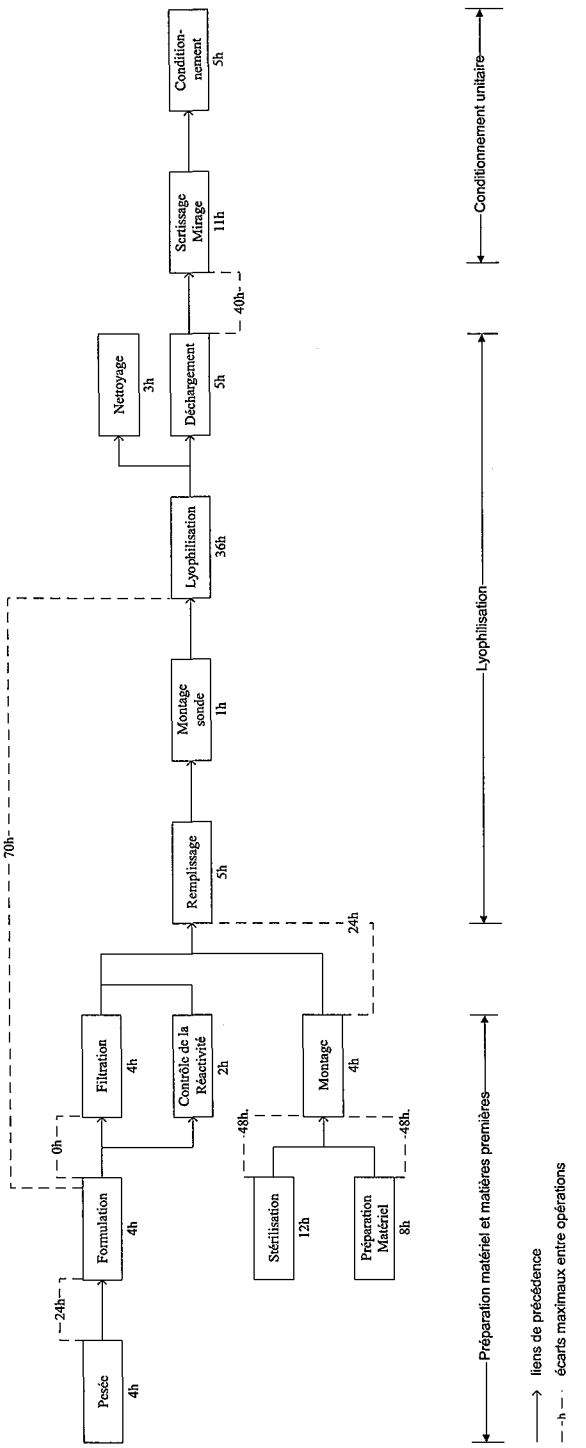


FIG. A.1.1 – Gamme du processus de lyophilisation PharmaCorp

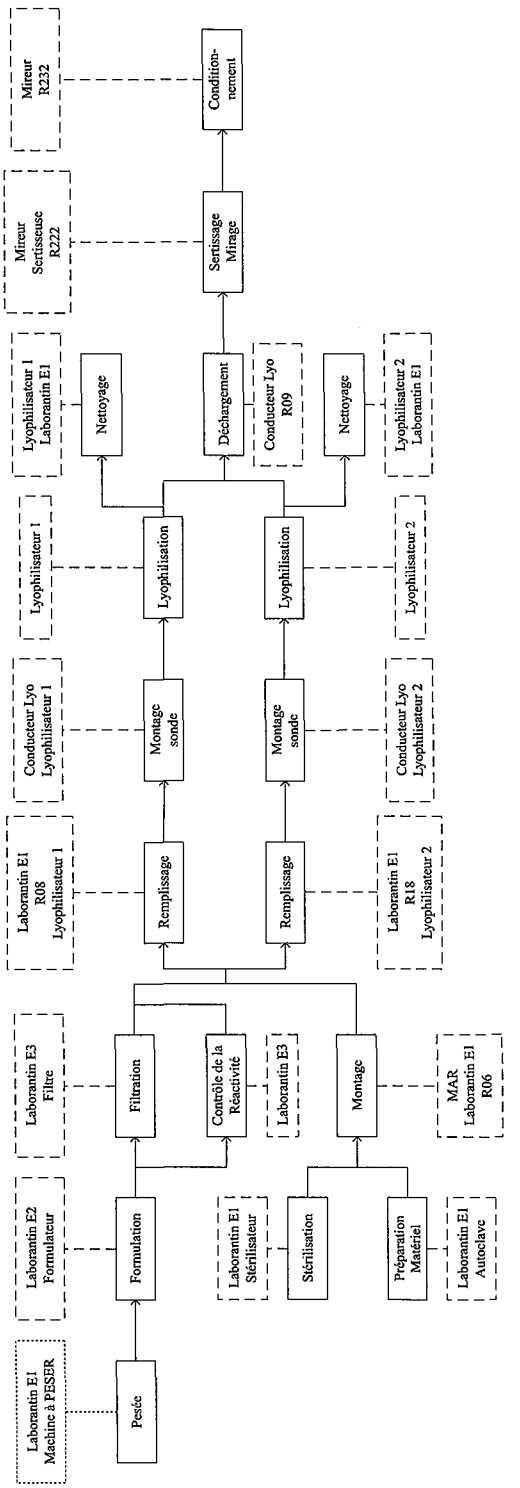


FIG. A.1.2 – Système de production PharmaCorp

2 Annexe du chapitre 2

Dans cette section sont exposés les résultats des expériences menées dans le second chapitre, section 2.4. Les résultats concernent les tests comparatifs entre les algorithmes ARP-G, AOS-G, ARP-MD et AOS-MD pour les familles de jeux d'essai 10x30 (figure A.2.1), 10x20 (figure A.2.2), 10x10 (figure A.2.3), 5x20 (figure A.2.4), 5x10 (figure A.2.5) et 5x5 (figure A.2.6).

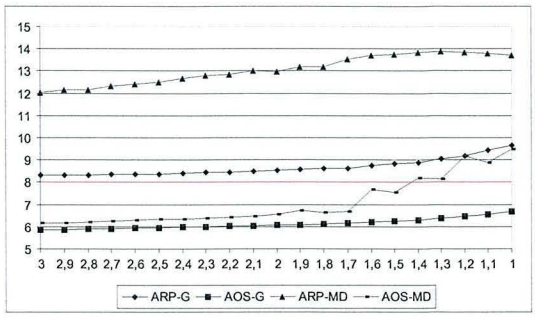
Exemple de la figure A.2.1. Cette figure résume les résultats pour les 30 instances de la famille de jeux d'essai de taille $m = 10$, $n = 30$ dont la valeur moyenne de la mesure d'entrelacement ε est 5.34.

Le premier graphique représente le nombre moyen de calculs de dates d'exécution en fonction de la tension sur les écarts maximaux (à chaque algorithme correspond une courbe) :
abscisse : tension sur les écarts maximaux ;
ordonnée : logarithme népérien du nombre moyen de calculs des dates d'exécution ;
droite horizontale rouge (repérable par une flèche à droite du dessin) : estimateur du nombre de calculs de dates d'exécution $ln(n.m^2)$.

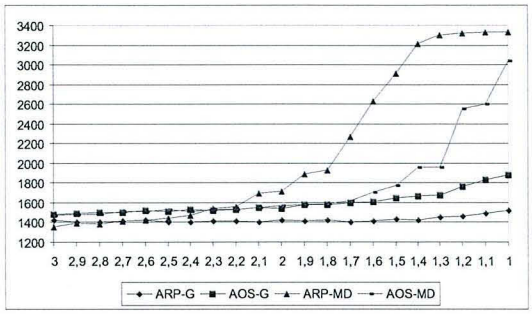
Le second graphique permet de visualiser, pour chaque algorithme, l'évolution du makespan moyen en fonction de la tension sur les écarts maximaux (à chaque algorithme correspond une courbe) :
abscisse : tension sur les écarts maximaux ;
ordonnée : makespan moyen sur les 30 jeux d'essai de la famille 10x30.

La troisième figure représente le nombre de solutions valides obtenues par les algorithmes ARP-MD et AOS-MD en fonction de la tension sur les écarts maximaux sur l'ensemble des 30 jeux d'essai de la famille :
abscisse : tension sur les écarts maximaux ;
ordonnée : nombre de solutions valides obtenues par chacun des algorithmes.
La troisième courbe (*succès cumulés*) est le nombre total de constructions valides ayant été obtenues par l'un ou l'autre algorithme.

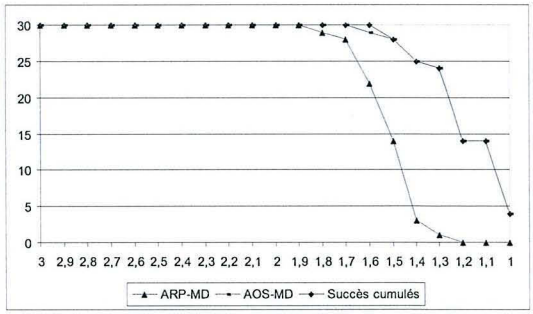
La quatrième figure reprend les mêmes informations que la seconde figure mais en ne tenant compte que des constructions valides.



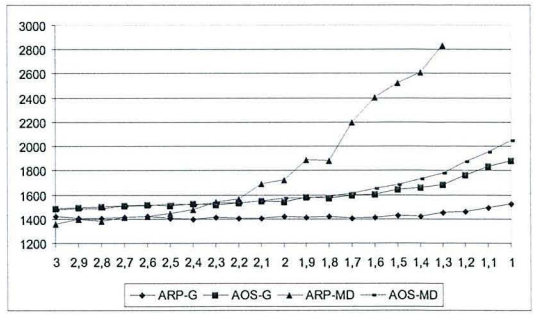
Nombre moyen de calculs de dates d'exécution



Moyenne des makespan

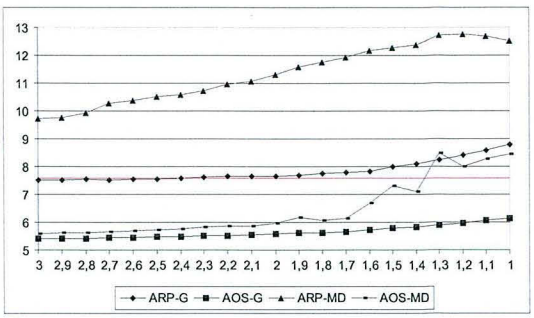


Nombre de constructions valides

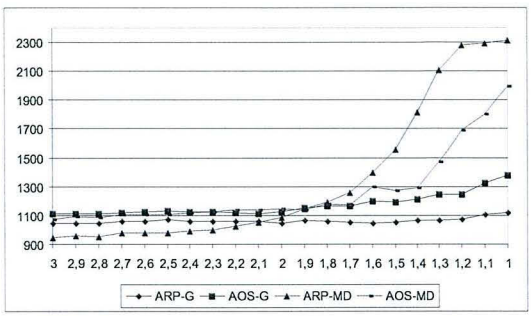


Moyenne des makespan en ne tenant compte que des solutions valides

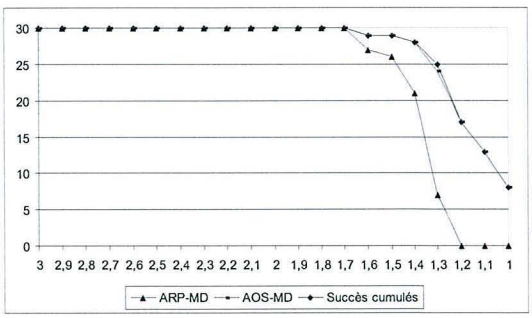
FIG. A.2.1 – Résultats numériques Jobshop m=10 n=30 $\varepsilon = 5.34$



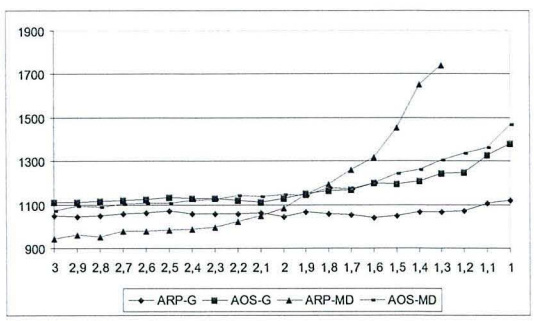
Nombre moyen de calculs de dates d'exécution



Moyenne des makespan

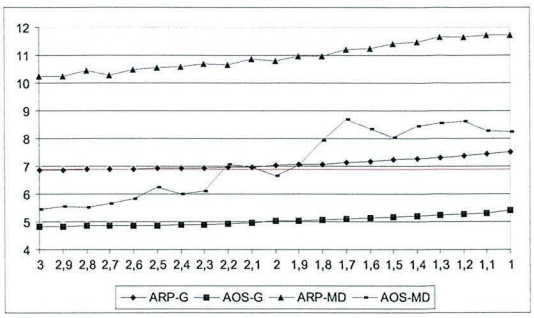


Nombre de constructions valides

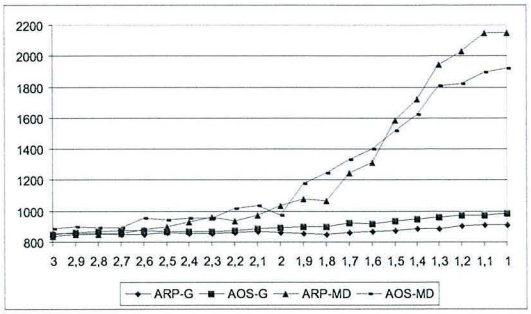


Moyenne des makespan en ne tenant compte que des solutions valides

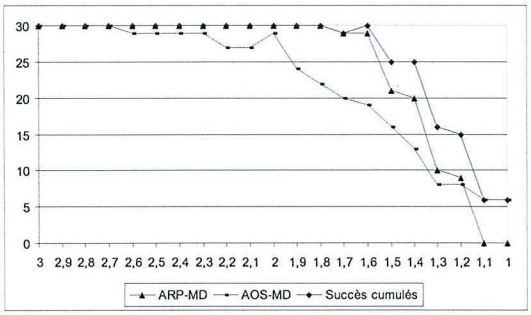
FIG. A.2.2 – Résultats numériques Jobshop m=10 n=20 $\varepsilon = 5.36$



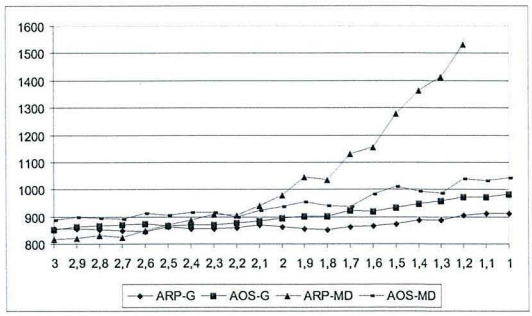
Nombre moyen de calculs de dates d'exécution



Moyenne des makespan

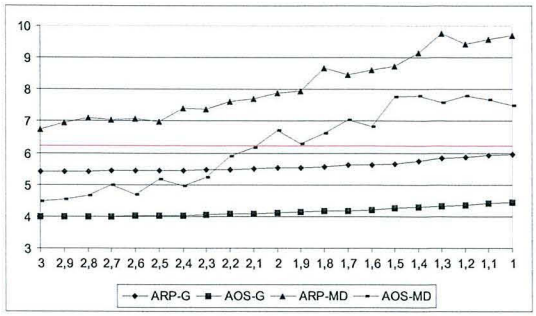


Nombre de constructions valides

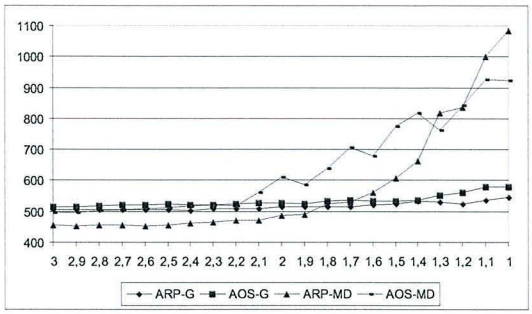


Moyenne des makespan en ne tenant compte que des solutions valides

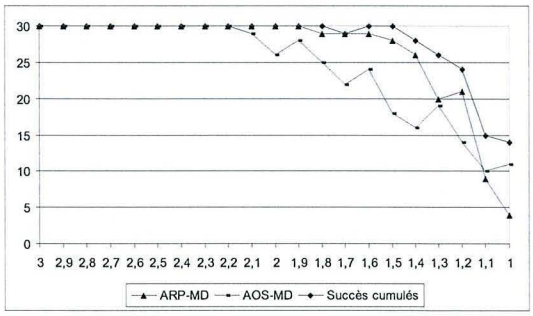
FIG. A.2.3 – Résultats numériques Jobshop m=10 n=10 $\varepsilon = 3.79$



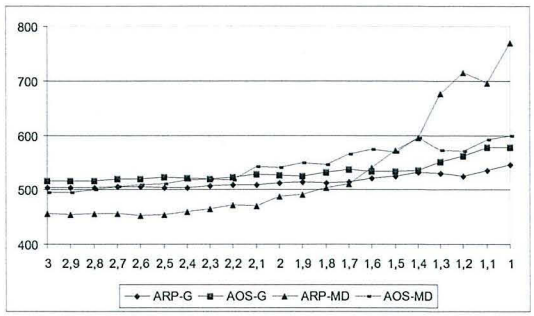
Nombre moyen de calculs de dates d'exécution



Moyenne des makespan

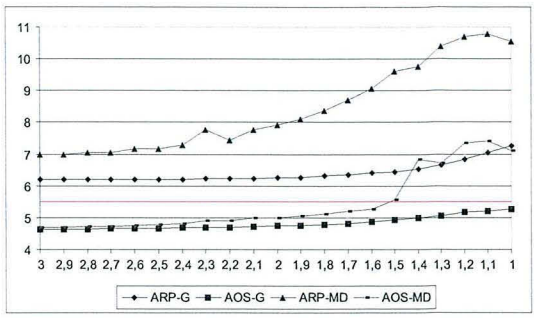


Nombre de constructions valides

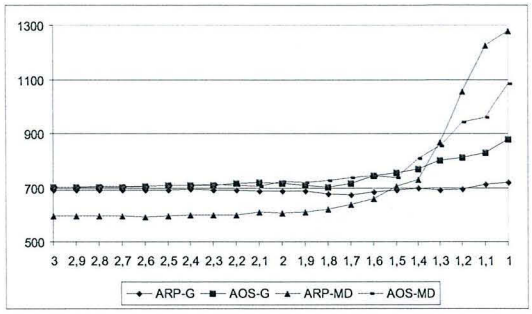


Moyenne des makespan en ne tenant compte que des solutions valides

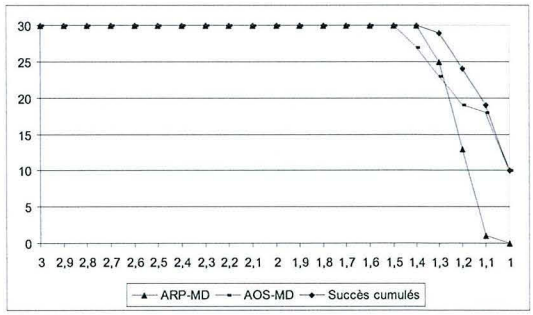
FIG. A.2.4 – Résultats numériques Jobshop m=5 n=20 $\varepsilon = 4.35$



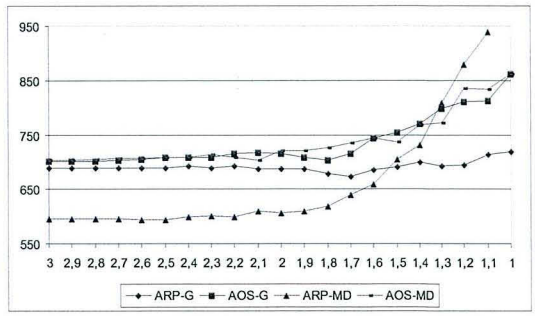
Nombre moyen de calculs de dates d'exécution



Moyenne des makespan

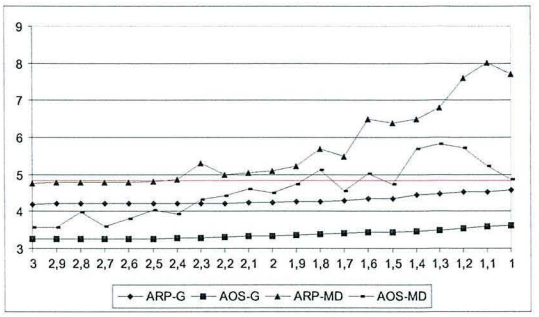


Nombre de constructions valides

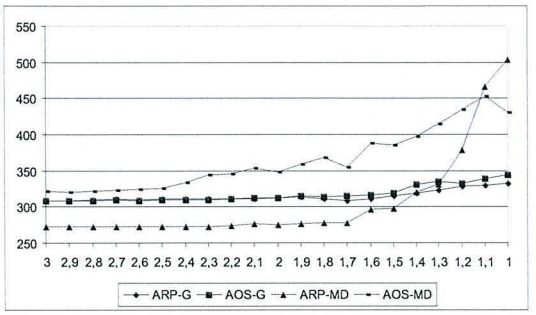


Moyenne des makespan en ne tenant compte que des solutions valides

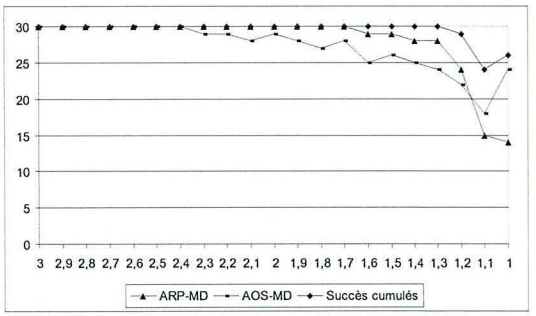
FIG. A.2.5 – Résultats numériques Jobshop m=5 n=10 $\varepsilon = 5.77$



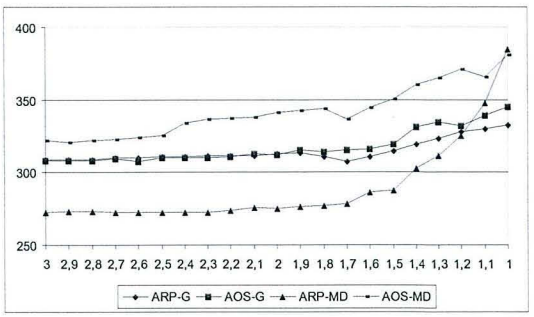
Nombre moyen de calculs de dates d'exécution



Moyenne des makespan



Nombre de constructions valides



Moyenne des makespan en ne tenant compte que des solutions valides

FIG. A.2.6 – Résultats numériques Jobshop m=5 n=5 $\varepsilon = 2.20$

3 Annexe du chapitre 3

3.1 Qualité des opérateurs de croisement - jeux d'essai 10x30

Cette section présente les résultats numériques obtenus sur les jeux d'essai de taille $m = 10$, $n = 30$, pour la mesure de la qualité des différents opérateurs de croisement au cours des expériences décrites dans le chapitre 3, section 3.4.
Le tableau A.3.1 (resp. A.3.2 et A.3.3) contient, pour les jeux d'essai 10x30 sans écarts maximaux (resp. avec écarts maximaux et sans attente), l'évolution des différents compteurs E1, E2, E3, R et Q au cours des cycles pour les opérateurs TSXW, TSX, TSXD, 1X, 2XL, 2XC et ERX.

La figure A.3.1 (resp. A.3.2 et A.3.3) permet de comparer, pour les jeux d'essai sans écarts maximaux (resp. avec écarts maximaux et sans attente), l'évolution du compteur E1 au cours des cycles pour les différents opérateurs de croisement. En ordonnée figure la valeur du compteur E1 (en %), les cycles sont en abscisse.

La figure A.3.4 permet de comparer les valeurs du compteur E1 à l'issue du cycle initial pour les différents types de jeu d'essai, sans écarts maximaux (SPN), avec écarts maximaux (PN) et sans attente (SA). On constate ainsi que les opérateurs TSX et TSXD se montrent de plus en plus performants lorsque les contraintes temporelles deviennent fortes.

La figure A.3.5 (resp. A.3.6 et A.3.7) permet de comparer, pour les jeux d'essai sans écarts maximaux (resp. avec écarts maximaux et sans attente), l'évolution du makespan moyen cours des cycles pour les différents opérateurs de croisement. En ordonnée figure la moyenne des scores du meilleur individu de chacune des 30 instances, les cycles sont en abscisse.

La section 3.2 reprend les mêmes informations pour les jeux d'essai de taille 5x20.

cycle	E1	E2	E3	R	Q
0	31,84	38,76	29,40	4,67	1,56
10	10,39	20,07	69,54	8,95	2,55
20	6,73	18,10	75,16	10,26	3,99
30	5,10	19,80	75,10	11,83	4,44
40	5,10	20,26	74,64	14,38	5,36
50	4,12	24,90	70,98	17,71	8,30

TSXW

cycle	E1	E2	E3	R	Q
0	23,15	36,24	40,61	1,01	1,54
10	9,02	10,98	80,00	2,22	3,20
20	6,41	9,54	84,05	3,73	3,20
30	4,31	10,07	85,62	4,38	3,73
40	3,27	9,48	87,25	4,71	3,92
50	3,07	12,55	84,38	6,93	3,73

TSX

cycle	E1	E2	E3	R	Q
0	24,27	37,90	37,83	0,88	1,63
10	7,58	13,99	78,43	3,07	2,88
20	5,49	14,38	80,13	5,16	4,31
30	5,03	13,92	81,05	6,54	3,73
40	3,79	15,82	80,39	10,13	3,92
50	3,20	16,67	80,13	10,98	4,77

TSXD

cycle	E1	E2	E3	R	Q
0	25,39	55,37	19,24	10,92	1,29
10	19,48	45,03	35,49	16,86	3,79
20	12,61	49,28	38,10	25,29	5,42
30	10,20	53,33	36,47	30,39	8,24
40	6,47	61,63	31,90	38,10	18,69
50	4,25	66,80	28,95	41,24	26,08

1X

cycle	E1	E2	E3	R	Q
0	36,75	46,24	17,02	0,81	1,44
10	21,24	30,92	47,84	1,57	5,29
20	10,65	34,58	54,77	3,59	17,45
30	6,27	46,47	47,25	6,60	36,34
40	4,58	52,09	43,33	9,87	44,31
50	3,07	59,93	36,99	12,42	59,02

2XL

cycle	E1	E2	E3	R	Q
0	37,27	44,37	18,37	1,10	1,71
10	18,24	29,02	52,75	1,57	2,61
20	12,68	36,80	50,52	5,23	10,20
30	8,37	45,49	46,14	9,74	17,84
40	7,78	55,56	36,67	16,21	27,52
50	4,44	68,50	27,06	20,39	41,96

2XC

cycle	E1	E2	E3	R	Q
0	36,14	34,31	29,54	0,00	1,52
10	7,45	8,10	84,44	0,00	3,40
20	5,36	4,58	90,07	0,00	2,81
30	2,94	3,14	93,92	0,00	3,92
40	1,90	2,35	95,75	0,00	4,31
50	2,22	2,09	95,69	0,00	4,18

ERX

TAB. A.3.1 – Jeux d'essai 10x30 sans écarts maximaux

cycle	E1	E2	E3	R	Q
0	29,97	42,97	27,06	4,75	1,65
10	14,84	20,78	64,38	8,30	2,29
20	9,22	20,65	70,13	10,39	3,33
30	7,91	24,18	67,91	15,82	3,27
40	4,84	24,12	71,05	17,91	3,46
50	4,84	26,99	68,17	21,18	3,92

TSXW

cycle	E1	E2	E3	R	Q
0	26,52	40,90	32,58	1,03	1,41
10	13,46	14,90	71,63	3,07	3,01
20	9,74	16,08	74,18	4,90	2,55
30	7,12	14,90	77,97	6,93	3,27
40	6,41	16,80	76,80	10,26	3,40
50	4,64	19,02	76,34	14,12	3,86

TSX

cycle	E1	E2	E3	R	Q
0	30,17	41,92	27,91	1,16	1,84
10	16,93	23,27	59,80	5,23	2,16
20	10,59	20,13	69,28	7,32	3,14
30	5,82	20,07	74,12	12,55	3,01
40	5,29	23,46	71,24	16,34	3,27
50	3,53	26,34	70,13	20,00	4,44

TSXD

cycle	E1	E2	E3	R	Q
0	27,93	51,62	20,44	10,04	1,54
10	17,39	39,15	43,46	19,80	3,20
20	14,31	42,42	43,27	25,36	2,81
30	12,03	41,24	46,73	26,54	4,71
40	11,50	50,33	38,17	35,69	4,44
50	9,80	51,11	39,08	37,84	5,49

1X

cycle	E1	E2	E3	R	Q
0	35,46	45,74	18,80	0,99	1,29
10	21,63	22,94	55,42	1,63	3,14
20	14,97	19,15	65,88	3,01	3,40
30	14,58	21,63	63,79	6,27	5,16
40	12,09	26,01	61,90	10,72	5,82
50	8,50	34,77	56,73	20,33	7,97

2XL

cycle	E1	E2	E3	R	Q
0	34,51	43,07	22,42	1,02	1,45
10	19,22	20,13	60,65	2,35	3,20
20	17,43	25,12	57,45	4,44	3,33
30	15,41	29,71	55,88	10,26	5,75
40	13,84	39,63	46,54	18,50	5,95
50	10,20	46,27	43,53	23,07	10,00

2XC

cycle	E1	E2	E3	R	Q
0	34,82	35,59	29,58	0,00	1,36
10	8,56	7,91	83,53	0,00	2,75
20	3,33	4,31	92,35	0,00	2,29
30	2,22	2,35	95,42	0,00	2,09
40	1,96	2,35	95,69	0,00	3,01
50	1,63	1,76	96,60	0,00	2,55

ERX

TAB. A.3.2 – Jeux d'essai 10x30 avec écarts maximaux

cycle	E1	E2	E3	R	Q
0	27,99	47,07	24,94	4,39	1,16
10	21,11	27,19	51,70	8,69	1,37
20	15,49	28,43	56,08	12,29	2,48
30	10,00	29,28	60,72	17,91	1,63
40	8,89	34,05	57,06	23,53	2,16
50	7,12	35,10	57,78	29,48	2,88

TSXW

cycle	E1	E2	E3	R	Q
0	45,99	39,91	14,10	0,93	1,19
10	29,54	27,71	42,75	3,86	1,76
20	19,35	25,75	54,90	8,37	2,16
30	12,88	28,56	58,56	15,36	2,61
40	10,13	34,31	55,56	24,31	3,27
50	7,19	38,69	54,12	31,63	2,68

TSX

cycle	E1	E2	E3	R	Q
0	51,02	37,40	11,58	0,98	1,18
10	38,10	33,01	28,89	5,56	1,63
20	21,05	34,71	44,25	15,36	2,29
30	10,72	37,39	51,90	25,42	2,61
40	8,43	40,85	50,72	30,59	3,07
50	6,86	41,31	51,83	33,86	3,27

TSXD

cycle	E1	E2	E3	R	Q
0	27,46	49,87	22,67	10,38	1,19
10	20,59	38,43	40,98	17,65	2,35
20	16,21	39,87	43,92	22,22	1,96
30	12,16	46,73	41,11	33,20	2,75
40	11,70	51,83	36,47	40,39	2,81
50	11,63	53,20	35,16	42,09	4,25

1X

cycle	E1	E2	E3	R	Q
0	33,27	44,65	22,08	0,97	1,19
10	22,81	22,35	54,84	1,90	1,96
20	18,10	23,27	58,63	3,86	2,68
30	14,97	23,07	61,96	6,41	2,22
40	12,88	28,89	58,24	11,83	3,86
50	12,09	32,61	55,29	20,20	3,86

2XL

cycle	E1	E2	E3	R	Q
0	35,01	40,99	24,00	1,02	1,25
10	17,97	22,55	59,48	2,48	1,96
20	17,84	24,12	58,04	5,82	2,81
30	16,82	28,76	54,42	11,83	2,88
40	16,14	33,33	50,52	17,71	3,79
50	14,38	34,64	50,98	22,09	3,66

2XC

cycle	E1	E2	E3	R	Q
0	32,56	34,98	32,46	0,00	1,15
10	6,60	7,84	85,56	0,00	1,83
20	3,99	4,38	91,63	0,00	2,48
30	2,75	2,48	94,77	0,00	2,48
40	1,57	2,29	96,14	0,00	1,76
50	0,78	1,50	97,71	0,00	2,09

ERX

TAB. A.3.3 – Jeux d'essai 10x30 sans attente

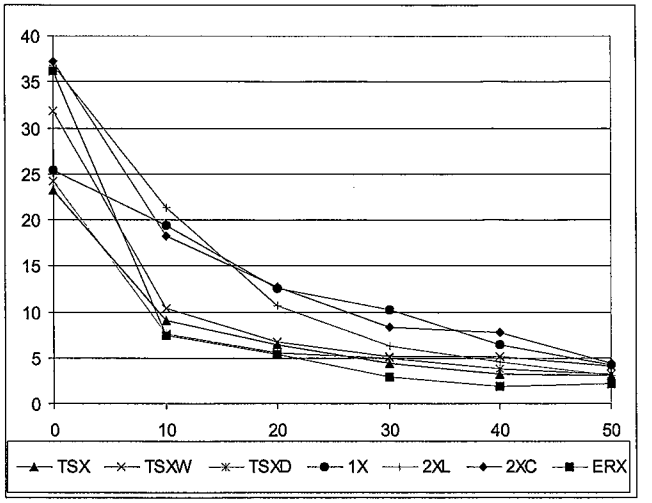


FIG. A.3.1 – Evolution du compteur E1 - jeux d'essai 10x30 sans écarts maximaux

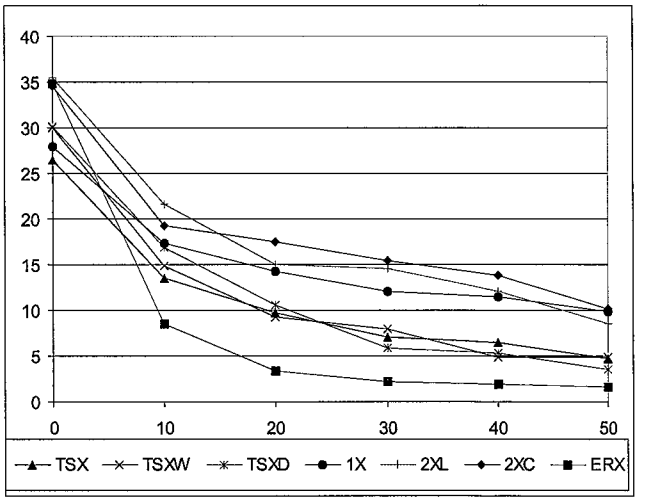


FIG. A.3.2 – Evolution du compteur E1 - jeux d'essai 10x30 avec écarts maximaux

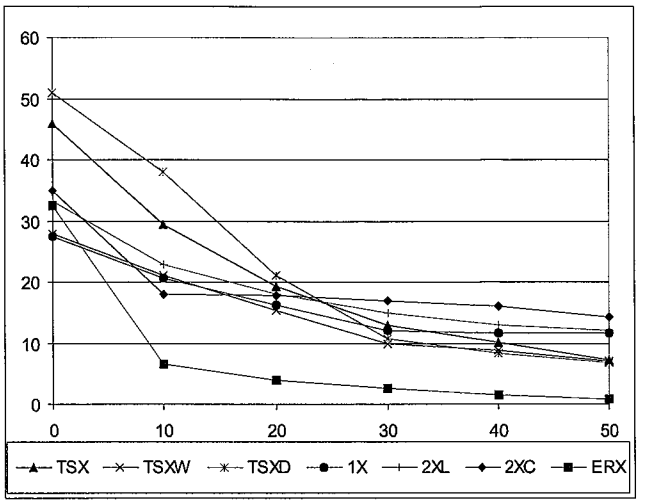


FIG. A.3.3 – Evolution du compteur E1 - jeux d'essai 10x30 sans attente

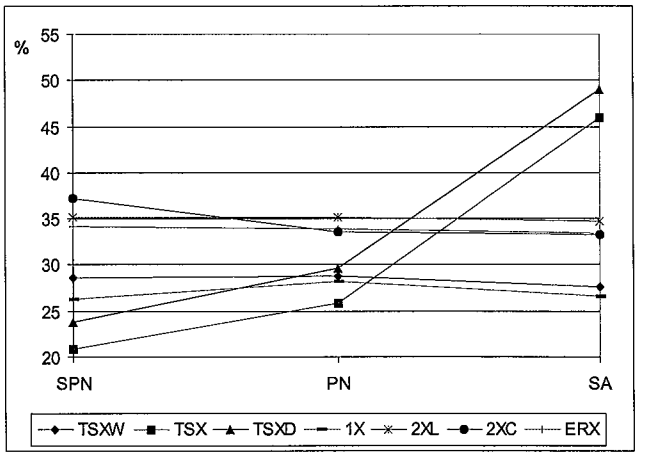


FIG. A.3.4 – Evolution du compteur E1 en fonction de la tension sur les écarts maximaux

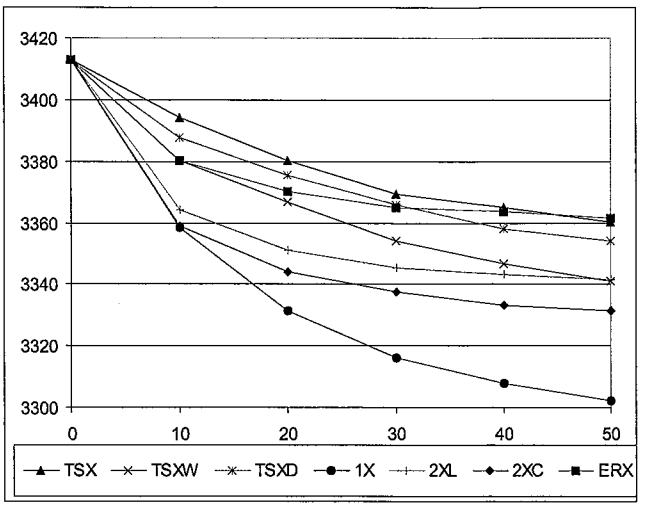


FIG. A.3.5 – Evolution du makespan moyen au cours des cycles pour les jeux d'essai sans écarts maximaux de taille 10x30

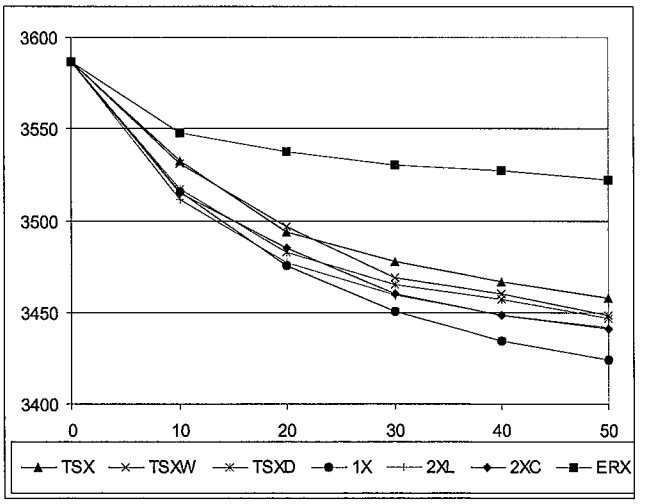


FIG. A.3.6 – Evolution du makespan moyen au cours des cycles pour les jeux d'essai avec écarts maximaux de taille 10x30

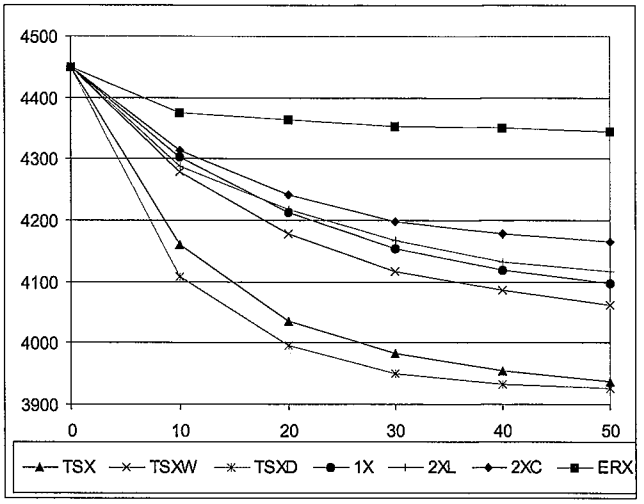


FIG. A.3.7 – Evolution du makespan moyen au cours des cycles pour les jeux d'essai sans attente de taille 10x30

3.2 Qualité des opérateurs de croisement - jeux d'essai 5x20

cycle	E1	E2	E3	R	Q
0	31,54	40,80	27,66	6,10	1,90
10	10,20	27,45	62,35	15,49	2,81
20	6,27	35,36	58,37	23,53	6,54
30	3,73	38,69	57,58	28,24	11,76
40	3,46	40,00	56,54	30,39	13,92
50	2,48	41,70	55,82	32,55	18,56

TSXW

cycle	E1	E2	E3	R	Q
0	24,81	37,56	37,63	1,40	1,84
10	8,63	16,27	75,10	5,23	3,01
20	5,49	18,17	76,34	8,76	5,23
30	4,51	17,12	78,37	10,13	5,88
40	3,59	20,00	76,41	13,92	7,84
50	2,55	21,11	76,34	16,86	10,07

TSX

cycle	E1	E2	E3	R	Q
0	28,16	38,51	33,33	1,28	1,76
10	9,35	19,80	70,85	7,39	4,25
20	5,75	23,01	71,24	12,22	6,54
30	3,66	25,49	70,85	16,34	9,22
40	2,75	29,48	67,78	22,35	13,46
50	2,07	27,71	70,22	19,54	13,59

TSXD

cycle	E1	E2	E3	R	Q
0	22,35	62,24	15,41	14,20	1,79
10	16,27	57,84	25,88	21,96	6,60
20	8,69	66,14	25,16	30,98	20,52
30	5,16	72,16	22,68	39,93	40,52
40	3,59	77,06	19,35	45,03	51,37
50	1,37	82,68	15,95	47,91	66,01

1X

cycle	E1	E2	E3	R	Q
0	32,55	51,53	15,92	1,10	1,82
10	16,99	51,44	31,57	2,88	17,32
20	3,73	71,50	24,77	10,13	66,41
30	1,11	77,39	21,50	14,25	76,93
40	0,72	79,08	20,20	16,41	87,78
50	0,92	77,71	21,37	16,14	92,29

2XL

cycle	E1	E2	E3	R	Q
0	35,54	50,20	14,26	1,12	1,83
10	12,09	47,06	40,85	4,18	8,43
20	8,63	54,84	36,54	9,08	18,30
30	4,84	64,05	31,11	17,97	38,69
40	2,61	72,55	24,84	21,24	45,10
50	1,76	74,31	23,92	21,63	48,24

2XC

cycle	E1	E2	E3	R	Q
0	35,41	34,69	29,90	0,00	1,78
10	8,24	9,48	82,29	0,00	2,81
20	5,03	5,75	89,22	0,00	4,25
30	2,94	3,66	93,40	0,00	5,10
40	2,35	2,94	94,71	0,07	5,75
50	2,03	1,83	96,14	0,00	6,67

ERX

TAB. A.3.4 – Jeux d'essai 5x20 sans écarts maximaux

cycle	E1	E2	E3	R	Q
0	31,07	43,19	25,74	6,04	1,58
10	12,16	29,48	58,37	16,34	2,61
20	8,76	35,42	55,82	22,68	3,86
30	6,21	39,93	53,86	30,92	4,77
40	5,29	40,26	54,44	32,48	6,67
50	4,44	41,96	53,59	35,62	6,93

TSXW

cycle	E1	E2	E3	R	Q
0	27,18	42,03	30,80	1,41	1,63
10	14,38	21,76	63,86	7,84	3,53
20	8,95	20,59	70,46	10,85	4,25
30	4,84	24,97	70,20	16,14	4,51
40	4,18	30,00	65,82	22,48	5,95
50	3,53	29,87	66,60	23,92	5,69

TSX

cycle	E1	E2	E3	R	Q
0	31,78	42,80	25,42	1,36	1,79
10	16,86	26,41	56,73	10,65	3,53
20	9,28	30,07	60,65	17,19	3,33
30	5,88	33,59	60,52	25,03	4,44
40	4,64	36,67	58,69	30,07	4,97
50	4,12	34,64	61,24	29,28	5,75

TSXD

cycle	E1	E2	E3	R	Q
0	24,85	55,99	19,16	14,78	1,67
10	19,80	42,48	37,71	22,68	3,79
20	14,90	42,09	43,01	25,82	4,25
30	10,20	49,41	40,39	33,40	5,62
40	7,84	55,69	36,47	40,92	8,24
50	7,56	59,15	33,29	43,92	7,52

1X

cycle	E1	E2	E3	R	Q
0	35,66	45,57	18,77	1,01	1,75
10	21,57	24,58	53,86	2,88	3,59
20	14,38	26,21	59,41	7,12	5,69
30	11,05	35,10	53,86	14,97	8,04
40	8,30	40,72	50,98	18,63	12,42
50	8,37	43,33	48,30	22,81	14,84

2XL

cycle	E1	E2	E3	R	Q
0	34,35	44,20	21,45	0,95	1,62
10	18,76	25,42	55,82	4,18	3,27
20	17,43	25,12	56,14	9,15	6,01
30	15,41	29,71	55,88	15,75	9,35
40	13,84	39,63	40,26	22,81	13,01
50	7,78	50,85	41,37	24,97	18,63

2XC

cycle	E1	E2	E3	R	Q
0	33,06	35,46	31,48	0,00	1,71
10	7,25	8,89	83,86	0,00	3,20
20	3,92	4,12	91,96	0,00	3,33
30	2,55	2,29	95,16	0,00	4,18
40	1,50	2,16	96,34	0,00	4,05
50	1,44	2,03	96,54	0,00	3,46

ERX

TAB. A.3.5 – Jeux d'essai 5x20 avec écarts maximaux

cycle	E1	E2	E3	R	Q
0	29,90	45,73	24,38	6,18	1,24
10	19,74	35,23	45,03	17,91	2,09
20	12,48	40,07	47,45	27,52	2,88
30	8,30	43,14	48,56	32,48	3,46
40	7,58	46,47	45,95	38,43	3,99
50	5,69	48,63	45,69	39,67	4,38

TSXW

cycle	E1	E2	E3	R	Q
0	38,09	42,71	19,20	1,40	1,58
10	23,92	32,88	43,20	9,35	2,35
20	12,35	37,52	50,13	23,14	3,92
30	8,30	41,18	50,52	31,63	4,90
40	5,23	43,14	51,63	35,42	3,86
50	5,23	43,46	51,31	36,47	5,03

TSX

cycle	E1	E2	E3	R	Q
0	44,33	40,48	15,19	1,20	1,50
10	27,71	37,58	34,71	13,07	3,33
20	12,81	46,08	41,11	30,52	3,73
30	8,69	50,92	40,39	38,69	3,66
40	7,65	48,04	44,31	40,00	3,20
50	5,69	46,67	47,65	39,15	5,29

TSXD

cycle	E1	E2	E3	R	Q
0	26,63	51,71	21,66	14,81	1,61
10	19,22	42,09	38,69	22,55	2,22
20	13,92	44,64	41,44	30,52	2,94
30	11,70	49,74	38,56	35,75	2,61
40	8,50	52,55	38,95	39,48	3,86
50	8,01	54,44	37,55	42,55	4,25

1X

cycle	E1	E2	E3	R	Q
0	35,76	43,78	20,46	1,16	1,62
10	22,22	23,46	54,31	2,68	2,94
20	15,23	20,72	64,05	5,95	3,53
30	14,64	28,10	57,25	13,46	5,36
40	10,85	32,22	56,93	17,71	4,44
50	10,63	37,93	51,44	22,35	6,14

2XL

cycle	E1	E2	E3	R	Q
0	34,76	40,08	25,16	0,86	1,29
10	18,37	24,31	57,32	4,31	3,46
20	17,25	26,60	56,14	10,39	3,20
30	16,82	32,22	54,42	16,01	4,25
40	14,18	34,44	51,37	20,59	4,64
50	13,59	37,45	48,95	22,94	5,03

2XC

cycle	E1	E2	E3	R	Q
0	34,90	33,86	31,24	0,00	1,54
10	7,84	8,30	83,86	0,00	2,03
20	3,01	4,31	92,68	0,00	2,22
30	2,75	2,03	95,23	0,00	3,46
40	1,57	1,57	96,86	0,00	2,88
50	1,70	1,57	96,73	0,00	2,68

ERX

TAB. A.3.6 – Jeux d'essai 5x20 sans attente

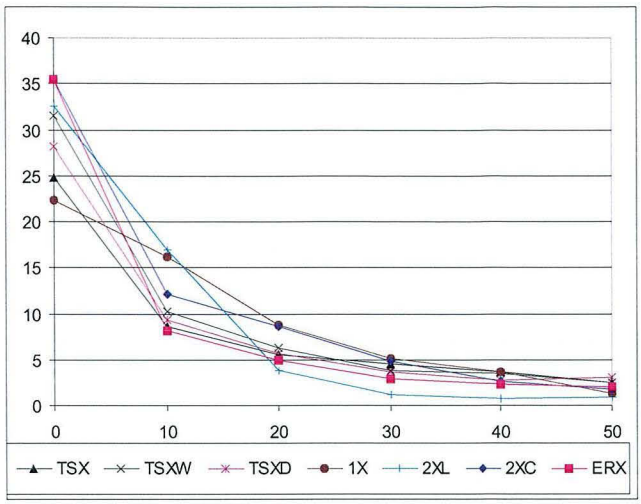


FIG. A.3.8 – Evolution du compteur E1 - jeux d'essai 5x20 sans écarts maximaux

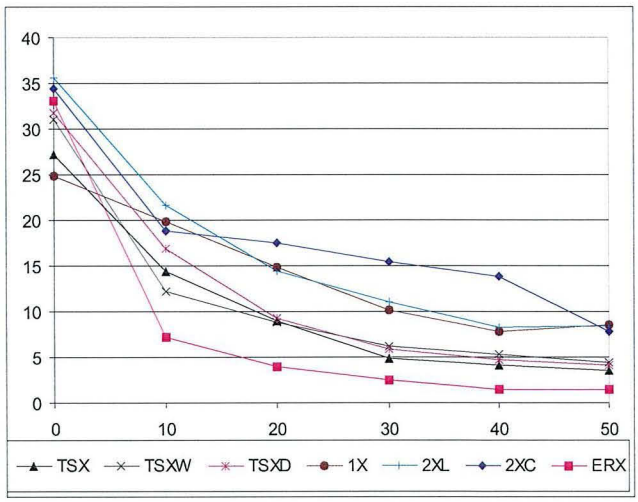


FIG. A.3.9 – Evolution du compteur E1 - jeux d'essai 5x20 avec écarts maximaux

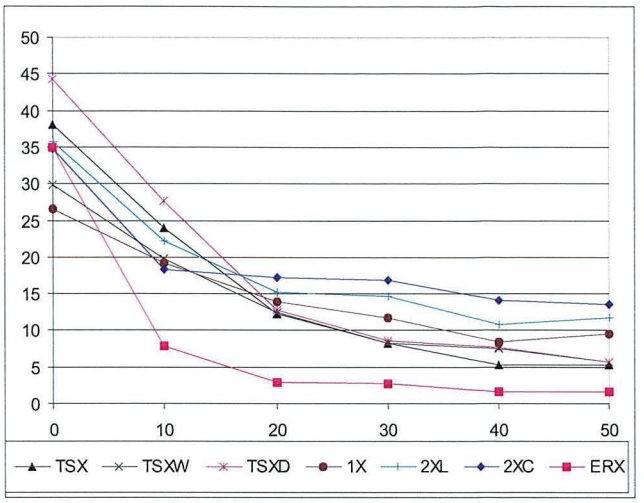


FIG. A.3.10 – Evolution du compteur E1 - jeux d'essai 5x20 sans attente

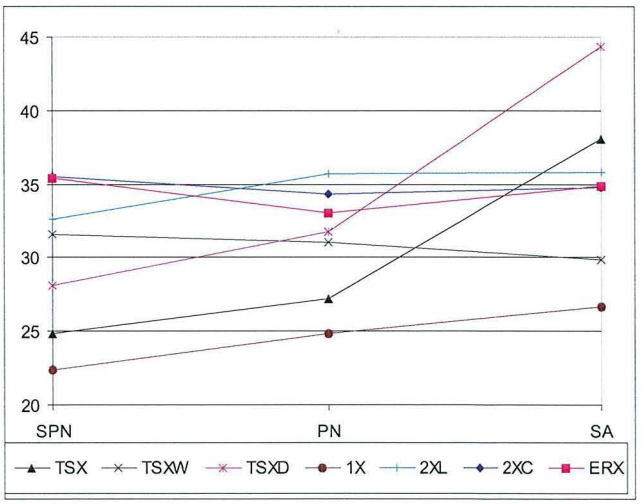


FIG. A.3.11 – Evolution du compteur E1 en fonction de la tension sur les écarts maximaux

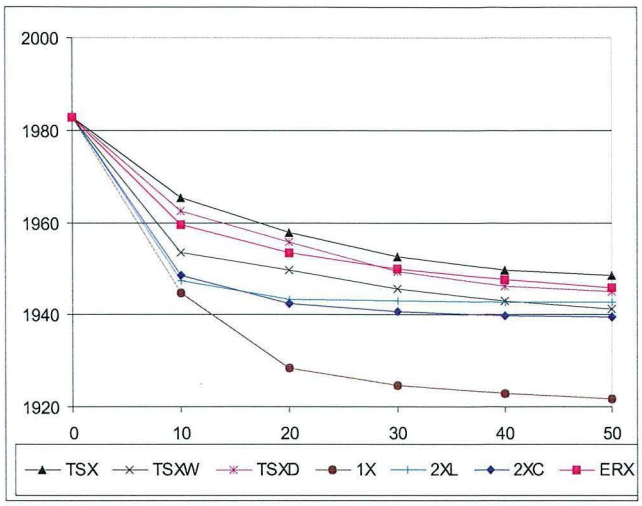


FIG. A.3.12 – Evolution du makespan moyen au cours des cycles pour les jeux d'essai sans écarts maximaux de taille 5x20

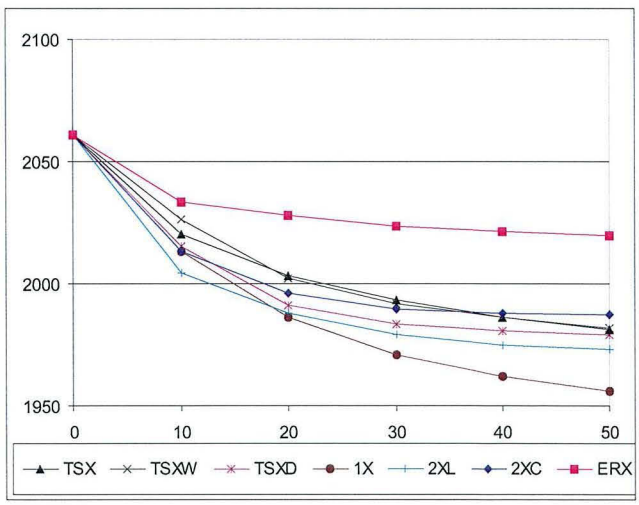


FIG. A.3.13 – Evolution du makespan moyen au cours des cycles pour les jeux d'essai avec écarts maximaux de taille 5x20

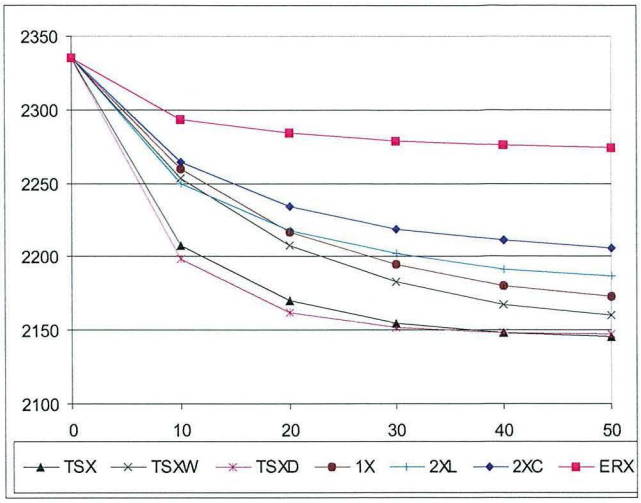


FIG. A.3.14 – Evolution du makespan moyen au cours des cycles pour les jeux d'essai sans attente de taille 5x20

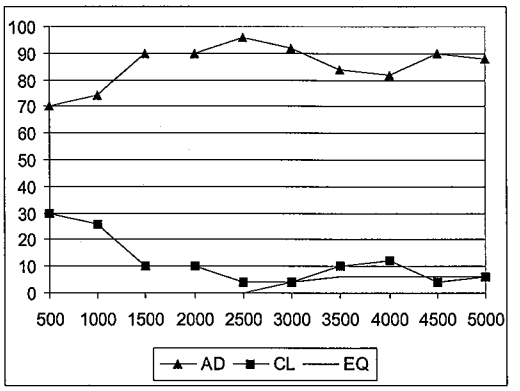
3.3 Autodétermination vs 1X - Jeux d'essai de taille 10x30

Cette section présente les résultats numériques obtenus sur les jeux d'essai de taille $m = 10$, $n = 30$, au cours des expériences décrites dans le chapitre 3, section 3.5.3, pour comparer la méthode d'autodétermination avec la méthode classique et l'utilisation de l'opérateur 1X .

La figure A.3.15 (resp. A.3.16 et A.3.17) permet de comparer la méthode d'autodétermination sur les jeux d'essai sans écarts maximaux (resp. avec écarts maximaux et sans attente) en tenant compte du score du meilleur individu. Le premier graphique permet de mesurer au cours des cycles (abscisse), le pourcentage (ordonnée) des jeux d'essai pour lesquels la méthode d'autodétermination obtient de meilleurs résultats (courbe AD), le pourcentage des jeux d'essai pour lesquels la méthode classique obtient de meilleurs résultats (courbe CL) et le pourcentage des jeux d'essai pour lesquels les deux méthodes obtiennent des résultats identiques (courbe EQ). Le second graphique permet de comparer les deux méthodes en tenant compte du score moyen des 10 meilleurs individus.

La section 3.4 compare la méthode d'autodétermination avec la méthode classique et l'utilisation de l'opérateur 1X sur les jeux d'essai de taille 5x20.
La section 3.5 compare la méthode d'autodétermination avec la méthode classique et l'utilisation de l'opérateur TSXD sur les jeux d'essai de taille 10x30.
La section 3.6 compare la méthode d'autodétermination avec la méthode classique et l'utilisation de l'opérateur TSXD sur les jeux d'essai de taille 5x20.
La section 3.7 compare la méthode d'autodétermination avec la méthode classique et l'utilisation de l'opérateur 2X sur les jeux d'essai de taille 10x30.
La section 3.8 compare la méthode d'autodétermination avec la méthode classique et l'utilisation de l'opérateur 2X sur les jeux d'essai de taille 5x20.
La section 3.9 compare la méthode d'autodétermination avec la méthode classique et l'utilisation combinée des opérateurs 1X et TSXD sur les jeux d'essai de taille 10x30.
La section 3.10 compare la méthode d'autodétermination avec la méthode classique et l'utilisation combinée des opérateurs 1X et TSXD sur les jeux d'essai de taille 5x20.
La section 3.11 compare la méthode d'autodétermination avec et sans mutation sur les jeux d'essai de taille 5x20 et 10x30.
La section 3.12 compare la méthode classique et l'opérateur 1X avec et sans mutation sur les jeux d'essai de taille 5x20.
La section 3.13 compare la méthode classique et l'opérateur TSXD avec et sans mutation sur les jeux d'essai de taille 5x20.

meilleur individu



10 meilleurs individus

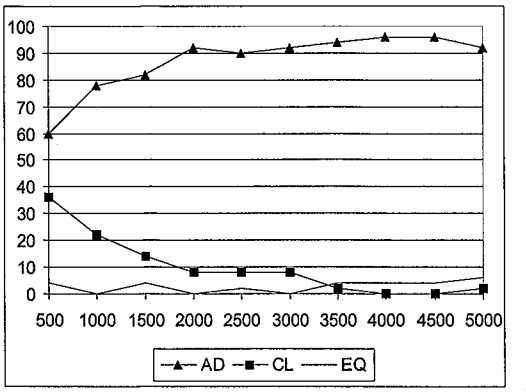
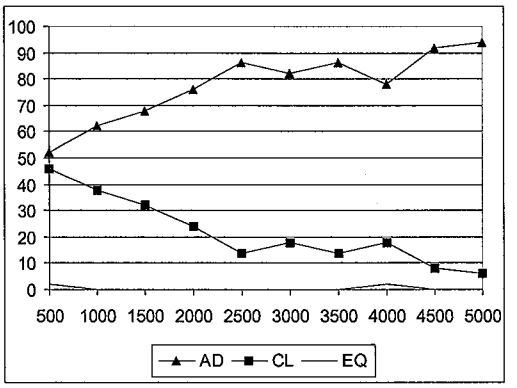


FIG. A.3.15 – Autodétermination vs 1X - 10x30 sans écarts maximaux



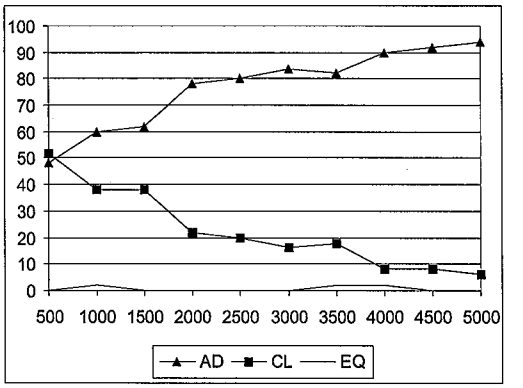
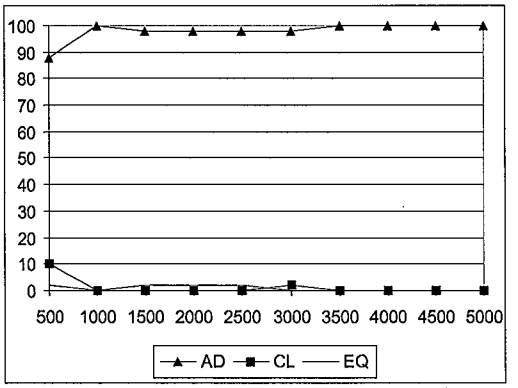


FIG. A.3.16 – Autodétermination vs 1X - 10x30 avec écarts maximaux



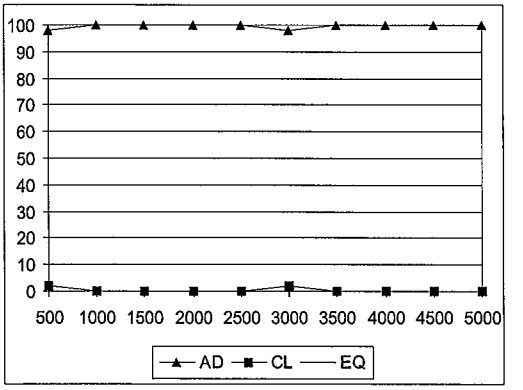
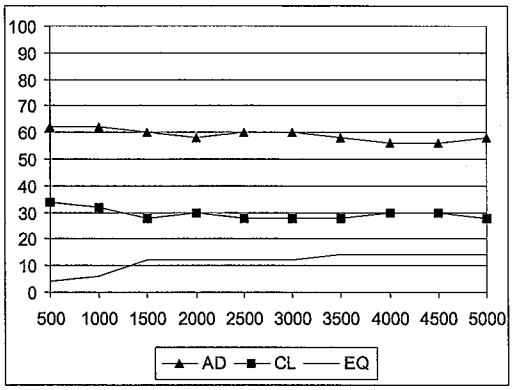


FIG. A.3.17 – Autodétermination vs 1X - 10x30 sans attente

3.4 Autodétermination vs 1X - Jeux d'essai de taille 5x20

meilleur individu



10 meilleurs individus

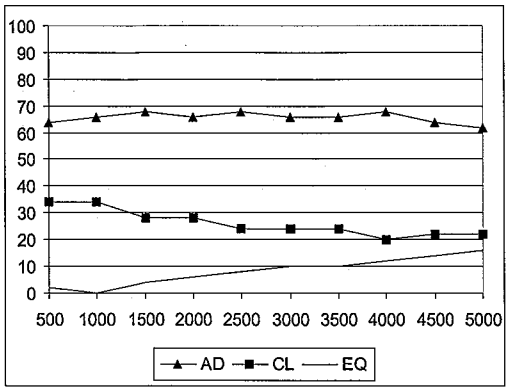
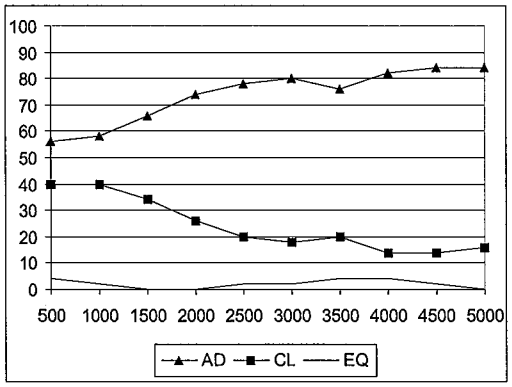


FIG. A.3.18 – Autodétermination vs 1X - 5x20 sans écarts maximaux



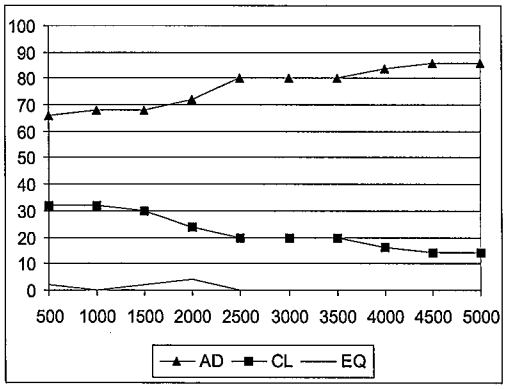
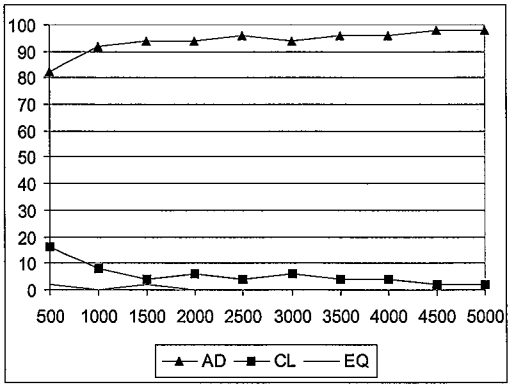


FIG. A.3.19 – Autodétermination vs 1X - 5x20 avec écarts maximaux



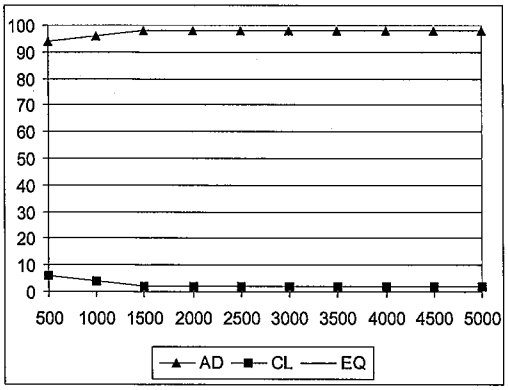
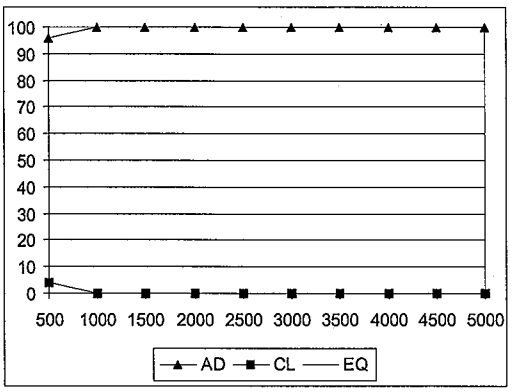


FIG. A.3.20 – Autodétermination vs 1X - 5x20 sans attente

3.5 Autodétermination vs TSXD - Jeux d'essai de taille 10x30

meilleur individu



10 meilleurs individus

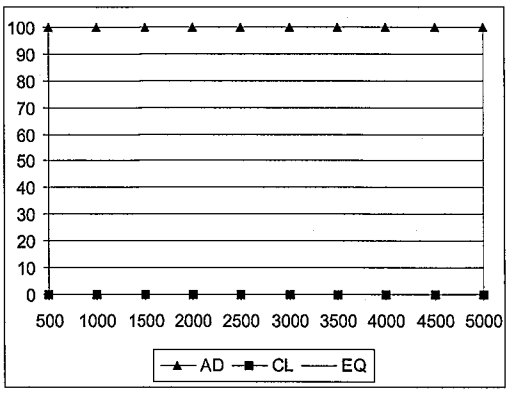
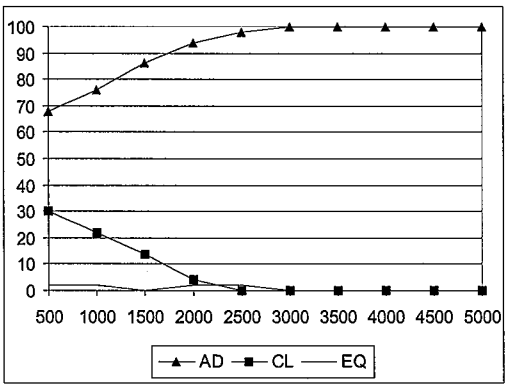


FIG. A.3.21 – Autodétermination vs TSXD - 10x30 sans écarts maximaux



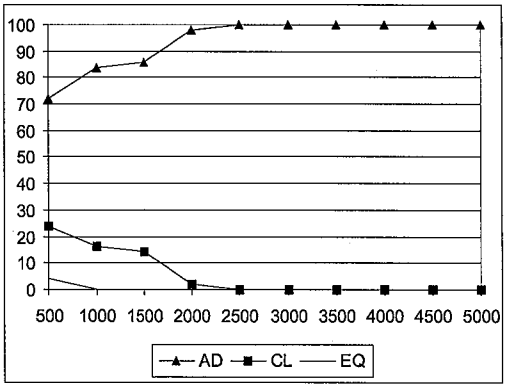
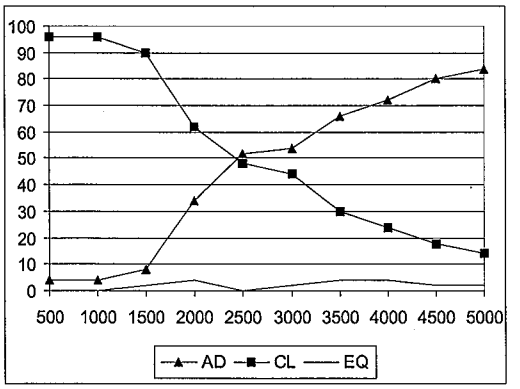


FIG. A.3.22 – Autodétermination vs TSXD - 10x30 avec écarts maximaux



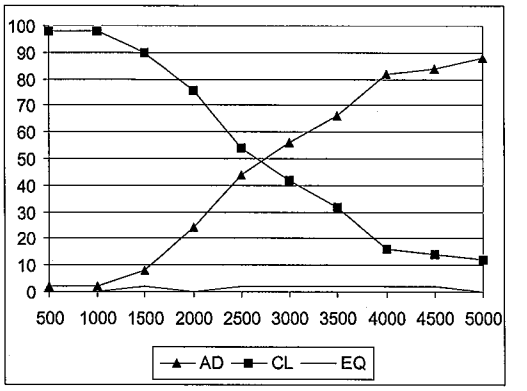
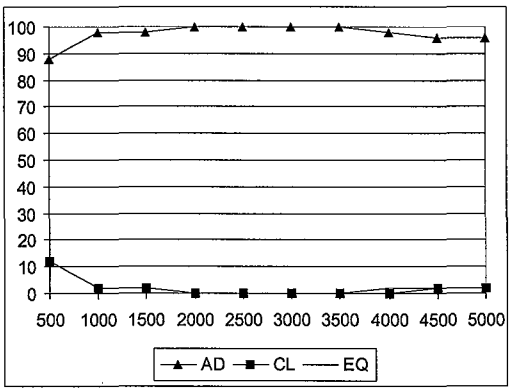


FIG. A.3.23 – Autodétermination vs TSXD - 10x30 sans attente

3.6 Autodétermination vs TSXD - Jeux d'essai de taille 5x20

meilleur individu



10 meilleurs individus

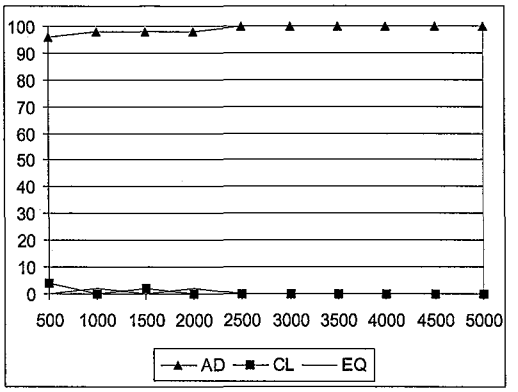
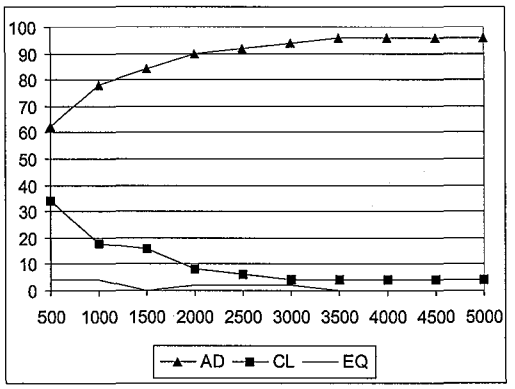


FIG. A.3.24 – Autodétermination vs TSXD - 5x20 sans écarts maximaux



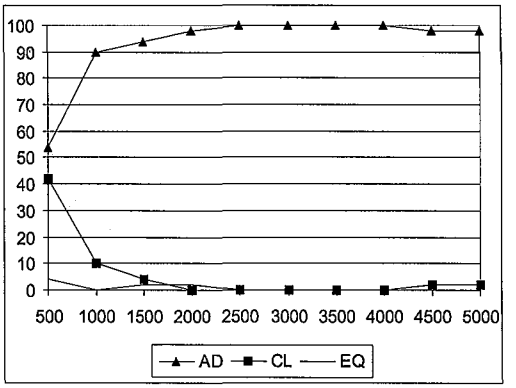
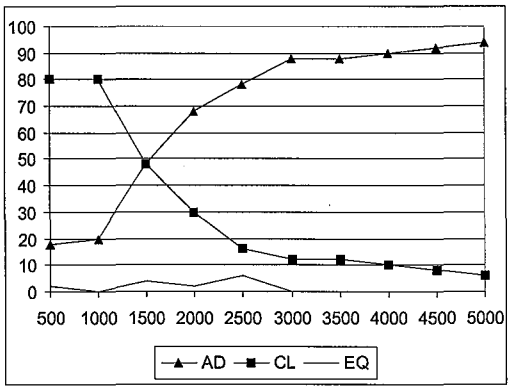


FIG. A.3.25 – Autodétermination vs TSXD - 5x20 avec écarts maximaux



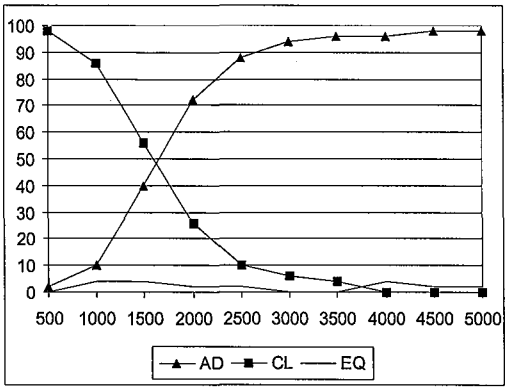
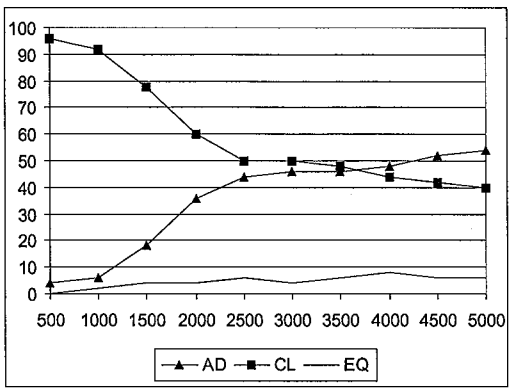


FIG. A.3.26 – Autodétermination vs TSXD - 5x20 sans attente

3.7 Autodétermination vs 2X - Jeux d'essai de taille 10x30

meilleur individu



10 meilleurs individus

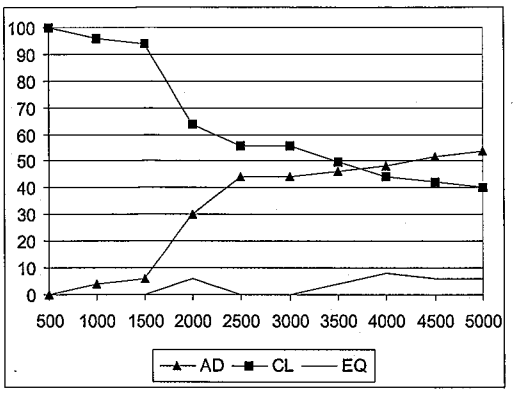
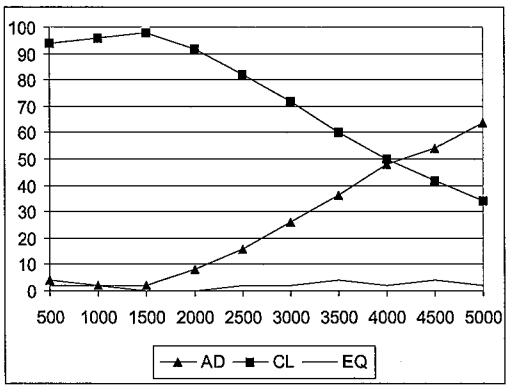


FIG. A.3.27 – Autodétermination vs 2X - 10x30 sans écarts maximaux



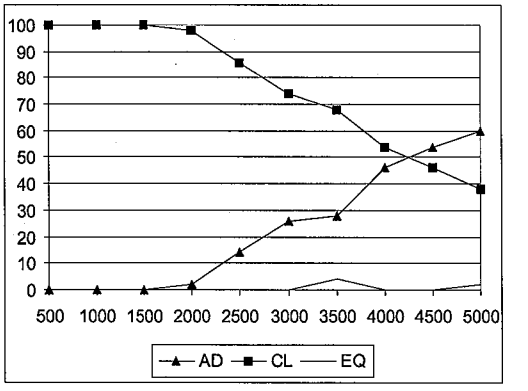
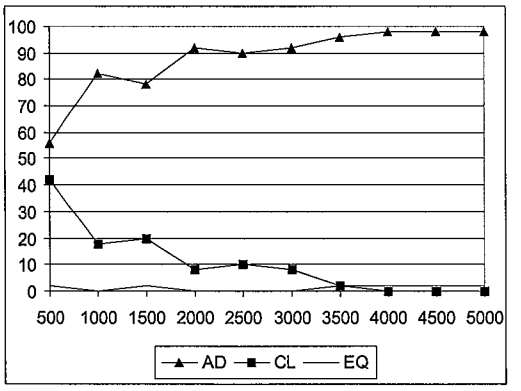


FIG. A.3.28 – Autodétermination vs 2X - 10x30 avec écarts maximaux



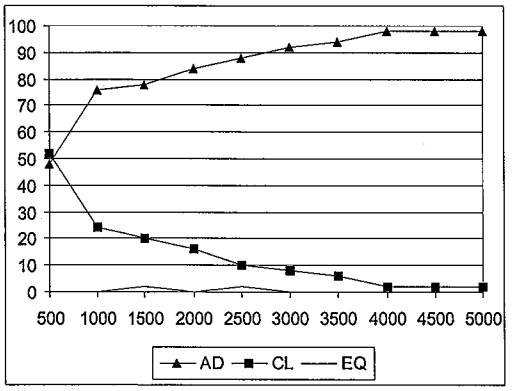
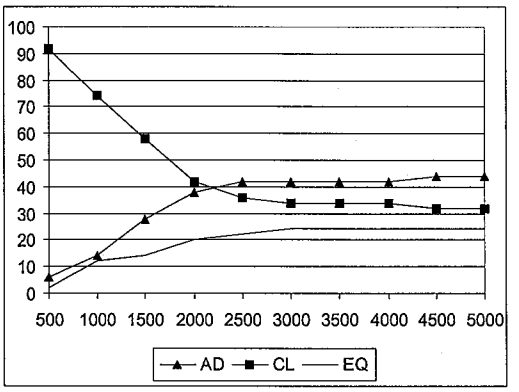


FIG. A.3.29 – Autodétermination vs 2X - 10x30 sans attente

3.8 Autodétermination vs 2X - Jeux d'essai de taille 5x20

meilleur individu



10 meilleurs individus

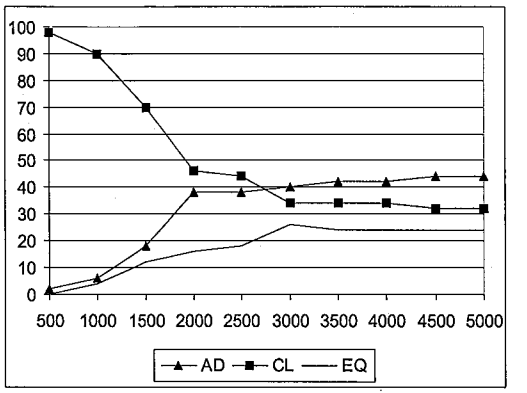
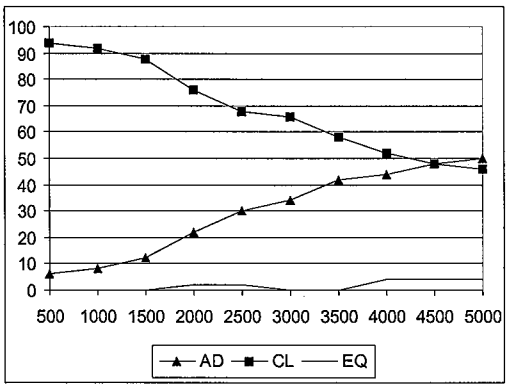


FIG. A.3.30 – Autodétermination vs 2X - 5x20 sans écarts maximaux



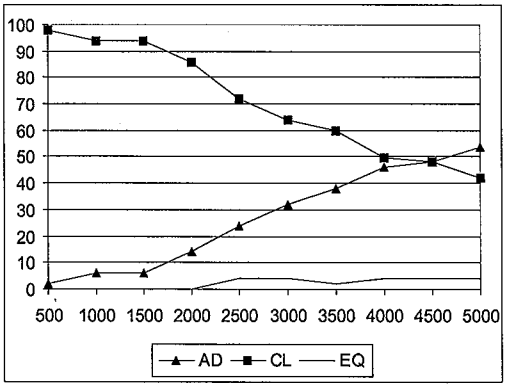
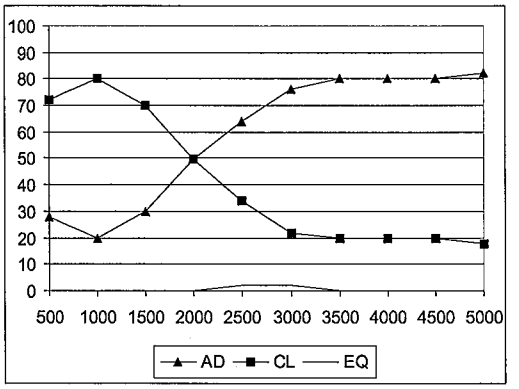


FIG. A.3.31 – Autodétermination vs 2X - 5x20 avec écarts maximaux



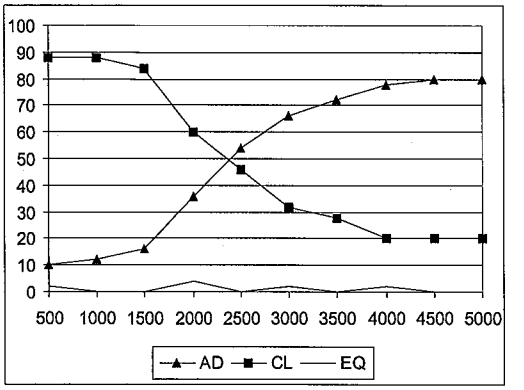
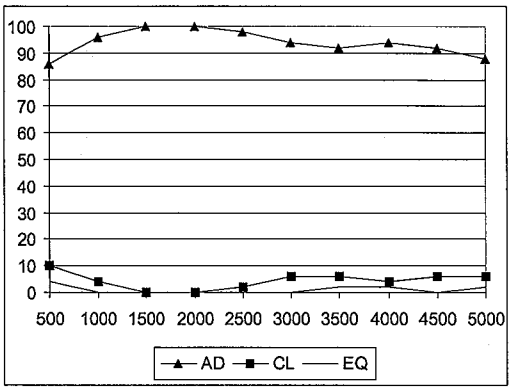


FIG. A.3.32 – Autodétermination vs 2X - 5x20 sans attente

3.9 Autodétermination vs 1X-TSXD - Jeux d'essai de taille 10x30

meilleur individu



10 meilleurs individus

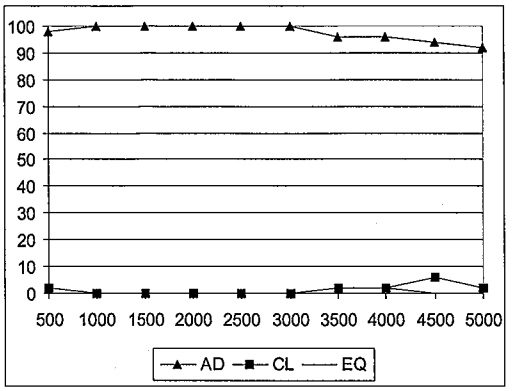
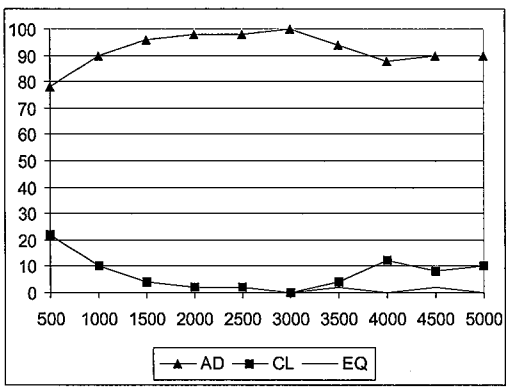


FIG. A.3.33 – Autodétermination vs 1X-TSXD - 10x30 sans écarts maximaux



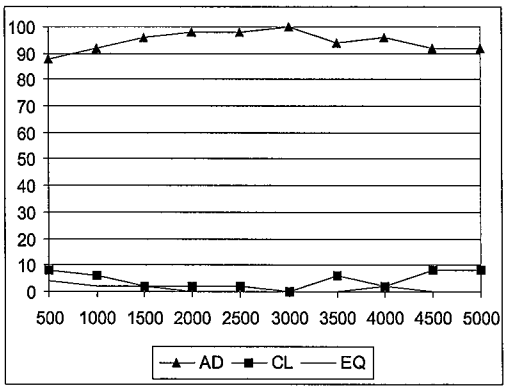
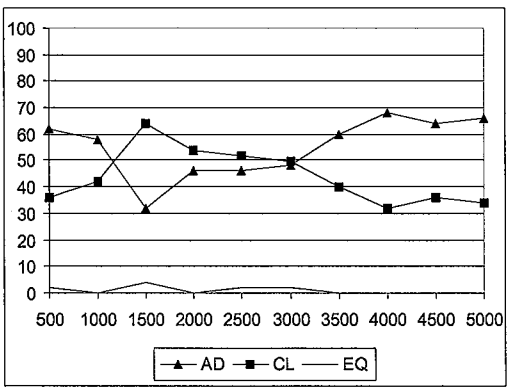


FIG. A.3.34 – Autodétermination vs 1X-TSXD - 10x30 avec écarts maximaux



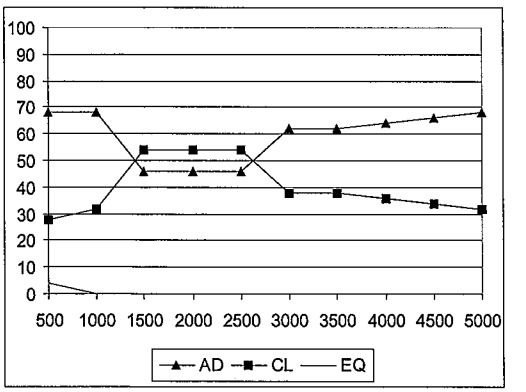
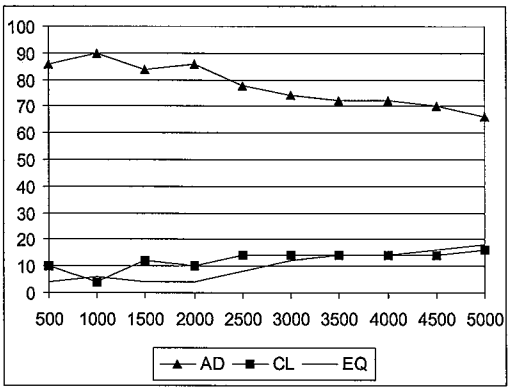


FIG. A.3.35 – Autodétermination vs 1X-TSXD - 10x30 sans attente

3.10 Autodétermination vs 1X-TSXD - Jeux d'essai de taille 5x20

meilleur individu



10 meilleurs individus

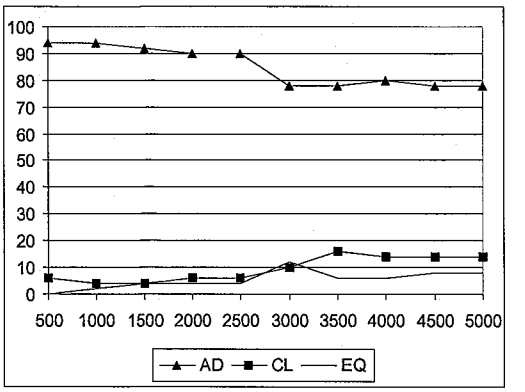
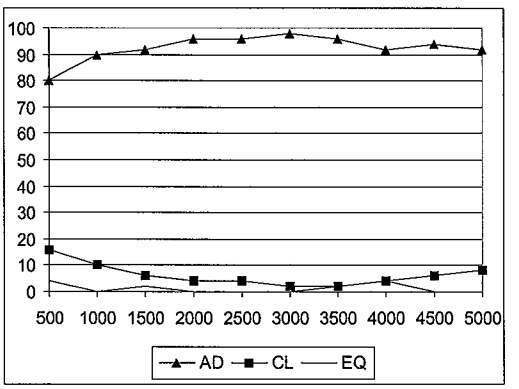


FIG. A.3.36 – Autodétermination vs 1X-TSXD - 5x20 sans écarts maximaux



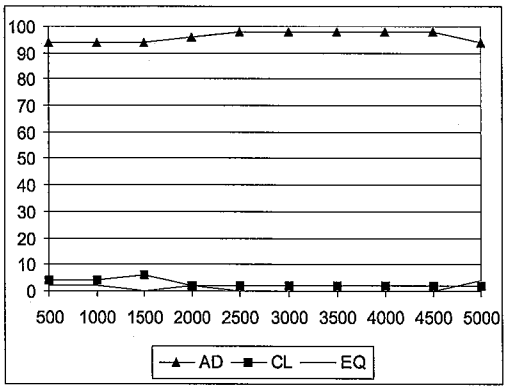
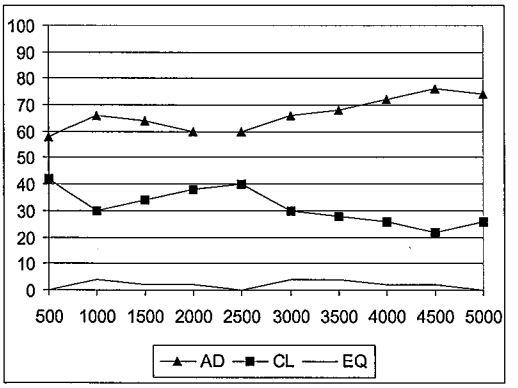


FIG. A.3.37 – Autodétermination vs 1X-TSXD - 5x20 avec écarts maximaux



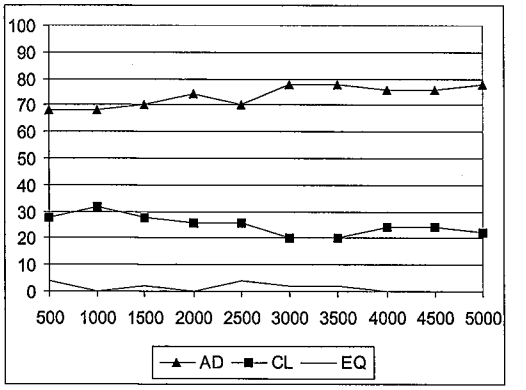
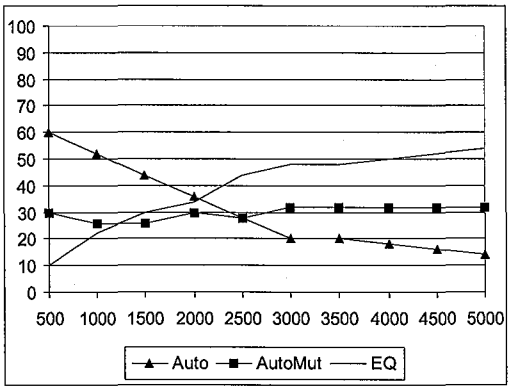


FIG. A.3.38 – Autodétermination vs 1X-TSXD - 5x20 sans attente

3.11 Autodétermination vs Autodétermination avec mutation

meilleur individu



10 meilleurs individus

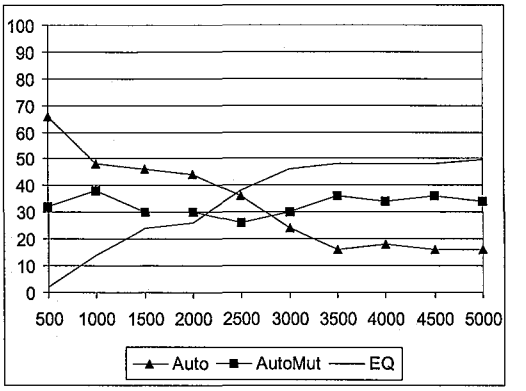
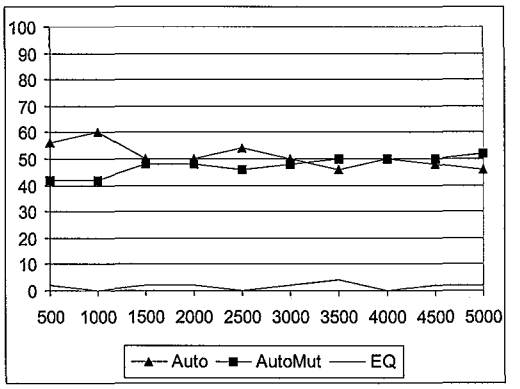


FIG. A.3.39 – Autodétermination avec et sans mutation - 5x20 sans écarts maximaux



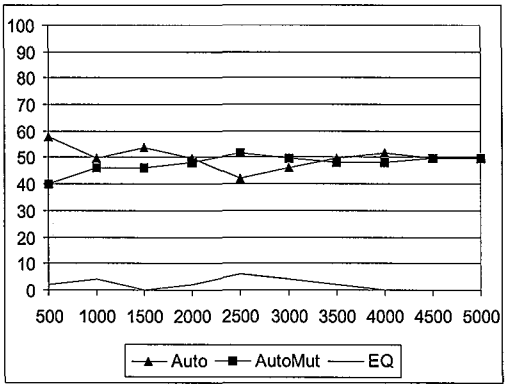
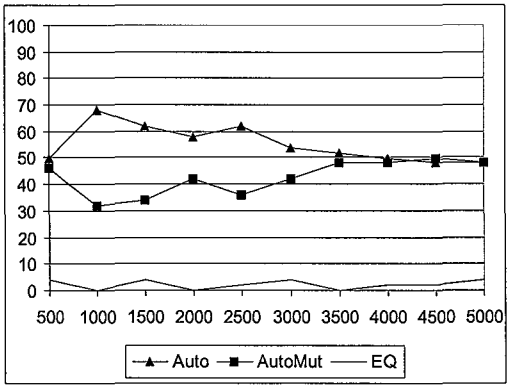


FIG. A.3.40 – Autodétermination avec et sans mutation - 5x20 avec écarts maximaux



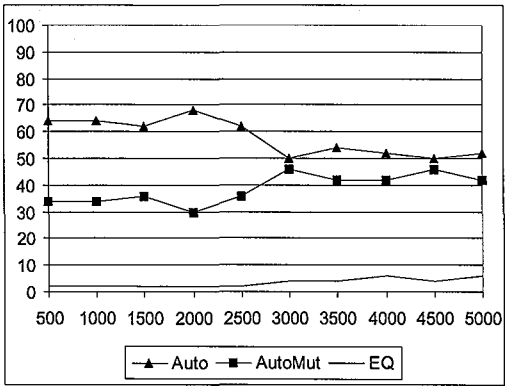
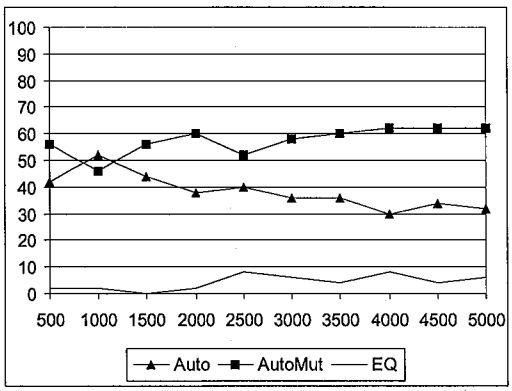


FIG. A.3.41 – Autodétermination avec et sans mutation - 5x20 sans attente

meilleur individu



10 meilleurs individus

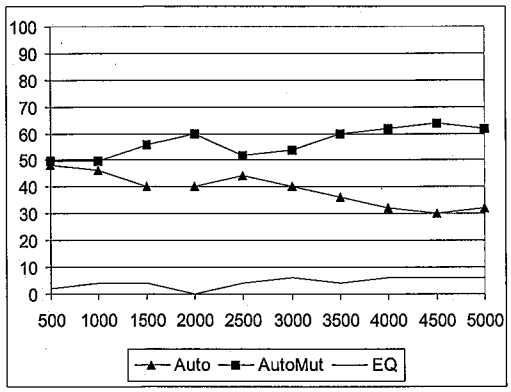
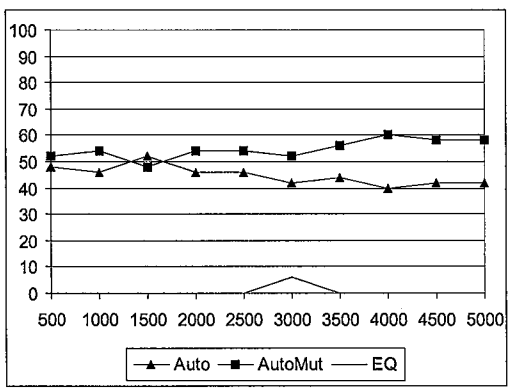


FIG. A.3.42 – Autodétermination avec et sans mutation - 10x30 sans écarts maximaux



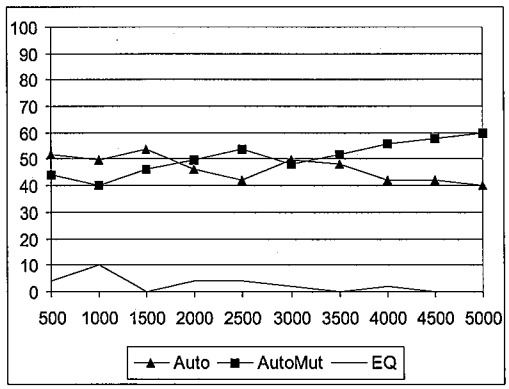
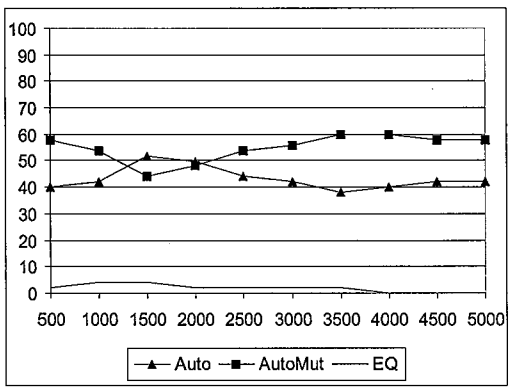


FIG. A.3.43 – Autodétermination avec et sans mutation - 10x30 avec écarts maximaux



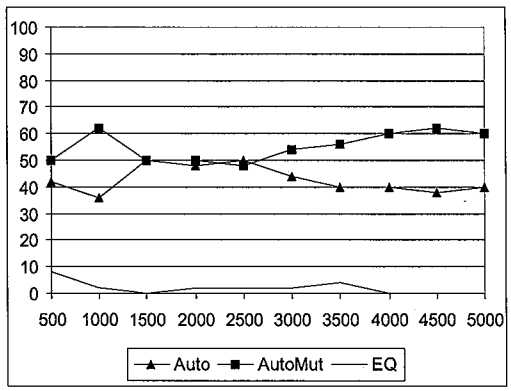
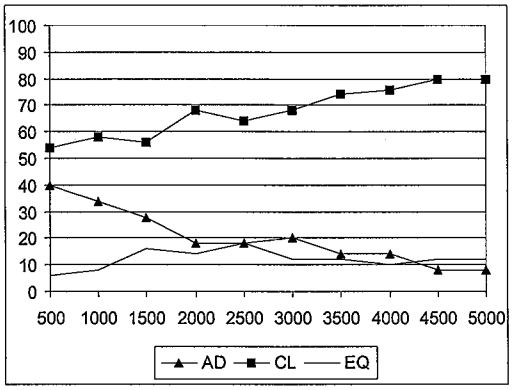


FIG. A.3.44 – Autodétermination avec et sans mutation - 10x30 sans attente

3.12 1X vs 1X avec mutation

meilleur individu



10 meilleurs individus

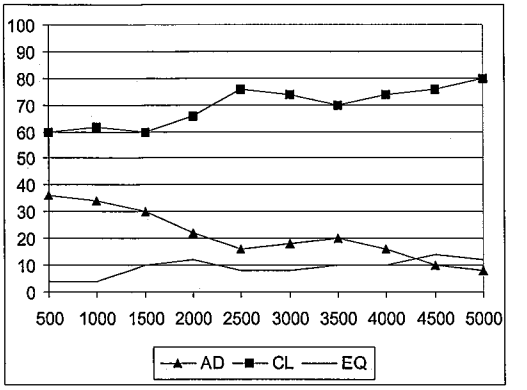
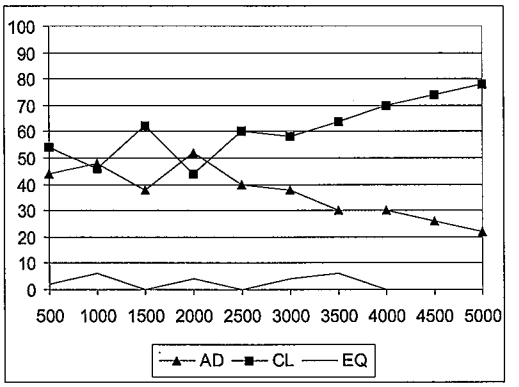


FIG. A.3.45 – 1X avec et sans mutation - 5x20 sans écarts maximaux



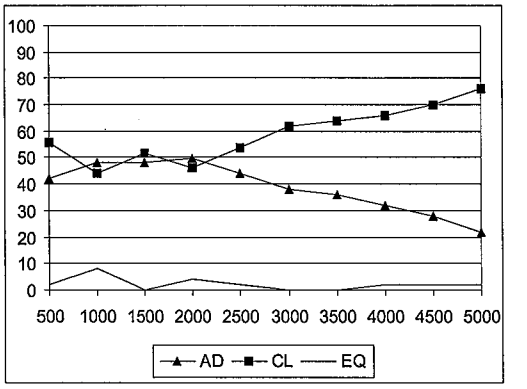
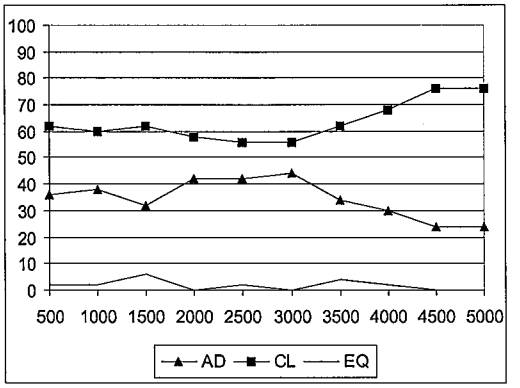


FIG. A.3.46 – 1X avec et sans mutation - 5x20 avec écarts maximaux



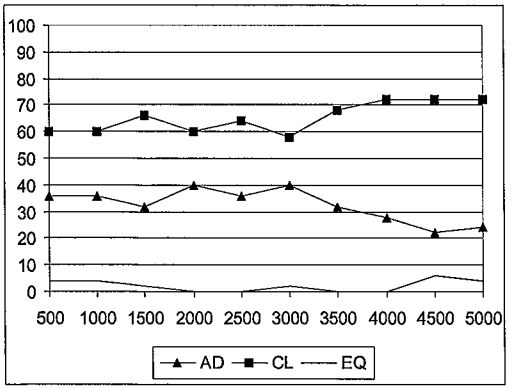
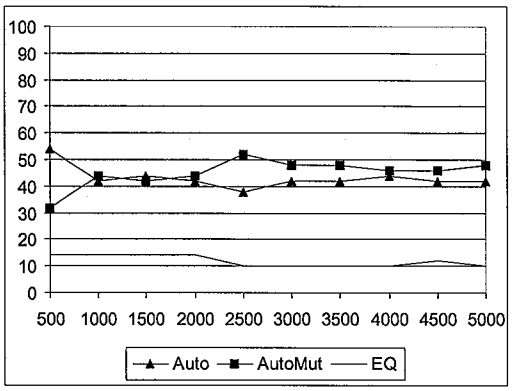


FIG. A.3.47 – 1X avec et sans mutation - 5x20 sans attente

3.13 TSXD vs TSXD avec mutation

meilleur individu



10 meilleurs individus

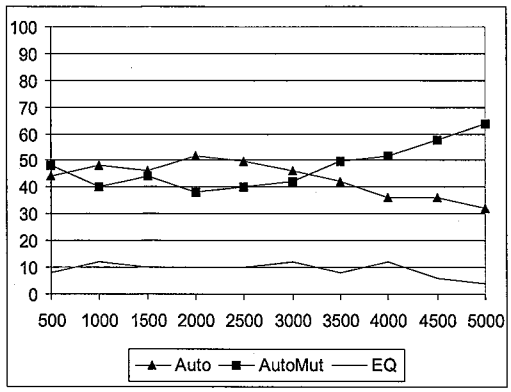
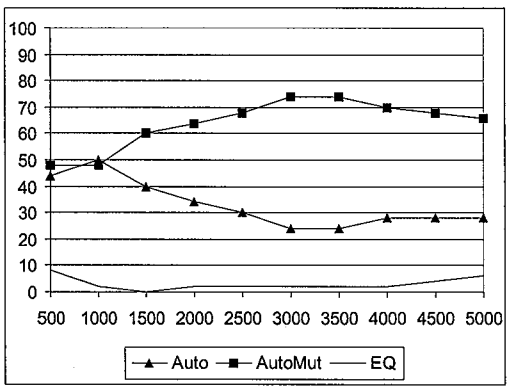


FIG. A.3.48 – TSXD avec et sans mutation - 5x20 sans écarts maximaux



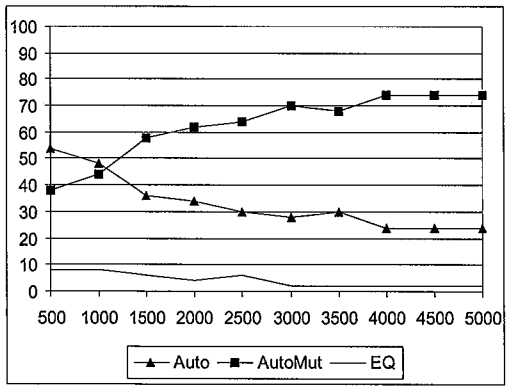
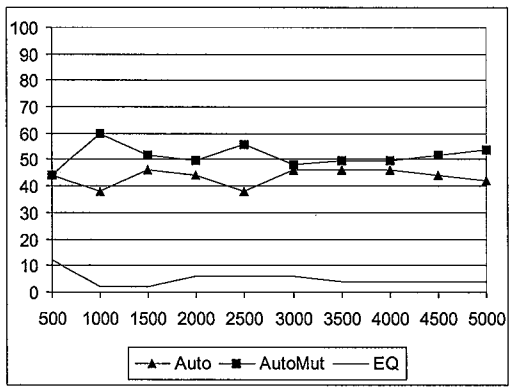


FIG. A.3.49 – TSXD avec et sans mutation - 5x20 avec écarts maximaux



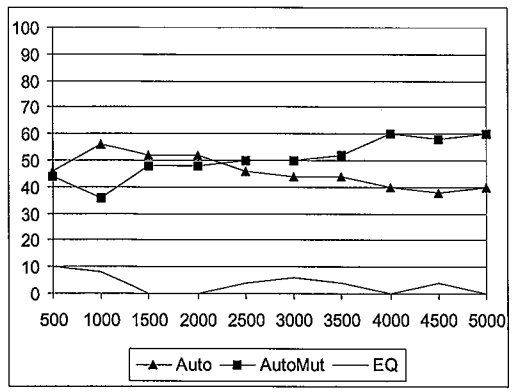


FIG. A.3.50 – TSXD avec et sans mutation - 5x20 sans attente

4 Annexe du chapitre 4

Cette section présente les résultats obtenus au cours des expériences décrites dans le chapitre 4, section 4.4.2.3, sur les jeux d'essai de Fondrevelle [Fon03], section 4.4.4.2 du même chapitre. Le tableau A.4.1, par exemple, contient les résultats obtenus pour chacune des 10 instances de la première famille des jeux d'essai. La colonne «C_{max}» est la valeur optimale sur l'ensemble des flowshops de permutation. La colonne «Noeuds» est le nombre de noeuds visités par la PSE de Fondrevelle. La colonne «Calculs» est le nombre moyen de calculs de dates d'exécution effectués par notre algorithme mémétique. La colonne «Pire» (resp. «Moyen» et «Meilleur») contient le pire (resp. moyen et meilleur) score obtenu par notre méthode au cours des 10 essais.

Jeu N°	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	576	87697	7831	590	587	586
2	553	83245	7531	558	554	553
3	576	11700	7753	587	577	576
4	622	95542	8294	626	623	622
5	570	40327	7570	576	572	570
6	562	13146	7297	563	562	562
7	599	3328	7577	601	599	599
8	586	116968	7868	591	589	588
9	599	24178	6832	602	599	599
10	594	18776	7082	598	596	594

TAB. A.4.1 – Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°1

Jeu N°	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	565	24	4779	565	565	565
2	535	1813	6495	536	536	535
3	575	159657	4836	576	576	575
4	518	18159	6172	521	521	518
5	527	8934	7047	530	530	527
6	524	10217	6902	529	529	524
7	502	29922	6219	503	503	502
8	507	56142	7357	510	510	507
9	488	43173	6595	491	491	488
10	487	35020	7204	491	491	487

TAB. A.4.2 – Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°2

Jeu N°	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	580	8366	6911	590	580	578
2	546	51899	6447	554	548	546
3	571	17425	6730	585	574	571
4	498	3119	5460	502	498	498
5	559	15135	6827	565	561	559
6	549	22861	5149	550	549	549
7	607	44711	6326	614	609	608
8	577	13540	5812	582	578	577
9	551	35288	5909	552	552	552
10	572	19436	6890	584	577	574

TAB. A.4.3 – Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°3

Jeu N°	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	566	13	3776	566	566	566
2	559	1555	5701	561	559	559
3	530	2463	4452	531	530	530
4	554	55808	4110	554	554	554
5	555	10441	4677	555	555	555
6	528	898	5298	534	529	528
7	574	21829	5656	578	576	574
8	554	2382	4917	554	554	554
9	618	792	3395	621	619	618
10	542	16662	4904	544	542	542

TAB. A.4.4 – Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°4

Jeu N°	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	464	5240	4958	464	464	464
2	539	125	3908	542	539	539
3	510	84384	4123	514	511	510
4	507	13	4745	507	507	507
5	539	24	4980	539	539	539
6	532	13	3665	532	532	532
7	511	159	3576	516	511	511
8	522	13	3607	522	522	522
9	520	107	4482	520	520	520
10	541	13	4103	541	541	541

TAB. A.4.5 – Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°5

Jeu N°	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	502	78	5010	502	502	502
2	537	13	3854	537	537	537
3	512	223	6653	523	514	512
4	499	78	4371	501	499	499
5	461	3632	6166	465	461	461
6	514	249	3598	519	515	514
7	482	105	4589	483	482	482
8	502	19089	6971	502	502	502
9	508	13	4796	508	508	508
10	505	13	3372	505	505	505

TAB. A.4.6 – Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°6

	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	807	362177	8151	826	814	807
2	869	1531963	8560	875	870	869
3	853	675478	7355	855	854	853
4	822	160903	8738	841	830	822
5	852	177444	8160	867	858	852
6	849	857147	7645	861	856	849
7	896	856930	7601	909	903	897
8	881	1427813	8097	894	887	881
9	761	112033	7416	765	761	761
10	861	572120	8805	873	866	861

TAB. A.4.7 – Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°7

Jeu N°	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	606	219453	7416	620	607	606
2	614	340744	7607	623	618	614
3	564	20252	7883	572	567	564
4	609	137793	7895	619	611	609
5	626	174472	8751	631	626	626
6	596	52398	8419	603	599	596
7	598	216500	7948	607	601	598
8	544	48691	8112	552	548	544
9	580	113092	7964	586	581	580
10	581	218675	7854	582	581	581

TAB. A.4.8 – Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°8

	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	754	202882	8828	758	754	754
2	768	181336	8582	776	773	768
3	825	354707	8452	832	828	825
4	780	62291	9426	787	781	780
5	765	236571	7855	775	771	767
6	835	473005	7985	837	835	835
7	802	397822	8331	808	803	802
8	787	299649	8632	791	787	787
9	794	56675	9422	814	798	794
10	758	68252	8625	766	759	758

TAB. A.4.9 – Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°9

	C_{max}	Noeuds	Calculs	Pire	Moyen	Meilleur
1	657	95385	7673	669	659	657
2	737	22961	8853	744	738	737
3	676	677367	8517	675	668	668
4	699	153824	8131	713	701	699
5	618	783098	7705	615	615	615
6	719	952600	8464	734	721	717
7	734	80629	8019	745	738	734
8	739	102734	7967	761	742	739
9	671	93032	8261	680	677	671
10	736	62279	6656	743	741	736

TAB. A.4.10 – Résultats détaillés des 10 instances de la famille Fondrevelle 2003 N°10

Bibliographie

[ABZ88] J. Adams, E. Balas et D. Zawack. The shifting bottleneck procedure for job shop scheduling. *Management Science*, 34 :391–401, 1988.

[BA03] F. Buscaylet et C. Artigues. A fast tabu search method for the job-shop problem with sequence-dependent setup times. *Proceedings of the 5th Metaheuristics International Conference (MIC2003), Kyoto*, 25-28 août 2003.

[Bak74] K.R. Baker. *Introduction to sequencing and scheduling.* John Willey & sons, Inc., 1974.

[BBV01] P. Baptiste, C. Bloch et C. Varnier. *Ordonnancement des lignes de traitement de surface*, chapter 9 de Ordonnancement de la production de F. Roubellat et P. Lopez, pages 259–293. IC2 Productique, Hermès Sciences Publications, 2001.

[BCKS01] P. Brucker, T.C.E. Cheng, S. Knust et N.V. Shakhlevich. Complexity results for shop problems with transportation delays. *OSM Reihe P*, 228, 2001.

[BDK+98] D. Brydges, F. Deppner, H. Künzli, K. Heuberger et R.D. Hersch. Application of a 3ccd color camera for colorimetric and densitometric measurements. *SPIE Proceedings*, 3300 :292–301, 1998.

[BEP+96] J. Blazewicz, K.H. Ecker, E. Pesch, G. Schmidt et J. Weglarz. *Scheduling Computer and Manufacturing Processes.* Springer - Verlag, Berlin, 1996.

[BHH99] P. Brucker, T. Hilbig et J. Hurink. A branch and bound algorithm for a single-machine scheduling problem with positive and negative time lags. *Discrete Applied Mathematics*, 94 :77–99, 1999.

[BJS94] P. Brucker, B. Jurisch et B. Sievers. A branch and bound algorithm for the job-shop scheduling problem. *Discrete Applied Mathematics*, 49 :107–127, 1994.

[BKM94] K.D. Boese, A.B. Kahng et S. Muddu. A new adaptative multi-start technique for combinatorial global optimization. *Operations Research Letters*, 16 :101–113, 1994.

[BMR88] M. Bartusch, R.H. Mohring et F.J. Radermacher. Scheduling project networks with resource constraints and time windows. *Annals of Operations Research*, 16 :201–240, 1988.

[BN95] K. Brinkmann et K. Neumann. Sequential and contraction b&b-based heuristics for rlp/max and rcpsp/max. *Report WIOR 471, Institut für Wirtschaftstheorie und Operations Research, Universität Karlsruhe*, 1995.

[BN96] K. Brinkmann et K. Neumann. Heuristic procedures for resource-constrained project scheduling with minimal and maximal time lags : the resource levelling and minimum project duration problems. *Journal of Decision Systems*, 5 :129–155, 1996.

[Car84] J. Carlier. *Problèmes d'ordonnancement à contraintes de ressources : algorithmes et complexité.* Thèse de doctorat d'état, Université Paris VI, 1984.

[CC88] J. Carlier et P. Chrétienne. *Problèmes d'ordonnancement - modélisation/complexité/algorithmes.* Mason, Paris, 1988.

[CGH96] I. Charon, A. Germa et O. Hudry. *Méthodes d'optimisation combinatoire.* Collection Pédagogique de Télécommunication, MASSON, 1996.

[COS02] A. Cesta, A. Oddi et S.F. Smith. A constraint-based method for project scheduling with time windows. *Journal of Heuristics*, 8 :109–136, 2002.

[CS97] C. Cheng et S. Smith. Applying constraint satisfaction techniques to job shop scheduling. *Annals of Operations Research*, 70 :327–357, 1997.

[Dav85] L. Davis. Job-shop scheduling with genetic algorithms. *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 136–140, 1985.

[DB03] E. Despontin et C. Briand. Ordonnancement avec prise en compte de multiples objectifs temporels : une heuristique couplée à une méthode d'exploration de voisinage. *Actes de la 4ème Conférence Francophone de MOdélisation et SIMulation (MOSIM), Toulouse*, pages 423–429, avril 2003.

[DDF02] L. Dupont et C. Dhaenens-Flipo. Minimizing the makespan on a batch machine with non-identical job sizes : an exact procedure. *Computers & Operations Research*, 29 :807–819, 2002.

[Del96] M. Dell'Amico. Shop problems with two machines and time laggs. *Operations Research*, 44 n°5 :777–787, 1996.

[Dep02] F. Deppner. Problème de job-shop avec contraintes d'écart minimal et maximal entre opérations pour des problèmes de taille industrielle. *Actes du 4ème congrès de la société française de recherche opérationnelle et d'aide à la décision (ROADEF 2002), Paris*, pages 107–108, 20-22 février 2002.

[Dep03] F. Deppner. Ordonnancement d'atelier avec contraintes d'écart minimal et maximal entre opérations. *Actes de la 4ème Conférence Francophone de MOdélisation et SIMulation (MOSIM'03), Toulouse*, pages 430–436, avril 2003.

[DGH96] F. Deppner, B. Gennart et R.D. Hersch. A probabilistic multi-cycle model for serving continuous streams from disks. *Proceedings of the IEEE International Conference on Multimedia Computing and Systems, Hiroshima*, pages 612–617, 17-21 juin 1996.

[DPST03] J. Dréo, A. Petrowski, P. Siarry et E. Taillard. *Métaheuristiques pour l'optimisation difficile.* Eyrolles, Paris, 2003.

[FM96] B. Freisleben et P. Merz. A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems. *Proceedings of the 1996 IEEE International Conference on Evolutionary Computation*, pages 616–621, 1996.

[FMM01] P.M. França, A. Mendes et P. Moscato. A memetic algorithm for the total tardiness single machine scheduling problem. *European Journal of Operational Research*, 132 :224–242, 2001.

[FNS01] B. Franck, K. Neumann et C. Schwindt. *Truncated branch-and-bound, schedule-construction and schedule-improvement procedures for resource-constrained project scheduling*, volume 23 of *OR Spektrum.* Springer Verlag, 2001.

[Fon03] J. Fondrevelle. Résolution exacte d'un problème d'ordonnancement en présence de contraintes d'attentes maximales entre opérations d'un même job : cas du flowshop de permutation. *Actes de la 4ème Conférence Francophone de MOdélisation et SIMulation (MOSIM), Toulouse*, pages 437–444, avril 2003.

[Gac99] L. Gacôgne. Multiple objective optimization of fuzzy rules for obstacles avoiding by an evolutionary algorithm with adaptative operators. *Proceedings of the 5th International Mendel Conference on Soft Computing (Mendel'99)*, pages 236–242, 1999.

[GG64] P.C. Gilmore et R.E. Gomory. Sequencing a one-state variable machine : a solvable case of the traveling salesman problem. *Operations Research*, 12 :675–679, 1964.

[GGRG85] J.J. Grefenstette, R. Gopal, B.J. Rosmaita et D.V. Gucht. Genetic algorithms for the traveling salesman problem. *Proceedings of the First International Conference on Genetic Algorithms and their Applications*, pages 160–168, 1985.

[Glo90] F. Glover. Tabu search : a tutorial. *Interfaces*, 20(4) :74–94, 1990.

[GMR02] J.F. Gonçalves, J.J. Mendes et M.G.C. Resende. A hybrid genetic algorithm for the job shop scheduling problem. *AT&T Labs Research Technical Report TD-5EAL6J (à paraitre également dans European Journal of Operational Research)*, septembre 2002.

[GNS88] J. Grabowski, E. Nowicki et C. Smutnicki. Block algorithm for scheduling operations in a job-shop system. *Przeglad Statystyczny XXXV*, 1 :67–80, 1988.

[GNZ86] J. Grabowski, E. Nowicki et S. Zdrzalka. A block approach for single machine scheduling with release dates dand due dates. *European Journal of Operational Research*, 26(2) :278–285, 1986.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search.* Optimization and Machine Learning. Addison Wesley, Reading, Mass, 1989.

[GT60] B. Giffler et G. L. Thompson. Algorithms for solving production scheduling problems. *Operations Research*, 8 :487–503, 1960.

[Hei03] R. Heilmann. A branch-and-bound procedure for the multi-mode resource-constrained project scheduling problem with minimum and maximum time lags. *European Journal of Operational Research*, 144 :348–365, 2003.

[HK01a] J.L. Hurink et J. Keuchel. Local search algorithms for a single-machine scheduling problem with positive and negative time lags. *Discrete Applied Mathematics*, 112 :179–197, 2001.

[HK01b] J.L. Hurink et S. Knust. Tabu search algorithms for job-shop problems with transport robot. *Memorandum N° 1579*, 2001.

[HM99] D. Holstein et P. Moscato. *Memetic algorithms using guided local search : a case study.* dans D. Corne, F. Glover, M. Dorigo, New Ideas in Optimization. McGraw-Hill, 1999.

[HMP85] A. Hodson, A.P. Muhlemann et D.H.R. Price. A micocomputer based solution to a practical scheduling problem. *Journal of Operational Research*, 36(10) :903–914, 1985.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems.* MIT Press, Cambridge, Mass, 1975.

[JM99] A.S. Jain et S. Meeran. Deterministic job-shop scheduling : past, present and future. *European Journal of Operational Research*, 113(2) :390–434, 1999.

[JM00] A.S. Jain et B. Rangaswamy S. Meeran. New and «stronger» job-shop neighbourhoods : a focus on the method of nowicki and smutnicki (1996). *Journal of Heuristics*, 6(4) :457–480, 2000.

[Joh54] S.M. Johnson. Optimal two- and three-stage production schedules with setup times included. *Naval Research Logistics Quarterly*, 1 :61–68, 1954.

[Kan76] A. Rinnooy Kan. *Machine scheduling problems : classification, complexity and computations.* The Hague : Martinus Nijhoff, 1976.

[KDHJ98] H. Künzli, F. Deppner, K. Heuberger et Y. Jiang. Mini-targets : a new dimension in print quality control. *SPIE Proceedings*, 3300 :286–291, 1998.

[KGV83] S. Kirkpatrick, C.D. Gelatt et M.P. Vecchi. Optimization by simulated annealing. *Science*, 220 :671–680, 1983.

[Kra98] S.A. Kravchenko. A polynomial algorithm for a two-machine no-wait job-shop scheduling problem. *European Journal of Operational Research*, 106 :101–107, 1998.

[KS00] Natalio Krasnogor et Jim Smith. A memetic algorithm with self-adaptive local search : Tsp as a case study. *Proceedings of the 2000 International Genetic and Evolutionary Computation Conference (GECCO2000)*, pages 987–994, 2000.

[LAL92] P.J.M. Van Laarhoven, E.H.L. Aarts et J.K. Lenstra. Job shop scheduling by simulated annealing. *Operations Research*, 40(1) :113–125, 1992.

[LC01] C.Y. Lee et Z.L. Chen. Machine scheduling with transportation considerations. *Journal of Scheduling*, 4(1) :3–24, 2001.

[LR01] P. Lopez et F. Roubellat. *Ordonnancement de la production.* IC2 Productique, Hermes Sciences Publications, Paris, 2001.

[MB03] M.A. Manier et C. Bloch. A classification of hoist scheduling problems. *Internationan Journal of Flexible Manufacturing Systems*, 15(1) :37–55, 2003.

[Mit58] L.G. Mitten. Sequencing n jobs on two machines with arbitrary time lages. *Management Science*, 5 :293–298, 1958.

[Mos89] P.A. Moscato. On evolution, search, optimization, genetic algorithms and martial arts : towards memetic algorithms. *Caltech Concurrent Computational Program, Technical Report 826, Caltech Pasadena, Californie*, 1989.

[Mos99] P.A. Moscato. Memetic algorithms : a short introduction. *dans D. Corne and F. Glover and M. Dorigo, New Ideas in Optimization*, pages 219–234, 1999.

[NEH83] M. Nawaz, E.E. Enscore et I. Ham. A heuristic algorithm for the m-machine, n-job flow shop sequencinq problem. *Omega*, 11, 1983.

[NS96a] E. Nowicki et C. Smutnicki. A fast taboo search algorithm for the job shop problem. *Management Science*, 42(6) :797–813, 1996.

[NS96b] E. Nowicki et C. Smutnicki. A fast tabu search algorithm for the permutation flow-shop problem. *European Journal of Operational Research*, 91(1) :160–175, 1996.

[NSZ02] K. Neumann, C. Schwindt et J. Zimmermann. Recent results on ressource-constrained project scheduling with time windows : models, solution methods, and applications. *CEJOR*, 10 :113–148, 2002.

[PA01] M.C. Portmann et M.A. Aloulou. Data oriented genetic operators for one-machine scheduling problem. page 1184, San Francisco, 9-11 juillet 2001.

[PAK02] M.C. Portmann, H. Amet et M.Y. Kovalyov. Dominance properties for permutation flow shop problems. 2002.

[PK02] P. Pujo et J.P. Kieffer. *Fondements du pilotage des systèmes de production.* IC2 Productique, Hermes Sciences Publications, Paris, 2002.

[Por87] M.C. Portmann. Méthodes de décomposition spatiale et temporelle en ordonnancement de la production. Université Nancy 1, 18 septembre 1987.

[Por96] M.C. Portmann. Genetic algorithms and scheduling : A state of the art and some propositions. pages I–XIV, Mons, Belgium, septembre 9-11 1996.

[PSW91] C.N. Potts, D.B. Shmoys et D.P. Williamson. Permutation vs. non-permutation flow shop schedules. *Operations Research Letters*, 10 :281–284, 1991.

[PT03] M. Pirlot et J. Teghem. *Résolution de problèmes de RO par les métaheuristiques.* IC2 Productique, Hermes Sciences Publications, Paris, 2003.

[PV00] M.C. Portmann et A. Vignier. Performances's study on crossover operators keeping good schemata for some scheduling problems. pages 331–338, Las Vegas, 10-12 juillet 2000.

[Ree95] C.R. Reeves. A genetic algorithm for flowshop sequencing. *Computers and Operations Research*, 22 :5–13, 1995.

[Ree98] C.R. Reeves. Landscapes, operators and heuristic search. *Annals of Operations research*, 1998.

[RH98] B. De Reyck et W. Herroelen. A branch-and-bound procedure for the resource-constrained project scheduling problem with generalized precedence relations. *European Journal of Operational Research*, 111 :152–174, 1998.

[Röc84] H. Röck. The three-machine no-wait flowshop is np-complete. *Journal of ACM*, 31 :336–345, 1984.

[Roy70] B. Roy. *Algèbre moderne et théorie des graphes - Tome 2*, chapter 8. Dunod, 1970.

[RS64] B. Roy et B. Sussman. Les problèmes d'ordonnancement avec contraintes disjonctives. *SEMA, Note DS No. 9bis*, 1964.

[RS99] D. Rebaine et V.A. Strusevich. Two machine open shop scheduling with special transportation times. *Journal of the Operational Research Society*, 50 :756–764, 1999.

[SC79] S. Sahni et Y. Cho. Complexity of scheduling shops with no wait in process. *Mathematics of Operations Research*, 4 :448–457, 1979.

[Sch98] C. Schwindt. A branch and bound algorithm for the resource-constrained project duration problem subject to temporal constraints. *rapport technique WIOR-544*, 1998.

[SGM03] N.E.H. Saadani, A. Guinet et M. Moalla. Three stage no-idle flow-shops. *Computers & Industrial Engineering*, 44 :425–434, 2003.

[SKD95] A. Sprecher, R. Kolisch et A. Drexl. Semi-active, active, and non-delay schedules for the resource -constrained scheduling problem. *European Journal of Operational Research*, 80 :94–102, 1995.

[Str99] V.A. Strusevich. A heuristic for the two machine open-shop scheduling problem with transportation times. *Discrete Applied Mathematics*, 93 :287–304, 1999.

[Svi03] M. Sviridenko. Makespan minimization in no-wait flow shops : a polynomial time approximation scheme. *SIAM Journal of Discrete Mathematics*, 16 :313–322, 2003.

[Szw83] W. Szwarc. Flow shop problems with time lags. *Management Science*, 29 n°4 :477–481, 1983.

[WBWH02] J.P. Watson, L. Barbulescu, L.D. Whitley et A.E. Howe. Contrasting structured and random permutation flow-shop scheduling problems : search-space topology and algorithm performance. *INFORMS Journal on Computing*, 14(2) :98–123, 2002.

[Wie72] D.A. Wiesmer. Solution of the flowshop-scheduling problem with no intermediate queues. *Operations Research*, 20 :689–697, 1972.

[WSF89] Y. Whitley, T. Starkweather et D. Fuquay. Scheduling problems and travelling salesman : the genetic edge recombination operators. *Proceedings of the 3rd International Conference on Genetic Algorithms and their Applications*, pages 133–140, 1989.

[WZ03] L. Wang et D.Z. Zheng. An effective hybrid heuristic for flow shop scheduling. *International Journal of Advanced Manufacturing Technology*, pages 38–44, 2003.

[YN96] T. Yamada et R. Nakano. *Job-Shop Scheduling by Simulated Annealing Combined with Deterministic Local Search.* Meta-heuristics : theory & applications. Kluwer academic publishers, MA, USA, 1996.

[YR98] T. Yamada et C.R. Reeves. Solving the c_{sum} permutation flowshop scheduling problem by genetic local search. *Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC98)*, pages 230–234, 1998.

Index

C_{max}, 20
C^*_{max}, 20

algorithme
    AOS, 37
    AOS-G-OS, 44
    AOS-G-RP, 44
    AOS-MD, 39
    ARP, 36
    ARP-G-OS, 44
    ARP-G-RP, 44
    ARP-MD, 38
algorithme de construction, 24, 36
    à base de règle de priorité, 36
    ordre strict, 37
algorithme génétique, 69
algorithme mémétique, 102
autodétermination, 81

bloc, 95
bloc critique, 95

chemin critique, 94
contrainte temporelle, 5
    écart maximal, 5
    écart minimal, 5
    sans attente, 6, 51
    temps d'attente maximal, 5
    temps d'attente minimal, 5
critère régulier, 23

entrelacement (mesure d'), 52

graphe
    conjonctif, 20
    disjonctif, 20
    disjonctif arbitré, 20
    potentiels-tâches, 20
grappe, 42

horizon de planification, 44
hypothèse H0, 29, 45
hypothèse H1, 45
hypothèse H2, 45
hypothèse H3, 47

inégalité de potentiels, 18
Incotec, 5

liste d'ordre strict, 37, 51

MACSI, 5
makespan, 15, 20
mesure ε, 52

opérateur de croisement, 71
    1X, 72
    2XC, 73
    2XL, 73
    ERX, 74
    TSX, 76
    TSXD, 76
    TSXW, 75
ordonnancement, 4
    actif, 23
    sans délai, 23
    semi-actif, 21, 22
    valide, 21

PharmaCorp, 13
profil de capacité, 18
PSE, 104

recherche locale, 26

saute-mouton (phénomène du), 40

tension sur les écarts maximaux, 51

voisinage, 94
    Nowicki et Smutnicki, 97
    Van Laarhoven et al, 95

INSTITUT NATIONA
POLYTECHNIQU
DE LORRAINI

AUTORISATION DE SOUTENANCE DE THESE DU DOCTORAT DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

o0o

VU LES RAPPORTS ETABLIS PAR :
Monsieur Lionel DUPONT, Professeur, Ecole des Mines d'Albi-Carnaux, Albi

Monsieur Michel GOURGAND, Professeur, ISIMA, Aubière

Le Président de l'Institut National Polytechnique de Lorraine, autorise :

Monsieur DEPPNER Freddy

NANCY BRABOIS
2, AVENUE DE LA
FORET-DE-HAYE
BOITE POSTALE 3
F - 54501
VANDOEUVRE CEDEX

à soutenir devant un jury de l'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE, une thèse intitulée :

« **Ordonnancement d'atelier avec contraintes temporelles entre opérations** »

en vue de l'obtention du titre de :

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE LORRAINE

Spécialité : « **Informatique** »

Fait à Vandoeuvre, le 20 avril 2004
Le Président de l'I.N.P.L.
L. SCHUFFENECKER

TEL. 33/03.83.59.59.5
FAX. 33/03.83.59.59.5

Cette thèse traite des problèmes d'ordonnancement d'atelier avec contraintes temporelles d'écart minimal et maximal entre opérations. Une contrainte d'écart minimal stipule qu'une opération ne peut débuter son exécution avant qu'un laps de temps ne se soit écoulé depuis le début ou la fin d'une opération précédente. Une contrainte d'écart maximal impose au contraire que l'opération débute son exécution dans un délai imparti.
Bien que de nombreux cas industriels soient soumis à ce genre de contraintes, la littérature sur le sujet reste réduite. Une explication possible tient au fait qu'en présence d'écarts maximaux, le problème de la construction d'une solution valide est NP-complet, dans les cas les plus généraux, même pour une machine. Pour obtenir une solution faisable nous généralisons les algorithmes de construction à base de règles de priorité ou de liste d'ordre strict. Nous étudions un algorithme utilisant une partition des opérations en sous-ensembles définis à partir des composantes fortement connexes du graphe conjonctif. Nous démontrons que, sous certaines hypothèses, nos algorithmes convergent et produisent des ordonnancements actifs.
Qui dit ordonnancement sous-entend optimisation d'un objectif. Nous nous intéressons à l'optimisation de la durée totale (makespan) pour des problèmes de machines en série (flowshop) avec contraintes d'écart minimal et maximal, organisation d'atelier rencontrée le plus fréquemment au cours de nos études de cas. Nous développons une approche génétique dont l'originalité consiste à travailler avec un pool d'opérateurs de croisement et à utiliser une méthode d'autodétermination qui sélectionne l'opérateur le plus approprié à la problématique et à l'état courant de la population. Nous hybridons notre méthode avec un algorithme d'optimisation locale de plus forte pente après avoir étudié et généralisé les voisinages en présence d'écarts maximaux.

Scheduling problems with minimal and maximal time lag constraints

In this thesis we study scheduling problems with minimal and maximal time lags between operations. A minimal time lag specifies that an operation can't start its execution before a given amount of time has elapsed since the beginning or the end of a previous operation. Conversely, with a maximal time lag, the operation must start its execution within a given delay. Despite many industrial processes are subjected to such time constraints, publications on the topic remain poor. A possible explanation could be that the problem of designing a feasible solution with maximal time lags is NP-complete for some cases including single machine problems. In order to obtain a feasible solution, we generalize various priority rule construction algorithms. We design an algorithm using a decomposition of the set of operations into clusters defined as the strong components of the conjunctive graph. We demonstrate that under some realistic hypotheses, our algorithms converge and generate active schedules. In a second part, we study the minimization of the makespan for flow shop problems. For this purpose, we develop a genetic algorithm. The main feature is that we are working with a pool of crossover operators and a selection method that automatically chooses the most adapted operator according to the problem and the current state of the population. We finally couple our genetic algorithm with a local search procedure.

Mots-clés : ordonnancement, écart minimal, écart maximal, faisabilité, flowshop, makespan, algorithme génétique, recherche locale
Keywords : scheduling, minimal time lag, maximal time lag, feasibility, flow shop, makespan, genetic algorithm, local search

LORIA - INRIA Lorraine
Campus Scientifique BP 239 - 54506 Vandoeuvre-Lès-Nancy CEDEX