Ordonnancement distribué réactif

pour

un atelier d'assemblage de type flow shop

THÈSE

présentée à la Faculté des sciences, pour obtenir

le grade de Docteur ès sciences, par

Thouraya DAOUAS

UNIVERSITÉ DE NEUCHÂTEL
Institut d'Informatique et d'Intelligence Artificielle
rue Émile-Argand 11
2007 Neuchâtel, Suisse

IMPRIMATUR POUR LA THÈSE

Ordonnancement distribué réactif pour un atelier d'assemblage de type flow-shop

de Mme Thouraya Daouas

UNIVERSITÉ DE NEUCHÂTEL

FACULTÉ DES SCIENCES

La Faculté des sciences de l'Université de Neuchâtel sur le rapport des membres du jury,

Messieurs J.-P. Müller, P.-J. Erard,
E. Verdebout (EPFL, Lausanne) et C. Proust (Tours).

autorise l'impression de la présente thèse.

Neuchâtel, le 24 juin 1996

Le doyen:

R. Dändliker

قل هل يستوي الّذين يعلمون و الّذين لا يعلمون، إنّما يتذكّر أولو الألباب.

قرآن كريم

Ceux qui savent et les ignorants sont-ils égaux, les hommes doués d'intelligence sont les seuls à réfléchir.

Coran

A ceux qui ont su si souvent me donner leur aide en silence...
A ceux qui me donnent leurs coeurs sans retour...
A mes parents.

Remerciements

Je tiens à présenter ma gratitude envers toutes les personnes qui m'ont encouragée et aidée dans ce travail de thèse.

Le Professeur Jean-Pierre Müller (IIIA, Neuchâtel), mon directeur de thèse pour son aide, surtout dans les moments critiques.
Le Professeur Christian Proust (EIII, Tours), d'une part pour avoir accepté d'être parmi les membres du jury, et d'autre part pour son encouragement.
Le Docteur Eric Verdebout (CIMPACT, Lausanne), pour sa collaboration ainsi que pour son enthousiasme durant les nombreuses séances de travail à l'IMT.
Le Professeur Pierre-Jean Erard (IIIA, Neuchâtel), pour avoir accepté d'être parmi les membres du jury et aussi pour son soutient moral.
Le Docteur Khaled Ghédira (IIIA, Neuchâtel), pour son apport scientifique.

Je ne saurais jamais exprimer mes sentiments d'amitié, de sympathie et de respect pour mes deux collègues et amis Madame Gianfranca Cerrito et Monsieur Miguel Rodriguez. Ils ont beaucoup fait pour moi ne serait-ce que par leur présence et leur bonne humeur.

De même, je n'oublierai jamais le rôle très important de mes chers parents dans toutes les réussites que j'ai eues. J'espère que cette thèse pourra compenser en partie les moments difficiles qu'ils ont passés avec moi.

Enfin, je présente toute mon affection à tous les membres de ma famille et à mes amis.

Table des matières

Chapitre 1

Introduction générale

1.1 Contexte et domaine de recherche

Récemment, les ateliers flexibles ont constitué un nouveau domaine d'application. C'est une technologie nouvelle qui a été introduite afin d'améliorer l'efficacité de la production dans les ateliers en mettant en valeur leur flexibilité. Un atelier flexible peut être défini comme étant un système intégré de fabrication. Il consiste en des machines flexibles dont les composants sont sous contrôle informatique.

Les systèmes d'ateliers flexibles doivent être conçus avec beaucoup de précautions pour:

- prévoir les problèmes qui peuvent arriver avec les pannes des machines et des outillages, ou encore avec les matières premières;
- récupérer le capital considérable d'investissement.

Différents problèmes doivent être résolus dans de tels environnements, tels que la conception et la planification de l'atelier et l'ordonnancement des tâches sur les différentes ressources. Par conséquent, le facteur vital qui influence les solutions pour l'ordonnancement est le matériel de l'atelier; spécialement le type des machines existantes et les moyens de changement d'outils.

Historiquement, l'ordonnancement a été traité comme un problème prévisionnel et sans possibilité de modifications; à présent, il est généralement reconnu qu'il a besoin d'être vu comme une partie d'un processus dynamique dans l'atelier qui est sujet à des événements externes ou imprévus.

Les problèmes d'ordonnancement appartiennent à la classe des problèmes NP-complets, vu que les algorithmes de résolution utilisés nécessitent un temps exponentiel en fonction de la taille du problème. Pour cette raison, il est important de

trouver les méthodes et les techniques appropriées afin de résoudre le problème d'ordonnancement dans des temps raisonnables et aboutir à des solutions de qualité.

Les premières méthodes et techniques utilisées sont celles de l'optimisation combinatoire appartenant à la recherche opérationnelle, à savoir la programmation linéaire. Ces techniques représentent des méthodes exactes et garantissent, généralement, une solution optimale au détriment du temps exponentiel nécessaire pour atteindre la solution.

Par la suite, des méthodes liées à l'intelligence artificielle sont apparues. Ce sont des méthodes approchées qui représentent de nouvelles stratégies de résolution et qui consistent en des constructions progressives, des recherches par voisinages ou par décompositions, des résolutions par relaxation de contraintes, etc. Dans ces approches une solution sous-optimale, trouvée en un temps raisonnable, est acceptée.

Pour présenter notre problème d'une manière simplifiée, nous disons qu'étant donné un ensemble de ressources dans l'atelier, nous disposons de Demandes Client (CDs) à ordonnancer. Chacune de ces demandes clients possède une date limite de livraison et est composée d'un graphe d'actions à placer sur les ressources en respectant les relations de précédences qui existent entre elles.

Chacune des actions a besoin d'un outil déterminé pour son exécution. Ainsi, un temps de réglage relatif à chacune des ressource est nécessaire pour les changements d'outils lors de l'exécution des actions.

Nous proposons le système MARSA pour *Multi-Agent Reactive system for Scheduling Assembly*, dans le cadre de la résolution d'un problème d'ordonnancement dans un atelier flexible. Il s'agit d'un système d'ordonnancement qui regroupe une modélisation distribuée des systèmes multi-agents avec une technique de recherche stochastique, celle du recuit simulé.

- la modélisation multi-agents est basée sur des entités intelligentes, appelées agents, qui coopèrent dans des sous-domaines différents. Ces entités prennent des décisions à travers des négociations en utilisant une connaissance spécifique du domaine. Ces connaissances sont distribuées entre les agents.

  Nous avons appliqué cette modélisation à notre problème en définissant deux classes principales d'agents: la classe des agents Demande Client et la classe des agents Ressources. Chaque type d'agent a un comportement différent afin d'atteindre sa propre satisfaction.

- le recuit simulé représente une simulation du recuit thermique, issu du domaine de la physique statistique [Laarhoven et Aarts, 1992a]. Il s'agit d'un outil d'optimisation combinatoire qui est utilisé pour guider la recherche vers une solution optimale dans le cas idéal.

Nous avons appliqué cette méthode de résolution en plaçant un recuit simulé au niveau de chaque agent Ressource. De même, nous avons défini une fonction objectif (fonction coût) locale à chacun de ces agents qui réunit les critères à optimiser . Cette fonction intervient dans la décision pour atteindre la solution.

Nous avons choisi d'utiliser ce type de système afin de contribuer dans la résolution des problèmes d'ordonnancement d'ateliers de type *flow shop* de permutation.

1.2 Notre contribution

Par l'accomplissement de ce travail, notre contribution se présente sous différents aspects:

- une contribution liée à l'**aspect temporel**. Partant de l'approche multi-agents présentée dans [Ghédira, 1993], il s'agit de tenir compte de deux types de contraintes temporelles.

  Les contraintes temporelles absolues, à savoir, la date de livraison imposée de chacune des DCs et le temps de réglage nécessaire pour le changement d'outils sur chacune des ressources.

  Les contraintes temporelles relatives, à savoir, les relations de précédences qui existent entre les actions d'une même DC à placer sur les ressources;

- une contribution liée à la **combinaison des systèmes multi-agents avec le recuit simulé**. Chaque agent Ressource est muni de son propre recuit simulé, par conséquent, il est responsable de la prise des décisions concernant son état local. Ainsi, tous les agents participent à la recherche de la solution finale.

  Par ailleurs, la décision d'un agent Ressource peut modifier l'état local d'un autre agent Ressource. Il s'agit là d'un traitement engendré par les systèmes multi-agents distribués. Un état final global est un état qui reflète tous les états locaux des agents Ressources;

- une contribution liée à l'**optimisation** consiste à formuler une fonction objectif dynamique, du fait qu'elle donne la possibilité d'ajouter les caractéristiques que nous espérons avoir dans la solution finale en donnant une pénalité à chacune d'elles (voir section 5.2);

- une contribution liée à l'**aspect dynamique** consiste à tenir compte des perturbations qui peuvent avoir lieu et qui ont pour conséquences, soit la modification des données du problème (l'ajout d'une nouvelle DC), soit celle

de la solution trouvée (les dates effectives de début et de fin d'exécution ainsi que les dates de début et de fin d'enrayage ou de réglage);

- une contribution liée à la **programmation** qui consiste au développement d'un prototype du modèle MARSA lié au simulateur d'assemblage SIMAS, qui représente l'atelier d'exécution. Une interface utilisateur est aussi intégrée dans ce prototype permettant à l'utilisateur d'introduire ses préférences concernant, et les données, et les solutions trouvées.

1.3 Plan de la thèse

Cette thèse est composée de dix chapitres regroupés en trois parties:

1. Etat de l'art.
2. Modèle MARSA.
3. Prototype, application à l'assemblage.

Chacune des trois parties commence par sa propre introduction. De même, une introduction générale et une conclusion générale sont réservées à la totalité de la thèse.

Partie I: Etat de l'art

Cette partie est consacrée à la présentation des définitions et du vocabulaire utilisés dans le traitement des problèmes d'ordonnancement. Elle a aussi pour but de donner une vue globale sur les travaux, utilisant différentes techniques, traitant le problème d'ordonnancement et qui ont été proposés pour sa résolution.

Partie II: Modèle MARSA

Cette partie principale, présente le modèle que nous proposons avec ses deux aspects statique et dynamique.

- pour l'aspect statique, le modèle est présenté en détail et une série d'expérimentations effectuées et illustrées;
- pour l'aspect dynamique, les différents traitements dynamiques sont spécifiés et un exemple réel est appliqué au modèle;

Partie III: Prototype, application à l'assemblage

Cette partie s'occupe du prototype que nous avons développé et qui est formé du module ordonnanceur MARSA et d'une interface utilisateur. Cette interface a pour but de faciliter la communication entre MARSA et le simulateur d'assemblage SIMAS afin de simuler un traitement ordonnancement-exécution réel de l'atelier.

Partie I

Etat de l'art

Introduction

La planification et l'ordonnancement peuvent être les processus de spécification, de formulation et de conception des activités à exécuter, des objets à arranger, etc. dans le but de réaliser quelques objectifs établis.

En Intelligence Artificielle (IA), la planification est le processus de génération d'une solution constituée des scénarios possibles pour un problème spécifique, alors que l'ordonnancement est un processus de choix d'un scénario d'une procédure pour réaliser un objectif particulier, en spécifiant le temps de séquencement pour chaque élément dans la procédure.

Les problèmes typiques d'ordonnancement sont l'ordonnancement de systèmes de production, l'emploi du temps dans les chemins de fer, etc. Ils ont été longtemps traités à l'aide de méthodes appelées exactes, issues de la recherche opérationnelle, qui sont caractérisées par l'obtention d'une solution optimale avec un temps de traitement très élevé.

A présent, l'ordonnancement occupe un champ très large dans les activités de plusieurs domaines de production et de recherche expérimentale. Par conséquent, pour le résoudre avec des temps de traitement raisonnables, des méthodes approximatives ou approchées liées à l'IA ont été proposées. Ces méthodes sont guidées soit par l'exploitation des connaissances propres au problème soit par des heuristiques.

Suivant la façon de modéliser un problème d'ordonnancement, un espace de recherche, ou espace d'états, est déterminé. C'est l'ensemble de tous les états (les solutions optimales ou non) et des sous-états possibles (solutions incomplètes).

La recherche dans l'espace d'état est réalisée en se déplaçant d'un état à un autre jusqu'à atteindre l'état ou la solution souhaitée, celle qui satisfait les contraintes du problème. L'obtention de cette solution est fortement influencée par la manière et par la méthode utilisée pour se déplacer à travers ces états.

Dans le domaine industriel, les ateliers de production sont souvent confrontés aux problèmes d'organisation et de pilotage en temps réel. Ces problèmes sont liés à la réalisation d'un plan prévisionnel qui assure un équilibre entre la charge de travail et les capacités de production. Ceci afin de:

- respecter les délais prévus ou imposés par les clients;
- utiliser au mieux les ressources de l'atelier afin d'assurer sa rentabilité;
- répondre aux événements qui surviennent quotidiennement au cours de la production.

La fonction "ordonnancement" dans un atelier est traditionnellement l'une des composantes de la gestion de production. Elle a pour but d'organiser le travail de telle manière à satisfaire les demandes, par le respect des délais et l'utilisation optimale des ressources.

Un nouveau domaine d'application pour la théorie de l'ordonnancement est celui des systèmes d'ateliers flexibles. Cette nouvelle technologie a été introduite afin d'améliorer l'efficacité de l'atelier.

C'est dans ce cadre que nous abordons l'ordonnancement. Nous présentons ainsi un état de l'art de différentes méthodes proposées pour résoudre le problème d'ordonnancement.

Cette partie est composée de troix chapitres principaux. Dans le chapitre 2, nous présentons l'environnement lié à l'ordonnancement dans un atelier avec les définitions et les représentations nécessaires. Dans le chapitre 3, nous citons une liste des principales méthodes et approches proposées. Enfin le chapitre 4 est réservé aux fondements et motivations de notre modèle.

Chapitre 2

Ordonnancement dans un atelier

Si l'ordonnancement n'intervient pas dans la détermination des moyens d'un atelier, son rôle est essentiel dans l'application de ces principes et de ces règles. Il a la charge de préparer le travail et de contrôler son exécution conformément aux normes fixées par l'atelier.

Cependant, l'ordonnancement peut jouer un rôle très différent selon l'atelier pour lequel il est utilisé, la branche professionnelle à laquelle l'atelier appartient, la clientèle qui lui est destinée et la conception du marché [GOTHA, 1993].

Dans ce chapitre nous présentons des définitions et des représentations liées au problème d'ordonnancement dans un atelier. De plus, nous mettons l'accent sur les ateliers de type *flow shop* de permutation qui représentent l'environnement que nous avons adopté pour notre travail expérimental.

2.1 Définitions et terminologie

Etant donné un ensemble de tâches à effectuer sur des ressources, ordonnancer les tâches revient à programmer l'ordre de leur exécution en fixant leurs dates de début d'exécution sur les ressources requises, tout en respectant des contraintes données, et éventuellement pour optimiser un objectif.

Par conséquent, un plan d'ordonnancement est l'ensemble des agendas des ressources qui fixent les dates de début et de fin des tâches à effectuer.

Un problème d'ordonnancement est caractérisé par deux entités de base: la tâche et la ressource.

2.1.1 Tâche

Une tâche est une quantité de travail considérée comme un tout indivisible. Elle peut correspondre à des notions très différentes: montage d'une charpente, usinage d'une pièce, etc. A chaque tâche, il peut être attribué une localisation dans le temps par:

- une durée requise pour son exécution (*processing time*);
- une date de début (*start time*);
- une date de fin (*completion time*).

Nous signalons que le premier attribut est connu au départ alors que les deux derniers résultent de l'ordonnancement.

La notion de préemption exprime le fait qu'une tâche peut être interrompue par une autre tâche lors de son exécution sur une ressource.

2.1.2 Ressource

Une ressource est un moyen technique ou humain disponible en quantité limitée ou renouvelable nécessaire pour l'exécution d'une tâche. Elle peut être fixe (machine, alimentateur, etc.) ou encore mobile (palette, chariot, etc).

Nous distinguons deux types de ressources:

- une ressource est dite renouvelable si elle redevient disponible après chaque exécution d'une tâche. Nous citons comme exemple la machine, le personnel, etc;
- une ressource est dite consommable si, après l'exécution d'une tâche, elle n'est plus disponible pour le reste des tâches à effectuer. Nous citons comme exemple l'argent, les matières premières, les pièces, etc.

2.1.3 Contraintes

Dans un problème d'ordonnancement il y a généralement des contraintes à respecter. Elles peuvent être de trois types:

Contraintes temporelles: Elles expriment les relations de précédences entre les différentes tâches [Allen, 1981] et, étant donné deux tâches A et B, elles peuvent être formulées de plusieurs façons:

- l'exécution de B ne peut commencer qu'après la fin de celle de A;
- l'exécution de B ne peut commencer qu'après le début de celle de A;

- l'exécution de B ne peut commencer qu'à un temps donné après la fin de celle de A;
- l'exécution de A doit commencer après une date d_1;
- l'exécution de A doit se terminer avant une date d_2;
- etc.

Contraintes cumulatives: Elles expriment les capacités des moyens nécessaires à l'exécution des tâches. En d'autres termes elles posent la condition que la quantité globale de la ressource, utilisée à un instant bien déterminé, doit être inférieure à la quantité disponible à ce même instant.

Contraintes disjonctives: Elles expriment le fait que deux tâches utilisent une même ressource disponible. Par conséquent ces deux tâches ne peuvent pas avoir un intervalle de temps commun sur la ressource.

2.1.4 Critères d'optimisation

Dans un problème d'ordonnancement optimal, mis à part le fait que les tâches sont affectées aux ressources en leur attribuant des dates de début et de fin, l'ordre d'exécution des tâches joue un rôle très important. L'importance de cet ordre dépend des critères adoptés pour juger le degré d'optimalité de l'ordonnancement en question.

Notons c_j la date de fin de la tâche j (*completion time*) et d_j sa date limite d'exécution (*deadline*). $L_j = c_j - d_j$ est le retard algébrique (*lateness*). Ci-après nous citons une liste non-exhaustive des critères d'optimisation d'un ordonnancement:

- la durée totale de l'ordonnancement (*makespan*) définie par:

$$C_{max} = Max_{j=1}^{nt} c_j$$

où nt est le nombre de tâches;

- la date de fin de tâche moyenne définie par:

$$C_{moy} = \frac{1}{nt} * \sum_{1}^{nt} c_j$$

- le retard algébrique maximum défini par:

$$L_{max} = Max_{j=1}^{nt} L_j$$

- le retard algébrique moyen défini par:

$$L_{moy} = \frac{1}{nt} * \sum_{1}^{nt} L_j$$

- le retard maximum (*tardiness*) défini par:

$$T_{max} = Max_{j=1}^{nt} T_j$$

où $T_j = Max(0, L_j)$;

- le retard moyen défini par:

$$T_{moy} = \frac{1}{nt} * \sum_{1}^{nt} T_j$$

- la somme pondérée des retards définie par:

$$\sum_{1}^{nt} w_j * T_j$$

où w_j étant le poids de la tâche j;

- le nombre de tâches en retard défini par:

$$\sum_{1}^{nt} U_j$$

où $U_j = \{0, 1\}$;

- la somme pondérée des tâches en retard définie par:

$$\sum_{1}^{nt} w_j * U_j$$

- le temps de réglages entre les tâches (*setup time*) défini par:

$$nr * dr_i$$

où nr est le nombre de réglages et dr_i est la durée de réglage d'une ressource i.

Nous mentionnons que la durée totale de l'ordonnancement est le critère le plus répandu dans la littérature [Taillard, 1993].

2.1.5 Aspects de l'ordonnancement

Un problème d'ordonnancement peut avoir deux aspects différents: l'aspect statique et l'aspect dynamique.

Aspect statique: Il s'agit de générer les agendas des ressources sur la base de données statiques: nombre de tâches et de ressources fixes. Ainsi chacune des tâches a une affectation statique, présentée sous la forme de dates fixes de début et de fin, sur une ressource particulière.

L'inconvénient d'un ordonnancement statique est qu'il n'est pas flexible vis-à-vis des nouvelles informations qui peuvent apparaître pour modifier, soit les données, soit l'état des ressources.

Aspect dynamique: Il s'agit, compte tenu de l'avancement de l'exécution des tâches et de l'état des ressources, de réagir aux événements qui peuvent arriver et de prendre des décisions au cours de l'exécution. Ainsi, les informations dont dispose l'ordonnancement sont toujours relatives à un instant donné. De plus, au cours de l'exécution des tâches sur lès ressources, de nouvelles tâches peuvent apparaître et des ressources peuvent tomber en panne. Les caractéristiques de certaines tâches ou de certaines ressources peuvent donc être modifiées.

L'inconvénient d'un ordonnancement dynamique est que la recherche d'un ordonnancement optimal perd de son intérêt car si les données changent constamment l'ensemble des solutions possibles change également.

Partant de l'aspect dynamique, nous définissons le réordonnancement. Il s'agit de la modification des dates et des affectations pour répondre aux imprévus de l'environnement tout en préservant la stabilité de la solution [Zweben et al., 1993].

2.1.6 Ateliers

Dans un problème d'ordonnancement d'atelier, les tâches ou opérations sont regroupées en entités appelées lots (*Jobs*). Ces lots nécessitent pour leur exécution des ressources qui sont ici des machines ou des postes de travail.

Une gamme de fabrication ou processus opératoire (*process*) est un ensemble de tâches ou variantes de tâches qui sont, soit sous la forme d'une chaîne en cas de production linéaire, soit sous la forme d'un graphe dans le cas de l'assemblage. Elle définit l'ordre partiel dans lequel s'exécutent les tâches pour la réalisation d'un produit déterminé.

Il existe une typologie des ateliers qui dépend, d'une part de la nature du système de production et, d'autre part de la nature de la gamme de fabrication.

Système de production

- atelier de production unitaire, où le travail consiste en la construction d'un produit par période;
- atelier de production discontinue, représenté par un ensemble de postes où les produits subissent des transformations dans chacun des postes;
- ateliers de production continue, où les ressources sont placées en série; le problème d'ordonnancement ne s'y pose pas;
- ateliers de production de masse, où la production est fixée à une cadence donnée où les produits sont fabriqués en très grandes quantités;
- atelier *On-line* (resp. *Off-line*), où la production est directe et le moindre problème est capable de ralentir ou de déranger la production.

Gamme de fabrication

- ateliers *Job shop*, à cheminement multiple, où chaque produit ou famille de produits possède une gamme spécifique. Ces gammes doivent être entrelacées de manière à assurer la production des produits demandée;
- ateliers *Flow shop*, à cheminement unique ou en flux, où les gammes sont identiques. Deux gammes identiques peuvent avoir des variantes de tâches différentes. Si, de plus, l'ordre des lots est identique sur toutes les ressources, il s'agit, alors, de *Flow shop* de permutation;
- ateliers *Flow shop* hybride, où les ressources sont regroupées par étages. Un étage est un ensemble de ressources susceptibles d'exécuter une tâche. Ces ressources sont équivalentes dans leur fonctionnement mais pas dans leurs performances;
- ateliers *Open shop*, à cheminement multiple, où les gammes ne sont pas complètement précisées. On dispose seulement de l'ensemble des tâches à effectuer mais sans connaître la manière de les enchaîner.

Une hiérarchie des types d'ateliers a été explicité dans [Vignier et al., 1995]. Comme remarque générale, nous dirons que le *Flow shop* hybride est un cas particulier du *Job shop* généralisé, lorsque les gammes sont identiques. S'il n'y a qu'une seule ressource dans chaque étage, l'atelier se ramène au cas du *Flow shop* traditionnel.

2.2 Représentation d'un problème d'ordonnancement

Elle peut être soit une représentation symbolique, soit une représentation graphique.

2.2.1 Représentation symbolique

Représentation mathématique

Il s'agit d'une formulation du problème d'ordonnancement sous la forme d'équations mathématiques. Elle a été utilisée surtout pour la résolution de problèmes d'ordonnancement avec la méthode de programmation linéaire en nombres entiers, et aussi pour formuler le problème d'ordonnancement sous la forme d'un problème de satisfaction de contraintes (*CSP*) [Fox et Sadeh, 1990].

Représentation classique

Il s'agit de la représentation la plus utilisée dans la littérature. Elle est composée de trois champs [Blazewicz et al., 1993] et est notée "$\alpha \mid \beta \mid \gamma$". Ces trois champs ont les significations suivantes:

- α décrit l'environnement du système;
- β décrit les caractéristiques des tâches et ressources;
- γ décrit le critère d'optimisation utilisé.

2.2.2 Représentation graphique

Il y a différents types de représentations graphiques (voir figure 2.1), nous citons les suivants:

Représentation par le diagramme de Gantt

C'est une représentation très ancienne et très facile à lire. Le diagramme de Gantt utilise deux axes, l'abscisse pour le temps et l'ordonnée pour les tâches [Lewis et El-Rewini, 1992].

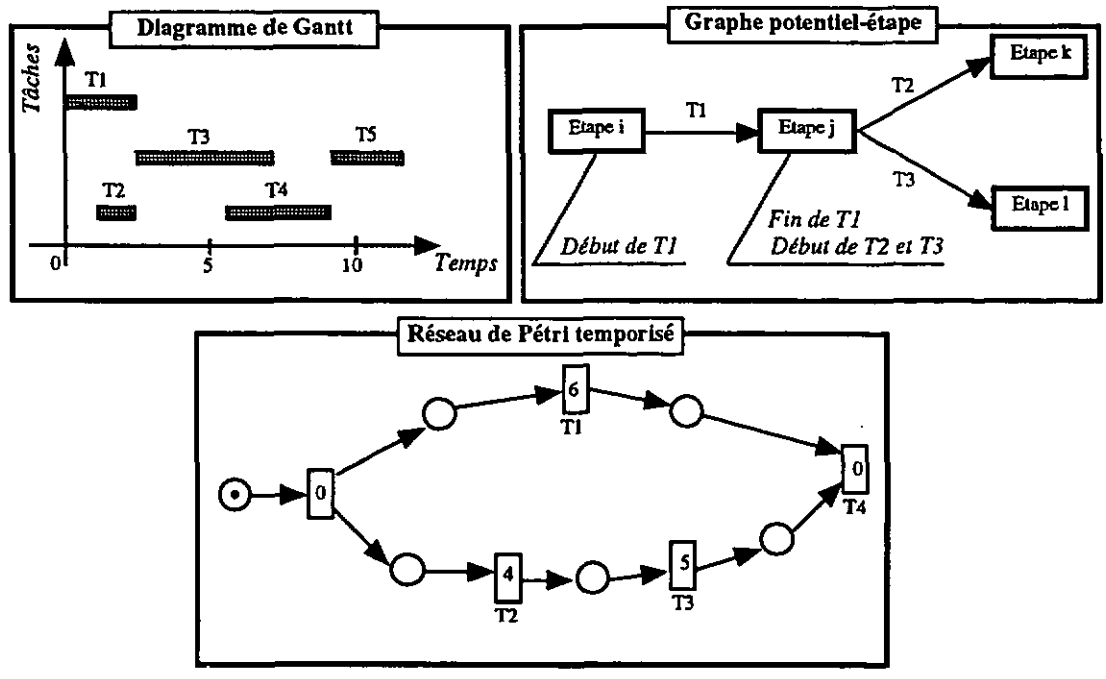


Figure 2.1: *Représentations graphiques d'un problème d'ordonnancement.*

Représentation par le graphe potentiel-tâche

C'est un graphe unidirectionnel qui concerne seulement les tâches et les relations entre elles, indépendamment des ressources. Deux tâches fictives sont ajoutées au graphe, une au début et une à la fin, ayant des durées nulles.

Représentation par le graphe potentiel-étape

C'est une des extension du diagramme de Gantt et qui est utilisée dans la méthode PERT. La tâche est représentée par une flèche qui relie une étape i à une étape $i+1$.

Représentation avec les réseaux de Pétri

Les réseaux de Pétri [Benasser et al., 1995] sont représentés par deux types d'objets: les places et les transitions. Pour représenter un problème d'ordonnancement, les places sont associées aux différents types de ressources et aux contraintes de succession, alors que les transitions sont associées aux tâches. Les franchissements des transitions correspondent à l'exécution des tâches.

2.3 Stratégies différentes

Pour résoudre un problème d'optimisation d'un ordonnancement, on cherche un algorithme qui nous donne l'ordonnancement optimal en un temps polynomial.

Si la réponse est positive, le problème en question est dit facile et il s'agit de trouver cet algorithme efficace. Dans le cas contraire, les stratégies qui peuvent être utilisées sont:

- la relaxation de certaines contraintes imposées du problème (une préemption possible, des dates limite repoussées, etc.) et la résolution du problème pour atteindre un ordonnancement très proche de la solution souhaitée [Balas, 1993];

- l'utilisation des algorithmes d'approximation qui tendent à trouver une solution optimal mais ne garantissent pas toujours d'atteindre cette solution;

- l'utilisation des algorithmes d'énumérations. Ils peuvent être sous la forme d'énumération implicite, ce qui revient à trouver la solution selon un critère bien déterminé sans pour autant tester toutes les solutions. Cette énumération conduit à construire un arbre et à éliminer les branches inutiles. Les techniques des procédures par séparation et évaluation entrent dans cette catégorie de traitement.

  Ils peuvent être aussi sous la forme d'énumération incomplète, ce qui revient à explorer la branche de l'arbre de recherche la plus intéressante à l'aide d'une heuristique (la première solution admissible, la solution meilleure que la courante, etc.). Cet énumération donne plus d'importance à la rapidité du temps d'obtention d'un résultat qu'au fait d'atteindre l'ordonnancement optimal.

2.4 Flow shop de permutation

Depuis quelques années les efforts de recherche, fournis dans le domaine de la gestion de production, sont de plus en plus remarquables. Le but est de développer des algorithmes d'ordonnancement d'ateliers efficaces et qui reflètent la situation réelle. Ceci afin de faire face aux investissements considérables utilisés pour installer des systèmes de fabrication automatisés.

De ce fait, des Systèmes Flexibles de Fabrication (SFF) ont été développés. Ces SFF contiennent des postes de travail et des facilités de manipulation automatisée. Ils permettent, en ayant une production diversifiée, de passer d'une configuration d'atelier *job shop* à une configuration d'atelier *flow shop*. Par ailleurs, il est bien

connu que l'utilisation des systèmes de fabrication caractérisés par une automatisation flexible peut être coûteuse pour l'atelier. Il s'est avéré que l'obtention d'un bénéfice en retour est liée à un ordonnancement efficace du système.

Les problèmes d'ordonnancement dans des ateliers de type *flow shop* sont souvent rencontrés dans des SFF. Les trois aspects suivants sont engendrés simultanément:

- l'exécution des tâches sur les ressources requises;
- le séquencement des tâches sur chaque ressource;
- le changement des outils des ressources;

De plus, les problèmes d'ordonnancement dans les SFF doivent tenir compte de contraintes additionnelles telles que le stockage des pièces, leur transfert d'une ressource à une autre et les facilités de changement d'outils. Ces contraintes rendent le problème d'ordonnancement encore plus difficile.

Rappelons qu'un *flow shop* est différent d'un *job shop* du fait que les gammes de fabrication sont identiques pour tous les produits. Rappelons aussi que si l'ordre des lots est le même sur toutes les ressources, alors le *flow shop* est appelé de permutation.

Bien que beaucoup de propositions pour résoudre les problèmes d'ordonnancement en *flow shop* soient basées sur des heuristiques, plusieurs auteurs ont proposé auparavant des méthodes exactes. Un des premiers algorithmes classiques dans ce domaine a été utilisé pour résoudre le problème "$F2 \mid\mid C_{max}$". Il s'agit d'un problème d'optimisation de la durée totale d'un ordonnancement dans un atelier de type *flow shop* à deux ressources.

Johnson [Johnson, 1954] a trouvé que cet algorithme nécessite un temps de résolution de $O(n \log n)$. Par la suite, il l'a généralisé au problème particulier de trois ressources, "$F3 \mid\mid C_{max}$", dans le cas où les temps opératoires sur la deuxième ressource sont inférieurs à ceux de la première et de la troisième.

Nous avons voulu mettre l'accent sur l'ordonnancement dans un atelier de type *flow shop* de permutation puisque c'est dans cet environnement que nous nous sommes placés pour l'élaboration de notre modèle, d'une part, et le développement d'un prototype sur la base de ce modèle, d'autre part.

Chapitre 3

Méthodes de résolution

Introduction

En général, un problème d'ordonnancement se ramène à l'attribution de valeurs à un ensemble de variables, satisfaisant un ensemble de contraintes. Le problème d'optimisation engendré est de maximiser ou minimiser une fonction objectif qui tient compte des critères de maximisation ou de minimisation imposés.

Les techniques en Recherche Opérationnelle (RO) et en Intelligence Artificielle (IA) ont été démontrées comme étant utilisables pour les problèmes d'ordonnancement. Des techniques d'ordonnancement classiques développées en RO, qui sont essentiellement des méthodes exactes, sont la Programmation linéaire [Freuder, 1982] et les procédures par séparation et évaluation [Hall et Wood, 1971]. Récemment, des méthodes approchées utilisant des techniques encore plus connues en Intelligence Artificielle ont été développées. Ce sont la Satisfaction de Contraintes, les Méthodes Connectionistes, les Systèmes Multi-Agents, la méthode du gradient, la méthode Tabou [Glover, 1990], le Recuit Simulé [Laarhoven et Aarts, 1992b] et les Algorithmes Génétiques.

Ces techniques ont des domaines différents dans lesquels elles sont spécialement efficaces. Par conséquent, le choix de la technique à utiliser pour résoudre un problème d'ordonnancement est loin d'être une tâche facile. Ainsi, la plupart des chercheurs utilisent généralement la technique la plus familière pour eux pour la résolution de tous les types de problèmes d'ordonnancement.

Par ailleurs, en présence de cette multitude de techniques, les réponses à quelques questions liées, d'une part au problème à résoudre et d'autre part à la technique à utiliser peuvent nous aider à faire notre choix.

Les questions liées au problème à résoudre et à l'optimisation:

- est-ce que le problème d'ordonnancement nécessite une solution qui satisfasse toutes les contraintes, ou une solution qui nécessite une certaine optimisation

avec une fonction à optimiser qui porte sur le nombre de contraintes violées?

- si une optimisation est nécessaire, est-ce que les solutions sous-optimales sont acceptées?

- relativement au temps que le programme nécessite pour trouver une solution, est-ce qu'il faut considérer les solutions optimales, bien qu'elles mettent un temps de traitement très élevé, ou est-ce que les solutions sous-optimales sont plus intéressantes dans ce cas?

Les questions liées à la technique à utiliser:

- comment sont exprimés les contraintes, les critères, les objectifs, les possibilités de relaxation des contraintes, etc.?

- quels sont les langages et architectures nécessaires à la mise en oeuvre de la technique?

- quelle est la complexité en place mémoire et en temps de calcul?

- est-ce que la technique peut traiter des problèmes distribués?

- est-ce qu'il y a possibilité de réviser la solution trouvée suite à des modifications des données du problème?

Dans ce chapitre, nous parcourons les différentes techniques de RO et IA en nous référant aux approches les utilisant pour la résolution des problèmes d'ordonnancement en général et spécialement dans un atelier.

3.1 Recherche Opérationnelle (RO)

Parmi diverses disciplines, la RO est incontestablement la première à s'être effectivement intéressée au problème d'ordonnancement. En effet, la programmation linéaire a été utilisée pour résoudre le problème d'affectation de ressources et le problème de transport.

Les techniques de la RO garantissent des solutions optimales. Elles sont limitées par l'explosion combinatoire et nécessitent un temps exponentiel en fonction de la taille du problème.

3.1.1 Programmation Linéaire (PL)

La PL est normalement appliquée aux problèmes qui peuvent être décrits par un ensemble d'équations linéaires et d'inégalités. La tâche est d'optimiser une fonction linéaire donnée [Bowman, 1959].

Une propriété particulière de la PL est qu'elle trouve toujours la solution si elle existe. Par contre, son applicabilité est limitée quand un grand nombre de contraintes sont engendrées.

3.1.2 Procédure par Séparation et Evaluation (PSE)

La PSE est un algorithme très connu et très utilisé pour l'optimisation en RO. Il représente une technique d'énumération implicite (voir section 2.3).

La technique de PSE a été appliquée aux problèmes d'ordonnancement qui nécessitent une optimisation [Baptiste et al., 1995]. En l'occurrence, pour les problèmes d'ordonnancement dans un atelier de type *job shop* [Brucker et al., 1992] et de type *flow shop* [Carlier et Rebai, 1996]. Son efficacité est déterminée par les facteurs suivants:

- la façon avec laquelle le problème est formulé: Un état de l'arbre de recherche peut être une affectation de valeurs à un ensemble de variables où les variables peuvent être les dates de début;
- la qualité de la fonction objectif et de son évaluation (quels états sont accessibles à partir de quels états);
- l'ordre dans lequel les branches de l'arbre sont traitées.

Il est important de noter que la connaissance du domaine peut aider dans ces trois aspects.

Plusieurs modèles ont été ainsi mis au point. Malheureusement, ceux-ci ont été vite incriminés à cause des difficultés qu'ils rencontrent à formaliser et à résoudre des situations réelles. En effet, la RO donne des méthodes algorithmiques qui sont rigoureuses mais qui sont souvent mal adaptées à la prise en compte des caractères spécifiques des problèmes d'ordonnancement: préférences de l'utilisateur, panne de machine, etc.

3.2 Intelligence Artificielle (IA)

L'avènement de l'IA a donné un très grand élan aux études des méthodes de résolution des problèmes d'ordonnancement. En effet, elle vise à faire intervenir des heuristiques (tirées des expériences humaines ou des connaissances propres au domaine) de manière à guider intelligemment la recherche de l'ordonnancement optimal et ainsi retarder et modérer l'explosion combinatoire.

Au cours des dernières années, les applications des techniques d'IA à des problèmes d'ordonnancement d'ateliers se sont multipliées. La raison principale est de résoudre les problèmes que la RO ne permet pas de résoudre. Mais aussi parce que:

- l'utilisation des techniques de l'IA permet de créer un système d'ordonnancement facile à transporter d'un atelier à un autre;
- elle permet d'exprimer et d'utiliser les connaissances de l'atelier de façon modulaire;
- elle facilite les interactions entre le système d'ordonnancement et l'utilisateur.

Les techniques utilisées sont coûteuses et il est nécessaire d'établir un compromis entre leur intérêt et leur coût d'utilisation.

3.2.1 Satisfaction de contraintes

La satisfaction de contraintes est un champ de la recherche en IA qui a démontré l'importance de son application dans l'ordonnancement. La plupart des techniques de satisfaction de contraintes développées auparavant sont utilisées pour trouver un ordonnancement qui satisfait toutes les contraintes. Des recherches récentes tendent à développer des méthodes pour attaquer les problèmes de satisfaction partielle [Ghédira, 1994b], les problèmes qui nécessitent une optimisation.

Problème de satisfaction de contrainte (CSP)

Un CSP comprend un ensemble de variables, un domaine fini pour chacune d'elles et un ensemble de contraintes qui restreint la combinaison de valeurs que les variables peuvent prendre.

Le problème est d'affecter à chaque variable une valeur de son domaine qui satisfait toutes les contraintes. Deux cas de problèmes peuvent être présentés: les problèmes pour lesquels il existe au moins une solution et, éventuellement, une solution optimale en cas d'optimisation, et les problèmes pour lesquels il n'existe pas de solution.

Un grand nombre de méthodes ont été développées pour résoudre les problèmes d'ordonnancement qui peuvent être formulés sous la forme de CSPs.

Dans [Fox et Sadeh, 1990], le problème d'ordonnancement est formulé de façon à déterminer son degré de difficulté et à montrer la nécessité de relaxer certaines contraintes pour trouver des solutions acceptables.

De même, une autre approche est proposée dans [Sycara et Liu, 1993] pour aborder l'ordonnancement sous la forme de CSP. Dans cette approche les variables sont représentées par les différentes activités (tâches) et une valeur représente une réservation pour l'activité (un temps de début et un ensemble de ressources requises). Une solution est une instanciation de l'ensemble des variables qui satisfait toutes les contraintes (les relations de précédences entre les activités et les capacités des ressources).

Dans un CSP, les algorithmes propagent les contraintes après l'affectation de chaque valeur à une variable pour réduire l'espace de recherche.

Propagation de Contraintes (PC)

La PC est une activité déductive effectuée par un système de propagation de contraintes pour résoudre un problème donné. Elle permet de décomposer le problème, sans négliger les interactions entre les sous-problèmes.

Les techniques de PC permettent de résoudre les problèmes les plus contraignants et les plus difficiles. Elles sont souvent utilisées pour résoudre les problèmes d'ordonnancement.

Le rapport [Pape, 1991] présente une discussion sur les travaux récents concernant des techniques de PC appliquées à différents problèmes d'ordonnancement. D'autre part, des résultats expérimentaux permettant de comparer ces techniques sur les différents problèmes d'ordonnancement sont exposés.

Plusieurs systèmes d'ordonnancement utilisant les techniques de PC seront explicités dans la section 3.2.3.

3.2.2 Langages issus des techniques IA

Les nombreuses recherches effectuées pour les problèmes de satisfaction de contraintes ont donné naissance à un ensemble de langages basés sur la programmation par contraintes.

Programmation Logique avec Contraintes (PLC)

La PLC est issue de l'intégration des techniques de satisfaction de contraintes avec la programmation logique.

La résolution d'un problème avec la PLC nécessite, généralement, plus de temps de traitement. Néanmoins, la facilité d'expression et de résolution favorise la maîtrise de la complexité. Ceci a été montré dans plusieurs applications [Baptiste et al., 1995].

Dans [Boucher et al., 1995], des techniques de PLC ensemblistes sont présentées pour la modélisation et la résolution d'un problème industriel d'affectation.

CHIP, CHARME et ILOG-SOLVER

L'optimisation est souvent une nécessité en ordonnancement. Les langages que nous citons ci-après fournissent des fonctions pour l'optimisation basées sur la méthode de PSE.

CHIP, *CHARME* et *ILOG-SOLVER* sont habiles à représenter et à résoudre les CSPs et par conséquent les problèmes d'ordonnancement qui peuvent être formulés sous la forme de CSP. Ce sont des langages qui, d'une part, déclarent les variables du problème et leurs domaines respectifs et, d'autre part, modélisent des contraintes afin de résoudre un problème donné. Le système de résolution est effectué en générant des valeurs pour les variables. Ces valeurs sont propagées à travers les contraintes ce qui aide à réduire l'espace de recherche.

CHIP est basé sur le langage *PROLOG*. Les termes de PROLOG facilitent la structure des données.

CHARME est dérivé de CHIP, qui est à l'origine développé au centre ECRC à Munich. C'est un langage simple à apprendre qui est muni d'une stratégie de génération de valeurs par défaut.

ILOG-SOLVER représente une bibliothèque orientée objet en version C^{++} et Le-lisp. *ILOG* a consacré une bibliothèque C^{++} pour représenter les problèmes d'ordonnancement: *ILOG SCHEDULE* [Pape, 1994]. Elle est écrite en *ILOG-SOLVER* et possède des classes d'objets spécialement utilisées pour représenter des types variés de ressources, d'activités (tâches) et de contraintes d'un problème d'ordonnancement.

3.2.3 Systèmes d'ordonnancement

La conception et la programmation de plusieurs systèmes d'ordonnancement représentent un pas important vers la précision, la flexibilité et l'efficacité des traitements. Nous citons les systèmes suivants:

ISIS: Il s'agit d'un système d'ordonnancement [Fox et Smith, 1984] conçu pour l'utilisation pratique dans la gestion de production et de contrôle en *job*

shop. Le but est d'obtenir un ordonnancement en tenant compte de toutes les contraintes. Il reconnaît tous les types de contraintes et permet de relaxer certaines d'entre elles en cas de conflits.

SOJA: Il s'agit d'un système d'ordonnancement journalier d'atelier [Pape et Sauve, 1985] qui facilite l'ordonnancement d'un ordre de fabrication au niveau d'une ligne de production. Il tient compte:

- des informations en provenance du service de vente;
- de la disponibilité des matériaux;
- de la disponibilité des moyens;
- de l'établissement de l'ordre de fabrication avec les dates de lancement et de livraison en question.

OPIS: Les travaux effectués avec ce système d'ordonnancement [Smith et al., 1986] ont engendré une approche plus opportuniste de l'ordonnancement. Les décisions de l'ordonnancement ne sont pas prises selon une stratégie fixe de décomposition du problème. La décomposition est dynamique et est effectuée selon les caractéristiques du problème, d'une part, et l'état courant de l'ordonnancement, d'autre part. *OPIS* développe et maintient les ordonnancements à différents niveaux d'abstraction sur différents horizons de planification [Pape et Smith, 1988].

SONIA: Il s'agit d'un ordonnanceur réactif qui s'intéresse aux perturbations en temps réel [Collinot et al., 1988]. Il utilise une approche basée sur les techniques d'IA pour guider la production et la gestion des processus. Il est organisé par un module de maintenance des décisions de l'ordonnancement et par un ensemble de modules de résolution de problème. De plus, il est muni d'un simulateur qui est utilisé pour tester les possibilités de perturbations.

OSCAR: Il s'agit d'un système [Badie et Verfaillie, 1989] développé pour traiter des problèmes de gestion de missions spatiales et, plus particulièrement, de gestion des conflits entre tâches pour l'accès à des ressources partageables et de capacité limitée.

Il est fondé sur des notions

- d'incrémentalité: les tâches sont placées l'une après l'autre dans un contexte croissant, leur placement n'étant pas définitif;
- de *backtrack* intelligent: en cas de conflits entre tâches, une analyse permet de revenir sur les choix précédents qui ont à voir avec le conflit.

Ce système a prouvé ses propriétés de terminaison et de complétude. En plus, en cas d'existence de solution satisfaisant toutes les contraintes, il donne des résultats excellents vis-à-vis de la rapidité du traitement.

3.3 Approches d'IA Distribuée (IAD)

Les approches distribuées et décentralisées sont en plein développement. Ces approches se distinguent principalement par le degré de complexification des entités qui contribuent à la résolution du problème mais aussi par le type de communication entre ces entités.

Le principe est l'interaction des entités. Ces entités communiquent par échange de messages ou via une mémoire partagée.

L'intérêt commun à l'IA distribuée et l'IA décentralisée (IADc) [Demazeau et Muller, 1989] est celui du comportement distribué des entités. Par ailleurs, la différence entre eux se présente dans la résolution des problèmes.

- en IAD, une tâche globale est définie initialement, il s'agit d'étudier la distribution et la résolution de la tâche en concevant des entités distribués;
- en IADc, des entités sont définies initialement, il s'agit d'étudier la structure de ces entités pour connaître le genre de problèmes qu'elles peuvent résoudre ou les tâches qu'elles peuvent accomplir.

La résolution d'un problème en IAD se fait par collaboration du fait que l'information est partagée entre les entités pour permettre de produire une solution ou d'accomplir une tâche globale. L'ensemble d'entités est distribué, de même, tous les contrôles et les données impliqués sont logiquement et, souvent géographiquement, distribués.

3.3.1 Modèles connexionnistes (MC)

Les MCs, et spécialement ceux de *Hopfield* et *Boltzmann*, offrent l'avantage d'un modèle numérique très simple sur lequel des résultats mathématiques sont utilisables.

Les MCs sont efficaces pour les problèmes de satisfaction et d'optimisation. Ainsi, en utilisant un grand nombre de ressources, ils ont la possibilité de générer des ordonnancements plus rapidement. Un inconvénient des approches connexionnistes est que la construction de réseaux spécialisés pour une application particulière pour, éventuellement gagner l'efficacité, pourrait être coûteuse.

Dans les MCs, un problème est représenté par un réseau. Les traitements des différents noeuds du réseau et la manière avec laquelle ils sont connectés entre eux,

sont la clé pour le succès de cette approche. De plus, ces modèles sont munis d'un mécanisme qui est conçu pour permettre au réseau de converger vers les solutions.

Genet est une approche connexionniste pour les problèmes de satisfaction de contraintes [Davenport et al., 1994]. Pour appliquer *Genet* à l'ordonnancement, les problèmes sont définis comme des CSPs. Chaque valeur d'une variable est représentée par un noeud donné. Chaque contrainte est représentée par un noeud contrainte. Par ailleurs, un ensemble de règles est conçu pour assurer que le réseau aboutit à certains états et un mécanisme d'apprentissage est utilisé pour sortir d'un optimum local.

3.3.2 Systèmes Multi-Agents (SMA)

D'une manière générale un SMA modélise un problème sous la forme d'un ensemble d'agents. Chaque agent cherche à atteindre sa propre satisfaction maximale et entre dans des relations de coopération, de concurrence ou de conflit avec d'autres agents. Cette approche semble prometteuse pour une utilisation sur architecture parallèle, d'une part, et pour le traitement des problèmes intrinsèquement distribués, d'autre part.

Qu'est-ce qu'un agent?

Un agent est une entité réelle ou abstraite capable d'agir sur elle-même et sur son environnement. Un agent a un comportement défini suivant ses observations, sa connaissance et ses interactions avec les autres agents avec lesquels il peut communiquer appelés accointances.

Nous distinguons deux aspects de l'agent:

- un aspect individuel caractérisé essentiellement par ses connaissances et son environnement;
- un aspect collectif caractérisé par le type de communication avec ses accointances. Elle peut être par partage d'information via un espace de travail commun à tous les agents appelé tableau noir, ou encore par envoi de messages synchrones ou asynchrones d'un agent à l'autre.

En parlant de l'aspect individuel de l'agent on distingue deux types d'agents:

- l'agent cognitif qui est un agent intelligent capable de résoudre des problèmes de manière autonome. Il raisonne sur la base de ses connaîssances et son environnement. Son comportement est lié à des intentions, des buts et il tient compte de son historique;

- l'agent réactif est un agent simple qui ne dispose d'aucune intelligence. Il a un comportement basé sur le réflexe et non sur des intentions, c'est un comportement biologique.

  La méthode de l'Eco-résolution dont nous parleront dans le paragraphe suivant utilise des agents réactifs.

Plusieurs chercheurs ont montrés que de l'intéraction d'un nombre d'agents réactifs, il peut émerger des organisations complexes. Dans le domaine de la robotique, nous citons [Brooks, 1991] et [Demazeau et Muller, 1990]. D'autres ont montré que les agents réactifs peuvent effectuer des tâches déjà réservées pour les agents cognitifs. Nous citons [Steels, 1991] et [Corbara et al., 1991].

Modèle Eco-résolution

D'après [Ferber et Jacopin, 1990], l'Eco-résolution est une approche interactive de résolution des problèmes. Chaque agent de la société d'agents, appelé éco-agent, a un comportement qui permet d'atteindre un état appelé solution ainsi qu'un critère de terminaison pour savoir si l'état en question est atteint ou non.

Le comportement d'un éco-agent est caractérisé par:

- la mise à jour de sa mémoire locale;
- l'envoi de messages et la réaction aux messages reçus;
- eventuellement, la création d'agents.

Il dépend de sa condition de satisfaction et est de la forme suivante:

"*Si* Condition-Satisfaction
alors Comportement-Satisfaction
sinon Comportement-Insatisfaction"

Applications à l'ordonnancement

Récemment, les chercheurs ont commencé à mener des études et des expériences sur la résolution des problèmes d'ordonnancement avec les SMAs. Nous citons quelques uns des plus importants parmi une liste non-exhaustive de travaux effectués.

- L'article [Sycara et al., 1990] présente une approche d'IA distribuée pour le contrôle d'atelier qui génère efficacement des ordonnancements pour la totalité de l'atelier. Un ensemble d'heuristiques ont été développées qui permettent à l'ordonnanceur de sélectionner la prochaine opération à ordonnancer

et la réservation de la prochaine ressource qui exécutera l'opération sélectionnée.

Des expérimentations ont été effectuées sur ces heuristiques pour le cas d'un seul agent Ordonnancement et ont montré l'efficacité des ordonnancement proposés. Par la suite, l'utilisation de ces heuristiques a été étendue dans le cadre distribué, avec plusieurs agents Ordonnancement.

- Un ordonnancement distribué intégré (*Integrated Distributed Scheduler IDS*) est introduit dans le rapport de recherche [Luo et al., 1993]. IDS intègre des approches distribuées variées et des approches à calcul parallèle, à savoir *distributed-agent-based*, *parallel-agent-based* et *function-agent-based*. Ce sont des stratégies de résolution de problèmes d'ordonnancement distribués.

- Dans [Neiman et al., 1994], la réalisation d'une interaction efficace dans un système d'ordonnancement distribué est étudiée. Dans ce système, les agents Ordonnancement peuvent se prêter mutuellement des ressources. L'étude montre comment l'analyse des besoins de ressources abstraites d'agents distants peut guider un agent dans le choix des activités (tâches) à ordonnancer.

  Ce travail est présenté dans le contexte d'un système distribué de gestion de ressources dans un aéroport: un système multi-agents pour résoudre les problèmes d'ordonnancement dans les services de l'aéroport.

- Un mécanisme de coordination *Anchor&Ascend* est présenté dans [Liu et Sycara, 1995] pour l'optimisation de contraintes distribuées. Son principe est de guider efficacement la recherche locale distribuée pour l'optimalité globale. Ceci est obtenu par l'attribution de différents sous-problèmes chevauchés aux agents qui doivent interagir et converger itérativement vers une solution.

  L'efficacité du mécanisme est évaluée expérimentalement sur une suite de problèmes d'ordonnancement de type *job shop*.

D'autres travaux sur les problèmes d'allocation de ressources ont été présentés dans [Berry et al., 1992] et [Choueiry et Faltings, 1992].

Les techniques Multi-Agents ouvrent, en effet, une nouvelle voie pour appréhender de façon naturelle et agréable les problèmes d'ordonnancement en terme de coopération, conflit et concurrence dans une société d'agents.

3.4 Approches stochastiques

Une technique stochastique est une technique de résolution de problèmes. Elle représente une séquence d'opérateurs utilisés pour transformer l'état initial de

l'espace de recherche en un des états finaux. En ordonnancement, un état peut être un ensemble de tâches placées sur des ressources à des dates données.

L'efficacité de ces techniques est liée aux heuristiques et à certaines mesures d'aléa. Elles sont utilisées pour les problèmes d'optimisation où les solutions sous-optimales sont acceptables.

Les techniques telles que la méthode du gradient, la méthode tabou, le recuit simulé et les algorithmes génétiques sont désignées pour les problèmes où la complétude est sacrifiée devant la vitesse de traitement, néanmoins la solution atteinte peut être sous-optimale.

3.4.1 Méthode du Gradient (MG)

La MG ou recherche locale a le traitement de base suivant:

1. Partir d'un état dans l'espace de recherche qui est généré soit aléatoirement soit avec des heuristiques.

2. Choisir le meilleur état voisin, selon la fonction à optimiser.

3. Répéter ce processus jusqu'à ce qu'il n'y ait plus de meilleur état disponible dans le voisinage de l'état courant.

La méthode *Min-Conflict Heuristic Repair* (MCHR) qui est développée dans [Minton et al., 1992], a été appliquée pour l'ordonnancement, de même, GSAT [Selman et al., 1994] est une autre technique de la MG.

La MG est une technique très importante pour attaquer les problèmes qui ne peuvent pas être traités par des algorithmes exactes. Par contre, l'inconvénient majeur de cette technique est qu'elle peut être piégée dans un optimum local ou se perdre dans un plateau.

3.4.2 Méthode Tabou (MT)

C'est une méthode dont le comportement ressemble à celui du MG et qui est utilisée pour essayer d'éviter les optima locaux. Son efficacité dépend de la définition et de la maintenance de ce qu'on appelle sa liste tabou. La liste tabou est une liste qui contient certaines restrictions sur les déplacements d'un état à un autre. Elle peut être, par exemple, la liste des n états déjà visités et dont la visite une deuxième fois est défendue. Elle doit être mise à jour après chaque déplacement.

La MT a le traitement de base suivant:

1. Partir d'un état initial généré soit aléatoirement soit avec des heuristiques.

2. Choisir un état voisin qui est en même temps, le meilleur selon la fonction à optimiser et permis par la liste tabou.

3. Répéter ce processus jusqu'à ce qu'il n'y ait plus d'états possibles dans le voisinage de l'état courant.

De cette manière, la liste tabou a la responsabilité d'éviter une oscillation infinie entre un état (peut-être un optimum local) et son voisin.

Pour résoudre le séquencement dans un atelier de type flow shop de permutation, la MT a été appliqué dans [Taillard, 1990]. L'optimalité n'est pas garantie, en revanche, les résultats expérimentaux ont montré que la solution optimale est toujours atteinte, avec plus de temps de traitement, pour les problèmes ayant une solution exacte connue. De même, une heuristique appelée FSHOPH pour *Flow Shop Heuristic*, qui a montré sa performance et dont le traitement est basé sur la MT, a été présentée dans [Moccellin, 1995].

Différentes recherches tabou peuvent utiliser différentes stratégies pour manipuler la liste tabou. La MT reste, par conséquent, une stratégie de recherche assez générale.

3.4.3 Algorithmes Génétiques (AG)

Les AGs sont utilisés pour trouver des solutions sous-optimales. En général, ils permettent de mieux explorer l'espace de recherche que la MG par exemple. Néanmoins, ceci se fait au détriment du temps de traitement.

Une stratégie de contrôle des AGs généralement utilisée est la suivante:

1. *Génération*: une population initiale P_0 de N individus est formée aléatoirement. Dans un problème d'ordonnancement, un individu peut être un chiffre binaire (0 ou 1) indiquant la date de début d'une tâche ou encore une séquence de tâches affectées à une ressource.

2. *Reproduction*: une nouvelle population, dérivée de l'initiale, est formée en affectant aux individus ayant les poids les plus élevés les plus grandes probabilités pour être copiés dans la nouvelle population.

3. *Croisement*: A chaque itération, un couple d'individus est remplacé par un nouveau couple d'individus, obtenu en appliquant un opérateur génétique de croisement.

4. *Mutation*: A chaque itération, un individu est remplacé par un nouvel individu, obtenu en appliquant un opérateur génétique de mutation.

5. *Arrêt*: Si le critère d'arrêt donné n'est pas vérifié, alors le traitement reprend depuis l'étape 2.

Les AGs sont utilisés pour les problèmes d'optimisation et ont montré leur efficacité dans certains problèmes d'ordonnancement. Parmi les diverses études effectuées, un exemple a été étudié et discuté dans [Zbigniew, 1994] qui applique les AGs à un problème d'ordonnancement de type *job shop*.

L'article [Fichera et al., 1995] montre l'efficacité des AGs dans la résolution de problème d'ordonnancement dans un atelier de type *flow shop* de permutation. Il s'agit d'examiner les formulation de l'opérateur de croisement et les paramètres de contrôle. Ceci est effectué de telle manière à obtenir de bonnes performances dans la détermination d'un ordonnancement ayant une durée (*makespan*) la plus courte possible.

De même, dans l'article [Portmann et Ghedjati, 1995], les AGs ont été utilisés pour résoudre le problème d'ordonnancement dans des ateliers de type *job shop* généralisés.

3.4.4 Recuit Simulé (RS)

Le RS est un algorithme basé sur une combinaison originale de deux domaines indépendants de la science: l'optimisation combinatoire et la physique statistique.

Le RS peut être considéré comme une extension du MG afin d'éviter l'optimum local dans la résolution des problèmes d'optimisation. Il peut être aussi vu comme une analyse d'un algorithme utilisé en physique statistique pour la simulation du recuit d'un solide afin d'atteindre son état minimum d'énergie.

Recuit thermique et recuit simulé

En thermodynamique, pour aboutir à un système physique ayant un état stable, le traitement suivant est effectué:

1. Le système est porté à une température très élevée afin qu'il atteigne l'état liquide; un état où tous les atomes sont répartis de manière aléatoire et désordonnée.

2. La température est baissée progressivement. Ainsi, les atomes s'organisent au fur et à mesure pour aboutir à une structure figée régulière, ce qui correspond à une valeur minimale de l'énergie.

La manière de baisser la température s'avère très importante, car une décroissance rapide et brutale aboutit, généralement, à un système physique non régulier.

ALGORITHME 1 Recuit simulé standard

Initialiser *;génération d'un état initial*
m = 0 *;compteur d'itérations*
REPETER
  REPETER
    Perturber *;génération d'un état j à partir d'un état i*
    calculer Δ_{ij} *;différence de coûts des états i et j*
    SI $\Delta_{ij} \leq 0$ **ALORS**
      accepter E_j *;acceptation de l'état j*
    SINON
      $expo \leftarrow e^{\frac{\Delta}{t_m}}$
      SI $expo > alea[0,1)$ **ALORS**
        accepter E_j *;aspect aléatoire du recuit simulé*
      FIN SI
    FIN SI
    SI accepter **ALORS**
      mettre-à-jour-état *;état j devient l'état courant*
    FIN SI
  JUSQU'A condition *;état suffisemment proche de l'état d'équilibre*
  $t_{m+1} = f(t_m)$ *;f est une fonction de décroissance de la température*
  $m = m + 1$ *;augmentation du compteur*
JUSQU'A arrêt = vrai *;système figé*

Dans les problèmes d'optimisation, étant donné un ensemble d'états formant l'espace de recherche, il s'agit d'atteindre l'état qui satisfait les contraintes du problème et optimise les critères représentés par une fonction de satisfaction appelée fonction coût.

Par analogie entre l'énergie et la fonction coût et entre les états du système physique et ceux de l'espace de recherche, le RS a été proposé comme une technique de résolution des problèmes d'optimisation combinatoire.

Par conséquent, un paramètre de contrôle qui dépend des données du problème simule la température. De même, le processus de déplacement aléatoire d'un état à un autre est représenté par une probabilité de transition.

Une description de l'algorithme du RS standard (voir algorithme 1) est présentée dans [Laarhoven et Aarts, 1992b].

La définition de la fonction coût et le schéma de refroidissement jouent un rôle important dans l'efficacité du RS.

Plusieurs études ont fait l'objet de l'application du RS au problème d'ordonnancement:

Il a été prouvé dans [Laarhoven et al., 1992] que l'algorithme d'approximation proposé, basé sur le RS, trouve des ordonnancements de durées plus courtes que deux autres approches d'approximation telles que celle d'Adams, de Balas et Zawack. Ces approches sont conçues pour les problèmes d'ordonnancement de type *job shop*.

Un algorithme de RS localisé a été proposé dans [Li et Jiang, 1995]. Cet algorithme a été appliqué aux problèmes d'ordonnancement d'un parc aérien. Les résultats expérimentaux ont montré que cet algorithme est plus performant que le RS standard pour ce type de problèmes.

Une étude des techniques du RS parallèle a été exposée dans [Greening, 1990] pour mettre en valeur leur performance et leur applicabilité.

Chapitre 4

Fondements du modèle

Suite aux travaux cités dans le chapitre précédent, nous explicitons les arguments qui nous ont motivés à utiliser les méthodes appliquées dans notre travail.

4.1 Approche utilisée

Une approche itérative a été proposée dans le cadre du problème d'allocation de tâches sur des ressources partageables à capacité limitée [Ghédira, 1993]. Elle résulte de la combinaison de deux domaines distincts déjà cités:

- les systèmes multi-agents;
- le recuit simulé.

C'est cette approche que nous avons prise comme point de départ dans notre recherche. Notre contribution consiste:

- d'une part à adapter l'approche à notre problème d'ordonnancement dans un atelier de type *flow shop* de permutation;
- d'autre part à l'étendre à la prise en compte des contraintes temporelles. Ces contraintes peuvent être soit absolues telles que les dates de livraison des demandes client (DC) et les temps de réglages des ressources, soit relatives telles que les relations de précédences entre les actions de chaque DC.

Les sections suivantes ont pour objectif de montrer comment les deux domaines sont utilisés dans cette approche.

4.1.1 Utilisation des systèmes multi-agents

La mise en oeuvre de cette approche associée au problème d'allocation a donné naissance à un modèle qui reprend les idées de l'Eco-résolution (voir section 3.3.2).

Deux classes d'agents sont engendrées dans cette approche; la classe d'agents Tâche et la classe d'agents Ressource. Cependant un agent problème, qui représente l'intermédiaire entre la société d'agents et l'utilisateur, s'est avéré nécessaire pour s'occuper (a) de la création des agents du problème et (b) d'informer l'utilisateur du résultat trouvé par cette société d'agents.
Ainsi, pour se satisfaire:

- chaque agent Tâche cherche à se placer sur l'une de ses ressources possibles. Par conséquent, il est satisfait s'il est placé;
- chaque agent Ressource essaye de placer le maximum de tâches possibles. Il est satisfait si sa capacité restante couvre la quantité demandée par ses agents tâches en attente.

4.1.2 Utilisation du recuit simulé

L'objectif essentiel espéré par toute approche est de chercher un mécanisme qui garantit la terminaison avec une solution optimale. Pour cela, le recuit simulé a été intégré dans l'univers multi-agents de manière à associer à chaque agent Ressource un recuit simulé pour décider des déplacements dans l'espace de recherche. Cet outil d'optimisation consiste à permettre des pertes locales de la fonction coût dans l'espoir qu'elles amènent un gain au niveau global [Kirkpatrick et al, 1983]. Chaque agent Ressource est responsable de gérer son propre processus de recuit simulé. Par analogie avec le recuit thermique, un schéma de refroidissement, déterminé en fonction des données du problème, a été proposé. Ce schéma relatif à chaque agent Ressource est composé de trois paramètres: la tolérance initiale, la décroissance de la tolérance et la terminaison.

Tolérance initiale

Le processus du recuit simulé est contrôlé par un paramètre: la température, appelée dans cette approche tolérance. Cette tolérance doit être suffisamment élevée au début du processus, afin de donner la possibilité de visiter tous les états de l'espace de recherche de manière équiprobable.

Décroissance de la tolérance

La convergence ne peut pas être assurée en gardant la même valeur de la tolérance tout le long du processus. Par conséquent, la tolérance est baissée progressivement

chaque fois qu'il y a acceptation de perte de satisfaction locale de l'agent Ressource en question.

Terminaison ou critère d'arrêt

La tolérance est baissée jusqu'à atteindre le niveau nul qui correspond au niveau de stabilité dans le recuit thermique. Dès lors, seuls les états qui augmentent le niveau local de satisfaction sont acceptés. Par ailleurs, si un agent Tâche demande à se placer sur un agent Ressource et que ce dernier le refuse, l'agent Tâche sera bloqué.
La terminaison du processus est donc vérifiée lorsque tous les agents Tâches sont soit placés soit bloqués.

4.2 Motivations

4.2.1 Motivations liées aux domaines multi-agents et recuit simulé

Certains résultats cités dans l'état de l'art (sections 3.3.2) nous ont motivés pour utiliser les systèmes multi-agents ainsi que le recuit simulé.

Comme première constatation, les techniques multi-agents ouvrent une nouvelle voie pour appréhender de façon naturelle et agréable divers problèmes en termes de coopération, conflit et concurrence dans une société d'agents. Ces techniques ont déjà été utilisées pour la résolution de problèmes d'ordonnancement, en l'occurrence dans le système DAS pour *Distributed Asynchronous Scheduler* [Burke et Prosser, 1990]. Le problème d'ordonnancement est dans ce cas décomposé à travers une hiérarchie d'agents communiquants où chaque agent expose les propriétés d'opportunité, de réaction et de maintenance de croyance. Chaque agent correspond à un processus distinct et tous peuvent s'exécuter en concurrence. Pour être capable de travailler pour une solution globale, chaque agent doit être capable de négocier avec les autres. DAS est maintenant installé au *Alcan Plate Ltd* et intégré à un système de planification de processus et à un système d'atelier *on-line*.

Dans ce même cadre, les techniques multi-agents ont été utilisées dans [Sycara et Liu, 1993] pour formuler le problème d'ordonnancement d'un atelier de type *job shop* sous forme d'un problème de satisfaction de contraintes (CSP). La méthodologie proposée est appelée CPCR pour *Constraint Partition and Coordination Reaction*, où chaque agent est responsable de l'instanciation d'un sous-ensemble de variables de manière à satisfaire les contraintes portant sur ces variables. Les variables sont représentées par les actions, les valeurs possibles par les

réservations (temps de début et ressources requises) et les contraintes par les relations de précédence et les capacités des ressources. Des résultats expérimentaux ont montré les performances de cette méthodologie.

Comme deuxième constatation, les techniques du recuit simulé ont déjà montré leur efficacité en temps de traitement et en qualité de la solution par rapport à d'autres méthodes et sur diverses applications, en l'occurrence sur les problèmes d'ordonnancement. En effet, il a été prouvé dans [Laarhoven et al., 1992] qu'un algorithme approximatif pour le problème de recherche du minimum *makespan* (durée d'un ordonnancement) dans un atelier de type *job shop*, converge asymptotiquement vers une solution minimale globale.

De même, dans [Zegordi et al., 1995] une nouvelle approche a été proposée pour appliquer le recuit simulé au problème d'ordonnancement dans un atelier de type *flow shop*. Cette approche combine la méthodologie du recuit simulé avec une information spécifique du problème, donnée dans un tableau appelé *Backward-Forward Exchange Priority Table* qui incorpore des règles obtenues à partir de l'examen de plusieurs heuristiques existantes dans la littérature. La performance de la méthode proposée a été testée sur des problèmes générés aléatoirement pour prouver qu'en utilisant le recuit simulé avec un nombre de paramètres de contrôle minimal, de bons résultats, en temps d'exécution et en qualité, sont obtenus.

Les résultats des approches proposées utilisant les systèmes multi-agents ou le recuit simulé nous ont motivés à les combiner dans notre approche et les rendre encore plus performantes. En plus de cette combinaison, nous ajoutons l'aspect distribué qui consiste à attribuer à chaque agent, appartenant à une certaine classe, son propre mécanisme de recuit simulé.

4.2.2 Motivations liées à l'approche utilisée

Les travaux présentés dans [Ghédira et Verfaillie, 1991] [Bel et al, 1992] et développés dans [Ghédira et Verfaillie, 1992a] et [Ghédira et Verfaillie, 1992b] ont montré l'intérêt d'une telle approche sur ce cas particulier de problème d'allocation. Les avantages se sont révélés être:

- le caractère naturel et agréable d'une modélisation multi-agents du problème: description des tâches et des ressources sous forme d'agents, connaissances statiques et dynamiques, accointances, comportements, heuristiques, etc;
- le caractère extrêmement simple et réactif du comportement de chaque agent Tâche ou Ressource, reprenant les idées de l'éco-résolution;

- l'optimisation à l'aide d'un mécanisme de recuit simulé distribué au niveau de chaque agent Ressource (chaque ressource a son propre recuit simulé qu'elle gère de manière autonome en fonction de ses connaissances propres et de ses interactions avec les tâches qu'elle connaît; ,

- mais, surtout, un mécanisme simple de révision à partir de la configuration courante, permettant de prendre en compte une modification du problème intervenant pendant ou après la résolution.

De plus, une modélisation sous la forme de chaînes de Markov a permis, à l'aide des résultats théoriques relatifs à la convergence vers un optimum global, de valider l'approche.

Ces avantages, associés aux bons résultats expérimentaux obtenus sur des séries d'exemples générés de façon automatique, nous ont encouragés à étendre cette approche et à la généraliser au problème d'ordonnancement.

De même, le fait que le problème d'allocation ne soit rien d'autre qu'un sous problème d'ordonnancement, et le fait que l'ordonnancement peut être formulé sous forme d'un problème de satisfaction de contrainte (CSP), justifie a priori cette démarche.

Partie II

Modèle MARSA

Introduction

Nous avons vu précédemment (section 2.1) qu'ordonnancer un ensemble de tâches revient à programmer l'ordre de leur exécution en les plaçant sur les ressources requises et en fixant leurs dates de début.

En parcourant la littérature, nous remarquons que les problèmes d'ordonnancement apparaissent dans des domaines très divers. Cependant, quel-que soit le domaine choisi, confronter l'aspect théorique d'un problème d'ordonnancement (analyse et conception) à son aspect pratique (résolution), engendre la nécessité de développer des méthodes de résolution qui tiennent compte de toutes les caractéristiques du problème.

De même nous pouvons constater qu'en dépit de la typologie des ateliers et de la diversité des applications d'ordonnancement, nous reconnaissons des problèmes de même nature. Ils sont caractérisés par un ensemble de ressources existant dans l'atelier et un ensemble de tâches à exécuter sur ces ressources selon un ordonnancement à déterminer.

Dans notre travail, nous nous attachons à une classe générale de problèmes d'ordonnancement en prenant en compte les particularités propres à une classe d'ateliers de type *flow shop* de permutation.

Nous disposons d'un ensemble de ressources et d'un ensemble de Demandes Client (DCs) (formée chacune d'un graphe d'actions) à placer sur les ressources. La solution que nous souhaitons atteindre est une combinaison optimale de l'ensemble des DCs placées sur les ressources où chaque DC est liée à un ordre de fabrication à exécuter.

Notre intérêt, dans ce travail, est porté sur les deux aspects de l'ordonnancement: statique et dynamique (voir section 2.1.5).

En se plaçant dans le cadre de l'aspect statique du problème d'ordonnancement, nos réflexions se basent sur:

L'objet visé: Un ordonnancement d'un ensemble de DCs, tout en ayant un compromis entre la qualité de cette solution et le temps de traitement nécessaire pour la trouver.

La modélisation du problème: Modéliser un problème dans sa globalité entraîne toutes sortes de difficultés, ne serait-ce que de pouvoir tenir compte de toutes les caractéristiques du problème. Les systèmes Multi-agents abordent le problème par un groupe d'entités appelées société d'agents. Ces agents ont une certaine autonomie et interagissent pour dégager une décision. Suivant cette modélisation, nous avons envisagé deux classes principales d'agents: les DCs et les Ressources.

La méthode de résolution: Etant donné que notre espace de recherche est composé d'états, un état étant une combinaison de DCs, la première idée qui vient à l'esprit est de choisir, parmi toutes les combinaisons possibles de DCs, celle qui satisfait le plus nos objectifs. Cette méthode de résolution favorise la combinatoire et nécessite un temps de traitement exponentiel, puisqu'elle consiste à parcourir tous les états de l'espace de recherche.

Par conséquent, il faut trouver une méthode nouvelle d'exploration pour éviter aussi bien un cheminement aveugle qu'un parcours exhaustif de l'espace de recherche. Nous avons opté pour un outil d'optimisation combinatoire: le recuit simulé, pour aider le système à converger vers la solution espérée.

En se plaçant dans le cadre de l'aspect dynamique du problème, les réflexions déjà citées restent toujours valables. Néanmoins, l'ordonnancement obtenu est relatif à un instant donné. En effet, les données du problème peuvent être modifiées et nous sommes à chaque fois en présence d'un problème différent. Les modifications peuvent être sous différentes formes, à savoir:

- ajout d'une ou de plusieurs DCs à placer;
- correction des dates effectives de début et de fin des DCs déjà placées;
- enrayages d'une ou plusieurs ressources pendant une durée de temps;
- etc.

Nos réflexions supplémentaires se basent sur la manière de tenir compte des modifications: le réordonnancement, suite à l'ajout de DCs ou l'ajustement des dates, suite aux différentes corrections, tout en gardant la stabilité de la solution trouvée.

Cette partie concernant le modèle MARSA, pour *Multi-Agent Reactive system for Scheduling Assembly*, est composée de cinq chapitres. Le chapitre 5 présente en détail le modèle de base sous son aspect statique. Le chapitre 6 concerne deux heuristiques conçues pour améliorer le modèle. Le chapitre 7 contient les expérimentations et les tests effectués sur différentes versions du modèle ainsi que les résultats correspondants. Le chapitre 8 présente l'aspect dynamique du modèle et sa capacité de réagir aux événements. Enfin le chapitre 9 traite un exemple industriel avec le modèle, en tenant compte de son aspect dynamique.

Chapitre 5

Modèle de base, aspect statique

D'une manière générale, nous nous intéressons à la résolution du problème d'ordonnancement pour des ateliers de type *flow shop* de permutation. Nous disposons d'un système caractérisé par un ensemble de ressources disponibles et nous souhaitons réaliser le meilleur placement d'un ensemble de Demandes Clients (DCs) sur les Ressources. Le placement veut dire programmer l'ordre d'exécution des DCs en fixant leurs dates de début.

Le plan de ce chapitre se présente comme suit: tout d'abord, nous formalisons notre problème dans la section 5.1. Dans la section 5.2, nous explicitons les étapes de formulation de la fonction objectif. Dans la section 5.3, nous décrivons les comportements des agents du modèle et leurs interactions. Enfin, dans la section 5.4, nous présentons en détail la programmation du Recuit simulé.

5.1 formalisation du problème

5.1.1 Vocabulaire et notations

Pour la formalisation du problème et pour la suite de la description du modèle MARSA, nous avons adopté un vocabulaire qui est inspiré, en partie, des références [Cox et al., 1992], [Afnor, 1990], [Télémécanique, 1990], [Tomme, 1993] et [Verdebout, 1992].

Les données du problème sont caractérisées, d'une part, par le modèle de l'installation qui représente l'atelier et d'autre part, le processus opératoire qui décrit les tâches à effectuer.

Le **modèle de l'installation** est l'ensemble des lieux fixes ou éléments de travail dans l'atelier. On distingue deux types d'éléments: les ressources et les

accumulations.

Ressource: Nous la notons R_{ik}. C'est un élément qui peut être:

- une alimentation: un lieu physique dans lequel des objets sont introduits et à partir duquel ils doivent être pilotés par le système;
- une évacuation: un lieu par lequel des objets quittent le système et à partir duquel ils ne doivent plus être pilotés car ils sont soit livrés soit rejetés;
- une machine: une agrégation d'une arborescence de postes de travail synchrones en un poste équivalent.

Accumulation: C'est un élément de stockage dans lequel des objets à traiter séjournent pour une certaine durée avant d'être acheminés vers une destination.

Le passage des objets d'une ressource R_{ik} à une autre $R_{i+1\ h}$ nécessite un **temps de transfert** $Tt_{(ik)(i+1\ h)}$. C'est le temps nécessaire pour réaliser le changement de localisation d'un objet. Il dépend de la capacité de l'accumulation et de la vitesse de passage des objets dans cette accumulation.

Le temps de cycle de production est l'intervalle de temps entre deux sorties de produit. De même, le **temps de cycle d'une ressource** est l'intervalle de temps entre deux sorties d'objets. On distingue deux types:

Temps de cycle technique: C'est le temps de cycle qui représente la capacité réelle de traitement de la ressource. Nous le notons Tct_{ik}.

Temps de cycle estimé: Il tient compte des enrayages et des défauts que peut avoir une ressource. Il dépend d'une probabilité d'enrayage Pg_{ik} et d'un temps moyen de réparation de la ressource Tm_{ik}. Nous le notons Tce_{ik}.

Une ressource est enrayée si elle s'arrête suite à l'exécution au moins partielle d'un cycle de production.

Chaque accumulation $AC_{(ik)(i+1\ h)}$ existant entre deux ressources R_{ik} et $R_{i+1\ h}$ est caractérisée par sa capacité maximale de stockage $Ca_{(ik)(i+1\ h)}$ et du temps de passage d'un objet d'une case à une autre dans l'accumulation $Tpa_{(ik)(i+1\ h)}$.

Le **processus opératoire** P_j est un graphe d'actions qui énumère les traitements nécessaires à la réalisation du produit en question.

Une **opération** a pour rôle de modifier les caractéristiques, le temps et la localisation d'un objet. Ainsi, une **action** est une opération appliquée à un ensemble d'objets, ayant pour but ou pour effet de modifier les caractéristiques des objets, de les déplacer, ou encore, de laisser écouler le temps. Dans notre cas, nous nous intéressons seulement à la modification des caractéristiques des objets.

Pour son traitement, une action a besoin d'un outil bien déterminé devant être installé sur la ressource qui la traite. Le **temps de réglage** Tr_{ik} correspond au temps nécessaire de changement d'outil de la ressource R_{ik} pour passer d'un traitement d'une action à un autre, si les outils des deux actions successives sont différents.

Une **Demande Client** DC_j (figure 5.1) est composée d'un processus opératoire dont les actions sont à placer sur les ressources. C'est une demande de la part du client pour une quantité (consigne) Q_j d'une variante de produit à fabriquer. Par conséquent un **Ordre de fabrication** est déterminé par un groupement ou un sequencement d'un ensemble de DCs.

Le **rang de la DC** est sa position sur les ressources, il détermine par conséquent son ordre d'exécution. Ainsi, selon le rang qu'elle peut avoir elle a une date de début de traitement Dd_j et une date de fin Df_j, qui permettent de calculer sa durée approximative Du_j.

Une DC a une **date limite de livraison** (*delivery date*) Dl_j qui est une date fixée par l'utilisateur représentant une limite supérieure pour livrer au client le produit en question.

On appelle la différence entre la date limite de livraison d'une DC DC_j et sa date de fin de traitement la **marge** Mr_j.

Chaque action A_{ij}^k d'une DC_j est caractérisée par $\{R_{ik}\}$ qui est l'ensemble des k ressources possibles qui peuvent la traiter.

Selon le processus opératoire P_j une action est caractérisée par:

- l'ensemble des k actions précédentes $\{A_{i-1\ j}^k\}$, celles qui se trouvent immédiatement avant sur les ressources précédentes et
- l'ensemble des k actions suivantes $\{A_{(i+1)\ j}^k\}$, celles qui se trouvent immédiatement après sur les ressources suivantes.

De même, selon son affectation à une ressource R_{ik}, une action a:

- une action amont $A_{i\ (j-1)}^k$, celle qui sera traitée immédiatement avant sur la même ressource et

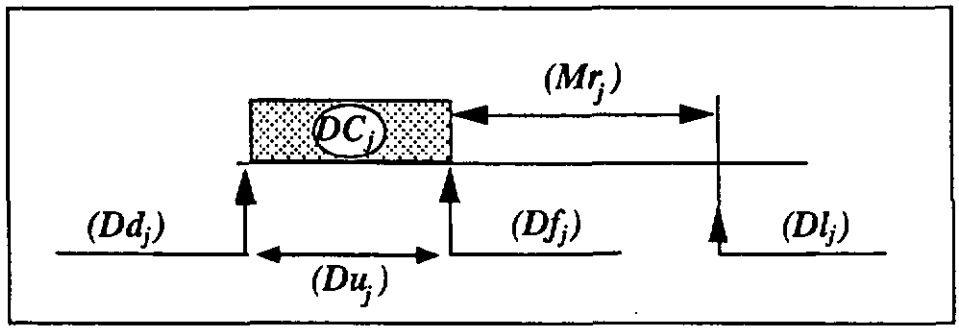


Figure 5.1: *Notations liées à une Demande client.*

- une action aval $A^k_{i\,(j+1)}$, celle qui sera traitée immédiatement après sur la même ressource.

Pour la suite des calculs, nous adoptons la méthode de **jalonnement en aval** qui permet de déterminer la date de fin de traitement au plus tôt.

Différentes relations et hypothèses sont établies entre ces informations.

5.1.2 Relations et hypothèses de travail

Nous avons conçu le modèle moyennant les relations et hypothèses suivantes:

- une action n'est pas préemptive et a besoin d'une et une seule ressource pour être exécutée;
- les ressources sont renouvelables (section 2.1.2);
- il existe une accumulation entre chaque couple de ressources successives. Cette accumulation peut être soit pleine soit vide;
- le temps de transfert d'un objet d'une ressource à une autre dépend de l'état de l'accumulation. Si l'accumulation est vide, il est égal au temps de passage de l'objet dans toute l'accumulation. Par contre si elle est pleine, il est égal au temps de traitement des objets qui sont stockés dedans, ainsi:

$$Tt_{(ik)(i+1\ h)} = \begin{cases} Ca_{(ik)(i+1\ h)} * Tpa_{(ik)(i+1\ h)} & \text{si } AC_{(ik)(i+1\ h)} \text{ est vide} \\ Ca_{(ik)(i+1\ h)} * Tce_{i+1\ h} & \text{sinon} \end{cases}$$

- nous travaillons dans un régime stationnaire avec un temps de cycle global déterminé par le temps de cycle estimé de la ressource la plus lente du modèle de l'installation. Le temps de cycle estimé d'une ressource R_{ik} isolée est le suivant:

$$Tce_{ik} = Tct_{ik} + (Pg_{ik} * Tm_{ik})$$

- les actions d'une même demande client DC_j possèdent des relations de précédence entre elles, de telle manière que si l'action $A^k_{i+1\ j}$ est l'action aval de A^k_{ij} sur une ressource $R_{i+1\ k}$, $R_{i+1\ k}$ ne peut commencer le traitement de $A^k_{i+1\ j}$ que si au moins la première opération de A^k_{ij} a été traitée par R_{ik}.

5.2 Fonction objectif

La fonction objectif est l'un des ingrédients de base des problèmes d'optimisation. En effet, sa formulation est très importante dans le sens où elle doit représenter le but du problème à atteindre. Cette représentation est en d'autres termes celle de tous les critères à optimiser; minimiser ou maximiser.
L'objectif des paragraphes suivants est de définir l'espace de recherche de solution et de spécifier les critères d'optimisation pertinents pour notre problème.

5.2.1 Espace de recherche

En général, la représentation d'un problème est une condition préalable pour la recherche d'une solution. En l'occurrence, dans notre problème d'ordonnancement dans un atelier de type *flow shop*, il est important de savoir à quel niveau le problème en totalité est affecté, de déterminer le début de la recherche, de situer le but à poursuivre par rapport à tout l'espace de recherche et de spécifier comment l'atteindre.

Pour répondre à toutes ces interrogations, nous définissons notre espace de recherche comme l'ensemble des états possibles. Un état est un placement des actions de DCs différentes, respectant les contraintes liées aux actions (les relations de précédences) et les contraintes liées aux ressources (les disponibilités des ressources).

Nous présentons l'espace d'état en termes de:

- état initial;
- états intermédiaires;
- états terminaux ou finaux;
- transformations.

Etat initial: A l'état initial, par lequel la recherche de la solution commence, aucune DC n'est placée, par conséquent toutes les ressources sont libres. C'est un état vide.

Etats intermédiaires: A un état intermédiaire, par lequel le système peut passer au cours de la recherche, des DCs sont placées. La possibilité d'avoir des DCs non placées ou placées partiellement sur les ressources (toutes les actions ne sont pas placées sur les ressources) n'est pas exclue.

Etats terminaux ou finaux: A un état final, qui représente une solution ou le but du problème, toutes les DCs doivent être placées.

Transformations: Une transformation représente le passage d'un état à un autre. Dans notre problème, une transformation est le placement d'une action sur une ressource.

Etant donné une telle représentation du problème, la recherche de solution devient un espace de recherche d'états (*state space search*) [Popovic et Bhatkar, 1994]. Les transformations d'états sont déterminées par la stratégie de recherche sélectionnée pour la poursuite de l'objectif selon des critères d'optimisation.

Ainsi, nous nous sommes basés effectivement sur:

- une fonction objectif, à expliciter par la suite, afin de comparer un état courant à un état nouveau;
- un outil d'optimisation combinatoire, le recuit simulé, pour décider d'accepter l'état nouveau ou non.

5.2.2 Critères d'optimisation

L'objectif est d'atteindre une séquence de demandes clients qui doit être optimale au sens des critères suivants:

- maximisation des marges;
- minimisation des temps de réglage.

Marges: On veut respecter au maximum les dates limites de livraison des DCs, par conséquent il est préférable que la marge qui sépare la date limite de livraison de la date de fin de traitement de chaque DC soit la plus grande possible. En d'autres termes, maximiser ces marges, c'est aussi minimiser les retards ou le nombre de DCs en retard.

Dans la fonction objectif, ce critère sera représenté par la somme des marges de l'ensemble des DCs placées, ou encore:

$$\sum_{j=1}^{nd} Mr_j$$

où nd est le nombre de DCs placées.

Temps de réglages: Généralement parlant, un réglage se produit chaque fois qu'une ressource doit changer d'outil pour traiter une action. Par conséquent, selon l'ordre des actions placées sur une ressource, celle-ci peut être amenée à changer d'outils plusieurs fois, ce qui implique un certain temps de réglage. Notre objectif est donc de déterminer un séquencement minimisant le temps de réglages. Dans la fonction objectif, ce critère sera représenté par le nombre de réglages multiplié par le temps d'un réglage de la ressource, ou encore:

$$nr_{ik} * Tr_{ik}$$

Sur la base des critères objectifs mentionnés, la fonction objectif h à maximiser est donc:

$$h = -(nr_{ik} * Tr_{ik}) + \sum_{j=1}^{nd} Mr_j$$

Dans le cas général, la complexité de la procédure de recherche locale dépend de la taille de l'ensemble des états voisins et du temps nécessaire pour évaluer le passage d'un état à son voisin. A mesure que le voisinage est grand (contient beaucoup d'états voisins), ce temps devient élevé.

Par ailleurs, il y a différentes manières de conduire la recherche locale dans l'espace d'états, et dans la majorité des cas la recherche locale commence par une solution arbitraire et se termine par un optimum local.

Nous voulons avoir une fonction objectif plus complète, tenant compte de tous les critères d'optimisation. Elle contribuera, ainsi, de manière plus efficace à la prise de décision des déplacements d'un état vers un autre.

5.2.3 Pénalités sur les caractéristiques de la solution

La manière que nous adoptons dans la recherche locale, et qui rejoint d'ailleurs les travaux de [Voudouris et Tsang, 1995], consiste à ajouter des pénalités à la fonction objectif selon les caractéristiques de la solution que nous souhaitons atteindre. Cet ajout permet:

- d'une part, d'accélérer les déplacements dans l'espace de recherche et
- d'autre part, d'introduire le caractère dynamique dans la solution objectif en ajoutant autant de caractéristiques désirées dans la solution à obtenir.

Nous proposons donc les caractéristiques suivantes de la solution souhaitée:

- toutes les DCs doivent être placées. Autrement dit, elles doivent toutes figurer dans la solution finale. Par conséquent, les DCs non placées sont pénalisées et nous représentons cette pénalisation par:

$$p1 * nda$$

où $p1$ est le paramètre de pénalisation et nda est le nombre de demandes clients en attente ou non placées;

- le nombre de DCs en retard, par rapport à leurs dates limites de livraison respectives, doit être le minimum possible. Par conséquent, les DCs en retard sont pénalisées et nous représentons cette pénalisation par:

$$p2 * ndr$$

où $p2$ est le paramètre de pénalisation et ndr est le nombre de demandes clients en retard.

Les valeurs des paramètres de pénalisations peuvent être fixées par l'utilisateur selon les priorités données aux différentes caractéristiques.

En dépit de l'ajout des pénalités, la recherche peut être guidée d'une façon biaisée, par le fait que dans plusieurs cas, une caractéristique peut dominer le reste des autres. Pour remédier à ce problème, nous ajoutons un paramètre régulateur des pénalités.

5.2.4 Paramètre régulateur

Le paramètre régulateur noté λ représente le degré au dessus duquel les caractéristiques vont affecter la recherche locale d'une solution. Ainsi, λ doit être le plus grand possible car:

- si λ est très petit, alors les changements d'un état à un autre vont être dirigés sur l'objectif combiné pour améliorer la solution. Il risque alors d'y avoir domination de certaines priorités sur d'autres;
- si λ est nul, alors la recherche locale risque d'être piégée dans un optimum local, car nous aurons en fait la fonction h;
- si λ est très grand, alors les changements d'un état à un autre vont éliminer de la solution les caractéristiques pénalisées.

Par conséquent, nous avons considéré quatre choix du paramètre λ et en avons jugé la pertinence en fonction des trois cas qui précèdent.

- $\lambda = \sum_{j=1}^{ndr} Mr_j$, où ndr est le nombre de demandes client en retard.
  Il aura toujours une valeur négative, donc ce premier choix est à éliminer;
- $\lambda = \sum_{j=1}^{nd} Mr_j$, où nd est le nombre de demandes client placées.
  Les valeurs négatives des Mr_j diminueront la valeur de λ, donc ce deuxième choix est aussi à éliminer;
- $\lambda = \sum_{j=1}^{ndb} Mr_j$, où ndb est le nombre de DCs qui ne sont pas en retard.
  Cette valeur sera toujours positive ou nulle, mais pas suffisante pour que λ soit très grand;
- $\lambda = \sum_{j=1}^{nd} \mid Mr_j \mid$, où nd est le nombre de demandes client placées.
  Ce choix nous donne la plus grande valeur de λ.

Sur la base des commentaires suivants, nous adoptons le quatrième choix et la fonction objectif complète et finale à maximiser est donc:

$$f = h - \sum_{j=1}^{nd} \mid Mr_j \mid * [(p1 * nda) + (p2 * ndr)]$$

Comment intervient cette fonction dans la résolution de notre problème d'ordonnancement, et à quel niveau est-elle utilisée dans le comportement de la société d'agents? C'est ce que nous nous proposons de préciser dans la section suivante qui contient une présentation des classes d'agents avec leurs comportements respectifs.

5.3 Agents et comportements

Conformément au modèle de l'Eco-résolution [Ferber et Jacopin, 1990], chaque agent possède des accointances; les agents qu'il connaît et avec qui il peut communiquer ainsi qu'un comportement cadre, indépendant de son type, basé sur la recherche de satisfaction. Le modèle de base est composé de deux types d'agents: les agents DC et les agents Ressources. Nous présentons, dans ce qui suit, la dynamique globale des agents et leurs comportements respectifs.

5.3.1 Dynamique globale

Un agent DC sollicite les agents Ressources par la demande de placement de ses actions. La demande de placement se fait en respectant les relations de précédences qui existent entre les actions, en les envoyant une à une. A chaque agent DC est associé un rang qui représente son ordre d'exécution sur les agents Ressources. Ce rang dépend du placement de la première action envoyée. Dans le modèle de base, la localisation de cette première action se fait aléatoirement. Les autres actions du même agent DC auront, par la suite, le même rang sur les agents Ressources suivants. Un agent Ressource sollicité par un agent DC exécute son comportement basé sur son propre recuit simulé. Ainsi

1. il génère un nouvel état comprenant obligatoirement la nouvelle action à placer, ensuite
2. il décide d'accepter ou non ce nouvel état.

Refuser cet état revient alors à refuser l'action à placer.

Comme expliqué dans [Daouas et al., 1994] et [Daouas et al., 1995a], ainsi que dans [Daouas et al., 1995d], nous allons maintenant détailler comment les agents du modèle communiquent et comment ils réagissent aux messages qu'ils reçoivent.

5.3.2 Agents Demande Client

Un agent DC a pour accointances tous les agents Ressource sur lesquels ses actions peuvent se placer. Il est satisfait quand toutes ses actions sont placées. Une fois satisfait, il ne fait rien.

En cas d'insatisfaction, son comportement consiste à demander le placement de l'action candidate. On distingue deux cas de candidature possibles pour chaque agent DC_j :

- si une action A_{ij}^k est placée sur un agent Ressource, les actions candidates sont les suivantes $\{A_{i+1\ j}^k\}$;

- si une action A_{ij}^k est refusée par l'agent Ressource et revient donc non placée, l'agent DC en question retire toutes ses actions et les actions candidates sont à nouveau $\{A_{1j}^k\}$.

ALGORITHME 2 Comportement d'insatisfaction de l'agent DC

SI liste-actions-courantes = vide **ALORS**
  Satisfaction *;toutes ses actions sont placées*
SINON
  A_{ij}^k < − − − **Extraire-Action** *;action à placer*
  R_{ik} < − − − Ressource(A_{ij}^k) *;ressource à affecter*
  agent(R_{ik}) < − − − AgentRessource *;agent Ressource correspondant*
  agent(R_{ik}) < − − − **Demande-Affectation**(A_{ij}^k, agent(R_{ik}))
FIN SI

La fonction qui exprime le comportement d'insatisfaction de l'agent DC est représentée par l'algorithme 2. Ainsi, l'agent DC est satisfait si toutes ses actions sont placées. Dans le cas contraire, l'action à la tête de la liste des actions courantes est choisie pour être affectée à l'agent Ressource correspondant.

5.3.3 Agents Ressource

Un agent Ressource a pour accointances tous les agents DC dont les actions sont susceptibles de se placer sur lui.
Il est satisfait quand il n'est plus sollicité. Il est d'autant plus satisfait que sa fonction objectif (voir section 5.2) est maximale.
Chaque fois qu'un agent Ressource est sollicité par un agent DC, son comportement d'insatisfaction basé sur le principe du recuit simulé se réalise en deux étapes: une génération d'un nouvel état E_g en présence de la nouvelle action et une prise de décision pour accepter ou non cet état.

Génération du nouvel état

En présence d'une action à placer envoyée par un agent DC, l'agent Ressource procède aux étapes suivantes:

Attribution du rang: Afin de couvrir au maximum l'espace des états, ou des séquences possibles, et afin d'éviter que le système ne boucle en fin de résolution, l'agent Ressource attribue de manière aléatoire un rang à l'agent DC qui le sollicite et ce seulement lors du placement de ses premières actions. Autrement dit, si l'action à placer est la première parmi les actions de l'agent DC (A_{1j}^k), l'agent Ressource lui attribue un rang aléatoire. Dans le cas contraire, elle aura le rang de son action précédente.

Calcul des dates: Ayant attribué un rang à l'action, l'agent Ressource lui calcule ses dates de début et de fin, ainsi que la date approximative de fin au plus tôt de l'agent DC correspondant. Pour le calcul des dates, nous utilisons la méthode de jalonnement en aval (voir section 5.1.1).
La date de fin au plus tôt de l'agent DC est la somme de la date de début de l'action et de la durée effective de ses actions suivantes, sans tenir compte des temps de réglages ni des temps de transferts entre les actions. Cette date est une information préalable sur l'état de l'agent DC, car dès le placement de sa première action, nous pouvons savoir s'il dépasse sa date limite ou non.

Prise de décision

Ayant déterminé le nouvel état généré E_g, l'agent Ressource calcule le coût $f(E_g)$ correspondant à la fonction objectif appliquée à l'état E_g et le compare au coût $f(E_c)$ correspondant à la fonction objectif appliquée à l'état courant E_c. La probabilité A_{cg} d'accepter l'état E_g est la suivante:

$$A_{cg} = \begin{cases} 1 & \text{si } f(E_g) \geq f(E_c) \\ 1 & \text{si } [(f(E_g) < f(E_c)) \text{ et } (random[0,1] < e^{\frac{f(E_g)-f(E_c)}{T(s)}})] \\ 0 & \text{sinon} \end{cases}$$

où $T(s)$ indique la tolérance de l'agent Ressource concerné à l'instant s, que nous explicitons dans la section suivante.

Par la suite, selon la décision de l'agent Ressource, le comportement des agents est le suivant:

- dans le cas où l'état généré est accepté, cet état devient l'état courant avec lequel le système continuera le traitement;
- dans le cas où l'état généré est refusé, l'agent Ressource informe l'agent DC qui le sollicite, de son refus. Par conséquent, cet agent DC retire toutes ses actions pour essayer de se placer à nouveau avec un autre rang.

La fonction qui exprime le comportement d'insatisfaction d'un agent Ressource est représentée par l'algorithmes 3. Ainsi, l'agent Ressource calcule son coût correspondant à son état courant et celui correspondant à son état généré, qui contient nécessairement une nouvelle action, et décide selon la différence Δ_{gc} des deux coûts. Si $\Delta_{gc} \geq 0$, l'état généré est meilleur donc il l'accepte automatiquement sinon un nombre décimal choisi aléatoirement entre 0 et 1, est comparé à la valeur exponentielle pour décider d'accepter l'état généré ou de le refuser.

Nous explicitons en détail dans la section 5.4, les techniques de recuit simulé utilisées localement par chaque agent Ressource.

ALGORITHME 3 Comportement d'insatisfaction de l'agent Ressource

$E_c <- - - -$ **Determiner-Etat-Courant**
$E_g <- - - -$ **Determiner-Etat-Généré**
$\Delta_{gc} <- - - - f(E_g) - f(E_c)$
SI $\Delta_{gc} \geq 0$ **ALORS**
  Acceptation(E_g)
SINON
  $nbdec <- - - - alea[0,1]$ *;nombre aléatoire entre* 0 *et* 1
  $expo <- - - - e^{\frac{\Delta}{tolérance}}$
  SI $nbdec < expo$ **ALORS**
    Acceptation(E_g)
    Baisse-Tolérance
  SINON
    Refus(E_g)
  FIN SI
FIN SI

5.3.4 Agent Interface

Pour le besoin de présenter à l'utilisateur une solution émergente qui présente l'état global du système, cet agent Interface a été ajouté (figure 5.2). Il active les agents DC pour aller solliciter les agents Ressource. Par contre il n'a aucun droit d'intervention dans le dialogue entre les agents Ressource et DC, son seul rôle est celui d'intermédiaire entre l'utilisateur et cette société d'agents; il connaît ainsi l'état de tous les agents. Ses accointances sont alors formées par l'ensemble des agents DC et l'ensemble des agents Ressource. Sa condition de satisfaction est remplie si les tolérances des agent $\{R_{1k}\}$ sont nulles et tous les agents DC sont placés. Dans ce cas il communique à l'utilisateur la meilleure solution obtenue.

5.4 Programmation du recuit simulé

5.4.1 Schéma de refroidissement

Le schéma de refroidissement, adopté et exposé précédemment (voir section 4.1.2), est relatif à chaque agent Ressource. Nous allons expliciter ses trois paramètres: une tolérance initiale, une manière de décroître la tolérance et un critère d'arrêt ou terminaison.

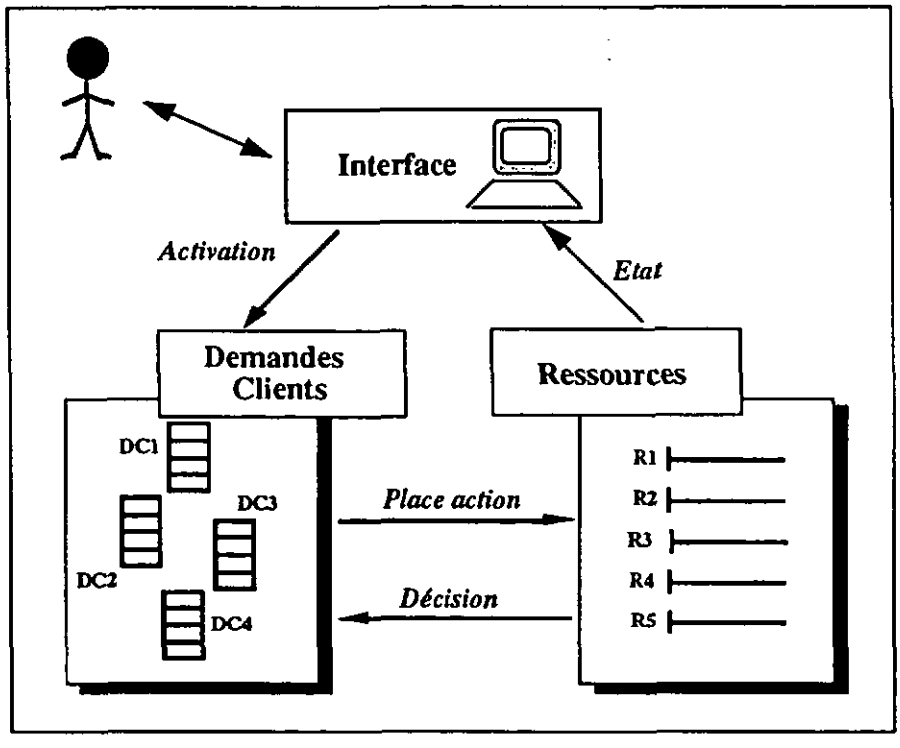


Figure 5.2: *Comportement général des agents.*

Tolérance initiale

La tolérance initiale doit être assez élevée, ainsi en pratique, nous la choisissons de telle manière que la probabilité d'acceptation soit supérieure à $\frac{1}{2}$, ce qui implique que $\forall\ \alpha$ et β, deux états voisins:

$$A_{\alpha\beta} = e^{\frac{f_\beta - f_\alpha}{T_{init}}} \succeq \frac{1}{2}$$

De même, en considérant $\Delta_{\alpha\beta} = f_\beta - f_\alpha$, nous obtenons

$$\frac{\Delta_{\alpha\beta}}{\ln 2} \leq T_{init}$$

Soient f_{Emax} l'estimation du coût du meilleur état souhaité et f_{Emin} l'estimation du coût de l'état le plus défavorable possible. f_{Emax} et f_{Emin} représentent, en fait, les deux bornes de coûts, ce qui vérifie la formule suivante:

$$| \Delta_{\alpha\beta} | \leq | f_{Emax} - f_{Emin} |$$

D'où la formule de la tolérance initiale:

$$T_{init} = \frac{| f_{Emax} - f_{Emin} |}{\ln 2}$$

Etant donné que la fonction objectif à maximiser est:

$$f = -(nr_i * Tr_i) + \sum_{j=1}^{nd} Mr_j - \sum_{j=1}^{nd} \mid Mr_j \mid *[(p1 * nda) + (p2 * ndr)]$$

Il nous reste à calculer les coûts f_{Emax} et f_{Emin} et leur différence. Pour cela, nous choisissons de:

- négliger les temps de réglages devant les marges;
- considérer $p1 = p2 = 1$;
- calculer la marge de chaque DC_j en la plaçant au premier rang. Ainsi sa Dd_j est nulle et sa Df_j est égale à sa durée approximative Du_j;
- calculer la marge de chaque DC_j en retard en la plaçant à partir de sa date limite. Ainsi sa Dd_j est égale à sa Dl_j.

Dans le meilleur état, tous les agents DC doivent être placés sur les agents Ressource sans aucun retard. Cela implique dans la fonction que la somme des marges est positive. Ainsi f_{Emax} est égale à:

$$f_{Emax} = \sum_{j=1}^{nd} Mr_j - \sum_{j=1}^{nd} \mid Mr_j \mid *(p1 * 0 + p2 * 0) = \sum_{j=1}^{nd} Mr_j$$

Dans l'état le plus défavorable, tous les agents DC sont placés en retard. Cela implique dans la fonction que la somme des marges est négative.
Ainsi f_{Emin} est égale à:

$$f_{Emin} = -\sum_{j=1}^{nd} Du_j - \sum_{j=1}^{nd} Du_j * p2 * nd$$

$$f_{Emax} - f_{Emin} = \sum_{j=1}^{nd} Mr_j + \sum_{j=1}^{nd} Du_j + \sum_{j=1}^{nd} Du_j * nd = \sum_{j=1}^{nd} Dl_j + \sum_{j=1}^{nd} Du_j * nd$$

La somme des marges, étant donné des DCs placées au premier rang, avec la somme des durées de ces mêmes DCs, est équivalente à la somme des dates limites. Finalement la tolérance initiale est donnée par:

$$T_{init} = \frac{\sum_{j=1}^{nd} Dl_j + \sum_{j=1}^{nd} Du_j * nd}{\ln 2}$$

Décroissance de la tolérance

En adoptant le recuit simulé comme outil d'optimisation, le système doit idéalement converger vers une solution optimale représentant la meilleure séquence de DCs. Une telle convergence n'est pas toujours possible en pratique, c'est pour cette raison que nous utilisons une décroissance infiniment lente et que seuls les états qui augmentent le coût sont acceptés.
Les expériences ont montré [Ghédira, 1994b] que le profil arithmétique donne un meilleur compromis dans notre cas. Ayant choisi le profil, nous avons déterminé empiriquement la raison arithmétique qui est de 7% de la tolérance initiale. Ainsi, la loi de décroissance est de la forme:

$$T_{n+1} = T_n - (0.07 * T_{init})$$

La tolérance n'est baissée que lorsque le nouvel état généré est accepté avec une baisse de coût.

Critère d'arrêt ou terminaison

Pour éviter que le système ne boucle indéfiniment, suite aux interactions entre les agents, un mécanisme de terminaison a été mis en oeuvre.
En effet, chaque agent Ressource diminue progressivement sa propre tolérance jusqu'à ce que le système se retrouve dans l'une des deux possibilités suivantes qui représentent son critère d'arrêt:

- les agents Ressource ne sont plus sollicités, autrement dit, tous les agents DC sont déjà placés;
- un ou plusieurs agents Ressource ont des tolérances nulles. Dans ce cas, ils acceptent de placer tout agent DC les sollicitant en lui attribuant un rang aléatoire, comme vu précédemment.

En revanche, la séquence d'agents DC obtenue à l'arrêt du traitement n'est pas nécessairement optimale (figure 5.3). Pour cette raison, nous ajoutons une phase d'amélioration que nous définirons dans la section suivante, et qui a pour but de permettre au système de visiter plus d'états dans l'espace de recherche.

5.4.2 Phase d'amélioration

Pour effectuer la phase d'amélioration, il faut que la condition suivante soit vérifiée: *"Les tolérances des premiers agents Ressource $\{R_{1k}\}$, parmi l'ensemble des agents Ressource, sont non nulles et tous les agents DC sont placés."*

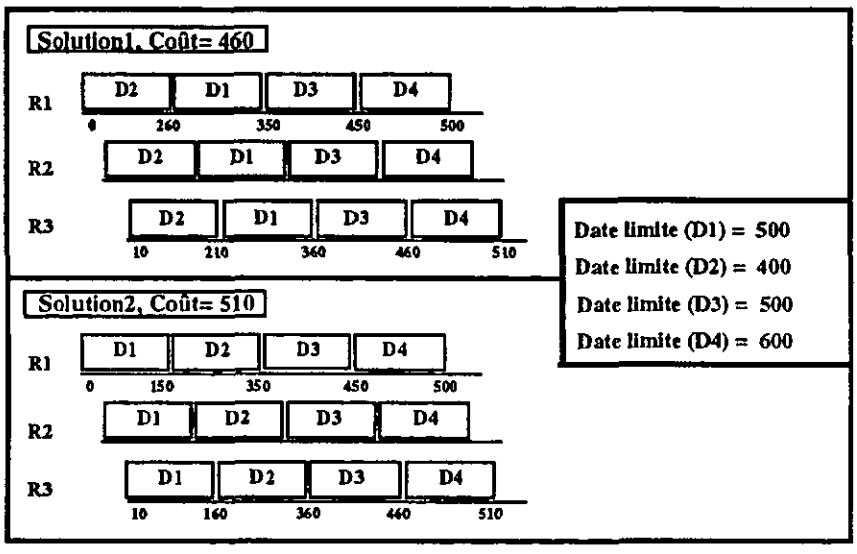


Figure 5.3: *Phase d'amélioration. La solution $S1$ obtenue n'est pas optimale. Il suffit de permuter D_1 et D_2 pour obtenir la solution $S2$ qui est meilleure.*

Le traitement de cette phase est défini comme suit: un des R_{1k} éjecte un agent DC parmi l'ensemble des agents placés. Cette opération perturbe le système à nouveau. ce qui lui permet de poursuivre son déplacement dans l'espace de recherche.

Le choix de l'agent DC pour être ejecté se fait de la façon suivante: *"S'il existe des agents DC en retard, il s'agit de l'agent DC le plus en retard. par conséquent il a l'occasion de se placer avant. Sinon, un agent DC est choisi aléatoirement."*

Enfin, le choix des $\{R_{1k}\}$ pour perturber le système provient du fait que les relations de précédences entre les actions obligent les agents DC à passer d'abord par les agents R_{1k} pour se placer.

5.4.3 Sauvegarde de la meilleure solution

Tout au long du traitement, l'agent Interface sauve le meilleur état rencontré du point de vu coût lors des déplacements dans l'espace de recherche.
A la fin du traitement, si le meilleur état sauvé contient des agents DC placés partiellement ou non placés, et n'est donc pas une solution; deux étapes sont à effectuer:

1. Compléter l'état sauvegardé en attribuant des rangs aléatoires aux agents DC restants, afin d'obtenir une solution.

2. Comparer cette solution avec la solution courante, qui est la dernière solution trouvée par le système.

En complétant l'état sauvegardé, cette manipulation nous donne l'occasion de retrouver une meilleure solution, au cas où nous avons perdu son chemin en cours de route.

Chapitre 6

Heuristiques

Les problèmes d'ordonnancement appartiennent à la classe des problèmes d'optimisation combinatoire. Pour résoudre ces problèmes, des algorithmes d'optimisation sont utilisés. Le but est d'essayer de trouver une solution optimale. Par ailleurs, tous les problèmes d'optimisation ne se résolvent pas en un temps polynomial; ils sont NP-difficiles. Dans de tels cas, il est souvent nécessaire de construire une structure de contrôle qui ne garantisse pas obligatoirement une "bonne" réponse, mais qui trouve le plus souvent une réponse satisfaisante. Par conséquent, on introduit des heuristiques[1] pour encore améliorer les algorithmes.

Dans ce chapitre, nous parlons dans la section 6.1 des argumentations pour l'utilisation d'heuristiques. Dans la section 6.2 et 6.3, nous explicitons les deux heuristiques adoptées.

6.1 Argumentation de l'utilisation des heuristiques

Une heuristique est une technique qui améliore l'efficacité d'un processus de recherche, parfois en sacrifiant le besoin de complétude.
En utilisant les heuristiques, nous pouvons espérer obtenir des solutions aux problèmes NP-difficiles, qui peuvent être non optimales en un temps raisonnable.
Sans heuristique, nous pouvons nous retrouver dans un cas d'explosion combinatoire. Cela peut être un argument suffisant en faveur de leur utilisation, mais il y en a bien d'autres. Nous citons les suivants:

[1]Le mot heuristique vient du mot grecque "heuriskein", qui signifie "découvrir", qui est aussi l'origine de "eureka" pour "j'ai trouvé", dérivé de l'exclamation d'Archimede quand il a découvert une méthode pour déterminer la pureté de l'or.

- très souvent, lorsque nous résolvons un problème, nous cherchons plutôt une solution satisfaisante qu'optimale. Autrement dit, nous cherchons une solution qui satisfasse un ensemble de besoins, et nous arrêtons la recherche dès qu'une telle solution est trouvée;

- en essayant de comprendre pourquoi une heuristique est efficace ou non pour un problème donné, nous sommes souvent amenés à comprendre plus profondément ce problème.

Dans notre modèle nous avons conçu deux heuristiques [Ghédira et Daouas, 1995] pour améliorer la recherche de la meilleure séquence de DCs. Le but essentiel de ces deux heuristiques est surtout d'améliorer la qualité de la solution:

- l'Heuristique du Meilleur Placement, utilisée au niveau de l'agent Ressource et basée sur le coût de la fonction locale de ce dernier;

- l'heuristique d'Arrangement, utilisée au niveau de l'agent Ressource et basée sur la date limite de livraison de l'agent DC qui demande à se placer.

6.2 Heuristique du Meilleur Placement (HMP)

Dans le modèle de base, chaque fois qu'un agent Ressource est sollicité par un agent DC, il génère un nouvel état comprenant l'action à placer. La génération du nouvel état consiste à:

1. Attribuer un rang à l'action. Si l'action est la première, en tenant compte des précédences entre les actions d'un même agent DC, le rang est aléatoire, sinon c'est celui de l'action précédente.

2. Calculer les dates de début et de fin de traitement, sachant que l'action a une position déterminée.

6.2.1 But de l'HMP

L'idée d'attribuer un rang aléatoire à une action a pour but, d'une part d'éviter que le système ne boucle et que les agents aient des interactions infinies, et d'autre part d'avoir une exploration plus ou moins exbaustive de l'espace de recherche. En revanche, l'aspect aléatoire peut amener la recherche vers une solution optimale comme il peut l'éloigner de cette solution.
L'HMP est utilisée au niveau de l'agent Ressource lors de son attribution d'un rang à une action à placer, pour attribuer à l'agent DC le meilleur rang possible.

6.2.2 Traitement de l'HMP

En présence d'une action, l'agent Ressource a la possibilité de lui attribuer un rang parmi un ou plusieurs, ceci dépend du nombre d'actions déjà placées.
A chaque rang possible, l'agent Ressource

1. calcule les dates correspondantes du nouvel état généré et

2. évalue sa fonction coût locale suivant cet état.

Le rang donnant le coût le plus élevé est choisi par l'agent Ressource pour être attribué à l'action et par conséquent l'état qu'il entraîne.

6.2.3 Avantages de l'HMP

L'utilisation de cette heuristique présente les deux avantages suivants:

- éviter de laisser échapper des états qui augmentent la valeur de la fonction coût, en choisissant le meilleur rang possible;

- éviter d'accepter des états qui causent une baisse de la tolérance locale de l'agent Ressource sans conduire à une solution optimale.

6.3 Heuristique d'Arrangement (HA)

Dans le modèle de base, chaque fois qu'un agent Ressource calcule les dates de début et de fin, en présence de l'action à placer, il calcule aussi la date approximative de fin au plus tôt de l'agent DC correspondant. Cette date nous donne une idée de la position de l'agent DC par rapport à sa date limite de livraison.

6.3.1 But de l'HA

En calculant la date de fin approximative de l'agent DC, l'agent Ressource peut savoir s'il y a un dépassement de la date limite ou non.
Sachant qu'on tient à placer l'agent DC de façon à respecter le plus possible sa date limite, l'HA est utilisée au niveau de l'agent Ressource qui, en cas de dépassement de la date limite, s'arrange pour avancer l'agent DC le plus possible dans le temps et ainsi agrandir sa marge.

6.3.2 Traitement de l'HA

Ayant attribué un rang et calculé les dates, l'agent Ressource vérifie si l'agent DC qui le sollicite dépasse sa date limite ou non.

S'il n'y a pas de dépassement, l'agent Ressource passe à l'étape de prise de décision. Sinon, il procède aux étapes suivantes:

1. Choisir, parmi les agents DC placés à gauche de l'agent DC en question (avant dans le temps), celui qui a la marge la plus grande.
2. Ejecter cet agent DC, ce qui revient à éjecter toutes ses actions placées.
3. Décaler le reste des agents DC vers la gauche en calculant les nouvelles dates.
4. Vérifier de nouveau le dépassement de l'agent DC.

Ces étapes sont répétées jusqu'à arriver à l'une des deux situations suivantes:

- l'agent DC se retrouve dans une position qui respecte sa date limite;
- il n'existe plus d'agent DC à gauche pouvant être éjecté.

Ainsi, dans la prise de décision, si le nouvel état est accepté, tous les agents DC à éjecter retirent leurs actions des agents Ressource sur lesquels ils sont placées.

6.3.3 Avantages de l'HA

L'utilisation de cette heuristique présente les deux avantages suivants:

- donner la possibilité d'empêcher le dépassement de la date limite, ou encore de réduire les retards, dès le placement de la première action de l'agent DC;
- permettre aux agents DC, ayant des marges plus grandes de se placer plus tard dans le temps et d'équilibrer ainsi les marges de l'ensemble des agents DC placés.

Nous verrons l'influence de ces deux heuristiques sur la solution dans les résultats obtenus par les expérimentations qui sont explicitées dans le chapitre suivant.

Chapitre 7

Expérimentations

L'objectif pratique de cette thèse est de développer un prototype à traitement continu où des DCs arrivent au fur et à mesure, sont ordonnancées puis envoyées une à une vers un simulateur d'assemblage sous forme d'ordres de fabrications à exécuter. Nous avons élaboré, auparavant, des expérimentations citées dans [Daouas et al., 1995b] et [Daouas et al., 1995c]. Nous voulons, d'une part mesurer les performances du modèle en comparant les résultats de la version de base avec ceux d'autres versions qui résolvent le même problème et, d'autre part évaluer l'influence de l'introduction des heuristiques sur les résultats de ce modèle.
Il est clair que l'aspect dynamique dans notre prototype d'ordonnancement présente une importance non négligeable dans notre travail. Néanmoins, nous nous occupons dans ce qui suit d'expérimenter l'aspect statique du problème; étant donné un ensemble de DCs et un ensemble de ressources à un instant donné. Par ailleurs, l'aspect dynamique est explicité dans le chapitre 8.

Ce chapitre s'organise comme suit: Dans la section 7.1, nous détaillons la génération des exemples. Dans la section 7.2 nous présentons les paramètres à mesurer. Dans la section 7.3, nous énumérons les versions de tests avec leurs traitements respectifs. Dans la section 7.4, nous commentons les résultats trouvés. Enfin, dans la section 7.5 nous concluons.

7.1 Génération des exemples

Les expérimentations ont été effectuées sur des exemples générés aléatoirement.

7.1.1 Paramètres de génération

Les paramètres de génération déterminent les caractéristiques du problème d'ordonnancement à traiter. Nous avons fixé comme paramètres pour la génération des exemples de problèmes à tester:

- le nombre de DCs;
- le nombre de ressources;
- le degré de difficulté.

Nombre de DCs: Avec la variation du nombre de DCs, le nombre de combinaisons possibles varie. Par conséquent, l'espace de recherche change de taille.

Nombre de ressources: En variant le nombre de ressources, on varie le nombre d'actions des DCs à ordonnancer. Cette variation ne touche pas la taille de l'espace de recherche, mais le volume des interactions entre les ressources ainsi que la remise en question des solutions partielles rencontrées au cours de la recherche.

Degré de difficulté: La date limite de livraison d'une DC est formulée en fonction d'un paramètre α:

$$Dl_j = Du_j + Du_{moy} * \alpha * nd$$

où Du_j est la durée approximative de DC_j, qui ne tient pas compte des temps de réglages et de transferts, Du_{moy} est la durée approximative d'une DC moyenne ayant une consigne moyenne égale à 50 et nd est le nombre de DCs.

On remarque que plus α est petit, plus la date limite est petite et plus le problème est difficile et contraint.

On appelle degré de difficulté le paramètre $\frac{1}{2\alpha}$. Il est facile de remarquer que plus le degré de difficulté est élevé plus le problème devient difficile.

Nous voulons voir quelle est l'influence des variations de ces trois paramètres dans les résultats de nos tests. Pour ceci nous sommes amenés à croiser certains paramètres et à en fixer d'autres pour aboutir aux exemples de tests.

7.1.2 Génération aléatoire

Les exemples de test ont été générés aléatoirement comme suit:

- dans un premier temps, nous fixons le nombre de ressources à 5 et nous croisons le nombre de DCs, ayant les valeurs $\{2, 4, 6, 8, 10\}$, avec le degré de difficulté ayant les valeurs $\{0.5, 0.7, 1\}$;

- dans un deuxième temps, nous fixons le nombre de DCs à 5 et le degré de difficulté à 0.7 et nous varions le nombre de ressources en lui attribuant les valeurs $\{2, 4, 6, 8, 10\}$.

Vu le type d'atelier que nous traitons, vu que les DCs arrivent au fur et à mesure et vu qu'elles sont envoyées après ordonnancement à l'exécution, notre horizon de planification adopté ne doit pas être long; nous choisissons donc qu'à un instant donné le nombre de DCs maximum à ordonnancer n'excède pas 10. C'est la raison pour laquelle nous varions le nombre de DCs de 2 à 10.

Pour chaque DC, on procède aux générations suivantes:

- la quantité de produit ou consigne est déterminée par un nombre aléatoire entre 1 et 100 selon une distribution uniforme;

- la date limite de livraison est déterminée par la formule suivante:

$$Dl_j = Du_j + Du_{moy} * ALEA[\alpha - 0.1, \alpha + 0.1] * nd$$

où α est égal à $\{0.5, 0.7, 1\}$.

Pour chaque ressource, on procède aux générations suivantes:

- le temps de cycle estimé est un nombre décimal aléatoire entre 0.1 et 3;

- la probabilité d'enrayage est un nombre décimal aléatoire entre 0.1 et 0.5;

- le temps moyen de réparation est un nombre décimal aléatoire entre 0.1 et 0.5.

Etant donné l'aspect aléatoire du recuit simulé qui est l'outil d'optimisation combinatoire utilisé dans notre modèle, on exécute 10 fois chaque exemple généré.

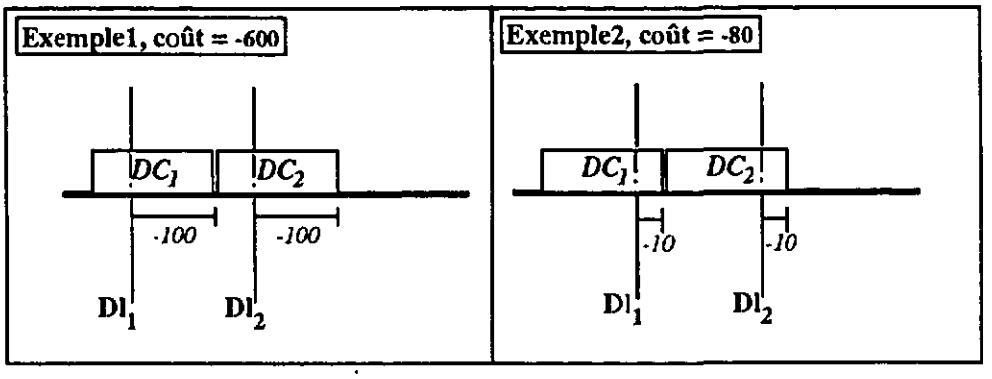


Figure 7.1: *Deux solutions ayant le même nombre de DCs en retards égal à 2, mais des coûts différents.*

7.2 Paramètres à mesurer

Dans les résultats des expérimentations nous avons deux centres d'intérêts:

- d'une part, nous voulons développer un prototype à traitement continu. Les DCs ordonnancées (ordres de fabrications) sont envoyées à un simulateur d'assemblage pour être exécutées, et leur envoi doit être effectué de telle manière que le simulateur travaille continuellement. Par conséquent, nous nous intéressons au temps d'obtention de la solution, autrement dit, de la séquence de DCs;

- d'autre part, la séquence de DC envoyée pour exécution doit être optimale selon les critères d'optimisation déjà mentionnés au chapitre 5. Par conséquent, nous nous intéressons à la qualité de la solution trouvée.

Pour mesurer le temps d'obtention de la solution, nous avons utilisé un compteur en millisecondes du temps mis depuis le début du traitement jusqu'à la rencontre de la meilleure solution du point de vue du coût global. Pour mesurer la qualité, nous l'avons représentée par le coût de la solution obtenue.

7.3 Versions de test du modèle

Notre modèle, ainsi conçu, est le résultat de la combinaison de deux domaines différents qui sont les systèmes multi-agents et les techniques du recuit simulé. D'une part, nous avons élaboré nos expérimentations dans le but de justifier notre modèle en faisant des tests sur les performances et en comparant ces performances

avec celles d'autres modèles n'ayant pas le même traitement. D'autre part, nous sommes intéressés aux performances que peut donner notre modèle en lui introduisant les heuristiques HMP et HA, explicitées dans le chapitre précédent.

C'est pour cette raison que nous avons programmé nos expérimentations en deux parties: une justification du modèle de base et une justification des deux heuristiques. Nous mentionnons dans les versions ci-après les différences par rapport à la version de base.

ALGORITHME 4 Version RC

REPETER
  Demande-Affectation(DC_j) *;demande de placement d'une DC*
  Générer-Etat-Global(EG_g) *;état de toute les ressources*
  SI Décision-RecuitG = Acceptation **ALORS**
    $EG_c <- - - - EG_g$ *;placement de toute la DC sur les ressources*
  FIN SI
JUSQU'A Liste-DC = vide *;toutes les DCs sont placées*

ALGORITHME 5 Version RCMA

REPETER
  Demande-Affectation-Action(A^k_{ij}) *;demande de placement de la i^{me} action de la j^{me} DC sur la K^{me} ressource*
  Générer-Etat-ressource(E_g)
  SI Décision-RecuitG = Acceptation **ALORS**
    $E_c <- - - - E_g$ *;placement de l'action sur la ressource*
  SINON
    DC-se-Retire *;la DC retire toutes ses actions*
  FIN SI
JUSQU'A Liste-DC = vide *;toutes les DCs sont placées*

7.3.1 Justification du modèle de base

Deux versions ont été établies pour être comparées au modèle de base:

Version RC pour Recuit simulé Centralisé (voir algorithme 4), où le système est gouverné par un seul recuit global centralisé. Sachant qu'un état du système représente l'état global comprenant celui de toutes les ressources, le recuit simulé prend la décision d'accepter un nouvel état du système ou pas.

Chaque DC à placer est introduite en totalité sur les ressources en plaçant chacune de ses actions sur sa ressource correspondante. Ainsi, l'état généré comprend nécessairement la nouvelle DC à placer.

Version RCMA pour Recuit simulé Centralisée avec Multi-Agents (voir algorithme 5), où la société d'agents est représentée par les agents DCs et les agents Ressources. Les comportements des agents respectifs sont semblables à ceux des agents du modèle de base à part que la décision appartient au système, muni d'un recuit centralisé, et pas à l'agent Ressource.

Puisque dans ces deux versions il existe un seul recuit simulé, il n'y a qu'une seule tolérance. Cette tolérance est baissée chaque fois que le nouvel état est accepté avec baisse de satisfaction. Par conséquent, la tolérance atteindra le niveau nul plus rapidement. Pour avoir une comparaison équitable, nous avons modifié le scbéma de refroidissement en rapport avec le recuit simulé en prenant pour chacun des exemples de test une tolérance initiale égale au nombre de ressources multiplié par la tolérance initiale normale. Ainsi

$$T_{init_{RC}} = T_{init_{RCMA}} = nr * T_{init_{Base}}$$

avec nr le nombre de ressources.

En examinant les algorithmes de RC et RCMA, nous pouvons remarquer la différence de traitement de ses deux versions.

- dans RC, le choix du placement d'une DC se fait une seule fois, puisque chaque DC est placée en totalité. Par contre, dans RCMA, chaque DC est placée action par action, donc, tant que la DC n'est pas placée en totalité son placement peut être remis en question: le refus d'une action implique l'éjection de toute la DC qui a déjà placé une partie de ses actions. Donc RCMA parcour davantage l'espace de recherche;

- RC correspond à un ordonnancement en entrée alors que RCMA semble réaliser implicitement un ordonnancement sur les ressources critiques.

Par conséquent, le traitement de RCMA pourrait nous donner une qualité meilleure des solutions.

D'autre part, parmi ceux qui ont utilisé le recuit simulé pour les problèmes d'ordonnancement en *flow shop*, nous citons [Zegordi et al., 1995] et [Ogbu et Smith, 1990]. La différence des versions RC et RCMA par rapport aux approches proposées dans les deux articles se présente dans:

- la fonction objectif à optimiser. Le premier optimise une fonction appelée *early/tardy* qui est une généralisation du problème du retard maximum pondéré. Quant au deuxième, il optimise la durée totale de l'ordonnancement qui est le critère le plus couramment utilisé;

- la programmation du recuit simulé. D'abord, l'état initial choisit appartient à l'ensemble des solutions possibles, alors que dans notre cas nous commencons par un état vide, nous passons par des états intermédiaires pour atteindre l'état solution. Ensuite, la température est diminuée après un nombre d'itérations, alors que dans notre cas, la tolérance n'est diminuée qu'en cas d'acceptation avec baisse de coût.

7.3.2 Justification des heuristiques

Trois versions ont été établies pour être comparées au modèle de base:

Version BaseH1 où seule l'heuristique HMP est ajoutée à la version de base. Nous rappelons qu'elle consiste à attribuer à l'agent DC qui veut placer sa première action le rang donnant le meilleur coût local de l'agent Ressource.

Cette heuristique remplace l'attribution aléatoire du rang dans le modèle de base.

Version BaseH2 où seule l'heuristique HA est ajoutée à la version de base. Nous rappelons qu'elle consiste à s'arranger pour placer l'action en respectant le plus possible la date limite de livraison de l'agent DC correspondant.

Par rapport au traitement dans la version de base, le traitement de cette heuristique aide à minimiser le nombre de DCs en retards.

Version BaseH12 où les deux heuristiques HMP et HA sont ajoutées à la version de base.

Ainsi, nous avons six versions à lancer sur chacun des exemples test générés.

Pendant les premiers tests que nous avons effectués, les mesures de la qualité de la solution ont été faites par le nombre de DCs en retards. Nous avons, par la suite, laissé tomber cette manière de représenter car nous avons remarqué qu'il est possible d'avoir deux solutions de deux exemples différents ayant chacune le même nombre de DCs en retard mais des coûts différents (voir figure 7.1). Ce résultat dépend des marges qui existent dans les deux solutions.
Il s'est avéré qu'il vaut mieux comparer les coûts, d'autant plus que notre fonction objectif tient compte de tous les critères d'optimisation en même temps.

7.4 Résultats expérimentaux et commentaires

Selon le même principe, nous avons présenté les résultats des versions pour justifier le modèle de base et ceux pour justifier les heuristiques séparément. De plus, il est à remarquer que dans le cas des premières versions, il est insensé de comparer les temps de solution des versions RC et RCMA pour deux raisons:

- dans RC, une transformation est un placement de toute une DC. Autrement dit c'est seulement les premières ressources $\{R_{1k}\}$ qui décident du placement. Par contre dans RCMA une transformation est un placement d'une action, donc toutes les ressources peuvent s'exprimer et participer au placement;
- dans le cas de RC, si une DC est placée elle l'est pour toute les ressources en même temps, alors que dans le cas de RCMA, une DC peut être placée partiellement sur quelques ressources (il existe encore des actions non placées) et par la suite éjectée à cause du refus de placement d'une action suivante.

Nous nous contentons, alors, de comparer RC et RCMA selon leurs qualités uniquement. Par contre, RCMA et Base ont des traitements de même nature. Une comparaison basée sur la qualité et le temps est donc justifiée.

Toutefois, deux types de campagnes d'expérimentations ont été lancées:

- le croisement du degré de difficulté avec le nombre de DCs en fixant le nombre de ressources;
- la variation du nombre de ressources en fixant le nombre de DCs et le degré de difficulté.

Nous avons choisi de présenter des tableaux qui illustrent l'évolution des paramètres à mesurer suivant les valeurs des différents paramètres de génération, plutôt que des courbes. La raison est qu'il y a beaucoup d'interprétations à tirer des résultats expérimentaux obtenus. Nous représentons seulement deux courbes en conclusion pour récapituler.

Nous dirons que la qualité est meilleure si le coût de la solution trouvée est plus élevé et nous dirons que le temps est meilleur si le temps d'obtention de la solution est inférieur.

7.4.1 Résultats justifiant le modèle de base

Première campagne d'expérimentation

Le tableau 7.1 montre que le coût pour RCMA est meilleur que celui pour RC, ceci indépendamment du nombre de DCs et du degré de difficulté. Ce résultat prouve

Degré difficulté \ # DCs		2	4	6	8	10
0.5	RC	-338	-1795	-6447	0	-2260
	RCMA	-226	-1007	-4679	0	0
	Base	-226	-1185 *	-4995 *	0	-1635 *
0.7	RC	-257	-174	-8353	-12507	-15644
	RCMA	-170	0	-6887	-8536	-11285
	Base	-170	0	-8056 *	-9231 *	-13963 *
1	RC	0	-1420	-10310	-13653	-33288
	RCMA	0	-543	-8685	-11641	-29480
	Base	0	-536	-8685	-11091	-29570 *

Tableau 7.1: *Tableau des coûts avec un nombre de ressources égal à* 5.

Degré difficulté \ # DCs		2	4	6	8	10
0.5	RCMA	1396	6109	6093	3739	6472
	Base	1341	3456	4398	3654	6116
0.7	RCMA	1120	2206	6358	7307	7488
	Base	1021	2109	4240	6836	7117
1	RCMA	805	5081	6305	6757	8152
	Base	786	3207	5087	5628	7262

Tableau 7.2: *Tableau des temps en millisecondes avec un nombre de ressources égal à* 5.

Gain / Perte / Degré difficulté	Temps	Coût
0.5	+0.32	- 0.05
0.7	+0.11	- 0.09
1	+0.27	0

RCMA -----> Base

Gain / Perte / # DCs	Temps	Coût
2	+0.22	0
4	+0.27	- 0.05
6	+0.26	- 0.07
8	+0.08	- 0.01
10	+0.23	- 0.11

Tableau 7.3: *Tableaux de synthèse avec un nombre de ressources égal à* 5.

Degré difficulté \ # DCs			2	4	6	8	10
0.5	RC	Coût Max Fréquence	-226 4	-995 1	-3840 1	0 10	0 6
	RCMA	Coût Max Fréquence	-226 10	-955 8	-3840 1	0 10	0 10
	Base	Coût Max Fréquence	-226 10	-955 6	-4098 * 1	0 10	0 10
0.7	RC	Coût Max Fréquence	-171 5	0 7	-7019 3	-5540 1	-8984 1
	RCMA	Coût Max Fréquence	-171 10	0 10	-6077 1	-5728 1	-7572 1
	Base	Coût Max Fréquence	-171 10	0 10	-6110 * 1	-5728 * 1	-8304 * 1
1	RC	Coût Max Fréquence	0 10	-503 3	-8712 1	-8681 1	-25344 1
	RCMA	Coût Max Fréquence	0 10	-445 5	-7754 1	-8629 1	-19395 1
	Base	Coût Max Fréquence	0 10	-445 3	-7754 1	-8629 1	-16510 1

Tableau 7.4: *Tableau des meilleurs coûts avec un nombre de ressources égal à* 5.

l'apport des systèmes multi-agents, concernant les déplacements dans l'espace de recherche, qui en pourcentages calculés sur les différences de coûts, améliorent la qualité de 34% en moyenne.

Par rapport à RCMA, la qualité dans Base s'améliore à mesure que le nombre de DCs et le degré de difficulté augmentent. Ce résultat montre l'avantage de l'aspect distribué lié aux systèmes multi-agents pour les problèmes NP-difficiles et de taille plus grande.

On remarque que les cases du tableau munies d'une étoile présentent une baisse du coût dans Base. En revanche, et puisque nous souhaitons avoir un bon compromis entre la qualité et le temps, cette détérioration de la qualité doit être compensée par une amélioration du temps de la solution. Le tableau 7.2 montre que le temps dans Base est meilleur que celui dans RCMA, indépendamment des variations du nombre de DCs et du degré de difficulté.

Le tableau 7.3 est une synthèse issue des deux tableaux précédents qui montre les pourcentages de gains et de pertes en temps et en qualité. Ainsi, selon les variations du degré de difficulté, le compromis est meilleur pour les problèmes très faciles ou très difficiles. De même, selon les variations du nombre de DCs le compromis est meilleur pour les problèmes ayant un nombre de DCs inférieur à 6 ou égal à 10.

Dans les résultats que nous venons de citer, des valeurs moyennes ont été calculées à partir des 10 lancements. Nous nous sommes intéressés, dans une deuxième phase, au meilleur coût trouvé par les différentes versions et la fréquence à laquelle ce coût est atteint sur les 10 lancements. Ces résultats sont illustrés dans le tableau 7.4. D'une manière générale, et pour les trois versions, si le nombre de DCs est inférieur à 6, les coûts maxima sont atteints avec une fréquence moyenne de 7. Si le nombre de DCs est égal à 6, les coûts maxima sont atteints avec une fréquence moyenne de 1 et si le nombre de DCs est supérieur ou égal à 8, les coûts maxima sont atteints avec une fréquence moyenne de 9 pour le degré de difficulté 0.5 et de 1 pour les degrés de difficulté 0.7 et 1.

Les coûts maxima atteints par la version Base sont supérieurs ou égaux à ceux atteints par la version RCMA qui, de son côté, présente de meilleurs coûts que la version RC. Dans les cases qui contiennent une étoile, le meilleur coût atteint présente une légère détérioration dans la version Base. Néanmoins, le temps mis pour atteindre ces solutions est moins élevé, donc si le traitement est poursuivi dans ce cas, on pourrait atteindre des coûts plus élevés.

# Res / Version	2	4	6	8	10
RC	-2247	-1542	-4824	-2829	-1987
RCMA	-1160	-1010	-3165	-1575	-806
Base	-1090	-937	-2920	-1570	-782

Tableau 7.5: *Tableau des coûts avec un nombre de DCs égal à* 5 *et un degré de difficulté égal à* 0.7.

# Res / Version	2	4	6	8	10
RCMA	1874	4272	5462	8746	15704
Base	1478	3085	4480	6405	7927

Tableau 7.6: *Tableau des temps en millisecondes avec un nombre de DCs égal à* 5 *et un degré de difficulté égal à* 0.7.

Deuxième campagne d'expérimentation

Les résultats du tableau 7.5 rejoignent ce qui a été montré dans la campagne précédente, du fait que la qualité dans RCMA est nettement meilleure que celle dans RC; elle s'améliore de 43% en moyenne. Dans ce même tableau, nous remarquons que la qualité s'améliore aussi dans la version Base mais seulement de 4% environ. Pour compenser cette faible amélioration, le temps de la solution illustré dans le tableau 7.6 s'améliore de RCMA à Base de 28% en moyenne.

Le tableau 7.7 montre qu'en variant le nombre de ressources la qualité des solutions s'améliore en allant de RC, à RCMA puis à Base. Nous remarquons que les fréquences sont en général inférieures ou égales à 2 pour tous les exemples. Ce résultat montre que le nombre de ressources n'influence pas la complexité du problème. Par contre, le degré de difficulté étant de 0.7 dans cette campagne, les exemples sont moyennement difficiles, ce qui explique les fréquences trouvées.

7.4.2 Résultats justifiant les heuristiques

Dans les tableaux que nous présentons et qui illustrent les résultats justifiant les heuristiques, nous nous intéressons aux comparaisons des versions Base et Ba-

# Ressources / Degré difficulté		2	4	6	8	10
RC	Coût Max	-1185	-1028	-3176	-1319	-781
	Fréquence	1	1	1	1	1
RCMA	Coût Max	-833	-908	-2794	-1134	-652
	Fréquence	2	2	1	1	4
Base	Coût Max	-829	-855	-2920 *	-1106	-648
	Fréquence	1	1	2	1	1

Tableau 7.7: *Tableau des meilleurs coûts avec un nombre de DCs égal à* 5 *et un degré de difficulté égal à* 0.7.

seH12. Malgré ce fait, nous avons choisi de présenter les résultats de BaseH1 et BaseH2 car nous avons remarqué que les deux heuristiques introduites séparément dans la version de base n'améliorent pas toujours les performances du modèle.

Première campagne d'expérimentation

Dans le tableau 7.8, les coûts de la version BaseH12 sont toujours supérieurs à ceux de Base. Ce résultat montre l'avantage de combiner les deux heuristiques afin d'améliorer la qualité des solutions. Par contre, le temps représenté dans le tableau 7.9 est plus élevé pour la version BaseH12. Nous rappelons que les heuristiques ont pour but d'améliorer la qualité des solutions. Ceci nous donne le choix d'utiliser la version Base si nous donnons la priorité au temps, ou la version BaseH12 dans le cas contraire.

Nous présentons, dans le tableau de syntbèse 7.11, les pourcentages de gains et pertes en variant le degré de difficulté puis le nombre de DCs . A première vue, on remarque que les pourcentages de gain en qualité sont relativement moins élevés que les pourcentages de pertes en temps. Mais, bien que le compromis soit donné à titre indicatif, ce résultat est bon en soi, vue le paysage de notre fonction objectif: nous pouvons avoir deux optima locaux très proches du point de vue coût mais séparés par un long plateau (voir figure 7.2) ou une région dense en pics (voir figure 7.3). Autrement dit, bien que la différence de coûts entre les deux optima locaux puisse être faible, le temps mis pour passer d'un optimum à un autre peut être très important.

En examinant le tableau 7.10, nous remarquons que, lorsque le nombre de DCs

Degré difficulté \ # DCs		2	4	6	8	10
0.5	Base	-226	-1185	-4995	0	-1635
	BaseH1	-226	-1348	-4429	0	-490
	BaseH2	-224	-1360	-5062	0	-2042
	BaseH12	-223	-1096	-4365	0	0
0.7	Base	-170	0	-8056	-9231	-13963
	BaseH1	-170	0	-7982	-8097	-11227
	BaseH2	-169	-109	-7349	-9617	-13844
	BaseH12	-168	0	-7148	-6905	-11908
1	Base	0	-536	-8685	-11091	-29570
	BaseH1	0	-842	-9099	-10373	-20880
	BaseH2	0	-788	-9700	-11962	-29827
	BaseH12	0	-529	-8299	-8620	-18191

Tableau 7.8: *Tableau des coûts avec un nombre de ressources égal à* 5.

Degré difficulté \ # DCs		2	4	6	8	10
0.5	Base	1341	3456	4398	3654	6116
	BaseH1	1302	2655	3467	5756	7804
	BaseH2	1908	3093	4804	4348	5535
	BaseH12	1926	3500	7056	4942	8045
0.7	Base	1021	2109	4240	6836	7117
	BaseH1	1161	2539	4599	6413	9942
	BaseH2	1610	2092	5302	5812	7138
	BaseH12	1765	3035	5654	7260	10437
1	Base	786	3207	5087	5628	7262
	BaseH1	875	2848	7216	7659	10173
	BaseH2	966	3557	5811	6274	8423
	BaseH12	878	3271	10089	8822	10794

Tableau 7.9: *Tableau des temps en millisecondes avec un nombre de ressources égal à* 5.

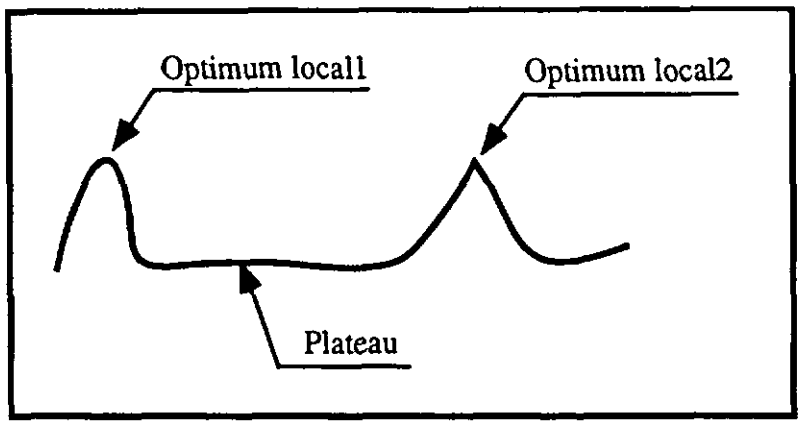


Figure 7.2: *Deux optima locaux ayant des coûts très proches mais séparés par un long plateau.*

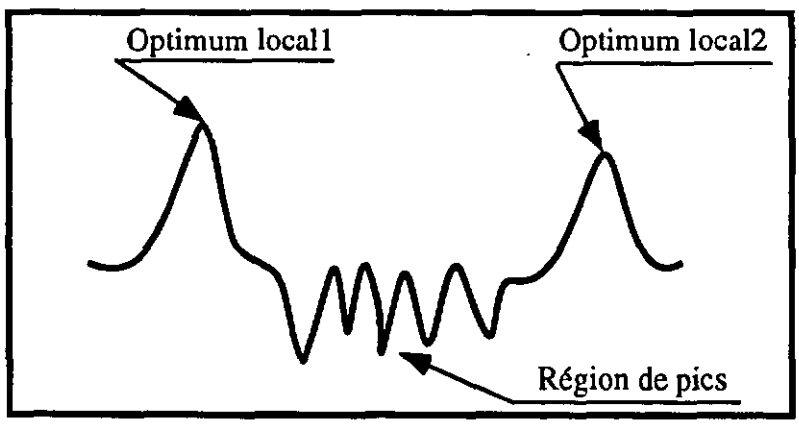


Figure 7.3: *Deux optima locaux ayant des coûts très proches mais séparés par une région dense en pics.*

Degré difficulté \ # DCs			2	4	6	8	10
0.5	Base	Coût Max Fréquence	-226 10	-955 6	-4098 1	0 10	0 10
	BaseH1	Coût Max Fréquence	-226 10	-955 2	-3862 3	0 10	0 10
	BaseH2	Coût Max Fréquence	-223 7	-955 3	-3882 1	0 10	0 8
	BaseH12	Coût Max Fréquence	-223 9	-955 8	-3776 2	0 10	0 10
0.7	Base	Coût Max Fréquence	-171 10	0 10	-6110 1	-5728 1	-8304 1
	BaseH1	Coût Max Fréquence	-168 1	0 10	-6110 1	-4886 2	-6404 2
	BaseH2	Coût Max Fréquence	-168 6	0 8	-6149 1	-6160 1	-7224 1
	BaseH12	Coût Max Fréquence	-168 10	0 9	-6007 1	-4734 1	-6404 3
1	Base	Coût Max Fréquence	0 10	-445 3	-7754 1	-8629 1	-16510 1
	BaseH1	Coût Max Fréquence	0 10	-445 6	-7631 1	-7002 1	-16510 1
	BaseH2	Coût Max Fréquence	0 10	-445 1	-8819 1	-8352 1	-20144 1
	BaseH12	Coût Max Fréquence	0 10	-445 8	-7622 2	-7002 2	-12353 2

Tableau 7.10: *Tableau des meilleurs coûts avec un nombre de ressources égal à* 5.

Gain / Perte / Degré difficulté	Temps	Coût
0.5	- 0.36	+0.24
0.7	- 0.39	+0.10
1	- 0.42	+0.13

Base -----> BaseH12

Gain / Perte / #DCs	Temps	Coût
2	- 0.42	0
4	- 0.18	+0.02
6	- 0.63	+0.09
8	- 0.32	+0.15
10	- 0.41	+0.5

Tableau 7.11: *Tableau de synthèse avec un nombre de ressources égal à* 5.

# Res / Versions	2	4	6	8	10
Base	-1090	-937	-2920	-1570	-782
BaseH1	-1232	-1188	-4549	-1505	-1359
BaseH2	-1161	-1308	-4450	-1678	-1520
BaseH12	-912	-873	-2796	-1500	-740

Tableau 7.12: *Tableau des coûts avec un nombre de DCs égal à 5 et degré de difficulté égal à 0.7.*

# Res / Versions	2	4	6	8	10
Base	1478	3085	4480	6405	7927
BaseH1	1697	2850	4791	5786	11790
BaseH2	1570	2986	5549	7897	11029
BaseH12	2342	3624	5641	9091	10308

Tableau 7.13: *Tableau des temps en millisecondes avec un nombre de DCs égal à 5 et un degré de difficulté égal à 0.7.*

est inférieur à 6, les coûts maxima atteints par la version BaseH12 ont des fréquences très élevées indépendamment du degré de difficulté; elles sont de 90% en moyenne. Lorsque le nombre de DCs est égal à 6, qui est une région du tableau qui correspond au nombre de DCs à peu près égal au nombre de ressources, les pourcentages sont de 16% en moyenne. Enfin, lorsque le nombre de DCs est supérieur à 6, les pourcentages sont de 100% pour le degré de difficulté égal à 0.5 et de 20% pour les degrés de difficulté égaux à 0.7 et 1. On remarque également que les coûts maxima atteints par la version BaseH12 sont les meilleurs comparés à ceux trouvés par toutes les autres versions.

Deuxième campagne d'expérimentation

Les résultats trouvés dans cette deuxième campagne pour la justification des heuristiques rejoignent ceux de la première campagne. Le tableau 7.12 montre que les coûts s'améliorent de 7% en moyenne en allant de la version Base à BaseH12. Par contre, nous observons une augmentation des temps dans le tableau 7.13 de 34% en moyenne.

# Ressources / Versions		2	4	6	8	10
Base	Coût Max	-832	-855	-2920	-1106	-648
	Fréquence	1	1	2	1	1
BaseH1	Coût Max	-832	-823	-3540	-1104	-729
	Fréquence	4	4	2	3	1
BaseH2	Coût Max	-1015	-1030	-2796	-1101	-891
	Fréquence	1	1	1	1	1
BaseH12	Coût Max	-828	-820	-2796	-1101	-640
	Fréquence	1	3	2	1	2

Tableau 7.14: *Tableau des meilleurs coûts avec un nombre de DCs égal à* 5 *et un degré de difficulté égal à* 0.7.

Les coûts maxima atteints par la version BaseH12 et représentés dans le tableau 7.14 sont meilleurs que ceux trouvés par toutes les autres versions. De même, les fréquences varient entre 1 et 4, ce qui montre, une fois de plus, que le nombre de ressources n'a pas d'influence sur la difficulté des problèmes traités. Le fait que le degré de difficulté soit égal à 0.7 nous donne des exemples moyennement difficiles.

7.5 Conclusion des résultats

Comme résumé et conclusion des expérimentations que nous avons effectuées, nous avons tracé deux courbes représentant les exemples test les plus difficiles, ayant un degré de difficulté égal à 1 et un nombre de DCs égal à 5.

- *apport des SMAs distribués*: La première courbe (voir figure 7.4) illustre l'évolution du temps en fonction du nombre de DCs. Elle montre le gain de temps en utilisant la version Base par rapport à RCMA. Ainsi, le caractère agréable et naturel des systèmes multi-agents, d'une part, et l'aspect distribué du mécanisme du recuit simulé, d'autre part, nous font gagner en temps de solution.

  Néanmoins, avec ces résultats les temps de traitement sont élevés relativement à la taille des exemples. La raison est que la fonction objectif que nous adoptons combime plusieurs critères, en plus des caractéristiques souhaités dans la solution. Ayant une telle fonction, le passage d'un état à un autre au cours du traitement de recherche devient très difficile et l'acceptation d'un

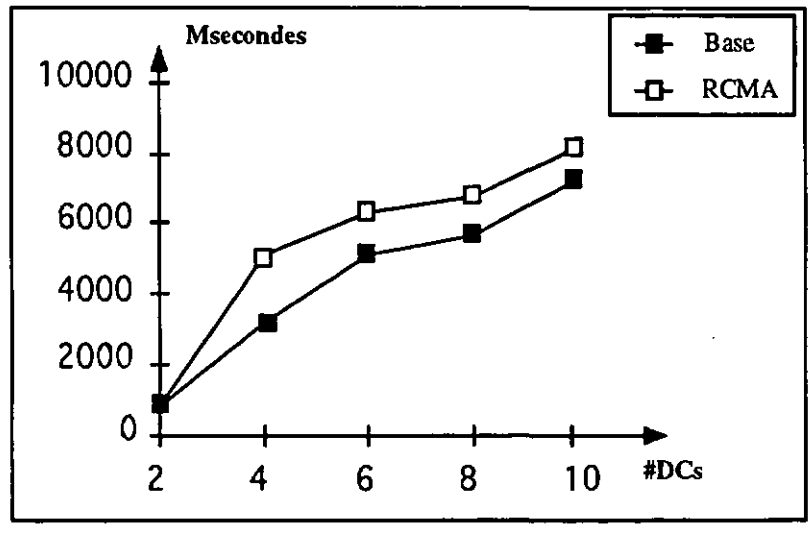


Figure 7.4: *Evolution du temps avec le nombre de DCs pour les versions RCMA et Base.*

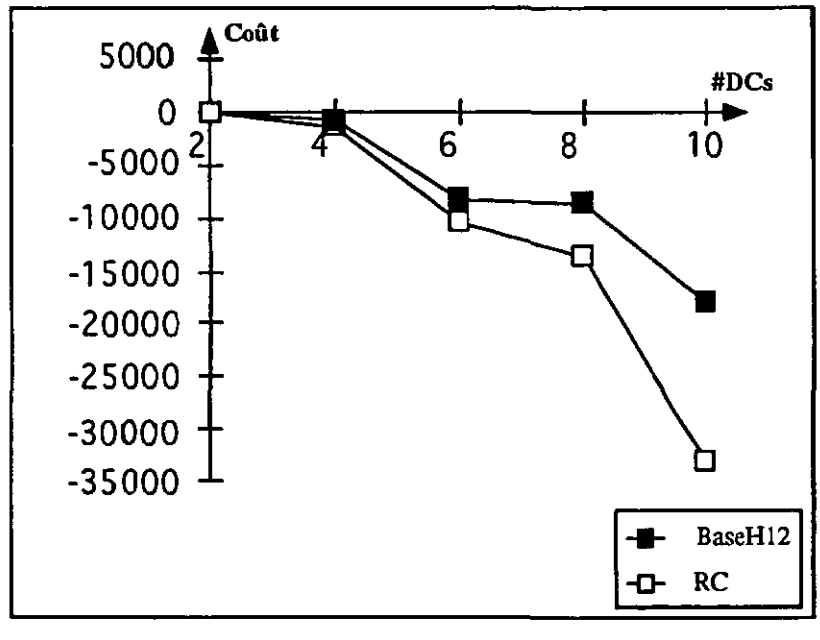


Figure 7.5: *Evolution du coût avec le nombre de DCs pour les versions RC et BaseH12.*

état nouveau très selectif. Par conséquent, le temps de génération de tous les états non acceptés est compté, ce qui donne des temps de traitement qui varient de 6000 à 8000 millisecondes pour un problème de 10 DCs.

Ce résultat est, par contre compensé par la qualité des solutions trouvées;

- *apport des deux heuristiques*: La deuxième courbe (voir figure 7.5) illustre l'évolution du coût en fonction du nombre de DCs. Elle montre le gain en qualité de solution en introduisant les deux heuristiques dans la version de base par rapport à la version RC. Ce gain s'accentue à mesure que la taille des exemples augmente. Ainsi, l'efficacité des deux heuristiques HMP et HA, d'une part, et le bon fonctionnement du schéma de refroidissement et du mécanisme de terminaison d'autre part, ont amélioré la qualité de la solution.

L'ensemble des résultats présentés dans ce chapitre nous a prouvé que nous pouvons utiliser la version Base qui représente un bon compromis en temps et en qualité, ou utiliser la dernière version BaseH12 au cas où l'intérêt porte plus sur la qualité de solution plutôt que sur le temps nécessaire à la trouver.

Pour la suite de notre travail, qui concerne surtout le prototype développé, nous avons opté pour l'utilisation de la version BaseH12.

Chapitre 8

Interactions, aspect dynamique

Jusqu'à présent, nous n'avons explicité que le côté statique de notre modèle de résolution du problème d'ordonnancement. Pourtant l'aspect dynamique du modèle a une importance très grande dans le prototype que nous avons développé. Nous souhaitons, en effet, avoir une simulation d'une production réelle pour le cas de l'ordonnancement en *flow shop* de permutation.

Il s'agit globalement de tenir compte des informations qui peuvent arriver au système et modifier, soit les données, soit les résultats obtenus. En l'occurrence, l'ajout d'une nouvelle DC, la date effective de début ou de fin d'une DC, l'enrayage d'une ressource, etc.

8.1 Pourquoi l'aspect dynamique?

Les problèmes d'ordonnancement peuvent être soumis à des modifications apparaissant sous la forme de perturbations du système et d'évolutions au cours du traitement. On appellera ces modifications des "événements".

Les origines de ces événements sont diverses. Nous citons en l'occurrence:

- les événements provoqués par l'utilisateur puisque le prototype développé sera suivi par un utilisateur qui doit pouvoir introduire, à tout moment, ses préférences sur les données ainsi que sur la solution;
- les événements liés à la production causés par les enrayages de certaines ressources de l'installation, le début et la fin effectifs de production ou l'arrivée de nouvelles DCs à ordonnancer.

Ces événements doivent être pris en considération au fur et à mesure qu'ils arrivent. Les réponses correspondantes doivent être fournies de manière à garder les mêmes objectifs que ceux que nous nous sommes fixés dans le modèle statique, tout en ayant un système efficace.

8.2 Avantages du modèle et motivations

Pour le type d'ordonnancement que nous traitons, résoudre les problèmes créés par la dynamicité avec notre modèle élaboré présente les avantages suivants:

- à chaque nouvel événement, le nouveau problème engendré peut être résolu en partant de la solution courante sans avoir à reprendre le travail depuis le début;
- l'aspect distribué du modèle des agents facilite les modifications suite à l'arrivée d'un événement. Chaque agent possède son propre comportement, donc les modifications se font sur une partie des données liées aux entités responsable, et ce en vue d'une solution liée à toutes les entités globalement.

En plus de ces avantages, des études ont été effectuées dans le même domaine:

- une étude des systèmes distribués pour les (CSP) dynamiques dont l'efficacité a été prouvée dans [Ghédira, 1994a];
- d'autres études pour les problèmes d'allocation qui confirment les mêmes idées ont été présentées dans [Ghédira et Verfaillie, 1992a].

Nous nous sommes basés sur les résultats trouvés ainsi que sur les avantages de notre modèle pour développer les modules nécessaires pour réagir aux événements considérés.

8.3 Présentation générale

Considérer l'aspect dynamique dans notre prototype revient à tenir compte des différents types d'interactions qui peuvent exister entre notre ordonnanceur MARSA et un simulateur d'assemblage SIMAS que nous présenterons plus tard dans le chapitre 11. Le but est d'avoir une simulation d'un traitement ordonnancement-simulation en temps réel.

La notion la plus importante dans ce contexte est celle du temps de réponse. C'est le temps qui sépare l'ordre de démarrage du traitement et la dernière opération de la sortie de la réponse. Il engendre, d'une part, le temps d'échange nécessaire pour l'entrée-sortie des données et, d'autre part, le temps que nécessite la machine pour traiter l'information.
Ainsi, suite à l'envoi d'un événement de la part de MARSA ou SIMAS, le traitement est d'autant plus efficace que les temps de réponse sont courts.

Dans ce chapitre nous explicitons les différents événements que nous avons créés et qui peuvent circuler entre MARSA et SIMAS, ainsi que les traitements associés

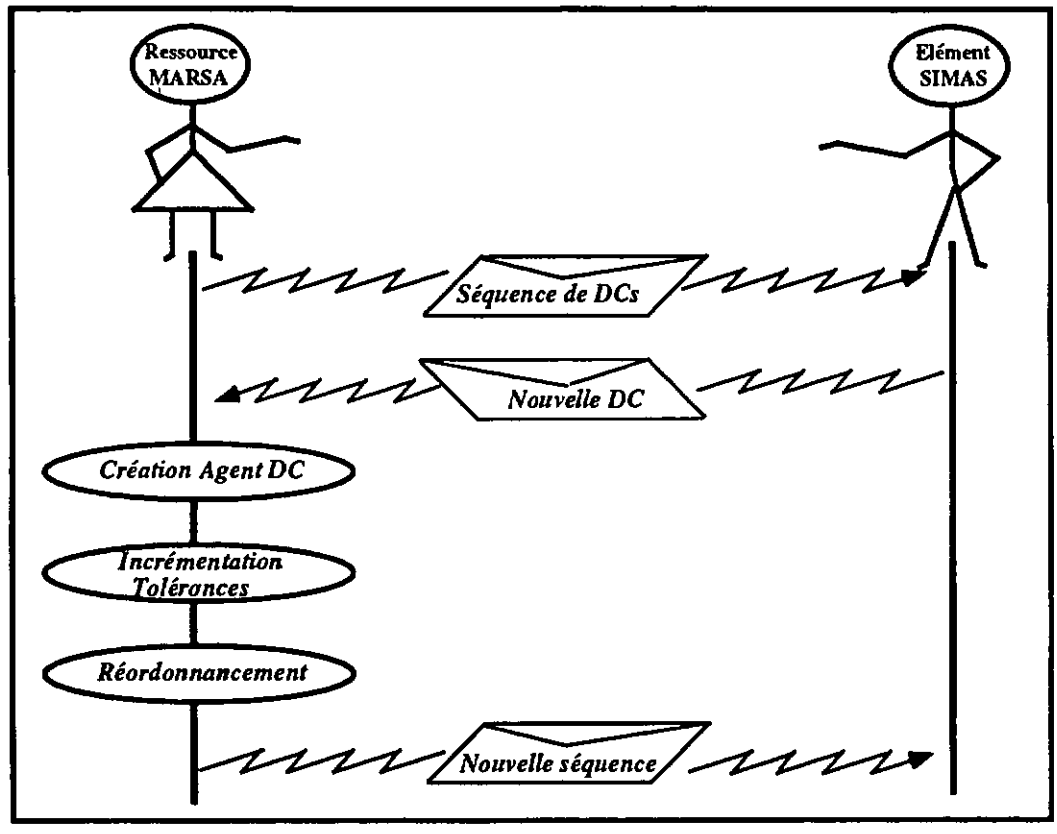


Figure 8.1: *Interactions MARSA-SIMAS suite à l'événement d'ajout d'une nouvelle DC.*

à chacun des événements. Nous signalons que l'envoi des événements MARSA est effectué par les ressources MARSA, et l'envoi des événements SIMAS est effectué par les éléments SIMAS, sachant que chaque ressource MARSA correspond à un élément SIMAS.

8.4 Ajout d'une nouvelle DC

Ce type d'événement est envoyé de SIMAS vers MARSA. Nous avons mentionné auparavant que les DCs n'entrent pas simultanément dans le système, mais au fur et à mesure qu'elles arrivent à l'atelier.

L'ordonnancement peut être déclenché dans MARSA avec un nombre déterminé de DCs, ensuite une ou plusieurs autres DCs nouvelles arrivent. A l'arrivée d'un événement de type "ajout d'une DC", MARSA réagit en déclenchant automatiquement un réordonnancement en tenant compte de toutes les DCs; anciennes

et nouvelles.

Dans le diagramme de la figure 8.1, lorsque MARSA présente une solution, sous la forme d'une séquence de DCs ordonnancées, et que SIMAS envoie une nouvelle DC, un agent DC correspondant se crée et la tolérance des agents ressources est incrémentée. Par la suite, la nouvel agent DC va solliciter les agents Ressource en demandant de placer ses actions et le déclenchement du réordonnancement a lieu.

Pour garder la cohérence avec le calcul des tolérances initiales des agents Ressource, leurs tolérances sont incrémentées de la valeur :

$$\frac{Mr_j}{ln2}$$

où Mr_j est la marge de la nouvelle demande client DC_j.

L'avantage que présente notre modèle est que le système est démarré à partir de la solution courante, sachant que la liste des agents DCs en attente comprend l'agent de la nouvelle DC. Par conséquent, le système n'a pas besoin de recommencer tout l'ordonnancement depuis le début.

8.5 Envoi de DC pour exécution

Ce type d'événement est envoyé de MARSA vers SIMAS. Les DCs sont envoyées pour exécution vers SIMAS, sous forme d'ordres de fabrication, après leur affectation à des ressources. Elles sont envoyées à une fréquence permettant un travail continu du simulateur d'assemblage. De même, elles sont envoyées selon leurs positions dans la séquence.

Une DC se trouvant en tête de la séquence est envoyée sous la forme de bons de travail, où chaque bon de travail correspond à une de ses actions. Autrement dit, chaque bon de travail est envoyé pour exécution vers l'élément SIMAS qui correspond à la ressource sur laquelle l'action est affectée. La réaction de SIMAS à ce genre d'événement est de mettre le bon de travail recu dans la liste des bons à exécuter de l'élément SIMAS.

8.6 Début et fin d'un bon de travail

Ce type d'événement est envoyé de SIMAS vers MARSA.

- lorsqu'un élément SIMAS commence l'exécution d'un bon de travail, la date de début de ce bon, qui peut être différente de la date donnée par l'ordonnancement, devient connue. A ce moment, l'élément SIMAS envoie un événement vers la ressource MARSA correspondante pour l'informer de la date effective du bon;

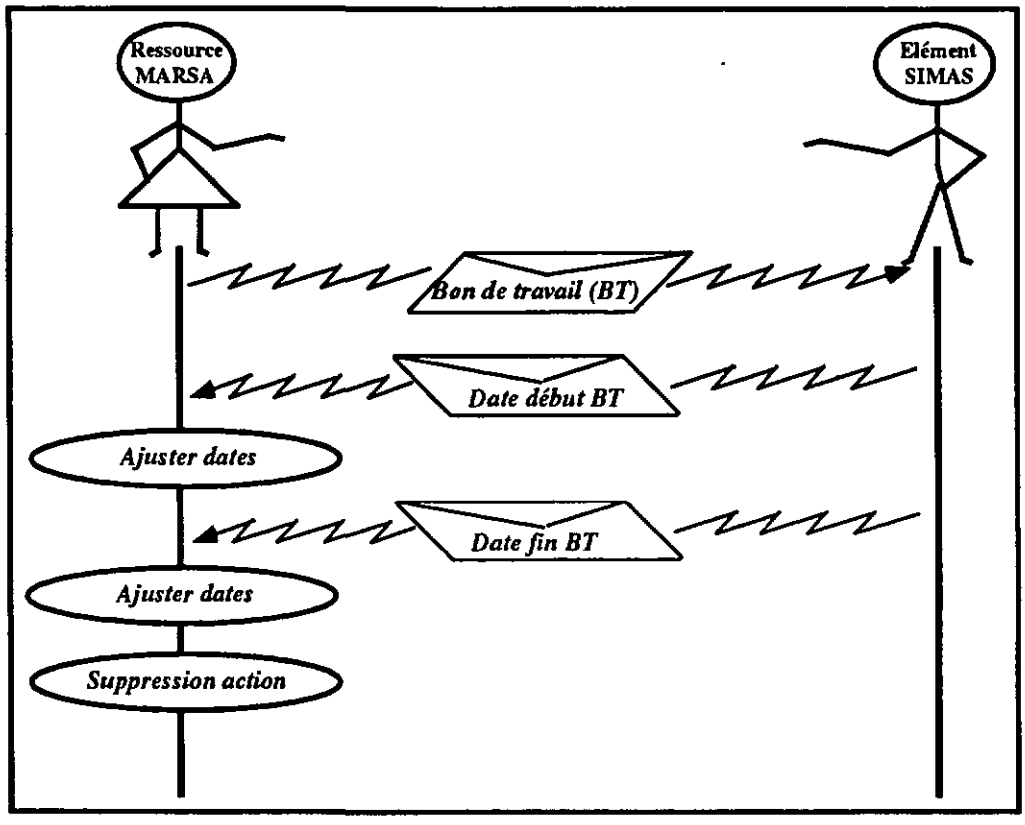


Figure 8.2: *Interactions MARSA-SIMAS suite à l'événement d'envoi d'un bon de travail à l'exécution.*

- lorsqu'un élément SIMAS termine l'exécution d'un bon de travail, la date de fin de ce bon devient connue. A ce moment, l'élément SIMAS envoie un événement vers la ressource MARSA correspondante pour l'informer de la date de fin effective du bon.

La réaction de la ressource MARSA vis à vis de l'un des deux événements est l'opération de correction et d'ajustement des dates de son ordonnancement courant. Cet ajustement se présente sous la forme de décalages de certaines actions dans le temps sur la ressource, en modifiant leurs dates de début et de fin respectives. Il peut impliquer aussi le décalage de certaines autres actions sur les ressources suivantes.

De plus, lorsque la ressource MARSA reçoit un événement de date de fin d'un bon de travail, l'action correspondante à ce bon est éliminée de l'ordonnancement. Par conséquent, avec la réception de la date de fin du dernier bon de travail d'une DC, toute la DC est éliminée de l'ordonnancement, donc exécutée en entier.

Dans le diagramme de la figure 8.2, nous représentons les interactions suite à l'envoi d'un bon de travail pour l'exécution.

Pour assurer la continuité de la simulation nous avons programmé l'envoi des bons de travail, correspondants aux DCs, vers le simulateur de la manière suivante: A la réception de l'événement de début du dernier bon de travail d'une DC, l'envoi des bons de travail correspondants à la DC suivante se fait automatiquement.

8.7 Début et fin d'un enrayage ou d'un réglage

Ce type d'événement est envoyé de SIMAS vers MARSA.

- lorsqu'un élément SIMAS est enrayé ou nécessite un certain réglage à un instant donné, il envoie un événement à la ressource MARSA correspondante pour l'informer de la date de début de l'enrayage ou du réglage.

  La réaction de la ressource à ce type d'événement se limite à noter cette date de début;

- lorsque l'enrayage ou le réglage du même élément SIMAS est terminé, SIMAS envoie à nouveau un événement à la ressource MARSA correspondante pour l'informer de la date de fin de l'enrayage ou du réglage.

  La réaction de la ressource à ce type d'événement est de:

  1. Déterminer, par rapport à la date de début de l'enrayage ou du réglage déjà notée, quelle action est en cours d'exécution.
  2. Calculer la différence en temps entre la date de fin donnée de l'enrayage ou réglage avec la date de début.

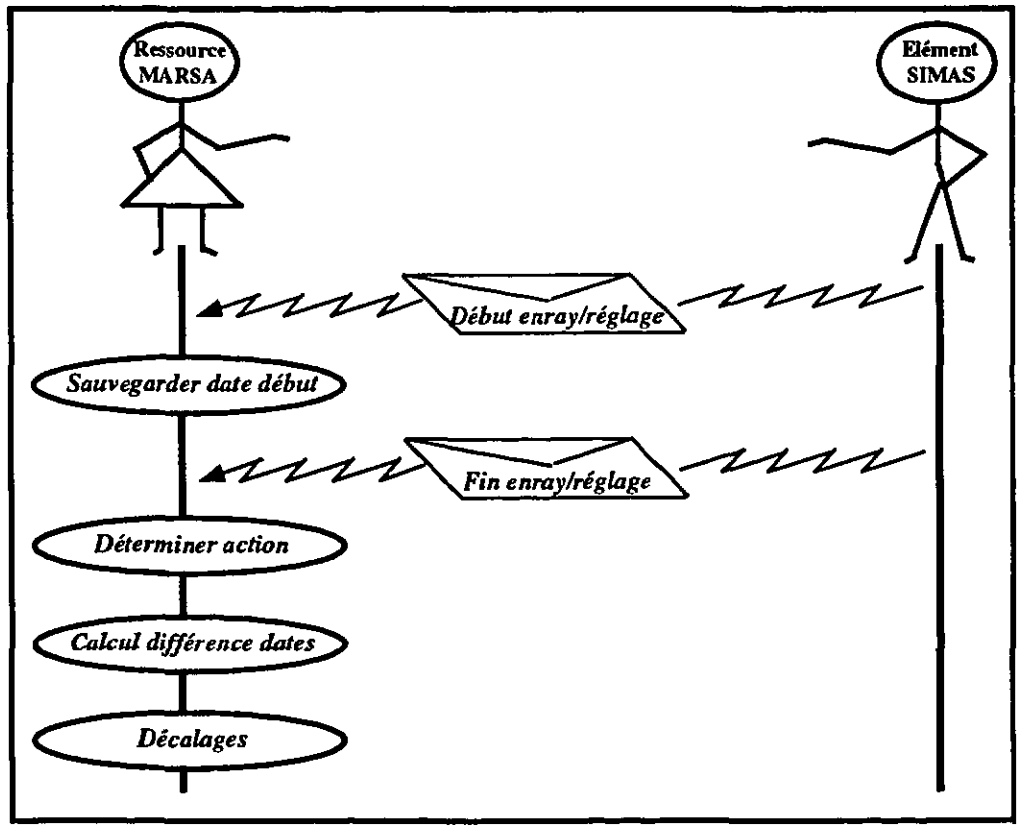


Figure 8.3: *Interactions MARSA-SIMAS suite à l'événement d'envoi des date de début et de fin de l'enrayage ou du réglage.*

3. Ajouter cette différence à la date de fin de l'action en question.
4. Propager les modifications des dates aux autres actions qui sont sur la même ressource ou sur les ressources suivantes.

En effet, l'enrayage ou le réglage d'une ressource ne fait que ralentir le traitement de l'action en exécution.

Dans le diagramme de la figure 8.3 nous représentons les interactions suite à l'envoi des dates de début ou de fin de l'enrayage ou du réglage.

Chapitre 9

Etude d'un exemple

Nous illustrons dans ce chapitre un exemple industriel réel d'assemblage que nous appliquons sur notre prototype.

En revanche nous ne nous intéressons pas, dans ce qui suit, à faire des tests aussi poussés comme nous l'avons déjà fait dans le chapitre expérimentations 7. Nous avons déjà fait notre choix en adoptant un modèle sur lequel nous avons fait des tests pour vérifier ses performances en qualité et en temps de solution. Pour le moment, et en appliquant l'exemple industriel au prototype, nous portons notre attention sur l'aspect dynamique de notre travail.

Ce qui est important, en effet, ce sont les temps de réponse de MARSA suite à l'arrivée des événements de la part de SIMAS.

L'exécution des bons de travail dans le simulateur doit être continue. Par conséquent, des DCs ordonnancées dans le temps doivent être disponibles continuellement.

9.1 Présentation de l'exemple

Nous traitons un exemple de Télémécanique fournit par l'Institut de Microtechnique et tiré de l'installation d'essai utilisée pendant le déroulement d'un projet Esprit[1]. La figure 9.1 illustre l'installation telle qu'elle est éditée par SIMAS.

Cette installation représente les solutions aux difficultés techniques que les partenaires désirent résoudre. En plus, elle représente une installation typique dans l'industrie de l'assemblage automatisé.

Les solutions techniques retenues sont telles que:

[1]Projet SCOPES # 6562.

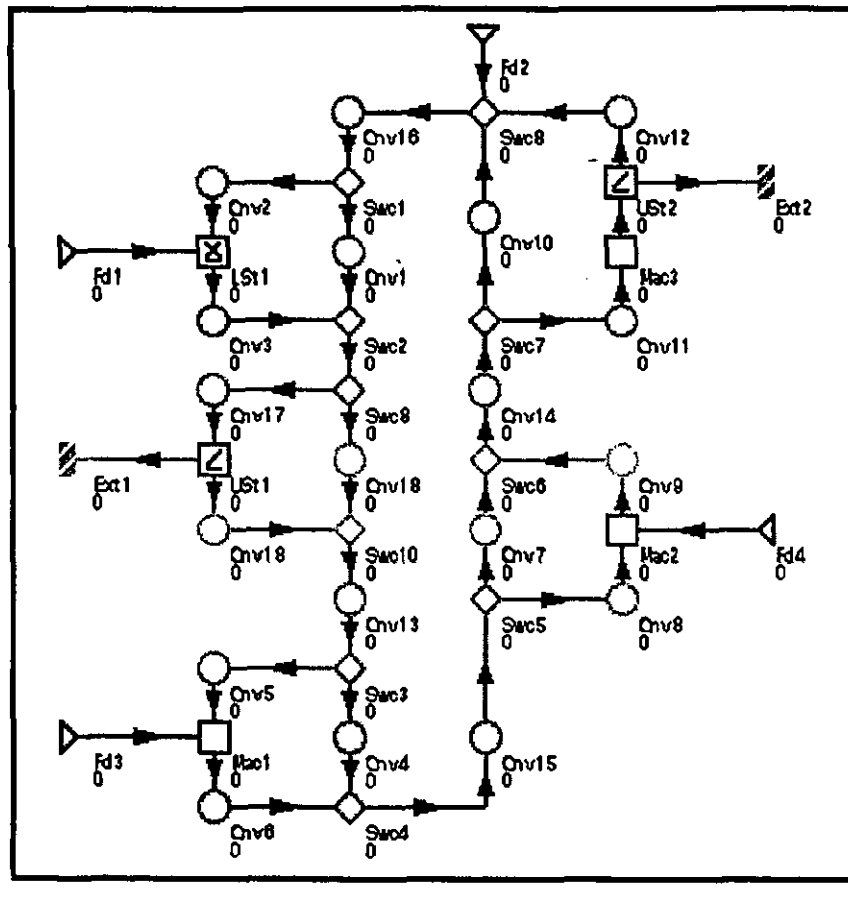


Figure 9.1: *Edition SIMAS de l'exemple industriel de Télémécanique.*

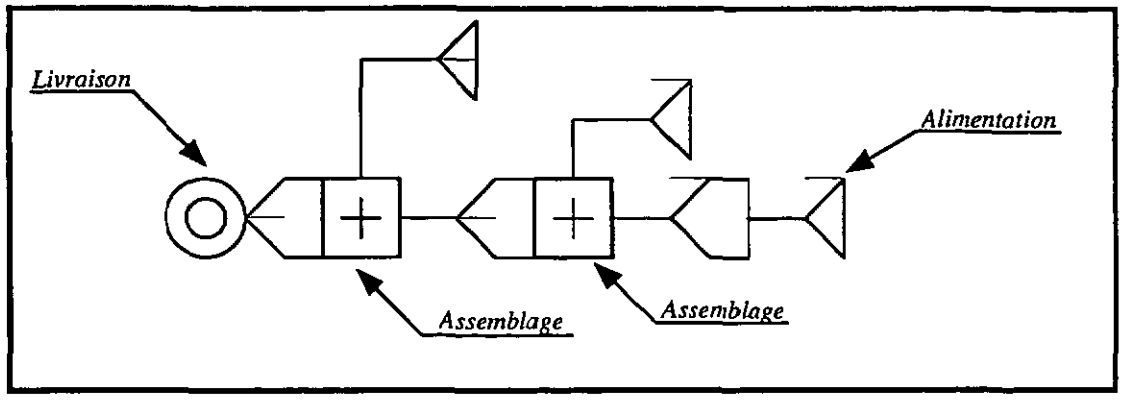


Figure 9.2: *Edition SIMAS de l'exemple industriel de Télémécanique.*

- les différents postes (éléments SIMAS) sont placés en dérivation et non pas sur le circuit principal (LSt1, USt1, USt2, Mac1, Mac2 et Mac3), ce qui d'une part augmente considérablement le coût de l'installation et d'autre part la complexité de pilotage;
- les postes d'assemblage sont également prévus pour la robotisation, ce qui les rend très chers par rapport aux installations classiques;
- Les temps de cycle sont assez typiques des installations munies de robots assurant la fonction de posage des composants.

9.2 Ressources MARSA correspondantes

Du côté de MARSA, nous n'avons besoin que des éléments qui sont susceptibles d'envoyer des événements ou d'en recevoir. Par conséquent, SIMAS nous indique les éléments correspondants.

Dans notre exemple, les éléments SIMAS qui peuvent envoyer des événements ou en recevoir sont ceux nommés dans la figure 9.1 par LSt1, USt1, Mac1 et Mac2. Nous avons, ainsi quatre ressources MARSA correspondantes à ces éléments.

Un processus, déterminé lors de la lecture du fichier processus, est lié à l'installation. Ce processus est représenté dans la figure 9.2 et est composé essentiellement de quatre actions:

- une action alimentation;
- deux actions assemblage;

- une action livraison.

Chacune de ces actions s'exécutera sur l'une des ressources en respectant les relations de précédences qui existent entre elles.

9.3 Génération d'exemples et tests effectués

Etant donné que le nombre de ressources est égal à quatre, nous avons pris comme paramètre de génération le nombre de DCs.

Pour effectuer les tests, nous avons attribué au nombre de DCs les valeurs égales à {2, 3, 4, 5, 6, 7, 8, 9,} et, pour chacune des valeurs, nous avons généré cinq exemples différents et calculé les moyennes des temps de traitement trouvés par les tests. Pour chacun des exemples:

1. Nous avons démarré l'ordonnancement et ensuite nous avons essayé d'améliorer la solution trouvée.

2. Nous avons provoqué l'arrivée d'une nouvelle DC, de telle manière qu'un réordonnancement se déclenche automatiquement avec un nombre de DCs supérieur au précédent d'une unité. Le réordonnancement implique l'affectation de la nouvelle DC aux ressources en l'insérant dans la séquence de DCs déjà affectées.

3. Nous avons déclenché à nouveau un ordonnancement à zéro de ces mêmes DCs, en prenant comme état initial l'état vide où aucune DC n'est affectée.

4. Nous avons provoqué les événements d'envoi de DCs, de début et fin de bon de travail et de début et fin d'enrayage ou réglage.

A chacune de ces étapes nous avons calculé le temps nécessaire pour l'exécuter. Nous signalons que ces temps sont exprimés en millisecondes et tiennent compte du traitement graphique utilisé dans notre prototype.

9.4 Résultats obtenus

Le tableau 9.1 des temps moyens de traitement illustre les résultats des différents tests.
D'une manière globale, nous remarquons que les temps augmentent proportionnellement au nombre de DCs, sauf dans le cas de l'envoi de DC ou le début du bon de travail. Nous allons maintenant examiner les résultats par ordre d'exécution dans le prototype.

Msec / #DCs	1er Ordo.	1ere Amélior.	2ème Ordo.	Ordo. à zéro	Envoi DC	Début Bon T.	Fin Bon T.	Début Enrayage	Fin Enrayage
2	5614	5219	3793	8819	4497	4365	4768	938	4096
3	9102	8944	7171	14622	4485	7350	7283	927	7273
4	14353	12881	13498	22506	4657	11445	11362	901	11449
5	22582	17768	15904	32901	4352	15452	15885	911	15311
6	33362	24088	20500	48723	4364	20805	20429	909	20543
7	45715	31077	25835	63367	4640	24640	25028	893	24711
8	62255	39047	30912	82541	4831	31335	30993	903	31115
9	79207	48930	33626	108243	5140	36361	36830	897	36606

Tableau 9.1: *Tableau des temps moyens de traitement d'ordonnancement et des traitement provoqués par les événements en millisecondes.*

%aug. de temps / #DCs	2ème Ordo. - Ordo. à zéro
2	132
3	103
4	62
5	106
6	137
7	145
8	167
9	221

Tableau 9.2: *Tableau des pourcentages d'augmentation des temps de l'ordonnancement à zéro par rapport au deuxième ordonnancement.*

%aug. de temps / #DCs	1er Ordo.	1ere Amélior.	2ème Ordo.	Ordo. à zéro	Début Bon T.	Fin Bon T.	Fin Enrayage
2-3	62	71	89	65	68	52	77
3-4	57	44	88	53	55	56	57
4-5	57	37	17	46	35	39	33
5-6	47	35	28	48	34	28	34
6-7	37	29	26	30	18	22	20
7-8	36	25	19	30	27	23	25
8-9	27	25	8	31	15	18	17

Tableau 9.3: *Tableau des pourcentages d'écarts entre les temps des exemples ayant des nombres de DCs successifs.*

9.4.1 Ordonnancement et amélioration

En comparant le temps mis par un ordonnancement (colonne 1) et celui par son amélioration (colonne 2), nous remarquons que l'amélioration met moins de temps. Dans les deux cas, nous remarquons également que le pourcentage d'augmentation de temps entre deux valeurs successives diminue avec l'augmentation du nombre de DCs (voir tableau 9.3). Autrement dit, l'amélioration met proportionnellement moins de temps pour les exemples ayant un nombre de DCs plus élevé.

9.4.2 Arrivée d'une nouvelle DC

Le réordonnancement déclenché par l'arrivée d'une DC nouvelle (colonne 3) met moins de temps de traitement que l'ordonnancement précédent (colonne 1). Nous remarquons qu'en augmentant le nombre de DCs, le pourcentage d'augmentation de temps entre deux valeurs successives diminue également. Par conséquent, la performance de l'ajout dynamique d'une DC est meilleure proportionnellement au nombre de DCs.

De même, nous remarquons que, lorsque le nombre de DCs est égal au nombre de ressources (nombre de DCs = 4), nous sommes en présence d'un exemple difficile et les temps mis pour le premier et le deuxième ordonnancement sont proches.

9.4.3 Ordonnancement à zéro

Nous avons voulu comparer l'ordonnancement suite à l'arrivée dynamique d'une DC nouvelle, avec l'ordonnancement des même DCs en partant d'un état vide (où aucune DC n'est placée).

Nous remarquons que l'ordonnancement à zéro met nettement plus de temps pour tous les exemples.

Dans le tableau 9.2, nous avons représenté les pourcentages d'augmentation du temps de l'ordonnancement à zéro par rapport à l'ordonnancement dynamique. Nous pouvons remarquer que l'augmentation devient plus importante à mesure que le nombre de DCs augmente. Par conséquent, l'avantage de l'ordonnancement dynamique s'accentue avec l'augmentation de la taille des exemples. Par contre, pour les exemples difficiles, où le nombre de DCs est égal au nombre de ressources, l'augmentation est moins importante.

9.4.4 Envoi des bons de travail d'une DC

Le temps d'envoi des bons de travail correspondants à une DC (colonne 5) dépend essentiellement du nombre de ressources, puisque chacune de ses ressource va envoyer à l'élément SIMAS correspondant le bon de travail qu'il doit exécuter. En effet, le temps d'envoi calculé est pratiquement le même, indépendamment du nombre de DCs. Il est en moyenne de 4620 millisecondes.

9.4.5 Début et fin d'un bon de travail

Les temps de début et de fin d'un bon de travail sont très proches (colonnes 6 et 7), car le traitement correspondant à ces événements est le même dans les deux cas: il consiste à ajuster la date de début ou de fin de l'action qui correspond au bon de travail et à effectuer les propagations nécessaires sur le reste des actions.

9.4.6 Début et fin d'enrayage ou réglage

Les temps de début d'enrayage ou réglage sont pratiquement les mêmes pour tous les exemples (colonnes 8 et 9). Il est en moyenne de 909 millisecondes. En effet, le traitement impliqué consiste à mémoriser la date. Il est donc indépendant des données de l'exemple.

En revanche, les temps de fin d'enrayage ou réglage augmentent proportionnellement au nombre de DCs. En effet, le traitement impliqué consiste à calculer la différence des temps de début et fin et à ajuster les dates des actions et faire les propagation nécessaires.

Dans le tableau 9.3, nous avons calculé les pourcentages d'augmentation des temps pour les exemples ayant des nombres de DCs successifs. Nous remarquons que les augmentations sont de moins en mois importantes proportionnellement au nombre de DCs. Par conséquent, le modèle est avantageux pour les exemples de grandes tailles.

Partie III

Prototype, application à l'assemblage

Introduction

Dans le développement d'un logiciel, nous sommes généralement confrontés au problème de la maintenance, puisque le monde réel représente un univers extrêmement fluctuant. Par conséquent, le logiciel doit être constamment adapté.

Le problème majeur est la complexité des systèmes. On a toujours cru que la réalisation d'un système d'information consiste à écrire des programmes. Or, la programmation ne constitue que la phase ultime d'un long processus. Au préalable il faut créer un modèle conceptuel de la réalité, qui est souvent mal structurée, ensuite il est nécessaire de structurer les traitements ainsi que les données et les insérer dans l'organisation existante. C'est seulement lorsque l'on a réalisé une architecture cohérente du système qu'intervient la programmation proprement dite.

Le prototype que nous avons développé dans le cadre du projet[2], représente le côté applicatif de MARSA, pour *Multi-Agent Reactive system of Scheduling Assembly.*
La réalisation de ce prototype a nécessité l'utilisation de techniques de différents domaines du côté modélisation et conception, à savoir les systèmes multi-agents et le recuit simulé, et du côté programmation, à savoir le langage orienté objet CLOS, pour *Common Lisp Oriented System.*

Le projet est sous forme de deux modules essentiels:

- MARSA qui est notre système d'aide à l'ordonnancement d'ateliers de type flow shop de permutation;
- SIMAS qui est un système événementiel capable de fournir des simulations et des pilotages des systèmes d'assemblages.

La figure 9.3 présente le cadre général du projet: SIMAS fournit les DCs à traiter accompagnées de leurs consignes et leurs dates de livraisons, ainsi que le modèle physique de l'atelier à savoir les ressources, les accumulations, etc. MARSA, notre module d'ordonnancement, fournit la meilleure séquence possible d'ordres de fabrication.

Les interactions entre MARSA et SIMAS sont bidirectionnelles:
Les DCs sont envoyées de SIMAS vers MARSA une à une. De même, MARSA renvoi vers SIMAS ces DCs, une fois ordonnancées dans le temps, afin d'être exécutées par ce simulateur. Les dates fixées par l'ordonnancement sont ajustées au

[2]Projet financé par le Fonds National Suisse des Recherches Scientifiques # SPP-IF 5003-034319

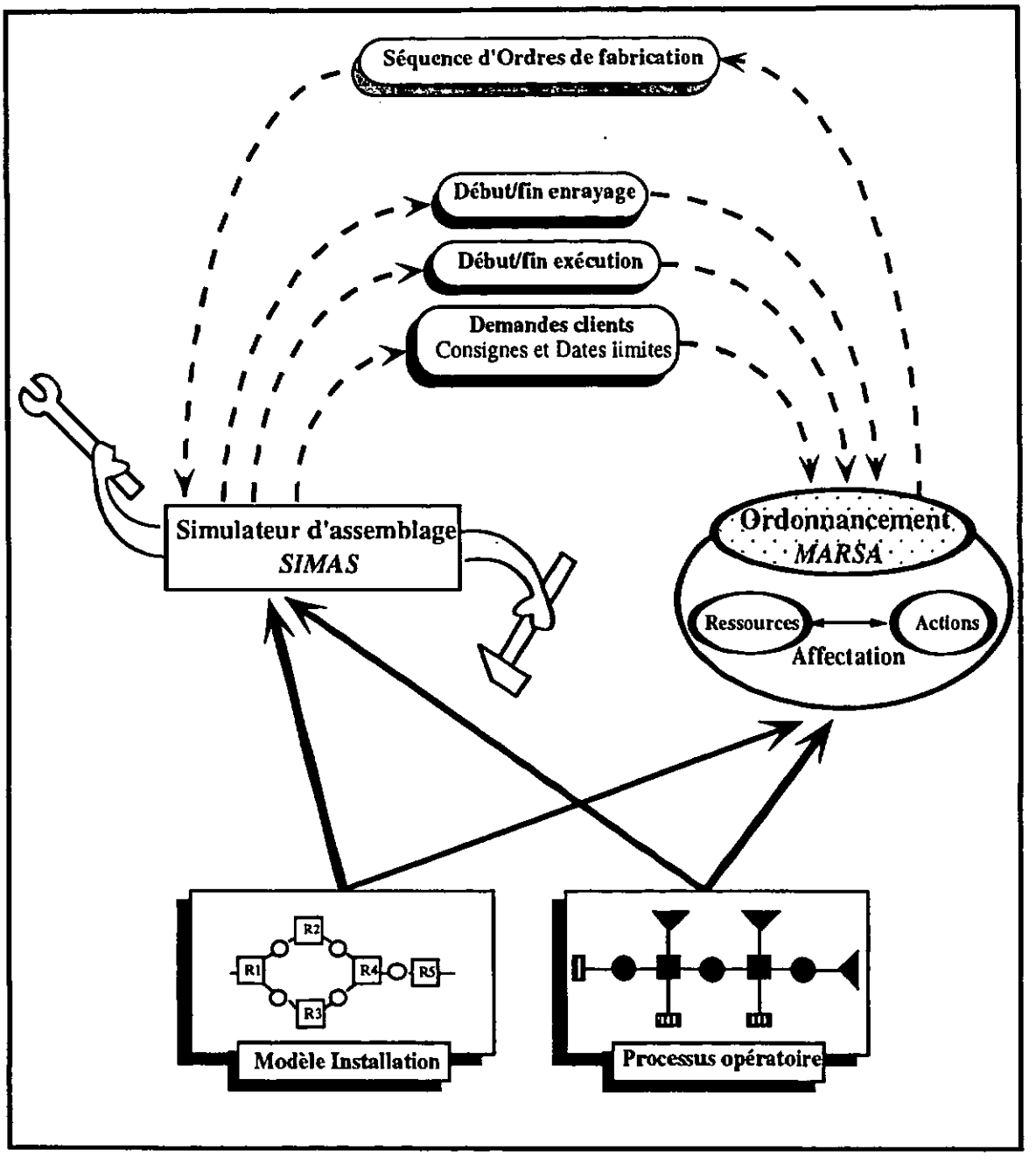


Figure 9.3: *Cadre général du projet.*

fur et à mesure que la simulation se déroule et qu'on obtient les dates effectives de l'exécution. Par ailleurs, des perturbations peuvent avoir lieu dans l'atelier à travers le simulateur, telles que les enrayages des ressources. Nous tenons compte de ces perturbations en modifiant l'ordonnancement.

Ce projet à des objectifs sur différents plans:

- un objectif lié à l'**intelligence artificielle** est la validation et l'extension d'une approche distribuée qui combine les systèmes multi-agents avec le recuit simulé, vu l'importance des systèmes distribués dans la recherche actuelle en intelligence artificielle;
- un objectif lié à la **simulation** est d'avoir une intégration efficace et continue d'un système d'aide d'ordonnancement d'ateliers de type flow shop de permutation avec un simulateur d'assemblage de ces ateliers;
- un objectif lié à la **manipulation** est de fournir une interface utilisateur. Elle doit permettre à l'utilisateur d'introduire ses préférences à tout moment, pour modifier soit les données du problème soit les solutions trouvées;
- un objectif lié à l'**optimisation** est d'optimiser le temps de réponse de l'ordonnancement, le temps de production et d'assurer que la production soit aussi continue que possible.

Ce projet a fait l'objet d'une collaboration entre l'Institut d'Informatique et Intelligence Artificielle de l'Université de Neuchâtel et l'Institut de Microtechnique de l'Ecole Polytechnique de Lausanne, afin de développer un prototype qui réponde à tous les objectifs que nous venons de citer.

Cette partie contient deux chapitres. Dans le chapitre 10, nous présentons la manière adoptée pour représenter le problème ainsi que pour le résoudre et dans le chapitre 11, nous parlons du simulateur SIMAS.

Chapitre 10

Représentation et résolution

Une bonne représentation du problème peut permettre de réduire la complexité de sa résolution. Par ailleurs, avant toute résolution d'un problème, le choix de la méthode d'analyse, du support machine et du langage informatique nous semblent être trois éléments importants pour l'élaboration d'un prototype. L'origine de l'importance réside dans le fait que ces éléments peuvent avoir un effet positif ou négatif sur les résultats souhaités, à savoir l'accès aux informations, la vitesse du traitement, etc. Nous présentons ci-après les choix que nous avons faits pour ces trois éléments. Ensuite, nous passons à la représentation du problème et à sa résolution.

10.1 Choix de la méthode d'analyse

Notre prototype est conçu suivant l'architecture distribuée, constituée d'agents participant de façon concurrentielle ou coopérative à la recherche de solutions. La nature et le comportement de ces agents rendent leur représentation orientée objet facile et immédiate.

La technologie orientée objet est plus qu'une manière de programmer. C'est, en fait, une manière de réflexion abstraite à propos d'un problème du monde réel. Elle fournit une méthode pratique et productive de développement de logiciels pour des applications. Dans cette méthode, la structure fondamentale utilisée est l'objet, qui incorpore les structures de données et le comportement en une seule entité. Ce concept "d'encapsulation" distingue la programmation orientée objet de la programmation procédurale, où les données sont séparées des instructions. Les objets communiquent entre eux par des messages, et ceux qui ont une même structure de données et un même comportement sont groupés en une classe.
Les modèles orientés objet sont utiles pour la compréhension des problèmes, la

communication avec des experts d'applications, la modélisation des entreprises, la préparation de documentations, et la conception de programmes et de bases de données.
Nous citons ces trois étapes de l'analyse orientée objet [Rumbaugh et al, 1991]:

- élaboration d'un modèle d'analyse qui comprend les aspects essentiels extraits du domaine d'application. Ce modèle est formé d'objets ayant une description de leurs propriétés et de leurs comportements;
- conception des décisions et ajout des détails pour décrire et optimiser la programmation;
- programmation.

10.2 Choix du support machine

Nous n'avons pas fait une étude exhaustive de toutes les machines sur lesquelles nous pourrions élaborer notre programmation, le choix a été fortement influencé par celui de notre laboratoire. La partie programmation a été faite, en partie sur un Macintosh LCIII, et en partie sur un Macintosh Performa 630. Ce dernier représente plus de rapidité de traitement.

10.3 Choix du langage de programmation

Le langage de programmation choisi est CLOS sur Macintosh car il présente plusieurs avantages:

- il fournit un environnement flexible pour le développement de logiciels et d'applications;
- étant l'extension orientée objet du langage Lisp, il est l'idéal pour résoudre les problèmes en intelligence artificielle;
- il compile le code aussi rapidement que le langage C, et ses caractéristiques orientées objet sont mieux intégrées et permettent des créations dynamiques et des destructions d'objets plus naturellement que C ou Pascal;
- il fournit un environnement de programmation interactif qui fait gagner du temps et améliore la correction des erreurs de code. Nous pouvons compiler, tester, et corriger les erreurs de fonctions individuellement, sans avoir à compiler le programme en entier;

- il fournit une gamme complète d'outils et d'aides de programmation. En plus, ces outils incluent un éditeur d'inspection, une trace de l'exécution, un éditeur rapide orienté Lisp, et d'autres outils qui offrent un accès rapide aux définitions des fonctions et des variables;
- enfin, il nous permet d'accéder aux codes, écrits dans d'autres langages, à l'aide d'une interface de fonctions exportées.

10.4 Représentation des données du problème

Pour la clarté de la représentation du problème et, par la suite, de la programmation, nous avons séparé les données concernant MARSA de celles concernant SIMAS. Dans la suite, chaque ressource MARSA est liée à un élément SIMAS correspondant, chaque DC MARSA est liée à une DC SIMAS correspondante et chaque action MARSA est liée à une action SIMAS correspondante.

De même, nous avons établi une structure des agents séparée de toutes les autres structures. Ainsi, pour représenter le problème que nous traitons, nous avons adopté l'analyse orientée objet et nous avons établi les six modèles de structures

- de l'installation;
- de l'élément de l'installation;
- de la DC;
- de l'action;
- de la solution;
- des agents.

Pour schématiser ces six modèles, nous représentons les classes de données par des rectangles droits, les champs fixes liés à ces classes par des rectangles inclinés et les champs dynamiques par des rectangles arrondis, sachant qu'un champs dynamique change plusieurs fois de valeur au cours du traitement. Les flèches foncées indiquent les relations d'héritage "est un" alors que les flèches discontinues indiquent les relations de possession "a".

10.4.1 Modèle de l'installation

Le modèle de l'installation d'assemblage est un graphe orienté de lieux auxquels sont rattachées des fonctions que les lieux sont à même de réaliser.
Du côté MARSA, l'installation représentée dans le schéma de la figure 10.1, est liée à:

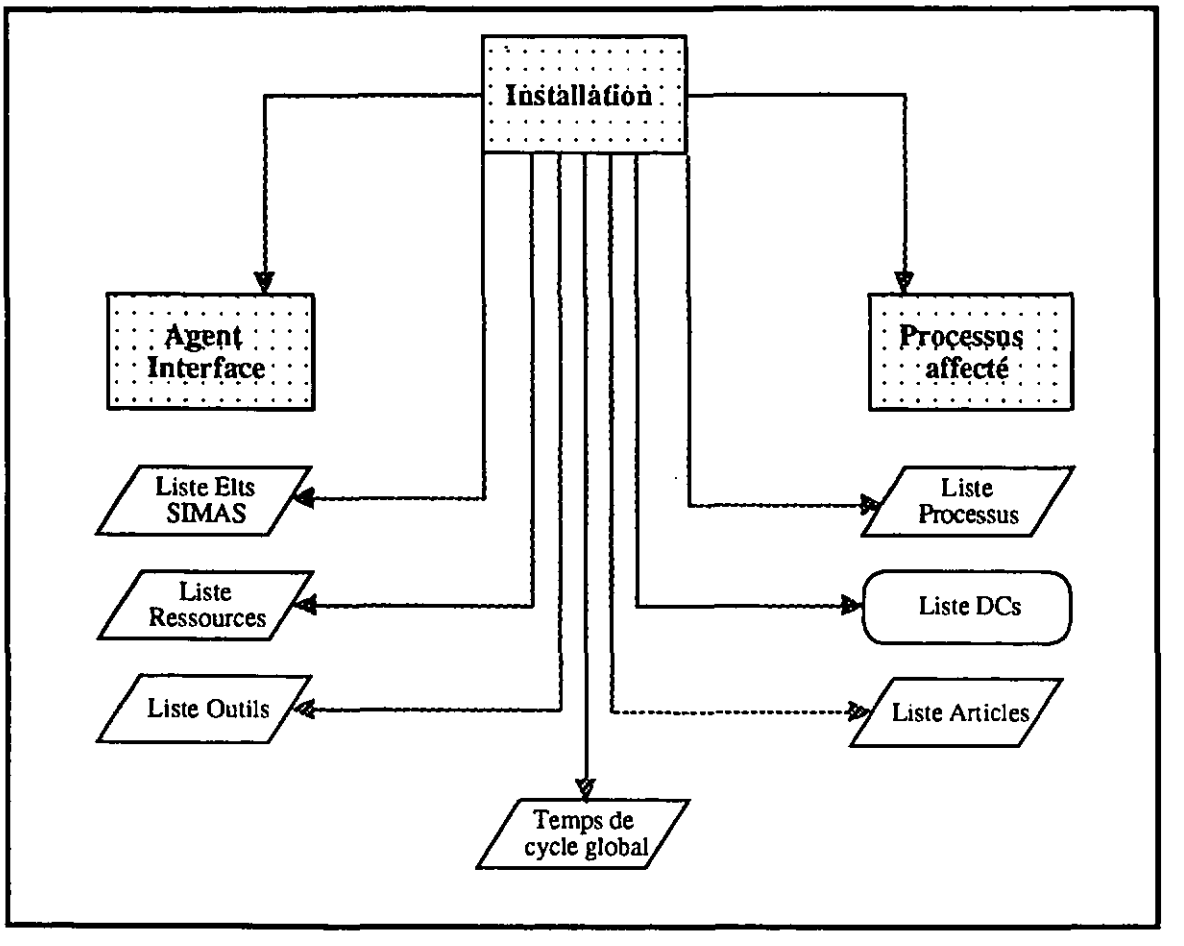


Figure 10.1: *Schéma du modèle de l'installation.*

- un agent Interface, qui est l'intermédiaire entre le système et l'utilisateur;
- une liste des éléments SIMAS. A chaque élément nous ferons correspondre une ressource;
- une liste des ressources correspondantes aux éléments SIMAS;
- une liste d'outils utilisés par les ressources dans l'installation;
- une liste des articles à produire;
- une liste de DCs à ordonnancer à un instant donné;
- une liste des processus possibles qui peuvent être utilisés pour le traitement;
- un processus affecté, à un instant donné;
- un temps de cycle global de toute l'installation.

10.4.2 Modèle de l'élément de l'installation

Dans le schéma de la figure 10.2 un élément SIMAS est caractérisé par

- un identificateur;
- une liste des liens avec les éléments qui le précèdent;
- une liste des liens avec les éléments qui le suivent.

Il peut être

- soit une ressource;
- soit une accumulation;
- soit un îlot.

Chaque lien ressource est représenté par une accumulation qui sépare deux ressources. Une ressource est caractérisée par trois informations nécessaires pour le calcul de son temps de cycle estimé, qui sont:

- un temps de cycle technique;
- un temps moyen de réparation;
- une probabilité d'enrayage.

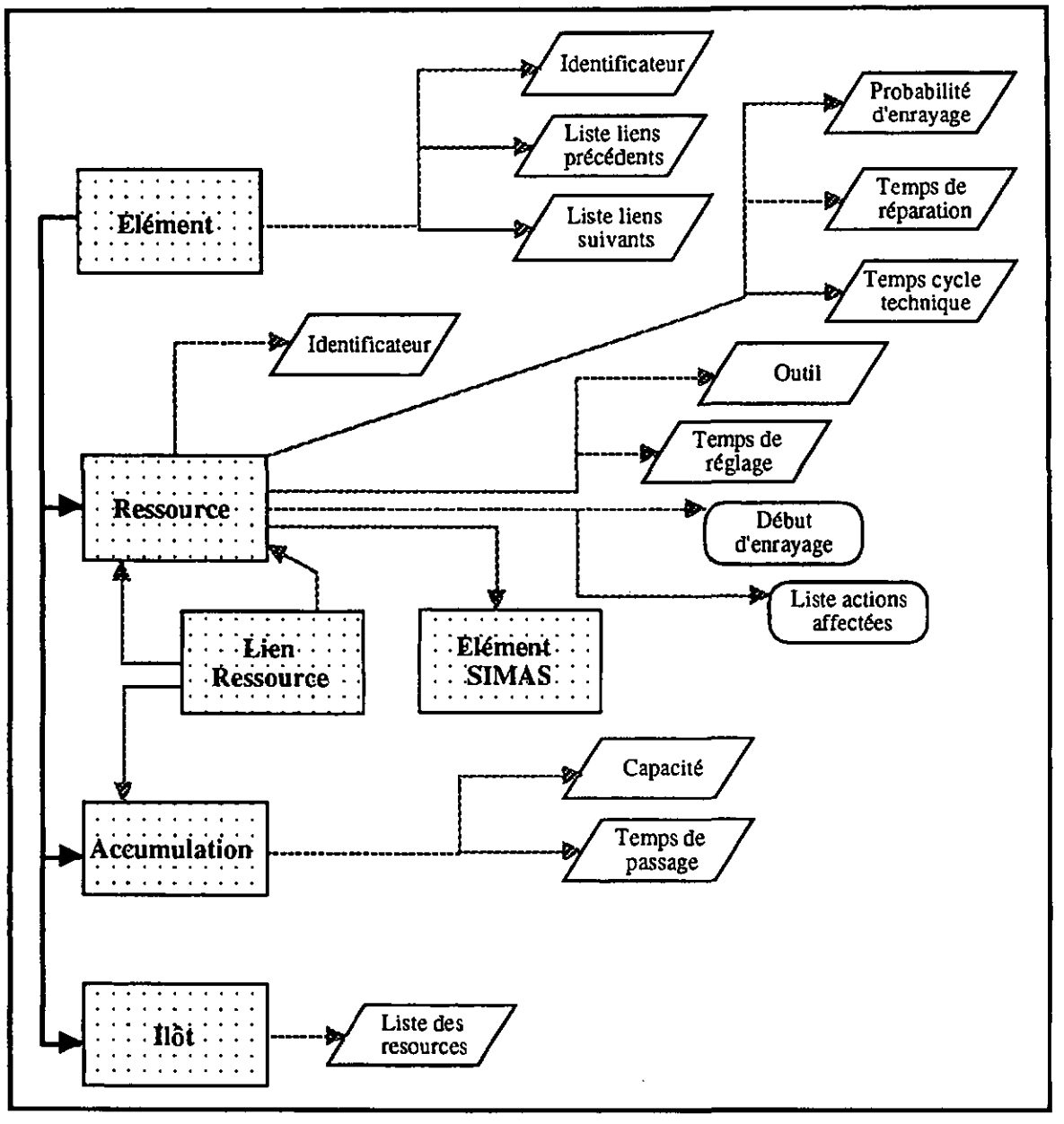


Figure 10.2: *Schéma du modèle de l'élément de l'installation.*

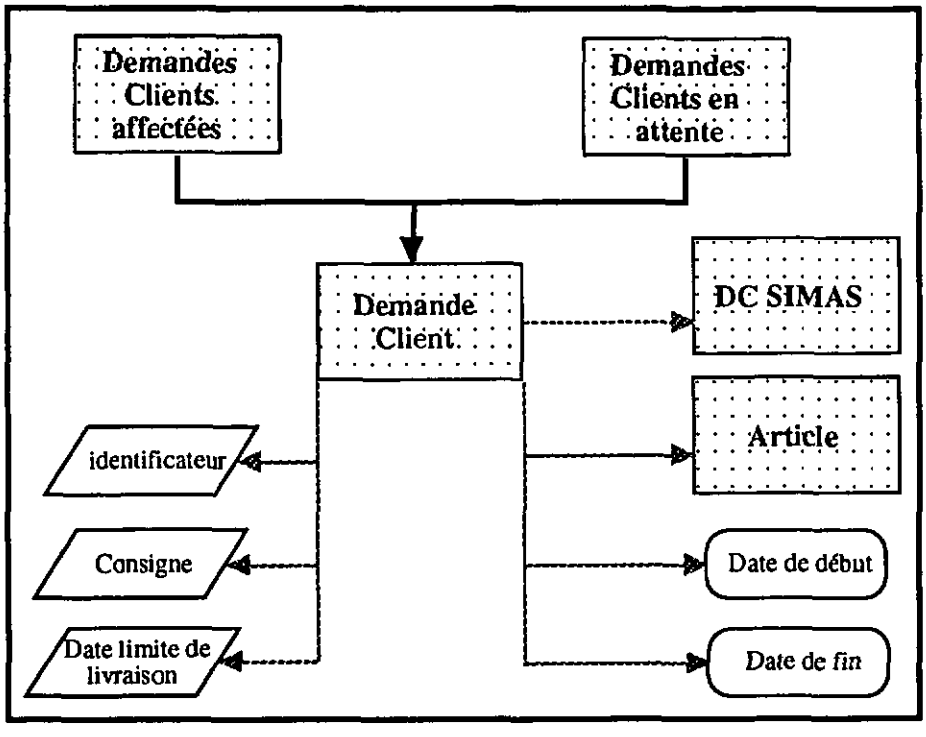


Figure 10.3: *Schéma du modèle de la demande client.*

Elle est caractérisée également par

- un outil installé initialement;
- un temps de réglage, au cas où il y a changement d'outils;
- un élément SIMAS correspondant;
- une liste ordonnée des actions déjà affectées à la ressource;
- une information dynamique qui indique le début de l'enrayage.

Une accumulation est constituée de cases et est caractérisée par sa capacité de stockage, autrement dit, le nombre de cases qu'elle peut contenir, et le temps que met un objet pour passer d'une case à une autre.

Enfin, un îlot est représenté par une liste de ressources.

10.4.3 Modèle de la demande client

Dans la figure 10.3, une DC peut appartenir à l'ensemble des DCs déjà affectées ou à celles qui sont encore en attente. Elle a une DC SIMAS correspondante et est caractérisée par

- un identificateur;
- un article à fabriquer;
- une consigne indiquant la quantité souhaitée;
- une date limite de livraison;

et par deux informations dynamiques, sa date de début et sa date de fin.

10.4.4 Modèle de l'action

Dans la figure 10.4, une action peut être

- une livraison;
- une alimentation;
- un contrôle;
- une opération;
- un transfert;
- un processus.

Elle est caractérisée par

- un identificateur;
- l'outil nécessaire pour son exécution;
- une liste de liens en sorties;
- une liste de liens en entrées;
- une action SIMAS associée;
- une ressource à laquelle elle est/sera affectée;
- le bon de travail SIMAS associé;

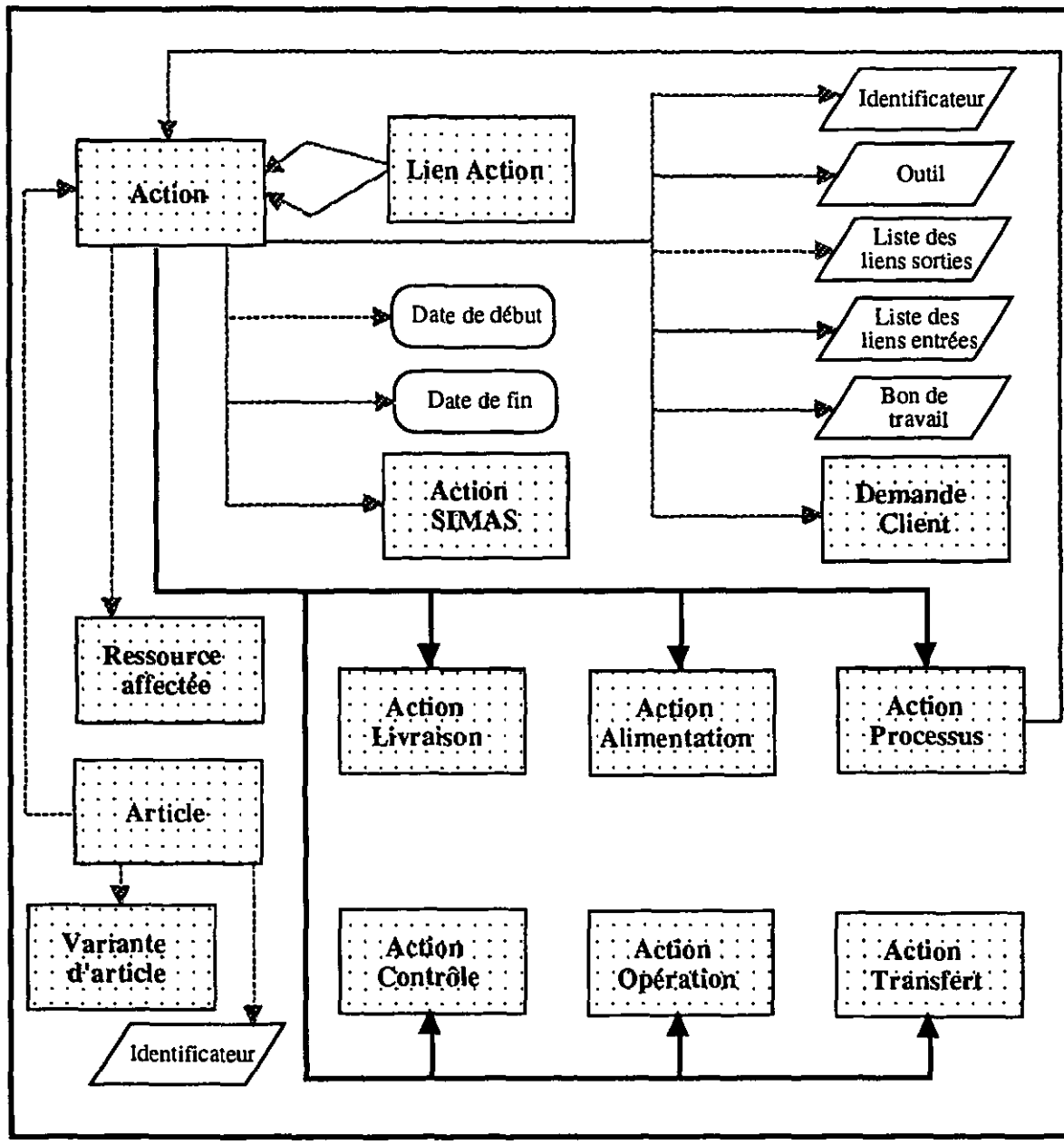


Figure 10.4: *Schéma du modèle de l'action.*

- la demande client à laquelle elle appartient;

ainsi que deux informations dynamiques qui sont sa date de début et sa date de fin. Un lien action est constitué de deux actions, une action précédente et une action suivante. Une action peut avoir autant de liens en entrées que le nombre d'actions précédentes et autant de liens en sortie que le nombre d'actions suivantes.

Nous avons déjà vu dans le modèle de demande client qu'une DC est liée à un article. Cet article peut être l'une des variantes d'articles, il est caractérisé par un identificateur et est lié à une action.

10.4.5 Modèle des agents

Dans la figure 10.5, un agent peut être

- un agent DC;
- un agent Ressource;
- un agent Interface.

Les agents DC sont liés aux agents Ressource et vise versa. De même, l'agent Interface est lié aux agents DC et aux agents Ressource.

Chaque agent Ressource est lié à sa ressource correspondante et est caractérisé par une tolérance et un coût courant.

Chaque agent DC est lié à sa DC correspondante et est caractérisé par son rang sur les ressources et son état de satisfaction qui indique s'il est affecté ou encore en attente.

L'unique agent Interface possède le nombre et la liste des meilleures solutions trouvées à fournir à l'utilisateur, car il joue le rôle d'intermédiaire entre la société d'agents et l'utilisateur. Il est caractérisé par la liste des agents DCs affectés et la liste des agents DCs en attente.

10.4.6 Modèle de la solution

Dans la figure 10.6, une solution représentant une séquence de DCs ordonnancées, est caractérisée par

- un identificateur;
- la liste des DCs ordonnancées;
- le coût global qu'implique cette solution;
- et le nombre de DCs en retard dans cette solution.

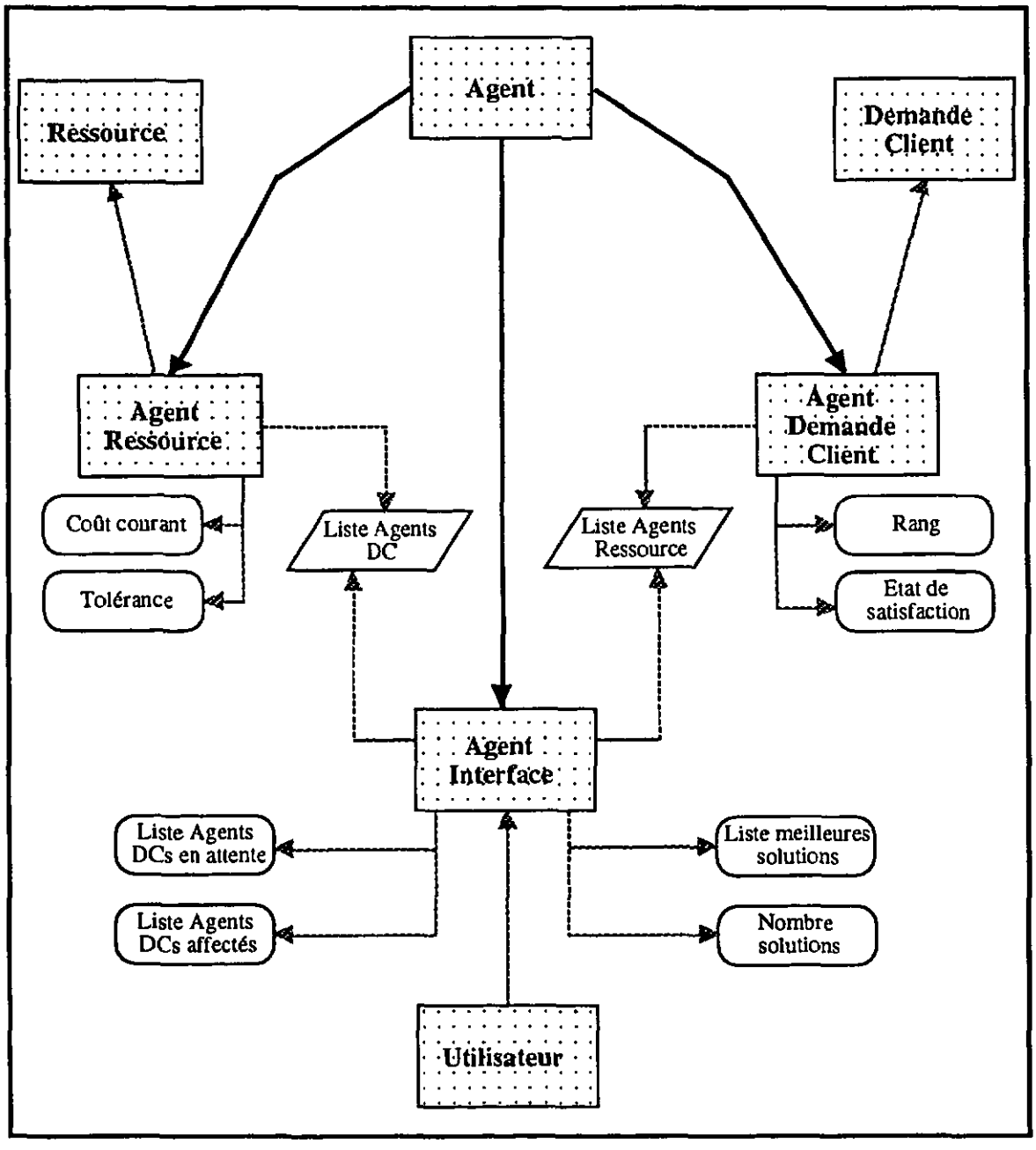


Figure 10.5: *Schéma du modèle des agents.*

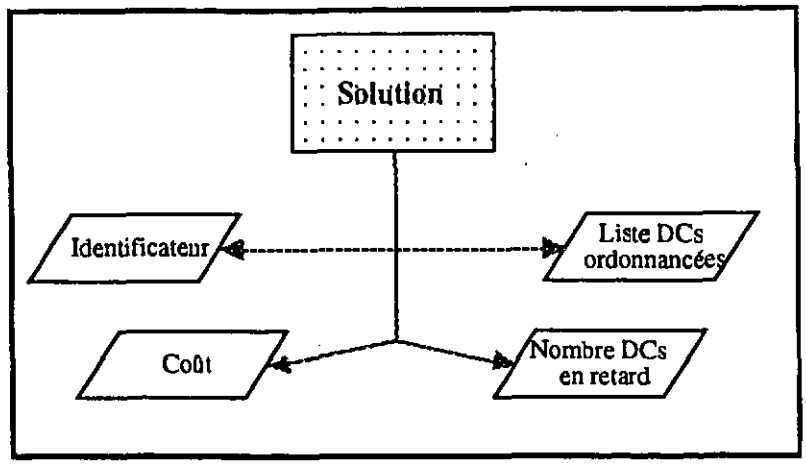


Figure 10.6: *Schéma du modèle de la solution.*

10.5 Résolution

Notre module MARSA peut être spécifié par ses entrées et ses sorties.

Les entrées apparaissent sous plusieurs formes. Initialement, elles apparaissent sous la forme de fichiers de données. Interactivement, elles apparaissent sous la forme d'événements envoyés de la part de SIMAS ou encore sous la forme de préférences de l'utilisateur introduites par lui-même.

Les sorties sont une ou plusieurs solutions. Une solution est un ensemble de DCs ordonnancées, autrement dit une séquence d'ordres de fabrication (OFs).

Nous allons présenter le fonctionnement général du système et, par la suite, entamer la structuration du prototype.

10.5.1 Fonctionnement général du système

Avant de commencer tout traitement, une phase initiale est nécessaire pour démarrer le prototype MARSA. Elle consiste en la lecture du fichier de spécification de l'installation et la lecture du fichier de spécification du processus opératoire.

En présence des données lues à partir de ces deux fichiers, le système est prêt à recevoir des DCs à ordonnancer. Les DCs qui entrent dans le système sont créées à l'aide d'un générateur de DCs où chaque DC est associée à son OF correspondant.

Un ordonnancement peut être déclenché avec l'ensemble des DCs qui existent et qui sont en attente. Par ailleurs, d'autres DCs peuvent entrer interactivement dans le système et un réordonnancement se déclenche, à chaque entrée d'une DC.

Le système peut proposer une ou plusieurs solutions. A partir d'une solution sélectionnée, l'utilisateur peut introduire ses préférences avant l'envoi des DCs vers SIMAS.

Les DCs ordonnancées sont envoyées au fur et à mesure à SIMAS sous la forme de bons de travail, sachant qu'un bon de travail correspond à une action.

Les fonctions Lisp de lecture du fichier de l'installation ont été développés en collaboration entre l'IIIA et l'IMT alors que les fonctions Lisp de lecture du processus opératoire et de génération de DCs ont été développés par l'IMT. Nous verrons, plus en détails, les différentes possibilités du prototype dans la section suivante.

10.5.2 Structuration et architecture du prototype

Le prototype est structuré de telle manière qu'il réalise les six types de traitements suivants:

- initialisation;
- traitement manuel;
- traitement multi-agents;
- modification;
- mise à jour;
- affichage des solutions.

ALGORITHME 6 Fonction principale d'initialisation

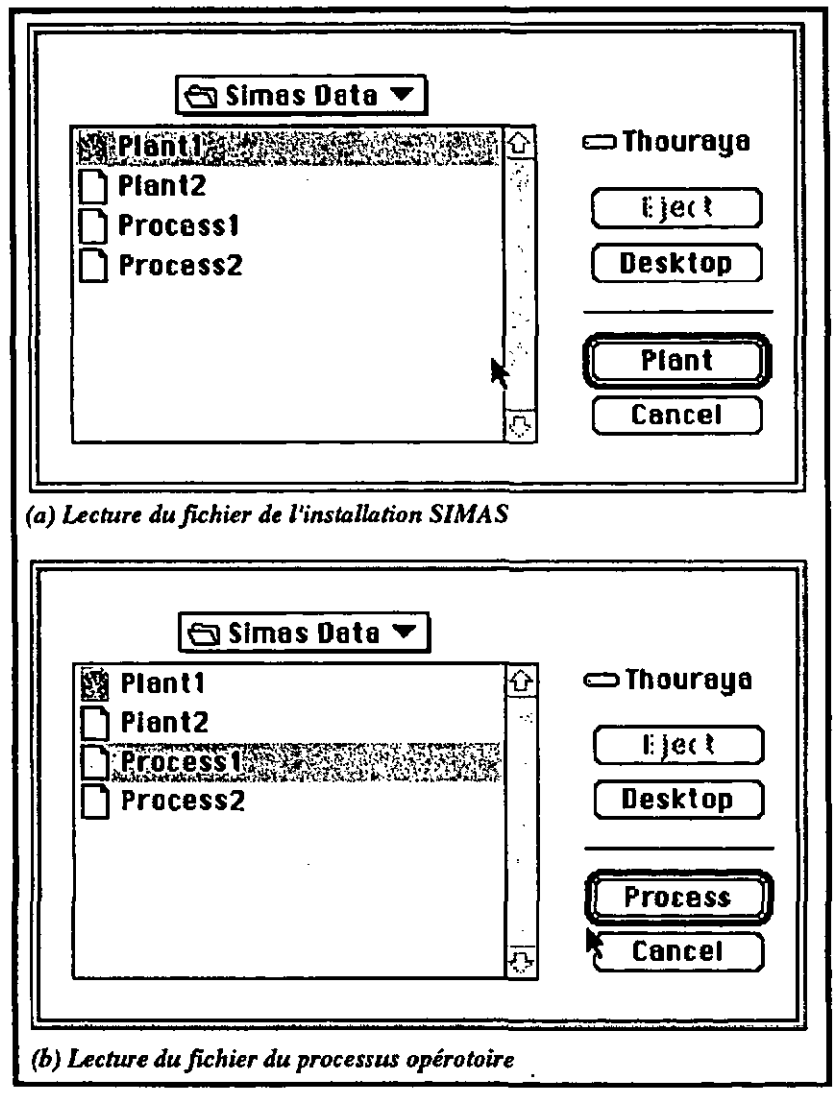
Lire-Installation-Simas *;lecture du fichier de spécification*
Créer-Instance-installation-Marsa
Créer-Ressources-Marsa *;à chaque élément correspond une ressource*
Créer-Liens-Ressources
Créer-AgentInterface
Calculer-TempsCycleGlobal

Initialisation

Une fonction principale d'initialisation est donnée par l'algorithme 6.

La lecture du fichier de l'installation se fait à l'aide d'une fonction:

(read-file (choose-file-dialog: button-string "Plant"))



(a) Lecture du fichier de l'installation SIMAS

(b) Lecture du fichier du processus opérotoire

Figure 10.7: *Lecture des deux fichiers de données de SIMAS.*

A l'appel de cette fonction, la boîte de dialogue (a) de la figure 10.7 s'affiche sur l'écran.
Cette lecture nous permet de lire les données qui sont représentées dans les modèles de l'installation et celui de l'élément. Comme résultat de cette lecture nous obtenons:

- l'instance de l'installation SIMAS;
- la liste des éléments SIMAS correspondants.

La lecture du fichier du processus opératoire se fait à l'aide d'une fonction :

(read-process (choose-file-dialog: button-string "Process") installation)

A l'appel de cette fonction, la boîte de dialogue (b) de la figure 10.7 s'affiche sur l'écran.
Cette lecture nous permet de lire la spécification du processus opératoire qui sera utilisé pour la création des DCs et leurs actions, donc les données des modèles de la DC et de l'action.

Traitement manuel

Le traitement manuel est effectué à l'aide de quatre options réservées à l'utilisateur:

Création de DCs: La boîte de dialogue (a) de la figure 10.8 est affichée suite à la sélection de cette option pour que l'utilisateur introduise le nombre de DCs à créer. La création de DCs se fait au moyen d'un générateur aléatoire de DCs qui crée une instance de DC SIMAS, son OF associé, le processus instancié associé et les bons de travail.
De notre côté, une création d'une DC MARSA correspondante se fait, ainsi que l'agent DC associé. La création d'une DC implique la création de toutes ses actions MARSA.

Insertion ou suppression d'une DC: Les boîtes de dialogue (b) et (c) de la figure 10.8 sont affichées suite à la sélection de l'option insertion ou suppression pour que l'utilisateur introduise le numéro d'ordre de la DC à insérer ou à supprimer de la séquence. Nous précisons que pour insérer une DC, il faut introduire la date à laquelle elle doit commencer.

Permutation de deux DCs: La boîte de dialogue (d) de la figure 10.8 est affichée suite à la sélection de cette option. L'utilisateur doit introduire les numéros d'ordre des deux DCs avec leurs dates de début respectives.

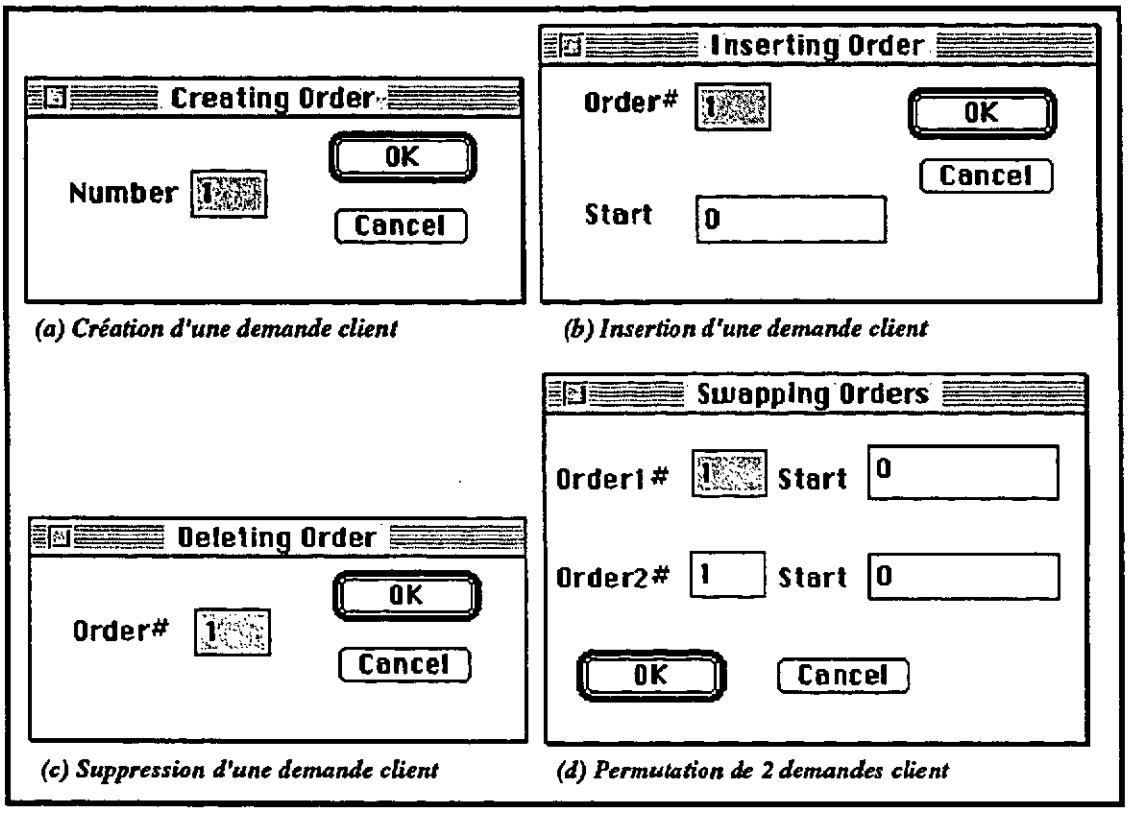


Figure 10.8: *Les boîtes de dialogue du traitement manuel du prototype.*

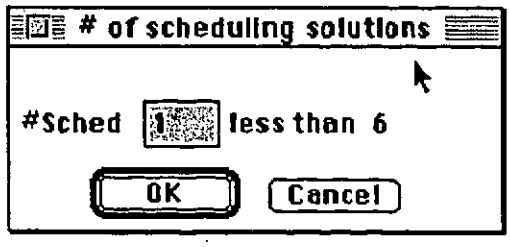
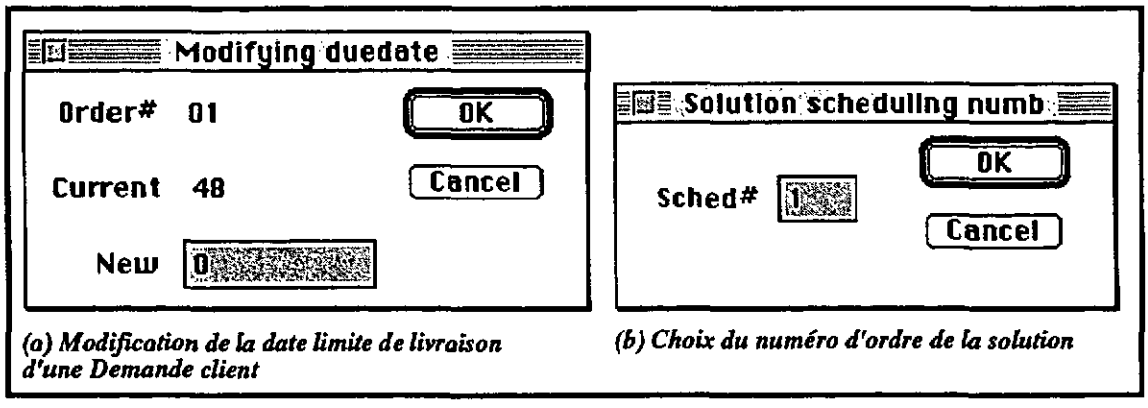


Figure 10.9: *boîte de dialogue pour introduire le nombre de solutions demandé par l'utilisateur.*



(a) Modification de la date limite de livraison d'une Demande client

(b) Choix du numéro d'ordre de la solution

Figure 10.10: *boîtes de dialogue pour modifier la date limite d'une Demande Client ou pour introduire le numéro d'ordre de la solution à visualiser.*

Traitement multi-agents

Les options du traitement multi-agents dans le prototype sont:

- une option qui déclenche l'ordonnancement de l'ensemble des DCs existantes;
- une option pour essayer d'améliorer la solution qui déclenche à nouveau l'ordonnancement.

La boîte de dialogue de la figure 10.9, s'affiche sur l'écran suite à l'appel de l'ordonnancement. L'utilisateur doit introduire le nombre maximum de solutions qu'il veut obtenir. Le nombre mentionné à droite correspond au nombre de combinaisons possibles avec les DCs dont dispose le système.
Au cours du traitement de l'ordonnancement, des messages sont affichés pour indiquer si une DC est affectée, éjectée ou si le traitement est terminé.
Les solutions seront sauvegardées selon l'ordre décroissant de leurs coûts globaux respectifs.

Modification

Pour l'instant, il existe une option dans le prototype qui est la modification de la date limite de livraison d'une DC. Il s'agit là d'une donnée importante pour notre traitement. Il est prévu d'ajouter d'autres options permettant la modification d'autres données du problème.

La boîte de dialogue (a) de la figure 10.10 s'affiche suite à la sélection de cette option. Elle contient le numéro d'ordre de la DC et sa date limite de livraison courante. L'utilisateur peut introduire donc la nouvelle date limite qu'il désire attribuer à la DC.

Mise à jour

Lorsque l'utilisateur sélectionne une solution parmi celles trouvées au cours du traitement, il peut choisir l'option de mise à jour pour remplacer la solution courante par la solution choisie. Il pourra continuer la recherche d'une solution meilleure à partir de celle là.

Affichage des solutions

La boîte de dialogue (b) de la figure 10.10 s'affiche suite à la sélection de l'option d'affichage des meilleures solutions. L'utilisateur peut introduire le numéro d'ordre de la solution qu'il veut visualiser.
Les solutions sont sélectionnées selon le critère du coût. D'autres critères de sélection vont être utilisés pour ordonner les solutions tels que:

- le nombre de DCs en retard;
- le nombre de changements d'outils;
- la somme des marges;
- la somme absolue des marges;
- etc.

L'affichage général dans le prototype est géré par un ensemble de fonctions chargées des entrées-sorties avec l'utilisateur. Elles réalisent des opérations de visualisations graphiques sous forme de diagramme de Gantt (voir section 2.2.2). Ces fonctions doivent être adaptées aux possibilités offertes par la machine sur laquelle s'exécute le programme.

Le traitement est visualisé à travers deux fenêtres (voir figure 10.11). La première appelée "*Scheduling*" pour ordonnancement, permet de visualiser les différentes actions sur les différentes ressources. La seconde, appelée "*Orders*" pour demandes clients, permet de visualiser les différentes DCs correspondantes aux actions. Il est aussi possible de visualiser, à travers une boîte (voir figure 10.12), la position ainsi que les dates de début et de fin d'une action ou d'une DC, juste en cliquant sur l'une ou sur l'autre.

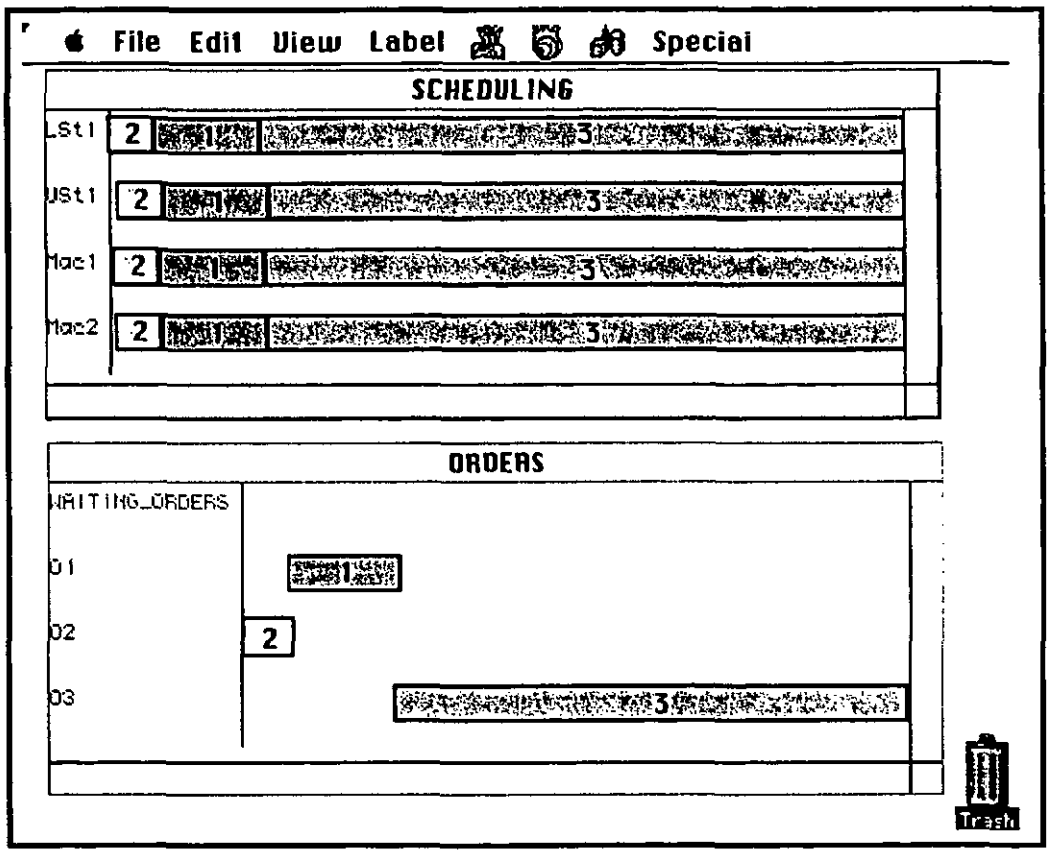


Figure 10.11: *Visualisation d'un exemple d'ordonnancement sur les deux fenêtres; Ordonnancement et DCs.*

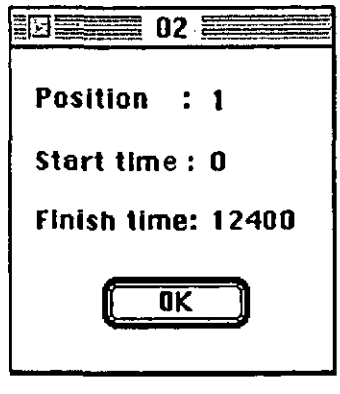


Figure 10.12: *boîte de visualisation de la position et les dates d'une DC ou d'une action.*

Chapitre 11

Simulateur SIMAS

Nous présentons dans ce chapitre le simulateur SIMAS, conçu à l'Institut de Microtechnique de Lausanne [Verdebout, 1992] et [Verdebout, 1994], et commercialisé par l'entreprise CIMPACT[1]. Il représente une simulation de l'atelier considérée dans notre travail, qui fait parvenir les DCs à notre système MARSA et qui les reçoit sous la forme de bons de travail pour exécution.

11.1 Présentation de SIMAS

SIMAS est un logiciel dédié à la simulation de systèmes industriels de production dans les domaines de l'assemblage et du conditionnement, dans le but d'optimiser leurs performances et leurs procédures de contrôle de flux.

Il rationalise la conception des stockages et des systèmes de transferts des machines et des stations manuelles de l'installation d'une manière simple à étudier, avec des outils nouveaux.

Au fur et à mesure que la modélisation d'une installation est effectuée, SIMAS fournit un moteur de simulation rapide et efficace. Ce moteur donne toutes les informations nécessaires d'une manière complètement configurable. Ces données sont directement transférables et exploitables sur n'importe quel tableur standard.

De même, à tout moment de la simulation, SIMAS génère un rapport des résultats, fournis sous la forme de données échantillonées périodiquement, à optimiser et à inclure dans nos études.

[1]CIMPACT, Eric Verdebout, EPFL Lausanne

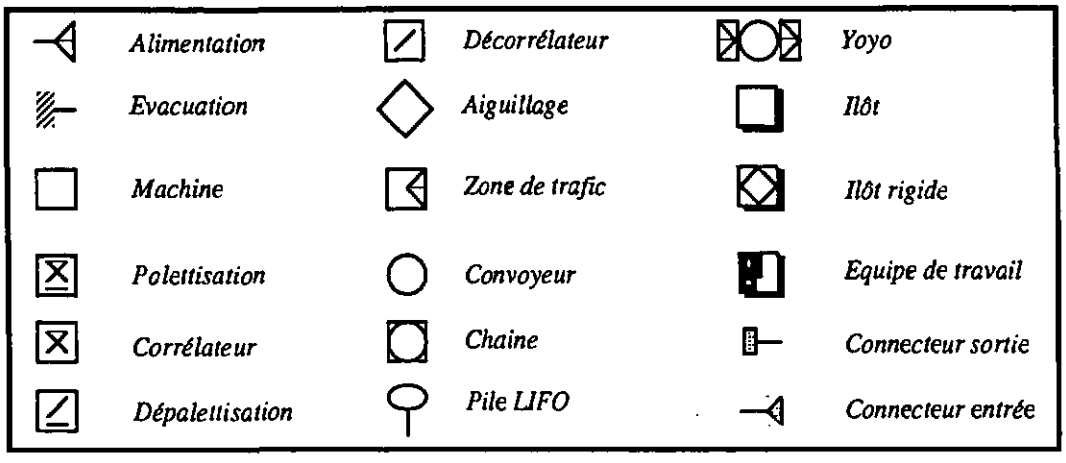


Figure 11.1: *Les éléments possibles de SIMAS.*

11.2 Représentation des éléments par SIMAS

Un élément est une partie du modèle de l'installation. Chaque élément est équipé d'un ensemble de canaux d'entrées et/ou de sorties. Ces canaux connectent les éléments entre eux au moyen de liens.
Ainsi, une installation est modélisée à l'aide de SIMAS par un ensemble d'éléments et de liens qui constituent un graphe. Lorsqu'un élément est créé, ses propriétés reçoivent un ensemble de paramètres définis par défaut et qui peuvent être modifiés par l'utilisateur. Nous présentons dans la figure 11.1 les éléments de SIMAS et leurs représentations.

11.3 Possibilités de SIMAS

Au niveau de l'édition d'un modèle d'installation, l'utilisateur a la possibilité de:

- identifier l'information de l'état de la validation de l'installation. Cette opération est nécessaire avant la simulation pour vérifier que tous les liens et paramètres sont complets et cohérents;
- sélectionner un élément (voir section 11.2) d'un lien. Un lien est représenté par une flèche qui part d'un élément et arrive à un autre élément;
- éliminer un objet non désirable du modèle. L'objet peut être un élément ou un lien;
- repositionner des éléments;

- créer un élément unique;
- faire une rotation d'un élément;
- créer un macro-élément qui est un élément, tel que la machine, qui n'est pas construit en une seule étape, comme l'alimentation par exemple, et doit être défini comme étant un ensemble de sous-éléments;
- valider l'installation et corriger les erreurs;
- établir un lien entre deux éléments;
- choisir un canal parmi les sorties possibles d'un élément;
- accéder aux paramètres d'un élément;
- définir le comportement du temps de cycle d'un élément;
- définir la probabilité d'enrayage d'un élément.

Au niveau de la simulation, l'utilisateur a la possibilité de:

- créer et adapter un résumé de description. C'est un rapport rapidement généré qui comporte toutes les informations appropriées;
- exporter un résumé de description;
- purger l'installation en remettant tous les éléments à leurs états initiaux. Ils seront vides et en état d'attente des composants;
- définir les données de production par défaut;
- purger l'installation en initialisant tous les compteurs de production à zéro;
- définir les paramètres du moteur de simulation;
- commencer une simulation;
- obtenir les résultats de simulation d'un élément;
- générer un rapport des résultats de simulation;
- etc.

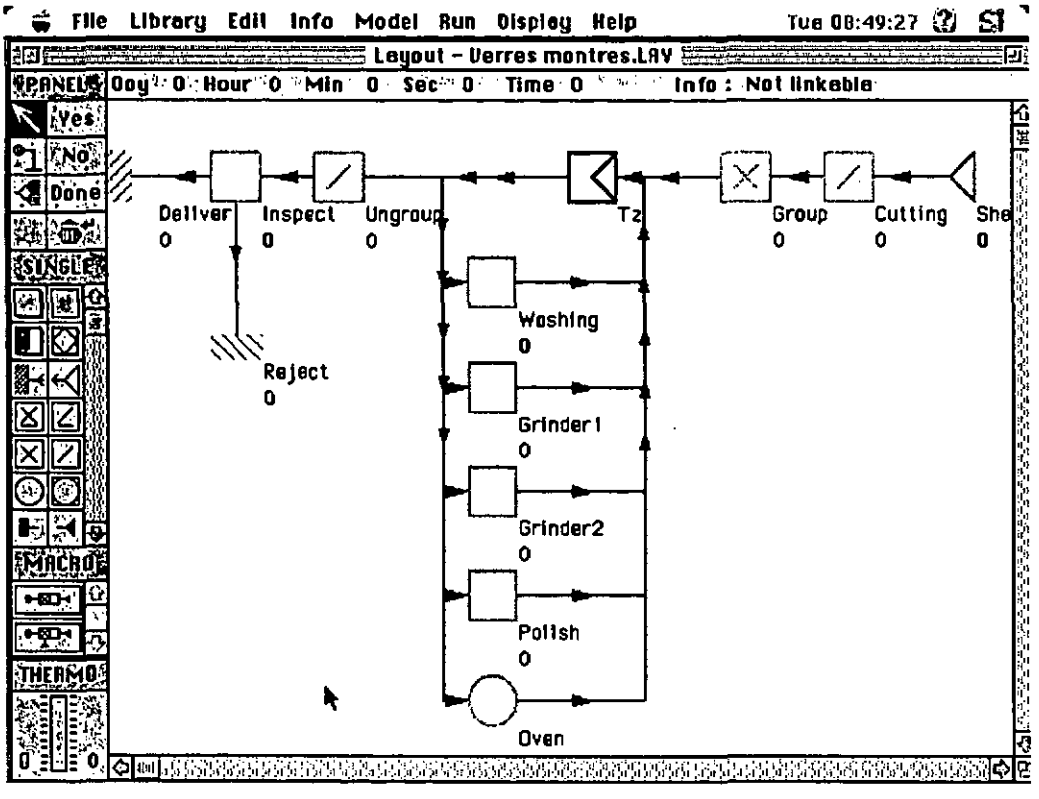


Figure 11.2: *Exemple d'installation pour la production des verres de montres.*

11.4 Exemple d'édition d'une installation

Nous avons choisi d'éditer un exemple extrait du manuel utilisateur de l'application SIMAS. Cet exemple consiste en la production de verres de montres qui sont successivement:

- découpés manuellement;
- lavés dans une machine à laver;
- meulés par deux meuleuses;
- lavés une deuxième fois dans la machine à laver;
- polis par une polisseuse;
- lavés une troisième fois;
- tempérés dans un four;
- lavés une quatrième fois;
- contrôlées avant la livraison pour faire le tri des mauvais verres et les rejeter.

L'installation correspondante à cet exemple est représentée dans la figure 11.2. C'est une image écran prise après l'édition de l'exemple avec l'application SIMAS. Les différents éléments ont les rôles suivants:

- *Sheets*: une alimentation qui insère les plaques de verres bruts;
- *Cutting*: un décorrélateur pour le découpage manuel des plaques de verres en disques bruts;
- *Group*: un corrélateur pour l'insertion des disques bruts dans des palettes, qui ne sont pas représentées;
- *Tz*: une zone de trafic qui représente tous les transports des palettes. Cet élément a six serveurs et six clients, chacun d'eux ayant une capacité de chariots nécessaire;
- *Washing*: une machine qui lave les palettes arrivant à son entrée à n'importe quel stade de production;
- *Grinder1 et 2*: deux meuleuses;
- *Polish*: une polisseuse;

- *Oven*: un convoyeur qui représente le four;
- *Ungroup*: un décorrélateur qui décharge les palettes pour le contrôle de chaque verre de montre résultant;
- *Inspect*: une machine qui représente la station d'inspection ou de contrôle. Son canal de sortie sépare les bons des mauvais composants;
- *Reject*: une sortie qui rassemble les composants rejetés après inspection. Dans un atelier réel, ces composants peuvent être meulés et polis, une deuxième fois, pour des diamètres inférieurs. Ils seront donc considérés comme étant des composants d'un autre bon de travail;
- *Deliver*: une sortie qui rassemble les bons produits.

Imaginons qu'un ensemble de chariots soit disponible à chaque point d'entrée de la zone de trafic manuelle, la machine à laver est un point critique de l'installation et doit être simulée. De plus, chaque élément qui renvoie des composants vers cette machine à laver sera considéré comme étant connecté au serveur de la zone de trafic.

D'une part, la macbine à laver utilise un mode concurrent de travail, car elle reçoit des composants à différents stades de production. D'autre part, elle doit envoyer les composants lavés à la destination appropriée par le moyen des règles de trafic de la zone de trafic.

Chapitre 12

Conclusion générale

12.1 Contexte et résolution

MARSA est une approche multi-agents avec recuit simulé des problèmes d'ordonnancement d'atelier de type *flow shop* de permutation. Elle combine la modélisation multi-agents avec la technique de recherche stochastique du recuit simulé, afin de fournir des ordonnancements de qualité en un temps raisonnable.

Disposant d'un ensemble de ressources et d'un ensemble de Demandes Clients (DCs), formée chacune d'un graphe d'actions à placer sur les ressources, nous avons modélisé le problème en définissant deux classes principales d'agents: les agents Ressource et les agents DCs. Ces agents interagissent (ensemble) afin d'atteindre un état optimal. Cet état est le meilleur placement possible des DCs sur les ressources, qui correspond à la séquence optimale d'ordres de fabrications à exécuter.

La recherche de l'état est dirigé par le principe du recuit simulé qui est distribué au niveau de chaque agent Ressource. Une fonction objectif englobe tous les critères que nous voulons optimiser. Elle intervient dans la prise de décision des agents Ressource pour passer d'un état à un autre.

12.2 Résultats

Aspect statique

Nous avons montré, par des séries d'expérimentations, qu'une version de base du modèle statique présente un bon compromis entre le temps de traitement et la qualité d'optimalité des solutions trouvées.

Néanmoins, si nous sommes intéressés par l'amélioration de la qualité de solution, une version de base, à laquelle nous avons ajouté deux heuristiques HMP et HA, a montré sa performance, en qualité en comparaison avec toutes les autres

versions testées.

Aspect dynamique

Pour tester le modèle dans son aspect dynamique, nous avons effectué des expérimentations sur un cas réel d'installation d'atelier, afin de calculer les temps de réponse du modèle MARSA suite aux différents événements qui peuvent arriver.

Les résultats ont montré que le réordonnancement dynamique, suite à l'ajout d'nne nouvelle DC, met moins de temps de traitement qu'un ordonnancement à zéro (voir chapitre 7) et que pour tous les événements, à mesure que la taille du problème est plus élevée, le pourcentage d'augmentation du temps de traitement diminue. Ainsi le modèle est favorisé pour les problèmes de tailles très grandes.

12.3 Perspectives et extension

Nous nous proposons de citer deux perspectives en vue qui peuvent être étudiées à la suite de ce travail:

- étendre et généraliser ce travail à la résolution des problèmes d'ordonnancement dans ateliers de type *Flow shop* hybride, serait très interessant pour l'étude du problème d'affectation des actions à des ressources en parallèle;
- étendre et généraliser ce travail à la résolution des problèmes d'ordonnancement dans ateliers de type *Job shop*, ferait un excellent projet d'étude surtout que le type *flow shop* présente un cas particulier du type *Job shop*;
- programmer le modèle MARSA sur des machines parallèles pourrait mettre, encore plus, en valeur la concurrence et la coopération entre les agents pour atteindre des solutions avec des temps de traitement encore plus intéressants.

Bibliographie

[Afnor, 1990] Afnor (1990). Organisation de la production industrielle, concepts fondamentaux de la gestion de production vocabulaire. Norme francaise NF X 50-310.

[Allen, 1981] Allen, J. F. (1981). An interval based representation on temporal knowledge. Dans *7th International Conference on Artificial Intelligence.*

[Badie et Verfaillie, 1989] Badie, C. et Verfaillie, G. (1989). Oscar ou comment planifier intelligemment des missions spaciales. Dans *Avignon'89.*

[Balas, 1993] Balas, E. (1993). Improvements of the shifting bottleneck procedure for job shop scheduling. Dans *Workshop on Models and Algorithms for Planning and Scheduling Problems Italy.*

[Baptiste et al., 1995] Baptiste, P., O.Grunder, et Chappe, D. (1995). Utilisation d'un branch and bound pour la détermination d'une implantation optimale pour une ligne de production en fonctionnement saturé et désservie par un seul robot. Rapport technique, Laboratoire d'automatique de Besancon.

[Bel et al, 1992] Bel, G. et al (1992). Représentation et traitement pratique de la flexibilité dans les problèmes sous contraintes. Dans *Actes des quatrièmes journées natinales PRC-IA, Marseille.*

[Benasser et al., 1995] Benasser, A., Bon, P., Lefort, A., et Yim, P. (1995). Un algorithme d'accessibilité pour les réseaux de pétri: Application aux sed. Dans *Journées d'Etude: Affectation et Ordonnancement, Ecole d'Ingénieurs en Informatique pour l'Industrie, Septembre, Tours*, page 165.

[Berry et al., 1992] Berry, P. M., Choueiry, B. Y., et Friha, L. (1992). A multi-agent architecture for a distributed approach to resource allocation using temporal abstraction. Rapport technique, EPFL.

[Blazewicz et al., 1993] Blazewicz, J., Ecker, K., Schmidt, G., et Weglarz, J. (1993). *Scheduling in Computer and Manufacturing Systems.* Springer-Verlag.

[Boucher et al., 1995] Boucher, E., Legeard, B., et Zidoum, H. (1995). Utilisation de la plc ensembliste pour la modélisation et la résolution d'un problème d'affectation. Dans *Journées d'Etude: Affectation et Ordonnancement, Ecole d'Ingénieurs en Informatique pour l'Industrie, Septembre, Tours*, page 151.

[Bowman, 1959] Bowman, E. H. (1959). The schedule-sequencing problem. Dans *Operations Research*, volume 7, page 621.

[Brooks, 1991] Brooks, R. (1991). A robot that walks: Emergent behaviors from a carefully evolved network. Rapport technique, Massachusetts Institute of Technology, A. I. Memo.

[Brucker et al., 1992] Brucker, P., Jurisch, B., et Sievers, B. (1992). A branch and bound algorithm for the job shop scheduling problem. Dans *Osnabrucker Schriften zur Mathematik.* Universitat Osnabruck.

[Burke et Prosser, 1990] Burke, P. et Prosser, P. (1990). Distributed asynchronous scheduling das. Rapport technique, University of strathclyde.

[Carlier et Rebai, 1996] Carlier, J. et Rebai, I. (1996). Two branch and bound algorithms for the permutation flow shop problem. Dans *European Journal of Operational Research.*

[Choueiry et Faltings, 1992] Choueiry, B. Y. et Faltings, B. (1992). Interactive resource allocation using temporal abstraction delay heuristic: a csp perspective. Rapport technique, EPFL.

[Collinot et al., 1988] Collinot, A., Pape, C. L., et Pinoteau, G. (1988). Sonia: A knowledge based scheduling system. Dans *AI in Engineering.*

[Corbara et al., 1991] Corbara, B., Drogoul, A., Ferber, J., et Fresneau, D. (1991). A behavioral simulation model for the study of emergent social structures. Dans *The first European Confence on Artificial Life.*

[Cox et al., 1992] Cox, J. F., Blackstone, J. H., et Spencer, M. S. (1992). *APICS Dictionary.* University of Georgia, Second edition.

[Daouas et al., 1994] Daouas, T., Ghédira, K., et Muller, J. P. (1994). Une approche multi-agents avec recuit simulé pour l'ordonnancement dans un atelier flexible. Dans *Rencontres Nationales sur la Résolution Pratique de Problemes NP-Complets, Montpelier, France.*

[Daouas et al., 1995a] Daouas, T., Ghédira, K., et Muller, J. P. (1995a). A distributed approach for the flow shop problem. Dans *International Conference on Artificial Intelligence and Applications, Cairo, Egypt*, page 290.

[Daouas et al., 1995b] Daouas, T., Ghédira, K., et Muller, J. P. (1995b). Distributed flow shop scheduling problem global versus local optimization. Dans *International Conference of Multi-Agent systems, San Francisco, USA*, page 443.

[Daouas et al., 1995c] Daouas, T., Ghédira, K., et Muller, J. P. (1995c). How to schedule a flow shop plant by agents. Dans *International Conference on Applications of Artificial Intelligence, AIENG, Udyne, Italy.*

[Daouas et al., 1995d] Daouas, T., Ghédira, K., et Muller, J. P. (1995d). Ordonnancement distribué dans un atelier de type flow shop. Dans *3èmes journées Francophones des SMA-IAD, Chambéry, France*, page 79.

[Davenport et al., 1994] Davenport, A., Tsang, E., Wang, C. J., et Zhu, K. (1994). Genet: A connectionist architecture for solving constraint satisfaction problems by iterative improvement. Dans *AAAI'94.*

[Demazeau et Muller, 1989] Demazeau, Y. et Muller, J. P. (1989). Decentralized artificial intelligence. Dans *Decentralized A.I. The first European Workshop on Modeling Autonomous Agents in a Multi-Agent World, Cambridge, England, August*, page 3.

[Demazeau et Muller, 1990] Demazeau, Y. et Muller, J. P. (1990). From reactive to intentional agents. Dans *Decentralized artificial intelligence 2. The second European Workshop on Modeling Autonomous Agents in a Multi-Agent World, Saint-Quentin en Yvelines, France*, page 3.

[Ferber et Jacopin, 1990] Ferber, J. et Jacopin, E. (1990). The framework of eco-problem solving. Dans *Decentralized artificial intelligence, Editions North Holland, Vol2.*

[Fichera et al., 1995] Fichera, S., Grasso, V., Lombardo, A., et Valvo, E. L. (1995). Genetic algorithms efficiency in flow shop scheduling. Dans *AIENG'95.*

[Fox et Sadeh, 1990] Fox, M. S. et Sadeh, N. M. (1990). Why is scheduling difficult? a csp perspective. Dans *Proceedings of the 1990 European Conference on AI.*

[Fox et Smith, 1984] Fox, M. S. et Smith, S. F. (1984). Isis: A knowledge-based system for factory scheduling. Dans *Expert Systems Journal*, volume 1, page 25.

[Freuder, 1982] Freuder, E. C. (1982). A sufficient condition fro backtrack-free search. Dans *JACM*, volume 29, page 24.

[Ghédira, 1993] Ghédira, K. (1993). *Une approche Multi-agents des problèmes de satisfaction de contraintes.* Thèse, "Ecole Nationale Supérieure de l'Aéronautique et de l'espace".

[Ghédira, 1994a] Ghédira, K. (1994a). Distributed simulted or re-annealing for dynamic constraint satisfaction problem. Dans *TAI'94 New-Orleans.*

[Ghédira, 1994b] Ghédira, K. (1994b). Partial constraint satisfaction by a multi-agent-simulated annealing approach. Dans *Quatrièmes Journées Internationales d'Avignon Paris*, page 241.

[Ghédira et Daouas, 1995] Ghédira, K. et Daouas, T. (1995). Distributed scheduling for the flow shop problem montréal. Dans *Intelligent Manufacturing Systems, Workshop IJCAI.*

[Ghédira et Verfaillie, 1991] Ghédira, K. et Verfaillie, G. (1991). Approche multi-agents pour les problèmes d'affectation. Dans *Journées internationnales: Les systèmes experts et leurs applications, Avignon.*

[Ghédira et Verfaillie, 1992a] Ghédira, K. et Verfaillie, G. (1992a). Approche multi-agents d'un problème de satisfaction de contrainte: optimalité et réactivité. Dans *Conférence internationnale: intelligence artificielle, systèmes experts et langage naturel, Avignon.*

[Ghédira et Verfaillie, 1992b] Ghédira, K. et Verfaillie, G. (1992b). A multi-agent model for the resource allocation problem: a reactive approach. Dans *Actes ECAI 92, Vienne.*

[Glover, 1990] Glover, F. (1990). Tabu search: part 2. Dans *Operations Research Society of America (ORSA), Journal on computing*, volume 1, page 4.

[GOTHA, 1993] GOTHA (1993). Les problèmes d'ordonnancement. *Recherche opérationnelle*, 27: 77. .

[Greening, 1990] Greening, D. R. (1990). Parallel simulated annealing techniques. Dans *Physica*, volume 42, page 293.

[Hall et Wood, 1971] Hall, P. A. V. et Wood, D. E. (1971). Branch-and-bound methods: a survey. Dans *Proc International Joint Conference on AI pp 641-650.*

[Johnson, 1954] Johnson, S. M. (1954). Optimal two and three-stage production schedules with setup times included. Rapport technique, Naval Research Logistics Quarterly 1 61-68.

[Kirkpatrick et al, 1983] Kirkpatrick, S. et al (1983). Optimization by simulated annealing. Dans *Science p 220.*

[Laarhoven et Aarts, 1992a] Laarhoven, P. J. M. V. et Aarts, E. H. L. (1992a). *Simulated annealing: theory and applications*, chapter 2. Kulwer Academic Publishers.

[Laarhoven et Aarts, 1992b] Laarhoven, P. J. M. V. et Aarts, E. H. L. (1992b). *Simulated annealing: theory and applications.* Kulwer Academic Publishers.

[Laarhoven et al., 1992] Laarhoven, P. J. M. V., Aarts, E. H. L., et Lenstra, J. K. (1992). Job shop scheduling by simulated annealing. Dans *Operations Research*, volume 40, page 113.

[Lewis et El-Rewini, 1992] Lewis, T. G. et El-Rewini, H. (1992). *Introduction to parallel computing.* Prentice-Hall Int. Editions.

[Li et Jiang, 1995] Li, Y. H. et Jiang, Y. J. (1995). Localized simulated annealing in constraint satisfaction and optimization. Rapport technique, IC-parc.

[Liu et Sycara, 1995] Liu, J. et Sycara, K. P. (1995). Exploiting problem structure for distributed constraint optimization. Dans *International Conference in Muli-Agent Systems ICMAS'95.*

[Luo et al., 1993] Luo, Q. Y., Hendry, P. G., et Buchanan, J. T. (1993). An integrated distributed scheduler. Rapport technique, University of Strathclyde Glasgow.

[Minton et al., 1992] Minton, S., Johnston, M. D., Philips, A. B., et Laird, P. (1992). Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Dans *Artificial Intelligence Journal*, volume 58, page 161.

[Moccellin, 1995] Moccellin, J. V. (1995). A new heuristic method for the permutation flow shop scheduling problem. Dans *ournal of the Operational Research Society*, volume 46, page 883.

[Neiman et al., 1994] Neiman, D. E., Hildum, D. W., et Lesser, V. R. (1994). Expoiting meta-level information in a distributed scheduling system. Dans *Twelveth National Conference on AI, Juillet, Washington*, page 394.

[Ogbu et Smith, 1990] Ogbu, F. A. et Smith, D. K. (1990). The application of the simulated annealing algorithm to the solution of the n/m/cmax flowshop problem. *Computers Operational research*, 17: 243. 3.

[Pape, 1991] Pape, C. L. (1991). Constraint propagation in planning and scheduling. Rapport technique, Standford University.

[Pape, 1994] Pape, C. L. (1994). Implementation of resource constraints in ilog schedule: A library for the development of constraint-based scheduling systems. Dans *Intelligent Systems Engineering.*

[Pape et Sauve, 1985] Pape, C. L. et Sauve, B. (1985). Soja: Un système d'ordonnancement journalier. Dans *Les systèmes experts et leurs applications: cinquièmes journées internationnales Avignon.*

[Pape et Smith, 1988] Pape, C. L. et Smith, S. F. (1988). Management of temporal constraints for factory scheduling. Dans *Temporal aspects in information systems.* Elsevier Science Publishers B.V.

[Popovic et Bhatkar, 1994] Popovic, D. et Bhatkar, V. P. (1994). *Methods and tools for applied artificial intelligence.* Marcel Dekker, Inc.

[Portmann et Ghedjati, 1995] Portmann, M. et Ghedjati, F. (1995). Affectation et ordonnancement par des algorithmes génétiques. Dans *Journées d'Etude: Affectation et Ordonnancement, Ecole d'Ingénieurs en Informatique pour l'Industrie, Septembre, Tours*, page 67.

[Rumbaugh et al, 1991] Rumbaugh, J. et al (1991). *Object-Oriented Modeling and Design.* Prentice-Hall International Editions.

[Selman et al., 1994] Selman, B., Jautz, H. A., et Cohen, B. (1994). Noise strategies for improving local search. Dans *twelvth National Conference for AI AAAI.*

[Smith et al., 1986] Smith, S. F., Ow, P. S., Pape, C. L., et Muscettola, N. (1986). Reactive management of factory schedules. Rapport technique, CMU.

[Steels, 1991] Steels, L. (1991). Towards a theory of emergent functionality. Dans *From animals to Animats, MIT Press.*

[Sycara et al., 1990] Sycara, K., Roth, S., et Fox, M. (1990). Distributed production control. Dans *Fourth International Conference on Expert Systems in Production and Operations Management.*

[Sycara et Liu, 1993] Sycara, K. P. et Liu, J. (1993). Emergent constraint satisfaction through multi-agent coordinated interaction. Dans *From Reaction to Cognition, 5th European Workshop on Modelling autonomous Agents in Multi-Agent World, MAAMAW'93, Neuchâtel, Switzerland, Août*, page 107.

[Taillard, 1990] Taillard, E. (1990). Some efficient heuristic methods for the flow shop sequencing problem. Dans *Journal of Operational Research*, volume 47, page 65.

[Taillard, 1993] Taillard, E. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research, North-Holland*, 64: 278. .

[Télémécanique, 1990] Télémécanique (1990). Vocabulaire de la fabrication et de la gestion de production. Document interne.

[Tomme, 1993] Tomme, V. (1993). Etude d'installations par simas ii. Rapport technique, Projet de diplôme IMT-EPFL.

[Verdebout, 1992] Verdebout, E. (1992). *Pilotage algorithmique du changement de campagne de production dans une installation flexible d'assemblage automatique.* Thèse, "EPFL".

[Verdebout, 1994] Verdebout, E. (1994). Simas ii 0.22, user's guide, the simulation package for assembly and production systems. Rapport technique, EPFL.

[Vignier et al., 1995] Vignier, A., Billaut, J.-C., et Proust, C. (1995). Les problèmes d'ordonnancement de type flow shop hybride: Etat de l'art. Dans *Journées d'Etude: Affectation et Ordonnancement, Ecole d'Ingénieurs en Informatique pour l'Industrie, Septembre, Tours*, page 7.

[Voudouris et Tsang, 1995] Voudouris, C. et Tsang, E. (1995). Guided local search. Rapport technique, Department of Computer Science, University of Essex, Colchester, UK.

[Zbigniew, 1994] Zbigniew, M. (1994). *Genetics Algorithms + Data structures = Evolution Program.* Second extended edition.

[Zegordi et al., 1995] Zegordi, S. H., Itoh, K., et Enkawa, T. (1995). A knowledgeable simulated annealing scheme for the early/tardy flow shop scheduling problem. Dans *Journal of Int. Prod. Res.*, volume 33, page 1449.

[Zweben et al., 1993] Zweben, M., Davis, E., Daun, B., et Deale, M. J. (1993). Scheduling and rescheduling with iterative repair. Dans *IIIE.*