

Gestion de production par système multi-agent auto-organisateur

Rapport de Master 2 Recherche
Web Intelligence (WI)



à l'Université Jean Monnet
et
l'École des Mines de Saint-Étienne

16 juin 2008

Gaël CLAIR

Laboratoire d'accueil : École Nationale Supérieure des Mines de Saint-Étienne
Centre Génie Industriel et Informatique
Responsable de stage : Gauthier PICARD
Equipe d'accueil : Département Systèmes Multi-Agents

Mots-clés : Auto-organisation, optimisation combinatoire, gestion de production, coopération.

Résumé : Dans le cadre du groupe de travail Colline (Collectif, Interaction, Émergence) de l'Association Française pour l'Intelligence Artificielle (AFIA), une plate-forme de simulation de gestion de production a été développée afin de comparer différentes approches. Le problème consiste en l'affectation de tâches à des opérateurs sur des machines d'usinage de pièces, en fonction d'un cahier de commandes. Nous souhaitons modéliser le problème sous la forme d'un problème de satisfaction de contraintes distribuées et le résoudre en utilisant des techniques d'auto-organisation, afin de proposer des solutions capables de répondre à la dynamique de l'environnement. Cette approche va mettre en action des agents qui vont coopérer pour parvenir à trouver la meilleure solution possible au scénario proposé. Après avoir analysé le problème, nous présenterons plusieurs approches classiques développées et les confronterons à l'approche auto-organisatrice imaginée. Pour comparer les avantages et les limites de cette approche nous verrons les résultats de quelques tests suivant divers critères et métriques propres à notre problème.

Remerciements

Je tiens à remercier dans un premier temps, toute l'équipe pédagogique de l'École des Mines de Saint-Étienne et de l'Université Jean Monnet ainsi que les responsables du Master Web Intelligence, pour avoir assuré la partie théorique de celle-ci.

Je remercie également Monsieur Picard Gauthier pour son aide et ses conseils concernant les problèmes évoqués dans ce rapport, qu'il m'a apporté lors du suivi de ce stage.

Je tiens à remercier tout particulièrement et à témoigner toute ma reconnaissance aux personnes suivantes, pour l'expérience enrichissante et pleine d'intérêt qu'ils m'ont fait vivre durant ces cinq mois au sein du département Systèmes Multi-Agents de l'EMSE :

Monsieur Olivier Boissier, responsable du département SMA pour m'avoir accueilli au sein de son équipe ; m'avoir accordé du temps lors de la recherche d'un sujet pour mon stage et son aide pour la recherche d'un sujet de thèse.

Messieurs Laurent Vercouter et Jomi Fred Hubner, professeur assistant et chercheur, pour m'avoir aidé sur de nombreuses questions aussi bien techniques que théoriques ; pour leurs remarques sur mon travail et pour m'avoir accueilli et supporté au sein même de leur bureau.

Mademoiselle Kitio Rosine, doctorante du centre, pour ses éclaircissements sur le déroulement d'une thèse et sa bonne humeur.

Enfin je tiens à remercier mon amoureuse pour son soutien permanent ; pour sa patiente et ses relectures dévastatrices mais productives.

Table des matières

Remerciements	iii
Introduction	1
I Analyse de l'existant	3
1 La gestion de production	5
1.1 Le contrôle manufacturier	5
1.2 Les systèmes multi-agents pour la gestion de production	6
1.2.1 Système Holonique	6
1.2.2 Stigmergie	7
1.3 Le groupe COLLINE	8
1.3.1 La plate-forme MASC	8
1.4 Bilan	10
2 Systèmes multi-agents et auto-organisation	11
2.1 Concepts	11
2.1.1 Systèmes Multi-Agents (SMA)	11
2.1.1.1 Agent	11
2.1.1.2 Environnement	12
2.1.1.3 Communication	12
2.1.2 Auto-organisation dans les systèmes multi-agents	12
2.1.2.1 Coopération	13
2.1.2.2 Émergence	14
2.2 Exemples d'approches auto-organisatrices par coopération	14
2.2.1 Affectation de fréquences (FAPP)	14
2.2.2 Emergent TimeTabling Organization (ETTO)	15
2.3 Bilan	16
3 Satisfaction de contraintes distribuées et dynamiques	17
3.1 Problème de satisfaction de contraintes (CSP)	17
3.1.1 Formalisation	17
3.1.2 Résolution	17
3.2 CSP distribué	19
3.2.1 Formalisation	19
3.2.2 Résolution	19
3.3 CSP dynamique	21
3.3.1 Formalisation	22
3.3.2 Résolution	22

3.4	Bilan	23
II	Comparaison et analyse de diverses approches	25
4	Gestion de production : étude de cas	27
4.1	Présentation	27
4.1.1	Description	27
4.1.1.1	Cahier des charges original	27
4.1.1.2	Cahier des charges modifié	28
4.1.2	Formalisation	28
4.1.3	Caractéristiques	29
4.2	Une première approche orientée agents : AmasCOP	29
4.3	Bilan	31
5	ABT et AWCS pour la gestion de production	33
5.1	Présentation	33
5.1.1	Principe	33
5.1.1.1	Asynchronous Backtracking	33
5.1.1.2	Asynchronous Weak-Commitment Search	35
5.1.2	Modélisation	37
5.1.2.1	Agentification	37
5.1.2.2	Réseau de contraintes	38
5.1.3	Implémentation	38
5.1.3.1	Pré-traitements	39
5.1.3.2	Communication	39
5.1.3.3	Terminaison	40
5.1.4	Heuristique	41
6	Approche multi-agent auto-organisatrice : ETTO4MC	43
6.1	Présentation	43
6.1.1	Motivations	43
6.1.2	Modélisation	44
6.1.2.1	Agentification	44
6.1.2.2	Réseau de contraintes	44
6.2	Principe général	45
6.2.1	Actions des agents	45
6.2.1.1	BookingAgent	46
6.2.1.2	RessourceAgent	48
6.2.2	Protocole de communication	49
6.2.3	Coopération	50
6.2.4	Limites	52
7	Expérimentations	53
7.1	Scénario de test	53
7.2	Résultats	54
7.2.1	Temps	54
7.2.2	Messages	56
7.2.3	Usinages	57
7.3	Analyse	58
7.3.1	Dynamique	58
7.3.2	Distribution	59

Conclusion	61
Bibliographie	63

Introduction

Ce rapport présente le travail effectué lors de mon stage de Master Recherche, dans le cadre de ma deuxième année de Master Web Intelligence, cohabilité par l'Université Jean Monnet de Saint-Étienne et l'École Nationale Supérieure des Mines de Saint-Étienne. Ce stage a été fait au sein de l'équipe Systèmes Multi-Agents du centre Génie Industriel et Informatique (G2I) de l'école des Mines de Saint-Étienne et a été encadré par Gauthier Picard.

Dans le cadre du groupe de travail "Self-organisation in Multi-Agent Systems" du réseau d'excellence européen AgentLink III et du groupe de travail COLLINE, un cas d'étude a été proposé afin de comparer diverses approches du problème face aux systèmes multi-agents auto-organiseurs. Il s'agit de planifier la gestion d'une production où la réalisation des pièces est soumise à diverses contraintes (délais de livraison, usinages, etc.). On devra prévoir les emplois du temps d'opérateurs, c'est-à-dire leur affectation sur les machines. Le cadre général du problème est dynamique, des perturbations imprévisibles pouvant surgir à tout moment (pannes, grèves, congés maladies, commandes de dernière minute, etc.). Le problème sera modélisé sous forme d'un réseau de contraintes distribuées. On opposera à l'approche classique, l'approche auto-organisationnelle d'un système multi-agent.

Les Systèmes Multi-Agents (SMA) permettent aujourd'hui d'aborder la résolution de problèmes de plus en plus complexes, dynamiques et ce de façon distribuée ; problèmes que des approches classiques peinent à résoudre efficacement. Le problème de la gestion de production semble donc être intéressant pour une étude comparative de cette approche face à certaines plus classiques. La formalisation utilisée, problème de satisfaction de contraintes, se prête bien au problème de la gestion de production et plus généralement à celle de la création d'emploi du temps. En outre, cette formalisation est utilisable assez simplement au sein de systèmes multi-agents et permet une résolution au moyen de divers algorithmes existants.

Tout d'abord nous verrons en détail en quoi consiste la gestion de production. Nous verrons plusieurs méthodes et approches utilisées, comme l'utilisation de systèmes multi-agents, pour résoudre ce type de problème. Nous présenterons aussi la plate-forme qui a été utilisée pour les implémentations.

Dans un second chapitre nous définirons la notion de système multi-agent et d'auto-organisation. En complément, deux algorithmes mettant en œuvre des systèmes multi-agents auto-organiseurs seront expliqués.

Un troisième chapitre, nous permettra de définir les problèmes de satisfaction de contraintes simples, distribués et dynamiques. Nous détaillerons et analyserons plusieurs algorithmes connus qui permettent de résoudre ce genre de problèmes.

La chapitre quatre présente en détail le scénario utilisé pour notre étude. Nous y verrons ses différentes caractéristiques ainsi que sa modélisation sous la forme d'un problème de satisfaction de contraintes. Une première

approche orientée agents sera présentée, AmasCOP.

Nous verrons avec les chapitres cinq et six, les approches qui ont été imaginées et implémentées pour résoudre notre problème. La partie cinq parlera des approches classiques adaptées, ABT et AWCS, alors que la partie six explicite l'approche multi-agent auto-organisatrice proposée.

Le dernier chapitre de ce rapport regroupe l'ensemble des expérimentations qui ont été effectuées ainsi que l'analyse et la comparaison de chacune d'elle.

Première partie

Analyse de l'existant

1

La gestion de production

1.1 Le contrôle manufacturier

Aujourd'hui la gestion d'entreprise devient de plus en plus difficile et particulièrement le domaine de la gestion de production pose de nombreux problèmes. Aussi appelée activité de planning et de contrôle de la production, cette gestion est définie par [Wu, 1994] :

Définition 1. *La planification et le contrôle de production sont l'application de méthodes scientifiques face à la gestion de production, où des matières et/ou d'autres éléments, fournis à un système industriel, sont assemblés et convertis d'une manière organisée afin d'y ajouter de la valeur conformément à la politique de production définie.*

La planification et le contrôle de production peuvent être décomposés suivant trois niveaux hiérarchiques :

- les activités stratégiques,
- les activités tactiques,
- les activités opérationnelles.

Le premier niveau, les activités stratégiques, regroupe tous les processus mis en œuvre lors de la conception d'un produit ainsi que la réflexion menée sur les caractéristiques principales du produit par rapport aux attentes des consommateurs.

Les activités tactiques regroupent les étapes de conception des plans de mise en production des produits définis par les activités stratégiques. Elles vont déterminer l'organisation globale de la chaîne de production, les machines à utiliser ainsi que les ressources nécessaires à la fabrication (matières premières, personnel, etc.).

Le niveau qui nous intéresse plus particulièrement est celui des activités opérationnelles ; il sera le support de notre comparaison des diverses approches étudiées. Cette étape est celle de la fabrication réelle du produit défini. On va ici s'occuper de gérer la chaîne de production en temps réel pour répondre aux demandes des autres niveaux ou clients. Cette activité est mise en œuvre par le contrôle de la production défini par [Duggan et Browne, 1991] :

Définition 2. *Le contrôle de la production regroupe les principes et techniques utilisés par la direction pour planifier à court terme, contrôler et évaluer les activités de production de l'entreprise.*

Le contrôle de la production peut lui aussi être décomposé en plusieurs sous-domaines qui sont :

- *La planification* qui est le domaine qui s'assure de l'utilisation optimale des ressources de production (machines, personnel, etc.). Il spécifie les prévisions des opérations de la chaîne de production afin d'assurer le

respect des dates de livraisons, des priorités et de la disponibilité des ressources. Il donne un programme temporel des opérations en temps réel, avec l'allocation des ressources correspondantes.

- *La distribution* qui est le domaine qui fait correspondre le programme défini par la planification aux ressources disponibles à un moment donné. Les principales perturbations sont gérées ici.
- *Le monitoring* qui observe l'état des ressources et s'assure de leur bon fonctionnement. Ce domaine permet a posteriori d'évaluer la performance ou la robustesse du système de production.
- *Le transport* qui s'occupe de l'acheminement des matières premières et produits au sein même de la chaîne de production, on y retrouve notamment la gestion de tapis roulants, robots, etc.
- *la production* est le domaine impliquant les éléments physiques du système tels que les machines, le personnel, les matières premières, etc.

Pour résumer, la gestion de production est le moyen de planifier l'utilisation des ressources pour la production en suivant des contraintes prédéfinies (délais de livraisons, emploi du temps des opérateurs, etc.). Tout cela doit se faire dans un environnement dynamique, soumis à l'apparition de perturbations imprévisibles (pannes de machines, grève des opérateurs, etc.), demandant une forte réactivité aux changements du système.

1.2 Les systèmes multi-agents pour la gestion de production

De nombreuses méthodes utilisant des systèmes multi-agents ont été développées pour résoudre des problèmes de gestion de production, certaines utilisant différents types d'agents, appelés holons et d'autres utilisant des phénomènes tels que la stigmergie. Nous détaillons ici deux approches mettant en scène chacune un de ces principes.

1.2.1 Système Holonique

L'architecture PROSA [Van Brussel *et al.*, 1998] définit un système inter-holon, identifie les types d'holons, leurs responsabilités et la façon dont ils inter-agissent. Un holon n'est en fait qu'un agent ordinaire (*cf. section 2.1.1.1*) mais qui peut être lui-même constitué d'un ensemble d'agents. L'architecture de base est composée de trois types d'holons :

- Les *product holons* s'occupent du processus et des connaissances associés à la fabrication du produit pour s'assurer d'une qualité de production suffisante. Ils fournissent aux autres holons une base de donnée sur le cycle de vie des produits, les besoins, le design, les processus et matières premières utiles.
- Les *resource holons* représentent les ressources de la production et les informations qui permettent de contrôler la ressource. Ils fournissent les capacités et fonctionnalités de la ressource, les méthodes de réservation et ont les moyens de contrôler les ressources (machines, convoyeurs, palettes, personnel, etc.).
- Les *order holons* représentent les commandes et sont en charge de la production d'une commande et vérifié qu'elle est correctement exécutée et ce dans les temps.

À partir de ces holons on construit un système constituant une holarchie (organisation regroupant différents holons). Les *products holons* connaissent toutes les informations nécessaires à la fabrication du produit. Ils déter-

minent par eux-mêmes comment le produit associé peut être fabriqué en utilisant les *resources holons* disponibles. Quand un *order holon* arrive dans le système il va d'abord chercher quels sont ses besoins via les *product holons* respectifs. Il va négocier avec les *resource holons* utiles, leur utilisation pour ses produits. En outre, l'*order holon* se préoccupe de l'allocation des ressources. Quand une opération débute, l'*order holon* laisse le *product holon* et le *resource holon* se communiquer les informations techniques nécessaires pour le déroulement de l'opération.

1.2.2 Stigmergie

La stigmergie est définie comme une méthode de communication indirecte où les individus communiquent entre eux en modifiant leur environnement. [Brueckner, 2000] propose un éco-système synthétique, dans lequel les agents se coordonnent en utilisant des traces de phéromones. Au-dessus de cette couche nécessaire à l'évolution des agents, nous trouvons deux systèmes : le système de contrôle et le système de guide. Le système de contrôle de production est distribué et réactif. Il se décompose en deux niveaux : réactif et interface. Le niveau réactif, cœur du système de gestion, contient cinq types d'agents :

- les *switch-agents* qui déterminent pour chaque *workpiece-agent* arrivant, le chemin à suivre ;
- les *loader-agents* qui représentent les entrées physiques du système et qui déterminent le prochain produit à introduire dans le système et lui associent un *workpiece-agent* ;
- les *processing-agents*, responsables des pièces d'un *workpiece-agent* utilisent des phéromones pour attirer à eux leurs pièces ;
- les *unloader-agents* correspondent aux sorties du systèmes et utilisent des phéromones pour en faire sortir les pièces finies.
- les *workpiece-agents*, responsables d'une pièce, il s'occupent de déterminer le chemin que doit suivre la pièce pour être traitée.

Le niveau d'interface forme le pont entre le système de contrôle et le système de guide. Son rôle est de récupérer les informations du système de contrôle. En plus, il étend le comportement des agents du niveau réactif afin d'intégrer les conseils provenant du système de guide. Le système de guide génère, implante et évalue les stratégies. Il tente d'influencer par le biais des phéromones le système de contrôle pour réaliser un modèle particulier.

Une autre solution basée sur le mécanisme de la stigmergie, est présentée dans [Karuna *et al.*, 2005]. Elle consiste à effectuer de la prévision émergente dans les systèmes de coordination et de contrôle. Des perturbations imprévisibles pouvant surgir à tout moment le système va chercher à prévoir le futur, les charges et emplois du temps des machines afin de pouvoir s'auto-organiser en cas de changement de l'organisation. Le système utilise plusieurs types d'agents : des agents ordres, ressources ou produits. Un agent ordre qui a pour but de trouver un moyen d'usiner un produit dans les délais, crée des agents explorateurs (fourmis). Ces fourmis vont explorer le réseau constitué par les ressources, y déposer des phéromones et ramener à leur agent ordre les informations sur les différents chemins possibles. Ce dernier va alors choisir le meilleur chemin et tenter de le réaliser. Pour cela il réserve des créneaux horaires auprès des ressources. De cette manière, grâce aux informations recueillies par les agents intentions et ce de manière émergente, il se construit une prévision de l'utilisation des ressources.

1.3 Le groupe COLLINE

Le groupe de travail COLLINE (COLLectif, INteraction, Emergence)¹ est un groupe de travail créé en 1993, directement rattaché à l'AFIA (Association Française d'Intelligence Artificielle)². Son principal thème de recherche est les systèmes complexes et plus précisément les systèmes multi-agents. Dans le détail, il y est effectué l'étude des théories, modèles et méthodes qui permettent l'obtention de phénomènes émergents (cf. 2.1.2.2); l'émergence étant vue ici comme le résultat d'un collectif (l'ensemble des agents), utilisant de l'auto-organisation (cf. 2.1.2) qui apporte une réponse ou solution au problème pour lequel le système a été conçu.

Pour son travail, le groupe COLLINE a choisi d'utiliser plusieurs études de cas, parmi toutes celles retenues, se trouve une adaptation de celle de P. Valckenaers³ traitant de la gestion d'atelier (cf. 4.1.1.1). Le choix de ce type de scénario peut s'expliquer au travers du fait que de plus en plus de systèmes de production industriels offrent des architectures distribuées, avec l'utilisation de machines communicantes, par réseau filaire ou aérien, et qui embarquent également de plus en plus de capacités de calcul. Ces caractéristiques sont intéressantes pour étudier facilement en milieu réel l'implémentation et le déploiement de systèmes multi-agents.

1.3.1 La plate-forme MASC

Suite au choix de la gestion d'atelier comme cas d'étude, une plate-forme java, MASC (cf. figure 1.1) pour *Multi-Agent for Supply Chaining*, a été développée par André Machonin (SMAC-IRIT). Cette plate-forme a pour but de faciliter l'implémentation des approches imaginées par les participants du groupe de travail COLLINE. MASC doit permettre de :

- implémenter de nouveaux algorithmes ;
- définir de nouveaux scénarios pour comparer les approches ;
- définir de nouvelles métriques et critères de comparaison.

La plate-forme, est constituée de trois parties :

- le *simulateur* qui prend un scénario, contenant toutes les informations relatives à son exécution, ensemble de conteneurs, d'opérateurs et de stations, ainsi que l'ensemble des perturbations possibles (pannes, grèves, etc.);
- le *scheduler* servant d'interface de communication entre le *simulateur* et le système de résolution développé par l'utilisateur. Il fournit les données du problème au *solveur* et récupère la solution. Il est codé par l'utilisateur ;
- le *solveur* de l'utilisateur est le cerveau de la plate-forme, il prend les données fournies par le *scheduler*, recherche une solution au problème formulé et renvoi une liste d'affectations au *simulateur* par le biais du *scheduler*.

1. <http://www.irit.fr/COLLINE/>

2. <http://www.afia-france.org/>

3. http://www.irit.fr/COLLINE/DOCUMENTS/Specification-Manufacturing_V1.pdf

La plate-forme fonctionne en mode tour par tour. À chaque pas elle fournit :

- les stations disponibles avec leurs qualifications,
- les opérateurs disponibles avec leurs qualifications,
- les conteneurs à usiner,
- les éventuelles perturbations survenues (pannes, indisponibilités, etc.).

L'algorithme 1 donne le fonctionnement global de la plate-forme. En premier lieu, on construit les éléments pour la simulation (conteneurs, opérateurs et stations). Ensuite, on fournit ces données au *scheduler* qui lance la résolution. Enfin on récupère la solution trouvée et la plate-forme simule l'exécution de cette solution.

Algorithme 1 : Principe général de MASC

Construire les opérateurs ;

Construire les stations ;

Construire les containers ;

tant que un conteneur est disponible faire

Scheduler.executeStep(StationsDisponibles, OpérateursDisponibles, ContainersDisponibles) ;

Exécuter la solution trouvée ;

fin

MASC fournit un simulateur de scénarios, mais aussi un éditeur de scénarios. Le simulateur gère les affectations faites des conteneurs, opérateurs et stations, et permet d'avoir un déroulement visuel des différents temps d'exécution du scénario utilisé.

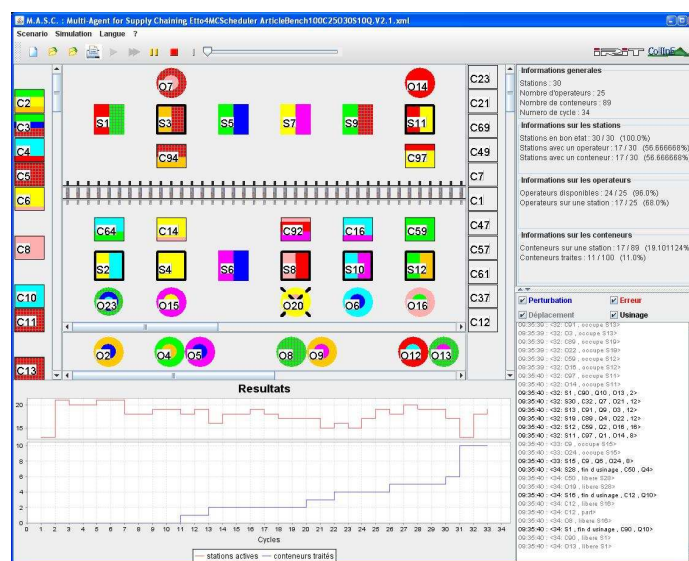


FIGURE 1.1 – Plate-forme MASC

1.4 Bilan

La gestion de production devient aujourd'hui un problème de plus en plus présent au sein des entreprises. La complexification des chaînes de production et de l'ensemble de la conception des produits confronte les décideurs à des problèmes toujours plus importants et difficiles à résoudre.

Devant ce constat de nombreuses approches et méthodes ont été imaginées pour palier à ces problèmes. Nous avons ainsi cité des architectures telles que PROSA (*cf. 1.2.1*) qui fournissent un modèle général, utilisant des agents pour répondre à une problématique touchant à la gestion d'une chaîne de production. Différentes voies ont été explorées et ont toutes donné des résultats très disparates.

Dans ce contexte, le groupe COLLINE, a décidé de prendre un problème de gestion de production, comme base pour une étude sur la décentralisation et l'auto-organisation dans les systèmes multi-agents. Une plate-forme a été développée afin de faciliter le développement d'approches multi-agents pour le scénario de gestion de production défini. L'intérêt des systèmes multi-agents pour résoudre ce type de problème réside dans leur grande adaptabilité et leur grande souplesse leur permettant de résister plus facilement aux perturbations du système, par rapport aux approches mettant en action de la recherche opérationnelle qui sont plus rigides.

2

2 Systèmes multi-agents et auto-organisation

2.1 Concepts

Dans le chapitre précédent nous avons commencer à parler d'agents et de système multi-agent. Nous détaillons dans ce chapitre en détail ce que veulent dire ces termes et en quoi ils consistent. De plus nous donnons des exemples précis de leur utilisation pour résoudre deux types de problèmes différents.

2.1.1 Systèmes Multi-Agents (SMA)

Depuis les années 80, les systèmes multi-agents ne cessent de se développer et d'étendre leur domaine d'application. Malgré toutes ces années d'existence, aucune définition, que ce soit aussi bien pour le terme d'agent que de système multi-agent, n'a encore fait l'unanimité au sein de la communauté. [Wooldridge et Jennings, 1995] tentent néanmoins de définir un système multi-agent comme suit :

Définition 3. *Un système multi-agent (SMA) est un ensemble d'agents évoluant au sein d'un même environnement. Il fournit des moyens de communication aux agents, qui interagissent afin d'effectuer les tâches globales du système.*

2.1.1.1 Agent

Une définition possible pour le terme agent est donnée par [Ferber, 1995] :

Définition 4. *Un agent possède des :*

- *compétences : il a, pour atteindre ses objectifs, une panoplie d'actions possibles ;*
- *croyances : il a une vue de son environnement et des autres agents ;*
- *accointances : il a des agents avec qui communiquer ;*
- *aptitudes : il a les capacités de percevoir et d'accomplir ses actions.*

Cette définition simple peut néanmoins être complétée et précisée par celle de [Wooldridge et Jennings, 1995] :

Définition 5. *Un agent est un système informatique situé dans un environnement qui est :*

- *autonome, il agit de lui-même sans ordres extérieurs ;*
- *sociable, il peut communiquer avec d'autres agents ;*

- réactif, il perçoit son environnement et réagit à ses changements ;
- proactif, il ne fait pas que réagir à l'environnement, il peut décider seul d'agir pour atteindre ses objectifs.

2.1.1.2 Environnement

Nous avons vu que les agents constituant un SMA sont des entités situées dans un environnement. L'environnement peut être de deux types différents.

Le premier est dit l'*environnement social* de l'agent, c'est-à-dire l'ensemble des agents avec lesquels il peut communiquer. Celui-ci peut être fixe ou non, ceci dépendant directement du choix effectué lors du codage de l'agent. Un agent peut dès le départ, avoir un ensemble de relations sociales définies avec certains autres agents, mais il peut également en cours de vie, rencontrer de nouveaux agents ou en oublier d'autres.

Le second environnement possible est l'*environnement physique*, qui représente les possibilités de déplacement physique de l'agent (ex. déplacement sur une grille représentant un ensemble de machines).

2.1.1.3 Communication

Les agents du système sont des entités autonomes cependant, il leur est souvent nécessaire de communiquer avec d'autres agents afin de parvenir à satisfaire certains de leurs objectifs. Il existe deux types de communications possibles dans les systèmes multi-agents : directe et indirecte.

La première possibilité, la *communication directe*, s'opère généralement au moyen de messages que les agents s'échangent directement. Un agent peut donc envoyer un message à un autre agent ou alors envoyer un message à tous les autres (*broadcast*). Ce mode de communication présente l'avantage de cibler directement les destinataires et de n'envoyer le message qu'à ceux-ci. En revanche, il implique de connaître le nom ou l'adresse du ou des destinataire(s) ce qui peut être gênant dans certains cas.

Le second mode de communication est la *communication indirecte*. On va avoir dans ce cas l'utilisation d'un vecteur de communication commun à tous les agents, l'environnement. Un agent va tout simplement marquer ou modifier l'environnement physique à un endroit précis et ainsi y laisser un message. Ici, il n'est donc pas nécessaire de connaître le destinataire du message, néanmoins il faut préciser que tous les agents doivent obligatoirement évoluer au sein du même environnement physique. Enfin, l'assurance que l'agent concerné par le message le reçoive (ou perçoive) n'est pas garantie.

2.1.2 Auto-organisation dans les systèmes multi-agents

Les SMA offrent une modélisation distribuée, sur plusieurs agents, intéressante d'un point de vue de l'approche à la dynamique. En effet, ne pas avoir de contrôle centralisé laisse la possibilité au système de s'adapter beaucoup plus facilement aux changements de l'environnement. Il est évidemment plus facile de modifier une partie des agents que l'ensemble de ceux-ci, mais cela demande aussi de nombreux efforts en terme de coordination. Ce phénomène d'adaptation peut-être défini comme de l'auto-organisation [Di Marzo Serugendo *et al.*, 2006], lorsque l'initiative vient des agents eux-mêmes :

Définition 6. *L'auto-organisation au sein d'un SMA est le fait qu'un système multi-agent change spontanément son organisation consécutivement à des changements locaux au sein de celui-ci.*

En complément de cette définition minimaliste, on peut citer deux types de systèmes auto-organiseurs :

- les systèmes auto-organiseurs *forts*, qui se réorganisent sans aucun contrôle explicite, interne, externe ou centralisé ;
- les systèmes auto-organiseurs *faibles*, qui se réorganisent en suivant un contrôle interne et centralisé, ou planifié.

2.1.2.1 Coopération

Le caractère auto-organiseur d'un système ne peut être obtenu qu'au moyen de mécanismes particuliers lui permettant de déterminer les actions à effectuer pour trouver la nouvelle organisation. Parmi ces critères on pourrait citer l'utilisation de phéromones (fourmis), le regroupement (oiseaux), le renforcement (utilisation de mécanismes de réaction/diffusion ou de champs de potentiels) [Di Marzo Serugendo *et al.*, 2006]. Pour notre étude, nous nous intéresserons à un autre critère possible, la *coopération*. Une définition simpliste de celle-ci peut être énoncée par :

Définition 7. *La coopération dans un système implique que tous les agents du système cherchent à travailler dans l'intérêt général de tous.*

On voit avec cette définition que ceux qui vont être les acteurs directs de la réorganisation du système sont les agents.

Définition 8. *Un agent est dit coopératif s'il vérifie $c_{per} \wedge c_{dec} \wedge c_{act}$:*

- c_{per} : les signaux (ou messages) sont compris sans ambiguïté.
- c_{dec} : les informations reçues peuvent être utilisées par l'agent pour raisonner.
- c_{act} : l'action de l'agent est profitable aux autres agents et à l'environnement.

Lors d'un changement dans le système, certains agents vont devoir agir afin de chercher à trouver une nouvelle organisation adaptée pour le système. Seuls ceux se trouvant face à des situations qu'ils jugeront non coopératives agiront.

Définition 9. *Un agent se trouve dans une situation non coopérative (SNC) si et seulement si $\neg c_{per} \vee \neg c_{dec} \vee \neg c_{act}$ est vraie.*

Un exemple de coopération est donné dans [Picard et Glize, 2006]. Cet article présente une notion de coopération d'un point de vue social. Elle est exprimée comme un compromis entre l'altruisme, situation dans laquelle un agent va préférer aider les autres à atteindre leur but au détriment du sien, et l'égoïsme, où l'agent privilégie ses propres objectifs sans se soucier des autres. La coopération est par définition, difficile à appréhender et à formaliser dans la mesure où l'évaluation des impacts d'une action par rapport à une autre, sous-entend de maîtriser parfaitement les conséquences de celle-ci. L'extraction des paramètres pertinents à prendre en compte est une étape

importante et souvent délicate ; même une fois tous ces paramètres identifiés ,encore faut-il savoir quelles sont leurs relations, les priorités de chacun et comment les relier. Toute cette réflexion ne peut être, de façon générale, menée que sur le problème concerné directement par l'approche multi-agent auto-organisatrice développée, les critères étant spécifiques à chaque problématique.

2.1.2.2 Émergence

Nous l'avons vu, l'auto-organisation au sein de systèmes multi-agents est une propriété non négligeable pour s'attaquer à des problèmes dynamiques. Il est aussi intéressant de remarquer qu'un phénomène tout aussi important peut en résulter : l'*émergence*, définie par [Di Marzo Serugendo *et al.*, 2006] comme suit :

Définition 10. *Un phénomène est dit émergent si les propriétés suivantes sont observées :*

- l'observation d'un phénomène ostensible (qui s'impose à l'observateur) au niveau global ou macro,
- la nouveauté radicale du phénomène (il n'est pas observé au niveau micro et n'est pas prévisible),
- la cohérence et la corrélation du phénomène (il a une identité propre mais liée fortement aux parties qui le produisent),
- l'observation d'une dynamique particulière (le phénomène n'est pas prédonné, il y a "auto-maintien" du phénomène).

Cette notion est intéressante puisqu'elle implique que les agents ne doivent pas obligatoirement connaître la fonction globale du système, mais seulement leurs propres objectifs. La fonctionnalité du SMA n'est a priori pas connue des agents et émerge de l'organisation et des interactions locales de ces derniers qui effectuent chacun des traitements simples et locaux. Il est alors beaucoup plus simple de définir le comportement des agents. [Gleizes, 2004] présente la théorie des AMAS (Adaptive Multi-Agent Systems), qui s'appuie sur l'auto-organisation basée sur la coopération pour des environnements dynamiques, ayant pour principal but l'obtention d'un phénomène émergent pour le système.

2.2 Exemples d'approches auto-organisatrices par coopération

2.2.1 Affectation de fréquences (FAPP)

Un autre problème est abordé dans [Picard *et al.*, 2007] ; l'affectation de fréquences avec polarisation. Ce problème consiste en un réseau hertzien de télécommunication, composé de récepteurs et d'émetteurs. Des contraintes modélisant les phénomènes d'interférences sont à prendre en compte.

Le choix effectué ici est de modéliser le trajet des communications par les agents. La difficulté de l'agent, critère de choix pour la coopération, est composée de quatre éléments :

- l'amélioration possible ;
- les possibilités ;
- le nombre de contraintes violées ;
- l'ancienneté de la dernière modification connue.

Lors de la comparaison des difficultés entre agents, certains de ces critères seront utilisés. Prenons en exemple le cas de deux agents devant déterminer parmi eux celui le plus en difficulté. Dans un premier temps, ils vont comparer la valeur associée à leur amélioration possible. Si ces valeurs sont différentes, ils peuvent alors directement élire un agent, sinon ils comparent la valeur de leurs possibilités et ainsi de suite. Si des agents sont équivalents, avec des valeurs égales sur tous les critères, un des deux est choisi aléatoirement.

L'agent élu démarre une séance de médiation avec les agents avec lesquels il est en relation. Durant cette session, l'élue présente l'ensemble de ses valeurs possibles. En réponse, il reçoit de la part des agents en médiation, pour chacune de ses possibilités, une valeur représentant le nombre de contraintes violées, si le médiateur la choisit. Ce dernier prend donc la valeur qui maximise le nombre de contraintes satisfaites pour lui et ses voisins.

Pour résumer, à chaque moment, un agent est élu suivant les valeurs permettant d'évaluer sa difficulté. Il lance une médiation et détermine la valeur qu'il doit prendre pour maximiser le nombre de contraintes satisfaites pour lui et ses voisins.

Cette approche présente une SMA auto-organisateur utilisant des agents coopératifs. Les communications permettent de déterminer les agents actifs et une session de médiation permet quant à elle, d'évaluer l'impact des choix afin de maximiser la qualité de la solution globale. Il faut remarquer que les critères utilisés pour le choix d'un agent sont des critères assez génériques qui peuvent être réutilisés dans d'autres problèmes et notamment celui qui nous intéresse ; la gestion de production.

2.2.2 Emergent TimeTabling Organization (ETTO)

La création d'emploi du temps, souvent difficile, du fait de nombreuses contraintes, est abordée au moyen d'un SMA auto-organisateur, faisant intervenir deux types d'agents dans [Picard *et al.*, 2005].

Des agents appelés *RepresentativeAgents (RA)* représentent les acteurs humains (professeurs et groupes d'étudiants) et connaissent les contraintes associées (disponibilité, équipement, etc.). Ces agents créent d'autres agents qui sont des *BookingAgents (BA)* qui vont effectuer la recherche. Tous les *BA* sont reliés à leur seul *RA* créateur et connaissent donc les contraintes qui leur sont liées. En plus de ces contraintes initiales, les *BA* devront gérer des ajouts et suppressions de contraintes au cours de la recherche, du fait de nouveaux partenariats, par exemple. Les *BA* se déplacent sur une grille modélisant les salles et créneaux horaires. Ils y recherchent des cases pouvant satisfaire leurs contraintes (nombre de places de la salle adéquat, créneaux horaires libres, etc.). Si une case semble être convenable, un *BA* va tenter de la réserver et chercher un *BA* partenaire (de type différent, professeur pour étudiants et inversement) pour cette case.

Il a été identifié cinq cas non coopératifs qui vont déclencher des actions chez le *BA* :

- un partenaire est incompatible ;
- une réservation incompatible ;
- un conflit de partenaire ;
- une salle déjà réservée ;
- une réservation inutile.

Cette approche montre une capacité de résolution de problèmes à grande échelle et semble être assez robuste

face à la dynamique, même si des améliorations peuvent encore être apportées, comme la détection d'une solution globale.

On voit ici différents agents se déplaçant sur une grille qui cherchent à réserver des ressources communes. Deux *BA* doivent réserver conjointement une case (créer un partenariat) pour satisfaire leurs contraintes. La gestion de production, qui nous concerne, pourrait être vue suivant le même principe, où les cases seraient les machines et les *BA*, un conteneur ou un opérateur. C'est une modélisation du problème qui pourrait être étudiée.

2.3 Bilan

Les systèmes multi-agents présentent un ensemble de caractéristiques qui sont, à première vue, intéressantes pour s'attaquer à la résolution de problèmes distribués. Nous pouvons ainsi citer la distribution des calculs, la décentralisation de la décision, l'auto-organisation ou encore l'obtention de phénomènes émergents.

L'auto-organisation au sein des SMA, ne signifie pas que les agents changent leur rôle au sein du SMA suite aux changements de celui-ci, mais elle permet d'avoir des agents qui ont des perceptions locales, qui agissent en fonction de leurs connaissances, pour obtenir de ce fait un contrôle décentralisé de tout le système.

L'utilisation d'agents pour la résolution de problèmes, comme le FAPP (*cf. section 2.2.1*), permet de diminuer considérablement la complexité de la mise en place de méthodes de résolution, en modularisant le problème et en le répartissant sur différentes entités qui résolvent les problèmes localement par des communications point-à-point.

3

Satisfaction de contraintes distribuées et dynamiques

3.1 Problème de satisfaction de contraintes (CSP)

Beaucoup de problèmes, comme l'allocation de ressources par exemple, peuvent être vus comme une affectation de valeurs sur un ensemble de variables liées entre elles par un ensemble de contraintes, *i.e.* problème de satisfaction de contraintes (CSP) dont nous donnerons une définition.

3.1.1 Formalisation

Définition 11. *Un problème de satisfaction de contraintes (CSP) consiste en un triplet (X, D, C) :*

- *Un ensemble fini de variables, $X = \{X_1, X_2, \dots, X_n\}$.*
- *Un ensemble domaine, contenant le domaine fini et discret de chaque variable, $D = \{D_1, D_2, \dots, D_n\}, \forall i \in [1, n], X_i \in D_i$.*
- *Un ensemble de contraintes, $C = \{C(R_1), C(R_2), \dots, C(R_m)\}$, où chaque R_i est un sous-ensemble ordonné de variables, et chaque contrainte $C(R_i)$ est un ensemble de tuples indiquant les valeurs consistantes entre les variables de R_i .*

Définition 12. *Une solution S pour un CSP est une affectation de toutes ces variables, telle que cette affectation satisfasse toutes les contraintes du CSP. On a donc :*

- *S est un ensemble ordonné, $S = \langle v_1, v_2, \dots, v_n \rangle, S \in D_1 \times D_2 \times \dots \times D_n$.*
- *$(\forall j \in [1, m]) (\exists S' \subseteq S) \wedge (S' \in C(R_j))$ est vrai. S' est un ensemble ordonné.*

3.1.2 Résolution

De nombreuses méthodes combinatoires ont été développées pour résoudre les problèmes de satisfaction de contraintes. Nous en présentons quelques unes, verrons très simplement leur principe général de fonctionnement et identifierons leurs principaux avantages et inconvénients. Les algorithmes décrits feront aussi bien intervenir de la recherche locale ou amélioration itérative que des méthodes constructives.

Amélioration Itérative

GSAT [Selman *et al.*, 1992] tente de trouver une solution en changeant itérativement la valeur d'une seule variable à la fois, en minimisant le nombre de contraintes violées. Le nombre global d'itérations est limité. *Breakout* [Morris, 1993] pondère chaque contrainte. On part d'un assignement (aléatoire), si on ne se trouve pas dans un minimum local (affectation complète avec aucune possibilité d'amélioration), on change la valeur d'une variable qui diminue le coût total (des contraintes violées), à défaut on augmente le poids des contraintes violées. Ces deux algorithmes utilisent l'amélioration itérative et le nombre d'itérations autorisées est limité, remarquons également qu'aucun d'eux n'est complet¹. De plus, à tout moment, ils fournissent un assignement qui est une affectation complète mais qui n'est pas forcément une solution pour le problème.

Backtracking (BT)

L'algorithme *Backtracking* fournit une recherche systématique. Le principe est de partir d'une affectation aléatoire et de construire une solution. On a à tout moment une solution partielle ; pour chaque variable on essaie de trouver une valeur consistante avec la solution partielle construite. Si on trouve une valeur consistante, on passe à la variable suivante, à défaut on revient à la variable précédente, on recherche une nouvelle valeur pour celle-ci et on garde en tant que nouvelle contrainte la solution partielle trouvée. L'heuristique *Min-Conflicts* [Minton *et al.*, 1994] peut être combinée au *Backtracking* afin d'augmenter son efficacité, Cette heuristique permet de choisir les variables à affecter en évaluant les conséquences de celle-ci par rapport à l'ensemble des contraintes. Cet algorithme est complet, mais a, dans le pire des cas, un temps d'exécution exponentiel, ce qui n'est pas acceptable pour de larges problèmes.

Weak-Commitment Search (WCS)

[Yokoo, 2001] présente l'algorithme *Weak-Commitment Search (WCS)* qui s'appuie sur le *Backtracking avec Min-Conflicts*. La seule différence réside dans le fait qu'au lieu de revenir à la dernière variable assignée lorsque l'on ne peut pas trouver de solution, on va chercher à réassigner toutes les variables, en gardant la solution partielle comme nouvelle contrainte. Cette petite modification par rapport au *Backtracking* le rend plus efficace. Lorsqu'une mauvaise solution partielle est construite, le *Backtracking* peine à en sortir puisqu'obligé de mener une recherche exhaustive, ce qui n'est pas le cas ici ; toute la solution étant abandonnée.

Il existe d'autres méthodes métaheuristiques permettant la résolution de CSP et l'optimisation combinatoire, que nous ne détaillerons pas. Nous pourrions ainsi citer des algorithmes fournis, du recuit simulé ou encore une recherche locale utilisant une liste tabou [Dréo et Siarry, 2003].

1. trouve toujours une solution s'il en existe une, ou se termine toujours s'il n'y en a aucune

3.2 CSP distribué

Il est souvent intéressant de répartir les variables sur différents agents qui vont uniquement s'occuper de gérer la ou les variables qui leur sont attribuées, permettant une résolution distribuée et parallèle. C'est un CSP distribué, que nous allons présenter.

3.2.1 Formalisation

Définition 13. Un CSP distribué est un CSP dans lequel les variables et contraintes sont distribuées sur plusieurs agents. Un CSP distribué est un quintuplet (X, D, C, A, ϕ) où X, D et C correspondent à ceux d'un CSP classique :

- $A = \{1, \dots, p\}$ est un ensemble d'agents,
- $\phi : X \rightarrow A$ est une fonction qui fait correspondre chaque variable à son agent.

3.2.2 Résolution

Pour la résolution de CSP distribués, les algorithmes qui seront présentés, admettent les hypothèses de travail formulées par [Yokoo, 2001] :

- les agents communiquent par message et ne peuvent envoyer de message qu'aux seuls agents dont ils connaissent l'adresse/identifiant,
- le délai de réception d'un message est aléatoire et fini,
- les messages sont reçus dans l'ordre dans lequel ils sont envoyés.

La résolution dans un milieu distribué permet une parallélisation, donc un possible gain de temps, mais implique obligatoirement de mettre en place des mécanismes de communication, afin d'assurer la cohérence du système durant la résolution. Un agent doit connaître les valeurs de ces voisins pour pouvoir agir correctement.

Distributed Breakout Algorithm (DBA)

Le *Distributed Breakout Algorithm*, adaptation de l'algorithme *Breakout*, est détaillé dans [Yokoo, 2001]. L'idée est de détecter les minimums locaux et de s'en échapper. La distribution des contraintes rend la détection d'un minimum local impossible. On va donc chercher à éviter des "quasi-minimums locaux". Chaque agent connaît ses contraintes et leur poids, ainsi à chaque tour, il calcule son évaluation (somme de ses contraintes violées) et son amélioration possible et envoie ces informations à ses voisins. Lorsqu'un agent a reçu toutes les données de ses voisins il peut alors détecter s'il se trouve dans un "quasi-minimum local". Dans ce cas, seul l'agent avec la plus grande possibilité d'amélioration agit. Cet algorithme, bien qu'incomplet, possède tout de même un avantage, il est capable de détecter une solution globale, soit sa terminaison. Contrairement à d'autres algorithmes présentés, ce mécanisme est directement inclus dans ce dernier.

Asynchronous Backtracking (ABT)

[Yokoo, 2001] introduit le *Asynchronous Backtracking*. Les agents y sont ordonnés et possèdent des connexions entrantes et sortantes. L'ordre est déterminé avec différentes méthodes : taille du domaine, nombre de contraintes,

etc. Un lien entre deux agents représente une contrainte entre les variables respectives des agents. Un agent choisit une valeur et la communique aux agents qui le suivent par un message *ok ?*. Ceux-ci vont en connaissant la valeur de tous leurs prédécesseurs tenter de se trouver une valeur. S'ils y parviennent, ils réitèrent le même processus, sinon ils envoient un message *nogood* à l'agent qui les précède directement dans l'ordre pour lui indiquer qu'avec sa valeur aucune possibilité ne leur est laissée. La réception d'un message *nogood* provoque une nouvelle recherche de valeur. Lorsqu'un *nogood* ne peut être résolu par un agent, il peut demander l'ajoute de liens vers des agents qui empêchent la résolution. *ABT* est complet et correct², néanmoins sur de larges problèmes, comme on l'a déjà vu, il a un temps d'exécution exponentiel malgré la parallélisation du calcul. Un autre point négatif réside dans le fait que les agents ont besoin de suivre un ordre total ce qui peut être contraignant dans des environnements dynamiques, où un agent peut apparaître et disparaître du système bouleversant du coup l'ordre définit.

Asynchronous Weak-Commitment (AWCS)

[Yokoo, 2001] explique l'algorithme *Asynchronous Weak-Commitment Search* qui est une extension du *WCS*. Une priorité est attachée à chaque agent (initiale à 0) et changée dynamiquement lors de la recherche. Comme pour le *ABT*, on retrouve l'utilisation de messages *ok ?* et *nogood*. Lors de la recherche, on cherche à minimiser le nombre de contraintes violées avec les agents de priorité moindre. Si pour un agent aucune valeur n'est possible, il devient alors le plus prioritaire et lance une nouvelle recherche de solution. Là encore on retrouve un ordre pour les agents mais qui n'est pas, comme dans *ABT*, statique. Il change dynamiquement en cours d'exécution, grâce à la mise à jour des priorités. Cet aspect semble être intéressant pour un passage sur des problèmes dynamiques.

Environment, Reactive rules, Agents (ERA)

L'approche *Environment, Reactive rules, Agents* est présenté par [Liu *et al.*, 2002]. Dans cet algorithme, l'environnement va être utilisé comme vecteur de communication (ce mode de communication est analogue à de la stigmergie où les phéromones ne souffriraient pas d'un phénomène d'évaporation). L'environnement regroupe l'ensemble des valeurs du domaine des agents, ceux-ci ne pouvant se déplacer qu'à l'intérieur de leur propre domaine. Chaque agent, lorsqu'il va choisir une position, va incrémenter la valeur des cases interdites pour les autres agents. Ces valeurs donnent le coût d'une case, (*i.e.* le nombre total de contraintes violées si un agent choisit cette case). Il existe trois mouvements, *least-move*, *better-move* et *random-move*, chacun associé à une probabilité. Lors d'un choix, un tirage détermine l'action à effectuer. L'action *least-move* choisit la case de coût minimal. L'action *better-move* tire une case au hasard, si son évaluation est meilleure on s'y place sinon on ne bouge pas. Enfin *random-move* permet de sortir des minimums locaux dans lesquels il n'existe pas de meilleure case. On accepte de choisir une case au hasard, quitte à dégrader la solution courante. Lorsque tous les agents sont sur une case ayant pour évaluation 0, on a trouvé une solution. Pour la première fois, nous voyons ici un algorithme faisant directement intervenir l'environnement. Utilisé pour la communication, il permet de s'abstraire de tous les mécanismes

2. pour toute instance du problème il se termine et produit une sortie correcte.

de la communication directe entre agents (connaissance des identifiants ou adresses, etc.) et surtout des temps de transmissions des messages qui sont par hypothèse finis mais aléatoires. Cependant, les agents se voient chargés de maintenir l'environnement à jour. Ensuite, cette approche permet d'obtenir rapidement des solutions partielles, ce qui peut être intéressant dans le cadre de problèmes dynamiques nécessitant une grande réactivité. La recherche de solutions partielles ne s'applique pas directement aux CSP mais plutôt aux COP (problèmes d'optimisation de contraintes), qui sont des problèmes où l'on cherche à trouver les meilleures solutions partielles à des problèmes souvent sur-contraints.

Asynchronous Partial Overlay (APO)

Une autre approche s'appuyant beaucoup plus sur la coopération a été imaginée par [Mailler et Lesser, 2004], *Asynchronous Partial Overlay*. Les agents y ont une priorité (nombre de voisins) et coopèrent au cours de séances de médiation. Lorsqu'un agent ne peut pas trouver de valeur consistante avec les agents qu'il connaît plus prioritaires que lui, il lance une séance de médiation, sinon il change sa valeur et la transmet à ses voisins. Lors d'une séance de médiation, les agents contactés qui acceptent, font part au médiateur, pour chaque valeur de leur domaine, des agents conflictuels. Le médiateur va alors choisir une affectation pour chaque agent en médiation et leur en faire part. Il va aussi se connecter avec les agents qui ne sont pas ses voisins mais pour lesquels il sait que les valeurs choisies vont entraîner des violations. La résolution est ici décentralisée ; malgré cela il faut noter que certains agents, les médiateurs, centralisent la résolution de sous-problèmes. L'ajout de liens possibles par ces agents, atténue la décentralisation dans le sens où un agent peut dès lors se connecter à tous les autres agents et de ce fait, centraliser la résolution globale du système.

Asynchronous Distributed OPTimization (ADOPT)

Asynchronous Distributed OPTimization [Modi *et al.*, 2006] a pour principale application, l'optimisation sous contraintes distribuée. Chaque contrainte est associée à un coût. Il va s'agir pour chaque agent de minimiser la fonction objectif globale (somme des coûts des contraintes). Un pré-traitement est effectué pour avoir une organisation des agents sous forme d'arbre ainsi on en déduit la priorité des agents, la racine étant celui le plus prioritaire. Chaque agent choisit une valeur minimisant le coût total des contraintes violées. On a une recherche au niveau des fils bornée, avec un fonctionnement analogue à celui d'un *Branch and Bound*. Cet algorithme fournit une détection de terminaison intégrée et une amélioration de la recherche en terme de temps grâce à l'utilisation de bornes.

3.3 CSP dynamique

Le caractère dynamique de la plupart des problèmes tirés de la réalité, nous oblige, pour pouvoir les modéliser et éventuellement les résoudre, d'adapter les modélisations utilisées, les méthodes et algorithmes de résolution connus.

3.3.1 Formalisation

Définition 14. Un CSP dynamique P est une séquence P_0, P_1, \dots, P_i de CSP statiques, où chacun d'entre eux résulte d'un changement dans le précédent. Soit P_t un CSP statique au temps t , on peut alors écrire un CSP dynamique sous la forme :

- $DP = \{P_0, P_1, \dots, P_t, P_{t+1}, \dots\}$ où P_{t+1} est un problème obtenu à partir de P_t .

Cette définition est adaptée de [Dechter et Dechter, 1988]. Un changement dans un CSP P à un moment donné t n'est en fait qu'un ajout ou une suppression de contrainte(s) ou variable(s).

3.3.2 Résolution

Concrètement un changement dans un CSP, ajout ou suppression de contrainte par exemple ne provoque pour l'organisation des agents que des changements au niveau des liens qu'ils maintiennent entre eux. Concernant l'ajout et la suppression de variable, nous n'avons là aussi affaire qu'à de nouvelles contraintes entre l'agent introduit ou supprimé et les agents en relation avec lui.

Dynamic Distributed Breakout

L'adaptation pour la dynamique du *Distributed Breakout* est explicitée dans [Roger, 2005]. et ajoute deux nouvelles procédures, *addConstraint* et *removeConstraint*. En plus de la liste des voisins de chaque agents, on introduit une nouvelle liste. Elle doit prévenir le phénomène du *deadlock* entre des agents qui seraient dans l'attente de messages de types différents. Lors d'un ajout de contrainte, *addConstraint* place les nouveaux voisins dans la liste de l'agent concerné et envoie un message afin de connecter les nouveaux voisins. Pour une suppression de contrainte, on ne fait que supprimer de cette liste, de celle des voisins, de la vue de l'agent et de la liste des messages en attente, les voisins qui ne sont plus connectés. Cette modification minimale rend l'entrée et la sortie d'agents dans le SMA très facile sans pour autant changer radicalement la méthode de résolution.

Dynamic Asynchronous Weak-Commitment (DynAWCS)

[Hattori et Ito, 2006] présente *Dynamic Asynchronous Weak-Commitment*, une adaptation de l'algorithme AWCS. Il y est question d'optimiser le temps de recherche d'une solution lors d'une modification. Lorsqu'un nouvel agent est connecté, soit une contrainte ajoutée, le processus de recherche lancé sera limité. En effet, seuls les agents directement liés à l'agent déclencheur seront activés. Par la suite, une recherche de type AWCS est lancée et peut être réinitialisée par un agent actif à tout moment, si celui-ci ne trouve pas de solution. Le nombre de redémarrages autorisés est passé en paramètre. On a donc une recherche limitée dans le temps, ainsi que dans l'espace des agents. *DynAWCS* permet de ne pas avoir de trop grosses perturbations de la solution courante lors de modifications dans les contraintes. L'ajout d'un agent ne déclenche qu'une recherche locale permettant de trouver rapidement une solution proche de la solution courante. Le critère d'arrêt qu'est le nombre d'itérations autorisées, est un moyen de limiter le temps de recherche et d'obtenir des solutions proches, si le temps le permet, voire partielles.

3.4 Bilan

De nombreux travaux ont été menés dans le domaine des problèmes de satisfaction de contraintes (CSP). Cette modélisation a été adaptée pour donner des CSP distribués, introduisant dans la résolution l'utilisation d'agents. Les agents résolvent le problème en cherchant à satisfaire les contraintes définies par le système. Plusieurs algorithmes ont été imaginés pour la résolution de ce type de problèmes notamment par Makoto Yokoo dans [Yokoo *et al.*, 1998]. Une grande contribution a aussi apportée par Rina Dechter avec [Dechter, 2003].

La modélisation sous forme de contraintes est intéressante pour s'attaquer à des problèmes très divers. Dans notre cas nous utiliserons une modélisation par CSP, pour résoudre notre problème de gestion de production. L'utilisation d'une telle modélisation est motivée par ses avantages qui nous permettent de méthodes de résolution déjà existantes pour pouvoir comparer l'approche SMA proposée. Il faut aussi noter que notre étude de cas, présente des caractéristiques telles que la dynamique ou de nombreuses contraintes, nous laissant penser que la modélisation CSP est adaptée.

Deuxième partie

**Comparaison et analyse de diverses
approches**

4

Gestion de production : étude de cas

4.1 Présentation

Comme il a été dit dans la section 1.3, le cas d'étude qui sera utilisé pour ce travail de recherche est un cahier des charges adapté de celui défini par P. Valckenaers. Ce cas a été choisi par le groupe COLLINE et sera donc utilisé pour le travail qui sera mené sur les différentes approches mises en place durant ce stage, celui-ci étant directement lié à ce thème de recherche.

4.1.1 Description

Le cahier des charges initial définit l'organisation globale d'une chaîne de production d'une usine. Il comporte différents points et met en action plusieurs systèmes plus ou moins indépendants, comme la production ou encore le transport des pièces au sein même de l'usine. Quelques modifications ont été apportées par rapport à ce cahier des charges, que nous détaillons ici.

4.1.1.1 Cahier des charges original

Le cahier des charges a pour but de présenter le fonctionnement d'une chaîne de production d'une usine. Des produits semi-finis ont besoin d'être envoyés à une date de livraison précise pour l'assemblage dans un autre département. Les pièces sont transportées dans des conteneurs qui généralement en contiennent environ une dizaine même si cela peut varier. La chaîne fournit un ensemble d'espaces réservés pour le stockage, dispersés dans le département de production. La chaîne comprend aussi dix machines avec des propriétés et capacités différentes. Les stations de travail ont chacune deux ou trois emplacements pouvant accueillir un conteneur. Un opérateur prend les pièces d'un conteneur, les usine de la première à la dernière et les place dans un autre conteneur de la station. Un système automatique de transport, sur rail, transporte les conteneurs, et peut déplacer deux conteneurs à la fois, à un moment donné. De plus, les opérateurs peuvent eux-mêmes déplacer sans conteneur des pièces. Les pièces arrivent dans le système dans un conteneur. La chaîne de production usine différents produits ce qui impose des temps de paramétrages pour les machines. Le système de transport a en général une capacité suffisante mais peut être de temps en temps surchargé. Les opérateurs ont une ou plusieurs qualifications.

Les perturbations possibles imaginées sont, par exemple, des pannes machines, des échecs d'usinages, des changements dans les dates de livraison, de nouvelles commandes prioritaires ou non, des grèves ou des congés maladie d'opérateurs, des remplacements d'opérateurs, etc.

4.1.1.2 Cahier des charges modifié

Par rapport au cahier des charges initial quelques modifications ont été apportées afin de simplifier les premières approches développées pour le problème de la gestion d'atelier, et surtout afin de pouvoir coller avec les spécifications de la plate-forme qui ne prend en compte que certaines caractéristiques du problème initial.

Tout d'abord, la partie de gestion du transport des conteneurs au sein même du système ne sera pas prise en compte, nous considérerons que les conteneurs peuvent librement circuler sur la chaîne et atteindre n'importe quelle station à tout moment. Il ne sera donc pas pris en compte des problèmes de surcharge de réseau de transport, les conteneurs pouvant tous circuler en même temps. De plus le fait que les opérateurs peuvent transporter eux-mêmes des pièces n'est pas considéré.

Ensuite nous partirons du principe qu'une machine ne peut accueillir qu'un unique conteneur à la fois à un moment donné. Le conteneur accueilli est celui qui amène les pièces à usiner sur la station mais également celui qui les récupère une fois usinées et les transporte sur une autre machine au besoin.

Nous voyons que le scénario utilisé est réduit à la création des emplois du temps des stations, opérateurs et conteneurs. Nous devons considérer les stations et leurs qualifications, les opérateurs et leurs qualifications, les conteneurs avec leurs qualifications et leur date de livraison (déterminé par celle des pièces qu'il contient). Nous devons également envisager certaines perturbations telles qu'une panne pour les stations, une grève ou maladie pour les opérateurs, mais aussi concernant les conteneurs, l'arrivée de nouveaux conteneurs plus prioritaires par rapport à ceux déjà présents dans le système.

4.1.2 Formalisation

Le problème à modéliser est constitué de plusieurs variables, de stations, d'opérateurs et de conteneurs, qui possèdent certaines qualifications. On va chercher à un moment donné t à affecter les opérateurs et conteneurs sur les stations fournissant les qualifications communes à eux trois.

Soit $M = \{m_1, \dots, m_n\}$ l'ensemble des stations. Nous avons les variables objectifs suivantes :

- $\forall i \in [1, n], o_i^t$ est l'opérateur affecté à la station m_i au temps t ,
- $\forall i \in [1, n], c_i^t$ est le container affecté à la station m_i au temps t .

La plate-forme MASC comme nous l'avons vu en section 1.1 fonctionne en tour par tour. Nous considérerons donc un CSP à chaque temps t d'exécution :

- $\mathcal{X} = \mathcal{X}_o^t \cup \mathcal{X}_c^t$
où $\mathcal{X}_o^t = \{o_i^t | i \in [1, n]\}$ et $\mathcal{X}_c^t = \{c_i^t | i \in [1, n]\}$,
- $\mathcal{D} = \{D(o_1), \dots, D(o_m), D(c_1), \dots, D(c_p)\}$
où m (resp. p) est le nombre d'opérateurs (resp. containers) fournis en entrée au temps t ,
- $\mathcal{C} = \mathcal{C}_{\text{diff}} \cup \mathcal{C}_{\text{qual}} \cup \mathcal{C}_{\text{time}}$
où $\mathcal{C}_{\text{diff}} = \{\text{alldiff}(\mathcal{X}_o^t), \text{alldiff}(\mathcal{X}_c^t)\}$, $\mathcal{C}_{\text{qual}} = \{(q(o_i^t) = q(c_i^t) = q(m_i)) | i \in [1, n]\}$ et $\mathcal{C}_{\text{time}} = \{(\text{end}(c_i) < t_{\text{max}}^i) | i \in [1, p]\}$.

\mathcal{X} est ici composé de $2n$ variables à affecter. En effet, chaque affectation est un triplet $\langle m, o, c \rangle$, où une station accueille un opérateur et un conteneur.

On a : $\forall i, D(o_i) \subset [1, n]$ et $D(c_i) \subset [1, n]$. Ceci signifie que les opérateurs ne peuvent être affectés qu'à certaines stations, et que les conteneurs se doivent de passer par certaines stations : ce sont les *qualifications* des opérateurs et des conteneurs qui permettent de définir ces affectations.

Les contraintes C_{qual} expriment que le triplet $\langle m, o, c \rangle$, représentant une affectation possible, doit regrouper trois éléments qui présentent au moins une qualification commune.

Les contraintes C_{diff} signifient qu'un opérateur ou un conteneur ne peut être affecté sur deux stations à un même temps t .

Les contraintes C_{time} indiquent qu'un conteneur doit être usiné avant un temps limite t_{max} .

4.1.3 Caractéristiques

Au-delà des contraintes définies précédemment, on peut encore identifier d'autres caractéristiques propres à la modélisation elle-même d'un point de vue de l'agentification ou de l'implémentation :

- Les contraintes C_{diff} , de notre CSP distribué, vont nous obliger lors du choix de l'agentification à prendre en compte un réseau de contraintes complet. Le réseau constitué des variables verra toutes les variables d'un même type être reliées entre elles du fait d'une contrainte binaire de type $\forall i, j, o_i \neq o_j$ et $\forall i, j, c_i \neq c_j$.
- Les contraintes C_{diff} ne sont applicables qu'à des variables affectées (e.g. s'il y a plus de stations que nécessaire, elles ne trouveront jamais de solution).
- Les contraintes temporelles C_{time} ne spécifient que des dates de fin ($\text{end}(c_i)$), même si des dates de début pourraient être attendues dans un tel cas. Ceci s'explique par la dimension temps réel de la plate-forme qui fournit les commandes à chaque temps t .

4.2 Une première approche orientée agents : AmasCOP

[Capera *et al.*, 2006] nous présente une méthode de résolution utilisant une modélisation par agent du cas d'étude détaillé précédemment.

Le but est d'usiner un maximum de pièces en un temps minimum. Le problème est modélisé sous la forme d'un CSP distribué. Néanmoins, il a été remarqué que le problème était sur-contraint (aucune solution possible pour l'ensemble des contraintes à tout moment), il a alors été ramené à un problème d'optimisation de contraintes distribué (DCOP).

L'algorithme utilisé, AmasCOP, choisit de représenter les conteneurs par les agents. L'idée est d'avoir des agents coopératifs et donc de privilégier les conteneurs comportant les pièces les plus difficiles à usiner. On calcule d'abord le degré de non coopération des conteneurs (leur difficulté) pour ensuite en déterminer celui des stations et opérateurs. On choisit les trois éléments les moins coopératifs et le triplet $\langle \text{Station}, \text{Opérateur}, \text{Conteneur} \rangle$ sélectionné devient l'affectation à utiliser.

Algorithme 2 : Évaluer le degré de non coopération d'un conteneur

```

Conteneur.valeurNC ← 0.0;
Conteneur.listeValeurNC ← ∅;
forall opérations op nécessaires du conteneur do
    marge ← tempsRestant * nbStations * nbOpérateurs;
    margeOpération ← tempsRestantPourOpération(op) * nbStationsCompatibles(op) *
    nbOpérateursCompatibles(op);
    degréOpération ← marge/margeOpération;
    Conteneur.listeValeurNC.add(degréOpération);
end
Conteneur.valeurNC ← maxConteneur.listeValeurNC;
    
```

$$NC = \frac{\text{tempsRestant} * \text{nbStations} * \text{nbOpérateurs}}{\text{tempsRestantPourOpération}(op) * \text{nbStationsCompatibles}(op) * \text{nbOpérateursCompatibles}(op)}$$

FIGURE 4.1 – Valeur de Non Coopération

La mesure de non coopération d'un conteneur, donné par l'algorithme 2, est fonction de la difficulté à effectuer les opérations d'usinage que le conteneur doit réaliser. Pour cela, chaque opération est évaluée sur la base d'un rapport entre les seules combinaisons compatibles avec l'usinage considéré et la totalité de l'espace de travail. Par espace de travail, on entend le temps restant de la simulation, les stations et opérateurs disponibles, au contraire de l'espace compatible qui regroupe seulement les stations et opérateurs disponibles et compatibles. Une fois la valeur de non coopération de chaque tâche indépendamment calculée, on garde la valeur maximale pour degré du conteneur.

Algorithme 3 : Évaluer le degré de non coopération d'une station

```

forall conteneurs c disponibles do
    if c est compatible then
        Station.listeValeurNC.add(c.valeurNC);
    end
end
Station.valeurNC ← maxStation.listeValeurNC;
    
```

Les algorithmes 3 et 4 nous montrent comment sont calculés les degrés de non coopération, respectivement des stations et des opérateurs. On voit que ces deux calculs sont effectués avec pour base la valeur calculée pour chaque conteneur. La station ou opérateur aura pour degré, le degré maximum de tous les conteneurs compatibles.

Il faut remarquer que cet algorithme effectue une résolution centralisée, supposant une connaissance de tous les éléments, amenant à une résolution déterministe. Dans le cadre de notre étude et de notre objectif, la centralisation est une chose que nous voulons éviter. En effet, une résolution centralisée n'est pas adaptée à l'esprit des systèmes multi-agents qui eux, sont beaucoup plus adaptés pour décentraliser la résolution du problème. Il est aussi normal de penser que nous obtiendrions une moins bonne résistance face à une grande dynamique.

Algorithme 4 : Évaluer le degré de non coopération d'un opérateur

```

forall conteneurs c disponibles do
  | if c est compatible then
  |   Opérateur.listeValeurNC.add(c.valeurNC);
  | end
end
Opérateur.valeurNC ← maxOpérateur.listeValeurNC;

```

4.3 Bilan

Comme nous l'avons vu dans le chapitre 1, il existe des approches multi-agents, pour le type de scénario détaillé précédemment dans ce chapitre. Nous pouvons noter que l'utilisation de deux types d'agentifications sont possibles pour notre problème.

La première est une agentification des variables, qui fait intervenir des agents qui ont pour rôle de gérer l'affectation de la ou les variable(s) qui leurs sont associées. Cette agentification permet de diviser le problème initial en sous-problèmes et de répartir sur différents acteurs la résolution de ceux-ci. Les algorithmes utilisant cette modélisation offrent en général de bons résultats mais ne sont pas très robustes face à la forte dynamique de certains problèmes.

La seconde possibilité est une agentification des entités du domaine. C'est dans ce cas que l'on voit l'utilisation de différents types d'agents au sein même du système. Les agents utilisés ont des caractéristiques différentes, des buts différents et des comportements différents. La fonction globale du système n'est pas un objectif à proprement dit pour les agents eux-même, mais elle émerge de l'interaction de ceux-ci. C'est le type d'agentification qui est utilisée par exemple dans l'architecture PROSA (*cf.* 1.2.1) ou celle de Hadeli Karuna [Karuna *et al.*, 2005]. Cette agentification implique de mettre en place des mécanismes permettant aux agents de se comprendre et de communiquer malgré leurs différences.

Les algorithmes développés par la suite seront l'occasion d'étudier les différences entre ces deux agentifications possibles. ABT et AWCS utiliseront une agentification des variables alors que l'approche par système multi-agent auto-organisateur utilisera une agentification des entités du domaine.

5

ABT et AWCS pour la gestion de production

5.1 Présentation

La base utilisée pour le développement des algorithmes ABT et AWCS est décrite dans [Yokoo *et al.*, 1998]. Il a été décidé l'implémentation de ABT et AWCS, comme approches classiques pour la comparaison, car ces algorithmes présentent quelques propriétés intéressantes pour notre étude.

Tout d'abord ce sont des algorithmes complets et corrects, c'est-à-dire qu'ils sont capables d'explorer l'ensemble des solutions possibles afin de trouver toutes les solutions. Cela va nous permettre de pouvoir éventuellement évaluer la capacité de l'approche multi-agent développée par la suite, à trouver des solutions complètes lorsqu'il en existe.

Ensuite il faut souligner que ces algorithmes sont assez simples à mettre en place et présentent le grand intérêt de pouvoir mettre en œuvre de nombreuses heuristiques qui peuvent améliorer leurs performances (*cf. MinConflicts* [Minton *et al.*, 1994]).

5.1.1 Principe

Les deux algorithmes expliqués par la suite sont très proches l'un de l'autre, en effet AWCS n'est ni plus ni moins qu'une amélioration directe de ABT que l'on détaillera. Nous expliquerons ici le principe de ces deux algorithmes.

5.1.1.1 Asynchronous Backtracking

Avant le lancement de l'algorithme, tous les agents sont ordonnés suivant un ordre total, déterminé avec différentes méthodes, taille du domaine, nombre de contraintes, etc. Lors de l'exécution de ABT les agents du système, une fois initialisés, ne font que réagir aux messages qu'ils reçoivent. Ils attendent de recevoir un message, le traitent et envoient éventuellement d'autres messages et se remettent à nouveau en attente. Ce cycle de vie est exécuté en permanence jusqu'à la réception d'un message signalant la fin de la résolution. L'algorithme général suivi par les agents qui exécutent ABT est l'algorithme 5.

Algorithme 5 : Algorithme général de ABT

```

Choisir une valeur aléatoirement ;
EnvoiOk ?();
tant que pas de message de fin faire
    ReceptionneMessageOk ?(X, D);
    ReceptionneMessageNoGood(X, NoGood);
fin
    
```

Dans cet algorithme on voit que deux fonctions de communication sont utilisées pour la réception des messages, ReceptionneMessageOk ? et ReceptionneMessageNoGood.

Algorithme 6 : Fonction traitant les messages Ok ? - ReceptionneMessageOk ?(X,V)

```

Ajouter (X, V) à la vue ;
VérifierSatisfaction();
    
```

Cette fonction capte les messages Ok ? en provenance des agents qui se trouvent avant, dans l'ordre défini et traite ce message. Le message Ok ? permet à un agent de prendre connaissance de la valeur V de l'agent X expéditeur et donc de compléter ou mettre à jour sa vue (sa représentation des autres agents). Une fois la valeur enregistrée, il peut alors vérifier que cette nouvelle valeur ne rend pas sa propre valeur inconsistente avec sa vue, en utilisant la fonction VérifierSatisfaction().

Algorithme 7 : Fonction assurant la satisfaction des contraintes ABT - VerifierConsistence()

```

if (Vue  $\cup$  Valeur) non consistant then
    if aucune valeur possible then
        | Backtrack();
    end
    Choisir une valeur  $d$  consistente ;
    Valeur  $\leftarrow d$ ;
    EnvoiOk ?();
end
    
```

Lorsqu'un agent doit vérifier si sa valeur est possible, il commence par regarder si elle n'est pas en conflit avec la vue qu'il a. Si ce n'est pas le cas, il tente de la modifier. Il regarde si une autre valeur est possible pour lui, auquel cas il la prend et en informe les agents suivants, sinon il exécute la fonction Backtrack().

L'algorithme 8 montre ce que fait un agent lorsqu'il ne peut pas trouver de valeur consistente. Il identifie le premier agent plus prioritaire que lui avec lequel il est en conflit et lui envoie un message NoGood. Le message NoGood échangé contient la valeur posant problème et le contexte de celle-ci (valeur des agents précédents), l'émetteur efface cette valeur de sa vue et en recherche une nouvelle. Lors de la recherche des conflits avec des agents plus prioritaires, si un ensemble vide est trouvé lors du parcours de la vue, alors cela signifie qu'il n'existe pas de solution et l'agent non satisfait termine la résolution en faisant un broadcast d'un message de fin. Si la vue de l'agent est vide, il est à la racine.

Algorithme 8 : Fonction déclenchant le backtracking ABT - Backtrack()

```

nogoods ← {Vc | Vc = valeur des agents en conflit };
if {} ⊂ nogoods then
  | Broadcast pas de solution ;
  | Fin de l'algorithme ;
end
forall Vc ∈ nogoods do
  | Sélectionner l'agent X le moins prioritaire de V ;
  | EnvoyerNoGood(X, V) à X ;
  | Supprimer valeur de X de la vue ;
end
VérifierSatisfaction();

```

Algorithme 9 : Fonction ReceptionneNoGood

```

Ajouter NoGood à liste des nogoods ;
forall Z ∈ NoGood non connecté do
  | DemandeConnexion(Z)
  | Ajoute valeur de Z à la vue
end
val ← valeur courante
VerifierlaConsistence()
if val = valeur courante then
  | EnvoiOk ?(X)
end

```

Lorsqu'un message NoGood est reçu par un agent, il l'enregistre s'il n'est pas déjà connu ; il mémorise donc la valeur interdite et son contexte. Si dans le contexte se trouvent des valeurs d'agents inconnus, l'agent va chercher à se connecter à eux pour connaître leur valeur, pour de futur résolution. Ce branchement s'opère au moyen de l'envoi d'un message de connection qui provoque en retour la réception d'un message Ok ? de la part de l'agent interrogé.

Les messages qui sont envoyés dans le système sont principalement de deux types, Ok ? et NoGood. Il faut noter qu'ils ont des directions opposées, les messages Ok ? ne circulent que d'un agent vers les agents moins prioritaires que lui et les messages NoGood ne vont que d'un agent vers un agent plus prioritaire.

5.1.1.2 Asynchronous Weak-Commitment Search

Lorsqu'un backtracking est effectué dans ABT, il n'a pour effet que de revenir une variable en arrière et de demander à en changer la valeur. Il est tout de suite visible que lorsque l'on se trouvera dans un minimum local (toutes les variables affectées mais il existe des violations de contraintes), il sera bien plus long d'en sortir en remettant en question toutes les variables une par une plutôt que de repartir d'une affectation vide. Ce constat est la base même de AWCS, qui lui, ne revient pas d'un pas en arrière mais va essayer de revoir la valeur de toutes les

variables précédentes.

Le principe général de l'algorithme reste inchangé, mais on aura un changement d'ordre dynamique sur les agents durant la résolution. La première étape reste inchangée, les agents sont ordonnés et se voient associer une priorité. Les messages, en plus de transmettre les informations évoquées pour ABT, informeront aussi le récepteur de la priorité (ou ordre) de l'émetteur. Il faut aussi remarquer que les connexions qui relient les agents entre eux ne sont plus distinguées entre les agents plus prioritaires et moins prioritaires. Lorsqu'un message Ok ? est envoyé, il l'est à tous les voisins, plus ou moins prioritaires. Le message NoGood, lui, est adressé à un seul voisin.

Algorithme 10 : Fonction assurant la satisfaction des contraintes AWCS - VerifierConsistence()

```

if (Vue  $\cup$  Valeur) non consistant then
    if aucune valeur possible then
        | Backtrack();
    end
    Choisir une valeur d consistente avec les agents de plus haute priorité et qui
    minimise le nombre de contraintes violées avec les agents moins prioritaires ;
    Valeur  $\leftarrow$  d ;
    EnvoiOk ?();
end

```

On voit que l'agent ne cherche qu'à satisfaire les contraintes communes avec des agents plus prioritaires et simplement à minimiser le nombre de violations avec les agents moins prioritaires. Cela permet un plus grand degré de liberté dans les choix des valeurs et diminue le nombre de backtracking.

Algorithme 11 : Fonction déclenchant le backtracking AWCS - Backtrack()

```

nogoods  $\leftarrow$  {Vc | Vc = valeur des agents en conflit } ;
if {}  $\subset$  nogoods then
    | Broadcast pas de solution ;
    | Fin de l'algorithme ;
end
if  $\nexists v \in$  nogoods | v  $\in$  nogoodSent then
    forall V  $\in$  nogoods do
        | Ajouter V à nogoodSent ;
        forall X  $\in$  V do
            | EnvoyerNoGood(X, V) à X ;
        end
    end
    pmax  $\leftarrow$  priorité max de la vue ;
    priorité  $\leftarrow$  1 + pmax ;
    Choisir une valeur d qui minimise le nombre de contraintes violées avec les agents moins prioritaires ;
    Valeur  $\leftarrow$  d ;
    EnvoiOk ?();
end

```

La fonction de backtracking adaptée, présentée par l’algorithme 11, explicite plusieurs mécanismes améliorant l’algorithme. Premier point, l’agent maintient à jour une liste de NoGood envoyés (nogoodSent), ce qui permet de ne pas envoyer une deuxième fois un NoGood. Deuxième point, on remarque aussi que l’agent change sa priorité, il identifie l’agent le plus prioritaire, en parcourant sa vue, et prend sa priorité incrémentée de 1. Une fois devenu le plus prioritaire il va alors prendre une valeur qui minimise le nombre de contraintes violées avec les agents inférieurs (tous puisqu’il est passé en tête de l’ordre), et relance une recherche par un message Ok ?.

5.1.2 Modélisation

Les deux algorithmes abordés ici sont très proches, on utilisera donc une modélisation commune. Nous expliquons ici les différents choix qui ont été effectués concernant le type de modélisations et les implications de celles-ci sur les agents et sur les relations qu’ils entretiennent.

5.1.2.1 Agentification

Comme nous l’avons vu dans la section 4.3, il peut être envisagé deux types d’agentifications. Celle qui sera utilisée ici est l’agentification des variables.

Dans des algorithmes comme ABT et AWCS, les agents sont tous de même type et ont tous les mêmes objectifs, satisfaire toutes les contraintes dans lesquelles interviennent la variable dont ils sont responsables. Notre problème présente trois entités différentes, des stations, des opérateurs et des conteneurs, qui n’ont a priori pas toutes le même but. Un conteneur dans le système cherche à usiner ses pièces donc à se placer sur une machine, il n’a, a priori, pas l’objectif de trouver lui-même un opérateur. Un opérateur lui va essayer de se placer sur une station pour travailler, il ne va pas directement rechercher un conteneur à usiner. La station, à la différence des deux autres entités, va avoir besoin de deux éléments, un conteneur et un opérateur, pour atteindre son objectif qui est d’effectuer des usinages.

En étudiant les deux agentifications possibles évoquées dans la section 4.3, on se rend tout de suite compte que celle la plus adaptée pour notre vision, est l’agentification des variables. Elle nous permet de n’avoir qu’un unique type d’agent, représentant les stations. Un agent est responsable de deux variables, un conteneur et un opérateur, et a pour but de créer des affectations de la forme $\langle Station, Opérateur, Conteneur \rangle$.

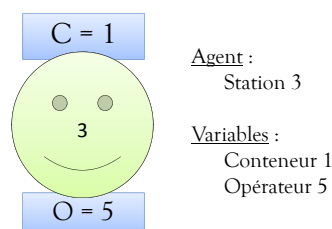


FIGURE 5.1 – Agent ABT / AWCS

5.1.2.2 Réseau de contraintes

Une fois les agents définis, il faut aussi étudier quels seront les liens les unissant. Les agents étant responsables des variables, conteneur et opérateur, vont entretenir des liens pour assurer la satisfaction de certaines contraintes. Lors de la formalisation du problème (cf. section 4.1.2) nous avons identifié trois types de contraintes, C_{diff} , C_{qual} et C_{time} .

Les contraintes C_{diff} , qui impliquent qu'un conteneur ou opérateur ne soit que sur une seule station à la fois, créent, dans notre réseau de contraintes, un lien entre chaque agent. En effet, les agents étant les stations, ils vont devoir s'assurer qu'aucun d'entre eux n'a choisit une valeur commune à un deuxième agent.

Les contraintes C_{qual} , qui s'assurent qu'une affectation regroupe bien trois entités ayant une qualification commune, sont gérées directement par l'agent lui-même et ne provoqueront pas l'ajout de lien entre agents du réseau. C'est lors de la recherche d'une valeur que l'agent devra prendre en compte ces contraintes et s'assurer de la cohérence entre ses qualifications, celles du conteneur et de l'opérateur choisi.

Les contraintes de type C_{time} , quant à elles, ne sont pas directement prises en compte. La plate-forme MASC, comme nous l'avons vu en section 1.1, fonctionne en mode tour par tour, fournissant donc à un instant t , un CSP à résoudre. Les conteneurs ayant encore du temps pour être usinés sont compris dans le problème et ceux qui ne peuvent plus l'être sont écartés. Les agents ne vont donc pas chercher à prendre en compte cette contrainte directement, ceux-ci cherchant, à chaque exécution, à occuper toutes les machines.

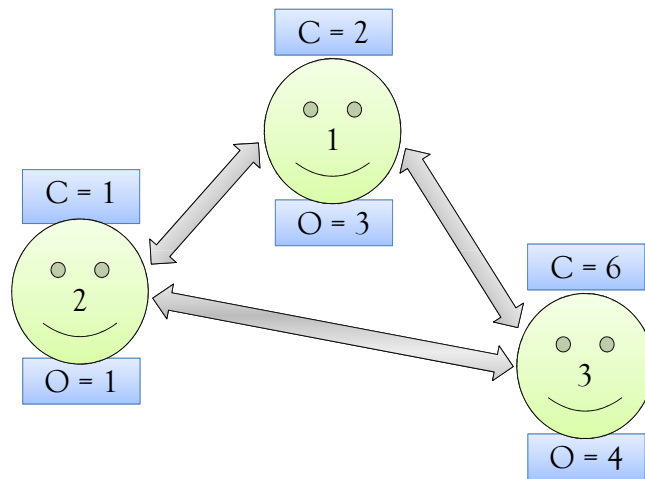


FIGURE 5.2 – Réseau de contraintes ABT / AWCS

5.1.3 Implémentation

L'implémentation des algorithmes ABT et AWCS pour le simulateur a été faite en écrivant principalement deux classes java qui sont, le Scheduler du simulateur, la classe qui récupère les données du problèmes et lance la résolution et une classe AgentABT ou AgentAWCS qui contiennent le cœur de la résolution ainsi que les fonctions détaillées dans les parties précédentes. La classe définissant un agent est directement dérivée de la classe Agent

utilisée dans la plate-forme JADE.

5.1.3.1 Pré-traitements

Un certain nombre de pré-traitements ont été mis en place afin d'effectuer un pré-filtrage sur les données passées aux agents. La création des arguments transmis aux agents se fait en respectant le principe suivant :

Algorithme 12 : Création des arguments des agents

```

Créer domaine opérateurs DO ;
Créer domaine conteneurs DC ;
if  $DO = \emptyset$  Ou  $DC = \emptyset$  then
| Annulation de la création de l'agent ;
else
| Créer association opérateur/conteneur compatible DOC ;
| Création de l'agent ;
end

```

En premier lieu on construit l'ensemble domaine pour la variable opérateur, pour cela on parcourt l'ensemble des opérateurs fournis par le simulateur et on ne sélectionne que ceux qui ont au moins une qualification commune avec la station (ou agent). Ensuite, nous exécutons la même opération pour la variable conteneur et les conteneurs du problème. On vérifie ensuite qu'aucune de ces deux listes n'est vide, afin de ne pas introduire dans le système un agent qui ne pourrait de toute façon pas trouver de valeur pour ses variables. Si un des domaines est un ensemble vide, on annule la création de l'agent, la station ne pouvant participer à aucune affectation. À partir de ces deux domaines on peut déterminer les associations possibles entre opérateurs et conteneurs. Une table de hachage est créée dans laquelle on associe les opérateurs (clef de la table) à une liste de conteneurs possibles (valeur de la table). Ces traitements sur les domaines des variables nous permettent de s'assurer du respect des contraintes C_{qual} , une station ne pouvant choisir que des opérateurs ou conteneurs compatibles entre eux et avec elle.

Le dernier élément vérifié avant la création du système multi-agent est le nombre d'agents entrant dans la résolution. Une fois tous les arguments des agents construits, on peut alors déterminer combien d'agents peuvent participer. S'il est possible de lancer au moins deux agents, le SMA est créé et la résolution lancée, sinon s'il n'y a qu'un seul agent une création directe d'affectation est faite à partir des domaines constitués.

5.1.3.2 Communication

La communication entre agents durant la recherche est entièrement effectuée au moyen de messages qui sont soit de type Ok ?, soit de type NoGood. On peut néanmoins signaler une petite adaptation de ces messages pour coller à la modélisation choisie.

Un agent est responsable de deux variables, conteneur et opérateur, ce qui implique que pour les contraintes de différence, il va devoir s'assurer qu'aucun agent n'a les mêmes valeurs. Il va, par des message Ok ?, informer ses voisins de ses deux valeurs à la fois. Si un agent est en conflit avec une ou même deux de ces valeurs et qu'il ne peut pas en changer, il va devoir envoyer un message NoGood. Le NoGood envoyé devra alors comporter, au-delà

de la valeur interdite et de son contexte, le type de la ou les variable(s) en conflit. On voit donc que deux listes de NoGood, distinctes et complémentaires, sont maintenues par un agent, une par variable.

Il existe donc en fait trois types de NoGood qui sont :

- NoGood Conteneur, indiquant un conflit avec la variable conteneur,
- NoGood Opérateur, indiquant un conflit avec la variable opérateur,
- NoGood Conteneur et Opérateur, indiquant un conflit avec les deux variables.

Dans le cas des deux premiers types, l'agent récepteur détermine le type du NoGood et tente dans un premier temps de ne changer que la variable en conflit et si ce n'est pas possible va ensuite tenter de changer ses deux valeurs.

5.1.3.3 Terminaison

ABT et AWCS sont deux algorithmes qui s'arrêtent s'il n'existe pas de solution au problème, mais si une solution est trouvée ils ne s'arrêtent pas automatiquement. Les agents n'ayant qu'une vue locale du problème, sauf le dernier agent de l'ordre qui a une vue sur l'ensemble des valeurs des autres agent, ne peuvent pas déterminer si une solution a été trouvée. Concernant le dernier agent, même s'il a une vue de la solution globale il ne peut pas être sûr que les informations dont il dispose sont l'exacte représentation des valeurs des agents à un moment donné t . Les messages du système étant asynchrones, il ne peut pas être certain qu'aucun message ne va plus lui parvenir.

L'algorithme implémenté pour la détection de la terminaison a pour base celui décrit dans [Dijkstra et Scholten, 1980]. On utilise un jeton qui circule dans le système et qui passe par tous les agents. Le premier agent dans l'ordre est celui qui détectera la terminaison.

Algorithme 13 : Réception du jeton

```
Jeton.valeur = Jeton.valeur + Compteur.valeur ;  
if Jeton.couleur = Blanc Et Agent.couleur = Noir then  
|   Jeton.couleur = Noir ;  
end  
EnvoiJeton() ;
```

L'ensemble des agents sont ordonnés suivant une liste prédéfinie et forment un cycle. Le jeton va parcourir ce cycle, passer par tous les agents et revenir au premier. Le jeton peut prendre deux couleurs, noir ou blanc, et porter en plus un chiffre, correspondant au nombre de messages encore dans le système. Chaque agent garde en mémoire un compteur de messages, qui est incrémenté lorsqu'un message est envoyé et décrémenté lorsqu'un message est reçu. Quand un agent reçoit le jeton il va ajouter la valeur de son compteur de messages à celui du jeton. De plus, une couleur est aussi associée à chaque agent, noir ou blanc. L'envoi d'un message Ok ? passe l'état de l'agent à blanc et l'envoi d'un message NoGood à noir. Envoyer un message Ok ? a pour signification que l'agent a trouvé une valeur et en informe les autres agents, alors qu'un message NoGood signifie que l'agent n'a pas de valeur satisfaisante possible. Les couleurs associées à ces messages reflètent donc l'état de l'agent, satisfait ou non. Lors de l'envoi du jeton si l'agent est noir, il le colore en noir, sinon il ne change pas sa couleur.

L'exemple de la figure 5.3 nous place dans le cas où une résolution vient d'être lancée, le premier agent (du haut) a donc émit deux messages Ok ?. Le second (à gauche) a envoyé un message Ok ? puis réceptionné un message Ok ? et ensuite envoyé un message NoGood. Enfin le dernier a reçu deux Ok ?. Le premier agent introduit un jeton blanc dans le cercle, portant le nombre des messages envoyés, donc 2 (figure 5.3 (a)). L'agent suivant réceptionne le jeton et l'incrémente de 1 (deux messages envoyés et un reçu), étant dans l'état noir il change également la couleur du jeton et l'envoi. Le dernier agent reçoit le jeton et le décrémente de 2 (deux messages reçus), il ne change pas la couleur étant lui-même blanc et le fait suivre. Le premier reçoit le jeton et vérifie sa valeur et sa couleur, comme le jeton n'est pas blanc (un agent dans le cycle n'est pas satisfait) et sa valeur n'est pas nulle (des messages sont encore dans le système), il relance un jeton blanc dans le système avec pour valeur 2 (il a reçu le message NoGood et envoyé deux messages Ok ?) (5.3 (b)). Le deuxième traite le jeton et ne change pas sa valeur (reçu un Ok ? et envoi d'un Ok ?) ni sa couleur. Le troisième agent ayant eu deux messages Ok ?, décrémente la valeur du jeton de deux et le fait passer. Enfin l'initiateur reçoit un jeton blanc, ayant pour valeur zéro, il termine donc la résolution.

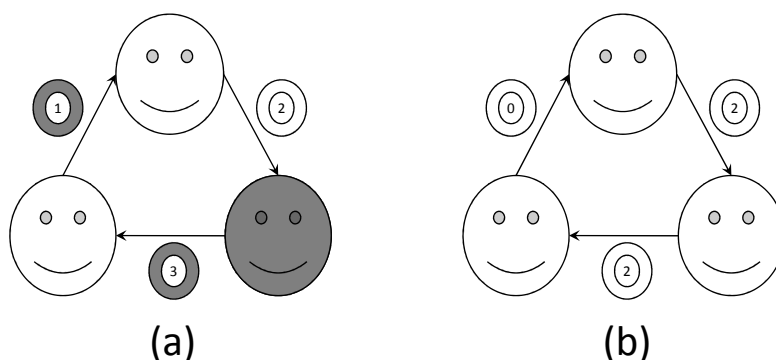


FIGURE 5.3 – Circulation du jeton pour la terminaison

5.1.4 Heuristique

Les algorithmes ABT et AWCS n'embarquent d'origine aucune heuristique pouvant en améliorer l'efficacité. Ils existent quelques éléments initiaux qui peuvent être traités et sur lesquels on peut influencer pour tenter de rendre ces algorithmes plus efficaces. Parmi ces paramètres on peut citer, l'ordre initial des agents ou l'ordre des domaines des variables. Concernant l'ordre, nous pouvons, lors de la création des agents, identifier, par exemple, le nombre de contraintes entretenues par l'agent avec d'autres, la taille du domaine pour ses variables. Un agent ayant beaucoup de contraintes sera plus facile à satisfaire s'il se trouve en haut de l'ordre, car il aura moins d'agents au dessus de lui qui vont limiter ses possibilités. L'ordre des domaines pour les variables peut avoir une influence

sur la rapidité à trouver des valeurs correctes. Si tous les agents possèdent beaucoup de valeurs communes, elles poseront plus de problèmes que des valeurs qui ne sont propres qu'à un unique agent.

Pour la première version des algorithmes développés, aucune heuristique spéciale n'a été mise en place. Concernant le nombre de contraintes par agent, nous l'avons vu en section 5.1.2.2, le réseau de contraintes est complet, chaque agent entretient donc des liens (ou contraintes) avec l'ensemble des autres agents. On voit que l'on ne peut pas se baser sur le nombre de contraintes concernant les agents pour les ordonner. L'ordre utilisé dans notre cas est celui de l'ordre d'apparition des stations dans le système. Lorsqu'une station est introduite dans le système, à sa création lui est attribué un numéro. Ce numéro est directement utilisé pour ordonner les agents. Pour le domaines des variables et son ordre aucun traitement spécifique n'a été fait là non plus. De la même manière que les stations, les opérateurs et conteneurs arrivant sont numérotés. Le domaine est alors constitué des numéros des opérateurs et conteneurs dans l'ordre dans lequel ils sont fournis au Scheduler.

Suite à cette première version, nous avons décidé d'implémenter une heuristique pour voir si celle-ci améliorerait les algorithmes basiques. L'heuristique choisie est celle utilisée pour résoudre notre problème dans l'approche AmasCOP (cf. section 4.2). Le détail de l'heuristique est donné par la figure 4.1. AmasCOP utilise cette formule pour calculer le degré de non coopération des conteneurs, afin de déterminer l'ordre dans lequel il va choisir les conteneurs, stations et opérateurs à affecter.

Algorithme 14 : Heuristique AmasCOP pour ABT/AWCS - Calcul de la valeur d'une station

```
forall conteneur c compatible do  
  | c.valeur ← Calculer valeur heuristique ;  
end  
forall opérateur op compatible do  
  | op.valeur ← max{valeur conteneurs compatibles à op} ;  
end  
Station.valeur ← max{valeur opérateurs compatibles à Station} ;
```

L'algorithme 14 nous donne la méthode de calcul de la valeur d'une station (ou agent). On calcule d'abord le degré de non coopération, à partir de la formule 4.1, de l'ensemble des conteneurs compatibles pour une station. À partir de ces valeurs on peut déterminer celle des opérateurs compatibles. Sachant tous les conteneurs compatibles à un opérateur, on peut déterminer la valeur de celui-ci en prenant le maximum des valeurs des conteneurs. Pour le degré des stations, et donc l'ordre des agents, chaque station prend le maximum des degrés des opérateurs compatibles. Cette heuristique nous permet d'ordonner les agents mais ordonne aussi les domaines des variables.

6

Approche multi-agent auto-organisatrice : ETTO4MC

6.1 Présentation

Afin d'étudier au mieux les possibilités et les intérêts de la décentralisation de la résolution de problèmes comme la gestion de production, nous expliquons dans ce chapitre l'approche multi-agent auto-organisatrice imaginée et développée, ETTO4MC.

6.1.1 Motivations

Lorsque l'on analyse notre problème de gestion de production, nous voyons qu'au-delà du problème de la création d'affectations liant des stations, conteneurs et opérateurs, il pourrait s'agir en fait plus précisément d'un problème d'affectation de ressources, les stations, pour les acteurs conteneurs et opérateurs.

Nous avons en fait deux acteurs principaux, conteneurs et opérateurs qui vont chercher à atteindre leurs objectifs, respectivement usiner toutes leurs pièces et effectuer des usinages de pièces sur une station. Les stations ne sont ici en fait que des ressources, qui vont être utilisées par ces deux entités, pour satisfaire leurs objectifs. Les stations accueillent un conteneur et un opérateur et servent de lien pour cette association en fournissant la qualification ou usinage nécessaire aux opérations.

Ce constat semble très proche de l'idée qui est utilisée dans ETTO (*cf. section 2.2.2*), approche multi-agent auto-organisatrice, où l'on retrouve des acteurs qui sont des enseignants et des groupes d'étudiants qui cherchent respectivement à trouver une salle pour donner un cours et trouver une salle pour suivre un cours, tout cela en respectant des créneaux horaires. Les deux types d'agents ici se déplacent dans leur environnement, une grille, représentant les ressources, c'est-à-dire les salles et créneaux horaires. Une fois une case trouvée il leur faudra trouver un partenaire pour satisfaire leurs objectifs.

L'analogie est tout de suite simple à voir avec notre problème. ETTO définit la base sur laquelle notre approche a été développée. Le nom de notre système, ETTO4MC (ETTO For Manufacturing Control), en est le reflet.

6.1.2 Modélisation

6.1.2.1 Agentification

En opposition à l’agentification des variables utilisée pour les approches ABT et AWCS, nous utilisons ici plutôt une agentification des entités du domaine.

Le problème définissant trois entités distinctes, nous laisse penser à utiliser trois types d’agents, un type par entité. Au delà de ce premier constat il faut aussi remarquer que deux entités sont assez proches dans leurs objectifs, les conteneurs et les opérateurs. Ces deux agents ne vont chercher qu’à se placer sur une station pouvant les accueillir, donc compatible, alors que les stations constituant ici les ressources ne vont en fait qu’être vues que comme des ressources à utiliser par les autres agents. On va dès lors ne distinguer que deux types d’agents, les conteneurs et opérateurs que l’on nommera BookingAgents (BA) et les stations que l’on appellera RessourceAgents (RA).

Nous avons fait l’analogie avec l’approche ETTO qui utilise l’environnement pour modéliser les ressources à réserver, ETTO4MC se distingue de cette approche en agentifiant également ces ressources permettant ainsi une plus grande possibilité dans les informations transmises par les stations.

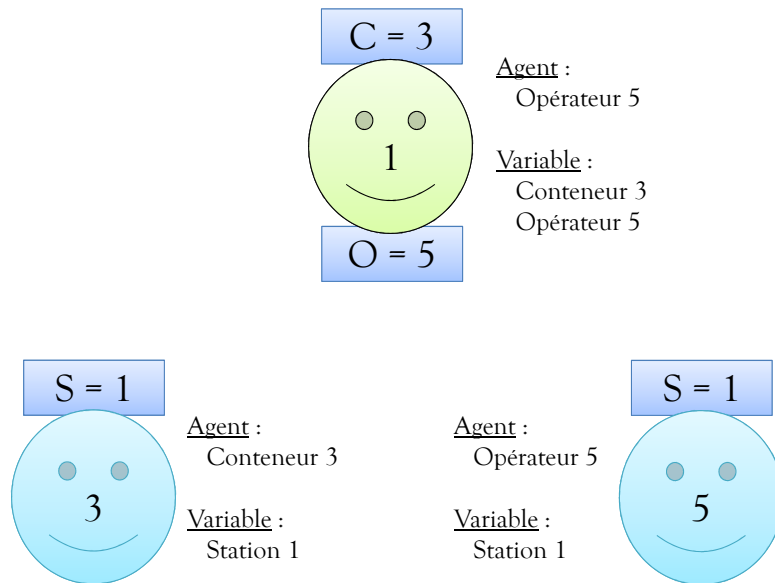


FIGURE 6.1 – Agents ETTO4MC

6.1.2.2 Réseau de contraintes

ETTO4MC utilise une agentification des entités du domaine contrairement à ABT ou AWCS qui eux utilisent une agentification des variables. L’agentification détermine directement le réseau de contraintes, c’est pour cela que nous avons ici un réseau différent de celui de ABT.

En n'utilisant plus simplement des agents station, nous nous dédouanons du réseau de contraintes complet. En effet, ici chaque agent ne va pas communiquer avec tous les autres pour s'assurer de satisfaire toutes ses contraintes. La communication ne s'effectuera qu'entre agents de type différent, BookingAgent et RessourceAgent pour des demandes d'informations et des réservations, ou alors entre BookingAgents pour des négociations. Toutes les relations qui seront utilisées dans le réseau sont temporaires dans le sens où un agent n'a qu'une vue locale et se déplace.

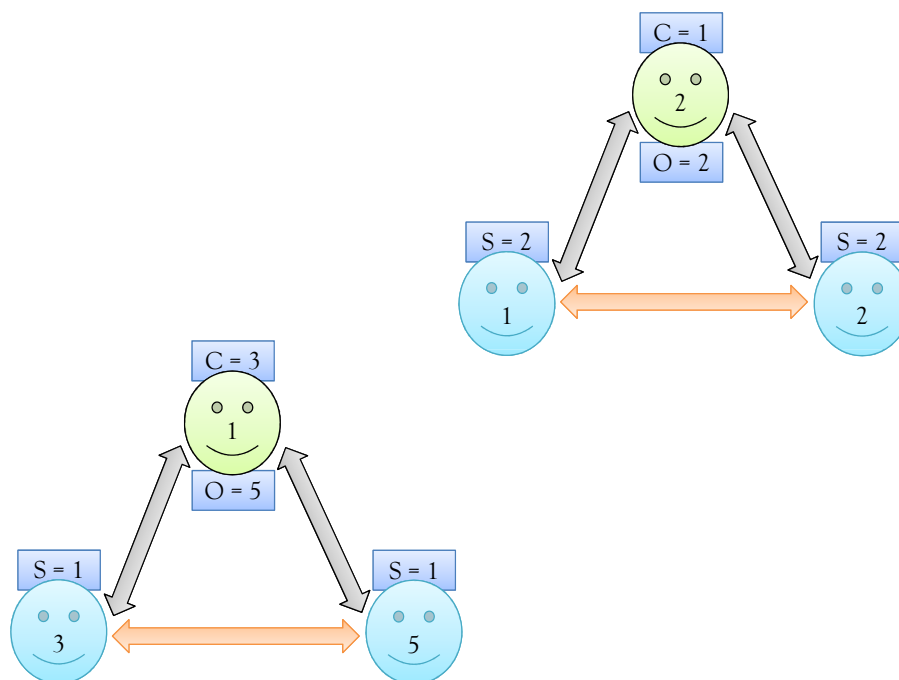


FIGURE 6.2 – Réseau de contraintes ETTO4MC

6.2 Principe général

Dans ETTO4MC tous les BookingAgents vont chercher à s'enregistrer sur un RessourceAgent, autrement dit les conteneurs ou opérateurs vont chercher à se placer sur des stations. Lorsqu'un BookingAgent s'est enregistré sur un RessourceAgent il va continuer à chercher un autre RA possible éventuellement meilleur que celui sur lequel il se trouve.

6.2.1 Actions des agents

Les deux types d'agents du système n'ont pas les mêmes objectifs et fonctionnement, c'est pour cela que nous décrivons distinctement le principe de fonctionnement de chacun d'eux.

6.2.1.1 BookingAgent

Un BookingAgent a pour but de trouver le RessourceAgent le mieux adapté pour lui. Au fur et à mesure du temps, les réservations de RA évoluent et leurs intérêts, pour le BA, changent eux aussi, c'est pourquoi les BA recherchent en permanence un meilleur RA.

Algorithme 15 : Cycle de vie d'un BA

```
if Attente Negociation then
  | TraiteMessageNegociation();
else
  if Attente Information then
    | TraiteMessageInform();
  else
    if  $\exists$  Messages en attente then
      | TraiterMessages();
    else
      | ChercherStation();
    end
  end
end
```

Un BA peut être en attente de différents messages suivant la situation dans laquelle il se trouve. L'envoi de certains messages, nécessitant une réponse, passe l'agent dans des états d'attente soit de négociation soit d'information.

Algorithme 16 : Fonction qui détermine la station à interroger - ChercherStation()

```
Sélectionner RA ;
Interroger RA ;
```

Lors de la recherche d'un RA, le BA sélectionne dans l'ensemble des stations connues, un RA à interroger. Afin de s'assurer de parcourir tous les RA, une liste des RA à consulter est constituée à la création de l'agent. L'ordre de cette liste est aléatoire et comporte tous les RA du système. Une fois le premier parcours terminé, la liste est vide, il recrée une liste avec cette fois-ci seulement les RA qui sont connus comme compatibles au niveau de leurs qualifications. Cette liste est recrée à chaque fois qu'elle est vide. À chaque choix on interroge le RA sélectionné pour recueillir des informations sur celui-ci. L'interrogation d'un RA place le BA dans un état d'attente, où celui-ci ne cherchera qu'à traiter le message de réponse du RA, tous les autres messages étant mis en attente.

Les agents peuvent être, suivant leur partenariat, dans plusieurs états. Ceux possibles pour les RA sont donnés dans la section 6.2.1.2. Concernant les BookingAgents ceux-ci peuvent se trouver dans trois états différents :

- *ALONE*, signifie que le BA n'est encore enregistré sur aucun RA,
- *HALF*, le BA est enregistré sur un RA,
- *FULL*, un partenariat complet est fait pour le RA, il a un RA et un BA partenaire.

Algorithme 17 : Fonction qui détermine l'action suivant les infos du RA - `TraiteMessageInform()`

```

if RA compatible then
  switch état RA do
    case EMPTY
      | TraiteStationEmpty();
    end
    case HALFC
      | TraiteStationHalfC();
    end
    case HALFO
      | TraiteStationHalfO();
    end
    case FULL
      | TraiteStationFull();
    end
  end
else
  | Cancel();
end

```

Lorsqu'un BA reçoit un message il vérifie tout d'abord que le RA est compatible. Si c'est le cas, l'état du RessourceAgent interrogé est déterminé et le BA agit en fonction.

Si le RA indique qu'il n'a encore aucune réservation, ni d'un conteneur ni d'un opérateur, le BA peut se placer sur le RA. Suivant son état le BA va choisir l'action à effectuer. S'il ne se trouve pas sur un RA, donc dans l'état ALONE, il se place sur le RA par un message Book. S'il se trouve dans l'état HALF, déjà placé sur un RA, le BA va alors évaluer l'intérêt du RA interrogé par rapport à celui déjà réservé. Si le BA est un conteneur, il va comparer les deux RA suivant deux critères : tout d'abord le nombre de qualifications consécutives possibles et ensuite le nombre de qualifications communes. Pour les opérateurs, seules les qualifications communes sont utilisées. Concernant le cas où le BA se trouve dans l'état FULL, donc appartenant déjà à un partenariat, il ne cherche pas à réserver le RA et envoie un message d'annulation au RA. En effet, un BA qui est pour le moment satisfait ne va pas quitter un partenariat créé pour se mettre sur une station ou ne se trouve même pas un partenaire (cf. algorithme 18), la fonction correspondante décrite ici est `TraiteStationEmpty`, algorithme 18.

Dans le cas où le RA indique qu'un opérateur s'est enregistré, état HALFO, deux configurations existent. La première, le BA est un conteneur, il agit alors suivant l'algorithme 19. S'il est dans un état ALONE ou HALF, il réserve (book) le RA sinon il évalue le nouveau RA et choisit ou non de le réserver. Pour un BA opérateur, celui envoie directement un message de négociation à l'opérateur en place pour déterminer s'il réserve ou non le RA. Au contraire si le RA indique un état HALFC, les rôles sont inversés. L'opérateur applique l'algorithme 20, et le BA conteneur, lui, négocie avec celui enregistré.

Si le RA est FULL, un conteneur et un opérateur sont enregistrés auprès de lui. Si le BA correspond à un conteneur, il vérifie que l'opérateur réservant le RA est compatible avec lui, si ce n'est pas le cas il annule sa

demande au RA. Si l'opérateur est compatible alors si le BA se trouve être ALONE ou HALF, il négocie avec le conteneur placé pour tenter de prendre sa place, sinon il annule. Si le BA est un opérateur, il reproduit le même mécanisme avec une inversion des types des BA.

Algorithme 18 : Fonction qui détermine l'action suivant les infos du RA - *TraiteStationHalfO()*

```

switch état BA do
  case ALONE
    | Book();
  end
  case HALF
    | if EvaluePartenariat() = 1 then
      | Book();
    else
      | Cancel();
    end
  end
  case FULL
    | Cancel();
  end
end
end

```

6.2.1.2 RessourceAgent

Le RessourceAgent contrairement au BookingAgent ne fait que réagir aux messages qui lui envoyés, il n'agit pas de lui-même pour rechercher des conteneurs ou opérateurs à placer. Il représente une ressource, il ne fait que fournir son état, les informations sur ses qualifications et sur les BA enregistrés auprès de lui. Un RA peut être dans quatre états possibles :

- *EMPTY*, signifie qu'aucun BA n'est enregistré sur le RA ;
- *HALFC*, signifie qu'un BA conteneur est BA est placé ;
- *HALFO*, veut dire qu'un Ba opérateur se trouve sur le RA ;
- *FULL*, le RA accueille un BA conteneur et BA opérateur.

Les trois types de messages que le RA peut recevoir sont :

- *OkForBooking*, qui est une demande d'informations sur le RA,
- *Book*, qui est une réservation de la part d'un BA,
- *Cancel*, qui est une annulation de réservation.

Lorsqu'un RA reçoit un message *OkForBooking*, il répond au BA émetteur par un message *Inform*, dans lequel se trouvent toutes les informations sur lui. Les informations regroupent, son état courant, les BA enregistrés s'il en existe ainsi que ses qualifications.

Si un RessourceAgent reçoit un message *Book*, il enregistre alors le BA. Si un BA du même type que le l'émetteur du message se trouve déjà sur le RA, il envoie alors un message *Cancel* au BA placé pour lui indiquer

Algorithme 19 : Fonction qui agit si le RA est HALFO - *TraiteStationHalfC()*

```

if opérateur booked est compatible then
  switch état BA do
    case ALONE
      | Book();
    end
    case HALF
      | Book();
    end
    case FULL
      if EvaluatePartenariat() = 1 then
        | Book();
      else
        | Cancel();
      end
    end
  end
else
  | Negotiate();
end

```

qu'il n'est plus sur cette station et enregistre le nouveau BA.

Si un message Cancel, parvient au RA, il sait alors que l'émetteur veut quitter le partenariat, il le supprime alors de ses réservations et informe si besoin est, l'autre BA partenaire de ce changement.

6.2.2 Protocole de communication

Nous détaillons ici les messages qui peuvent être envoyés dans le système au moyen d'un exemple de recherche menée par un BA de type conteneur. Le cas déroulé ici est celui de la figure 6.3.

Le conteneur 2 interroge la station 1 pour connaître ses dispositions, au moyen d'un message OkForBooking. Suite à ce message le RA, renvoie une réponse avec un message Inform. Une fois le message reçu, le conteneur 2, a deux possibilités, soit la station n'est pas compatible et il répond par un message Cancel, car il ne peut de toute façon pas se placer sur la station, soit il va tenter de réserver le RA. Dans le cas où le BA souhaite réserver le RA, il faut prendre en compte l'état du RA. Si celui-ci est EMPTY ou à demi-réservé, donc HALFO avec un opérateur compatible, le conteneur se place sur le RA avec un message Book. Si ce n'est pas le cas, et qu'un conteneur est déjà sur la station, il va chercher à négocier avec celui-ci pour tenter d'obtenir la place. La négociation est lancée au moyen de l'envoi d'un message Negotiate au conteneur en place. Dès qu'il reçoit le message de négociation, le conteneur 1 évalue alors s'il doit ou non laisser la place au conteneur 2 (pour les critères d'évaluation voir section 6.2.3). Il répond alors et indique le résultat de son analyse par un message Result. La réception d'un Result, peut amener l'initiateur de la négociation dans deux possibilités d'action. Soit la réponse indique qu'il peut réserver la station et alors il envoie un message Book, soit il ne peut pas et il envoie un message Cancel. Quand la station

Algorithme 20 : Fonction qui agit si le RA est HALFC - *TraiteMessageInform()*

```

if conteneur booked est compatible then
  switch état BA do
    case ALONE
      | Book();
    end
    case HALF
      | Book();
    end
    case FULL
      if EvaluatePartenariat() = 1 then
        | Book();
      else
        | Cancel();
      end
    end
  end
else
  | Cancel();
end

```

reçoit un message Book, elle informe alors l'opérateur qui l'a réservé, d'un changement de partenaire, le conteneur 2 ayant remplacé le conteneur 1, par l'envoi d'un message Partner.

6.2.3 Coopération

Dans ETTO4MC, la coopération entre agent permet de s'assurer que les agents les plus prioritaires sont traités et privilégiés par rapport aux autres agents qui le sont moins. Ainsi les BookingAgents, représentant les conteneurs, peuvent être ordonnés suivant un ordre de priorité en se basant sur leur cycle de fin, par exemple.

Le calcul de la valeur de coopération utilise différents critères suivant les situations et éléments à évaluer. Lorsqu'un BA conteneur doit choisir entre deux RA stations, il évalue deux choses pour les comparer. En premier lieu il calcule le nombre de qualifications qu'il a en commun avec celles-ci, s_q , et en deuxième lieu il calcule le nombre de qualifications consécutives qu'il peut effectuer sur chacune d'elle, c_q . En combinant ces deux critères il peut choisir quel RA est le plus avantageux pour lui. La formule est la suivante :

$$\mathcal{V}_{coop}^s = s_q c_q$$

Lorsque deux conteneurs veulent occuper le même RA, il vont négocier entre eux pour déterminer lequel est le plus prioritaire, pour que celui-ci réserve la station. L'évaluation entre conteneurs s'opère au moyen de plusieurs critères précis. Le premier est le temps restant, t_r , au conteneur pour effectuer tous ces usinages. Le second élément utilisé est le temps restant total nécessaire pour usiner toutes les pièces du conteneur, δ_c . À partir des ces deux informations, le BA calcule la priorité ou marge pour lui et le BA en négociation avec la formule $t_r - \delta_c$. Entre

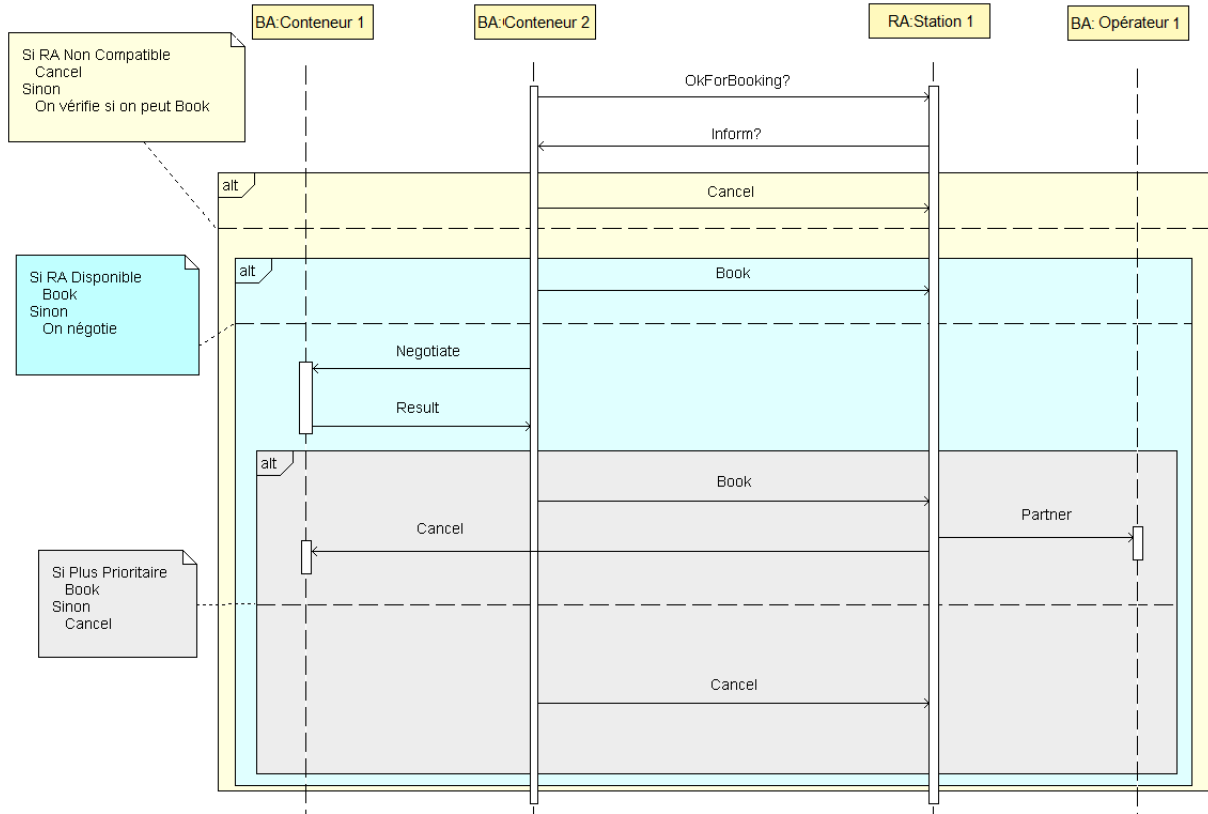


FIGURE 6.3 – Exemple de cycle ETTO4MC

le temps restant, t_r , et la marge $t_r - \delta_c$, il est clair que la marge est beaucoup plus importante à respecter que le temps restant, un coefficient μ est utilisé pour privilégier la marge sur la priorité. Le conteneur ayant la valeur de coopération la plus petite gagne la négociation. La valeur est calculée avec la formule suivante :

$$\mathcal{V}_{coop}^c = \mu \frac{1}{t_r - \delta_c} + t_r \quad \text{avec } \mu > t_r$$

Dans le cas d'une négociation entre opérateurs, ou des BA de types différents (conteneur et opérateur), d'autres critères sont utilisés. Les agents cherchent à évaluer la difficulté de chacun à se placer sur un RA compatible. Les possibilités de placement sont évaluées, en calculant le ratio des stations compatibles, m_s , par rapport aux stations disponibles, a_s , et en prenant l'inverse, $\frac{m_s}{a_s}$, afin de privilégier le BA avec le ratio le plus petit. De plus le ratio $\frac{p}{r}$, avec p représentant le nombre de partenariat déjà trouvés et r le nombre de demandes de partenariats faits, mesure l'efficacité des BA à se trouver des partenaires. Comme pour la formule précédente une valeur μ de priorité est utilisée pour permettre une priorisation des valeurs évoquées. Le calcul de la valeur est donné par la formule suivant, qui fait également intervenir la formule v_s explicité plus haut.

$$\mathcal{V}_{coop}^o = \mu^2 \frac{a_s}{m_s} + \mu \frac{1}{v_s} + \frac{r}{p} \quad \text{with } \mu > \max\left\{\frac{1}{v_s}, \frac{r}{p}\right\}$$

6.2.4 Limites

L'algorithme ETTO4MC est une approche qui reste perfectible. Les BA dépendent fortement de la valeur de coopération mise en place, pour les phases de négociation. Les critères qui ont été choisis, de manière empirique, sont ceux qui semblent, a priori, être déterminants pour évaluer quel BA est à privilégier par rapport à un autre. Suite à la première version de ETTO4MC, développée, qui est celle exposée dans ce rapport, le développement d'une seconde version a été entamée. Cette seconde version a notamment pour but d'essayer une nouvelle valeur de coopération faisant intervenir de nouveaux critères, comme la difficulté prenant en compte le temps depuis la dernier book, ou le pourcentage de qualifications communes avec un RA. La composition de la valeur peut aussi être influencée par les coefficients utilisés (coefficient μ pour les valeurs de la section précédente). Déterminer les critères pertinents pour notre problème ainsi que les bons coefficients, ne peut se faire qu'au moyen d'expérimentations et d'analyse.

Un second point est limitant concernant notre approche, elle ne comporte aucun mécanisme propre pour sa terminaison. À chaque cycle de résolution les agents cherchent indéfiniment à créer des partenariats. Tous les agents ont une vue locale du problème, ils communiquent en point à point avec les agents proches. De ce fait aucun d'eux ne peut détecter que tous les agents sont satisfaits, c'est-à-dire faisant parti d'un partenariat. De plus il n'est pas possible avec cette vue locale, d'être sûr qu'il existe à un moment t , une solution faisant intervenir tous les agents à la fois. Le nombre d'affectations possibles ne peut pas directement être connu et on ne donc pas se baser sur le nombre d'affectation créées pour stopper le système. Le critère de terminaison en place dans ETTO4MC est pour cette version, la durée maximale d'un cycle. La résolution est lancée et elle est stoppée au bout d'un certain temps. Le scheduler récupère à cet instant les affectations courantes et les simule. Une première piste qui pourrait être explorée pour la terminaison est la détections des pseudo minima locaux, afin d'en déduire la situation dans un minimum local global du système. Ce mécanisme est notamment celui utilisé par le Distributed Breakout Algorithm détaillé en section 3.2.2.

7 Expérimentations

Suite au développement des approches expliquées dans les chapitres précédents, plusieurs expérimentations ont été menées. Nous avons cherché à comparer les cinq approches mise en place suivant des critères comme le temps de résolution, le nombre de messages ou encore le nombre d'usinages effectivement effectués.

7.1 Scénario de test

Pour les expérimentations un scénario mettant en œuvre une multitude de facteurs a été imaginé. Afin de tester les algorithmes face à un problème très dynamique, le scénario créé regroupe tous les éléments dynamiques possibles d'introduire dans la plate-forme MASC. Le scénario a les caractéristiques suivantes :

- 10 qualifications de durée entre 1 et 6 cycles d'usinage,
- 22 opérateurs ayant entre 2 et 3 qualifications chacun, ainsi que 3 en ayant d'une seule, nous donnant au total 25 opérateurs ;
- 26 stations avec entre 2 et 3 qualifications, plus 4 qui n'en possèdent qu'une seule, nous faisant au total 30 stations ;
- 100 conteneurs.

Concernant les perturbations imaginées elles sont les suivantes :

- panne de station :
 - 1 station à $t = 10$ pour 15 cycles,
 - 1 station à $t = 20$ pour 10 cycles,
 - 1 station à $t = 100$ pour 10 cycles,
 - 1 station à $t = 150$ pour 30 cycles,
 - 1 station à $t = 150$ pour 20 cycles ;
- grève ou maladie d'opérateur :
 - 1 opérateur à $t = 10$ pour 15 cycles,
 - 1 opérateur à $t = 30$ pour 20 cycles,
 - 1 opérateur à $t = 100$ pour 10 cycles,
 - 1 station à $t = 150$ pour 20 cycles ;
- arrivé de nouveaux conteneurs :

- 50 conteneurs à $t = 0$ avec fin à $t = 200$,
- 25 conteneurs à $t = 5$ avec fin à $t = 300$,
- 25 conteneurs à $t = 10$ avec fin à $t = 150$.

7.2 Résultats

Toutes les données présentées dans les sections suivantes, sont les résultats d'une série de 50 expérimentations lancées avec le scénario présenté en section 7.1. Les tests ont tous été effectués sur la même machine ayant les caractéristiques suivantes ; processeur AMD 2 4200+, 3 Go de DDR2 800 Mhz. Les critères détaillés sont :

- le temps de résolution par cycle d'usinage, en ms,
- le nombre de messages échangés,
- le nombre de conteneurs usinés et restants.

Nous comparons ici six approches différentes :

- AmasCOP, qui est présentée en section 4.2,
- ABT, présentée en section 5.1.1.1,
- AWCS, présentée en section 5.1.1.2,
- ABTCOP, algorithme ABT amélioré avec l'heuristique AmasCOP,
- AWCSCOP, algorithme ABT amélioré avec l'heuristique AmasCOP,
- ETTO4MC, approche multi-agent auto-organisatrice, définie en section 6.2.

Ces algorithmes ont été implémentés en Java. La plate-forme multi-agent utilisée pour la création des agents et qui s'occupe de leur communication est la plate-forme JADE ¹.

7.2.1 Temps

Le premier critère pris en compte pour comparer toutes nos approches est le temps de résolution. La figure 7.1 présente les temps de résolution des six approches étudiées. Le temps de résolution n'est pas le critère le plus pertinent pour l'étude d'algorithmes utilisant des méthodes de résolution distribuées. En effet la transmission des messages dans ces systèmes est asynchrone et de plus la résolution est parallèle et distribuée sur une multitude d'agents.

L'approche ETTO4MC présente un temps de résolution beaucoup plus élevé que les autres approches. On constate ici la faiblesse de ETTO4MC, qui réside principalement dans le critère temporel de terminaison des cycles de résolution. Pour les expérimentations, le temps par cycle était réglé à 300 ms. On constate également que cet algorithme a un temps de résolution global qui augmente beaucoup au début pour au fur et à mesure diminuer. Cette diminution est due au fait que l'approche utilise une agentification des entités du domaine. ETTO4MC est une approche plus adaptée pour la réparation de solutions que pour la résolution a proprement dit, ceci explique aussi en partie son relativement mauvais résultat. En utilisant des communications locales, nous avons un algorithme

1. <http://jade.tilab.com/>

s'adaptant facilement aux changements mais assez long à trouver l'organisation initiale. Au début de l'algorithme beaucoup d'agents sont présents dans le système (30 stations + 25 opérateurs + 100 conteneurs = 155 agents), au fur et à mesure de l'avancée dans le temps des conteneurs sont usinés et sont donc sortis du système. Le nombre d'agent diminue et du coup la combinatoire du problème aussi. Les autres approches utilisant une agentification des variables, ont durant toute la résolution, le même nombre d'agents, c'est-à-dire 30 agents, un par station. L'algorithme AmasCOP utilisant une modélisation par agents mais étant centralisé, a un temps de résolution peu élevé et variant peu.

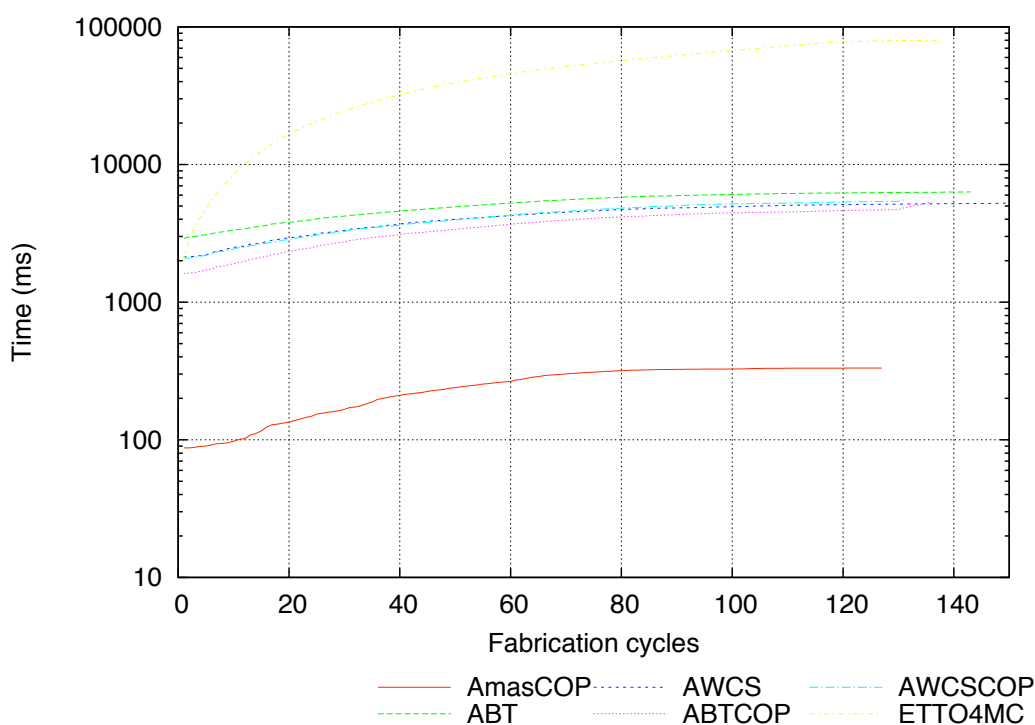


FIGURE 7.1 – Temps de résolution en fonction des cycles d'usinage

La figure 7.2 montre plus précisément les courbes des algorithmes ABT, AWCS et leurs améliorations, ABTCOP et AWCS COP. Les courbes explicitent deux choses intéressantes. Premier élément à noter, la faible différence entre AWCS et son amélioration AWCS COP. AWCS utilise un ordre dynamique sur les agents, celui évoluant en cours de résolution. L'heuristique utilisée ordonne les agents dès le départ suivant leur valeur de coopération. Les changements d'ordre durant la résolution ne sont donc pas nombreux et n'influence pas tellement le temps de résolution. Par contre, nous constatons que pour ABT et ABTCOP la différence est flagrante, un grand écart sépare les deux algorithmes. L'heuristique améliore grandement ABT et permet de diminuer considérablement le temps de résolution. Les agents étant ordonnés en prenant en compte leur utilité, la résolution devient plus facile.

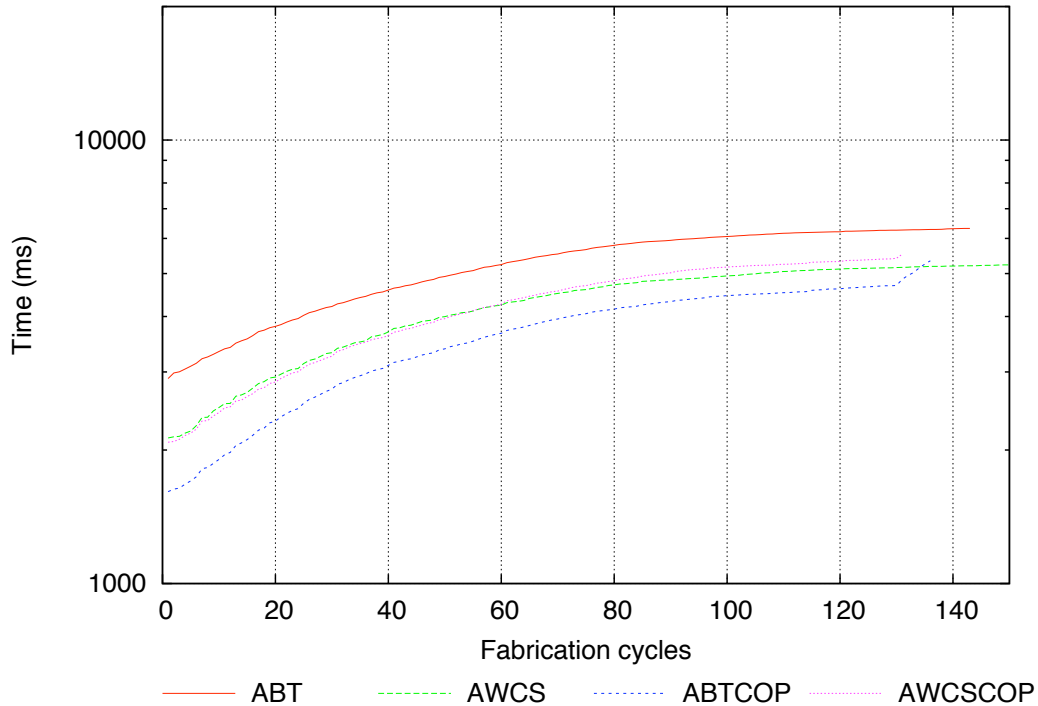


FIGURE 7.2 – Temps de résolution en fonction des cycles d’usinage

7.2.2 Messages

La figure 7.3 représente le nombre de messages échangés par cycle d’usinage. L’étude du nombre de messages transmis par les agents permet d’évaluer la charge du réseau de communication induite par toutes ces approches.

ETTO4MC se distingue vraiment des approches par agentification des variables par son grand nombre de messages, beaucoup plus important que ces dernières. Ceci s’explique par le plus grand nombre d’agents utilisé ainsi que le mécanisme de coopération gourmand en terme de messages. L’évaluation de la priorité entre agents nécessite au moins deux messages (Negotiate et Results), de plus presque à chaque fois qu’un RA doit agir pour réserver un RA sur lequel se trouve déjà un BA, il doit négocier. Le placement d’un BA sur un RA implique l’envoi de trois messages (OkForBooking, Inform et Book ou Cancel) augmentant encore le nombre total de messages. Comme pour le temps de résolution la courbe augmente rapidement pour se stabiliser vers la fin du scénario. Ceci semble logique dans la mesure où plus le nombre de messages transmis par cycle est important plus le temps de résolution est grand. Cette caractéristique doit être notée car elle pourrait éventuellement poser des problèmes lors du déploiement de ETTO4MC dans un environnement réellement distribué. Il faut aussi noter que AmasCOP ne génère aucun message car cet algorithme glouton ne fait qu’utiliser une modélisation agents mais a une résolution centralisée.

Comme c’était le cas pour la figure 7.2, une forte différence existe entre ABT et son amélioration ABTCOP. L’heuristique diminue considérablement le nombre de messages nécessaires à la résolution alors que pour AWCS cela est plus nuancé même si l’heuristique diminue là aussi les communications.

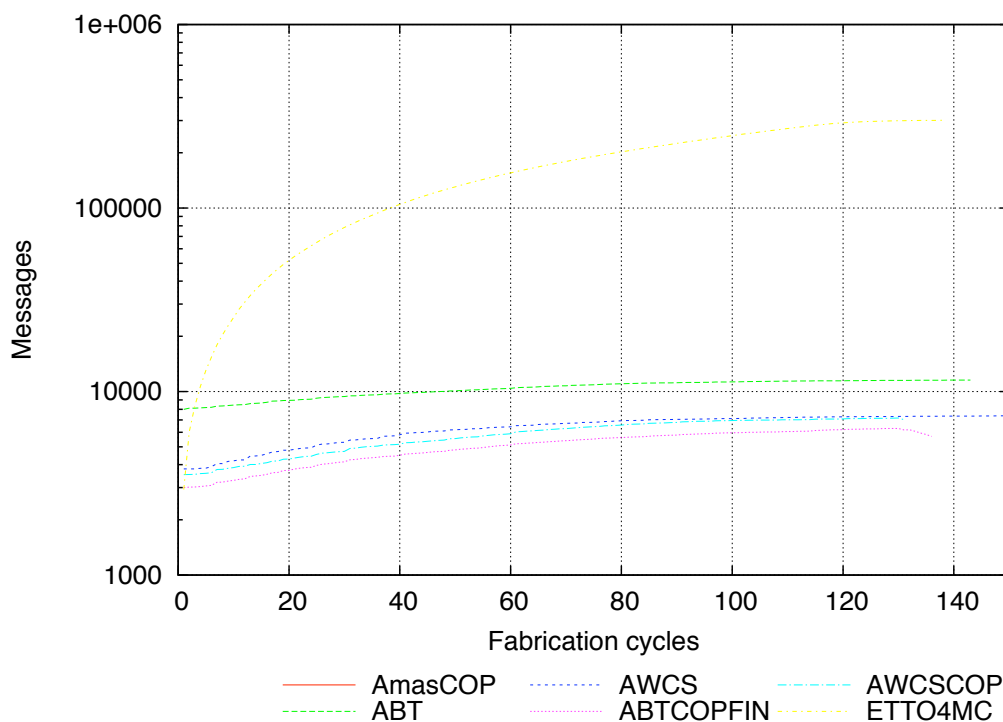


FIGURE 7.3 – Nombre de messages en fonction des cycles d'usage

7.2.3 Usinages

Nous voyons ici les résultats obtenus concernant le nombre de conteneurs usinés par cycle de résolution, avec la figure 7.5. Cette mesure nous permet d'appréhender l'efficacité des différentes approches à usiner tous les conteneurs dans les temps.

Nous constatons d'abord que les approches de type ABT ou AWCS donc par agentification des variables, usinent beaucoup de conteneurs très vite, contrairement à ETTO4MC qui a une courbe croissant moins rapidement. Les algorithmes par agentifications des variables donnent dès le départ beaucoup d'affectations ce qui résulte en beaucoup de conteneurs traités. Au contraire ETTO4MC, qui lui cherche plutôt à privilégier en priorité les conteneurs les plus en difficultés ou plus prioritaires au niveau du temps, crée moins d'affectations mais usinent les conteneurs les plus complexes. Ces conteneurs sont donc traités plus tardivement. Les courbes de ABTCOP et AWCSOP confirment que la coopération, mise en œuvre par l'heuristique, intervient bien dans le phénomène cité pour ETTO4MC. Il faut noter que vers le cycle 95 toutes les approches se rejoignent, pour voir les algorithmes qui étaient jusque là supérieurs, passer en dessous de AmasCOP ou ETTO4MC. À partir de ce point les courbes ont la disposition inverse à celle vue entre le début de la résolution et le cycle 95. La coopération montre son intérêt vers la fin du scénario. Les conteneurs difficiles à usiner sont privilégiés et sont traités avant leur cycle de fin. ABT ou AWCS ne tiennent pas compte de cette priorité pour les placer et n'arrivent pas à traiter tous les conteneurs.

Le tableau 7.2 résume les résultats des différentes approches. Nous constatons que les approches par agentification des variables simples n'arrivent pas dans les temps à traiter tous les conteneurs. L'heuristique améliore le nombre de conteneurs usinés par ces approches pour même finir par tous les traiter. ETTO4MC usine tous les

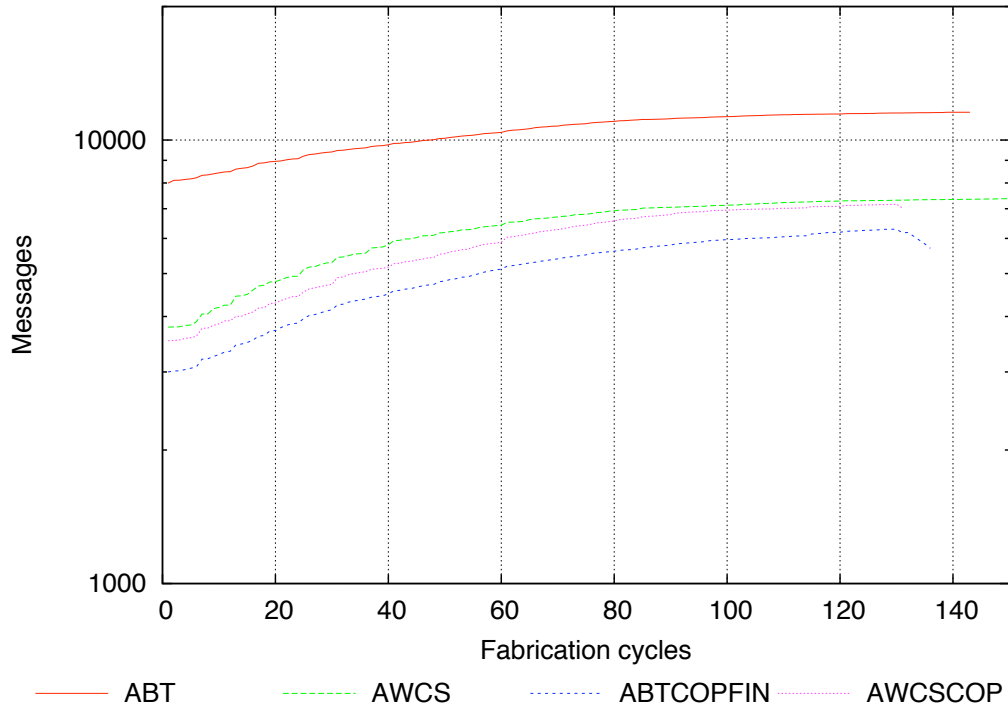


FIGURE 7.4 – Nombre de messages en fonction des cycles d’usinage

conteneurs dans les temps lui aussi.

	Traités	Restants
AmasCOP	100	0
ABT	99	1
AWCS	98	2
ABTCOP	100	0
AWCS COP	100	0
ETTO4MC	100	0

TABLE 7.1 – Nombre de conteneurs traités et restants en fin de simulation

7.3 Analyse

Les approches multi-agents auto-organisatrices ont vocation à être distribuées et à s’attaquer à des problèmes présentant une grande dynamique. La décentralisation de la décision est abordée et discutée dans cette section.

7.3.1 Dynamique

Un critère déterminant pour s’attaquer à des problèmes réels reste le caractère dynamique de ceux-ci. Le scénario utilisé essaie de modéliser un maximum de perturbations possibles pouvant surgir. Parmi toutes celles imaginées, ce sont les arrivées successives de conteneurs qui sont les plus intéressantes à étudier. Devant ces changements les agentifications des variables des algorithmes ABT et AWCS ne leur permettent pas d’atteindre une

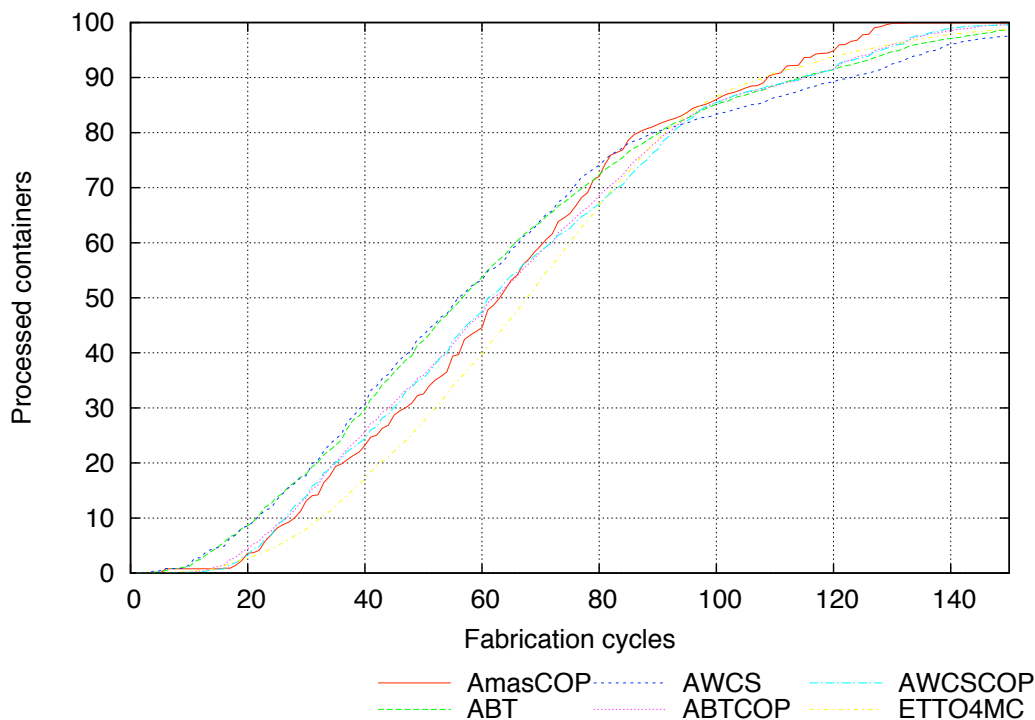


FIGURE 7.5 – Nombre de conteneurs traités en fonction des cycles d’usinage

résolution complète des usinages. Tous les autres algorithmes, qui eux utilisent une valeur de coopération arrivent à traiter tous les conteneurs en moyenne, même ABTCOP et AWCSCOP. Nous voyons alors que la coopération est un bon moyen d’aborder la dynamique au moyen d’agents communiquant localement et qui essaie d’évaluer ensemble les meilleures actions pour l’ensemble d’entre eux. Pour nuancer ce fait, il faut noter que la coopération dans l’approche ETTO4MC, basée sur une agentification des entités, améliore les résultats en terme d’usinages mais est également coûteuse en terme de messages. Elle nécessite un grand nombre de messages pour être efficace et charge très vite le réseau de communication. Nous aurions souhaiter pouvoir injecter plus d’éléments de dynamique dans le scénario de test, comme par exemple des conteneurs n’arrivant pas jusqu’aux stations ou des temps de préparation des stations entre changements de qualifications. La plate-forme MASC ne nous permet pas tout cela, en grande partie à cause de son mode de fonctionnement en tour par tour, qui fournit à chaque temps t l’ensemble des entités du problème.

7.3.2 Distribution

Tous les algorithmes cités dans cette étude ne présentent pas tous le même niveau de distribution. La distribution de la résolution ne signifie pas automatiquement la décentralisation de la prise de décision. Nous voyons ici où se situent les algorithmes développés.

AmasCOP est une approche qui utilise une modélisation par agents mais présente une résolution qui n’est pas distribuée et qui est même centralisée. Les agents n’existent pas vraiment, ils ne sont que des variables utilisées par l’algorithme glouton pour disposer d’un calcul de coopération entre les entités du domaine. La décision est prise en ayant toutes les connaissances sur l’ensemble des caractéristiques du problème. Cela empêche toutes distribution

et décentralisation de la résolution pour cette approche.

ABT, AWCS, ABTCOP et AWCSCOP utilisent une agentification des variables, où un agent représente une station. Ici les agents existent vraiment et peuvent être distribués dans un réseaux de machines communicantes. Ils communiquent ensemble et effectuent chacun indépendamment une partie du calcul de la solution. Néanmoins ces algorithmes reste centralisés dans la décision dans la mesure où tous les agents sont dépendants du premier agent dans l'ordre qui décide de la terminaison et renvoi la solution.

ETTO4MC, avec son agentification des entités, fournit une approche distribuable avec une décentralisation de la décision. Les agents représentant les entités peuvent se déplacer et avec leur mécanisme de communication point à point résoudre chacun une partie du problème. La décision est bien décentralisée dans le sens où aucun agent ne peut lui-même décider de renvoyer une solution ni de résoudre lui-même tout le problème. Il faut aussi dire que cette décentralisation et l'utilisation de la coopération augmente considérablement le nombre de messages utiles par rapport aux autres approches.

ETTO4MC est une approche qui décentralise au maximum la prise de décision. Elle permet d'obtenir une grande réactivité à la dynamique et fournit donc une méthode de réparation plus qu'une méthode de résolution. La décentralisation de la décision est une caractéristique très importante pour palier à certains problèmes dynamiques. Les approches la mettant en application, comme ETTO4MC, au contraire des méthodes ne le faisant pas, comme ABT, AWCS ou AmasCOP, résistent beaucoup plus facilement aux perturbations qui touchent directement les entités. Si le premier agent disparaît dans une approche distribuée de type ABT, il est alors impossible d'assurer la fin de la résolution. Dans ETTO4MC, les agents n'étant pas ordonnés et aucun ne prenant de décision pour tout le système, la disparition de l'un d'entre-eux n'empêche en aucun cas la bonne marche du système.

Pour résumer cette analyse nous citons le tableau synthétique suivant spécifiant si les approches utilisées présentent un caractère distribué ou de décentralisation de décision, ainsi que leur bonne adaptation à la dynamique du problème.

	Dynamique	Distribution	Décentralisation
AmasCOP	X	-	-
ABT	-	X	-
AWCS	-	X	-
ABTCOP	X	X	-
AWCSCOP	X	X	-
ETTO4MC	X	X	X

TABLE 7.2 – Caractéristiques des approches

Conclusion

Bilan

Dans ce rapport nous avons abordé le problème de la gestion de production par système multi-agent auto-organisateur. Après avoir défini la notion de gestion de production et explicité les activités qui s’y rattachent, nous avons décidé d’utiliser une modélisation CSP pour résoudre le cas d’étude qui nous était proposé. Afin de prouver le bien fondé d’une approche multi-agent auto-organisatrice, nous avons choisi de la confronter à des algorithmes bien connus, qui sont ABT et AWCS.

Dans un premier temps nous avons apporté une modélisation adaptée au problème posé, qui nous a permis d’adapter les algorithmes génériques, que sont ABT et AWCS, à notre cas d’étude spécifique. La conception de notre approche multi-agent auto-organisatrice avait pour caractéristique, l’utilisation de la coopération entre agents pour guider la recherche de l’organisation globale du système. Afin de voir l’influence directe de ce mécanisme, une heuristique de coopération a été ajoutée aux algorithmes ABT et AWCS avec pour but, de démontrer les apports de la coopération dans les systèmes présentant l’utilisation d’agents.

Notre majeure contribution réside dans la proposition et l’implémentation d’un système multi-agent auto-organisateur basé sur des agents coopératifs, ETTO4MC. Nous avons obtenu un système constitué d’un ensemble d’agents résonnant localement, permettant une distribution des calculs et une décentralisation totale de la décision, aboutissant à un système souple et adaptatif.

La comparaison des différentes approches, a clairement démontré l’intérêt de la coopération au sein des systèmes multi-agents, au travers des résultats des expérimentations menées. En effet les algorithmes utilisant soit une heuristique coopérative, AmasCOP, ABTCOP et AWCSOP, soit des négociations coopératives, ETTO4MC, ont tous réussi à résoudre totalement le scénario proposé, alors que les approches classiques, ABT et AWCS, n’ont pas pu terminer la totalité des usinages. Il a aussi été clairement mis en évidence que ces améliorations s’obtiennent au prix d’une forte augmentation du nombre de messages échangés. Enfin nous pouvons également faire remarquer que notre travail a permis d’identifier plusieurs points bloquants pour l’étude de la dynamique au sein de la plate-forme MASC. Le mode tour par tour ce celle-ci est très limitant et ne permet pas d’avoir une dynamique réelle et totale, car le simulateur ne fait que fournir à un temps t une configuration précise d’un problème à résoudre. Nous avons fait ces remarques au concepteur de la plate-forme qui a pu faire évoluer MASC pour de futures expérimentations.

Nous tenons également à signaler que ce travail a fait l’objet d’une publication acceptée aux Journées Fran-

cophones sur les Systèmes Multi-Agents de 2008². Cet article est le fruit d'une collaboration avec l'Institut de Recherche en Informatique de Toulouse et notamment Elsy Kaddoum, stagiaire de Master 2 Recherche, qui a elle aussi développée une approche multi-agent auto-organisatrice pour la gestion de production [Clair *et al.*, 2008].

Perspectives

Parmi les perspectives qui seront à explorer, il y a la mise en place d'un mécanisme de terminaison pour l'approche ETTO4MC, par détection des quasi minima locaux. En effet, pour le moment la terminaison est faite sur un critère temporel, ce qui dégrade considérablement les solutions retournées, celles-ci étant simplement une *photographie* de l'état du système à un moment précis. Il serait intéressant de pouvoir avoir une détection de solution globale, ou une évaluation de la solution courante pour détecter la terminaison de la résolution.

De plus, des améliorations concernant le calcul de la valeur de coopération sont à imaginer. Un début de modification est en cours et a pour but d'identifier les critères pertinents et leur ordre d'importance dans la coopération entre agents.

Enfin, il serait également important de tester le déploiement de ETTO4MC dans un environnement réellement distribué, afin d'évaluer la charge du réseau de communication induite, pour aboutir ou non, éventuellement, à des modifications ayant pour principal but de diminuer le nombre de messages transmis. Par rapport au cahier des charges initial, certaines contraintes ont été abandonnées, comme celles concernant le transport des conteneurs par exemple. Il serait intéressant de pouvoir également les prendre en compte dans une prochaine version de la plate-forme MASC et surtout dans la prochaine version de ETTO4MC.

2. <http://www.cerv.fr/jfsma08/>

Bibliographie

- S. BRUECKNER : *Return from the Ant*. Thèse de doctorat, PhD Dissertation, Humboldt-Universität Berlin, Germany, 2000.
- D. CAPERA, C. BERNON et P. GLIZE : Etude d'un Processus d'Allocation Coopératif de Ressources Entre Agents pour la Gestion de Production. In *Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, Lille, France, pages 369–383. Presses Universitaires de Valenciennes, février 2006.
- G. CLAIR, M.-P. GLEIZES, E. KADDOUM et G. PICARD : Approches multi-agents auto-organisatrices pour un contrôle manufacturier intelligent et adaptatif. In *Journées Francophones sur les Systèmes Multi-Agents (JFSMA'08)*, Brest, France, Octobre 15-17. Cépaduès, 2008.
- R. DECHTER : *Constraint Processing*. Morgan Kaufmann, 2003.
- R. DECHTER et A. DECHTER : Belief maintenance in dynamic constraint networks. *Proceedings of AAAI*, pages 37–42, 1988.
- G. DI MARZO SERUGENDO, M.P. GLEIZES et A. KARAGEORGOS : Self-Organization in Multi-Agent Systems. *The Knowledge Engineering Review*, 20(02):165–189, 2006.
- EW DIJKSTRA et CS SCHOLTEN : Termination Detection for Diffusing Computations. *INFO. PROC. LETT.*, 11 (1):1–4, 1980.
- J. DRÉO et P. SIARRY : *Métaheuristiques pour l'optimisation difficile*. Eyrolles, 2003.
- J. DUGGAN et J. BROWNE : Production Activity Control : A Practical Approach to Scheduling. *International Journal of Flexible Manufacturing Systems*, 4(1):79–103, 1991.
- Jacques FERBER : *Les systèmes multi-agents : vers une intelligence collective*. InterÉditions, 1995.
- Marie-Pierre GLEIZES : *Vers la résolution de problèmes par émergence*. Habilitation à diriger des recherches, Université Paul Sabatier, Toulouse, France, décembre 2004. URL ftp://ftp.irit.fr/IRIT/SMAC/DOCUMENTS/RAPPORTS/HdR_MPGleizes_1204.pdf.
- H. HATTORI et T. ITO : A Quick Adaptation Method for Constraint Satisfaction in a Real-time Environment. *IJCSNS*, 6(7B):107, 2006.
- H. KARUNA, P. VALCKENAERS, B. SAINT-GERMAIN, P. VERSTRAETE, C.B. ZAMFIRESCU et H. VAN BRUSSEL : Emergent Forecasting Using a Stigmergy Approach in Manufacturing Coordination and Control. *Engineering Self-organising Systems : Methodologies And Applications*, 3464, 2005.
- J. LIU, H. JING et YY TANG : Multi-Agent Oriented Constraint Satisfaction. *Artificial Intelligence*, 136(1):101–144, 2002.
- R. MAILLER et V. LESSER : Using Cooperative Mediation to Solve Distributed Constraint Satisfaction Problems. *Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 1:446–453, 2004.

- S. MINTON, M.D. JOHNSTON, A.B. PHILIPS et P. LAIRD : Minimizing Conflicts : a Heuristic Repair Method for Constraint Satisfaction and Scheduling Problems. *Constraint-Based Reasoning*, 58(1-3):161–205, 1994.
- P.J. MODI, W.S. SHEN, M. TAMBE et M. YOKOO : ADOPT : Asynchronous Distributed Constraint Optimization with Quality Guarantees. *Artificial Intelligence*, 161(1-2):149–180, 2006.
- P. MORRIS : The Breakout Method for Escaping From Local Minima. *Proceedings of AAAI*, 93(1993):40–45, 1993.
- G. PICARD, C. BERNON et M.P. GLEIZES : Emergent Timetabling Organization. *Multi-agent Systems And Applications IV : 4th International Central and Eastern European Conference on Multi-Agent Systems, (CEEMAS '05), Budapest, Hungary, September 15-17, 2005*, 2005.
- G. PICARD, M.P. GLEIZES et P. GLIZE : Distributed Frequency Assignment Using Cooperative Self-Organization. *First International Conference on Self-Adaptive and Self-Organizing Systems (SASO'07)*, pages 183–192, 2007.
- G. PICARD et P. GLIZE : Model and Analysis of Local Decision Based on Cooperative Self-Organization for Problem Solving. *Multiagent and Grid Systems*, 2(3):253–265, 2006.
- M. ROGER : Comparing two approaches to dynamic, distributed constraint satisfaction. In *AAMAS '05 : Proceedings of the fourth international joint conference on Autonomous agents and multiagent systems*, pages 1049–1056, New York, NY, USA, 2005. ACM. ISBN 1-59593-093-0.
- B. SELMAN, H. LEVESQUE et D. MITCHELL : A New Method for Solving Hard Satisfiability Problems. *Proc. of the Tenth National Conference on Artificial Intelligence*, pages 440–446, 1992.
- H. VAN BRUSSEL, J. WYNS, P. VALCKENAERS, L. BONGAERTS et P. PEETERS : Reference architecture for holonic manufacturing systems : PROSA. *Computers in Industry*, 37(3):255–274, 1998.
- M. WOOLDRIDGE et N.R. JENNINGS : Agent Theories, Architectures, and Languages : A Survey. *Intelligent Agents*, 22, 1995.
- B. WU : *Manufacturing Systems Design & Analysis : Context & Techniques*. Springer, 1994.
- M. YOKOO : Distributed constraint satisfaction : foundations of cooperation in multi-agent systems. *Springer Series On Agent Technology*, page 143, 2001.
- M. YOKOO, E.H. DURFEE, T. ISHIDA et K. KUWABARA : The Distributed Constraint Satisfaction Problem : Formalization and Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 10(5):673–685, 1998.