N° d'ordre :02/2012-A/INF

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE DES SCIENCES ET DE LA TECHNOLOGIE HOUARI BOUMEDIENE

Faculté d'électronique et d'informatique

THESE

Présentée pour l'obtention du diplôme de DOCTORAT

En : Informatique

Option : Intelligence artificielle

Par : IBRI SARAH

Sujet

Techniques de l'optimisation distribuée: application au problème de gestion intégrée des véhicules d'urgence

Soutenu le 15/02/2012 devant le jury composé de :

Mr Larabi Slimane	Professeur USTHB	Président
Mme Drias Habiba	Professeur USTHB	Directrice de thèse
Mme Habbas Zineb	Professeur Univ Metz	Examinateur
Mr Khadraoui Djamel	Professeur Univ Luxembourg	Examinateur
Mme Moussaoui Samira	Maître de conférence USTHB	Examinateur
Mr Djennouri Djamel	Maître de recherche CERIST	Examinateur
Mr Nourelfath Mustapha	Professeur Univ Laval	Invité

à la mémoire de mon cher père, ma source
d'encouragement infinie

à ma chère mère, ma source
d'amour infinie

Remerciements

Je tiens tout d'abord à remercier sincèrement Professeur Habiba Drias pour avoir accepté de diriger cette thèse. Je la remercie pour avoir mis à ma disposition son expérience, pour ses encouragements et son aide.

Je remercie aussi Professeur Mustapha Nourelfath pour m'avoir accueilli au sein de son équipe du CIRRELT, pour m'avoir proposé une application qui m'a beaucoup intéressée : les systèmes de gestion des urgences. Je le remercie également pour ses précieux conseils qui ont énormément contribué à la réalisation de cette thèse.

Que Professeur Slimane Larabi reçoive mes plus sincères remerciements pour l'honneur qu'il m'a fait en présidant ce jury.
Un témoignage de ma profonde reconnaissance s'adresse à Professeur Zineb Habbas, Professeur Djamel Khadraoui, Docteur Samira Moussaoui et Docteur Djamel Djennouri pour avoir accepté d'examiner ma thèse et pour leurs remarques et critiques constructives.

Un grand merci à toute personne ayant contribué de près ou de loin à la réalisation de ce travail.

Résumé

La coordination de plusieurs agents pour la résolution distribuée coopérative d'un même problème d'optimisation reste une problématique d'actualité qui suscite l'intérêt des chercheurs à cause de la multiplicité de ses domaines d'application. Dans cette thèse nous nous intéressons aux différentes possibilités qu'offre le domaine de l'intelligence artificielle distribuée pour la résolution de ce genre de problèmes. Plus précisément, nous traitons le cas de la gestion intégrée des véhicules d'urgence qui possède les caractéristiques principales des applications traitées par l'intelligence artificielle distribuée.

Dans un premier temps nous commençons par le traitement du système de gestion centralisée des véhicules d'urgence avec un double objectif, celui de l'étude de l'intégration du problème de dispatching des véhicules avec le problème de couverture des zones dans un même modèle, mais aussi sa résolution par l'intelligence artificielle parallèle en proposant un algorithme parallèle basé sur l'hybridation de la recherche tabou et les algorithmes d'optimisation par les colonies de fourmis.

Nous passons ensuite à l'étude du système distribué du même problème. Pour cela, nous proposons un système multi-agents qui modélise les différentes entités et les interactions entre elles. L'objectif dans cette partie est de trouver une méthode d'optimisation et de coordination des agents de sorte que les solutions distribuées obtenues soient comparables aux solutions du système centralisé tout en gardant les données, les traitements ainsi que la prise de décision décentralisée. Dans ce sens nous proposons deux types d'approches : dans la première approche la solution est basée sur un mécanisme d'enchères qui sous-tend un processus de coordination implicite dans lequel il n'existe aucune entité qui possède l'information nécessaire au contrôle des contraintes ce qui explique en partie l'écart obtenu entre cette approche et les solutions centralisées. Dans la deuxième approche nous proposons un protocole de coordination impliquant un processus de recherche basé sur un échange direct entre les agents et s'inspirant de la méthode de recherche synchrone avec déviations, permettant ainsi l'amélioration de la performance. Le protocole est conçu pour la classe de problèmes d'optimisation distribuée ayant des contraintes globales dans leur modèle.

L'évaluation des différentes solutions proposées dans ce travail est faite par un simulateur à événements discrets sur des instances artificielles que nous avons préparées en utilisant une carte avec un réseau routier réel et un outil de géolocalisation et de calcul des itinéraires et des temps de déplacement.

Abstract

Coordinating several agents to solve the same optimization problem in a distributed cooperative way is still a topical issue that interest researchers because of the multiplicity of its applications. In this thesis, we aim to study the different possibilities that the distributed artificial intelligence offers to solve these kinds of problems. More exactly, we take as example the problem of the integrated emergency vehicle management which has the main characteristics of the distributed artificial intelligence applications.

First, we begin by the study of the centralized emergency vehicle management problem with a twofold objective: Studying the integration of vehicle dispatching with the zone covering problem in the same model; and its solution by the parallel artificial intelligence using a parallel algorithm based on a combination of tabu search and ant colony optimization metaheuristics. Then, we turn to the study of the distributed system of the same problem. We propose a multi-agent system for a distributed modelling of the different entities of the problem with the interaction between them. The aim in this part of work is to find an agents' optimization and coordination method so that the obtained distributed solutions are comparables to the solutions of the corresponding centralized system, while keeping data, processing and decision making decentralized. For this aim we propose two kinds of approaches: in the first one, the solution is based on an auction mechanism that underlies the implicit coordination process in which there isn't any entity that has the required information to control the constraints between agents. This is partly the reason of the gap between this approach and the centralized solutions. In the second approach, we propose a coordination protocol that implies a search process based on a direct exchange between agents. It is inspired by the synchronous discrepancy search method allowing the performance' improvement. The protocol is designed for the class of problems with global constraints in their models.

The evaluation of the different solutions proposed in this work is performed by a discrete event based simulator on artificial instances that we prepared using a map with real road network and a geolocalization tool to compute itineraries and travelling times.

Table des matières

Page

Dédicaces ii
Remerciements iii
Résumé iv
Abstract v
Table des matières vi
Liste des figures viii
Liste des tableaux ix

I. Introduction générale 10
I.1 Problèmes distribués : exemples et caractéristiques 10
I.2 Solutions distribuées : les trois axes de l'IAD 12
I.3 Un exemple typique : la gestion des véhicules d'urgence 13
I.4 Objectif de la thèse 14
I.5 Contributions et organisation de la thèse 14

II. L'intelligence artificielle parallèle 17
II.1 Les architectures parallèles 17
II.2 L'algorithmique parallèle 19
II.2.1 Processus 20
II.2.2 Synchronisation des processus 21
II.2.1 Mesures de performance 21
II.3 Les méta-heuristiques parallèles 22
II.3.1 Les stratégies de parallelisation des méta-heuristiques 23
II.3.2 La recherche tabou parallèle 25
II.3.3 L'ACO parallèle 26
II.4 Conclusion 27

III. Les systèmes multi-agents & la résolution distribuée des problèmes 29
III.1 Les systèmes multi-agents : SMA 30
III.1.1 Notion d'agent 30
III.1.2 Systèmes multi-agents 30
III.1.3 Interactions entre agents 31
III.1.4 Le protocole Contract Net 33

III.1.5 Les enchères 33
III.2 La résolution distribuée des problèmes : RDP 35
III.2.1 La planification distribuée 35
III.2.2 La satisfaction distribuée des contraintes 37
III.2.3 L'optimisation distribuée 39
III.3 Conclusion 45

IV. Systèmes de gestion des véhicules d'urgences 47
IV.1 Dispatching 49
IV.2 Couverture 51
IV.2.1 Couverture statique 51
IV.2.2 Couverture dynamique 54
IV.3 Conclusion 56

V. Le système de gestion des véhicules d'urgences centralisé 57
V.1 Gestion intégrée des véhicules d'urgences : "*DISCOV*" 57
V.1.1 Modèle mathématique 58
V.1.2 Etude de l'intégration 61
V.2 Solution centralisée : Antabu 65
V.2.1 Antabu séquentiel 65
V.2.2 Antabu parallèle 70
V.3 Conclusion 77

VI. Le système de gestion des véhicules d'urgences distribué 78
VI.1 Gestion intégrée des véhicules d'urgences multi-agents : "*MA-DISCOV*" 79
VI.1.1 Les agents 79
VI.2 Solutions distribuées 80
VI.2.1 Coordination par le mecanisme des enchères 80
VI.2.2 Coordination par communication directe 87
VI.3 Conclusion 101

VII. Conclusion & perspectives 102
Bibliographie 105
Récapitulatif de la production scientifique 112
Annexe : L'environnement JADE 113

Liste des figures

Figure 5.1	:	Etude de l'intégration : Temps de réponse moyen pour dataset1	63
Figure 5.2	:	Etude de l'intégration : Temps de réponse moyen pour dataset2	63
Figure 5.3	:	Etude de l'intégration : Pourcentage de zones non couvertes pour dataset1	63
Figure 5.4	:	Etude de l'intégration : Temps moyen de non couverture pour dataset1	64
Figure 5.5	:	Etude de l'intégration : Pourcentage de zones non couvertes pour dataset2	64
Figure 5.6	:	Etude de l'intégration : Temps moyen de non couverture pour dataset2	64
Figure 5.7	:	L'algorithme AnTabu	66
Figure 5.8	:	La procédure tabu_search	69
Figure 5.9	:	AnTabu parallèle	71
Figure 5.10	:	Tabou parallèle synchrone	72
Figure 5.11	:	Impact du nombre de processus sur le t emps d'exécution	73
Figure 5.12	:	Impact du nombre de processus sur la qualité de la solution	74
Figure 6.1	:	MA-DISCOV	81
Figure 6.2	:	Diagramme de séquence	81
Figure 6.3	:	Mécanisme des enchères : Approche1	82
Figure 6.4	:	Mécanisme des enchères : Comparaison de la fonction objectif pour dataset1	85
Figure 6.5	:	Mécanisme des enchères : Comparaison de la fonction objectif pour dataset2	85
Figure 6.6	:	Mécanisme des enchères : Comparaison du temps de réponse pour dataset1	86
Figure 6.7	:	Mécanisme des enchères : Comparaison du temps de réponse pour dataset2	86
Figure 6.8	:	SyncLDS – Pseudocode	88
Figure 6.9	:	Procedure de recherche locale	89
Figure 6.10	:	SynchLDS pour instance1	89
Figure 6.11	:	SynchLDS pour instance2	89
Figure 6.12	:	SynchLDS pour instance3	90
Figure 6.13	:	SynchLDS pour instance4	90
Figure 6.14	:	Liste des solutions	91
Figure 6.15	:	Liste des références	91
Figure 6.16	:	Phase de calcul	93

Figure 6.17 : Phase de communication 93
Figure 6.18 : PCGD pour instance1 96
Figure 6.19 : PCGD pour instance2 96
Figure 6.20 : PCGD pour instance3 96
Figure 6.21 : PCGD pour instance4 96
Figure 6.22 : Comparaison de fonction objectif 97
Figure 6.23 : La meilleure fonction objectif pour un temps maximum d'exécution de 60 secondes 97
Figure 6.24 : Nombre de messages 98
Figure 6.25 : Taille de la liste 99
Figure 6.26 : Centralisé, enchères et PCGD pour dataset1 99
Figure 6.27 : Centralisé, enchères et PCGD pour dataset2 100

Liste des tableaux

Tableau 5.1 : L'effet de l'intégration 65
Tableau 5.2 : Moment de la communication inter-processus 74
Tableau 5.3 : Fréquence de la communication inter-processus 75
Tableau 5.4 : Comparaison des deux stratégies parallèles et l'algorithme séquentiel 76
Tableau 5.5 : Comparaison de l'algorithme séquentiel et le parallèle asynchrone pour le même temps d'exécution 76
Tableau 6.1 : Mécanisme des enchères : Résultats de la couverture des zones. 87
Tableau 6.2 : Différence par rapport à la centralisée 100

1 Introduction

Par définition, l'intelligence n'est pas seulement les facultés de réfléchir, comprendre, apprendre et analyser ; l'intelligence c'est aussi la capacité de communiquer, coopérer avec les autres et s'adapter aux changements de l'environnement.

De même, dans beaucoup de systèmes informatiques conçus pour la résolution des problèmes complexes, il ne suffit pas d'avoir un seul calculateur doté des techniques de l'intelligence artificielle "IA" car l'émergence des meilleures solutions pour ces systèmes est le résultat de l'effort collectif d'un ensemble de noeuds de calcul intelligents. On parle donc de l'Intelligence Artificielle Distribuée "IAD".

I.1 Problèmes distribués : exemples et caractéristiques

Les problèmes distribués sont présents dans multiples applications de la vie quotidienne liées à différents domaines.

En industrie par exemple on trouve les problèmes de l'ordonnancement distribué comme le cas du job shop qui consiste à coordonner les machines d'un ou de plusieurs ateliers manufacturiers afin d'ordonnancer leurs activités sans conflit et à moindre coût. Dans ce problème, des ressources en un nombre/capacité limité(e) sont partagées par les activités, ce qui nécessite des mécanismes appropriés permettant l'allocation distribuée des ressources [92]. Dans le domaine des télécommunications, les méthodes de l'IAD sont souvent utilisées pour la gestion et le contrôle distribué des réseaux d'ordinateurs pour la détection, le diagnostique, la réparation et la maintenance des congestions et de tous autres échecs matériels dans le système [97][98]

Dans le domaine du transport les problèmes liés aux systèmes de contrôle distribué du trafic routier ou aérien ainsi que ceux liés à la planification distribuée des opérations logistiques pour l'acheminement des marchandises peuvent se baser sur les techniques de l'IAD pour faire coopérer et coordonner les différents éléments distribués de ces systèmes [34][91][45].

Les problèmes distribués sont également présents en robotique pour la planification et la coordination des robots autonomes mobiles. Et aussi dans le domaine de la vision par ordinateur pour la reconnaissance distribuée des images et des formes.

Les problèmes distribués sont en général caractérisés par:

- Un ensemble d'entités distribuées logiquement ou géographiquement.

- Les entités possèdent des compétences complémentaires nécessaires à la résolution du problème en question.
- Des données/ressources distribuées. Elles peuvent être partagées ou non partagées.
- Un contrôle décentralisé.
- Des traitements distribués et souvent asynchrones.
- Des changements fréquents de l'environnement.

D'un autre côté, les études sur l'IAD montrent que ce domaine sert à [41] :

- exploiter le parallélisme pour une résolution plus rapide des problèmes. Le gain en temps dépend principalement du degré de parallélisme inhérent au problème à résoudre et de l'architecture parallèle utilisée dans sa résolution.
  Exemples : le traitement du langage naturel, les problèmes de vision, les algorithmes de recherche évolutionnaire, les réseaux de neurones…

- faire coopérer les compétences ou toutes autres capacités nécessaires à la résolution d'un problème donné et qui peuvent être distribuées de façon inhérente.
  En ingénierie concurrente par exemple, des agents spécialisés dans des domaines différents doivent concevoir individuellement des composants et les combiner en un produit collectif. Le produit qu'ils doivent formuler doit satisfaire tous les besoins pour permettre la fusion des solutions individuelles (assemblages des différents composants).

- coordonner les données distribuées du problème.
  Exemple : dans le problème de contrôle distribué des véhicules "distributed vehicle monitoring", pour contrôler le mouvement des véhicules sur une région large, plusieurs capteurs *"sensor"* distribués sont utilisés, car le contrôle complet de la région ne peut se faire d'un seul endroit. Ce problème peut être traité de façon centralisée : chaque capteur envoie ses données brutes (image non traitée) à un site central où l'interprétation du résultat final (la formulation d'une carte complète) sera effectuée. Mais cette méthode peut produire beaucoup de communications inutiles. L'approche distribuée, par contre, permet aux capteurs de faire des interprétations locales, de les coordonner ensuite de façon distribuée pour former le résultat global.

- coordonner les résultats de la planification ou de la résolution d'un problème donné lorsque celles-ci doivent être distribuées pour être utilisées par plusieurs entités.
  Exemple : le problème de livraison distribuée consiste à livrer un ensemble d'objets à des emplacements différents, en utilisant un ensemble de ressources partagées (véhicules). La

formulation des plans que les véhicules vont exécuter peut aussi se faire de façon centralisée par un coordinateur central, et tous les changements de l'environnement doivent être communiqués au coordinateur pour réviser les plans. Mais il est préférable que les véhicules, modifient leurs plans unilatéralement de façon distribuée afin de limiter la communication avec le coordinateur.

- ajouter de la flexibilité et de la distribution au calcul ainsi que de l'hétérogénéité dans la réalisation des systèmes pour faciliter l'ingénierie logicielle qui est impliquée dans la création des modules autonomes conçus par plusieurs personnes, différentes équipes et différentes compagnies. [44]

I.2 Solutions distribuées : les trois axes de l'IAD

Les premiers travaux réalisés dans le domaine de l'IAD étaient concentrés sur la mise en valeur et l'exploitation de la puissance des réseaux d'ordinateurs pour la résolution parallèle des problèmes de l'IA ; ensuite les chercheurs se sont tournés vers l'exploitation de la robustesse des multiples sources de compétences, d'expertises et de perspectives. Ainsi, l'IAD recouvre principalement les trois axes suivants :

1. Intelligence artificielle parallèle "IAP" : On ne s'intéresse pas à la nature du raisonnement ni à l'intelligence des comportements mais à l'amélioration des performances des systèmes de l'IA par la proposition de langages concurrents, d'architectures parallèles et des algorithmes parallèles performants pour la résolution des problèmes de l'intelligence artificielle. L'objectif est d'accélérer les méthodes classiques de l'IA.

2. Systèmes multi-agents "SMA" : consistent en un ensemble d'entités autonomes et relativement indépendantes appelées "agents" qui peuvent avoir une tâche commune ou des objectifs individuels à accomplir et qui ne nécessitent aucun mécanisme de contrôle global. Ainsi, les agents doivent eux même raisonner sur les moyens pour coordonner leurs actions, connaissances, solutions et plans. Les SMAs sont des architectures ouvertes car elles permettent l'ajout et la suppression d'agents même au cours du fonctionnement du système ce qui les rend compatibles à l'évolution et l'adaptation nécessaire aux systèmes réels [44].

3. Résolution coopérative distribuée des problèmes "RDP": c'est le domaine où l'on développe des systèmes distribués coopératifs pour résoudre un problème donné. Un réseau d'entités de calcul semi autonomes, adaptables et généralement dépendantes les unes des autres (appelés parfois agents) travaillent ensemble pour résoudre un problème qui nécessite un effort collectif, car une entité toute seule est incapable d'accomplir la tâche. Les

principales questions traitées par la RDP sont la décomposition d'un problème et son affectation à l'ensemble des entités distribuées et coopérantes, le partage des connaissances et des données entre les entités, le processus de résolution, de coordination des entités et la synthèse des résultats.
Les trois axes de l'IAD seront développés en détail dans les deux premiers chapitres de ce mémoire.

I.3 Un exemple typique : la gestion des véhicules d'urgence

Dans cette thèse nous traitons un cas typique des problèmes distribués. Il s'agit de la gestion des véhicules d'urgence, qui consiste à contrôler un ensemble de véhicules d'urgence tels que les ambulances, les véhicules de la protection civile et les véhicules de police dans le but de garantir un service satisfaisant pour la protection de la population. En général, le contrôle des véhicules d'urgence inclut leur distribution, le choix de leur emplacement et les services qui leurs sont attribués ainsi que le suivi de leur déplacement.
Les systèmes de gestion des véhicules d'urgence sont sujets à des environnements dynamiques et très peu stables à cause de l'imprévisibilité des événements d'urgence qui peuvent survenir, leurs types, leur origine et leur fréquence ; et aussi les incidents pouvant arriver au cours des déplacements des véhicules.
Du fait que les systèmes de gestion des véhicules d'urgence soient un élément clé dans les services des urgences et de la protection publique dont l'objectif est d'assurer la sécurité des personnes et la protection des biens et de l'environnement ; ils sont soumis à des contraintes de temps draconiennes afin de pouvoir sauver les vies humaines et minimiser les dégâts matériels et ainsi contribuer à l'efficacité des services.
Tous les facteurs cités ci-dessus sont ajoutés au fait que les éléments qui constituent les systèmes de gestion des véhicules d'urgence sont géographiquement distribués. Ceci concerne les appels d'urgence, les stations des véhicules d'urgence ainsi que les véhicules eux mêmes. Par conséquent, les données, la prise de décision, la planification ainsi que l'exécution sont toutes distribuées et requièrent des mécanismes appropriés pour gérer et coordonner efficacement les différents composants du système et prendre les décisions nécessaires de manière distribuée.
Dans les systèmes de gestion des véhicules d'urgence, deux questions majeures se posent. La première concerne le choix des véhicules les plus adéquats pour répondre rapidement aux urgences quand elles arrivent : "*dispatching*"; et la deuxième consiste à trouver les bons

emplacements aux véhicules libres afin de pouvoir prendre en charge les futures urgences dans les meilleurs délais : "*couverture*". Ces deux objectifs sont complémentaires et les travaux qui les ont traités montrent la complexité de leur résolution [57] [78].

I.4 Objectif de la thèse

Dans cette thèse nous nous intéressons à l'étude des diverses solutions du domaine de l'IAD pour la résolution des problèmes de l'optimisation distribuée. Plus précisément, nous proposons des approches distribuées comparables aux solutions centralisées et qui sont applicables aux systèmes où les données sont distribuées, la prise de décisions est décentralisée, un objectif global doit être optimisé et des contraintes globales entre les différents composants peuvent exister.

L'application de ces approches sera illustrée par le problème de gestion des véhicules d'urgence qui, comme nous l'avons expliqué plus haut, est un bon candidat pour représenter cette classe de problèmes.

Notre objectif est donc double : D'un côté, nous cherchons des méthodes d'optimisation et de coordination pour les systèmes distribués permettant d'obtenir des performances comparables aux systèmes centralisés où toutes les données et les traitements sont disponibles dans un nœud central. D'un autre côté, nous contribuons à l'étude du problème de gestion des véhicules d'urgence commençant par sa modélisation qui intègre le dispatching et la couverture ainsi que sa résolution centralisée et distribuée par les approches que nous proposons.

L'étude du problème de gestion des véhicules d'urgence dans des conditions proches des systèmes réels nécessite sa simulation afin de reproduire les effets de l'environnement caractérisé par l'ensemble des événements externes. C'est pour cette raison que la réalisation d'un simulateur à événements discrets pour les deux versions centralisée et distribuée fait aussi partie de nos objectifs.

I.5 Contributions et organisation de la thèse

Dans la première étape de ce travail nous proposons un modèle pour le problème de gestion intégrée des véhicules d'urgence. Le modèle proposé considère l'optimisation des problèmes de dispatching et de couverture conjointement et utilise la notion de "*prepardness"* pour mesurer la couverture des zones. Nous considérons dans cette partie la version centralisée et nous procédons à sa résolution séquentielle par un algorithme basé sur la recherche tabou et

l'optimisation par colonies de fourmis que nous appelons *Antabu*. Cette étape va nous nous permettre plus loin dans cette thèse d'estimer l'effet de la décentralisation sur l'optimisation globale, d'évaluer et d'améliorer les approches distribuées proposées. La contribution suivante consiste à paralléliser l'algorithme *Antabu* en proposant une approche synchrone et une autre asynchrone, toutes les deux basées sur l'évaluation parallèle du voisinage ce qui permet d'accélérer considérablement l'algorithme.

Nous passons ensuite à l'étude du système distribué dans lequel les données, les traitements ainsi que la prise de décision sont décentralisés. Nous proposons une architecture multi-agents pour modéliser les différentes entités du système avec l'objectif de trouver une méthode pour coordonner les différents agents de manière à ce que les solutions distribuées obtenues par les agents soient comparables, en terme de fonction objectif, à celles obtenues par la résolution centralisée du même problème.

Pour optimiser le modèle avec cette architecture nous avons proposé une approche inspirée par le mécanisme des enchères pour optimiser de façon implicite le système complet. Elle consiste à distribuer les données sur les agents stations qui ont également le rôle d'optimiser localement le modèle, ensuite faire des offres à l'agent zone ; ce dernier a la responsabilité de décider des offres à accepter/rejeter. Alors que cette méthode réduit le coût de communication dans le système, elle ne permet pas une coordination et une évaluation globale de la fonction objectif et des contraintes ce qui constitue une limite importante pour cette d'approche.

La dernière contribution dans cette thèse consiste à proposer un protocole de coordination globale distribuée "PCGD" pour une résolution plus efficace qui diffère de la solution des enchères d'un point de vue organisationnel et conceptuel puisque la prise de décision est effectuée grâce à un mécanisme de coordination qui fait coopérer les stations au lieu d'affecter cette tâche à la zone. La coopération se fait donc de façon explicite en faisant communiquer les stations entre elles directement. Nous proposons aussi dans cette partie une adaptation du protocole de recherche avec déviations synchrone "SynchLDS" à notre problème et nous le comparons avec notre protocole PCGD conçu pour pallier aux points faibles du SynchLDS.

Dans les trois chapitres qui suivent, nous présentons un état de l'art sur l'IAD : pour chacun de ses axes les principaux concepts et méthodes sont décris; nous y présentons également une étude bibliographique sur les systèmes de gestion des véhicules d'urgence où nous présentons en détail les deux principales problématiques à savoir le dispatching et la couverture.

Dans le chapitre 5, nous décrivons le problème de gestion intégrée des véhicules d'urgence étudié avec le modèle que nous proposons pour sa version intégrée. Nous présentons également la solution centralisée basée sur l'algorithme Antabu avec les deux stratégies de parallèlisation de l'algorithme.

Le chapitre 6 décrit l'architecture multi-agents que nous proposons pour le système distribué avec le simulateur multi agents, ainsi que l'approche d'optimisation basée sur les enchères et son application au problème distribué. Dans le même chapitre nous décrivons une adaptation de la recherche à base de déviations synchrone à notre problème, et nous montrons l'effet de l'ordre de priorité des agents sur la qualité des solutions obtenues. Ensuite nous présentons le nouveau protocole de coordination globale distribuée "*PCGD*" que nous proposons à cet effet ainsi que son adaptation au problème de gestion intégrée des véhicules d'urgence.

Tout au long des deux derniers chapitres des résultats numériques et des comparaisons des différentes approches sont fournis. Nous terminons cette thèse par les conclusions et les perspectives envisageables pour ce travail.

II L'intelligence Artificielle Parallèle "IAP"

Le domaine de l'IAP s'intéresse à la fois au développement des architectures parallèles ainsi qu'à la programmation parallèle par la proposition des algorithmes et des langages de programmation appropriés aux problèmes de l'IA.
Après une brève introduction sur les architectures parallèles, nous présentons dans ce chapitre les principaux concepts liés aux algorithmes parallèles et plus particulièrement aux méthodes de parallélisation des meta-heuristiques vues dans la littérature.

Parmi les applications dont la réalisation fait appel au calcul parallèle on trouve les systèmes d'exploitation, l'algorithmique numérique utilisée souvent dans les applications avioniques, le traitement et la synthèse d'images, la biologie moléculaire, l'analyse sismique, les applications en intelligence artificielle qui doivent en général traiter des bases de données (ou des bases de connaissances) de tailles considérables en des temps raisonnables ou qui font appel aux techniques heuristiques pour des buts d'optimisation ainsi que d'autres applications qui doivent respecter des contraintes temporelles absolument draconiennes comme l'ordonnancement des tâches, la compréhension de la parole et la planification du déplacement des robots. Pour toutes ces applications le calcul parallèle est un bon moyen d'obtenir une performance acceptable [4].

II.1 Les architectures parallèles

Les machines de Von Neumann, étant séquentielles, sont une base inadéquate pour le développement des systèmes d'intelligence artificielle à grande échelle. De même les langages conçus pour ces architectures ne sont pas adéquats car ils sont séquentiels par nature et restreignent la possibilité de concurrence.

D'un autre côté, le ralentissement du gain en vitesse des circuits électroniques a conduit les chercheurs vers de nouvelles techniques pour accélérer les calculs. Ces techniques consistent à faire fonctionner plusieurs processeurs pour réaliser un même travail.

Sur les monoprocesseurs, le parallélisme reste très limité et caché à l'utilisateur. Il peut se faire grâce à plusieurs unités fonctionnelles (addition, multiplication, division, décalage…) ou bien grâce à la présence d'unités arithmétiques et logiques parallèles ou encore par multiprogrammation et temps partagé. Ces deux derniers consistent à utiliser en parallèle des

unités d'entrée/sortie et le processeur, le temps partagé consiste à limiter la durée d'occupation des unités afin d'améliorer le temps d'exécution [10].

Le multi-tâches quant à lui fait référence aux capacités d'un système d'exploitation de lancer plusieurs applications en même temps et de passer rapidement d'une application à une autre donnant l'impression que les différentes applications s'exécutent simultanément. Ces dernières années, la technologie du multithreading a été introduite pour améliorer le support des codes multithread. Un processeur équipé en multithreading prétend faire office de plusieurs processeurs "logiques" permettant au système d'exploitation de prévoir plusieurs threads simultanément. Le multithreading offre un temps de réaction et de réponse amélioré. Il étend le principe du multitâches aux applications de sorte qu'une même application est divisée en threads (processus) individuels pouvant s'exécuter en parallèle. Alors le système d'exploitation divise le temps de traitement non seulement entre des différentes applications mais aussi entre les threads d'une même application.

Dans les architectures parallèles plusieurs processeurs et unités de contrôle sont disponibles. Différents critères sont utilisés pour décrire et classifier les machines parallèles tel que le nombre de processeurs, la présence ou non d'un mécanisme de contrôle global, le mode de synchronisation des opérations et les topologies d'interconnexion des processus.

La classification la plus connue est celle de Flynn qui se base sur le type de flux des données et des instructions. Une autre classification est donnée dans [10] considèrent trois grandes classes :

1. Les architectures synchrones telles que les architectures pipeline et vectorielles qui sont des architectures MISD (multiple instruction single data), les architectures SIMD (single instruction multiple data) et les architectures systoliques.
2. les architectures asynchrones MIMD (multiples instructions multiple data) pour un parallélisme partagé ou distribué
3. les architectures à flots de données (Data Flow) où l'exécution d'un programme est liée aux contraintes de dépendances entre les données.

L'inconvénient des systèmes multiprocesseurs est qu'ils sont très coûteux parce qu'ils nécessitent plusieurs circuits. La nouvelle tendance pour faire face à cet obstacle économique sont les processeurs double-cœurs et multi-coeurs qui consistent en un ensemble de processeurs occupant un seul et unique circuit. Contrairement aux systèmes multiprocesseurs

où certaines ressources comme le cache L2/L3 et le bus frontal sont dupliquées, dans les systèmes multicoeurs ces ressources sont partagées par les processeurs.
Dans la technologie des grilles de calcul, on trouve de nombreuses ressources informatiques hétérogènes, géographiquement éloignées et mises en réseau permettant d'exploiter les puissances de calcul de toutes ces ressources en les combinant, créant en quelque sorte un supercalculateur virtuel extrêmement puissant. Ce dernier met aussi en commun les applications, les données et la capacité de stockage. Ces infrastructures sont utilisées pour certaines applications aux temps de calcul très longs permettant d'obtenir des résultats en un temps raisonnable.

II.2 L'algorithmique parallèle

Contrairement au concept séquentiel, le parallélisme consiste à diminuer le temps de calcul en exécutant plusieurs petites tâches en même temps au lieu d'exécuter l'ensemble de ces tâches séquentiellement sur un processeur plus rapide.

La parallélisation d'un algorithme consiste donc à le partitionner en plusieurs tâches. Une tâche est définie comme un ensemble d'instructions s'exécutant de façon séquentielle. La taille d'une tâche peut aller de la simple opération arithmétique ou logique à une application entière. Après l'étape de décomposition, il faut déterminer les relations de précédence (contraintes temporelles d'exécution) entre les tâches et affecter les tâches aux processeurs tout en respectant les contraintes d'accès aux données et de synchronisation. L'implémentation consiste à exécuter les instructions sur une machine parallèle, tout en gérant les échanges de données et la communication. La programmation des tâches indépendantes peut se faire au niveau du système d'exploitation (parallélisation automatique de programmes, compilateurs intelligents) ou au niveau algorithmique [10].

Il peut exister plusieurs façons de paralléliser un algorithme suivant le type de découpage en tâches, d'accès aux données et l'affectation des tâches aux processeurs. Selon la machine cible, une méthode de parallélisation peut être bonne ou pas. Sur une machine SIMD par exemple on ne peut pas exécuter un algorithme asynchrone alors que sur une machine MIMD, les problèmes de synchronisation n'ont pas d'influence sur la structure de l'algorithme.

La granularité d'une décomposition est une mesure du rapport entre le temps moyen d'exécution d'une tâche et le temps total de l'algorithme [85] elle peut être fine, moyenne ou grosse. La décomposition fine n'est pas réaliste car elle engendre des coûts de communication et d'accès mémoires prohibitifs.

Avant de pouvoir affecter les tâches aux processeurs il est nécessaire d'établir les contraintes de précédence entre les tâches. Ceci peut être modélisé par un graphe de précédence qui traduit le parallélisme interne à l'algorithme. Il consiste en un ensemble de tâches (les sommets du graphe) caractérisées par une durée d'exécution et éventuellement une date au plus tôt ou une date au plus tard. Les arcs du graphe représentent les liens de précédence entre les tâches. Si deux tâches sont liées par une relation de précédence, leur exécution doit être séquentielle, sinon elles peuvent s'exécuter en parallèle.

Le problème d'ordonnancement des tâches sur les architectures parallèles est statique dans le cas où toute l'information est connue a priori, donc avant le début de l'exécution. Quand l'information n'est pas connue à priori ou change au cours de l'exécution, le problème d'ordonnancement devient dynamique.

Le problème fondamental de la parallelisation d'une méthode est de trouver le degré maximal de parallélisme qu'elle renferme. Il faut de plus que l'algorithme d'ordonnancement trouvé soit descriptible et implantable sur une machine existante (ou au moins simulable si une telle machine n'est pas disponible) [10].

Les principaux facteurs de choix de la meilleure décomposition sont le nombre de processeurs, le rapport entre le temps de calcul et le temps de communication, les accès mémoire ainsi que la nature du problème.

II.2.1 Processus

Un processus est une tâche en cours d'exécution. Ce concept est apparu initialement dans les systèmes informatiques pour décrire l'exécution d'un programme séquentiel, avec ses données, par un processeur. Il est inhérent au calcul parallèle car ce dernier est le travail d'un ensemble de processus, chacun avec son propre algorithme, et parfois exécutant le même algorithme sur des données différentes.

Les processus peuvent être indépendants et activés simultanément pour les exécuter parallèlement. Le programme se termine avec l'achèvement de tous ses processus. Dans le cas contraire ils doivent échanger des informations. La communication peut se faire *par partage de mémoire* quand certains processus mettent à jour des données que d'autres processus peuvent également consulter et modifier. Pour conserver la cohérence de ces données, il est nécessaire d'en contrôler l'accès par des mécanismes d'exclusion mutuelle.

Elle peut se faire aussi par *échange explicite d'information/données* via des primitives de communication (d'envoi et de réception de messages).

II.2.2 Synchronisation des processus

Nous distinguons deux types de synchronisation [4] :

- *Mode synchrone :* C'est le cas où la communication ne peut être effectuée que lorsque le processus expéditeur est prêt à effectuer l'opération d'envoie au processus destinataire, et que le destinataire lui-même est prêt à recevoir l'information (envoi bloquant et réception bloquante).
- *Mode asynchrone :* Dans ce mode, dés que l'expéditeur est prêt à envoyer un message, il effectue cet envoi, même si le récepteur n'est pas prêt à recevoir le message. Le récepteur devant consommer un message ne se bloque que si celui-ci n'a pas été produit au préalable, dans le cas contraire, il consomme le message et continue son exécution (envoi non bloquant, réception bloquante).

Il existe un troisième mode de synchronisation qui pourrait se rapprocher du traitement d'interruption : Lorsqu'un récepteur demande à recevoir un message et que ce dernier n'est pas prêt, il continue en séquence (envoi non bloquant et réception non bloquante).

Dans un programme parallèle, les processus peuvent être crées statiquement ou dynamiquement. Dans le premier cas les processus sont en nombre fixe et sont crées une fois pour toutes lors de l'initialisation du programme. Alors que dans la création dynamique le nombre de processus présents lors de l'exécution d'un programme peut varier d'une exécution à une autre en fonction des données traitées.

II.2.3 Mesures de performance

- *Facteur d'accélération* S_p : permet de mesurer le gain en temps dû à la parallélisation

  $S_p=T_1/T_p$

  T_1 : est le temps d'exécuter l'algorithme sur un seul processeur.

  T_p : le temps d'exécuter l'algorithme sur p processeurs.

- *L'efficacité E_p:* est l'équivalent d'un rendement. Elle permet de mesurer le taux moyen d'utilisation des processeurs. La parallélisation d'un algorithme est d'autant meilleure que E_p est proche de 1.

  $E_p = S_p / P$

  $E_p \in [0, 1]$

II.3 Les méta-heuristiques parallèles

Les méta-heuristiques sont des algorithmes d'optimisation qui partent d'une solution initiale et cherchent à l'améliorer itérativement. Contrairement aux heuristiques, les métaheuristiques acceptent des mouvements vers des solutions moins bonnes afin d'éviter les optimum locaux. D'autres mécanismes contrôlent l'évolution des meta-heuristiques pour éviter de boucler, pour apprendre des mouvements précédents, des solutions déjà rencontrées et ainsi assurent une recherche plus approfondie. Deux classes principales de méta-heuristiques existent : les métaheuristiques de parcours (de voisinage) et les méta-heuristiques de population. Dans la première classe l'algorithme fait évoluer une seule solution dans l'espace de recherche à chaque itération en passant d'une solution vers sa voisine. Dans cette classe on trouve : le recuit simulé, la recherche tabou, la recherche locale guidée, Grasp, …

Les méta-heuristiques basées sur des populations, manipulent plusieurs solutions à chaque itération pour pouvoir explorer différentes régions de l'espace de recherche simultanément. Les méthodes les plus connues de cette classe sont les algorithmes génétiques, la recherche dispersée, les colonies de fourmis, l'optimisation par essaims de particules…

D'autres critères de classification tels que la façon dont la solution est obtenue à chaque itération (constructive ou évolutive), l'utilisation de mémoire (avec ou sans mémoire) sont également utilisés pour les distinguer.

Dans la parallelisation des méta-heuristiques, la décomposition peut concerner l'algorithme, l'instance du problème ou bien la structure du problème.

- Le premier cas appelé **parallélisme fonctionnel** consiste en un ensemble de tâches s'exécutant en parallèle et pouvant s'échanger des informations.
- Le second appelé **parallélisme de données** ou **décomposition de domaine** consiste à décomposer l'espace de recherche et utiliser une certaine méthodologie pour aborder le problème dans chaque composant de l'espace de recherche.

- Le troisième cas consiste à **décomposer le problème** suivant des ensembles d'attributs en utilisant des techniques de programmation mathématique ou des heuristiques [20].

Les deux principales approches pour partitionner l'espace de recherche est la décomposition du domaine et la multi recherche. La première décompose l'espace explicitement alors que la seconde le fait implicitement en lançant plusieurs recherches simultanément et en utilisant certaines stratégies pour éviter le chevauchement mais sans le garantir.

II.3.1 Les stratégies de parallélisation des méta-heuristiques

Un état de l'art assez exhaustive sur les méta-heuristiques parallèles a été faite par Crainic [20]. Les auteurs les classifient en quatre groupes de stratégies :

- **Les stratégies de parallélisation de bas niveau**

Dans cette classe de stratégies où on exploite le parallélisme intrinsèque offert par le calcul de base des meta-heuristiques, le calcul principal (les boucles internes de l'algorithme) est décomposé et effectué sur plusieurs processeurs sans que l'espace de recherche ni la logique algorithmique ne soient modifiés.

- **La décomposition du domaine :**

Dans cette approche le problème à résoudre est partitionné en sous problèmes qui sont résolus en parallèle. Dans le problème de routage des véhicules par exemple l'ensemble des clients est divisé en sous ensembles. En décomposant l'espace de recherche, on doit préciser :

- Si les partitions sont strictes ou bien les chevauchements sont permis.
- Si les processus considèrent des solutions partielles ou complètes au problème.
- Si les mouvements appliqués à un sous problème sont restreints à l'espace de recherche local ou bien impliquent les sous-espaces voisins engendrant ainsi un chevauchement indirect entre les sous-problémes.

L'inconvénient du partitionnement strict est qu'il résulte en une partie de l'espace inexplorée car il interdit des mouvements qui impliquent des solutions appartenant à plusieurs sous-ensembles. Une conséquence directe est que la qualité de la solution est moins bonne. De son côté le chevauchement direct ou indirect nécessite un coût supplémentaire pour garder les processus dans leurs propres sous espaces. Une solution

à cet inconvénient consiste à utiliser un partitionnement strict ou un chevauchement très limité et lui ajouter une re-décomposition à des intervalles réguliers et ainsi redémarrer la recherche avec la nouvelle décomposition. A la fin de la recherche une reconstruction de la solution complète est nécessaire.

- **Recherches multiples indépendantes**

Consiste à effectuer plusieurs recherches simultanées partant de la même ou de différentes solutions initiales et utilisant une même ou différente stratégie de recherche. A la fin de toutes les recherches, la meilleure solution obtenue est retenue.

Cette stratégie est simple à implémenter et efficace grâce à la grande puissance de calcul utilisée pour résoudre le problème. Cette méthode a été utilisée dans beaucoup de travaux tels que le tabu search pour le problème de routage de véhicules [81] le GRASP pour le problème d'affectation quadratique [77], le recuit simulé pour le partitionnement de graphes [21][65**]** et le problème de voyageur de commerce [72]. Dans [5] on montre que lorsque le nombre de processeurs augmente, la probabilité de « succès » augmente aussi et le temps moyen nécessaire diminue

- **Recherches multiples coopératives**

C'est une stratégie de recherches multiples où un mécanisme de coopération par partage d'information décrit comment les différentes recherches interagissent. Les éléments clés dans la conception d'une telle stratégie sont la détermination de l'information pertinente à échanger, le moment de l'échange, les processus impliqués dans un échange et la façon d'utiliser l'information reçue par les autres processus. Dans la plus part des travaux, l'information échangée entre les processus est la meilleure solution trouvée. Celle-ci n'est pas toujours un bon choix car souvent elle diminue la diversité de la recherche. Pour cette raison d'autres types d'informations ont été proposés telles que l'envoi de l'optimum local seulement et le choix aléatoire de la solution à envoyer.

D'autres informations sur le contexte de la solution ou de la recherche peuvent remplacer l'échange des meilleures solutions. Par exemple les informations statistiques sur les éléments présents dans les bonnes solutions comme les mémoires à moyen et à long terme dans la recherche tabou ou la matrice de phéromone dans le cas de la méthode ACO.

Les échanges d'information peuvent se faire de façon directe ou indirecte, c'est à dire par envoi de message ou par partage de mémoire (blackboard). Ce dernier cas peut être

centralisé ou distribué. Les communications se font suivant une certaine topologie d'interaction représentée par un graphe de communication spécifiant les processus pouvant avoir des échanges directs. Les communications peuvent être synchrones ou asynchrones.

Quand la solution optimale n'est pas garantie ou que la recherche dans les méthodes exactes est interrompue avant d'atteindre l'optimalité, il devient difficile de déterminer les mesures d'accélération des méthodes parallèles car dans la majorité des stratégies de parallelisation, les versions parallèles et séquentielles donnent des solutions différentes. Ainsi, en parallélisant les heuristiques, il devient aussi important de concevoir des méthodes qui surpassent leur version séquentielle en terme de temps de calcul et idéalement en qualité de solution. La méthode parallèle ne doit pas demander un effort de calcul plus grand que la méthode séquentielle ou doit justifier l'effort par de meilleures qualités de solutions.

II.3.2 La recherche tabou parallèle

Parmi les techniques les plus communes de paralléliser la recherche tabou, il y a l'évaluation parallèle du voisinage. Le voisinage de la solution courante est partagé équitablement entre des processus chacun effectue les mouvements nécessaires avec l'évaluation de la fonction objectif ensuite envoie le meilleur voisin trouvé au processeur maître. Ce dernier, choisit la meilleure solution reçue et la transmet aux processus esclaves pour qu'ils mettent à jour leurs copies locales de la solution. Cette méthode nécessite un temps de communication considérable à cause de la nature itérative de l'algorithme, mais elle est facilement applicable aux problèmes d'optimisation combinatoire [54].

La stratégie de parallélisation de voisinage a été aussi appliquée par (garcia, potvin & rousseau) au problème de routage de véhicules avec fenêtres de temps [49], au problème d'affectation quadratique dans [13], au problème de voyageur de commerce [14] et au problème d'ordonnancement des tâches [79]

Dans [93] une approche de recherches indépendantes a été appliquée au problème de l'affectation quadratique. Chaque recherche utilise des paramètres différents comme la solution initiale ou la taille de la liste tabou. Dans [35] la stratégie des recherches parallèles coopératives a été appliquée au problème de mapping (mapping problem). Durant le processus de recherche les différentes recherches interagissent à des intervalles de temps bien définis, l'interaction consiste à déterminer la recherche la plus fructueuse et transmet son

résultat aux autres processus de recherche. La stratégie des recherches indépendantes a aussi été appliquée dans [94] au problème d'ordonnancement job shop. Dans [19] les auteurs proposent l'approche de mémoire partagée pour la recherche tabou asynchrone. Dans cette méthode des recherches taboues individuelles envoient en mémoire leurs meilleures solutions locales et importent une solution de la mémoire suivant une méthode probabiliste, avant de s'engager dans une phase de diversification.

II.3.3 Parallèlelisation de la methode ACO

Vu le parallélisme inhérent à cette meta-heuristique, beaucoup de travaux ont été réalisé dans ce sens. Dans la majorité des travaux, la stratégie de parallelisation consiste à diviser la colonie de fourmis en sous colonies et de les affecter aux processeurs disponibles. La granularité, l'information à échanger ainsi que la fréquence d'échange sont des facteurs décisifs pour une parallelisation performante.

La granularité est le nombre de fourmis par sous colonie. Elle peut être fine, moyenne ou grosse. Blondi et Bondanza [71] l'ont appliquée au problème de voyageur de commerce, avec une parallelisation fine (fine grained) où chaque sous colonie contient une seule fourmi et est affectée à un processeur différent. Mais cette méthode n'ést pas très performante et devient mauvaise en augmentant la taille du problème à cause du volume élevé de communication. Une autre implémentation coarse grained (gros grain) de la même méthode a donné de meilleurs résultats [9].

Dans [37] deux méthodes de parallélisation de l'ACS (Ant Colony System) suivant ce modèle pour le problème de satisfiabilité maximale ont été proposés. Alors que la première est synchrone et consiste à faire des échanges d'informations par partage de mémoire, la deuxième est une méthode asynchrone permettant une communication explicite. Les résultats expérimentaux ont révélé l'importance et l'influence de la communication entre les sous colonies sur la qualité de la recherche, et ont permis de déduire que, dans le cas d'un algorithme parallèle asynchrone, une communication modérée est plus bénéfique qu'une communication intensifiée.

Dans Bullheimer [9], la stratégie appliquée consiste à effectuer un échange d'information entre les sous colonies de fourmis après k générations. Alors que cette approche est préconisée pour réduire la communication globale considérablement, de bonnes et prometteuses valeurs obtenues durant les itérations locales peuvent être ignorées par les autres sous colonies. Pour cette raison, le rapport des itérations locales/globales doit être choisi rigoureusement.

Kruger et Merkle [71] ont prouvé qu'il est préférable d'échanger seulement les meilleures solutions trouvées que de faire un échange de toutes les matrices de phéromone. Et dans [71] les auteurs ont comparé l'échange de la meilleure solution globale, l'échange circulaire des meilleures solutions locales (un voisinage virtuel est établi entre les sous colonies formant un anneau orienté), l'échange circulaire des migrants (les processus forment un anneau orienté virtuel et seulement les m meilleures fourmis sont échangées) et l'échange circulaire des meilleures solutions locales avec migrants. Ils ont trouvé que les meilleurs résultats sont obtenus lorsque les meilleures solutions locales sont échangées entre les colonies voisines. Ces résultats renforcent ceux obtenus par Kruger, confirmant qu'un échange de petites quantités d'informations est plus utile pour la recherche.

Dans [90] les auteurs ont appliqué la stratégie des recherches parallèles indépendantes, qui est facilement implémentable et permet d'utiliser différents ensembles de paramètres pour chaque recherche et que sous certaines conditions elle donne de meilleurs résultats qu'une seule longue recherche.

II.4 Conclusion

Dans beaucoup de problèmes de l'IA gourmands en temps de calcul et plus particulièrement pour les méthodes de recherche et d'optimisation longues, le calcul parallèle reste la solution par excellence qui garantit l'accélération du calcul et parfois même l'amélioration des qualités des solutions obtenues. Cependant, la clé de voûte dans la parallélisation des méthodes de recherches est la capacité de repérer le parallélisme intrinsèque à l'algorithme ou au problème à résoudre et de bien gérer les échanges d'information entre les processus.

III. Les systèmes multi-agents et la résolution distribuée des problèmes

Comme nous l'avons expliqué dans l'introduction, dans les systèmes où les données/traitements sont distribués, on fait toujours recours aux méthodes issues des systèmes multi agents "SMA" ou de la résolution distribuée coopérative des problèmes "RDP".

La différence entre les SMA et la RDP a été discutée dans [40] où les auteurs proposent trois points de vue : le premier considère que la RDP est un sous ensemble des SMAs càd qu'un SMA est une RDP dans le cas où les agents sont coopératifs, partagent le même but et que le système a été conçu de façon centralisée. Le deuxième point de vue juge que les SMAs servent de base au RDP dans un sens hiérarchique de sorte que le SMA s'occupe des propriétés internes du système et des agents pour les coordonner et les faire interagir alors que la RDP traite les questions liées à l'initialisation, le contrôle et l'interaction afin d'avoir les propriétés et les performances désirées du système. Le troisième point de vue quand à lui considère que les SMAs et la RDP sont complémentaires car les questions posées par les concepteurs SMA (concernant la rationalité des agents, leur préférences de partage des données, leurs capacités…) sont différentes de celles posées par un concepteur RDP (quelle impact aura une certaine méthode de coordination sur la capacité de résoudre un problème donné, comment la communication va affecter la performance du système…) mais qui sont complémentaires pour un même système.

La distinction entre RDP et SMA est toutefois importante afin d'être capable d'identifier la technologie appropriée à adopter, en se basant sur les propriétés exigées par le système en question. Par exemple pour la conception d'un système d'IAD pour le contrôle et l'administration d'un réseau d'ordinateurs où la mesure principale de performance est de minimiser les plaintes des utilisateurs et où l'implémentation est sous le contrôle du concepteur, des techniques empruntées de la RDP peuvent être employées fructueusement. Cependant, pour un système ouvert où les protocoles entre agents sont mal identifiés, permettre l'émergence des modèles d'interaction et des protocoles par les principes de base (préférences, habilités et rationalités d'agents) des SMAs est une approche prometteuse [38].

III.1 Les Systèmes Multi-Agents :

Dans cette section nous présentons les notions de base liées aux SMAs. Nous abordons également le protocole '*contract net*' ainsi que le principe des enchères qui sont des mécanismes très utilisés pour la négociation inter-agents.

III.1.1 Notion d'Agent

Plusieurs définitions ont été données au concept d'agent. Dans son livre [44] Freber résume cette notion comme suit :

Un agent est une entité physique ou virtuelle qui :

a. est capable d'agir dans un environnement
b. peut communiquer directement avec les autres agents
c. est guidée par un ensemble de tendances (sous forme d'objectifs individuels ou une fonction de satisfaction/survie qu'il essaye d'optimiser)
d. possède ses propres ressources
e. capable de percevoir son environnement à une certaine limite.
f. possède des compétences et peut offrir des services

III.1.2 Systèmes multi-agents

Un SMA est définit comme un ensemble d'agents situés dans un environnement donné, qui interagissent en communiquant, coopérant et négociant.

Jennings [63] décrit les SMAs par les caractéristiques suivantes :

a. chaque agent a des informations ou des capacités de résolution des problèmes incomplets et donc un point de vue limité.
b. Il n'y a pas de contrôle global du système
c. Les données sont décentralisées
d. Les calculs sont asynchrones.

III.1.3 Interaction entre agents

L'interaction entre les agents est la base d'un SMA efficace. Elle passe par la communication qui permet aux agents d'échanger les informations et les données nécessaires.

L'interaction peut être de différents types selon la nature des objectifs des agents, les options d'accessibilité aux ressources et les performances du système. On peut distinguer deux situations essentielles [44] :

III.1.3.1 La collaboration/coopération : c'est une situation de coordination assez complexe puisqu'elle combine l'allocation des tâches avec des ressources limitées pour des objectifs communs et compatibles. Les éléments de coopération les plus importants sont :

A. Le partage de tâches et de ressources : quand plusieurs agents travaillent ensemble pour réaliser un objectif commun, il est important de répondre à la question : 'qui fait quoi ?' en fonction du travail à réaliser. Pour distribuer les tâches et les moyens deux techniques sont utilisées :

   - les techniques basées sur les réseaux de connaissances (acquaintance networks) où les capacités de chaque agent sont présentées.
   - Les techniques basées sur le concept du marché avec les protocoles d'appels d'offres tel que le protocole de réseau de contrat "Contract Net protocol".

B. La coordination des actions : c'est le mécanisme nécessaire pour accomplir les tâches affectées aux agents et engendrer un comportement globale cohérent pour le SMA.

   Les actions des agents doivent être coordonnées pour les trois raisons suivantes :

   - Les agents ont besoin d'informations et de résultats offerts seulement par les autres agents.
   - Les ressources sont limitées.
   - Les coûts doivent être optimisés.

   La coordination est elle même un problème que les agents doivent résoudre. Elle consiste à ce que chaque agent : déterminer les agents avec lesquels il doit communiquer, le moment de la communication, détecte les conflits et les synergies possibles et trouve une façon pour se coordonner. Les différentes formes de coordination des actions sont :

   - La coordination par synchronisation : c'est la forme la plus élémentaire (de bas niveau). Elle consiste à déterminer de façon précise le séquencement des actions, et aboutit à la synchronisation de leur exécution. Cette forme s'impose quand il y a

besoin de gérer la concurrence de plusieurs actions et de vérifier que le résultat est cohérent.

- La coordination par la planification : c'est une technique de l'IA qui consiste à déterminer les actions nécessaires à accomplir un certain but et produire un ensemble de plans, ensuite sélectionner un plan qui sera exécuté. Des options de re-planification dynamique doivent être incluses pour faire face à l'évolution de l'environnement durant l'exécution des plans.
- La coordination réactive : considère des mécanismes de coordination basés sur des agents réactifs. Les décisions sont prises dynamiquement et non pas à priori. Ces approches ne donnent pas des résultats nécessairement optimaux mais elles sont applicables dans des contextes évolutifs et des situations où il est difficile d'anticiper sur ce qui va arriver.
- La coordination par régulation : s'applique à des systèmes qui nécessitent une coordination limitée. Le principe est d'imposer des règles de comportement pour éliminer un conflit possible.

C. La résolution des conflits : sert à résoudre les conflits entre les agents et garder un bon niveau de performance du système complet. On peut citer la technique de l'arbitrage qui définit un ensemble de règles de comportement (lois) qui agissent comme des contraintes sur l'ensemble des agents, de même que la technique de négociation qui laisse les agents résoudre leurs conflits eux mêmes par interaction directe. Cette dernière peut être bilatérale (one-to-one), multilatérale dans le cas où plusieurs agents cherchent à s'entendre sur un deal ou à base d'enchères.

   Pour établir un mécanisme de négociations, les quatre éléments suivants doivent être définis :

   - Un ensemble de propositions que les agents pourraient faire.
   - Un protocole de négociation.
   - Des stratégies privées pour chacun des agents.
   - Une règle qui précise quand l'accord est atteint.

III.1.3.2 La compétition : caractérisée souvent par des buts incompatibles, des ressources insuffisantes et des capacités limitées. Les situations de conflits dans ce cas sont à la fois des effets et des raisons d'interaction entre les agents. Les techniques issues de la théorie des jeux et la théorie d'aide à la décision conviennent également à la résolution des conflits et la prise de décisions distribuées dans le contexte compétitif.

III.1.4 Le protocole Contract Net : Les réseaux de contrats (Contract Nets : CN) correspondent à un protocole permettant d'implémenter des algorithmes spécifiques.

Le protocole CN peut être utilisé par les agents pour l'allocation des tâches dans le réseau. Dans ce cas des contrats peuvent être élaborés de haut en bas. A chaque niveau l'agent "manager" décompose la tâche en sous tâches à accomplir par d'autres contractants. Cette procédure se poursuit récursivement jusqu'à effectuer un contrat sans assistance. Quand une annonce est lancée sur le réseau, les agents avec les ressources nécessaires répondent par une offre 'bid'. Le manager évalue les offres reçues et alloue la tâche aux agents les plus offrants, ensuite le manager et les contractants s'échangent des messages concernant les données, la progression et les résultats obtenus par l'exécution de la tâche.

Le protocole contract net 'CNP' fournit des formats structurés de messages. Cependant pour l'utiliser, le manager doit utiliser des connaissances sur le problème en question pour décider quelles tâches annoncer, comment les décrire, comment envoyer l'annonce et comment sélectionner les meilleures propositions.

De son côté, le contractant doit aussi être capable d'interpréter les besoins de la tâche à partir du message et de décider s'il doit faire une offre ou pas et comment doit il faire cette offre.

Dans le cas de l'allocation des tâches le CN convient aux applications qui ont des hiérarchies de tâches bien définies et qui peuvent être décomposées en sous tâches presque indépendantes [41].

III.1.5 Les enchères

L'enchère est l'un des éléments importants du mécanisme de négociation permettant de résoudre les conflits entre les agents. Une enchère est un processus de vente qui consiste à vendre un produit au plus offrant. Son but est de maximiser le profit du vendeur et de minimiser les coûts d'un acheteur. Dans les SMA, une enchère prend place entre un agent connu comme "commissaire-priseur" et des agents "enchérisseurs".

Plusieurs dimensions d'une enchère existent ce qui permet de les différencier les unes des autres :

- La stratégie utilisée par le commissaire-priseur pour déterminer le gagnant.
- La stratégie utilisée par l'agent enchérisseur pour déterminer la valeur à attribuer à l'objet mis aux enchères. On peut distinguer les trois possibilités suivantes :
  Valeur privée : la valeur de l'objet dépend seulement des préférences de l'agent
  Valeur commune : la valeur de l'objet dépend seulement des préférences des autres agents
  Valeur corrélée : la valeur de l'objet dépend en partie des préférences de l'agent et pour l'autre partie, des préférences des autres agents
- La confidentialité des mises des différents agents enchérisseurs. Les mises peuvent être :
  Ouverte : tous les agents savent les mises des autres agents
  Secrète : tout agent ne connaît que ses propres mises
- La façon d'annoncer les mises :
  Directe: les agents ne misent qu'une fois
  Ascendant : les agents essaient de miser plus haut que la dernière plus haute mise
  Descendant : le commissaire-priseur annonce des mises en ordre décroissant et le premier agent à signaler son intérêt remporte l'objet au dernier prix annoncé

Parmi les enchères les plus connues on cite:

- L'enchère Anglaise où le prix du produit à vendre débute bas et croît jusqu'à ce qu'il n'y ait plus d'offres émises.
- L'enchère Hollandaise consiste à ce que le prix du produit à vendre débute haut et décroît jusqu'à ce qu'un acheteur accepte de le prendre.
- L'enchère du meilleur prix consiste à faire des mises de manière discrète et le gagnant est celui qui donne le plus haut prix.
- L'enchère Vickery qui considère que le gagnant est celui dont la mise est la plus élevée mais il paye la deuxième plus haute mise.

Un SMA fait donc cohabiter des agents autonomes dans un même environnement. Ces agents peuvent être coopératifs ou compétitifs ou les deux à la fois.
Les agents interagissent afin de partager le travail qu'ils ont à accomplir ou de résoudre des conflits lors de l'utilisation de ressources limitées. En utilisant les différents mécanismes

d'interactions les agents peuvent aboutir à une entente garantissant la cohérence du système complet.
Dans ce qui suit nous allons aborder la situation particulière où les agents coopèrent pour résoudre un problème donné de façon distribuée.

III.2 La Résolution Distribuée des Problèmes

La RDP est une technique permettant de faire collaborer plusieurs agents pour résoudre un même problème. Ces agents sont prédisposés à la coopération. Dans ce contexte, la coordination consiste à organiser les comportements des agents dans le temps et dans l'espace de sorte que l'action du groupe est améliorée soit par de meilleures performances ou bien par la réduction des conflits [44].

Dans la RDP, on suppose que les agents individuels du système, sont des nœuds de calcul programmés qui prennent les bonnes actions aux bons moments, tels qu'ils étaient conçus.
En collaborant, les nœuds font face à deux contraintes importantes : la première est que les solutions de leurs sous problèmes doivent être mutuellement cohérentes pour pouvoir les intégrer en une solution globale. Ainsi les nœuds doivent coordonner leur résolution parallèle et asynchrone du problème. La seconde contrainte concerne les limites de communication, ce qui signifie que les noeuds doivent compter sur un raisonnement local sophistiqué pour décider sur les actions et les interactions appropriées. Chaque noeud doit être capable de modifier son comportement quand les circonstances changent, et doit planifier ses stratégies de coopération et de communication avec les autres nœuds [40].

III.2.1 La planification distribuée

La planification distribuée peut être considérée comme un cas particulier de la résolution distribuée de problèmes, où le problème à résoudre est la conception d'un plan. Mais à cause de la particularité des problèmes de planification, il est généralement utile de considérer des techniques plus spécifiques [41].
Trois cas de planification distribuée sont possibles :

- ***Planification centralisée pour des plans distribués :*** Dans ce cas, étant donnée une description du but, un agent génère un plan, le décompose en sous plans tout en insérant les actions de synchronisation nécessaires. Il alloue les sous plans aux autres agents, puis initie l'exécution des plans et contrôle sa progression. Il est également

souhaitable de trouver parmi tous les plans réalisables possibles, le plan qui peut être décomposé et distribué le plus efficacement.

- ***Planification distribuée pour des plans centralisés :*** comme n'importe quel autre problème, la formulation de plans complexes peut nécessiter plusieurs agents. Chacun génère un plan partiel qui complète ceux des autres. Dans certains types d'applications, en logistique par exemple, l'interaction entre les agents de planification peut être réalisée en faisant passer un seul plan partiel autour des agents de façon séquentielle. Si l'un des agents n'arrive pas à effectuer sa tâche de planification avec le plan partiel reçu, il fait un retour arrière et d'autres choix sont explorés.

  Pour exploiter le parallélisme, les agents peuvent générer en parallèle des plans partiels ensuite, ils les échangent et les rassemblent pour converger vers un plan complet sans conflits. La coordination des plans peut s'effectuer en utilisant les techniques de l'optimisation distribuée de satisfaction de contraintes.
- ***Planification distribuée pour des plans distribués :*** pour certains types d'applications l'approche centralisée n'est pas une option viable pour les raisons suivantes : limites de calcul (la coordination de plusieurs nœuds dépasse les capacités de calcul d'un coordinateur unique), limites de communication (un seul coordinateur devient un goulot d'étranglement dans le réseau), et la fiabilité (la performance du système ne doit pas compter sur un seul nœud).

  Dans le cas de la planification distribuée pour les plans distribués, il est inutile d'avoir un plan complet n'importe où dans le système, car les parties du plan sont générées de façon distribuée pour être exécutées de façon distribuée. Mais ces parties doivent être compatibles c.à.d que les agents doivent pouvoir au moins exécuter leurs plans sans conflits, et chaque agent doit aider les autres à accomplir leurs plans quand c'est possible**.**

Vue la difficulté du problème de planification, les premières recherches en intelligence artificielle dans ce domaine lui ont apporté des simplifications importantes en considérant qu'un agent a toute la connaissance nécessaire, qu'il est le seul à changer l'état du système, que l'objectif ne change ni au cours de la planification ni au cours de l'exécution : La planification est complètement séparée de l'exécution du plan.

Cependant, dans les applications réelles où les environnements sont dynamiques et incertains, les agents ne peuvent pas faire des prédictions exactes sur les résultats de leurs actions.

Une approche pour gérer l'incertitude, consiste à énumérer les états contingents et de construire un plan conditionnel pour chacun de ces états.

Quand les connaissances de l'agent sont insuffisantes pour énumérer les états, ou quand leur nombre est très grand, une autre approche appelée "contrôler et réparer" peut être utilisée.
La progression de l'exécution est contrôlée et quand des déviations sont détectées l'agent arrête l'exécution et revisite ses décisions de planification en créant un plan révisé.
Dans certains environnements complexes, des changements inattendus peuvent offrir des opportunités pour accomplir les objectifs plus efficacement ; et parfois même les objectifs peuvent changer et la continuation de l'exécution du plan devient inutile. Dans tels cas les agents doivent évaluer et réviser leurs plans continuellement, ceci est appelé "*la planification continue"* qui consiste à interfolier la planification et l'exécution [36].

En résumé, les agents s'engagent dans la planification distribuée quand les données ou la responsabilité de la planification sont distribués entre différents agents ou quand les capacités et les compétences d'exécution nécessaires sont distribuées par nature.
Ils doivent planifier de façon continue quand l'environnement peut changer dynamiquement au delà de leur contrôle, quand ses aspects sont révélés progressivement, quand les contraintes de temps nécessitent le début de l'exécution avant la terminaison de la formulation du plan ou quand les objectifs évoluent dans le temps.

III.2.2 La satisfaction distribuée des contraintes

Quand un ensemble d'actions coordonnées est requis pour accomplir le but du système mais chaque agent a seulement des ressources limitées disponibles pour accomplir les actions qui lui sont affectées, la combinaison des contraintes de ressources locales et le besoin de coordination entre les agents fait apparaître un ensemble de contraintes interdépendantes [40].
La satisfaction distribuée des contraintes est une négociation multi-étapes qui étend le protocole CN pour permettre une négociation itérative entre l'appel d'offre et l'affectation des tâches. A chaque itération, chaque agent détecte si le choix local viole les exigences des autres agents concernant l'utilisation des ressources. A travers l'échange itératif des informations pertinentes, les agents convergent vers des choix compatibles. Les agents s'échangent seulement l'information nécessaire pour trouver une configuration qui satisfait leurs contraintes au lieu d'avoir dans chaque agent une vue globale des choix de tous les agents avec leur besoin en ressources.
Formellement le problème de satisfaction de contrainte (Constraint Satisfaction Problem : CSP) est décrit comme suit :

Un ensemble de N variables $\{x_1,\ldots,x_n\}$ dont les valeurs sont prises dans des domaines finis et discrets $D_1,D_2,\ldots D_n$; et un ensemble de contraintes sur les valeurs des variables. Une contrainte est définie par un prédicat $P_k(x_{k1},\ldots,x_{kj})$, ce prédicat est vrai si les valeurs assignées aux variables satisfont cette contrainte. Résoudre un CSP revient à trouver un assignement des valeurs aux variables tel que toutes les contraintes soient satisfaites ou déterminer qu'une telle affectation n'existe pas.

Un problème de satisfaction de contraintes distribué (Distributed Constraint Satisfaction Problem : DCSP) est un CSP où les variables et les contraintes sont distribuées entre les agents. Chaque agent a quelques variables et essaye de déterminer leurs valeurs. Cependant, il existe des contraintes inter-agents, et les valeurs assignées doivent les satisfaire.

Formellement :

Ayant m agents chaque variable x_i appartient à un agent i : "belongs(x_i, i)". Les contraintes sont aussi distribuées entre les agents. Le fait qu'un agent i connaît un prédicat de contrainte P_k est représenté par : known(P_k,i). Un DCSP est résolu si :

$\forall$ i, x_i tel que : belongs(x_i, i), si d_i est la valeur assignée à x_i et $\forall$ l, P_k tel que known(p_k,l), P_k est vrai sous l'assignement $x_i=d_i$.

Plusieurs problèmes d'applications en IAD qui concernent la recherche de combinaisons d'actions cohérentes entre les agents peuvent être formulés en DCSPs. Dans le problème d'allocation de ressources distribuées, chaque agent a ses propres tâches, et il y a plusieurs plans possibles pour réaliser (exécuter) chaque tâche. Et comme les ressources sont partagées entre les agents, il existe des contraintes/conflits entre les plans. Le but est donc de trouver la combinaison de plans qui permet à toutes les tâches de s'exécuter simultanément. Ce problème peut être formulé par un DSCP en représentant chaque tâche comme une variable et les plans possibles comme les valeurs des variables [100].

Vu la multitude des problèmes de l'IAD pouvant être formulés en DCSP, les algorithmes de résolution des DSCPs sont considérés comme une importante infrastructure en IAD.

Le défi principal dans la résolution d'un DCSP est d'assurer la terminaison et la complétude de la recherche. Les solutions à ces problèmes imposent généralement une certaine structure globale dans la recherche distribuée.

Assurer la complétude est possible si les agents ont la connaissance sur l'agent qui doit changer les valeurs de ses variables, pour que les autres gardent leurs valeurs inchangées. Ceci permet une recherche systématique dans l'espace des assignements collectifs, ce qui veut dire que la recherche est complète et qu'il est possible de détecter quand toutes les

combinaisons ont été testées pour terminer la recherche. Ceci peut parfois mener à une recherche complètement séquentielle, éliminant ainsi, tout bénéfice du parallélisme.
Les deux principaux algorithmes pour le DCSP sont :

III.2.2.1 Retour arrière synchrone (Synchronous backtracking : SyncBT)

Cet algorithme revient à réaliser une recherche en profondeur dans un arbre représentant l'espace des solutions. L'arbre est construit dynamiquement par l'ensemble des agents [101].
On considère que nous avons une seule variable par agent et que les agents/variables sont ordonnés *a priori*. Deux agents qui se partagent la même contrainte ont un lien entre eux. Les agents prennent leur décision locale à tour de rôle. Lorsqu'un agent détecte la violation d'une contrainte, il envoie un message à son prédécesseur lui demandant de changer la valeur de sa variable ce qui revient à un retour arrière chronologique.

III.2.2.2 Retour arrière asynchrone (Asynchronous backtracking : AsyncBT)

L'idée dans l'algorithme AsyncBT est de permettre aux agents de travailler en parallèle. Ceci est possible en permettant à chaque agent de changer à tout moment la valeur de sa variable. Lorsqu'un agent change la valeur de sa variable, il avise les agents de priorité inférieure. Lorsqu'un agent reçoit la valeur d'un agent de priorité supérieure, il choisit pour sa variable une valeur compatible en vertu des contraintes. Si ce n'est pas possible, un message est envoyé aux agents impliqués dans ce "blocage". Ces agents définiront une nouvelle contrainte qui les force à ce qu'au moins l'un d'entre eux change de valeur.
Plusieurs extensions du AsyncBT ont été proposées dans la littérature telle que le ABT_not [8] et le *Distributed Dynamic Backtracking* (DisDB) [7].

III.2.3 L'optimisation distribuée

Dans l'optimisation distribuée, étant donnée une certaine fonction objectif globale, on s'intéresse à savoir comment des agents peuvent optimiser cette fonction de façon distribuée.
Selon le problème à optimiser, différentes techniques telles que la programmation dynamique distribuée, les algorithmes distribués pour les problèmes décisionnels de markov, et les algorithmes d'optimisation à base d'économie sont applicables [87].

A. Algorithmes des enchères

Dans le protocole Contract Net, le problème global est divisé en sous tâches qui sont distribuées sur un ensemble d'agents. Chaque agent a différentes capacités. Il y a une fonction "C" qui donne pour chaque ensemble de tâches T un coût $C_i(T)$ pour accomplir les tâches de T par l'agent i. Chaque agent commence par un certain ensemble de tâches qui n'est pas optimal dans le sens où la somme des coûts de tous les agents n'est pas minimale. Les agents entrent donc dans un processus de négociation qui améliore l'affectation dans l'espoir d'aboutir à l'affectation optimale. En plus, même si le processus est interrompu avant sa terminaison il peut accomplir des améliorations signifiantes à l'affectation initiale.
Pour négocier dans ce type d'algorithmes, les agents sous-traitent les affectations entre eux. Chaque contrat implique l'échange des tâches ainsi que de l'argent. La question est de savoir comment le processus des enchères prend place et quels contrats sont maintenus en se basant sur ces enchères. Le protocole Contract Net est ouvert sur ces questions. Une approche particulière consiste à ce que chaque agent effectue une offre avec un coût marginal de l'agent pour cette tâche, càd, le coût supplémentaire d'ajouter cette tâche à son ensemble de tâches courantes. Les tâches sont allouées à l'offre la moins chère, et ce processus se répète.

Cette approche a été appliquée au problème d'affectation défini comme suit [6] :
• Un ensemble N de n agents
• Un ensemble X de n objets
• Un ensemble $M \subseteq N \times X$ de S affectations possibles.
• Une fonction $v : M \rightarrow \mathbb{R}$ qui donne une valeur à chaque paire d'affectation.
• Une affectation est un ensemble de paires de $S \subseteq M$ tel que chaque agent $i \in N$ et chaque objet $j \in X$ est au plus dans une paire dans S.

Une affectation est réalisable quand tous les P agents sont affectés à un objet.

Une affectation S est optimale si elle maximise $\sum_{(i,j)\in S} v(i,j)$

L'idée derrière la méthode des enchères est d'attribuer à chaque objet un prix p_j, ensuite chercher un *équilibre compétitif* entre l'affectation et le vecteur de prix.
Etant donné un vecteur de prix $P=(p_1,p_2,\ldots,p_n)$ et une affectation $S \subseteq M$, l'*utilité* d'affecter j à l'agent i est définie comme suit : $u(i,j)=v(i,j)-p_j$.
Une affectation et un vecteur de prix sont en équilibre compétitif si à chaque agent est affecté l'objet qui maximise son utilité selon les prix courants.

Le théorème suivant explique la relation entre la notion économique (équilibre compétitif) et l'optimisation combinatoire.

Théorème : Si une affectation réalisable S et un vecteur de prix P satisfont la condition d'équilibre alors S est une affectation optimale. En plus pour n'importe quelle solution optimale S, il existe un vecteur de prix P tel que P et S satisfont la condition d'équilibre compétitif [87]**.**

La recherche des solutions du programme linéaire revient à chercher l'espace des équilibres compétitifs. Et la façon de les chercher est la procédure des enchères.

L'algorithme des enchères utilisé pour la recherche commence d'un ensemble d'affectation vide et un vecteur de prix nul. Chaque itération est composée d'une étape d'offre et une étape d'affectation.

- *Etape d'offre*

  Chaque agent non affecté fait une offre pour l'objet $j \in X$ qui donne la valeur maximale aux prix courants:

  $$j \in \arg\max_{k/(i,k)\in M} (v(i,k) - p_k)$$

  La valeur de l'offre est la différence entre la valeur du meilleur objet et celui qui vient juste après (l'objet au deuxième rang).

  $$b_i = (v(i,j) - P_j) - \max_{k/(i,k)\in M; k\neq j} (v(i,k) - P_k) + \varepsilon$$

- *Etape d'affectation*

  Ajouter la paire (i,j) à S

  S'il existe une autre (i',j) alors l'enlever de S

  Incrémenter p_j par la valeur b_i

Ces deux étapes sont répétées jusqu'à ce que S devienne réalisable.

ε est une petite valeur ajutée au bid pour assurer que les prix continuent à augmenter et par conséquent l'algorithme se termine dans le cas où plusieurs agents concurrencent pour les mêmes objets.

La complexité de l'algorithme est de $O(n^3k)$ quand $\varepsilon=O(1/n)$. k est une constante qui ne dépend pas de n.

Le problème d'affectation est polynomial, les propriétés de terminaison et de convergence de l'algorithme pour ce problème sont garanties. Cependant, pour des problèmes plus complexes atteindre l'équilibre compétitif n'est pas toujours garantit et même la convergence de

l'algorithme vers la solution optimale n'est pas sûre. C'est le cas par exemple du problème de scheduling qui est NP-complet [87]

B. Le problème d'optimisation distribuée avec contraintes: (Distributed Constraints Optimization Problem : DCOP)

DCOP est une extension du DCSP aux problèmes d'optimisation dans lesquels en plus des contraintes entre les agents il existe une fonction objectif globale à optimiser par les agents.
DCOP est defini par un ensemble de variables $X=\{x_1,\ldots,x_n\}$. Chaque agent $A_i \in A$, $A=\{A_1,\ldots, A_n\}$, est responsable d'une variables x_i qui doit prendre une valeur dans un domaine $D_i \in D=\{D_1,\ldots,D_n\}$. Il existe aussi un ensemble de contraintes binaires $C=\{C1,\ldots,Cn\}$. Chaque contrainte C_j est une fonction $f_j(x_\alpha, x_\beta): D_\alpha \times D_\beta \rightarrow R^+$ $\alpha \neq \beta$. L'objectif est de donner des valeurs aux variables de manière à minimiser la fonction $F= \sum_{j=1..m} f_j$

Pour résoudre le DCOP, deux méthodes équivalentes aux SynchBT et AsynchBT existent:

- Synchronous Branch & Bound "SynchBB" [73].

  C'est une adaptation de SynchBT au DCOP dans laquelle l'algorithme ne s'arrête pas à la première solution trouvée. Quand un agent atteint une solution globale ou lorsque il se trouve devant une impasse il envoie un message à son prédécesseur lui demandant une autre alternative. Les messages échangés dans cette méthode contiennent la meilleure solution globale courante, qui sera utilisée par les agents pour élaguer l'arbre de recherche. Cette méthode est *complète*.

- Optimisation distribuée asynchrone "ADOPT" [74].

  Dans cette méthode, on suppose que chaque agent détient une seule variable et qu'il y a un ordre de priorité entre les agents : deux agents liés par une contrainte ont des niveaux de priorité différents. Un agent doit être plus prioritaire que tous les autres, ce qui donne une structure d'arbre.

  L'idée dans ADOPT est que chaque agent stocke pour chaque valeur de variable une borne minimale de la fonction objectif correspondante à cette valeur. Il choisit toujours la valeur de plus petite borne minimale (on considère que la fonction objectif est à minimiser) à sa variable et envoie cette valeur à tous ses descendants dans l'arbre. Quand un agent reçoit une valeur de son prédécesseur, il cherche la valeur ayant la plus petite borne inférieure en fonction de la valeur qu'il vient de recevoir, affecte cette valeur à sa

variable et envoie la nouvelle borne inférieure au père pour qu'il mette à jour les bornes inférieures des valeurs de sa variable. Après la mise à jour, il change la valeur de sa variable si nécessaire.

Cet algorithme est asynchrone et complet pour les DCOP [75]. La borne minimale de la fonction objectif permet de garantir la convergence vers la solution optimale.

ADOPT ne peut être adaptée à tous les problèmes distribués car cette méthode impose certaines conditions qui font ses limites :

- les contraintes doivent être binaires
- la fonction objectif doit s'exprimer comme la somme des "valeurs" des contraintes.
- La fonction objectif doit être monotone.

Une autre limite des algorithmes complets est qu'ils doivent traverser tout l'espace de recherche pour garantir l'optimalité de la solution et comme les DCOPs sont NP-complets, cet inconvénient limite l'utilisation de ces algorithmes à des problèmes relativement petits. Pour cette raison d'autres algorithmes approximatifs distribués sont proposés dans la littérature tels que les algorithmes de recherche distribués arborescents et les algorithmes de recherche locale distribués.

C. La recherche distribuée à base de déviations (Limited discrepancy search : LDS)

La recherche basée sur le principe de déviations est une méthode de recherche arborescente distribuée générale considérée comme une technique de retour arrière dans un arbre de recherche selon laquelle le retour arrière n'est pas nécessairement chronologique. La méthode définit un "sélectionneur" de nœud, et à chaque fois que les conditions d'un retour arrière sont rencontrées, le sélectionneur choisit le nœud avec le plus faible nombre de déviations pour reprendre la recherche.

L'explication de ce choix est que les solutions pour chaque nœud (les fils de ce nœud) sont générées en vertu d'un critère défini par l'algorithme utilisé. Si on considère que la qualité de ces solutions décroît suivant l'ordre de leur génération alors le nombre total de déviations d'un nœud qui est le nombre de fois qu'il faut brancher à droite pour aller de la racine à ce nœud est plus faible pour les meilleures solutions.

Dans le contexte distribué, cette méthode a été appliquée par Gaudrault [47] [48] pour un problème d'optimisation distribué hiérarchique : à savoir la coordination des agents dans un réseau de production de bois d'œuvre. L'auteur a proposé principalement une adaptation

distribuée synchrone à cette méthode “Synchronous Limited Discrepancy Search : SynLDS” ainsi qu’une version asynchrone “Multi-Agent Concurrent Discepancy Search : MACDS”

- **SyncLDS :** est l’équivalent d’une LDS centralisée faite par plusieurs agents distribués. La méthode est synchrone et un seul agent est actif à la fois.

Lorsqu’un agent envoie un nouveau nœud (une nouvelle solution) au successeur, il ajoute le chemin qui mène de la racine vers ce nœud. A chaque fois qu’un agent rencontre la condition de retour arrière (une solution globale est trouvée ou aucune nouvelle solution n’est possible pour un chemin donné), il demande à tous les autres agents d’identifier le nœud local avec le plus faible nombre de déviations. Il donne ensuite le contrôle à l’agent avec le nœud qui a la plus petite valeur pour continuer la recherche à partir de ce point en résolvant séquentiellement les sous problèmes restants.

- **MACDS :** Dans cet algorithme les agents travaillent simultanément et utilisent des communications asynchrones. A chaque fois qu’un agent génère une solution et la transmet au successeur (en lui attachant le chemin correspondant), il commence immédiatement à générer la suivante. Lorsqu’il reçoit un nouveau nœud (une nouvelle solution), il recalcule les priorités pour décider s’il doit continuer son calcul en cours ou bien s’il doit passer vers le nouveau nœud car il le trouve plus prioritaire. Cette sélection se fait suivant une certaine fonction qui définit la stratégie de recherche.

D. La recherche locale distribuée

Cette classe d’algorithmes ne garantit pas l’optimalité des solutions obtenues mais sont capables de trouver de bonnes solutions approximatives.

Parmi ces algorithmes, on peut citer le Message du Gain Maximum (Maximum Gain Message : MGM) qui est une version simple du DBA (Distributed Breakout Algorithm). Dans l’algorithme MGM [69] on considère que chaque agent a une seule variable et communique seulement avec ses voisins (les agents avec qui il partage des contraintes) dans l’objectif de minimiser le poids des contraintes insatisfaites. A chaque étape l’agent échange la valeur de sa variable avec ses voisins, calcule la meilleure réduction du poids, envoie cette réduction et collecte celles de ses voisins. L’agent qui a le maximum de réductions est autorisé à changer sa valeur. L’algorithme DBA est basé sur le même principe sauf que pour échapper aux minimums locaux, quand aucun agent ne peut faire d’amélioration, le poids des contraintes

violées est incrémenté et le processus d'échange des valeurs de réduction est poursuivi [102]. L'algorithme stochastique distribué (Distributed Stochastic Algorithm : DBA) est un autre algorithme de recherche locale distribuée où les agents décident de changer les valeurs de leurs variables de façon stochastique [103].

Une autre approche de passage de messages basée sur l'algorithme Max-Sum a été utilisée dans [43] pour la coordination dans les réseaux de capteurs. L'objectif est de trouver les valeurs de variables de chaque agent tel que la somme des utilités individuelles des agents est maximisée. Le problème est représenté par un graphe de facteurs biparti composé de deux types de nœuds : les nœuds variables et les nœuds fonctions représentant respectivement pour chaque agent sa variable et sa fonction d'utilité. Avec cette décomposition on considère que la somme des fonctions dans le graphe est égale à la fonction originale à optimiser par les agents. L'algorithme utilise deux types de messages pouvant être échangés d'un nœud variable à un nœud fonction et d'un nœud fonction à un nœud variable.

- Un nœud variable i envoie à un nœud fonction j la somme des valeurs de toutes les autres fonctions ($\neq$j) qui lui sont liées.
- Un nœud fonction j envoie à un nœud variable i le maximum des sommes des valeurs reçues des variables qui lui sont connectées (sauf la variable i) plus la valeur correspondante à $F_j(x_i)$. Selon les valeurs envoyées et reçues, l'agent i décide quelle valeur doit-t-il choisir à sa variable pour maximiser la somme des utilités des agents.

Dans cet algorithme les messages peuvent être initialisés et mis à jour à chaque fois qu'un agent reçoit un nouveau message d'un voisin et aucune synchronisation des messages n'est nécessaire.

Il a été montré que l'algorithme Max-Sum est capable d'atteindre de bonnes solutions en peu d'itérations. C'est un algorithme très utile car il peut être exécuté indéfiniment dans les systèmes où les utilités des agents ou l'interaction entre eux change dans le temps.

III.3 Conclusion

Bien que souvent confondue avec les SMAs, la resolution distribuée des problemes continue à occuper une place particulière dans beaucoup de travaux de recherche qui s'intéressent aux qualités des solutions obtenues par la résolution distribuée des problèmes plus qu'aux propriétés du système lui-même et des agents qui le composent.

Dans ce travail, nous nous basons sur cet axe pour proposer des approches efficaces pouvant s'appliquer aux problèmes d'optimisation distribué et que nous adaptons au problème de gestion intégrée des véhicules d'urgence décrit dans le chapitre qui suit.

IV Systèmes de gestion des véhicules d'urgences

Dans une étude statistique sur les catastrophes dans le monde établi en fin 2007, le centre de recherche sur l'épidémiologie des désastres "CRED" en Belgique [68] confirme que le nombre de catastrophes naturelles a augmenté de 60% en dix ans dont 56% sont liés aux conditions atmosphériques. Les inondations, les tremblements de terre, les incendies, les feux de forêts sont les risques les plus cités. D'un autre côté, les accidents technologiques, les attaques terroristes, les épidémies, les accidents de la route, les cas de maladies individuels, les accidents domestiques…etc. sont également des risques omniprésents partout dans le monde.
Tous ces dangers ont fait que les systèmes de gestion des urgences prennent une place de plus en plus importante ces dernières années.

Un système de gestion des urgences est un système qui gère les ressources humaines et matérielles pour s'occuper d'une situation de catastrophe. Ce genre de systèmes incluent la prévention, la préparation, la réponse, la reprise, la réhabilitation, le soutien et la législation des urgences de tout types, de toutes ampleurs et à tout emplacement avec l'objectif de réduire la mortalité, les invalidités, les dégâts et la destruction.
Plusieurs services se trouvent impliqués dans les systèmes de gestion des urgences :

1. Les services de la santé publique et des urgences
2. Les services de sécurité
3. Les services de la protection civile
4. Les services de la protection des infrastructures critiques
5. Les services de recherche et de secours
6. les services de la prévention des catastrophes
7. les services d'aide humanitaire

…

Les activités de ces systèmes quant à elles diffèrent d'un service à l'autre : l'évacuation des espaces, la coordination des services des urgences médicales, la gestion des équipements d'urgences, des équipes médicales spécialisées et des véhicules d'urgences en sont des exemples.

D'autres systèmes tels que les systèmes d'information géographiques 'SIG', les systèmes de prévision du trafic, les systèmes de geo-localisation et de surveillance des flux…. sont également impliqués dans le processus de gestion des situations d'urgences.

Dans notre travail nous nous intéressons aux services de gestion des flottes de véhicules pour la réponse aux appels d'urgences.

Dans la plupart des pays un numéro d'appel d'urgence est mis au service de la population (Algérie 14 pour la protection civile, 911 pour les USA et le Canada, 112 en Europe). Dés qu'un appel est reçu, le processus de réponse aux urgences est activé pour coordonner les différents intervenants. Les étapes de ce processus sont souvent caractérisées par l'incomplétude de l'information et une demande élevée de prise de décision soumise à une complexité et des contraintes de temps strictes. Trois étapes principales décrivent ce processus :

1. la collecte d'information concernant l'urgence : le type de l'événement provoquant l'urgence, son degré de gravité, son ampleur, sa localisation…
2. La coordination des différents intervenants et la prise de décision : cette étape est généralement difficile car à ce stade les informations sont imprécises et incomplètes pour cette raison l'utilisation des systèmes d'aide à la décision est vitale.
3. Le suivi et l'assistance des premiers intervenants depuis leur départ jusqu'à l'accomplissement de leur tâche : cette phase nécessite des moyens de communication efficaces et des systèmes de prise de décision hautement adaptables aux changements.

L'ultime objectif du processus complet est la minimisation du temps de réponse aux appels d'urgence.

La durée d'une urgence est divisée en 4 phases [99]: le temps de détection, le temps de préparation, le temps de déplacement et le temps de traitement. Le temps de réponse est défini comme la durée entre le temps de réception de l'appel d'urgence et le temps d'arrivée du véhicule d'urgence à l'appel, qui est la somme du temps de préparation et de déplacement.

Le temps de réponse est nettement affecté par la disponibilité des véhicules, l'état du trafic, et les décisions prises pour le choix du véhicule à envoyer. Ainsi, le système de gestion de flotte de véhicules pour la réponse aux urgences doit optimiser les opérations pour améliorer la performance du service et réduire les pertes.

Les véhicules sont l'élément de base dans un système de gestion des flottes de véhicules pour la réponse aux appels d'urgences. Ils peuvent être de différents types : véhicules de pompiers, ambulances, voitures de police… Certains types de véhicules ont des emplacements fixes au

moment du dispatching, d'autres comme les patrouilles de police sont mobiles. La distinction est importante à des fins administratives ainsi que analytiques [11]. Mais en pratique, la différence entre véhicules fixes et mobiles devient moins importante dans les périodes où la demande est élevée car les véhicules peuvent être dispatchés directement d'un incident à un autre ou quand ils sont en route. Dans de telles conditions le système devient insensible à l'emplacement initial des véhicules.

La conception d'un système efficace de gestion des flottes de véhicules pour la réponse aux appels d'urgences passe par les trois niveaux : stratégique, tactique et opérationnel.

Les décisions stratégiques sont prises à long terme, elles concernent par exemple la localisation des stations où les véhicules se rendent quand ils ne sont pas en service (pour la maintenance, pour faire la réserve ou tout simplement attendre une autre demande de service) et la détermination de la taille de la flotte nécessaire au bon service pour une région donnée;

Le niveau tactique concerne la programmation des équipes de secours ainsi que les activités de maintenance des équipements à moyen terme sur une durée d'une semaine par exemple.

Alors que le niveau opérationnel consiste à décider du véhicule à dispatcher à l'arrivée d'un appel d'urgence, de dévier les véhicules de leur destination afin de servir un appel plus urgent, changer l'itinéraire d'un véhicule en cas d'incident de trafic et la relocalisation des véhicules lorsque des régions surveillées deviennent non couvertes à la suite du dispatching des véhicules.

Nous classons les principales activités au niveau opérationnel en deux problèmes à savoir : Le problème de dispatching et le problème de couverture.

IV.1 Dispatching

Ce problème consiste à choisir le véhicule à envoyer à l'arrivée d'un appel d'urgence. Dans beaucoup de travaux des stratégies de dispatching telle le véhicule le plus proche sont utilisées. Dans [57] les auteurs ont comparé quatre stratégies de dispatching différentes :

1. premier arrivé premier servi où les appels sont affectés dans l'ordre de leur arrivée ; si plusieurs véhicules sont disponibles, celui qui était oisif pour longtemps est affecté à l'appel.
2. le véhicule le plus proche : dans ce cas le véhicule le plus proche de l'appel lui est affecté
3. dispatching flexible : à chaque point de simulation on optimise les affectations courantes de sorte que le temps total de réponse soit minimisé. Dans ce cas la 'diversion' et le 'rerouting' sont permis [voir section IV.2.2].

4. le déploiement : les véhicules peuvent changer de destination vers un autre appel ou une autre destination sous le contrôle du dispatcher.

Les résultats ont montré que les stratégies 3 et 4 sont meilleures que la 2eme stratégie qui est meilleure que la première stratégie.

Dans [1] Les auteurs proposent une stratégie pour décider du véhicule à dispatcher suivant les priorités de l'appel: pour les appels de haute priorité, le véhicule avec le plus court temps de voyage est affecté, en cas d'égalité celui qui affecte le moins la couverture est choisi. Pour éviter de faire attendre longtemps les appels les moins prioritaires ils utilisent des pseudo priorités qui sont incrémentés quand un appel n'est pas satisfait dans une limite de temps donnée.

Le problème de dispatching peut également être modélisé comme un problème d'affectation généralisé 'GAP : Generalized Assignement Problem'

Le GAP est un problème np-difficile qui consiste en un ensemble de *m* agents et un ensemble de *n* tâches n>=m. Chaque agent peut effectuer n'importe quelle tâche encourant un certain coût et nécessite une quantité de ressources. Chaque agent a un nombre limité de ressources disponibles ; la somme des ressources nécessaires aux tâches qui lui sont affectées ne peuvent dépasser ce nombre. L'objectif est de trouver une affectation telle que tous les agents n'excèdent pas leurs ressources disponibles et le coût total de l'affectation est minimisé [76].

La formulation mathématique du GAP est comme suit :

c_{ij} le coût que la tâche *j* soit affectée à l'agent *i*

r_{ij} la quantité de ressources requise par l'agent *i* pour la tâche *j*.

b_i les ressources disponibles à l'agent *i*

$$x_{ij} = \begin{cases} 1 & \text{si la tâche } j \text{ est affectée à l'agent } i \\ 0 & \text{sinon} \end{cases}$$

$$\text{Min} \sum_{i=1}^{m}\sum_{j=1}^{n} c_{ij}x_{ij}$$

$$\text{Tel que : } \sum_{j=1}^{n} r_{ij}x_{ij} \le b_i \qquad \forall \text{ i}$$

$$\sum_{i=1}^{m} x_{ij} = 1 \qquad \forall \text{ j} \in \text{N}$$

N={1,2,…,n} l'ensemble des tâches.

$x_{ij} \in \{0,1\} \quad \forall\ i,j$

La résolution du GAP avec des solveurs exacts [46], [84] peut être très coûteuse en temps de calcul. Pour cette raison des méthodes heuristiques et des méta heuristiques telles que les algorithmes génétiques sont utilisées [16].

Dans [56] un modèle mathématique a été proposé pour le dispatching et le re-routing avec l'objectif de minimiser le temps de voyage dans le système. Les décisions prises sont basées sur des informations concernant les conditions du trafic en temps réel.

IV.2 Couverture

Le problème de couverture est en étroite relation avec le problème de dispatching car son objectif principal est de déterminer le nombre de véhicules nécessaires et leur répartition géographique pour que les régions soient bien couvertes et ainsi les futurs appels qui arrivent pourront être servis dans les plus courts délais.

Ce problème a été largement étudié depuis plus de 40 ans. Les premiers travaux considéraient le cas statique où les décisions ne sont pas revues au cours de l'exécution. Les travaux les plus récents par contre traitent le cas dynamique en reconsidérant les décisions au fur et à mesure que l'état du système change.

IV.2.1 Le problème de couverture statique

Parmi les méthodes les plus utilisées pour traiter le problème de couverture, on peut citer celles qui appliquent certaines normes officielles imposées pour la couverture des régions dans une ville donnée. Par exemple "à 2 kilomètres de chaque point, il doit y avoir au moins trois stations d'urgence" ou bien "chaque point ne doit pas être plus loin de 5km d'une station d'urgence". Ces standards peuvent changer d'une ville/région à une autre, dans les zones rurales par exemple des standards plus souples s'appliquent.

Le modèle proposé par [96] "Location Set Covering Problem : LSCP" et celui de [17] "Maximal Covering Location Problem : MCLP" considèrent qu'un point de demande (d'où un appel peut survenir) est couvert par un site (où les véhicules d'urgence sont placés) donné si le temps de déplacement du site vers le point est inférieur à une norme de couverture préétablie.

Dans le LSCP l'objectif est de minimiser le nombre de véhicules nécessaires à couvrir tous les points de demandes, alors que le MCLP considère un nombre limité de véhicules disponibles et détermine leurs emplacements de sorte que la population couverte soit maximisée. Le deuxième modèle peut être ramené au premier en l'exécutant de façon répétée tout en

incrémentant le nombre de véhicules disponibles jusqu'à ce que toutes les demandes soient couvertes. Un compromis entre le coût des véhicules et la couverture peut être réalisé.

Location set and covering problem 'LSCP' a été modelisé par Toregas [96] comme suit:

$$Min\ Z = \sum_{j \in S} x_j$$

Tel que:

$$\sum_{j \in N_i} x_j \geq 1 \qquad \forall i \in I$$

$$x_j \in \{0,1\} \qquad \forall j \in S$$

S: l'ensemble des sites (indexé par j)

I: L'ensemble des points de demande (indexé par i).

$$x_j = \begin{cases} 1 & \text{si un vehicule se trouve au site j} \\ 0 & \text{sinon} \end{cases}$$

$N_i=\{j / d_{ji}<=\varphi\}$

d_{ji} est la plus courte distance du site j au point de demande i

φ est la norme de couverture.

N_i est l'ensemble de tous les sites qui sont à une distance$<\varphi$ du point i. Si un véhicule est situé dans l'un de ces sites le point i sera considéré couvert. La première contrainte exige que chaque point de demande doit être couvert par au moins un véhicule à une distance $<\varphi$.

Le Maximal Covering Location Problem "MCLP" a été modélisé par Church & ReVelle [17] comme suit :

$$Max\ Z = \sum_{i \in I} a_i y_i$$

$$y_i \leq \sum_{j \in N_i} x_j \qquad \forall i \in I$$

$$\sum_{j \in S} x_j = P$$

$x_j, y_i \in \{0,1\} \quad \forall j \in S, i \in I$

P est le nombre de véhicules disponibles.

a_i la population au point de demande i.

Le LSCP a été utilisé pour localiser les arrêts de bus [53]. Le MCLP a été utilisé pour localiser les cliniques médicales [42] et plusieurs autres applications.

Une extension du MCLP a été proposée par [86] 'TEAM' où deux types de véhicules A et B sont pris en considération, avec deux normes de couverture différentes. Le modèle maximise le nombre de points couverts par les deux types de véhicules en imposant une contrainte de hiérarchie entre eux.

Dans le modèle de [22] on utilise un objectif hiérarchique pour maximiser le nombre de points couverts plus d'une fois alors que dans [59] le nombre total de points couverts deux fois est maximisé. Ces modèles tentent d'assurer une meilleure couverture pour faire face au manque de couverture quand les véhicules sont dispatchés sans augmenter le nombre total de véhicules.

Dans [59], les auteurs proposent un modèle qui maximise le nombre de points de demandes couverts une fois dans un rayon r et ceux couverts deux fois dans le même rayon, les deux sommes sont pondérées dans la même fonction objectif.

Le modèle dans [51] utilise deux rayons $r_1$et r_2 tel que $r_1<r_2$. Toutes les demandes doivent être couvertes dans le rayon r_2 et une proportion α dans le rayon r_1. (Aux états unis par exemple la loi fixe r_1 à 10 minutess et α à 0.95. r_2 n'est pas limité).

L'inconvénient des modèles déterministes précédents est que les normes imposées ne remplacent pas le temps de voyage effectif des véhicules vers les appels car ce dernier dépend d'autres facteurs comme la disponibilité des véhicules et leur vitesse de déplacement.

Les premiers travaux ayant pris cet aspect en considération étaient basés sur la théorie des files d'attente. Ils considèrent pour chaque secteur un seuil maximal pour la probabilité qu'un appel soit mis en attente (ou bien un temps d'attente moyen) ensuite les secteurs sont choisis suffisamment petits pour que le seuil soit respecté. Le nombre de secteurs ainsi conçu détermine le nombre de véhicules nécessaires à mettre en service [11].

Les applications réelles des services d'urgence nécessitent souvent un modèle de file d'attente multi-serveurs. Dans le modèle le plus simple un appel est placé en attente seulement si tous les véhicules sont occupés par d'autres appels. Les appels ont la même priorité, le temps de service est exponentiel et le processus d'arrivée est poissonien. Dans [89] cette version a été appliquée à l'état stationnaire pour déterminer le nombre d'ambulances nécessaires dans une région afin de s'assurer que la probabilité d'attente ne dépasse pas un seuil donné. Dans [18] les auteurs introduisent les priorités des appels, ils considèrent que les plus prioritaires sont servis en premiers mais gardent l'hypothèse qu'un seul véhicule répond à chaque appel et que

le temps de service a une distribution exponentielle. Chaken [12] a développé un modèle de file d'attente aux véhicules de pompiers qui a des caractéristiques plus réalistes :

- Les appels peuvent nécessiter différents nombres d'unités de différents types.
- Les unités peuvent arriver / partir seules ou en groupe.
- Le temps d'occupation des véhicules dépend du type de l'incident.

C'est un modèle avec un nombre infini de serveurs tel que les véhicules nécessaires dans une région sont dispatchés dans la même région ou dans d'autres régions si nécessaire.

Le résultat commun à tous ces modèles, ainsi qu'à d'autres modèles, est que le nombre de véhicules nécessaires augmente avec le taux des appels, mais pas avec la même proportion ce qui explique que la relation n'est pas linéaire.

Daskin [23] propose un modèle MEXCLP qui considère qu'une ambulance a une probabilité q d'être non disponible pour répondre à un appel et que tous les véhicules sont indépendants. La probabilité d'occupation est estimée en divisant le temps total de toutes les demandes par le nombre de véhicules disponibles. Le modèle maximise le nombre de points couverts par un minimum de k véhicules. TIMEXCLP [82] est une extension du MEXCLP qui prend en considération la variation de la vitesse de déplacement des véhicules au cours de la journée, l'application de ce modèle à un cas réel a amélioré le temps de réponse de 36% et a augmenté la couverture de 90%.

Dans [55] une variante du MEXCLP a été développée et prend en considération des temps de déplacement stochastiques. L'objectif étant de maximiser les nombres d'appels couverts à 8 minutes. D'autres travaux réalisés dans [83] maximisent la demande couverte avec une certaine probabilité α . Dans [2] les auteurs ont développé une extension du LSCM pour intégrer une contrainte sur le nombre de véhicules nécessaires à garantir un certain niveau de fiabilité.

IV.2.2 Le problème de couverture dynamique

Dans le cas dynamique, les véhicules sont redistribués pour restituer la couverture des zones qui deviennent non couvertes après le dispatching des véhicules.

Afin de garder un bon niveau de couverture, des stratégies telles que la déviation (diversion), la relocalisation (relocation) et la réorientation (rerouting) sont utilisées. La déviation veut dire que le véhicule qui est en route peut changer de destination (vers un appel d'urgence plus prioritaire par exemple). La relocalisation consiste à changer l'emplacement d'un véhicule libre afin d'améliorer la couverture, alors que la réorientation (rerouting) permet de changer l'itinéraire d'un véhicule en route en se basant sur les informations du trafic.

Peu de travaux ont été consacrés à ce cas. Parmi ceux qui traitent la ‘relocation’ des véhicules [64] pour les services de pompiers de la ville de New York ‘FDNY’ le problème a été modélisé par un programme linéaire dont l’objectif est de garder une couverture maximale (définie par des normes officielles) de toutes les régions de la ville ainsi que la minimisation du temps de déplacement des véhicules. La méthode proposée pour résoudre le modèle est devenue une partie du système de contrôle et de gestion de l’information en temps réel du FDNY (Fire Department of New York). C’est un algorithme heuristique rapide qui consomme peu de mémoire. Les résultats des tests ont montré que la méthode est efficace surtout en temps de crise.

Vu que ce problème doit être résolu de façon fréquente (soit périodiquement soit quand l’administrateur décide que c’est nécessaire parce qu’il y a un manque de couverture dans une région donnée) et en un temps court, les méthodes de résolution approchées telles que les meta-heuristiques sont très utilisées. Dans [52] les auteurs adaptent le DSM (Double Standard Model) au cas dynamique et résolvent le modèle par une heuristique tabou parallèle pour satisfaire les contraintes temporelles du problème. Le système a été testé sur des données réelles de la région de Montréal. L’algorithme a permis de pré-calculer la stratégie de redéploiement des véhicules dans 95% des cas.

Dans [57], un modèle dynamique intégrant le dispatching et la couverture a été résolu par la recherche tabou et a été appliqué à des données réelles de la région de Washington DC aux USAs. Les tests effectués ont montré que l’approche de résolution proposée est prometteuse.

Dans [70] on propose une approche basée sur la programmation dynamique approximative pour la prise de décisions de redéploiement des ambulances en temps réel. L’approche proposée sert à contourner le problème des espaces d’état de grande taille de la programmation dynamique et les résultats obtenus montrent la performance de l’approche par rapport aux politiques statiques.

Le cas dynamique a également été traité dans [67] par un système multiagents. Les auteurs utilisent le Contract Net Protocol pour choisir l’ambulance qui transporte le patient. Ils incluent un agent TRUST qui indique au coordinateur la véracité du temps de voyage donné par les conducteurs. Un autre travail dans le même sens [66] a été appliqué pour coordonner les services des ambulances avec les experts neurologistes pour assister les cas de maladie cardiovasculaire.

IV.3 Conclusion

Dans ce chapitre nous avons dressé un panorama des différents modèles et méthodes proposés dans la littérature pour le problème de gestion des véhicules d'urgence pour le dispatching et la couverture. Au terme de notre état de l'art, nous avons conclu qu'il n'y a pas à l'heure actuelle de travaux qui ont traité l'aspect distribué du problème de gestion intégrée des véhicules d'urgence. Le peu de travaux qui existent dans ce sens traitent soit le problème intégré séquentiellement avec un contrôle centralisé, soit proposent des systèmes multi-agents comme organisation naturelle reflétant les systèmes réels mais sans se concentrer sur l'optimisation liée à l'intégration des deux problèmes dans le même modèle.

Dans les chapitres qui suivent nous allons présenter un modèle pour sa version intégrée ainsi que des approches issues des différents axes de l'IAD pour sa résolution.

V Le système de gestion des véhicules d'urgence centralisé

Dans notre travail, nous nous intéressons au problème de dispatching et de couverture intégré "DISCOV" qui consiste à affecter les véhicules d'urgence aux meilleurs emplacements de façon à traiter le dispatching et la couverture simultanément.

Nous traitons dans ce chapitre la version centralisée de DISCOV qui va nous servir de référence pour l'étude et l'évaluation de la version distribuée du même système qui fera l'objet du chapitre suivant.

Dans ce qui suit, nous présentons le problème de dispatching et de couverture intégré étudié ainsi qu'un algorithme hybride basé sur la combinaison des colonies de fourmis et la recherche tabou pour sa résolution séquentielle. Nous présentons ensuite deux versions parallèles du même algorithme.
L'effet de l'intégration est également présenté grâce à des résultats numériques comparant le traitement séquentiel des deux problèmes avec la solution intégrée que nous proposons.

V.1 Gestion intégrée des véhicules d'urgences "*DISCOV*"

Le problème étudié est caractérisé par les hypothèses et les contraintes suivantes:
- On considère un centre responsable de la réception de tous les appels d'urgence et du contrôle du mouvement des véhicules dans le système. La région gérée par ce centre contient un ensemble de stations géographiquement dispersées ; chacune possède un ensemble de véhicules d'urgence tous de même type.
- Quand un appel d'urgence arrive, le centre doit prendre des décisions concernant le véhicule à envoyer pour assister l'appel. Sans perte de généralité, on considère que chaque appel a besoin d'un seul véhicule. Les appels ont des priorités pour indiquer leur degré de gravité.
- La région sous contrôle est divisée en zones à partir desquelles les appels arrivent. Pour répondre aux appels dans les meilleurs délais, le centre doit contrôler l'affectation des véhicules aux stations afin de garantir un bon taux de couverture pour toutes les zones. Le taux de couverture d'une zone est donné par sa préparation "*prepardnes*s" ; celle-ci indique si le système est bien préparé pour répondre aux futurs appels arrivants de cette zone.
- Un véhicule en station est considéré oisif. Quand il est en route vers un appel ou une station il peut être dévié. Quand il arrive à l'appel il doit finir le service avant d'être considéré disponible.

- Pour éviter de réaffecter le même véhicule plusieurs fois ou de dévier beaucoup de véhicules, ce qui peut perturber les équipes de secours ainsi que les appels, une limite est imposée en n'acceptant que les réaffectations qui apportent un gain important pour le système complet.

V.1.1 Modèle mathématique

Pour décrire formellement le problème, les notations suivantes sont introduites.

- **Ensembles**

V	ensemble des véhicules d'urgence
V^{idle}	ensemble des véhicules disponibles (qui sont en stations)
V^{call}	ensemble des véhicules en route vers un appel d'urgence
V^{busy}	ensemble des véhicules en service
$V^{station}$	ensemble des véhicules en route vers une station
S	ensemble des stations
W	ensemble des appels en attente d'un service
Z	ensemble des zones

- **Indices**

j	indice des véhicules
s	indice des stations
i	indice des urgences
z	indice des zones

- **Paramètres**

$Prio_i$	priorité de l'urgence i
$T_{ij}(t)$	estimation du temps nécessaire au véhicule j pour atteindre l'urgence i partant à l'instant t
$T_{js}(t)$	estimation du temps nécessaire au véhicule j pour atteindre la station s partant à l'instant t
$Ts_Z(t)$	temps de déplacement nécessaire de la station s à la zone z.

C_z	poids représentant le besoin de la zone z en couverture
φ	Taux de couverture requis
τ	Taux de déviations permis
M	Un grand nombre
Penal1	pénalité appliquée quand un appel d'urgence n'est pas satisfait.
Penal2	pénalité appliquée quand une zone n'est pas couverte.
Penal3	pénalité appliquée quand un véhicule est dévié de sa destination

- **Variables**

$PR_z(t)$ preparedness de la zone z à l'instant t

X_{ji}^0 = $\begin{cases} 1 & \text{si le vehicule } j \text{ etait affecté à l'urgence } i \text{ à la derniere étape de simulation} \\ 0 & \text{sinon} \end{cases}$

X_{js}^0 = $\begin{cases} 1 & \text{si le vehicule était affecté à la station } s \text{ à la derniere étape de simulation} \\ 0 & \text{sinon} \end{cases}$

$unsat_i^0$ = $\begin{cases} 1 & \text{si l'urgence } i \text{ n'était pas satisfaite à la derniere étape de la simulation} \\ 0 & \text{sinon} \end{cases}$

$uncov_z^0$ = $\begin{cases} 1 & \text{si } PR_z < \varphi \text{ à la derniere étape de simulation} \\ 0 & \text{sinon} \end{cases}$

$X_{ji}(t)$ = $\begin{cases} 1 & \text{si le vehicule } j \text{ est affecté à l'urgence } i \text{ à l'instant } t \\ 0 & \text{sinon} \end{cases}$

$X_{js}(t)$ = $\begin{cases} 1 & \text{si le vehicule } j \text{ est affecté à la station } s \text{ à l'instant } t \\ 0 & \text{sinon} \end{cases}$

$unsat_i(t)$ = $\begin{cases} 1 & \text{si l'urgence } i \text{ n'est pas satisfaite à l'instant } t \\ 0 & \text{sinon} \end{cases}$

$uncov_z(t)$ = $\begin{cases} 1 & \text{si } PR_z(t) < \varphi \\ 0 & \text{sinon} \end{cases}$

$dev_j(t)$ = $\begin{cases} 1 & \text{si le vehicule } j \text{ est devié at l'instant } t \\ 0 & \text{sinon} \end{cases}$

- **Modèle**

L'objectif du modèle proposé est de trouver une affectation des véhicules aux urgences et aux stations qui minimise à l'instant t, la somme du temps de déplacement total vers les urgences, Le temps de déplacement total vers les stations, le nombre des urgences insatisfaites, le nombre de zones non couvertes et le nombre de véhicules déviés.

$$\text{MIN} \quad \sum_{j \notin V^{busy}} \sum_{i \in W} (X_{ji}(t) T_{ji}(t) \text{prio}_i) + \sum_{j \notin V^{busy}} \sum_{s \in S} (X_{js}(t) T_{js}(t)) + Penal1 \sum_{i \in W} unsat_i(t) + Penal2 \sum_{z \in Z} un\text{cov}_z(t) + Penal3 \sum_{j \in V} dev_j(t)$$

La fonction objective est minimisée sujet aux contraintes C1 à C8.

La première contrainte (C1) indique que chaque véhicule doit être affecté à une et une seule location (urgence ou station) à un moment donné. La Contrainte (C2) fait que chaque appel lui est affecté au plus un véhicule. (C3) détermine si une urgence est satisfaite ou pas. Les Contraintes (C4) & (C5) déterminent si une zone est couverte ou pas. Les contraintes (C6), (C7) & (C8) contrôlent la déviation de véhicules; elles permettent seulement les déviations qui apportent un gain plus grand que τ.

$$\sum_{i \in W} X_{ji}(t) + \sum_{s \in S} X_{js}(t) = 1 \qquad \forall\ j \notin V^{busy} \qquad \text{(C1)}$$

$$\sum_{j \in V} X_{ji}(t) \le 1 \qquad \forall\ i \in W \qquad \text{(C2)}$$

$$1 - \sum_{j \notin V_{busy}} X_{ji}(t) \le M \cdot unsat_i(\text{t}) \qquad \forall\ i \in W \qquad \text{(C3)}$$

$$\varphi - PR_z(t) \le M \cdot un\text{cov}_z(t) \qquad \forall\ z \in Z \qquad \text{(C4)}$$

$$PR_z(t) = 1/C_z \cdot \sum_{j \notin V^{busy}} \sum_{s \in S} (X_{js}(t) / T_{sz}(\text{t})) \qquad \forall\ z \in Z \qquad \text{(C5)}$$

$$X_{ji}(t) - X_{ji}(t) \cdot X_{ji}^0 \le M \cdot dev_j(t) \qquad \forall\ j \in V^{call} \quad \text{i} \in W \qquad \text{(C6)}$$

$$X_{js}(t) - X_{js}(t) \cdot X_{js}^0 \le M \cdot dev_j(t) \qquad \forall\ j \in V^{station} \quad s \in S \qquad \text{(C7)}$$

$$\sum_{l \in W \cup S} (X_{jl}^0 \cdot t_{jl}(t)) - \sum_{l \in W \cup S} (X_{jl}(t) \cdot t_{jl}(t)) + \qquad \forall\ j \in V^{call} \cup V^{station} \qquad \text{(C8)}$$

$$penal1 \cdot (\sum_{\text{i} \in \text{W}} unsat_i^0 - \sum_{i \in W} unsat_i(t)) +$$

$$penal2 \cdot (\sum_{z \in Z} un\text{cov}_z^0 - \sum_{z \in Z} un\text{cov}_z(t)) - \tau \le M \cdot dev_j(t)$$

V.1.2 Etude de l'intégration

Nous avons réalisé ce système par un simulateur à événements discrets qui nous permet de le faire évoluer dans le temps, et ainsi reproduire les principaux événements liés à la gestion des urgences.

Les événements traités par le simulateur sont principalement la réception d'un nouvel appel, l'arrivée du véhicule affecté à l'urgence, la fin de service et l'arrivée du véhicule en station. A chacun de ces points, le modèle est optimisé et les variables d'état du système sont mises à jour. A l'arrivée d'un nouvel appel par exemple, on doit décider quel véhicule doit être dispatché et éventuellement les autres véhicules sont redistribués pour améliorer la couverture.

La fréquence d'arrivée des appels au système ainsi que le temps de service nécessaire pour chaque appel sont des paramètres déterminants qui permettent d'analyser le modèle ainsi que l'approche de résolution dans différentes situations.

Selon l'étude effectuée dans [99] sur des données réelles des systèmes de gestion des urgences l'arrivée des appels suit une loi exponentielle d'un paramètre lambda représentant le temps moyen entre deux arrivées successives et le temps de traitement d'une urgence suit une loi normale. Ces résultats sont utilisés en expérimentation et vont nous servir à déterminer le moment d'arrivée de chacun des événements liés à l'arrivée d'un nouvel appel et la fin de service. Les événements d'arrivée à l'appel ainsi que l'arrivée à la station sont déterminés par le temps de voyage du véhicule qui est calculé dans ce travail par un outil de cartographie qui nous permet d'avoir les itinéraires avec la distance et le temps de voyage entre deux emplacements sur une carte avec un réseau routier réel.

Afin d'étudier l'impact de l'intégration des deux problèmes en un seul modèle, nous avons comparé les solutions obtenues par deux approches :

Approche1 **:** la minimisation de la somme du temps de déplacement total à l'instant t vers les urgences, le nombre des urgences insatisfaites et le nombre de véhicules déviés (pour le problème de l'affectation de véhicules) suivie par la minimisation du temps de déplacement total à l'instant t vers les stations, le nombre de zones non couvertes et le nombre de véhicules déviés (pour la couverture).

Approche2 **:** la minimisation de la fonction objectif du modèle (section V.1).

Le système est implémenté par un simulateur à événements discrets en langage java. Vu l'indisponibilité des données réelles pour ce problème, nous avons préparé un ensemble d'instances. Toutes les données : les appels d'urgences, les stations et les zones correspondent à des adresses geo-localisées sur une carte avec un réseaux routier réel.

La geo-localisation, l'itinéraire, les distances et les temps de voyages sont calculés en utilisant l'outil MapPoint. Chaque instance est composée de:

1. Un ensemble de véhicules, chacun a une station comme position initiale.
2. Un ensemble de stations, une station est caractérisée par sa position sur la carte.
3. Un ensemble de zones, chacune a une position sur la carte (le centre de cette zone) et une valeur représentant son besoin en couverture qui sert à calculer sa *preparedness*.
4. Un ensemble d'appels, chacun caractérisé par sa position sur la carte, la zone à laquelle il appartient, sa priorité, son temps d'arrivée et de service.

Deux ensembles de données sont testés. Dans le premier, le temps d'arrivée moyen entre deux appels successifs est égal à 30 minutes (dataset1) et dans le second il est égal à 15 minutes (dataset2).

Les figures 5.1 et 5.2 présentent une comparaison du temps de réponse moyen pour dataset1 et dataset2 respectivement. Les figures 5.3, 5.4 et 5.5, 5.6 présentent une comparaison du pourcentage de zones non couvertes ainsi que le temps moyen de non couverture calculés par rapport au temps total de simulation.

Les mêmes résultats sont résumés dans le tableau 5.1 qui donne la différence entre l'approche2 et l'approche1 pour les deux ensembles de données.

On constate qu'en terme de temps de réponse l'approche1 est légèrement meilleure que l'approche2 pour certaines instances. Cependant l'approche 2 garde une meilleure couverture soit en nombre de zones non couvertes soit en temps de non couverture soit pour les deux.

Plus précisément, nous remarquons que pour dataset1 une différence moyenne de 3.01% en temps de réponse contre 17.50% en temps moyen de couverture et 18.10% de zones non couvertes. La différence est plus notable pour dataset2 avec 8.84% de différence moyenne en temps de réponse contre 31.29% de temps de non couverture et 67.22% de zones non couvertes.

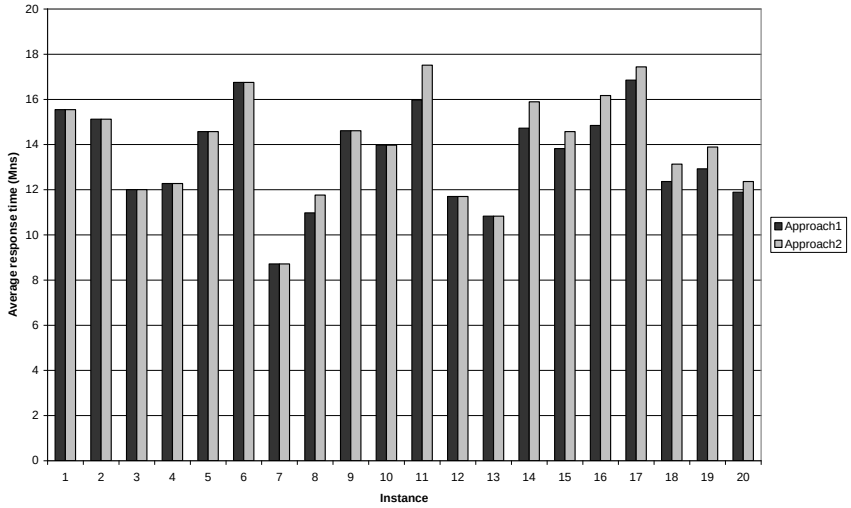

Figure 5.1 Temps de réponse moyen pour dataset1

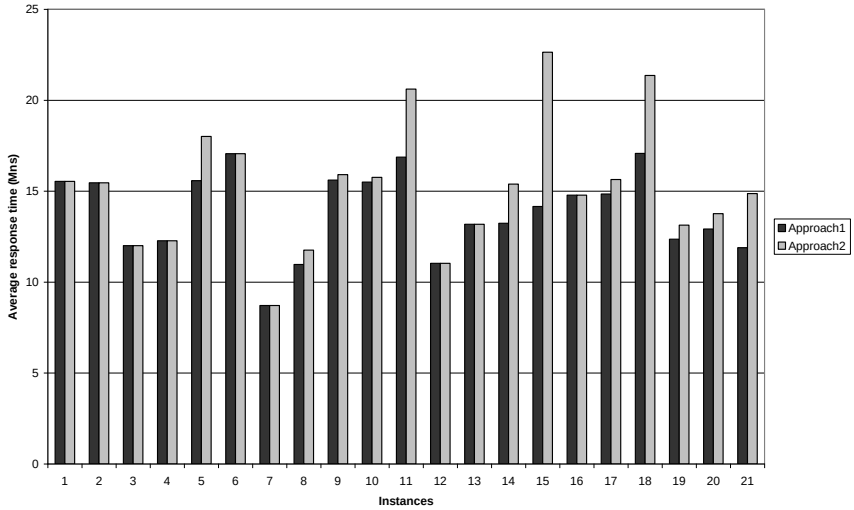

Figure 5.2 Temps de réponse moyen pour dataset2

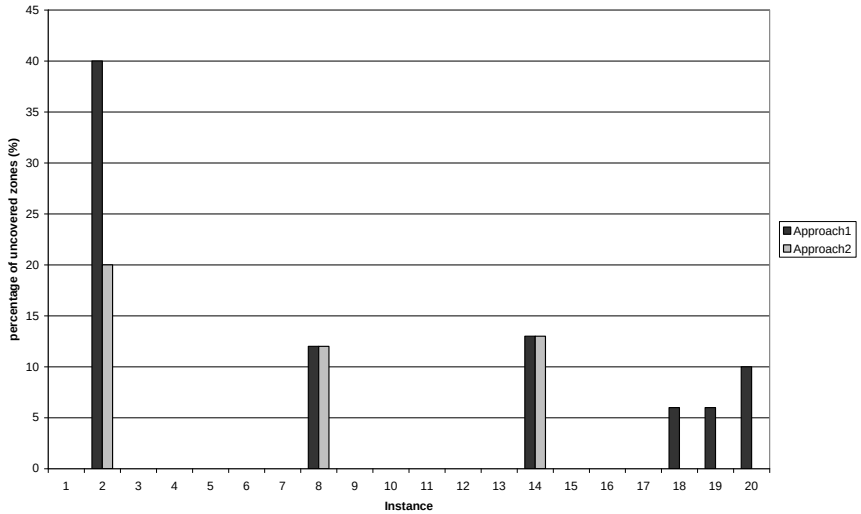

Figure 5.3 Pourcentage de zones non couvertes pour dataset1

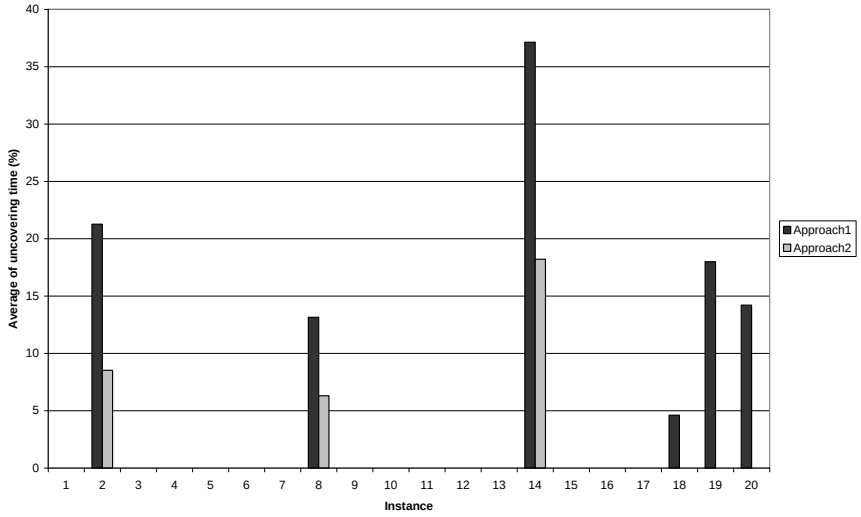

Figure 5.4 Temps moyen de non couverture pour dataset1

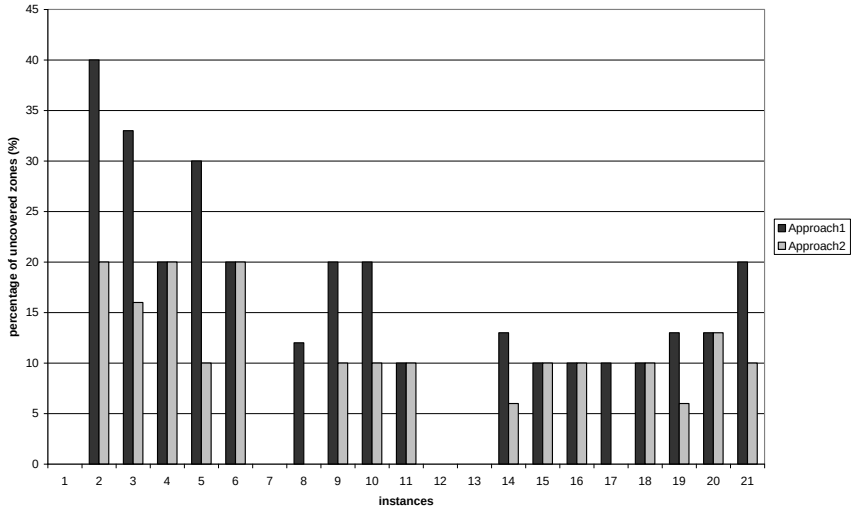

Figure 5.5 Pourcentage de zones non couvertes pour dataset2

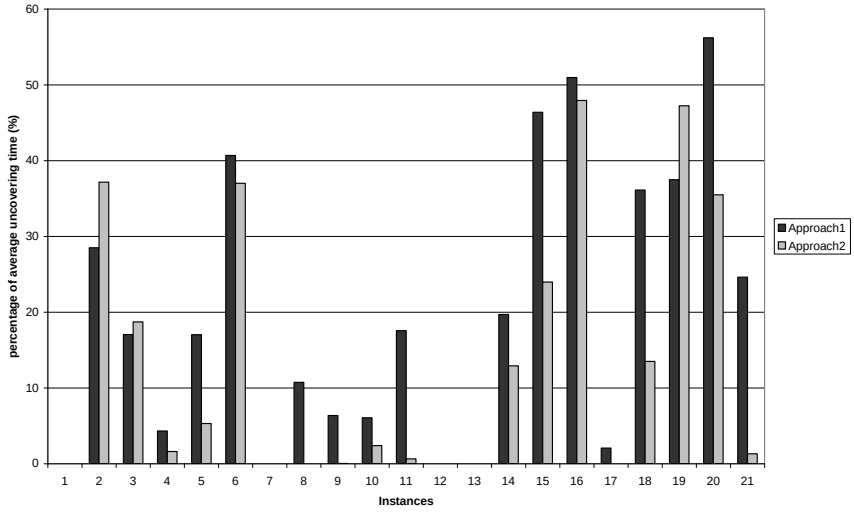

Figure 5.6 Temps moyen de non couverture pour dataset2

	Temps de réponse moyen	pourcentage des zones non couvertes	temps moyen de non couverture
Dataset1	+3.01%	-18.10%	-17.50%
Dataset2	+8.84%	-67.22%	-31.29%

Tableau 5.1. L'effet de l'intégration

Nous faisons remarquer que le critère de couverture est aussi important que le temps de réponse car quand les zones sont bien couvertes, même si le temps de réponse obtenu par la solution du modèle n'est pas satisfaisant, le décideur aura le choix pour un autre véhicule aux appels les plus critiques. Avec le modèle intégré on peut mieux contrôler les deux critères dispatching et couverture, ce qui n'est pas possible s'ils sont considérés séquentiellement.

V.2 Solution centralisée : AnTabu

V.2.1 AnTabu séquentiel

Pour résoudre le modèle de la section précédente, nous proposons une métaheuristique hybride qui combine la recherche tabou et l'optimisation par les colonies de fourmis : "AnTabu".

Tel qu'il est mentionné dans [95], un bon équilibre entre l'intensification et la diversification est la clé de conception des métaheuristiques. Les méthodes orientées population comme les algorithmes génétiques, la recherche dispersée ou les colonies de fourmis sont de bonnes méthodes d'exploration de l'espace de recherche mais faibles en exploitation. D'un autre côté, les métaheuristiques comme la recherche tabou, le recuit simulé et les procédures de recherche locale sont de bons candidats pour l'exploitation mais relativement moins bonnes en exploration. Pour cette raison beaucoup de meta-heuristiques hybrides combinent des méthodes trajectoires avec des méthodes basée-population.

Parmi les travaux qui ont étudié ce genre d'interactions on trouve celui de [50] où les auteurs ont considéré différentes instances de l'algorithme génétique local qui est une méthode qui combine les AG et recherche locale. Dans [2] la recherche tabou a été couplée à l'algorithme génétique pour le problème de l'affectation quadratique. Dans [58] les auteurs utilisent un algorithme ACO avec la recherche locale pour résoudre efficacement le problème de la prochaine version (the next release problem).

Dans [80] [95] on trouve une classification des différentes conceptions et exécutions possibles des metaheuristiques hybrides. Les auteurs distinguent principalement l'hybridation de bas niveau et de haut niveau. Dans la première, les composants sont échangés de façon que certaines fonctions d'une metaheuristique soient remplacées par une autre métaheuristique, alors que dans l'hybridation de haut niveau le fonctionnement interne de chaque métaheuristique ne change pas.
Le séquencement des interactions est un autre critère utilisé dans la classification : dans l'hybridation à relais les metaheuristiques sont appliquées l'une après l'autre de façon pipeline ; alors que dans l'hybridation par collaboration (teamwork) elles sont appliquées de façon collaborative.

L'algorithme Antabu que nous proposons peut être classé comme une méthode collaborative de haut niveau. Il est décrit en détail dans la figure 5.7.

```
Begin
    Pour chaque génération
        Pour chaque Ant
            S=Construire solution()
            Màj phéromone online(S)
        Fin Pour
        S*=Déterminer la meilleure solution
        Tabu_search(S*)
        Màj pheromone offline(S*)
    Fin Pour
End
```

Figure 5.7 L'algorithme AnTabu

V.2.1.1 L'algorithme Ant Colony System

Pour construire une solution chaque fourmi choisit itérativement un emplacement *l* (appel ou station) pour l'affecter à un véhicule avec une probabilité P_j^l calculée par la règle proportionnelle pseudo aléatoire suivante :
Soit q une variable aléatoire uniformément distribuée sur [0,1] et $q_0 \in [0,1]$ un paramètre réglable

Si ($q<=q_0$) Alors

$$P_j^l(t) = \begin{cases} 1 \text{ if } (j,l) = \text{argmax}\left\{phero[j,l]^{\alpha}\, heur[j,l]^{\beta}\right\} \\ 0 \text{ else} \end{cases} \quad (1)$$

Sinon

$$P_j^l(t) = \frac{phero[j,l]^{\alpha}\, heur[j,l]^{\beta}}{\sum_{v \in V(t)} phero[v,l]^{\alpha}\, heur[v,l]^{\beta}} \quad (2)$$

phero[j,l]: La quantité de phéromone pour l'affectation (*j*,*l*).

heur[j,l]: l'heuristique de l'affectation du véhicule *j* à l'emplacement *l.*

α ,β: paramètres qui contrôlent respectivement l'équilibre de phéromone/ heuristique.

V(t): ensemble des véhicules déviables à l'instant t.

La mise à jour de phéromone est appliquée à chaque solution construite (online) et à la meilleure solution de chaque génération (offline). Dans les deux cas elle commence par une phase d'évaporation qui consiste à réduire la quantité de phéromone de tous les éléments en multipliant la quantité courante par un facteur (1-ρ) où $0 \leq \rho \leq 1$ est le taux d'évaporation. La deuxième étape consiste à ajouter une quantité relative à la qualité de la solution en considération aux éléments présents dans cette solution. L'importance relative aux quantités évaporées et ajoutées est contrôlée par le paramètre ρ.

V.2.1.2 La recherche Tabu

Dans la recherche tabou le processus de recherche se déplace d'une solution à sa voisine en évaluant à chaque étape le voisinage de la solution courante et en sélectionnant la meilleure.

A. Voisinage

Pour passer d'une solution à une autre, deux types d'opérateurs sont utilisés

- Swap (v_i,v_j) échange les emplacement des véhicules v_i et v_j.
- Move(v,l) déplace le véhicule v de son emplacement courant vers l'emplacement *l.*

Avec l'opérateur swap (qui est équivalent à deux moves consécutifs) on échange les emplacements de deux véhicules en même temps, ensuite on évalue la solution obtenue une seule fois. Alors qu'avec l'opérateur move, à chaque changement de l'emplacement d'un véhicule une évaluation de la fitness est effectuée. Pour accélérer l'exploration de l'espace de recherche, nous commençons par l'opérateur swap dans la procédure tabou principale et quand nous décidons d'intensifier la recherche dans une région donnée nous utilisons l'opérateur move qui est plus raffiné que le swap et peut mener à de meilleures solutions.

B. La liste tabou

Pour éviter de cycler durant le processus de recherche une liste tabou est utilisée. Elle contient les opérateurs "*move"* récemment appliqués. Quand un opérateur "*move"* est appliqué ce dernier doit être inséré dans la liste (dans le cas du swap les deux "*move"* correspondants sont insérés) ; il devient tabou et ne peut être appliqué pour un nombre d'itérations égal à la taille de la liste tabou : "*t_lenght"*.

Cependant quand un "*move"* tabou peut mener à une solution meilleure que celle trouvée du début de la recherche, il peut quitter l'état tabou en appliquant le critère d'aspiration objectif.

La taille de la liste est un paramètre important qui affecte beaucoup le processus de recherche et par conséquent la qualité de la solution.

Le processus de recherche peut être facilement attrapé par les minimums locaux lorsque la taille de la liste est petite, alors qu'une liste tabou de grande taille peut empêcher la recherche de visiter des solutions voisines possiblement bonnes.

En se basant sur les travaux précédents sur la recherche tabou, fixer la valeur de *t_lenght* durant toute la recherche aboutit à un algorithme incapable de s'adapter à l'espace de recherche. Nous proposons donc de déterminer la valeur de *t_lengh* dynamiquement. Après chaque "*move"*, sa valeur est déterminée comme suit : *t_lenght*= *t_lenght*+β. Step.

$$\text{step} = \begin{cases} -1 & \text{si le critere d'aspiration est appliqué dans cette iteration.} \\ 1 & \text{si le vhicule a été remis à son emplacement sans critére d'aspiration.} \end{cases}$$

β : est un paramètre empirique $\in$ [0,1].

Pour éviter les valeurs incohérentes de *t_length* nous fixons empiriquement des bornes minimales et maximales que la liste ne doit pas dépasser. *t_length*$\in$ [*min_lenght,max_length*].

La liste tabou est implémentée par une table, l'élément (i,j) contient la dernière itération où le véhicule *i* a été affecté à l'emplacement *j*. pour savoir si le *move* est dans l'état tabou, nous avons juste à comparer la différence entre l'itération courante et la valeur correspondante dans la table avec la taille de la liste tabou.

C. Intensification/diversification

Le principe d'intensification consiste à exploiter les propriétés des meilleures solutions pour raffiner la recherche aux côtés de ces solutions. La diversification permet au processus de recherche d'atteindre les régions inexplorées de l'espace de recherche.

Dans l'algorithme on procède à l'intensification quand le nombre d'itérations sans amélioration de la solution atteint une certaine limite. Dans cette étape, nous utilisons l'opérateur move pour raffiner la recherche.

Après une phase d'intensification, une phase de diversification est appliquée en choisissant les mouvements les moins récemment utilisés et ainsi rediriger la recherche vers de nouvelles régions de l'espace. La procédure de recherche tabou est résumée dans la figure 5.2 en utilisant les notations suivante.

best_s :	La meilleure solution trouvée depuis le début de l'algorithme
current_s:	La position actuelle du processus de recherche
neighbourhood(*current_s*) :	La procédure qui retourne le meilleur voisin "*best_n"* de current_s qui n'est pas tabou ou qui satisfait le critère d'aspiration.
no_improve:	Le nombre d'itérations successives sans amélioration de best_s
update_t_lenght():	La procédure de mise à jour de la taille de la liste tabou
Max_no_improve:	Le nombre maximum du nombre d'itérations sans amélioration avant de commencer le processus d'intensification.

```
Input: S
Output: best_s
  (a) best_s=S; no_improve=0; current_s=S;
  (b) if (stopping criterion==true)
                return(best_s).
  (c) current_s=neighbourh(current_s)
  (d) update_t_length();
  (e) if(cost(current_s)>cost(best_s))
        no_improve++;
  (f) if(no_improve==max_no_improve)
        no_improve=0;
        current_s=Intensif(current_s);
        if(cost(current_s)<cost(best_s))
          best_s=current_s;
        diversif(current_s);
  (g) go to (b)
```

Figure 5.8 La procédure tabu_search

V.2.2 AnTabu parallèle

L'algorithme Antabu séquentiel inclut deux parties importantes : la génération des solutions par l'ACO et l'amélioration de la solution par la recherche tabou. La première partie présente une caractéristique de parallelisation inhérente à cause de l'aspect d'indépendance dans la génération des solutions. Dans la seconde partie, l'élément le plus coûteux est l'évaluation du voisinage, ce dernier peut aussi être parallélisé avec peu d'interactions entre les processus. En se basant sur ces observations nous proposons la stratégie de parallélisation maître/esclave pour accélérer la version séquentielle de l'algorithme.

A chaque itération le maître lance n processus esclaves, attend leur terminaison pour sélectionner la meilleure solution construite, applique une recherche tabou parallèle à cette solution, effectue une mise à jour offline de phéromone et relance la nouvelle génération de fourmis. Pour équilibrer la charge entre les processus, le nombre de fourmis est divisé par le nombre de processus. Chaque processus effectue la construction et la mise à jour on-line de la phéromone pour toutes les fourmis qui lui sont affectées. Il est important de noter que pour garder la cohérence de la table de phéromone, les accès sont contrôlés par des mécanismes de synchronisation et d'exclusion mutuelle.

La deuxième phase importante de parallélisation réside dans la procédure de recherche tabou, et plus exactement dans la partie qui consomme le plus de temps à savoir l'évaluation du voisinage. Cette évaluation est présente deux fois dans la procédure de recherche tabou séquentielle avec deux opérateurs différents : *swap* et *move*. Dans les deux cas l'ensemble des swaps/*moves* est partagé sur les processus et ainsi un voisinage partiel est évalué par chaque processus. Les solutions globales sont retrouvées en utilisant la communication inter processus. Selon la façon de communiquer on distingue l'approche synchrone et l'approche asynchrone.

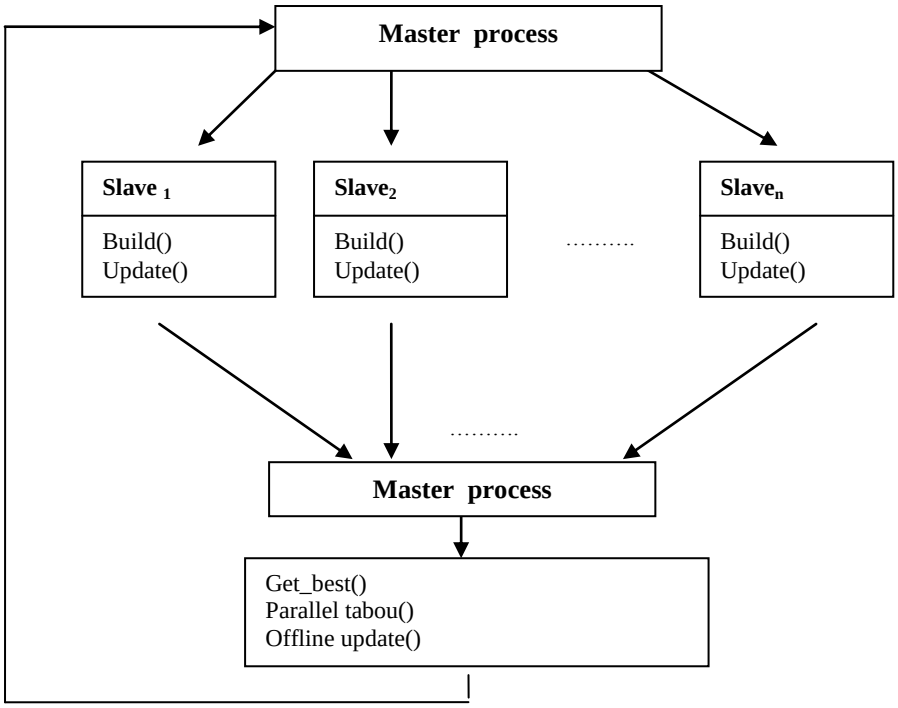

Figure 5.9 AnTabu parallèle

V.2.2.1 Algorithme synchrone

Quand une évaluation parallèle du voisinage est requise dans la procédure de recherche tabou principale ou bien dans la phase d'intensification, l'algorithme synchrone utilise un processus maître qui initialise les processus esclaves avec la solution initiale (la solution courante) et leurs sous ensembles d'opérateurs et les lancent. Quand tous les esclaves terminent leur évaluation partielle, le maître choisit la meilleure solution trouvée et continue son exécution séquentielle. Dans le cas de l'évaluation principale de la procédure tabou, il met à jour la liste tabou ensuite, selon le cas, soit il commence une phase d'intensification parallèle suivie par une diversification (si le nombre d'itérations sans amélioration atteint sa limite) ou bien recommence un nouveau cycle de la recherche tabou. L'intensification parallèle suit le même schéma synchrone que dans l'évaluation parallèle du voisinage.

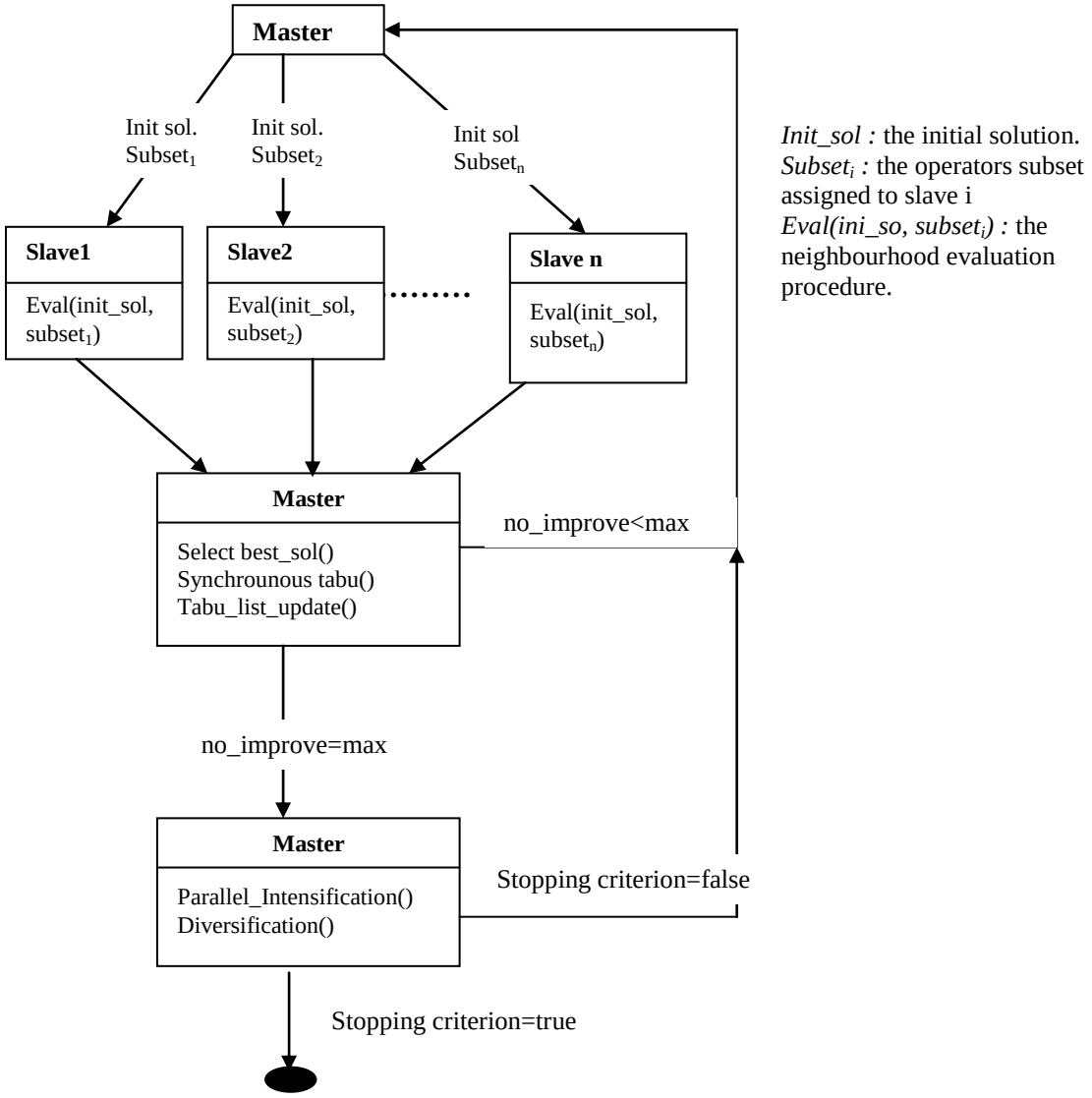

Figure 5.10 Tabou parallèle synchrone

V.2.2.2 L'algorithme asynchrone

La stratégie asynchrone limite le rôle du master à l'initialisation des processus avec la meilleure solution de la génération courante de fourmis. Les étapes de retrouver et distribuer les solutions dans la recherche tabou est remplacée par une interaction directe entre les processus. Chaque processus exécute la procédure tabou séquentielle complète dans son propre voisinage partiel en utilisant les opérateurs qui lui ont été assignés et communique ses résultats aux autres processus à des moments différents. Comme nous l'avons mentionné dans la section II.3.3, les travaux précédents ont montré que le moment et la fréquence d'échange des informations peut affecter remarquablement la performance de l'algorithme pour la

qualité des solutions ainsi que le temps d'exécution. Pour cette raison, nous comparons empiriquement trois cas pour le moment d'échange d'information :

a. Après chaque évaluation de voisinage avec les opérateurs *swap/move*

b. seulement après l'évaluation par les opérateurs swap

c. seulement après l'évaluation par l'opérateur *move*.

L'information échangée est la nouvelle solution trouvée de l'évaluation du voisinage.

- **Résultats expérimentaux**

Les algorithmes décrits précédemment ont été implémentés en Java et exécutés sur une machine bi-processeur: un Intel coreTM 2 Duo processor T5200, 1.60 GHZ et 1G de RAM

A. Impact du nombre de processus sur le temps d'exécution

Le figure 5.11 montre la variation du temps d'exécution en fonction du nombre de processus.

On remarque que pour les deux méthodes synchrone et asynchrone le temps d'exécution diminue quand le nombre de processus augmente. Cependant à partir d'une certaine limite il commence à augmenter car l'introduction de plusieurs threads réduit la performance à cause de la surcharge du changement de contexte.

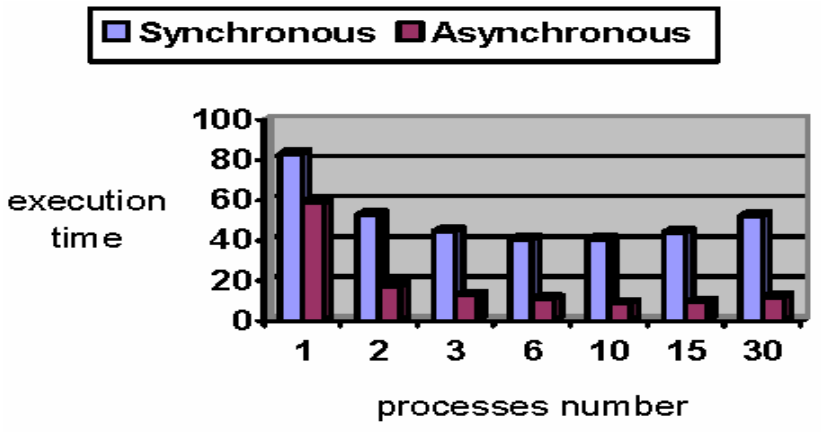

Figure 5.11 Impact du nombre de processus sur le temps d'exécution

La deuxième remarque importante est que l'algorithme asynchrone améliore considérablement le temps d'exécution comparé à l'algorithme synchrone.

B. Comparaison de la qualité de solution

On remarque que contrairement à l'approche synchrone, dans la stratégie asynchrone, la qualité de la solution diminue quand le nombre de threads augmente. Quand les données sont divisées sur plusieurs processus dans l'approche asynchrone, la solution qui résulte du calcul parallèle devient moins bonne car l'évaluation indépendante des voisinages partiels dans la recherche tabou n'est pas suffisante pour faire émerger de bonnes solutions.

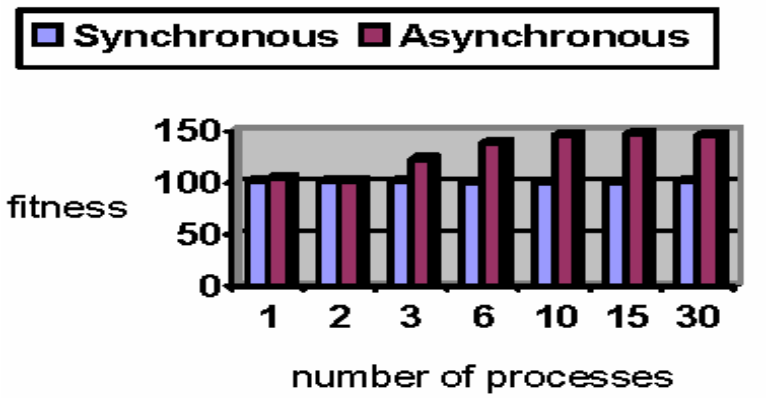

Figure 5.12 Impact du nombre de processus sur la qualité de la solution

C. Impact du moment de communication

Les trois cas décrit dans la section V.2.2.2 correspondent aux différents moments d'échange d'information.

La table 5.2 montre que le cas (c) donne les meilleurs résultats. Ceci indique que quand les processus s'exécutent indépendamment, ils sont plus performants en s'échangeant les solutions seulement après l'application de l'opérateur *Move*. Dans le cas (a) beaucoup d'échanges sont faits sans ameliorer la performance de l'algorithme et le cas (b) est clairement moins efficace que les deux autres cas. On conclue que l'opérateur *Move* produit de meilleures solutions à échanger que l'opérateur swap.

(|V|,|S|,|Z|,|W|)	(a)	(b)	(c)
(30,10,10,20)	108.99	135.81	**101.95**
(50,23,30,20)	56.53	104.34	**46.29**
(100,23,20,20)	48.99	75.11	**45.61**

Tableau 5.2 Moment de la communication inter-processus

D. Impact de la fréquence de communication

Tout comme le moment de communication, la fréquence de communication influence la qualité des résultats. Dans [9] les auteurs ont montré qu'une fréquence moyenne de communication peut donner de meilleurs résultats. Dans notre cas (Tableau 5.3) montre que la fréquence de communication nécessaire pour trouver de bons résultants doit augmenter avec la taille de l'instance.

Step[1]	1		2		5		10	
Instance	Min[2]	Mean[3]	Min	Mean	Min	Mean	Min	Mean
(10,10,5,5)	19.50	23.57	19.50	23.85	**19.50**	**23.40**	20.89	36.45
(10,10,10,5)	58,53	62,05	58,22	62,10	**58.03**	**61.94**	59.68	77.01
(10,10,15,10)	246.95	249.68	**246.95**	**248.91**	247.74	254.50	251.07	255.04
(30,10,10,20)	101.84	115.04	**101.84**	**113.16**	108.55	116.82	109.43	117.80
(30,23,30,5)	**14.38**	**18.57**	14.59	20.15	14.38	21.89	18.20	25.40
(30,23,20,30)	**441.00**	**471.38**	448.75	473.73	453.24	478.78	454.61	481.66

Tableau 5.3 Fréquence de la communication inter-processus

E. **Comparaison des algorithmes séquentiel, parallèle synchrone et parallèle asynchrone :** Le tableau 5.4 montre une comparaison entre les résultats obtenus par l'algorithme séquentiel et ceux des deux stratégies parallèles. Nous avons reporté le minimum, le maximum, la moyenne, médiane et l'écart type de 100 exécutions pour les instances de 1 à 5 et de 50 exécutions pour les instances de 6 à 10. Les résultats sont obtenus pour les mêmes valeurs de paramètres pour les trois algorithmes. On remarque que les valeurs minimales et maximales de la fonction objectif trouvées par les versions séquentielles et parallèle synchrones sont presque les mêmes. Cependant, la moyenne et la médiane de l'algorithme synchrone sont meilleures que la méthode séquentielle. En plus l'algorithme synchrone est plus rapide que le séquentiel.

Les valeurs moyennes de l'algorithme asynchrone sont moins performantes que les algorithmes séquentiel et synchrone mais il est plus rapide que les deux.

Dans le tableau 5.5 nous comparons les résultats quand l'algorithme séquentiel est exécuté pour le même temps que l'algorithme asynchrone. On remarque que pour les petites instances (1-5) il n'existe pas de différence entre les deux méthodes, cependant la différence est plus notable sur de plus grandes instances où l'algorithme asynchrone est plus performant.

[1] *Step* représente le nombre d'itérations entre deux échanges d'information consécutifs.
[2] La plus petite valeur de la fonction objectif trouvée.
[3] La plus grande valeur de la fonction objectif trouvée.

	Sequentiel					Synchrone					Asynchrone				
	Min	Max	Mean Mdn	Std	Avg_ time	Min	Max	Mean Mdn	Std	Avg- time	Min	Max	Mean Mdn	Std	Avg_ time
1	19.50	25.74	24.39 25.74	1.88	1,50	19.50	25.23	**22.45** **22.92**	1.40	0.76	19.04	28.48	23.69 24.20	1.03	0,50
2	58.03	64.20	62.00 62.32	2.13	2,05	58.03	64.06	**60.00** **60.19**	1.38	1.15	58.03	78.45	62.27 62.85	2.28	0,80
3	166.95	170.33	167.98 167.72	0.99	2,59	166.95	168.75	**167.35** **167.40**	0.40	1.80	166.95	172.34	168.91 169.03	1.18	0,93
4	246.95	250.86	248.39 247.97	1.10	4,60	246.95	249.90	**247.55** **247.40**	0.67	2.10	233.85	252.35	248.61 248.31	2.01	1,10
5	97.59	109.44	102.95 102.65	2.60	65,73	97.92	104.13	**100.98** **100.83**	1.06	24.82	101.84	119.02	112.37 112.80	3.37	18,84
6	443.77	478.26	453.32 452.65	7.06	310,67	427.77	458.64	**449.65** **451.43**	6.75	108,39	441.00	469.75	456.38 457.28	6.66	89,82
7	41.07	119.44	57.65 52.96	16.6	230,37	40.69	63.82	**51.15** **50.85**	4.96	95,84	41.89	67.94	54.55 53.96	5.25	86,0
8	133.66	174.06	160.60 154.11	19.2	516,5	127.92	166.66	**140.56** **138.86**	8.08	221,70	101.14	177.82	152.77 156.04	15.58	155,90
9	38.44	53.08	45.44 45.23	2.98	1131,7	39.92	48.01	**44.13** **43.80**	2.04	727,14	42.05	53.08	48.72 48.34	3.52	400,82
10	60.00	75.97	**68.90** **69.70**	3.90	899,50	62.89	82.06	70.91 70.72	4.44	873,45	71.92	130.20	100.56 98.33	14.06	419,21

Tableau 5.4 Comparaison des deux stratégies parallèles et l'algorithme séquentiel

	Séquentiel				Asynchrone			
	Min	Max	Mean Mdn	Std	Min	Max	Mean Mdn	Std
1	19.50	25.74	24.85 25.74	1.60	19.04	28.48	**23.69** **24.20**	1.03
2	58.53	66.14	63.48 64.20	1.53	58.03	78.45	**62.27** **62.85**	2.28
3	166.95	171.33	**168.79** **168.54**	1.22	166.95	172.34	168.91 169.03	1.18
4	247.40	252.99	250.19 250.86	1.13	233.85	252.35	**248.61** **248.31**	2.01
5	99.84	117.89	**106.91** **106.61**	3.64	101.84	119.02	112.37 112.80	3.37
6	445.01	478.26	458.76 458.09	6.62	441.00	469.75	**456.38** **457.28**	6.66
7	49.83	115.19	80.98 80.57	15.44	41.89	67.94	**54.55** **53.96**	5.25
8	133.66	184.33	161.50 161.25	6.08	101.14	177.82	**152.77** **156.04**	15.58
9	80.57	187.99	138.59 138.92	25.11	42.05	53.08	**48.72** **48.34**	3.52
10	85.03	265.70	169.49 170.08	36.52	71.92	130.20	**100.56** **98.33**	14.06

Tableau 5.5 Comparaison l'algorithme séquentiel et le parallèle asynchrone pour le même temps d'exécution

On peut résumer les résultats dans cette section comme suit :

- Quand on utilise l'approche de l'évaluation parallèle du voisinage, il est préférable d'opter pour un schéma maître-esclave synchrone pour garder une bonne qualité des solutions tout en accélérant l'algorithme séquentiel original.
- Si nous cherchons une méthode plus rapide, l'algorithme asynchrone est convenable mais nécessite plus de contrôle de l'information ainsi que de la fréquence d'échange pour atteindre des résultats acceptables.
- Echanger seulement les solutions trouvées par l'opérateur *Move* dans l'approche asynchrone est préférable que d'échanger toutes les solutions.
- Dans la méthode asynchrone, les grandes instances nécessitent une fréquence de communication inter-processus plus importante que les petites instances.

V.3 Conclusion

Dans ce chapitre nous avons contribué à la modélisation du problème de gestion des véhicules d'urgence en intégrant le dispatching et la couverture, ensuite nous avons étudié l'effet cette intégration sur la qualité de service. Les résultats obtenus sont encourageants puisque le gain obtenu en couverture des zones pour le nombre de zones ainsi que le temps de couverture est plus important que la baisse en temps de réponse. Pour la résolution du modèle nous avons proposé une metaheuristique hybride parallèle et étudié les conditions favorables liées à la synchronisation des processus parallèles, et l'échange de l'information afin d'améliorer la performance en qualité et en temps d'exécution.

Dans le chapitre suivant nous abordons le problème du point de vue de la distribution qui est une caractéristique inhérente aux systèmes de gestion des véhicules d'urgence.

VI Le système de gestion des véhicules d'urgences distribué

La structure du système DISCOV et la répartition géographique de ses composants lui donne une nature facilement distribuable. La résolution centralisée de ce système nous conduit à faire des suppositions pas très réalistes. En effet, les organisations réelles des systèmes d'urgence sont caractérisées par un environnement dynamique, et dans la majorité des méthodes d'optimisation centralisées beaucoup d'informations doivent être connues à l'avance. On doit par exemple supposer que le centre de prise de décision connaît à tout moment les emplacements, les états et la destination de tous les véhicules dans le système. D'un côté ceci devient difficile à gérer avec un nombre important de véhicules surtout lorsque la région sous contrôle est grande et que la fréquence et le nombre d'appels arrivant est élevé. D'un autre côté, le fait de confier la gestion du système complet à un seul centre va réduire le contrôle des stations sur leurs véhicules et leurs équipes ce qui n'est pas très pratique.
Un autre inconvénient important, qui est inhérent à l'architecture centralisée, est que l'échec du centre de décision mène à l'échec de tout le système ce qui peut avoir des conséquences sérieuses dans le contexte des urgences.

Les solutions distribuées présentent les avantages suivants:

- Elles respectent la nature distribuée du problème ce qui le rend facile à comprendre et à implémenter
- Elles sont plus adaptables car les différents composants du système ont des décisions moins complexes à prendre, elles sont donc capables de mieux contrôler leur environnement et réagir aux changements plus rapidement ; alors que les algorithmes d'optimisation centralisés sont plus sensibles aux changements et les moindres variations peuvent avoir un impact sur les plans de plusieurs véhicules.
- L'intégrité des données dans chaque composant peut facilement être assurée.
- Le système est décomposé en des unités de calcul indépendantes ce qui augmente la puissance du calcul considérablement et ainsi la résolution du problème peut être accélérée contrairement aux solutions centralisées qui peuvent ne pas permettre des réponses rapides aux événements d'urgence.

Dans ce chapitre nous proposons deux organisations distribuées pour le système DISCOV et pour chacune d'elles nous proposons des méthodes pour optimiser le modèle de la section V.1 de manière distribuée, c'est-à-dire en gardant la propriété de distribution des données, des

traitements et de la prise de décision. Ces méthodes seront comparées entre elles ainsi qu'à la version centralisée du système pour l'étude de performance.

La différence entre les deux organisations réside dans la nature de coopération inter agents. Dans la première, quand une urgence arrive, l'agent *appel* contacte son agent *zone,* ce dernier contacte à son tour tous les agents *station* ; chaque station collecte les informations concernant ses agents *véhicule,* fait une proposition à la zone qui décide quel véhicule accepter en se basant sur un processus de coordination implicite. Alors que dans la seconde organisation, la coordination et l'optimisation se fait par un processus de recherche impliquant une communication directe entre les stations ce qui permet de prendre les décisions de l'affectation des véhicules tout en optimisant le modèle.

VI.1 Gestion intégrée des véhicules d'urgence multi-agent "MA-DISCOV"

Dans cette section nous proposons une architecture multi agents au système DISCOV que nous appelons *MA-DISCOV*.

Dans cette architecture (Figure 6.1) un agent *appel* est sous la responsabilité d'un agent *zone*. Ce dernier interagit avec les agents stations pour trouver le véhicule approprié pour répondre à l'appel. D'un autre côté, un agent véhicule est lié à l'agent *station* et les deux agents communiquent pour s'échanger les informations ainsi que les décisions prises.

VI.1.1 Les agents

- **Agent Appel "CA"**

  Chaque appel d'urgence est pris en charge par un agent *appel* qui a la responsabilité de contacter son agent *zone* pour demander un véhicule d'urgence. La demande contient l'emplacement de l'appel et sa priorité. Après le processus de prise de décision qui implique les agents stations et véhicules, l'agent *zone* envoie à l'agent *appel* une estimation du temps d'arrivée du véhicule comme réponse à sa demande.

- **Agent Zone "ZA"**

  L'agent *zone* a le rôle de recevoir les demandes des agents appels et de les transmettre aux agents stations. Après la prise de décision, il informe l'agent *appel* du temps d'arrivée estimé du véhicule.

- **Agent Station "SA"**

  Lorsqu'un agent *station* reçoit une demande d'un agent *zone*; il contacte ses véhicules pour avoir les informations actualisées, il participe au processus de coordination pour

l'affectation des véhicules et informe ses véhicules des éventuelles affectations. A la fin de service d'un véhicule, la station a aussi le rôle de trouver la meilleure destination au véhicule qui améliore la performance du système.

- **Agent véhicule "VA"**

Cet agent agit sous le contrôle de l'agent *station*. Principalement, il informe l'agent *station* du temps de voyage estimé de sa position courante aux autres emplacements. Quand il termine le service d'un appel, il envoie une demande à son agent *station* pour lui indiquer sa prochaine destination. Il doit également l'informer de son arrivée à l'appel/station.

VI.2 Solutions distribuées

Quand un appel d'urgence arrive, une interaction doit être établie entre les différents agents pour déterminer le véhicule à affecter à l'appel.

La prise de décision distribuée implique une certaine rigueur, car les données et les traitements étant distribués, chaque agent *station* a accès seulement aux informations de ses propres véhicules, il va choisir un de ses véhicules pour l'affecter à l'appel et si nécessaire redéployer le reste de ses véhicules pour garder une bonne couverture des zones. Si on suppose que chacune des stations prend ces décisions localement sans communication ni coordination avec les autres agents, ce processus peut mener à une situation chaotique pour le système complet. Pour cette raison, et afin d'éviter de confier cette tâche à un seul agent pour ne pas retrouver les même inconvénients que le système centralisé, nous proposons deux solutions pour coordonner les agents.

VI.2.1 Coordination par le mécanisme des enchères

Tel qu'il a été expliqué dans le chapitre III, dans ce genre d'approches, les agents enchérisseurs (bidders) font des offres à l'un des agents (auctioneer) suite à un appel d'offre, et cet agent décide quelle offre accepter. Dans la solution que nous proposons dans cette section, les agents stations font des offres à l'agent *zone* (responsable de l'appel) et ce dernier décide de l'offre à accepter (figure 6.2).

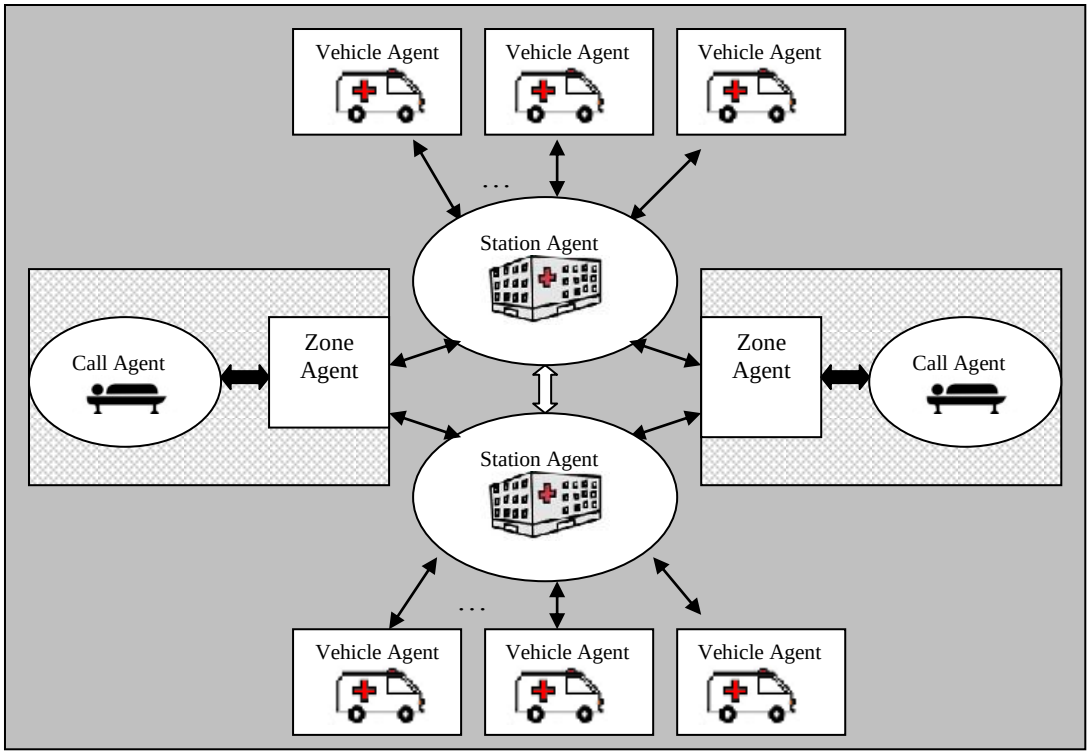

Figure 6.1 MA-DISCOV

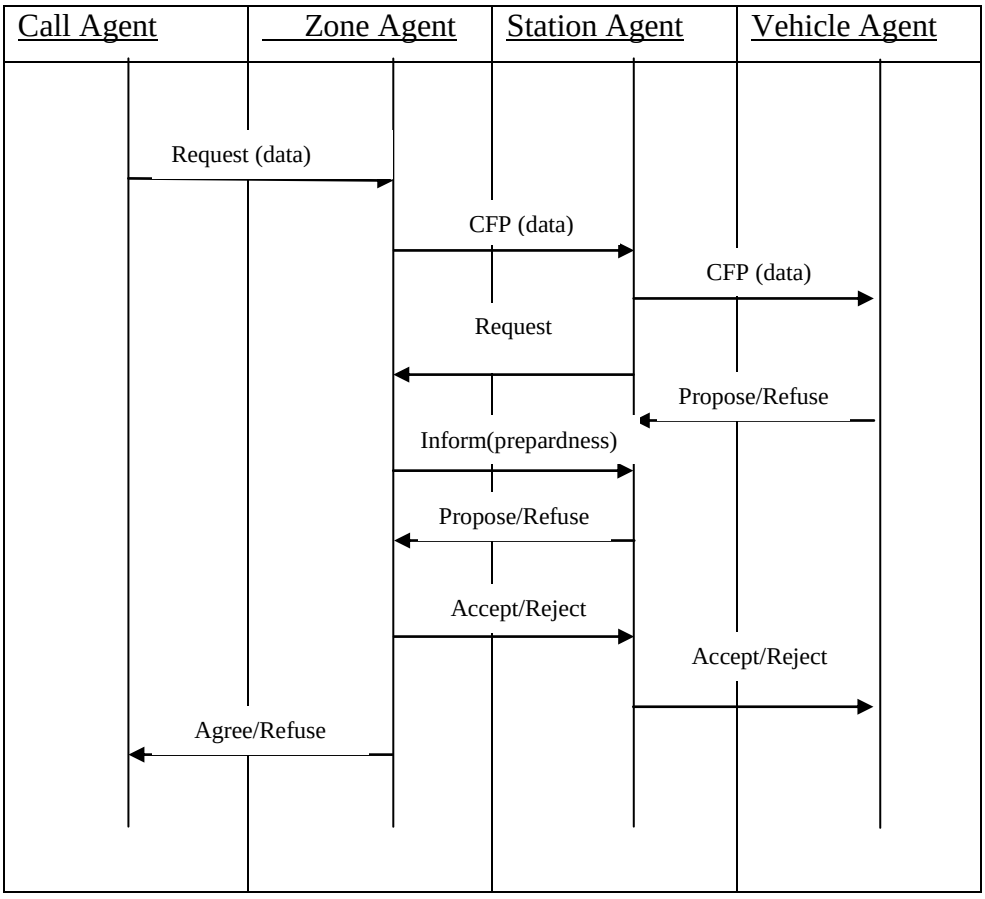

Figure 6.2 Diagramme de séquence

Notre objectif est de coordonner les agents à travers un mécanisme des enchères qui permet d'aboutir à des solutions distribuées comparables aux solutions centralisées en terme de fonction objectif.

A. Approche1 : négociation basée sur les enchères

Dans un premier temps, les agents interagissent pour affecter un véhicule à l'appel : quand une urgence arrive, l'agent *appel* envoie un message de demande à son agent *zone*. Ce dernier lance un appel d'offre aux stations. Les agents station contactent leur véhicules pour avoir leur temps de réponse estimé vers cet appel, choisissent le meilleur véhicule en terme de fonction objective, ensuite envoient leurs propositions à l'agent *zone*. L'agent *zone* choisit la meilleure proposition (en terme de fonction objectif), envoie des réponses d'acceptation/rejet aux stations et informe l'appel du temps d'arrivée estimé du véhicule.

Pour limiter les déviations, nous permettons seulement l'affectation d'un véhicule à un autre appel si celui-ci est plus prioritaire que le premier. Et pour éviter de pénaliser le même appel continuellement, on incrémente sa priorité à chaque fois jusqu'à ce qu'il atteigne la plus haute priorité.

Dans la seconde étape, la station S_i dont le véhicule à été accepté dans l'étape précédente, vérifie si la couverture des zones a été affectée par la décision de l'étape précédente. Si c'est le cas, S_i envoie un appel d'offre (recovery) à toutes les stations S_j:j≠i. Quand une station S_j reçoit un appel d'offre *(recovery),* elle le fait suivre à ses véhicules et collecte le temps de leur déplacement vers S_i, elle calcule le gain en couverture pour chaque véhicule et envoie le meilleur résultat à S_i. Quand elle reçoit la réponse à son offre elle avise le véhicule par acceptation/rejet.

S_i sélectionne la meilleure proposition (avec le maximum de gain en couverture). En cas, la proposition dont le temps est le plus court est choisie.

When receive *CFP(call)* from zone *Z*

```
Collect_data_veh();
Best_veh_to_call(); //évalue la fonction objectif pour chaque(veh,call) et retourne la  meilleure
Envoyer(fitness, veh, time) à  Z
Attendre réponse de Z
Si (Accept) alors
  Envoyer Accept à veh
  Si (nbuncov0<nbuncov)
     Lancer un CFP (recovery) à toutes les stations
  Fsi
```

Sinon
Envoyer *Reject* à veh
Fsi

When receive CFP(recovery) from station S_i

Collect_data_veh();
Best_veh_to_stat();//calcule le gain en nombre de zones couvertes pour chaque (veh, S_i) et retourne le. // meilleur
Envoyer(fitness, veh, time) à S_i
Attendre réponse de S_i
Si (*Accept*) alors
Envoyer *Accept* à veh

Figure 6.3 Approche1

B. Approche2

Dans cette approche nous utilisons un mécanisme d'enchères incluant tous les emplacements (appel & stations). L'agent zone joue le rôle du commissaire-priseur (auctioneer) et les véhicules ont le rôle d'enchérisseur. Chaque agent véhicule est représenté par son agent station.

Quand un agent station "SA_k" reçoit un CFP d'une zone donnée, il demande à ses véhicules "$V(SA_k)$" une estimation du temps de déplacement de leurs positions vers tous les emplacements ($W \cup S$), ensuite il calcule la valeur de la fonction objectif locale pour chaque couple (véhicule,location) comme suit :

Soit (v_i,l_j) le couple du i^{éme} véhicule et le j^{éme} emplacement. $l_j \in W \cup S$, $v_i \in V(SA_k)$

$$fitness(v_i, l_j) = \begin{cases} T_{il_j}(t).Prio_j + expr1 + expr2 + expr3 & \text{if } l_j \in W \\ T_{il_j}(t) + expr1 + expr2 + expr3 & \text{if } l_j \in S \end{cases}$$

$$\text{Expr1} = Penal1 \sum_{i \in W} unsat_i(t)$$

$$\text{Expr2} = Penal2 \sum_{z \in Z} uncov_z(t)$$

$$\text{Expr3} = Penal3 \sum_{j \in V} dev_j(t)$$

Chaque véhicule fait une offre pour le meilleur emplacement. La valeur de l'offre est égale à la différence entre la valeur de la fonction objectif des deux meilleurs emplacements.

De son côté, l'agent zone affecte l'appel à la meilleure offre (avec la valeur minimale puisque les offres sont négatives). Si un agent véhicule fait une offre pour une station alors son offre est acceptée. Les agents véhicules sont affectés à l'emplacement suivant si leur offre pour l'appel est refusée.

Une différence importante entre les deux approches est la complexité en communication :

- La taille des messages contenant les offres dans la seconde approche est plus grande que celle des messages de la première approche car chaque station fait une offre pour tous ses véhicules et l'offre doit contenir au moins le premier et le second meilleur emplacement pour chaque véhicule. L'offre doit également contenir le temps de voyage estimé. Dans la première approche le message des stations contient seulement l'offre pour le meilleur véhicule avec le temps de voyage estimé.
- Le nombre de messages pour la première approche est cependant plus grand que la deuxième au pire des cas c'est à dire quand la couverture des zones est affectée par l'affectation du véhicule à l'appel un second mécanisme d'enchère est lancé pour la restitution de couverture ce qui implique l'envoie de plus de messages.

C. Résultats expérimentaux

Le système multi-agent proposé a été implémenté en langage java sous la plateforme JADE "Java Agent Development Environment" qui prend en charge la création des agents et la gestion des communications et des différents protocoles.

- ***La fonction objectif***

Dans ce qui suit nous comparons les deux approches en terme de fonction objectif donnée dans la section V.1. Dans le système MA-DISCOV que nous proposons, il n'existe aucune entité où les données nécessaires à l'évaluation de la fonction objectif sont présentes. Cependant, nous avons ajouté un module dans le simulateur pour calculer cette valeur à des fins de comparaison.

Les résultats de la comparaison pour dataset1 (figure6.4) et dataset2 (figure6.5) montrent que l'approche2 améliore l'approche1 avec 35.69% pour dataset1 et 25.13% pour dataser2.

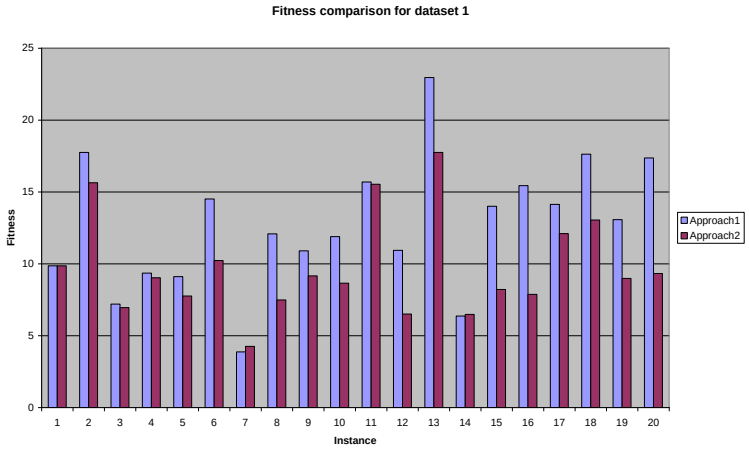

Figure 6.4 Comparaison de la fonction objectif pour dataset1

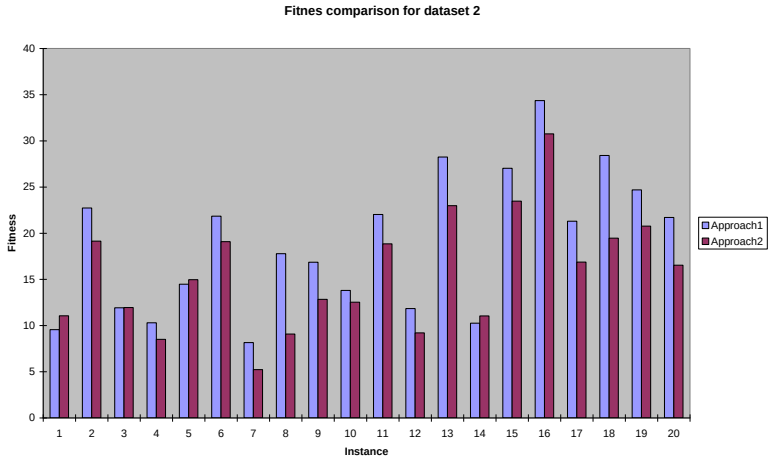

Figure 6.5 Comparaison de la fonction objectif pour dataset2

Afin d'analyser les conséquences de cette différence sur la qualité de service : temps de réponse et couverture des zones nous avons comparé aussi ces deux critères.

- ***Temps de réponse moyen***

Les résultats représentés dans les figures 6.6 et 6.7 montrent que l'approche1 est également la moins performante en temps de réponse pour les deux ensembles.

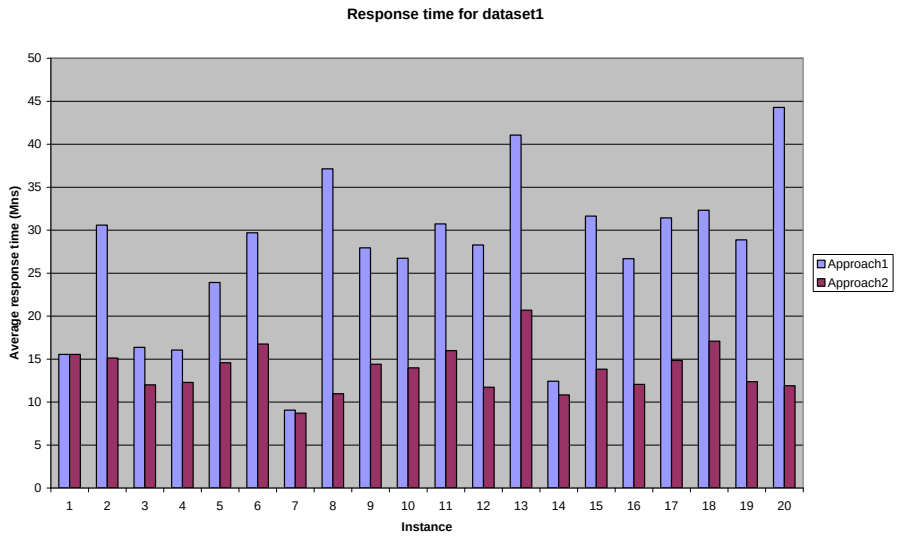

Figure 6.6 Comparaison du temps de réponse dataset1

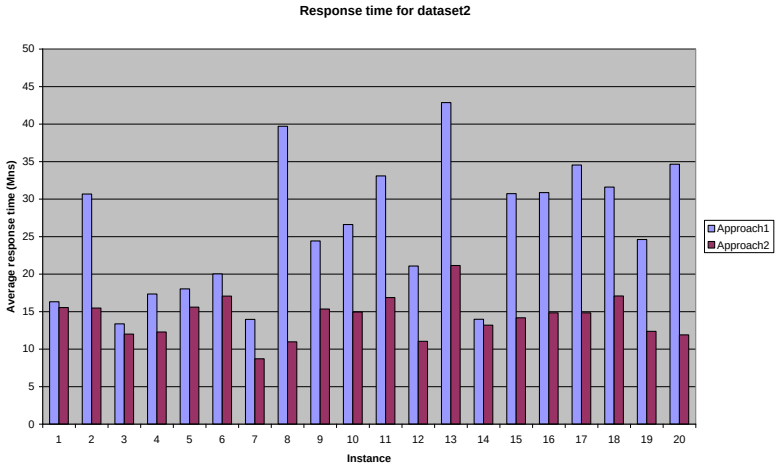

Figure 6.7 Comparaison du temps de réponse pour dataset2

- ***Couverture des zones***

La table 6.1 résume les résultats obtenus en terme de couverture des zones pour les deux ensembles de données.

Nous nous sommes basé sur deux critères : le pourcentage des zones non couvertes et le pourcentage de temps de non couverture par rapport au temps complet de la simulation.

Les résultats montrent que pour dataset1 la première approche est meilleure que la seconde pour les deux critères alors que pour dataset2 la deuxième approche donne de meilleurs résultats surtout pour le temps de couverture.

		Pourcentage de zones non couvertes	Pourcentage de temps de non couverture.
Dataset 1	Approach 1	2.05%	3.91%
	Approach 2	2.85%	4.36%
Dataset 2	Approach 1	9.55%	17.37%
	Approach 2	8.7%	11.62%

Tableau 6.1 Résultats de la couverture des zones.

VI.2.2 Coordination par communication directe

Dans cette section nous abordons un autre type de solutions basé sur la recherche distribuée dans laquelle la résolution du modèle émerge d'une coopération directe entre les agents stations. L'adaptation des algorithmes de recherche locale distribuée tels que le DBA et le DSA à notre problème nécessite l'utilisation d'un graphe complet à cause de la fonction globale, ce qui a l'inconvénient de produire un surcoût en communication. L'algorithme Max-sum quant à lui est connu pour sa convergence rapide mais son application dans ce cas demande la définition d'un nœud fonction pour la fonction globale sous le contrôle de l'un des agents qui doit être lié à tous les autres nœuds variables et qui aura la responsabilité de recevoir leurs valeurs, d'évaluer et d'optimiser cette fonction et d'envoyer le résultat à ces nœuds ce qui va aboutir à une architecture centralisée et contraint grandement nos objectifs en distribution. Nous commençons cette section donc par l'adaptation du protocole SynchLDS à notre problème, et nous montrons que pour cette méthode le processus de recherche peut être très influencé par l'ordre de priorité des agents. Pour pallier à cet inconvénient, nous proposons un protocole de coordination globale distribuée " PCGD " qui permet l'échange des solutions complètes entre tous les agents ce qui accélère à la fois le processus de recherche.

A. La SynchLDS au Problème de gestion intégrée des véhicules d'urgence

Pour adapter la SynchLDS telle qu'elle est décrite dans la figure 6.8 on doit :

1. définir un ordre de priorité entre les agents stations.
2. considérer qu'il existe une heuristique qui génère les solutions locales de la meilleure à la moins bonne solution.

D'un côté, pour la plupart des problèmes, il n'est pas facile de trouver une telle heuristique à cause de la complexité de leurs modèles. D'un autre côté, cette méthode peut être efficace

quand chaque agent a une seule variable à contrôler et les domaines des variables sont petits. Dans le cas contraire, le processus de recherche devient long et ne garantit pas de trouver de bonnes solutions lorsqu'il est interrompu précocement. Ceci devient un important inconvénient surtout pour les applications critiques où aussi bien le temps d'exécution que la qualité de la solution sont importantes, comme c'est le cas pour le problème de la gestion des véhicules d'urgence.

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

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

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

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

WhenReceive MsgBacktrack(node) do
   Work(node)

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

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

Figure 6.8 SyncLDS – Pseudocode [47].

Pour cette raison nous adaptons la SynchLDS à notre problème en utilisant une procédure de recherche locale (figure 6.9) qui génère des solutions de la moins bonne à la meilleure. Ainsi le critère de sélection du nœud suivant à développer devient le maximum de déviations au lieu du minimum de déviations.

Dans les figures 6.10, 6.11, 6.12, 6.13 suivantes nous montrons l'effet de l'ordre de priorité sur la qualité de la solution obtenue par la SynchLDS adaptée au PGIVU. Nous comparons les deux cas suivant:

1. Les stations triées par ordre croissant du nombre de véhicules libres.
2. Les stations triées par ordre décroissent du nombre de véhicules libres.

```
Recherche locale(S)
Assign←∅ //liste des affectations
F*=fitness(S)
Tantque (¬condition arrêt)
      Sélectionner véhicule v
       l*=l ;//emplacement actuel de v dans S
      Pour tout emplacement l
        f=fitness(S,v,l)//la function objectif obtenue par l'affectation de v à l dans S
        Si (f<f*)
            l*=l ;
            f*=f ;
        fsi
      fpour
     Ajouter (v,l*) à Assign .
Ftq
Return Assign
```

Figure 6.9 Procédure de recherche locale

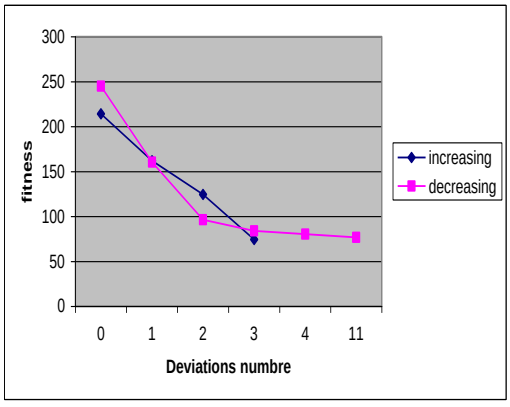

Figure 6.10 SynchLDS pour instance1

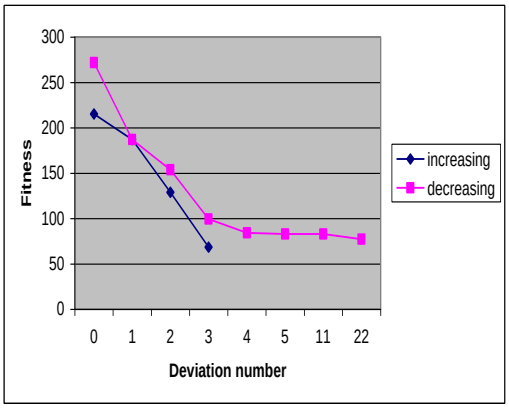

Figure 6.11 SynchLDS pour instance2

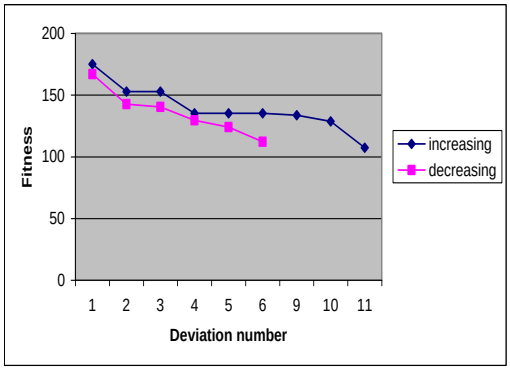

Figure 6.12 SynchLDS pour instance3

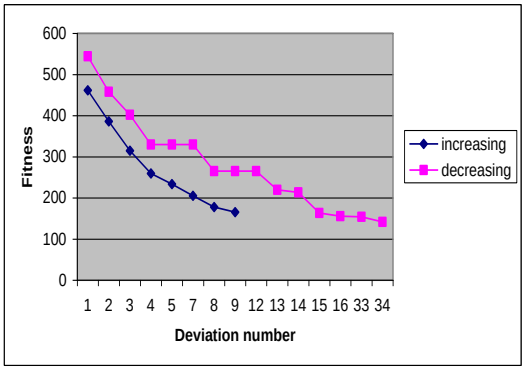

Figure 6.13 SynchLDS pour instance4

Les résultats montrent que l'ordre de priorité influence la qualité des solutions trouvées. La différence de la séquence des solutions visitées est importante ce qui signifie que si l'algorithme est interrompu avant son exécution complète par faute de temps, la différence du résultat obtenu sera très importante pour les deux priorités.

B. Le protocole de coordination globale distribuée "PCGD"

Le protocole que nous proposons combine le principe des méthodes de recherche distribuée où chaque agent reçoit la somme des fonctions des autres agents et décide quelle valeur attribuer à sa variable pour optimiser l'objectif global (tel que l'algorithme max-sum), avec l'idée du SynchLDS où chaque agent produit des solutions locales ordonnées et les explore dans l'ordre.

Sans perte de généralité, nous considérons le cas où le problème d'optimisation distribué à résoudre a une fonction objectif liée à l'état global du système et qui peut être décomposée comme suit:

$F=F_1(x_1)+ F_2(x_2)+\ldots+ F_n(x_n)+ F_{n+1}(x_1, x_{2\ldots} x_n)$.

Un ensemble de n agents $\{A_1,A_2,\ldots.,A_n\}$

Un ensemble de n variables $\{x_1, x_{2,\ldots,} x_n\}$ contrôlées respectivement par les agents $A_{1,} A_{2\ldots}A_{n.}$

$F_i(x_i)$ est la fonction utilité de l'agent $A_{i.}$ et $F_{n+1}(x_1, x_{2\ldots} x_n)$,i=1..n est la fonction de toutes les variables dans le système qui peut, être liée à des contraintes globales.

Il est important de noter que $F_i(x_i)$ n'est pas nécessairement l'objectif local de l'agent$_i$ mais plutôt la part de contribution de l'agent$_i$ dans l'objectif global.

Dans le protocole proposé nous définissons :

1. une relation de précédence entre les agents en donnant des identificateurs de 1..n aux agents pour définir leur ordre de priorité de sorte que chaque agent a un prédécesseur (sauf pour le premier) et un successeur (sauf pour le dernier). Cet ordre peut être déterminé heuristiquement ou arbitrairement.
2. on considère qu'une solution initiale complète existe et est connue par tous les agents. Elle peut être l'état courant du système (comme dans notre exemple) ou bien obtenue par une étape d'initialisation.

Chaque agent garde deux structures (figure 6.14 et 6.15). La première est la liste des nœuds reçus des autres agents (son prédécesseur ou le dernier agent), et pour chaque nœud, la liste des solutions générées localement partant de la configuration de ce nœud et utilisant la procédure d'optimisation locale.

La seconde structure contient les référence *(#Node, #Solution)* ordonnée de la meilleure à la moins bonne fonction objectif. Cette structure est utilisée pour sélectionner la solution suivante à explorer plus rapidement : avec une liste triée, on peut chercher les éléments avec des algorithmes à complexité logarithmique tel que la recherche dichotomique.

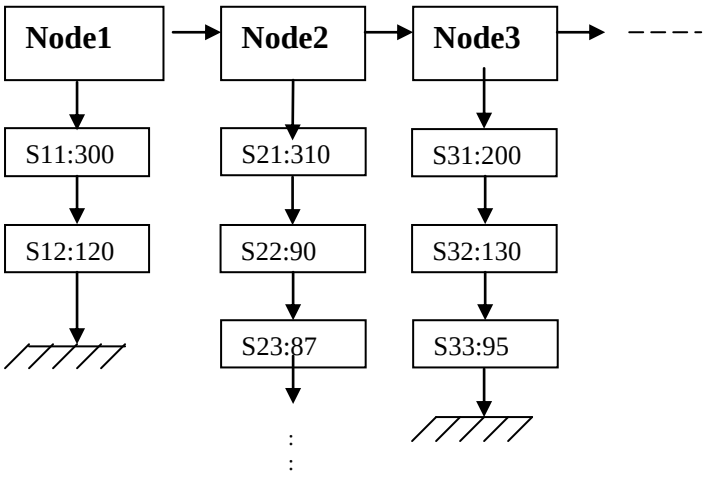

Figure 6.14 Liste des solutions

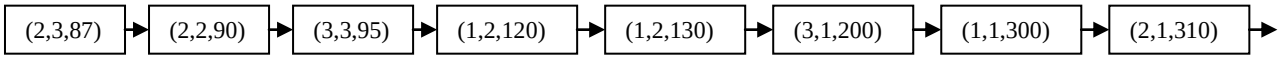

Figure 6.15 Liste des références

Le PCGD est un protocole synchrone composé de deux phases consécutives qui sont répétées itérativement jusqu'à la condition de terminaison : La phase de calcul et la phase de communication.

Dans la phase de calcul tous les agents appliquent la procédure d'optimisation locale à la meilleure solution dans leur liste qui n'a pas encore été considérée. Quand l'agent trouve une nouvelle solution il l'ajoute à la liste.

La phase de communication commence quand le premier agent (le plus prioritaire) termine le calcul d'une nouvelle solution et envoie à son successeur un message *NEW* contenant le nœud de la meilleure solution dans sa liste (qui n'est pas nécessairement la solution la plus récemment trouvée). Les autres agents (sauf le premier) attendent la réception d'un message *NEW* de leur prédécesseur (s'il n'a pas été encore reçu avant la terminaison de la phase de calcul), insèrent le nœud reçu dans leur liste s'il n'existait pas déjà et envoient à leur tour la meilleure solution de la liste qui n'a pas encore été envoyée.

De cette façon, à la fin de chaque phase de communication le dernier agent (le moins prioritaire) connaît la meilleure solution de tous les agents dans ce cycle. Cette solution est aussi connue par tous les agents les moins prioritaires que celui qui l'a trouvé. Le dernier agent envoie un message *BEST* contenant la meilleure solution dans sa liste à tous les agents qui ne l'ont pas encore reçu (les plus prioritaires que l'agent origine de cette solution). Pour avoir cette information, nous utilisons un index appelé "*origin*" inclus dans la solution pour former un nœud ; il est égal à l'identificateur de l'agent le plus prioritaire qui a trouvé cette solution.

Quand un problème contient une contrainte globale, un agent qui a produit une nouvelle solution qui ne satisfait pas la contrainte vérifie s'il est le n^{ieme} agent qui a reçu cette solution, si c'est le cas il l'élimine et elle ne sera plus envoyée dans la phase de communication.

Quand un agent ne trouve pas une nouvelle solution dans la phase de calcul et que toutes les solutions dans sa liste ont été envoyées ; il envoie un message *NULL* à son successeur. Une nouvelle phase de calcul commence quand le dernier agent envoie son message *BEST*, et le protocole se termine quand le dernier agent reçoit un message *NULL* et n'a pas de nouvelle solution à envoyer. On peut utiliser d'autres conditions d'arrêt comme un temps d'exécution limite ou une qualité désirée de la meilleure solution globale.

En permettant l'envoie de la meilleure solution à tous les agents nous leur donnons la possibilité d'effectuer le calcul concurremment sur la meilleure solution globale ce qui estompe les effets de l'ordre de priorité.

```
When receive BEST( sol,origin)
If((sol ≠NULL)&&(sol∉list))
   Insert sol into the list
   sol.sent=true
S = LS(S*)     //LS is the local procedure that return a new solution starting
                //from S*:  the best solution in the list not yet explored
Insert S into the list.
S1=the best solution in the list not yet sent
If (predecessor = -1)  //The first agent
   Send S1 to the next agent
   S1.sent=true
   Wait for the message of the last agent
Else
   Wait for the NEW message of the predecessor agent.
```

Figure 6.16. Phase de calcul

```
When receive New(sol,origin)
If(sol ≠ NULL)
    If(sol ∈ list)
       update the origin index if necessary
    Else
       Insert sol into the list
Endif
S = NULL
S = the best solution in the list not sent
If(next ≠ -1)
   Send S to the next agent
   S.sent=true
   wait for the message of the last agent.
Else
   If((S = NULL) && (sol = NULL))
      Send NULL message to all the agents
    Else
      Send S to the agents with priority > S.origin
      S.sent=true
      Start a new computation cycle.
```

Figure 6.17. Phase de communication

Proposition :

- L'algorithme se termine quand le dernier agent reçoit un message NULL et n'a aucune solution à envoyer aux autres agents.
- Un agent termine l'exploration quand aucun nœud dans sa liste ne peut produire une nouvelle solution et que toutes les solutions qui sont dans la liste ont été envoyées au successeur.

Preuve :

Nous allons prouver que :

A. Si l'algorithme se termine alors tous les agents ont fini l'exploration de leurs solutions.

B. Si tous les agents finissent l'exploration de leurs solutions alors l'algorithme se termine.

Pour la première implication nous procédons par contradiction : on suppose qu'il existe un agent A_i qui n'a pas fini l'exploration de ses nœuds alors que l'algorithme se termine.

L'agent A_i n'a pas fini l'exploration de ses nœuds veut dire :

1. qu'il y a au moins un nœud dans sa liste qui peut produire une nouvelle solution "S_1" qui n'existe pas déjà dans sa la liste. Ou bien

2. il existe au moins une solution "S_2" dans sa liste qui n'a pas encore été envoyée.

Dans les deux cas, A_i aurait envoyé S_1/S_2 à A_{i+1} dans le cycle de communication courant. Mais puisque les agents $A_{i+1}…A_n$ ont fini l'exploration (d'après la supposition), ils ont donc envoyé un message NULL dans le cycle courant de communication : on conclue que S_1/S_2 est déjà connue et envoyée par $A_{i+1}…A_n$, ce qui veut dire que A_n l'a déjà reçu mais ne l'a pas encore envoyé à A_i (car dans le cas contraire A_i n'aurait pas considéré S_1 comme nouvelle solution et S_2 aurait été marquée '*SENT*') ce qui implique que A_n a encore dans sa liste au moins une solution (S_1/S_2) non envoyée et donc l'algorithme ne s'est pas terminé ce qui est en contradiction avec notre supposition de départ.

Si les agents $A_1…A_{n-1}$ ont fini l'exploration alors ils n'ont pas produit de nouvelles solutions dans la phase de calcul et toutes les solutions dans leurs listes ont été envoyées, donc ils ont envoyé un message NULL à leurs successeurs et ainsi l'agent A_n aurait reçu un message NULL et si A_n a lui aussi fini l'exploration alors il n'aura aucune solution à envoyer : les deux conditions seront satisfaites et l'algorithme se termine.

Les deux implications A et B étant vraies, la terminaison et la complétude sont donc prouvées.

C. Le protocole de coordination globale distribuée appliquée au problème de gestion intégrée des véhicules d'urgence

Le processus d'optimisation est effectué entre les agents stations. Chacun contrôle un sous-ensemble de véhicules qui représente les variables du problème. Le domaine des variables est composé des appels d'urgence non satisfaits et toutes les stations qui existent.
La fonction objectif du modèle de la section V.1 :

$$\sum_{q_k \in A} \sum_{j \in V(a_k)} \sum_{i \in U} (t_{ji} \times prio(i)) + \sum_{a_k \in A} \sum_{j \in V(a_k)} \sum_{s \in S} t_{js} + penal3 \times \sum_{a_k \in A} \sum_{j \cup (a_k)} dev_j + penal1 \times \sum_{i \in U} unsat_i + penal2 \times \sum_{z \in Z} un\,\mathrm{cov}_z$$

Est factorisée de la façon suivante :

$$F = \sum_{k \in |A|} F_k + F_{n+1}$$

Tel que

$$F_k = \sum_{j \in V(a_k)} \sum_{i \in U} (t_{ij} \times prio(i)) + \sum_{j \in V(a_k)} \sum_{s \in S} t_{js} + penal3 \times \sum_{j \in V(a_k)} dev_j$$

Et

$$F_{n+1} = penal1 \times \sum_{i \in U} unsat_i + penal2 \times \sum_{z \in Z} un\,\mathrm{cov}_z$$

$V(A_k)$: l'ensemble de véhicules de l'agent A_k.
Dans la phase de calcul, un agent A_k optimise F en optimisant F_k et F_{n+1}.

Au lieu d'échanger les affectations des valeurs aux variables (véhicules aux locations) avec toute l'information requise, nous proposons d'échanger uniquement les sommes nécessaires à évaluer la fonction objectif globale. Plus précisément, l'information à envoyer d'un agent à un autre appelée c*onfiguration* contient :

- La valeur de la fonction objectif correspondante à la solution du nœud en cours
- La liste des appels insatisfaits
- Le nombre de véhicules déviés
- La *prepardness* des zones dans la solution courante
- La valeur du gain des déviations faites par les autres agents.

De cette façon, chaque agent peut garder l'information locale confidentielle (l'emplacement des véhicules…) moins de données seront envoyées et seulement le calcul local concernant les véhicules locaux sera effectué pour l'évaluation de la fitness.

Localement, chaque agent garde avec chaque configuration la liste des affectations locales correspondantes à la solution en considération.

D. Résultats expérimentaux

Dans les figures 6.18, 6.19, 6.20, 6.21 nous testons l'effet de l'ordre de priorité sur le PCGD pour les mêmes instances testées pour le SynchLDS. On remarque que contrairement au cas du synchLDS, la différence des séquences des solutions visitées par le processus de recherche est minime ce qui le rend plus adaptable aux problèmes dont la nature n'est pas hiérarchique.

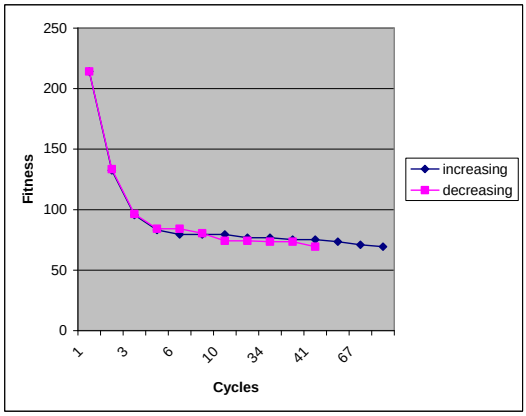

Figure 6.18 PCGD pour instance1

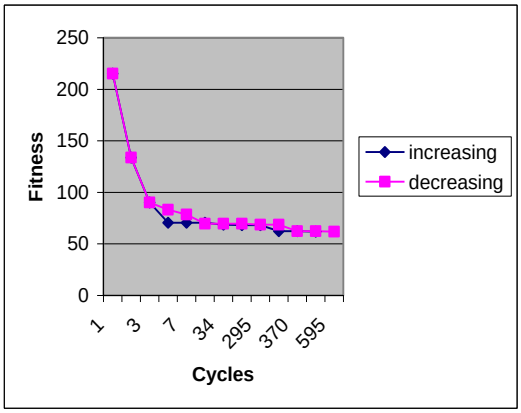

Figure 6.19 PCGD pour instance2

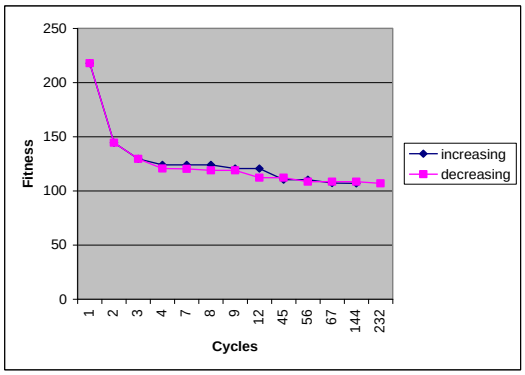

Figure 6.20 PCGD pour instance3

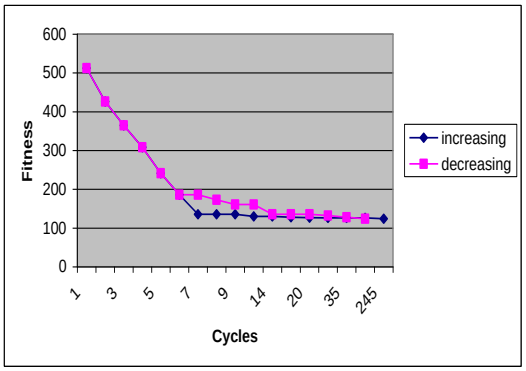

Figure 6.21 PCGD pour instance4

- **Qualité de solution**

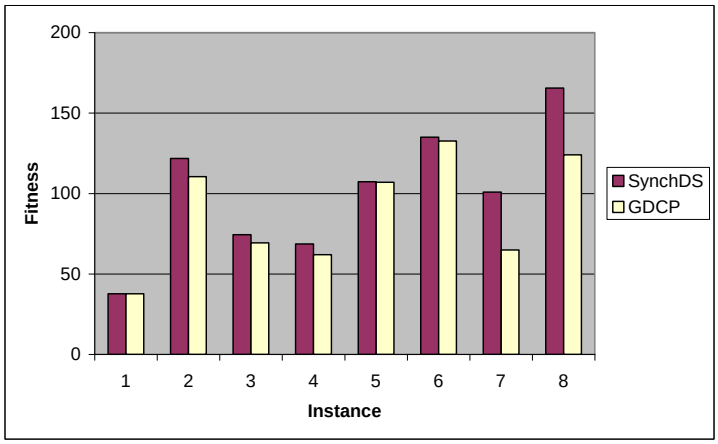

Figure 6.22 Comparaison de fonction objectif.

Dans la figure 6.22 nous comparons la qualité des solutions obtenues par le SynchLDS et le PCGD pour des exécutions complètes. Nous notons que même avec la même procédure d'optimisation locale, le PCGD donne de meilleures solutions que la SynchLDS.

La figure 6.23 montre une comparaison de la meilleure qualité obtenue par les deux algorithmes pour un temps d'exécution maximum de 60 secondes. On note que le PCGD trouve des résultats nettement meilleurs et plus rapidement pour presque toutes les instances testées.

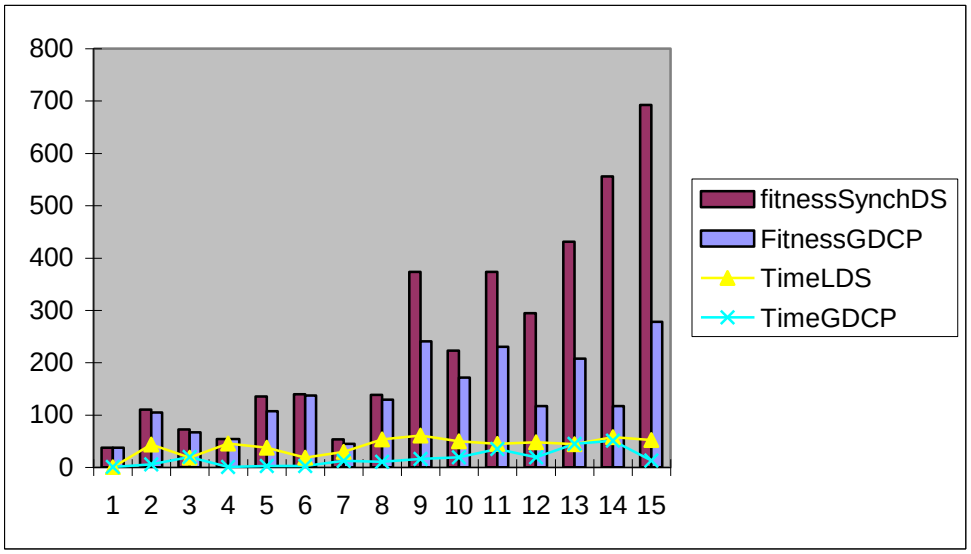

Figure 6.23 La meilleure fonction objectif pour un temps maximum d'exécution de 60 secondes.

- **Le nombre de messages échangés**

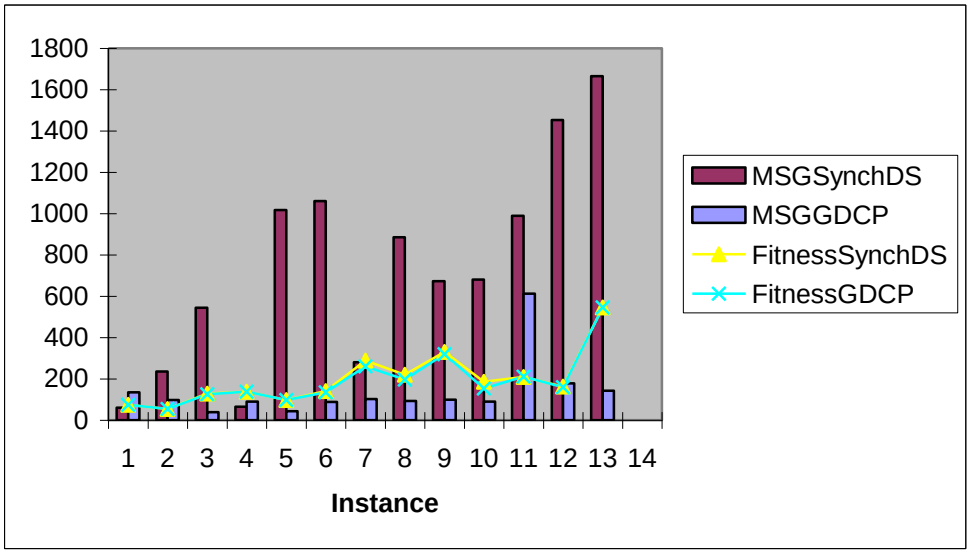

Figure 6.24 Nombre de messages.

Pour évaluer le besoin en communication des deux protocoles, nous comparons le nombre de messages échangés pour atteindre la même (ou presque la même) qualité de solutions. Nous remarquons dans la figure 6.24 que le PCGD atteint des solutions comparables à celles de SynchLDS en échangeant beaucoup moins de messages que SynchLDS.

- **Espace mémoire**

Dans la SynchLDS l'espace mémoire requis pour chaque agent augmente avec la profondeur de l'arbre. Le dernier agent aura $m^{|A|-1}$ nœuds dans sa liste si chaque agent produit en moyenne m solutions [47]. Pour le PCGD, la quantité de mémoire nécessaire est la même pour tous les agents et elle est autant importante que celle requise par le dernier agent. Cependant, vu que le PCGD permet le partage des solutions complètes entre tous les agents, les meilleures solutions sont en général trouvées dans les premiers cycles du processus de recherche permettant ainsi d'atteindre de bonnes performances en utilisant moins d'espace mémoire que la SynchLDS. Ce résultat est illustré dans la figure 6.25 où nous comparons la taille de la liste nécessaire de trouver des solutions de même performance pour les deux protocoles.

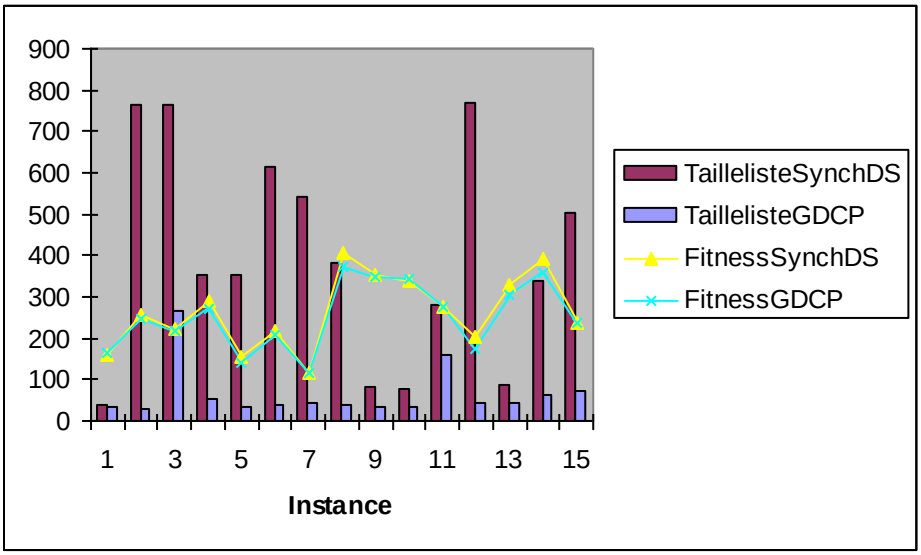

Figure 6.25 Taille de la liste.

Les techniques de contrôle de consommation de mémoire citées dans [47] sont également applicables pour le PCGD puisque la notion de priorité entre les nœuds est aussi utilisée:

1. La première technique repose sur la limite de temps. Après chaque insertion dans la liste, elle consiste à ne garder en mémoire que les nœuds les plus prioritaires dont le temps estimé à leur exploration est inférieur à une certaine limite donnée au temps global du processus de recherche.
2. La deuxième technique consiste à limiter la quantité de mémoire pouvant être allouée. Après chaque insertion dans la liste, on supprime les nœuds les moins prioritaires causant le dépassement de cette limite.

Nous concluons cette partie d'évaluation par une comparaison des solutions centralisées, la méthode des enchères et le protocole PCGD (figure 6.26 & 6.27) ; les résultats correspondants sont résumés dans le tableau 6.2 où on remarque que le PCGD réduit l'écart de 12,91 à 1,13% pour dataset1 et de 12,37% à 3,02% pour dataset2.

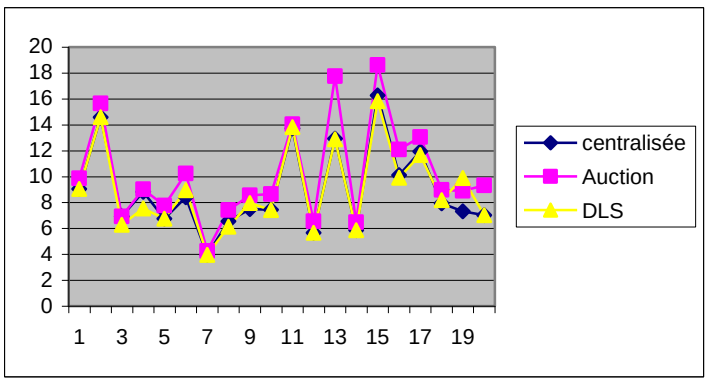

Figure 6.26 Centralisé, enchères et PCGD pour dataset1

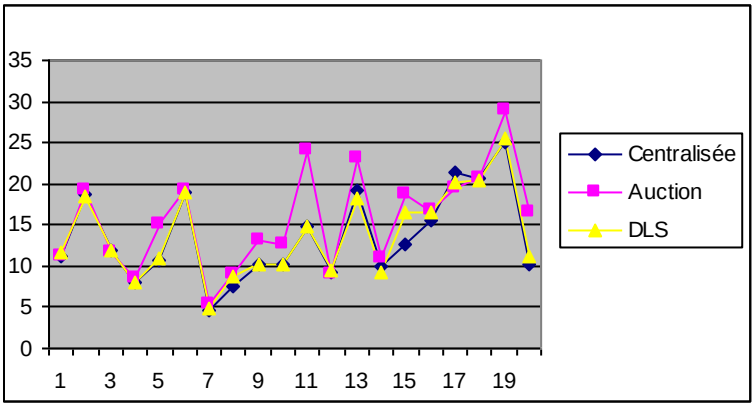

Figure 6.27 Centralisé, enchères et PCGD pour dataset2

	Enchères	PCGD
Dataset1	12,91%	1,13%
Dataset2	12,37%	3,02%

Tableau 6.2 Différence par rapport à la solution centralisée

Cette différence de performance peut, en partie, être expliquée par le fait que dans un problème (comme le notre) qui contient des contraintes globales entre les agents, et une méthode distribuée comme celle des enchères qui ne permet pas une coordination directe inter-agents, l'évaluation de la contrainte globale n'est pas possible car les agents n'ont pas les données nécessaires pour ça. Ceci résulte en une partie de l'espace de recherche non explorée et ainsi les solutions sont moins bonnes. La solution de communication directe proposée permet l'échange des solutions complètes entre les agents (tout en évitant d'attribuer le rôle de coordinateur central à l'un d'eux ce qui pourrait aboutir aux mêmes inconvénients que la solution centralisée) ce qui évite la restriction de l'espace de recherche et améliore la qualité des résultats. Cette hypothèse est vérifiée par la comparaison du nombre de déviations des véhicules appliquées par chacune des approches. Les résultats montrent que pour les instances testées, nous enregistrons deux déviations seulement pour la méthode des enchères contre trente deux déviations pour le PCGD et trente cinq pour la solution centralisée.

VI.3 Conclusion

La coordination distribuée des agents pour un problème d'optimisation distribué peut s'avérer une tâche difficile dans le cas où les décision à prendre sont basées sur l'état global du système. L'application des méthodes de recherche arborescente distribuée générales telle que la SynchLDS est possible, mais nécessite la définition d'ordre de priorité entre les agents. Quand cet ordre n'est pas défini par la nature du problème à résoudre, on remarque souvent un certain effet sur la séquence des solutions visitées au cours de la recherche ce qui influence les résultats obtenus surtout quand le processus est arrêté avant sa terminaison. Les résultats expérimentaux montrent l'efficacité du protocole PCGD que nous avons proposé dans ce chapitre comparé au SynchLDS. Pour les instances testées, le protocole donne également des résultats très proches des solutions centralisées comparées à la méthode des enchères. Ceci est possible à cause de la communication directe entre les agents stations permettant un meilleur contrôle des contraintes du problème et ainsi une meilleure exploration distribuée de l'espace de recherche.

VII. Conclusion et perspectives

L'objet de cette thèse était d'étudier les possibilités qu'offre le domaine de l'intelligence artificielle distribuée dans la résolution des problèmes d'optimisation distribuée. Nous nous sommes intéressés plus particulièrement au cas où les données et les traitements sont distribués et un objectif global doit être optimisé. Cette classe de problèmes est représentée dans ce travail par le problème de gestion intégrée des véhicules d'urgences dans lequel deux objectifs complémentaires sont considérés : Le dispatching qui consiste à choisir le meilleur véhicule à envoyer pour assister les appels d'urgences et la couverture qui sert à choisir le bon emplacement des véhicules d'urgences afin de garantir une bonne préparation de la zone à contrôler pour répondre aux futurs appels. Dans notre travail nous avons considéré une version qui optimise ces deux objectifs conjointement.

L'étude bibliographique que nous avons effectuée dans la première partie de cette thèse et qui a porté sur les trois axes de l'IAD nous a permis de dresser un panorama des principales méthodes et outils qui existent dans le domaine de l'optimisation distribuée coopérative des problèmes. Nous avons également fait un état de l'art sur les problèmes de gestion des véhicules d'urgence avec ses deux volets : dispatching et couverture.

Après la revue de la littérature, nous avons procédé à la modélisation du problème de gestion intégrée des véhicules d'urgence qui consiste à affecter les véhicules d'urgence à des emplacements (des appels d'urgence ou des stations) de façon à optimiser à la fois les temps de déplacements avec le nombre d'appels insatisfaits, ainsi que le nombre de zones non couvertes et le nombre de véhicules déviés de leurs destination. Nous avons utilisé une quantité appelée "*prepardness"* pour mesurer et décider si une zone donnée est bien couverte.

La complexité de ce problème nous a ramené à proposer une méthode de résolution basée sur une hybridation des deux meta-heuristiques : la recherche tabou et l'optimisation par les colonies de fourmis :Antabu. Nous avons ensuite procédé à l'accélération du Antabu en introduisant du parallélisme dans l'évaluation du voisinage. Les résultats expérimentaux montrent que l'approche de parallélisation synchrone permet d'atteindre de meilleures solutions que l'approche asynchrone mais avec des temps d'exécution plus longs.

La résolution du problème régie par le simulateur à événements discrets nous a permis d'étudier l'impact de l'intégration des deux problèmes en un seul modèle sur la qualité de service**.** Les résultats obtenus ont montrés qu'en intégrant les deux problèmes nous marquons un gain important en nombre de zones couvertes ainsi qu'en temps de couverture contre une petite hausse en temps de réponse qui peut, en cas de nécessité, être compensé par les véhicules disponibles par la couverture des zones.
Les solutions centralisées obtenues nous ont également servi de référence pour évaluer les approches de résolution distribuées abordées par la suite.

Nous nous sommes ensuite intéressé à la problématique principale de ce mémoire et qui consiste à chercher des méthodes issues de l'IAD pour la résolution de la version distribuée du problème décrit précédemment. Dans ce but nous avons commencé par proposer une architecture distribuée du système centralisé de gestion des véhicules d'urgence ; celle-ci est un système multi-agents composé de quatre types d'agents : appel, véhicule, zone et station qui coopèrent pour optimiser le problème avec des données et des traitements distribués.
Nous avons abordé ce problème distribué par deux conceptions différentes :

- Dans la première, on considère que les données ainsi que les traitements sont distribués sur les agents et la prise de décision est attribuée à l'agent zone par un mécanisme des enchères conçu pour coordonner les autres agents de façon implicite. Cette méthode a donné de meilleurs résultats comparés au cas où la décision est prise en se basant seulement sur les valeurs de fitness trouvées localement par les stations. Cependant, elle n'apporte pas les résultats escomptés comparée à la centralisée à cause du manque d'interaction directe qui permet le contrôle global des contraintes et ainsi la visite d'un espace plus large de solutions.

- La seconde conception considère que la décision est prise par un processus de recherche distribué basé sur une communication directe entre les stations. Dans ce cas nous proposons un protocole de coordination globale distribuée PCGD qui a l'avantage d'estomper l'effet de l'ordre de priorité des agents et d'accélérer la recherche en échangeant les solutions complètes. Sa comparaison à une adaptation de la méthode LDS synchrone au même problème a montré l'efficacité du protocole proposé en terme de qualité de solutions ainsi qu'en temps d'exécution.

En fin, la comparaison des deux solutions distribuées à la solution centralisée montre une réduction importante de l'écart des solutions du protocole PCGD par rapport à la centralisée ce qui prouve que pour un problème distribué difficile avec des contraintes globales entre les agents, et en utilisant les moyens de communication actuels rapides, il est plus efficace d'opter pour une coordination explicite avec une communication directe entre les agents afin de garantir une bonne qualité de solution.

Les perspectives ouvertes par ce travail de recherche sont liées à la mise en œuvre du modèle et des approches de résolution proposées dans des systèmes réels pour des organisations spécialisées telles que la protection civile ce qui permettra de renforcer et améliorer les différents résultats présentés dans cette thèse.

Afin de pouvoir tirer profit des différentes possibilités de l'IAD en même temps, il est également possible de paralléliser le traitement de la procédure d'optimisation au niveau de chaque agent pour gagner du temps et éventuellement améliorer la qualité des solutions du système distribué. Introduire de l'asynchronisme dans les phases de calcul et de communication du protocole proposé est aussi envisageable afin d'accélérer les temps d'exécution du protocole. Une autre façon de réduire la complexité de la recherche et rendre le protocole proposé plus performant, consiste à inclure une élimination des solutions à explorer en se basant sur une stratégie d'estimation de bornes.

Une amélioration importante pourrait être apportée à l'étude du modèle ainsi qu'à sa résolution en utilisant les techniques de l'optimisation multi-objectifs.

Bibliographie

[1] Andersson.T, Varbrand.P. (2007). Decision support tools for ambulance dispatch and relocation. *Journal of the Operation Research Society*; 58:195-201.

[2] Bachelet.V, Hafidi.Z, Preux.P and Talbi.E-G. (1998). Diversifying tabu search by genetic algorithms. *INFORMS'98 on Operations Research and Management Sciences meeting, Montreal, Canada.*

[3] Ball.M.O, Lin.L.F. (1993). A reliability model applied to emergency service vehicle location. *Operation Research,* (41): 18-36.

[4] Banatre.J.P. (1990). La programmation parallèle. *Edition Eyrolles Octobre 1990*.

[5] Battiti.R, Tecchiolli.G. (1992). Parallel biased search for combinatorial optimization: genetic algorithms and TABU. *Microprocessors and Microsystems,* 16(7): 351-367.

[6] Bertsekas.D.P. (1991). Linear Network optimization. *MIT Press 1991*.

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

[8] Bessiere.C, Maestre.A, Brito.I and Meseguer.P. (2005). Asynchronous backtracking without adding links: a new member in the ABT family. *Artificial Intelligence*, 161(1-2): 7-24.

[9] Bulheimer.B, Kotsis.G and Straub.C. (1997). Parallelization strategies for the ant system. *Technical Report POM 9/97, University of Vienna.*

[10] Casnard.M, Trystram.D. (1993). Algorithmes et architectures parallèles. *Intereditions 1993.*

[11] Chainken.J.M, Larson.R.C. (1971). Methods for allocating urban emergency units. *Operations Research working paper: OR 003-71.*

[12] Chaken.J. (1971). Number of emergency units busy at alarms which require multiple servers. *New York City-Rand Institute, R-531-NYC/ HUD.*

[13] Chakrapani.J and Skorin-Kapov.J. (1993a). Massively parallel tabu search for the quadratic assignment problem. *Annals of Operations Research*. 41:327-341.

[14] Chakrapani.J and Skorin-Kapov.J. (1993b). Connection Machine implementation of a tabu search algorithm for the travelling salesman problem. *Journal of computing and information Technology,* 1(1): 29-36.

[15] Chen.R, Sharman.R, Raghav Rao.H and Upadhyaya.S. (2005). Design principles of coordinated multi-incident emergency response systems. *IEEE international conference on intelligence and security informatics*, 3495:81-98.

[16] Chu.P.C, Beasley.J.E. (1997). A genetic algorithms for the generalized assignment problem. *Operations Research*, 24 (1):17-23.

[17] Church.R, ReVelle.C.S. (1974). The maximal covering location problem. *Paper of the regional science association*, 32:101-118.

[18] Cobham.A. (1954). Priority assignment in waiting line problems. *Operations Research*, 2(1):70-76.

[19] Crainic.T.G, Toulouse.M and Gendreau.M. (1996). Parallel asynchronous tabu search for mulicommodity location-allocation with balancing requirements. *Annals of Operations Research*. 63: 277-299.

[20] Crainic.T.G, Toulouse.M. (2009). Parallel mataheuristics. *Cirrelt 2009-22*. Depôt legal bibliotheque nationale du quebec.

[21] Czech.Z.J. (2000). A parallel genetic algorithm for the set partitioning problem. *In the 8th Euromicro Workshop on parallel and distributed processing*, 343-350.

[22] Daskin.M.S, Stern.E.H (1981). A hierarchical objective set covering model for emergency medical service vehicle deployment. *Transportation Science*, 15:137-152.

[23] Daskin.M.S. (1983). A maximum expected location model : formulation, properties and heuristic solution. *Transportation Science*, 7: 48-70.

[34] Davidsson.R, Henesey.L, Ramstedt.L, Tornquist.J, Wernstedt.F. (2005). Analysis of agent based approaches to transport logistics. *Transportation Research*,13(C):255-271.

[35] De Falco.l, Del Balio.D, Tarantino.E. (1994). An effective parallel heuristic algorithm for the mapping problem. *In proceedings of the 2nd IEEE Australian and new Zealand conference on intelligent information systems*, 160-164.

[36] Desjardins.M.C, Durfee.E.H, Ortiz.C.L, Wolverton.M.J. (1999). A survey of research in distributed, continual planning. *AI Magazine*,1(4):13-22.

[37] Drias.H, Ibri.S. (2003). Parallel ACS for Weighted MAX-SAT. The international Work-Conference on Artificial Neural Networks 2003 (1):414-421.

[38] Durfee.E.H, Lesser.V.R. (1987). Using partial global plans to coordinate distributed problem solvers. International Joint Conferences on Artificial Intelligence, 875-883.

[39] Durfee.E.H, Lesser.V.R, Corkill.D.D. (1989). Trends in cooperative distributed problem solving. *IEEE transactions on knowledge and data engineering*, 1(1): 63-83.

[40] Durfee.E.H and Rosenschein.J.S. (1994) Distributed Problem Solving and Multi-Agent Systems: Comparisons and Examples. In *Proceedings of the Thirteenth International Distributed Artificial Intelligence Workshop*, 94-104.

[41] Durfee.E.H. (2001). Distributed problem solving and planning. *Mutli-agents systems and applications*, 118-149.

[42] Eaton.D.J, Daskin.M.S, Simmons.D, Bulloch.B and Jansma.G. (1985). Determining emergency medical deployment in Austin, Texas. *Interfaces*, 15(1): 96-108.

[43] Farinelli.A, Rogers.A, Pectu.A, Jennings.N.R (2008). Decentralized coordination of lower-power embedded devices using the max-sum algorithm. In 7th International conference on Autonomous agents and Multi-agent Systems, 639-646.

[44] Ferber.J. (1999). Multi-Agent Systems: An Introduction to Distributed Artificial Intelligence. *Addison Wesley, 1999.*

[45] Findler.N.V, Lo.R. (1986) An examination of distributed planning in the world of air traffic control. *Journal of Parallel and Distributed Computing,* 3(3):411-431.

[46] Fisher.M.I, Jaikumar.R, Wassenhove.L.N. (1986). A multiplier adjustment method for the generalized assignment problems. *Management Science,* (32): 1095-1103.

[47] Gaudreault.J. (2009). Algorithmes pour la prise de décision distribuée en contexte hiérarchique. Thèse de doctorat, Ecole polytechnique de Montréal.

[48] Gaudreault.J, Frayret.J.M, Pesant.G. (2009) Distributed search for supply chain coordination. *Computers in industry: collaborative engineering for concurrent engineering to enterprise collaboration*. 60(6):441-451.

[49] Garcia.B.L, Potvin.J.Y and Rousseau.J.M (1994). A parallel implementation of the tabu search heuristic for the vehicle routing problems with time window constraints. Computer and Operations Research, 21(9): 1025-1033.

[50] Garcia-Martinez.C, Lozano.M. (2008) Local search based on genetic algorithms. Advances in metaheuristics for hard optimization, natural computing, 199-221.

[51] Gendreau.M, Laporte.G, Semet.F. (1997). Solving an ambulance location model by tabu search. *Location Science,* 5:75-88.

[52] Gendreau.M, Laporte.G, Semet.F. (2001) A dynamic model and paralle Tabu search heuristic for real time ambulance relocation. *Parallel Computing,* 27:1641-1653.

[53] Gleason.J.M. (1975). A Set Covering Approach to Bus Stop Location. *Omega* 3: 605–608.

[54] Glover.F, Laguna.M. (1997). Tabu Search. *Kluwer Academic Publishers 1997.*

[55] Goldberg.J, Dietrich.R, Chen.J.M, Mitwasi.M.G. (1990). Validating and applying a lodel for locating emergency medical services in Tuscon, AZ. *European Journal of Operational Research,* 49:308-324.

[56] Haghani.A, Hu.H and Tian.Q. (2003). An Optimization Model for Real-Time Emergency Vehicle Dispatching and Routing. *TRB Annual Meeting, Transportation*

Research Board, Washington D.C.

[57] Haghani.A, Yang.S (2007). Real time emergency response fleet deployment concepts, systems, simulation and case studies. *Dynamic fleet management*, 7:133-162.

[58] He.J, Jingyuan.Z, Jifeng.X, Zhilei.R, Yan.H. (2010). A hybrid ACO algorithm for the next release problem. *2nd International Conference on Software Engineering and Data mining*, 161-171.

[59] Hogan.K, Revelle.C.S. (1986). Concepts and applications of backup coverage. *Management science* 34:1434-1444.

[60] Ibri.S, Drias.H., Nourelfath.M. (2009). A parallel hybrid ant-tabu algorithm for integrated emergency vehicle dispatching and covering problem. *International Journal of Innovative Computing and Applications*. 2 (4), 226–236.

[61] Ibri.S, Drias.H, Nourelfath.M. (2010a). Integrated emergency vehicle dispatching and covering: A parallel Ant-tabu approach. *8th International Conference of Modeling and Simulation, MOSIM'10*. 1039-1045.

[62] Ibri.S, Nourelfath.M, Drias.H. (2010b). On the integration of dispatching and covering for emergency vehicles management system. *International Conference on Machine and Web Intelligence, IEEE ICMWI'2010*. 198-204. Accepted for publication in International Journal of Advanced Operations Management (IJAOM).

[63] Jennings.N.R, Sycara.K, Wooldridge.M. (1998). A roadmap of agent research and developpement. *Autonomous Agent and Multiagent Systems*, 1(1):7-38.

[64] Kolesar.P, Walker.W.E. (1974). An algorithm for the dynamic relocation of fire companies. *Operations Research*, 22:249-274.

[65] Lee.S.Y, Lee.K.G. (1996). Synchronous and asynchronous parallel simulated annealing with multiple Markov chains. *Transactions on Parallel and Distributed Systems*, 7(10): 993-1007.

[66] Lopez.B, Innocenti.B, Aciar.S, Cuevas.I. (2005). A multi-agent system to support ambulance coordination in time-critical patient treatment. *In Proceedings ASAI 2005, 7th Argentine Symposium on Artificial Intelligence*, 43-54, Rosario, Argentina.

[67] Lopez.B, Innocenti.B, Busquets.D. (2008). A multiagent system for coordinating ambulances for emergency medical services. *IEEE Inteligent Systems.* 1541-1672.

[68] Magdelaine.C. (2009). Augmentation dramatique des catastrophes naturelles dans le monde. *document internet : http://www.notre-planete.info*

[69] Mailler.R, Lesser.V. (2004). Solving distributed constraint optimization problems using cooperative mediation. *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, 1: 438-445.

[70] Maxwell.M.S, Restrepo.M, Henderson.S.G, Topaloglu.H. (2010). Approximate

dynamic programming for ambulance redeployment. Informs Journal on Computing, 22:266-281.

[71] Middendorf.M, Reishle.F, Schmeck.H. (2000). Information exchange in multi-colony ant algorithms.In: Parallel and Distributed Computing. Proceedings of the 15th IPDPS 2000 Workshops on Parallel and Distributed Processing, 645-652.

[72] Miki.M, Hiroyasu.T, Wako.J and Yoshida.T. (2003). Adaptative temperature schedule determined by genetic algorithm for parallel simulated annealing. *In CEC'03 the 2003 Congress on evolutionary computation*, 1:459-466.

[73] Modi.P.J. (2003). Distributed constraint optimization for multi-agent systems. Thèse de doctorat. *University of California 2003.*

[74] Modi.P.J, Shen.W.M., Tambe.M. and Yokoo. M. (2003). An asynchronous complete method for distributed constraint optimization. *Proceedings of the 2nd International Joint Conference on Autonomous Agents and Multi-Agent Systems* 2003, 161 - 168.

[75] ModiP.J, Shen.W.M, Tambe.M and Yokoo.M. (2005). Adopt: asynchronous distributed constraint optimization with quality guarantees. *Artificial Intelligence*, 161(1-2):149-180.

[76] Nauss.R.M. (2003). Solving the Generalized Assignment Problem: An Optimizing and Heuristic Approach. *Informs Journal on Computing*, 15(3): 249-266.

[77] Pardalos.P.M, Pitsoulis.L and Resend.M.G.C. (1995). A parallel GRASP implementation for the quadratic assignment problem. *Parallel Algorithms for Irregularly Structured Problems.* Irregular'94:115-130.

[78] Pirckul.H, Schilling.D. (1989). The capacitated maximal covering problem with backup service. Annals of Operations Research, 18:141-154.

[79] Porto.S.C.S, Kitajima.J.P.F.W and Ribeiro.C.C (2000). Performance evaluation of a parallel tabu search task scheduling Algorithm. *Parallel Computing*, 26:73-90.

[80] Raidl.G.R. (2006). A unified view on hybrid metaheuristics. *Third International Workshop, HM 2006*, LNCS 4030:1-12.

[81] Rego.C, Roicairol.C. (1996). A parallel Tabu search algorithm using ejection chains for the VRP. In I.H. Osman and J.P. Kelly (Eds.), *Meta-Heuristics: Theory and Applications*, Kluwer Academic Publishers, pp. 661–675.

[82] Repede.J.F, Bernardo.J.J. (1994) Developping and valdating a decision support system for locating emergency medical vehicles in Louisville, Kentucky. *European Journal of Operations Research*, 75:567-581.

[83] Revelle.C.S, Hogan.K, 1989 the maximum availability location problem. *Transportation Science*, 23:192-200.

[84] Ross.G.T, Soland.M.S. (1975). A branch and bound algorithm for the generalized

assignment problem. *Mathematical programming,* 8:91-103.

[85] Sarkar.V. (1989). Partitioning and scheduling parallel programs for multi processing. *MITPress 1989.*

[86] Schilling.D.A, Elzinga.D.J, Cohon J., Church R.L, ReVelle.C.S (1979). The TEAM/FLEET models for simultaneous facility and equipment sitting. *Transportation Science.* 13:163-175.

[87] Shoam.Y, Leyton-Brown.K (2009). Multiagent systems :Algorithmic, game-theoretic and logical foundations. *Cambridge University Press 2009.*

[88] Smith.R.G (1980). The contract net protocol : High level communication and control in distributed problem solver. *IEEE Trans on computers*, 29(12):1104-1113.

[89] Stevenson.K. (1971). Operational aspects of emergency Ambulance services. *MIT Operations Research center,* technical report N°61.

[90] Stutzle.T. (1998). Parallelization strategies for ant colony optimization. *PPSN V Proceedings of the 5th International Conference on Parallel Problem Solving from Nature*. 722 – 731.

[91] Su.B, Shi.C, Wang.K, Hu.P, Wang.J, Wu.Y (1990). Distributed problem solving system for transport dispatching. *Proceeding of the 3rd International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert System*.141-150.

[92] Sycara.K, Roth.S, Sadeh.N, Fox.M. (1991). Distributed constrained heuristic search. *IEEE transactions on systems, man and cybernetics*, 21(6):1446-1461.

[93] Taillard.E (1991). Robust Tabu search for the quadratic assignment problem. *Parallel computing.* 17: 443-455.

[94] Taillard E.D. (1994). Parallel tabu search techniques for the job shop scheduling problem. *Journal on Computing*, 6(2):108-117.

[95] Talbi.E.G. (2002). Taxonomy of hybrid metaheuristics. *Journal of heuristics*, 8(5):541-564.

[96] Toregas.C.R, Swain.R, ReVelle.C.S, Bergman.L (1971). The location of emergency service facilities. *Operations Research,* 19:1363-1373.

[97] Weihmayer.R, Brandau.R. (1990). Cooperative distributed problem solving for communication network management. *Computer Communications: Network Management,* 13(9):547-557.

[98] White.T, Bieszczad.A, Pagurek.B. (1998). Distributed fault location in networks using mobile agents. *Intelligent Agents for Telecommunication Applications,*130-141.

[99] Yang.S. (2006). Integrated management of emergency vehicle fleet. Thèse de doctorat, university of Maryland.

[100] Yokoo.M, Durfee.E.H, Ishida.T, , Kuwabara.K. (1992). Distributed constraint satisfaction for formalizing distributed problem solving. *IEEE International Conference on Distributed Computing Systems,* 614-621.

[101] Yokoo.M, Durfee.E.H, Ishida.T, , Kuwabara.K. (1998). Distributed constraint satisfaction problem: formalization and algorithms. *IEEE transaction on knowledge and data engineering,* 10: 673-685.

[102] Yokoo.M, Hirayama.K. (2005). The distributed breakout algorithms. *Artificial Intelligence : distributed constraint satisfaction,* 161(1-2):89-115,

[103] Zhang.W, Wang.G, Xing.Z, Wittenburg.L. (2005). Distributed stochastic algorithm and distributed breakout algorithm: properties, comparison and applications to constraint optimization problems in sensor networks. *Artificial intelligence* 161:55-87.

Récapitulatif de la production scientifique

Dans le cadre de cette thèse, nous avons écrit six articles dont trois dans des journaux internationaux et trois dans des conférences internationales.
Nous présentons dans ce qui suit les références des différentes publications par ordre chronologique.

Articles publiés dans des journaux internationaux

Ibri S., Drias H., Nour El Fath M, (2010) A hybrid parallel Ant-tabu algorithm for integrated emergency vehicle dispatching and covering problem, *International Journal of Innovative Computing and Applications*, 2(4), 226-236, 2010

Ibri S., Nour El Fath M., Drias H. (2011) A multi-agent approach for the integrated emergency vehicle allocation and covering problem. *Engineering Applications of Artificial Intelligence.* doi: 10.1016/j.engappai.2011.10.003.

Ibri S., Nour El Fath M., Drias H. (2012) A contribution for integrating dispatching and covering in the emergency vehicles management services. *International Journal of Advanced Operations Management, 2012.*

Actes de conférence

Ibri S., Nour El Fath M., Drias H Ant-Tabu Heuristic for Real-Time Emergency Vehicle Dispatching. I*nternational Conference on Industrial Engineering and Systems Management,* Montreal, Canada, May 13-15 2009.

Ibri S., Drias H., Nour El Fath Mustapha, Integrated emergency vehicle dispatching and covering: A parallel ant-tabu approach, *MOSIM'10 - 8e Conférence Internationale de Modélisation et Simulation*, Hammamet, Tunisie, 10-12 mai, 2010

Ibri S., Nour El Fath M., Drias H, On the integration of dispatching and covering for emergency vehicles management systems, emergency vehicles management system. International Conference on Machine and Web Intelligence, IEEE ICMWI'2010. 198-204.

Annexe: L'environnement JADE

L'environnement de développement d'agents en Java : *Java Agent Development Environment 'JADE'* permet de créer des agents, de les faire exécuter des tâches ainsi que de les faire communiquer, le tous en utilisant le langage java.
En plus de ces fonctions de base, jade fournit des caractéristiques avancées, telles que le support des protocoles d'interactions complexes et l'utilisation des otologies définies par l'utilisateur.

Jade est composé de :

- Un environnement d'exécution où les agents *'vivent'*
- Une librairie de classe que le programmeur utilise pour le développement de ses agents.
- Un ensemble d'outils graphique pour l'administration des activités des agents.

Les plateformes et les containers :

Un container est une instance d'environnement d'exécution jade. Il peut contenir plusieurs agents. Une plateforme est l'ensemble de tous les containers actifs.
Chaque plateforme a un container principal qui est un container qui est toujours actif et sert à enregistrer les nouveaux containers.
Un container principale contient aussi deux agents spéciaux : l'AMS et le DF.
L'AMS est l'agent de gestion du system : '*Agent Management System'* son rôle est de gérer et de contrôler tous les agents du system.
Le DF : '*Directory Facilitator'* fournit le service des pages jaunes qui permet à un agent de trouver un autre agent avec les services qu'il cherche.

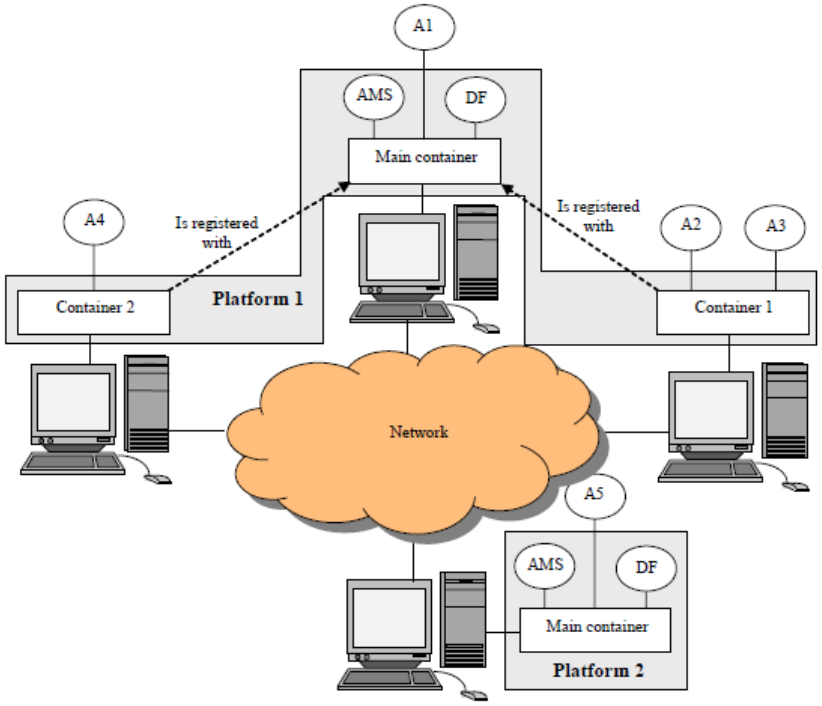

Figure1. Plateformes et Containers

1. Creation d'agent

N'importe quelle classe permettant creer un agent Jade, doit etendre la classe `jade.core.Agent` et implementer la methode `setup()` correspondante comme le code suivant le montre.

```
import jade.core.Agent;
public class Vehicule extends Agent {
protected void setup() {
// Printout a welcome message
System.out.println("L'agent vehicule "+getAID().getName()+" est prêt.");
}
}
```

La methode `setup()` est sensée initialiser l'agent. Le travail à effectuer par l'agent est inclut dans ses `"behaviours"` .

Chaque agent est identifié par un identificateur, une instance de la classe `jade.core.AID`. La methode `getAID`() retourne l'identificateur de l'agent.

Pour mettre fin à l'execution d'un agent, la methode `doDelete()` doit etre appelée. L'appel de la methode `takeDown()` juste avant la terminaison de l'agent permet d'effectuer les operations clean-up.

2. Communication entre agents

Le paradigme de communication adopté par JADE est '*le passage asynchrone de messages : asynchronous message passing'*. Chaque agent a une sorte de boite aux lettres (la queue de messages de l'agent) où les messages envoyés par les autres agents lui sont postés. L'agent est notifié quand un nouveau message est posté dans la queue.

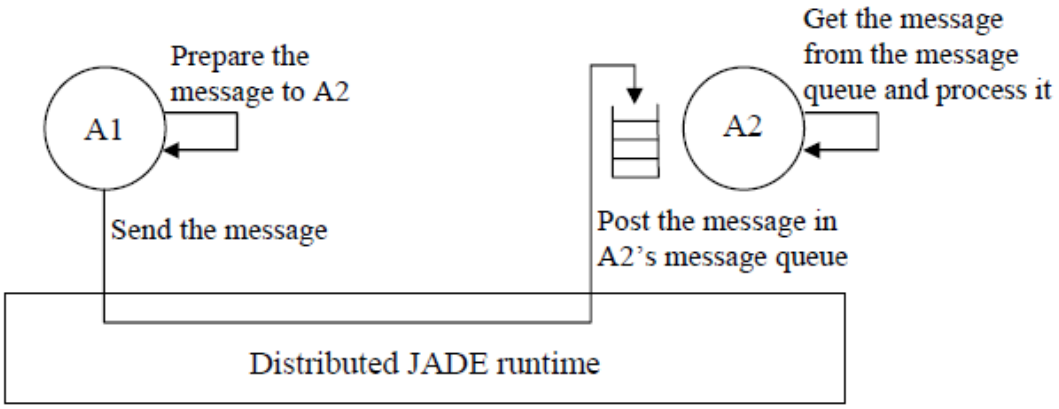

Figure 2. Le paradigme de passage de message asynchrone

Le langage de communication entre agents utilise par les agents de JADE a le format spécifié par FIPA (la fondation des agents physiques intelligents). Ce format comprend un nombre de champs dont les plus importants:

• L'expediteur du message

• La liste des destinataires

• L'intention de communication (***performative***) pour indiquer ce que l'expediteur veut faire par l'envoie du message : REQUEST, INFORM, PROPOSE, ACCEPT_PROPOSAL, REJECT_PROPOSAL…etc

• Le contenu du message

• Le langage du contenu pour déterminer la syntaxe qui sert à comprendre le message

• L'ontologie utilisée.

L'envoie d'un message à un autre agent consiste à remplir les champs d'un objet `ACLMessage`, ensuite appeler la methode `send()` de la classe `Agent`.

Un agent peut recuperer les messages de sa queue de messages par la methode `receive()`. Celle-ci retourne le premier message dans la queue en le supprimant, ou bien un `null` si la queue est vide.

3. Les conversations complexes

Les tâches exécutées par chaque agents sont définies par ses behaviours (comportements) qui sont des instances de la classe `jade.core.behaviours`.

Une conversation est une séquence de messages échangés par deux agents ou plus avec des relations causales et temporelles bien définies. Le `RequestPerformer` behaviour par exemple doit envoyer un message `CFP` (Call For Proposal: un appel d'offre) à plusieurs agents, reçoit les reponse en retour et s'il a au moins une proposition `PROPOSE`, il envoie un message `ACCEPT_PROPOSAL` et reçoit la réponse `/ REJECT_PROPOSAL`.
Les conversations complexes sont effectuées typiquement suivant un protocole d'interaction bien défini. JADE fournit un riche support pour implémenter les conversations suivant les protocoles (tel que le contract net) dans le package `jade.proto`.