Stratégies d'échange d'informations dans un système de calcul distribué pour l'optimisation des problèmes combinatoires

Kamel Belkhelladi

Année : 2010
N° d'ordre : 1028

UNIVERSITÉ D'ANGERS

ÉCOLE DOCTORALE STIM
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE MATHÉMATIQUES

T H È S E

pour obtenir le titre de

Docteur en Sciences

de l'Université d'Angers
Mention : INFORMATIQUE

Présentée et soutenue par

Kamel BELKHELLADI

STRATÉGIES D'ÉCHANGE D'INFORMATIONS DANS UN SYSTÈME DE CALCUL DISTRIBUÉ POUR L'OPTIMISATION DES PROBLÈMES COMBINATOIRES

Soutenue publiquement le 15 Février 2010

Devant le jury ci-dessous :

Rapporteurs :	Van-Dat CUNG	-	Professeur, INP (Grenoble)
	Alain BUI	-	Professeur, PRiSM (Versailles)
Directeur :	Pierre CHAUVET	-	Dr-Hdr, LISA-IMA (Angers)
Co-Directeur :	Arnaud SCHAAL	-	Enseignant-Chercheur, ISAIP (Dijon)
Examinateurs :	Atulya K. NAGAR	-	Professeur, FBCS (Liverpool, UK)
	Flavien BALBO	-	Maître de conférences, LAMSADE (Paris 9)
	Xavier GANDIBLEUX	-	Professeur, LINA (Nantes)

ED 503

Remerciements

La réalisation d'un doctorat est une tranche de vie à part entière qui laisse paraître un profond égoïsme en fin de parcours mais qui s'est nourrie, tout du long, de nombreuses et diverses influences. Beaucoup de personnes participent directement ou indirectement à sa concrétisation et méritent d'être remerciées quel qu'en soit leur degré d'investissement. Malheureusement, ici tout le monde ne peut apparaître nommément et je me dois de faire une sélection subjective.

D'un point de vue professionnel, je commencerai par remercier mon directeur de thèse Dr-Hdr. Pierre Chauvet et mon co-encadrant Dr. Arnaud Schaal pour le travail qu'ils ont effectué à mes côtés et qui était emprunt tout autant de conseils pour orienter mes recherches que de critiques nécessaires à la pertinence de mon étude.

Ensuite, je souhaite remercier mes rapporteurs Pr. Van-Dat Cung et Pr. Alain Bui pour avoir accepté d'évaluer mon travail et pour les commentaires qu'ils ont formulés afin de le perfectionner. Je tiens aussi à exprimer ma gratitude envers Abderrahim Oulidi et Régis Huez qui ont eu la gentillesse de lire l'intégralité de mon manuscrit et qui ont usé de leurs expertises pour déceler certains aspects qui m'avaient échappés.

Mes remerciements s'adressent aux chercheurs et aux membres du personnel de l'Institut de Mathématiques Appliquées (IMA) pour leur accueil et leur aide.

Je n'oublierai pas non plus les différents doctorants de l'IMA qui m'ont permis de faire régulièrement une synthèse de mon sujet. Je citerais en particulier ceux qui sont devenus de véritables amis : Olivier et Mamoun qui m'ont fait passer d'excellents instants aux moments où la rédaction se faisait des plus pesante.

D'un côté plus personnel, je ne saurais montrer trop de reconnaissance aux membres de ma famille. À mes parents dans un premier lieu, Boumediène et Bakhta, qui ont toujours su m'accorder leur confiance et leur soutien dans les choix parfois difficiles que j'ai pu faire durant mes études. À mes frères et sœurs ensuite, Abdelaziz, Mourad, Djamila, Hassiba, Zahia et Kamila, qui ont été capables de me montrer le pragmatisme qui parfois manque dans les domaines scientifiques. Enfin, à mes belles sœurs, Fatiha et Batoul et à mes neveux et nièces, Islam, Djawad, Wafaa,..., qui au travers de leur fraîche arrivée savent apporter une touche d'innocence que j'apprécie tant. Famille, voyez dans cet ouvrage le témoignage des profonds sentiments que je vous porte.

Table des matières

Chapitre 1

Introduction générale

1.1 Contexte de l'étude

Dans sa vie, l'homme est confronté quotidiennement à des problèmes d'optimisation plus ou moins complexes. Cela commence au moment où il se rend à son bureau, le range et aller jusqu'à construire un processus industriel, pour la planification des différentes tâches de production par exemple. Ces problèmes peuvent souvent s'exprimer sous la forme d'un « problème d'optimisation ». Les problèmes d'optimisation combinatoire (POCs) sont complexes et difficiles (Garey et Johnson (1979)). À titre d'exemple, chercher le chemin le plus court ou le plus rapide entre deux points est un problème assez simple à résoudre, mais si l'on désire faire plusieurs étapes (profiter du trajet pour chercher ses enfants à l'école, passer à la poste, acheter du pain, etc.), le problème se complique très vite. Quand le nombre d'étapes croît, le nombre de possibilités augmente au-delà de ce que l'on peut énumérer, et même au-delà de ce que peuvent calculer les ordinateurs les plus puissants : c'est l'« explosion combinatoire ». En outre, les problèmes d'optimisation issus de l'industrie sont caractérisés par une évolution continuelle de leur modélisation en termes de contraintes et objectifs et leur résolution nécessite des ressources matérielles importantes (puissance CPU, capacité mémoire, etc.).

On distingue deux grandes classes de méthodes de résolution des POCs : les méthodes exactes et les méthodes approchées ou heuristiques. Les méthodes exactes permettent de trouver des solutions optimales. Cependant, elles deviennent très vite inutilisables pour des instances de taille importante. Par contre, les heuristiques tentent de trouver en un temps raisonnable de bonnes solutions sans garantie d'optimalité mais acceptables au regards des contraintes des POCs et des délais souvent imposés pour leur résolution. Elles regroupent les heuristiques spécifiques à un POC particulier et les métaheuristiques. Les premières sont plus pointues, en général plus « efficaces » mais peu réutilisables (les méthodes constructives, gloutonnes, etc.). Les métaheuristiques sont générales et indépendantes des POCs traités. Elles sont classées en deux catégories : les métaheuristiques à population de solutions (algorithmes évolutionnaires, colonies de fourmis,...) favorisant une diversification de la recherche ; les métaheuristiques à solution unique (recuit simulé, recherche tabou,...) manipulant une solution à la fois et caractérisées par une exploitation de la région englobant la solution initiale dans l'espace de recherche du problème traité.

De nombreuses études réalisées au cours des dernières années (Blum et Roli (2003), Birattari (2005), Le Bouthillier et Crainic (2005), Wang et Tang (2009), etc.) ont démontré l'utilité et l'efficacité des métaheuristiques pour la résolution des POCs. Les métaheuristiques permettent d'améliorer à la fois la robustesse et la qualité des solutions trouvées. Cette amélioration réside dans un bon équilibre entre l'exploration de l'espace de recherche et l'exploitation prometteuse. Il demeure toutefois que ces algorithmes demandent un temps de calcul et une quantité de mémoire considérables qui sont étroitement liés à la taille du problème et à l'ob-

tention d'une certaine qualité de solution. De ce fait, ces algorithmes deviennent intéressants à paralléliser. Les algorithmes évolutionnaires (AEs) simulent le processus de l'évolution naturelle pour trouver une meilleure solution à travers la sélection et la re-création sur plusieurs générations. L'utilisation de ces derniers est devenue de plus en plus populaire pour la résolution de problèmes d'optimisation dans différents domaines. De manière générale, le calcul évolutionnaire est une approche de recherche dans laquelle des opérateurs génétiques, tels que la reproduction, le croisement et la mutation, sont utilisés pour atteindre une solution satisfaisante dans un espace dont la dimension est déterminée par la longueur des chromosomes (solutions). Par exemple, dans un schéma où les chromosomes sont codés en binaire, la longueur n d'un chromosome indique que les techniques évolutionnaires sont censées trouver la solution appropriée dans un espace de 2^n candidats. Ainsi, l'augmentation de la longueur du chromosome entraîne une augmentation dans la complexité de la tâche, l'espace de solutions croîtra de manière exponentielle et rend la recherche de plus en plus difficile.

Afin de résoudre des problèmes plus difficiles par les AEs, deux de leurs caractéristiques intrinsèques doivent être améliorées. (1) La *convergence prématurée* : l'une des caractéristiques attractives d'un algorithme évolutionnaire est qu'il peut rapidement concentrer sa recherche dans des régions prometteuses de l'espace des solutions. Mais cette caractéristique peut diminuer la diversité de la population avant que l'objectif soit atteint. En d'autres termes, l'algorithme évolutionnaire converge vers des optima locaux. Bien que l'opérateur de mutation puisse maintenir la diversité de la population, il détruit des individus. Avec un taux de mutation élevé, un AE peut accroître la diversité, mais la meilleure solution peut cependant être écartée (égarée). Ainsi, une méthode pouvant maintenir la diversité de la population pour explorer de nouvelles régions de l'espace de solutions sans détruire les solutions courantes est nécessaire. (2) Le *temps de calcul* : Comme nous le savons, un AE est une approche à population, il doit évaluer tous les membres de la population et peut ensuite sélectionner les plus adaptés pour survivre. Fondamentalement, la taille de la population doit être suffisamment grande pour permettre à un AE d'effectuer une recherche globale dans l'espace de solutions. De façon générale, la taille de la population s'accroît par rapport à la difficulté du problème à résoudre. En conséquence, un temps de calcul excessif peut être nécessaire pour effectuer toutes les évaluations pour un problème de grande taille.

Quelques travaux récents (Eksioglu *et al.* (2001), Cung *et al.* (2002), Crainic et Toulouse (2003), Cahon *et al.* (2004), Crainic *et al.* (2005), Talbi et Bachelet (2006), Alba *et al.* (2007) et Belkhelladi *et al.* (2007)) ont identifié et abordé le calcul parallèle comme étant une approche très prometteuse pour l'amélioration de la performance des métaheuristiques. Cependant, les comportements et les effets des mécanismes parallèles dans ce cadre sont encore peu étudiés.

Le calcul distribué offre une approche naturelle pour la résolution des problèmes de calcul intensif issus de l'industrie et du contrôle. De nombreuses approches de calcul distribué ont été développées au cours des dernières décennies. Il s'agit notamment de la programmation par Sockets, le modèle client-serveur(RPC[1]), les environnements de calcul distribué orienté-objet, CORBA, Java RMI, etc. Ces approches ont chacune des avantages et des inconvénients (Tanenbaum et Steen (2001)). Les limites majeures à l'égard de la conception robuste des applications réseau impliquant de nombreuses entités autonomes sont les suivantes : (1). Les interactions entre les entités participantes sont fixées a priori par le biais des instructions

[1]Remote Procedure Call

codées explicitement par le concepteur d'applications. En conséquence, elles n'ont pas de comportement adaptatif. (2). Une interaction implique une communication, ce qui le rend impropre pour les applications qui doivent fonctionner dans des environnements où le maintien de communication continue est coûteux ou quasi-impossible et les connexions ne sont pas fiables. Ces considérations ont motivé le développement d'approches de calcul distribué basée sur les agents qui fournissent les moyens de maintenir une interaction permanente sans communication permanente (White (1997)). Les systems multi-agents offrent un modèle très efficace pour la conception des applications distribuées en réseau qui impliquent un grand nombre d'entités autonomes (Honavar *et al.* (1998)).

Les systèmes multi-agents (Ferber (1995), Ferber (1997)) forment une partie de l'intelligence artificielle distribuée (DAI[2]). L'approche multi-agents propose un cadre méthodologique bien adapté pour la modélisation et l'analyse des systèmes complexes. Elle considère les systèmes comme des sociétés composées d'entités autonomes et indépendantes, appelées agents, qui interagissent en vue de résoudre un problème ou de réaliser collectivement une tâche. Elle a été utilisée de façon fructueuse dans de nombreux domaines et en particulier : la robotique, la résolution distribuée de problèmes, la modélisation et la simulation des systèmes complexes. Cette approche semble intéressante dans le cadre des métaheuristiques parallèles, tout d'abord pour aborder la distribution et l'adaptation, mais aussi pour apporter des outils conceptuels et méthodologiques permettant de les analyser et de les modéliser (Meignan (2008)).

Ainsi, l'évolution rapide des capacités de traitement des stations de travail, et l'amélioration de la qualité du réseau reliant ces stations, ont motivé l'utilisation du réseau comme support pour les calculs distribués. Une bonne utilisation de cette plate-forme permet ainsi d'augmenter la puissance potentielle des ordinateurs. La recherche d'un maximum de puissance dans les systèmes distribués nécessite une répartition et un équilibrage de charge efficaces. Ceci est d'autant plus vrai lorsque l'on se trouve dans un contexte hétérogène. Travailler avec un grand nombre de machines perd de l'intérêt si le système est mal équilibré, car une partie de la puissance totale disponible est inutilisée. La bonne répartition d'une application nécessite des connaissances préalables sur son évolution permettant une modélisation assez fine de l'application. Dans la plupart des cas, cette connaissance est très coûteuse voire impossible. Nous adoptons une approche itérative où l'équilibrage de charge se fait de proche en proche, les machines les moins chargées « aidant » les machines trop chargées. Nous proposons une architecture pour l'équilibrage de charge dynamique basée sur la technologie des « agents mobiles ». Dans cette architecture, la prise de décision s'effectue en fonction de la charge locale de la machine ce qui permet de réduire les échanges d'informations. L'algorithme de prise de décision utilise deux seuils dynamiques et lorsque les seuils sont dépassés un appel d'offre est lancé afin de faire la migration des tâches. Dans le but d'avoir une vision globale sur le système, nous proposons un agent de statut global qui va explorer le réseau afin de construire une vision sur la charge moyenne. Cette charge moyenne sera transportée aux agents de décision afin d'ajuster les seuils (El-Falou (2006)).

Tout comme pour la thèse de Meignan (2008), nos travaux se situent donc à l'intersection des domaines de l'optimisation combinatoire et des systèmes multi-agents. Ils s'inscrivent dans la continuité des propositions de modèles ou de plates-formes pour les métaheuristiques

[2]Distributed Artificial Intelligence

parallèles. Cependant, en plus de la proposition du cadre de modélisation commun aux différentes métaheuristiques parallèles à base de population, l'accent est surtout mis sur la potentialité de mise en œuvre distribuée des approches et sur l'utilisation de techniques de l'adaptation dynamique des entités de recherche.

Dans la suite de ce chapitre, nous avons opté pour le même style de présentation utilisé par Meignan (2008) pour définir les objectifs de nos travaux, décrire nos différentes plates-formes de calcul parallèle et structurer le plan général de la thèse.

1.2 Objectifs de la thèse

Les objectifs de cette thèse consistent essentiellement en la :

- Proposition d'une plate-forme appelée MAF-DISTA[a], dédiée à la conception flexible et réutilisable de métaheuristiques parallèles à base de population mono et multi-objectifs ; en s'appuyant ainsi sur une approche orientée agent mobile.
- Détermination d'un ensemble de stratégies d'échange d'informations entre les agents participant à la résolution du même problème d'optimisation.

[a] Mobile Agent Framework Distributed population based Algorithms

Pour atteindre ces objectifs, nous adoptons la démarche suivante. Tout d'abord, nous développons une métaheuristique parallèle fondée sur un modèle client/serveur afin de valider notre première stratégie d'échange. Cet outil servira par la suite pour faire des comparaisons. Pour appliquer la démarche de conception des systèmes d'agents mobiles aux métaheuristiques parallèles, nous définissons un cadre commun de modélisation de ces dernières en proposant une plate-forme fondée sur une approche organisationnelle. Ensuite, pour valider cet outil et transférer les concepts qui nous semblent pertinents des systèmes d'agents aux métaheuristiques parallèles, nous proposons une métaheuristique parallèle utilisant la métaphore de la coalition. Enfin, nous mettons en œuvre cette métaheuristique parallèle pour traiter deux problèmes d'optimisation combinatoire afin d'évaluer expérimentalement l'approche que nous proposons.

1.2.1 Un algorithme génétique parallèle pour le CARP(CSM-PGA)

Pour vérifier notre hypothèse de départ (l'activité concurrente de plusieurs algorithmes peut accélérer des traitements par le biais d'une coopération entre ces algorithmes) et transférer les concepts qui nous semblent pertinents du parallélisme aux métaheuristiques, nous proposons un algorithme génétique parallèle fondé sur le modèle Client/Serveur. Cet algorithme est nommé CSM-PGA pour « Client/Server Model for Parallel Genetic Algorithm ». Il est le premier algorithme génétique parallèle conçu pour des problèmes CARP[3].

Dans cette étude, l'environnement de calcul distribué est réalisé par plusieurs PCs connectés en réseau, et nous proposons un algorithme génétique distribué en utilisant le modèle de gestion délégué. Ce modèle facilite la mise en œuvre des algorithmes génétiques parallèles et

[3] Capacitated Arc Routing Problem

la réduction de la quantité des communications. La population est divisée en un ensemble de sous-populations. Le modèle proposé est mis en œuvre sous forme d'un client-serveur. Le serveur consiste en la gestion déléguée des clients qui exécutent chacun un algorithme génétique sur une sous-population. En outre, à ce modèle s'ajoutent des règles de communication entre les sous-populations.

L'efficacité de cette approche est évaluée expérimentalement par le traitement d'un problème de tournées sur arcs (CARP[3]).

1.2.2 Une plate-forme orienté-agent pour la conception de métaheuristiques parallèles(MAF-DISTA)

Pour intégrer l'approche multi-agents dans la conception de métaheuristiques parallèles à base de population, nous proposons une plate-forme nommée *MAF-DISTA* pour « Multi-Agent Framework for DISTributed population-based Algorithms ». Une plate-forme est un ensemble d'outils facilitant le développement d'applications. Dans le cadre des métaheuristiques parallèles à base de population, elle doit fournir un modèle ou un squelette assez générique pour être réutilisé lors de la conception de différentes métaheuristiques parallèles à base de population. Ainsi, en donnant un cadre de modélisation commun, une plate-forme doit faciliter l'analyse des algorithmes existants, et assister la conception de nouveaux algorithmes.

MAF-DISTA se distingue des plates-formes existantes en introduisant les concepts permettant de traiter la distribution et l'adaptation dans les métaheuristiques parallèles à base de population. Nous exploitons pour cela plusieurs notions issues du domaine des systèmes multi-agents. Tout d'abord, le modèle de métaheuristique que nous proposons dans le cadre de *MAF-DISTA* est un modèle organisationnel qui décrit le système sous la forme d'une organisation composée de rôles en interaction. L'intérêt de cette approche, actuellement utilisée dans les systèmes multi-agents, est de pouvoir décrire un système aussi bien comme un tout, le système multi-agent, que comme un assemblage de composants, les agents. De plus, cette approche permet de distinguer l'analyse des fonctions du système, de l'analyse de son architecture. Enfin, l'approche organisationnelle encourage la modularité et la réutilisation des modèles. Nous proposons en complément de ce modèle un guide méthodologique. Il définit un ensemble d'étapes permettant de passer du modèle organisationnel à une métaheuristique parallèle exprimée en termes d'agents. En proposant cette plate-forme, nous souhaitons, d'une part, proposer une démarche de conception de métaheuristiques parallèles à base de population suivant un cadre commun de modélisation, et d'autre part, encourager la distribution et l'utilisation de mécanismes d'adaptation.

1.3 Plan de la thèse

Après ce bref exposé de nos propositions, cette section introduit le plan de la thèse. Le mémoire est organisé en trois parties. La première présente un état de l'art sur les métaheuristiques et les systèmes multi-agents. La seconde partie introduit les plates-formes ainsi que la métaheuristique que nous proposons. La dernière partie présente les résultats expérimentaux de notre approche sur deux problèmes d'optimisation. Cette structuration est illustrée par la figure 1.1

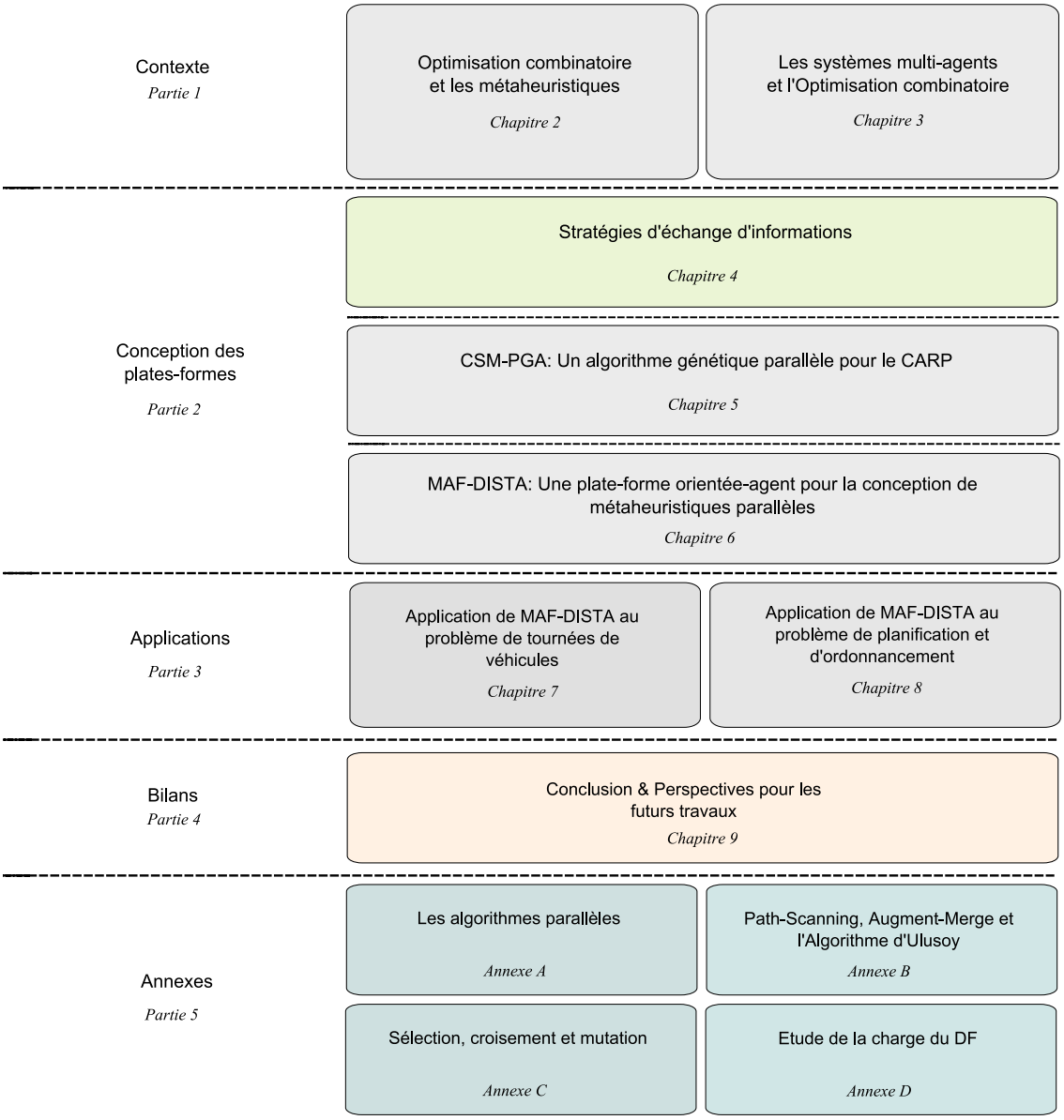


FIG. 1.1 – Plan de la thèse

Ce mémoire est scindé en neuf chapitres :

- Dans le second chapitre, nous présentons tout d'abord quelques définitions sur l'optimisation et les métaheuristiques. Ensuite, un panorama des métaheuristiques est dressé. Enfin, un état de l'art dédié aux modèles et plates-formes de métaheuristiques séquentielles/parallèles est présenté.
- Le troisième chapitre a trait à l'utilisation des systèmes multi-agents dans le domaine de l'optimisation. Pour cela, quelques définitions sur les systèmes multi-agents sont énoncées, puis nous donnons une vue d'ensemble des aspects multi-agents présents dans les heuristiques et les métaheuristiques. Ce chapitre permet de mettre en avant l'intérêt

des systèmes multi-agents dans la conception de métaheuristiques parallèles.

- Dans le quatrième chapitre, nous présentons les différentes stratégies d'échange d'informations que nous mettons en œuvre. Trois stratégies d'échange d'informations ont été élaborées.
- Dans le cinquième chapitre, nous présentons *CSM-PGA* : un algorithme génétique parallèle fondé sur le modèle Client/Serveur. Nous décrivons dans un premier temps le modèle de cette métaheuristique parallèle puis nous détaillons la stratégie de coopération mise en œuvre. De plus, *CSM-PGA* a été testée sur des instances de problème de tournées sur arcs (CARP) et comparée avec les meilleures méthodes existantes pour le CARP.
- Le sixième chapitre présente de manière approfondie la plate-forme *MAF-DISTA*[4] en commençant par la description du modèle organisationnel de métaheuristique parallèle. Ensuite, nous détaillons le guide méthodologique.
- Le septième chapitre constitue une première application de notre approche sur un problème de tournées de véhicules. Dans un premier temps, nous essayons de valider expérimentalement les mécanismes d'auto-adaptation et de coopération mis en œuvre dans *MAF-DISTA*. Ensuite, nous confrontons les résultats expérimentaux de *MAF-DISTA* à différentes approches de résolution existantes.
- Le huitième chapitre propose une seconde application de *MAF-DISTA* sur un problème de planification et d'ordonnancement multi-produits. En plus d'apporter une seconde validation expérimentale de l'efficacité de l'approche, nous cherchons à illustrer son aspect de flexibilité, c'est-à-dire, la facilité d'adaptation sur différents problèmes d'optimisation.
- Le mémoire se termine par un bilan des travaux de recherche. Nous dressons un ensemble de perspectives pouvant être développées par la suite, ouvrant ainsi de nouvelles voies et propositions dans le but d'améliorer ce travail.

1.4 Publications

Cette thèse a conduit à des publications, en particulier autour du projet *SEISCO*[5]. Ces publications sont les suivantes :

Chapitres de livres

[1] **K. Belkhelladi, P. Chauvet and A. Schaal.** *Cooperation Control in Distributed Population-based Algorithms using a Multi-agent Approach - Application to a real-life Vehicle Routing Problem.* Robotics, Automation and Control. Accepted and to appear in IN-TECH Book(February 2010). ISBN : 978-953-7619-39-8.

[4] Mobile Agent Framework Distributed population based Algorithms
[5] Stratégies d'Échange d'Informations dans un Système de Calcul réparti pour l'Optimisation

Revues internationales

[2] **K. Belkhelladi, P. Chauvet and A. Schaal.** *MAF-DISTA : A mobile agent framework as a tool for implementing parallel population-based algorithms.* En révision, International Journal of Multi-Agent and Grid Systems(2010).

Conférences internationales avec comité de lecture

[3] **K. Belkhelladi, P. Chauvet, F. Réveillère and A. Schaal.** *An intelligent distributed genetic algorithm for the capacitated arc routing problem.* In proceedings of the European Conference On complex Systems (ECCS'07), Dresden, Germany (2007), pp. 117.

[4] **K. Belkhelladi, P. Chauvet and A. Schaal.** A*n efficient information exchange strategy in a distributed computing system - Application to the CARP.* In proceedings of the 5^{th} International Conference on Informatics in Control, Automation and Robotics, Madeira, Portugal (2008), pp. 342-346. ISBN : 978-989-8111-30-2.

[5] **K. Belkhelladi, P. Chauvet and A. Schaal.** A*n Agent Framework with an Efficient Information Exchange Model for Distributed Genetic Algorithms.* In proceedings of the 2008 IEEE World Congress on Computational Intelligence(WCCI'08), Hong Kong, China (2008), pp. 848-853. ISBN : 978-1-4244-1823-7.

[6] **K. Belkhelladi, P. Chauvet, B. Daya and A. Schaal.** *A mobile agent framework to support parallel computing - Application to Multi-product Planning and Scheduling Problems.* In proceedings of the 2^{nd} International Conference on Developments in eSystems Engineering (DeSE'09), Abu Dhabi, UAE (2009), pp. 335-342. ISBN : 978-0-7695-3912-6.

[7] **A. H. Akoum, B. Daya, K. Belkhelladi, L. Prevost and P. Chauvet.** *Fusion of basic algorithms for detection and localization of vehicle plate numbers.* In proceedings of the 30^{th} International Conference on Information Systems, Architecture, and Technology (ISAT'09), Szklarska Poręba, Poland (2009), pp. 119-128. ISBN : 978-83-7493-478-7 (Prix du meilleur papier).

Conférences nationales avec comité de lecture

[8] **K. Belkhelladi, P. Chauvet and A. Schaal.** *Une approche multi-agents pour l'optimisation par algorithme génétique distribué.* In proceedings du 9^{e} Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la décision (ROADEF'08), Clermont-Ferrand, France (2008). p. 1-16, ISBN : 978-2-84516-378-2.

[9] **K. Belkhelladi, P. Chauvet, L. Péridy and A. Schaal.** *Un système d'agents pour la planification et l'ordonnancement multi-produits, multi-modes avec approvisionnements.* In proceedings du 10^{e} Congrès de la société Française de la Recherche Opérationnelle et d'Aide à la Décision(ROADEF'09), Nancy, France (2009).

Communications sans actes

[10] **K. Belkhelladi.** *Stratégies d'échange d'informations dans un système de calcul réparti pour l'optimisation des problèmes combinatoires.* Séminaire du PRiSM, Université de Versailles Saint-Quentin-en-Yvelines, France (2010).

[11] **K. Belkhelladi.** *MAF-DISTA : Une approche d'agents mobiles pour la parallélisation des algorithmes à base de population.* Séminaire du CReSTIC, Université de Reims Champagne- Ardenne, France (2010).

[12] **K. Belkhelladi, P. Chauvet and A. Schaal.** *Une approche Multi-agents pour la modélisation des métaheuristiques parallèles - Application aux problèmes de tournées de ramassage des conteneurs enterrés.* Séminaire du LISA, Angers, France (2009).

[13] **K. Belkhelladi, P. Chauvet and A. Schaal.** *Stratégies d'échange d'informations dans les systèmes de calcul réparti intelligent.* 7^{e} Forum de l'école doctorale d'Angers, Angers, France, Juin 2007.

Partie 1 : Contexte

Chapitre 2

Optimisation combinatoire et métaheuristiques parallèles

Sommaire

2.1 Introduction

Les métaheuristiques ont connu un essor considérable depuis leur apparition dans les années 1970. Elles sont présentées par Osman et Laporte (1996) comme étant des méthodes d'approximation conçues dans le but de s'attaquer à des problèmes complexes d'optimisation qui n'ont pu être résolus de façon efficace par les heuristiques et les méthodes d'optimisation classiques. Ces mêmes auteurs définissent formellement la notion de métaheuristique comme étant un processus itératif qui guide une heuristique subordonnée en combinant intelligemment différents concepts pour explorer et exploiter l'espace de recherche, et qui utilise des stratégies d'apprentissage pour structurer l'information dans le but de trouver efficacement des solutions les plus rapprochées possible de la solution optimale. Le développement des métaheuristiques fait partie d'un effort soutenu investi dans le domaine de l'optimisation combinatoire. Ce dernier est défini comme étant l'étude mathématique de la recherche d'un arrangement, d'un groupement, d'un ordonnancement ou d'une sélection d'objets discrets habituellement finis en nombre (Osman et Laporte (1996)). Malgré les progrès remarquables qu'ont connu les algorithmes exacts durant ces dernières années, l'optimisation combinatoire constitue toujours un défi de taille pour eux. Par conséquent, les algorithmes d'approximation sont devenus une sphère importante de recherche et d'applications dans ce domaine (Delisle (2002)).

Un problème d'optimisation se définit formellement par un triplet $\mathcal{P} = (\mathcal{I}, \mathcal{R}, f)$ avec,

- $\mathcal{I}$ l'ensemble des instances de $\mathcal{P}$.
- Étant donné une instance $x \in \mathcal{I}, \mathcal{R}(x)$ désigne l'ensemble fini des solutions réalisables et donc candidates.
- Pour une instance $x \in \mathcal{I}$, il existe une fonction f de coût (ou fitness) associant à chaque solution candidate $\sigma \in \mathcal{R}(x)$ une mesure de qualité $f(x, \sigma)$ en considération du problème traité. Cette métrique est nommée la fonction objectif.

Résoudre un problème d'optimisation combinatoire consiste à déterminer l'ensemble des solutions optimales $\mathcal{S} \subset \mathcal{R}(x)$ maximisant la fonction objectif et vérifiant ainsi,

$$\forall \sigma^* \in \mathcal{S}, \forall \sigma \in \mathcal{R}(x), f(x, \sigma^*) \geq f(x, \sigma)$$

Les problèmes d'optimisation combinatoire, issus de domaines tels que les transports, les télécommunications, etc., sont en majorité *NP-difficiles*, signifiant qu'il n'existe pas d'algorithmes produisant une solution optimale en un temps d'exécution polynômial par rapport à la taille des données manipulées. Leur résolution est donc toujours limitée par la capacité des ressources matérielles utilisées (puissance de calcul, mémoire, etc.).

Même si les métaheuristiques offrent des stratégies efficaces pour la recherche de solutions de problèmes d'optimisation combinatoire, les temps de calcul associés à l'exploration de l'espace de recherche peuvent être très importants (Cung *et al.* (2002)). Une façon d'accélérer cette exploration est l'utilisation du parallélisme. Une autre contribution du calcul parallèle aux métaheuristiques est de plus en plus constatée : en leur appliquant des paramètres appropriés, les métaheuristiques parallèles peuvent être beaucoup plus robustes que leurs équivalents séquentiels en termes de qualité de solutions trouvées (Crainic et Toulouse (1998)). Certains mécanismes parallèles permettent donc de trouver de meilleures solutions sans nécessiter un nombre total d'opérations plus grand. Le parallélisme peut donc représen-

ter un apport considérable aux métaheuristiques et un nombre grandissant de travaux sont effectués afin d'exploiter ce potentiel.

2.2 Classification des métaheuristiques

Il existe plusieurs façons de classer les métaheuristiques. On peut faire la différence entre les métaheuristiques qui s'inspirent de phénomènes naturels et celles qui ne s'en inspirent pas. Par exemple, les algorithmes génétiques et les algorithmes des fourmis s'inspirent respectivement de la théorie de l'évolution et du comportement de fourmis à la recherche de nourriture. Par contre, la méthode Tabou n'a semble-t-il pas été inspirée par un phénomène naturel. Une telle classification ne semble cependant pas très utile et est parfois difficile à réaliser. En effet, il existe de nombreuses métaheuristiques récentes qu'il est difficile de classer dans l'une des deux catégories. Certains se demanderont par exemple si l'utilisation d'une mémoire dans la méthode Tabou n'est pas directement inspirée de la nature (Hertz (2005)).

Une autre façon de classer les métaheuristiques est de distinguer celles qui travaillent avec une population de solutions de celles qui ne manipulent qu'une seule solution à la fois. Les méthodes qui tentent itérativement d'améliorer une solution sont appelées méthodes de recherche locale ou méthodes de trajectoire. La méthode Tabou (Glover (1989), Glover (1990)), le Recuit Simulé (Metropolis *et al.* (1953)), la Recherche à Voisinages Variables (Mladenović et Hansen (1997)) et la procédure de recherche gloutonne aléatoire adaptative (GRASP[1])(Feo et Resende (1989)) sont des exemples typiques de méthodes de trajectoire. Ces méthodes construisent une trajectoire dans l'espace des solutions en tentant de se diriger vers des solutions optimales. L'exemple le plus connu des méthodes travaillant avec une population de solutions est l'algorithme génétique (Holland (1992b)).

Les métaheuristiques peuvent également être classées selon leur manière d'utiliser la fonction objectif. Étant donné un problème d'optimisation consistant à minimiser une fonction f sur un espace S de solutions, certaines métaheuristiques dites statiques, travaillent directement sur f alors que d'autres, dites dynamiques, font usage d'une fonction g obtenue à partir de f en ajoutant quelques composantes qui permettent de modifier la topologie de l'espace des solutions, ces composantes additionnelles pouvant varier durant le processus de recherche (Hertz (2005)).

Des chercheurs préfèrent classer les métaheuristiques en fonction du nombre de structures de voisinages utilisées. Étant donné qu'un minimum local relativement à un type de voisinage ne l'est pas forcément pour un autre type de voisinage, il peut être intéressant d'utiliser des métaheuristiques basées sur plusieurs types de voisinages (Hertz (2005)).

Certaines métaheuristiques font usage de l'historique de la recherche au cours de l'optimisation, alors que d'autres n'ont aucune mémoire du passé. Les algorithmes sans mémoire sont en fait des processus Markoviens (Derman (1970)) puisque l'action à réaliser est totalement déterminée par la situation courante. Les métaheuristiques qui font usage de l'historique de la recherche peuvent le faire de diverses manières. On distingue généralement les méthodes ayant une mémoire à court terme de celles qui ont une mémoire à long terme (Hertz (2005)).

Finalement, mentionnons que certaines métaheuristiques utilisent des concepts additionnels que sont la diversification et l'intensification. Par diversification, on sous-entend généra-

[1] Greedy Randomized Adaptive Search Procedure

lement une exploration assez large de l'espace de recherche, alors que le terme intensification vient plutôt mettre l'accent sur l'exploitation de l'information accumulée durant la recherche. Il est important de bien doser l'usage de ces deux ingrédients afin que l'exploration puisse rapidement identifier des régions de l'espace de recherche qui contiennent des solutions de bonne qualité, sans perdre trop de temps à exploiter des régions moins prometteuses. Dans notre description des principales métaheuristiques, nous allons nous appuyer sur la classification qui distingue les méthodes à solution unique des méthodes basées sur des populations de solutions (Hertz (2005)).

Le diagramme de la figure 2.1 tente de présenter où se placent quelques unes des méthodes (métaheuristiques) les plus connues dans la littérature. Un élément présenté sur différentes catégories indique que l'algorithme peut être placé dans l'une ou l'autre classe, selon le point de vue adopté.

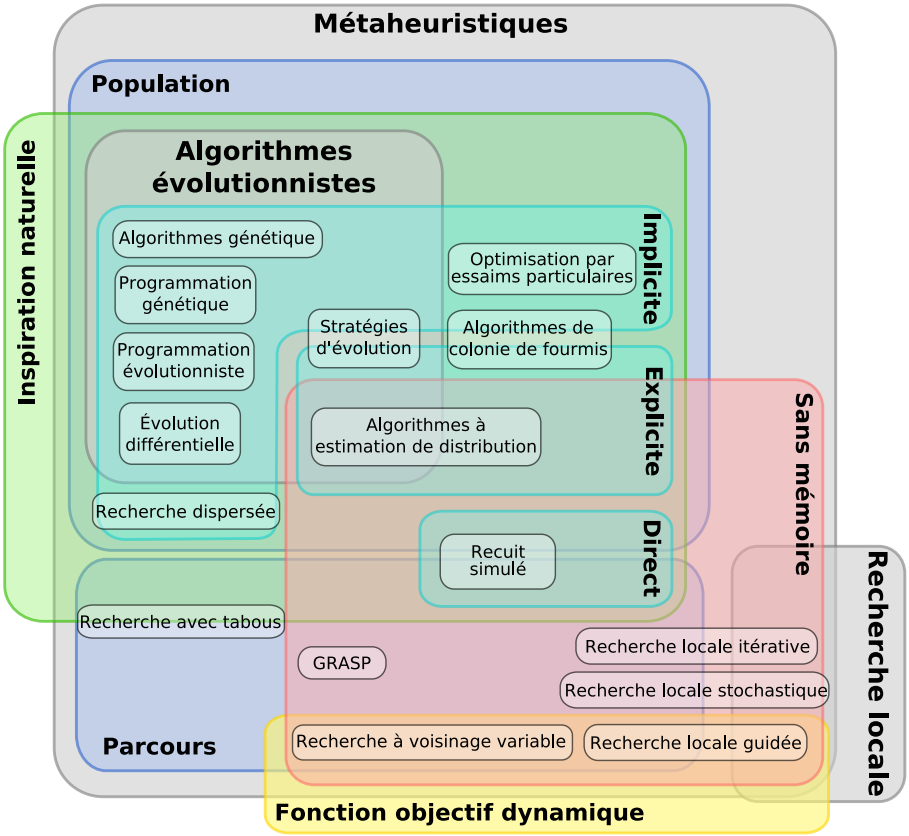


FIG. 2.1 – Différentes classifications des métaheuristiques (source, Dréo (2007))

Après avoir dressé une classification des métaheuristiques séquentielles, la section suivante présente une classification des métaheuristiques parallèles.

2.3 Classification des métaheuristiques parallèles

Crainic et Toulouse (1998), Cung *et al.* (2002), Gras *et al.* (2003) ont tenté, en examinant les particularités communes aux différentes implémentations parallèles retrouvées dans le monde des métaheuristiques, d'en établir une classification. Le contenu de cette section s'inspire des classifications décrites par Delisle (2002).

2.3.1 La classification de Crainic et Toulouse (1998)

Cette classification, divisée en trois catégories, est basée sur le degré d'impact de la stratégie de parallélisation sur la structure algorithmique de la métaheuristique.

La première catégorie représente une stratégie de parallélisation à l'intérieur d'une itération de la métaheuristique. Cette stratégie, considérée comme étant une parallélisation de bas niveau ou une parallélisation interne, s'applique de façon relativement directe et a pour but d'accélérer les calculs sans chercher à effectuer une meilleure exploration. La méthode de résolution parallèle est la même que son équivalent séquentiel dans la mesure où le nombre total d'opérations est identique pour les deux versions. À cet effet, il est possible de profiter de la puissance de calcul disponible et d'améliorer ainsi la recherche de solutions en utilisant cette stratégie sans changer fondamentalement l'algorithme. Il s'agit alors d'effectuer un plus grand nombre d'opérations en parallèle tout en conservant un temps d'exécution global égal ou inférieur à la version séquentielle.

La deuxième catégorie est une stratégie de décomposition du domaine du problème ou de l'espace de recherche. Le principe de cette approche consiste à diviser le problème en sous-problèmes et à utiliser la puissance de calcul afin de résoudre ces sous-problèmes en parallèle. Le comportement de recherche est alors modifié par rapport à l'algorithme séquentiel. Un modèle *maître-esclave* est généralement utilisé pour appliquer cette stratégie. Le *maître* détermine séquentiellement les partitions initiales et les modifie durant l'exécution à des intervalles qui peuvent être prédéfinis ou déterminés dynamiquement durant l'exécution. Les *esclaves* explorent de façon concurrente et indépendante l'espace de recherche qui leur a été assigné par le *maître* et ce dernier assemble les solutions partielles trouvées par les *esclaves* en une solution complète au problème.

Dans la troisième catégorie, plusieurs processus de recherche (*multi-search threads*) sont appliqués avec différents degrés de synchronisation et de coopération. Par cette stratégie, on cherche à effectuer une recherche plus complète en mettant en œuvre plusieurs processus qui opèrent simultanément et qui sont guidés de différentes façons. La métaheuristique peut être calibrée différemment pour chacune des recherches, mais il est également possible d'utiliser de façon concurrente différentes métaheuristiques. Les processus peuvent communiquer entre eux durant leur recherche ou seulement à la fin pour identifier la meilleure solution.

On peut alors subdiviser les processus de recherche multiples de cette troisième catégorie en deux classes : *approches indépendantes* et *approches coopératives*. Dans la première classe, plusieurs recherches sont initiées et le meilleur résultat est sélectionné parmi celles-ci à la fin. Cette approche est alors équivalente à une stratégie de redémarrage accéléré. Plutôt que de redémarrer l'algorithme plusieurs fois et selon différents paramètres, toutes les exécutions sont effectuées en parallèle. Afin de définir une recherche multiple et coopérative, c'est-à-dire une approche de la deuxième classe, plusieurs paramètres importants doivent être déterminés.

Ceux-ci sont :

- la topologie du réseau d'interconnexion qui définit comment les processus sont reliés entre eux ;
- la méthode de communication entre les processus (diffusion, point à point, mémoire centrale, etc.) ;
- le choix des processus participant aux échanges d'informations ;
- le type de communication (synchrone ou asynchrone) ;
- les moments où l'information sera échangée ;
- l'information à échanger.

L'ajustement de ces paramètres peut avoir un impact significatif sur le comportement et la performance de la recherche. En effet, selon les auteurs de cette classification, on commence à réaliser, dans la communauté des métaheuristiques parallèles, l'importance de la définition de mécanismes de coopération et de l'étude de leur impact sur la performance des métaheuristiques. Cet aspect comporte donc encore, selon eux, plusieurs enjeux à attaquer et représente un domaine de recherche prometteur. Toujours dans l'optique d'en dégager les principales caractéristiques, Cung *et al.* (2002) ont proposé une classification des stratégies de parallélisation des métaheuristiques qui est indépendante de l'architecture utilisée. Cette classification, qui se rapproche beaucoup de celle de Crainic et Toulouse (1998) tout en comportant ses propres particularités, se base sur les principes de recherche locale parallèle tels que présentés par Verhoeven et Aarts (1995).

2.3.2 La classification de Cung *et al.* (2002)

Les métaheuristiques basées sur la recherche locale peuvent être vues comme une exploration d'un graphe de voisinage associé à une instance de problème. Dans ce graphe, les nœuds correspondent à des solutions et les arcs connectent des solutions voisines. Chaque itération consiste en l'évaluation des solutions présentes dans le voisinage de la solution courante et au déplacement vers une de celles-ci, tout en évitant le plus possible de rester pris au piège prématurément dans un optimum local. Ce processus d'évaluation et de déplacement est répété jusqu'à l'atteinte d'un nombre maximal d'itérations ou jusqu'à ce qu'il ne soit plus possible de trouver de meilleures solutions. Les solutions visitées durant cette recherche définissent une marche, aussi appelée trajectoire, dans le graphe de voisinage. Les implémentations parallèles des métaheuristiques utilisent alors plusieurs processeurs afin de générer ou explorer ce graphe de façon concurrente. Comme ce dernier n'est pas connu d'avance, les parallélisations de métaheuristiques sont des applications irrégulières (Roch *et al.* (1995)) dont l'efficacité dépend fortement du choix de la granularité de l'algorithme parallèle et de l'utilisation de techniques d'équilibrage de charges.

On distingue alors deux approches pour la parallélisation de la recherche locale basées sur le nombre de marches qui sont effectuées dans le graphe de voisinage :

1. **Marche simple :** tâches de granularité fine à moyenne.
2. **Marches multiples :** tâches de granularité moyenne à grosse :
   - Processus de recherche indépendants.
   - Processus de recherche coopératifs.

Dans le cas d'une parallélisation à marche simple, une trajectoire unique est traversée dans le graphe de voisinage. La recherche du meilleur voisin est effectuée en parallèle à chaque ité-

ration par la parallélisation de la fonction d'évaluation des solutions ou par décomposition de domaine. Cette approche, dont le but est d'accélérer la traversée du graphe de voisinage, correspond sensiblement à la première catégorie définie plus tôt par Crainic et Toulouse (1998). Si la fonction d'évaluation des solutions voisines est parallélisée, alors une accélération substantielle peut être atteinte sans modifier la trajectoire parcourue par la méthode séquentielle. Dans l'autre cas, la décomposition du voisinage et sa répartition sur plusieurs processeurs permet d'examiner une part plus large du voisinage que par une approche séquentielle. Par conséquent, la trajectoire suivie peut être mieux guidée et conduire ainsi à de meilleures solutions.

La plupart des implémentations parallèles de métaheuristiques, excepté le recuit simulé, sont basées sur des stratégies de marches multiples. L'idée principale est d'effectuer l'exploration de plusieurs trajectoires en parallèle sur des processeurs distincts. Les tâches exécutées en parallèle ont alors une granularité plus grosse. En plus d'accélérer les traitements, ces stratégies cherchent à améliorer la qualité des solutions trouvées. Cette catégorie, se subdivisant selon le caractère indépendant ou coopératif des processus de recherche, est semblable à la troisième catégorie définie par Crainic et Toulouse (1998).

Dans le cas des processus de recherche indépendants, on distingue deux approches de base. La première est l'exploration de plusieurs trajectoires provenant de différents endroits du graphe en débutant la recherche à partir de différentes solutions ou populations de départ. Les processus de recherche peuvent utiliser ou non le même algorithme avec des paramètres semblables ou différents. La deuxième correspond à l'exploration en parallèle de sous-graphes obtenus par décomposition du domaine sans qu'il n'y ait d'intersection entre les trajectoires correspondantes.

Si les processus de recherche sont coopératifs, ils échangent ou partagent l'information qu'ils ont accumulé durant leur parcours afin d'améliorer leur recherche respective. Ce partage d'informations peut être implémenté par des variables globales dans une mémoire partagée ou par un bassin de variables locales à un processeur dédié accessible par tous les autres processeurs. Dans ce modèle, on cherche non seulement à accélérer la convergence vers de bonnes solutions mais aussi à trouver de meilleures solutions qu'en utilisant des stratégies de marches indépendantes tout en leur accordant le même temps d'exécution. Comme il a été mentionné plus tôt, la principale difficulté dans ce modèle de parallélisation est de déterminer la nature des partages ou des échanges d'informations afin d'améliorer la recherche sans trop sacrifier de temps ou de mémoire.

2.3.3 La classification de Gras *et al.* (2003)

Gras *et al.* (2003) ont présenté une nouvelle classification des métaheuristiques parallèles en considérant la propriété de la coopération entre les entités explorant l'espace de recherche. Ce schéma de classification comprend trois sous-classes : centralisée, individuelle et coopération concertée, selon la manière dont la coopération est accomplie. Ils ont donné de nouvelles définitions et décrit les nouveaux développements de ces approches, et ont essayé de montrer leurs avantages. Ils démontrent que la division hiérarchique du problème en tâches indépendantes peut conduire à étendre et à simplifier les étapes d'optimisation. Les résultats de ces optimisations sont ensuite associés pour produire une solution globale au profit des structures figurant dans les différents niveaux considérés. Ils ont appliqué ces techniques à deux pro-

blèmes centraux de la protéomique : identification automatique des protéines et alignement des séquences multiples basé sur le motif d'inférence. Pour le premier problème, ils ont mis au point un algorithme à fourmis parallèle coopératif (Gras *et al.* (2003)), où les fourmis trouvent des solutions et échangent des informations sur ces solutions entre elles. Pour le deuxième problème, une stratégie multi-agents coopérative a été élaborée, qui tire avantage de la coopération concertée pour parvenir à un regroupement entièrement automatisé des séquences biologiques. L'algorithme global suit un cours asynchrone où chaque agent effectue une succession d'actions, indépendamment de l'état d'avancement des autres agents. Une fois les agents initialisés, ils exécutent un algorithme évolutionnaire pour calculer une première solution. Les agents échangent des informations sur leurs solutions, de sorte que certains individus d'un agent peuvent être envoyés à un autre agent pour améliorer la qualité de la population.

Le diagramme de la figure 2.2 récapitule les différentes classes de métaheuristiques parallèles ; en se basant ainsi sur les différentes classifications mentionnées ci-dessus.

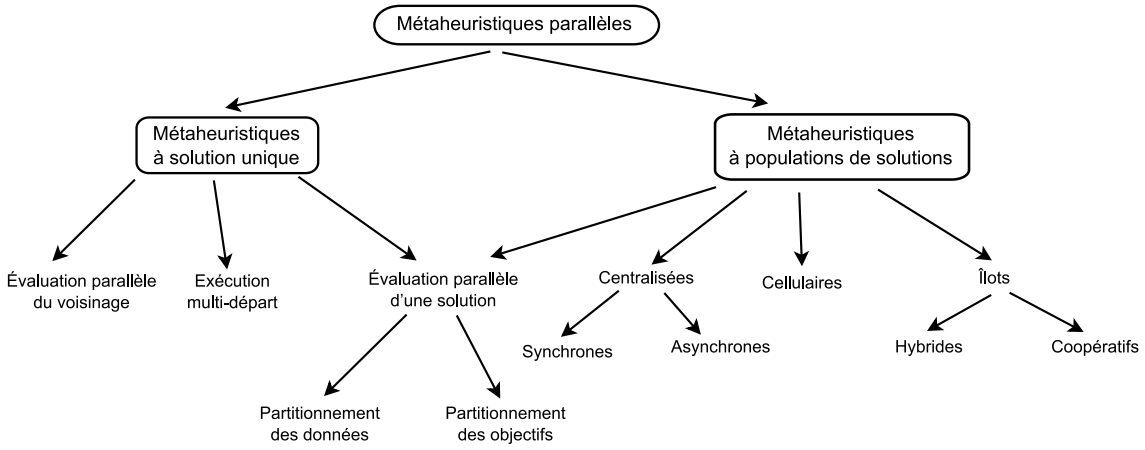


FIG. 2.2 – Différentes classifications des métaheuristiques parallèles.

Les classifications présentées soulignent l'importance de la stratégie de parallélisation sur la qualité de la métaheuristique résultante. Afin d'évaluer cette qualité, on peut, dans certains cas, utiliser les mesures standards de performance des algorithmes parallèles qui sont : l'accélération, l'efficacité, le nombre de processeurs et l'iso-efficacité (cf. l'annexe A). Cependant, ces mesures ne sont pas suffisantes pour la plupart des cas de parallélisation de métaheuristiques. La section suivante aborde cette question de façon plus détaillée.

2.4 Mesures de performance des métaheuristiques parallèles

Si la stratégie de parallelisation utilisée correspond à la première catégorie de la classification de Crainic et Toulouse (1998), c'est-a-dire à une parallelisation interne, l'algorithme parallèle suit alors la même voie d'exploration que son équivalent séquentiel et ne modifie pas son comportement. Le bénéfice apporté par la parallelisation consiste alors à réduire le temps d'exécution de la métaheuristique tout en conservant la même qualité de solution. On

peut ainsi mesurer la performance de l'algorithme parallèle en lui appliquant directement les critères d'évaluation standard, notamment l'accélération et l'efficacité. Cependant, si la stratégie de parallelisation implique un nombre total d'opérations différent de la méthode séquentielle ou que des modifications ont été apportées dans le processus de recherche de solutions, il faut alors mettre la performance de l'algorithme parallèle en relation avec la qualité des solutions qu'il permet d'obtenir (Delisle (2002)).

Cung *et al.* (2002) abordent l'efficacité des implémentations parallèles des métaheuristiques en utilisant une stratégie de marches multiples indépendantes basée sur l'exécution de plusieurs copies du même algorithme séquentiel. Une valeur cible τ pour la fonction à optimiser est diffusée à tous les processeurs qui exécutent ensuite l'algorithme séquentiel de façon indépendante. Tous les processeurs s'arrêtent aussitôt lorsque l'un d'entre eux a trouvé une solution égale ou meilleure à τ. L'accélération est alors donnée par le ratio entre le temps requis pour trouver une solution au moins aussi bonne que τ en utilisant l'algorithme séquentiel et celui requis en utilisant l'implémentation parallèle avec P processeurs. Cette approche est intéressante si l'algorithme est basé sur un processus de recherche aléatoire (Stützle et Hoos (1998)). Dans ce cas, les exécutions parallèles indépendantes ont de fortes chances d'explorer des regions différentes de l'espace de recherche. Certaines d'entre elles peuvent être plus intéressantes que d'autres et mener à de bonnes solutions plus rapidement qu'avec une seule exécution séquentielle. Cela revient à affirmer que l'algorithme séquentiel serait plus efficace si on le redémarrait k fois en lui donnant un temps t à chaque fois plutôt que de l'exécuter une fois seulement en lui donnant un temps $k \times t$ (Delisle (2002)).

La contribution du parallélisme aux métaheuristiques peut être plus grande qu'une simple accélération du temps de calcul. Par des marches multiples indépendantes mais surtout coopératives, il est possible d'améliorer, dans plusieurs cas, la qualité des solutions trouvées par la métaheuristique séquentielle (Crainic et Toulouse (1998), Cung *et al.* (2002)) et ce, sans nécessiter un plus grand nombre de calculs. Afin de mesurer cette contribution, une démarche relativement simple et logique consiste à comparer les valeurs des meilleures solutions obtenues pour la résolution d'un même problème par les versions séquentielle et parallèle de la métaheuristique. Notons qu'il faut accorder un même nombre d'opérations total aux deux versions pour que la comparaison soit valide. Cette mesure peut relever davantage d'un jugement de valeurs que de calculs mathématiques. En effet, il n'est pas suffisant de déterminer si les solutions obtenues par l'implémentation parallèle sont supérieures à l'algorithme séquentiel car il faut également se demander si le gain obtenu justifie les moyens de calcul et l'effort algorithmique déployés. Cette justification est dépendante du problème abordé et des objectifs recherchés et nécessite l'apport d'un certain jugement (Delisle (2002)).

Si la stratégie employée implique des marches multiples coopératives, une évaluation adéquate des bénéfices apportés par les mécanismes de coopération pourrait être souhaitable (Huberman (1990)). Il serait alors approprié de comparer les résultats d'une parallélisation à marches multiples indépendantes et d'une à marches multiples cooperatives de la même métaheuristique afin de souligner les effets de la cooperation. Dans la prochaine section, le concept de cooperation impliqué dans les strategies de marches multiples cooperatives sera présenté plus en details.

2.5 La coopération dans les métaheuristiques parallèles

L'idée de base de la coopération est qu'un groupe d'agents coopérants engagés dans la résolution d'un problème peut résoudre ce dernier plus efficacement qu'un agent seul ou qu'un groupe d'agents opérant isolément les uns des autres. Plusieurs exemples tirés de la vie humaine et animale tendent à confirmer cette idée. D'un point de vue informatique, il peut être intéressant d'appliquer cette approche dans la conception d'algorithmes de résolution de problèmes complexes. L'idée de résoudre un problème à l'aide de plusieurs agents coopérants ou isolés qui opèrent en même temps présente un parallélisme naturel. Une architecture parallèle semble donc appropriée pour transposer cette idée dans un contexte informatique.

De plus, un des principes fondamentaux des métaheuristiques est qu'elles mémorisent des solutions ou des caractéristiques des solutions visitées durant le processus de recherche passé et utilisent cette information pour les guider dans la recherche à venir (Taillard *et al.* (1998)). Un processus d'apprentissage est donc effectué durant l'exécution de l'algorithme et influence son comportement. Comme la qualité de l'algorithme est intimement liée à la pertinence de l'information conservée et utilisée, l'intégration de mécanismes de coopération peut créer de l'information nouvelle qui permettra à l'algorithme d'effectuer une meilleure recherche. La coopération permet donc une nouvelle forme d'apprentissage susceptible d'améliorer la performance des métaheuristiques (Delisle (2002)).

Clearwater *et al.* (1992) définissent la coopération comme un processus impliquant un ensemble d'agents qui interagissent en se communiquant de l'information les uns aux autres durant la résolution d'un problème. La communication peut se faire, par exemple, par des diffusions entre les processeurs ou par l'accès à une source d'informations centralisée. Selon ces mêmes auteurs, le comportement des agents impliqués dans un processus coopératif de recherche de solutions est déterminé par plusieurs éléments. Quelques uns de ceux-ci sont la nature du processus de recherche des agents (chaque agent peut avoir un processus semblable ou différent), la nature de l'information partagée, la façon dont elle est partagée et la structure organisationnelle du groupe d'agents.

Toulouse *et al.* (1995) ont cherché à identifier les principales questions qui devraient être posées lors de la conception de processus de recherche coopératifs et montrent que les questions reliées aux communications inter-agents sont de première importance. Selon ces auteurs, il faut déterminer principalement la nature de l'information à partager, le moment où elle devrait être partagée et entre quels processus elle devrait être partagée.

2.5.1 La nature des informations à partager

Quand vient le temps de déterminer quelles informations doivent être partagées, un facteur important à considérer est la quantité d'informations à partager parmi toutes celles qui ont été produites par les processus parallèles. Rendre disponible à tous les processus une grande quantité d'informations peut engendrer des coûts de communication et/ou d'accès à la mémoire très prohibitifs. De plus, il ne sera pas nécessairement intéressant pour les autres processus d'avoir accès à toute l'information disponible. Il est possible que certaines informations produites par un processus n'apportent rien et même nuisent aux autres. L'étude de la pertinence des données à partager est donc importante. Cependant, cette pertinence est généralement difficile à représenter dans un modèle quantitatif. La connaissance du pro-

blème et le jugement sont bien souvent les meilleures façons d'évaluer l'intérêt de partager une information plutôt qu'une autre (Delisle (2002)).

2.5.2 Le moment où l'information est partagée

Si les processeurs ont un accès sans restriction à l'information partagée, il peut se produire une détérioration de la performance de recherche de solutions. Par exemple, si l'information partagée fait trop souvent partie du processus de décision, il pourra alors se produire une certaine uniformisation des processus de recherche parallèles. Des comportements de recherche trop semblables pourraient impliquer un manque de diversité et une convergence prématurée de l'algorithme. Par conséquent, il est important de trouver un équilibre entre l'influence de l'information partagée et celle des paramètres locaux. Cela revient non seulement à déterminer la nature de l'information à partager, mais également le moment où elle est accessible par les processus de recherche. Ces conditions d'accès peuvent être déterminées de différentes façons. Elles peuvent être prédéfinies et fixées au début de l'exécution ou varier dynamiquement selon l'évolution de l'algorithme. Un exemple simple de condition d'accès pourrait être l'accès à la mémoire partagée toutes les t itérations (Delisle (2002)).

2.5.3 Les processus entre lesquels l'information est partagée

Finalement, il faut déterminer entre quels processeurs se feront les échanges d'informations. Autrement dit, il faut élaborer les liens logiques qui connectent les processus entre eux et qui contrôlent la propagation de l'information partagée. Cette structure de communication entre les processus doit être définie explicitement lors de la conception de l'algorithme. Par exemple, tous les processeurs pourraient participer à l'échange en diffusant certaines informations à un point de synchronisation donné et reprendre ensuite leur exécution. Une autre stratégie pourrait faire en sorte que tous les processeurs puissent écrire certaines informations dans une mémoire partagée qui ne serait lue que par certains processus sélectionnés. Mentionnons que l'échange d'informations peut être synchrone ou asynchrone. S'il est synchrone, les processus impliqués doivent tous avoir atteint un certain point de synchronisation avant que l'échange ne soit effectué et que l'exécution ne soit reprise par ceux-ci. S'il est asynchrone, l'échange s'effectuera selon la logique interne de chaque processus. Apporter des réponses aux trois questions présentées ci-dessus permet de déterminer les principales règles d'interaction entre les processus et de définir le comportement coopératif de l'algorithme dans la résolution d'un problème (Delisle (2002)).

2.5.4 La co-évolution

Le concept de processus coopératifs prend diverses formes dans la littérature. Dans la famille des algorithmes évolutionnaires, dont font partie les algorithmes génétiques, la coopération peut prendre la forme d'un phénomène de co-évolution. Ce dernier représente l'évolution simultanée de plusieurs entités (par exemple des individus ou des populations) qui cherchent à s'adapter à leur environnement tout en interagissant entre elles (Koza (1991)). Un exemple d'algorithme co-évolutif est l'algorithme génétique parallèle en îles où plusieurs populations indépendantes (une par île/processeur) coopèrent en échangeant des individus qui migrent périodiquement d'une île à une autre (Calégari (1999)). Le choix des individus appelés à

migrer peut se faire de différentes façons. Par exemple, il peut être fait de façon probabiliste parmi les individus de la population ou en sélectionnant le meilleur individu de chaque population pour remplacer le pire d'une autre population (Cantù-Paz (2000)). La co-évolution peut impliquer la compétition plutôt que la coopération. Le principe sous-jacent est que les entités peuvent lutter entre elles afin d'assurer leur subsistance dans l'environnement où elles évoluent. Juille et Pollack (1996) montrent qu'un modèle de co-évolution compétitive entre les individus dans un algorithme génétique peut apporter une certaine diversité dans une population, aider à prévenir la convergence prématurée de l'algorithme et permettre l'émergence de nouveaux comportements menant à de meilleures solutions (Delisle (2002)).

Dans ce qui suit, nous allons montrer l'intérêt et les objectifs des plates-formes logicielles. En d'autres termes, nous identifions les principaux critères à considérer dans la conception de plates-formes logicielles, afin d'en assurer l'exploitation et la réutilisation maximale de code. Nous proposerons également un état de l'art de celles dédiées aux métaheuristiques parallèles.

2.6 Plate-formes pour les métaheuristiques parallèles

2.6.1 Les approches de conception des métaheuristiques parallèles

Une plate-forme peut être vue comme un ensemble d'outils facilitant le développement d'applications. Elle fournit un modèle ou un squelette assez générique pour être réutilisé lors de la conception de différentes applications. Ainsi, chacune des plates-formes présentées dans cette section est fondée sur un modèle permettant de décrire les différents composants des métaheuristiques séquentielles/parallèles (Cahon (2005)).

Il existe trois principales approches de réutilisation et d'exploitation des métaheuristiques parallèles existantes : *à partir de rien, seule la réutilisation du code et des modèles de conception.* La réutilisabilité de composants logiciels se définit par leur capacité à être exploités au sein de différentes applications sans effort significatif de la part du développeur (Fink *et al.* (1998)).

L'approche « *À partir de rien*[2] » est l'apparente simplicité des algorithmes mis en œuvre, le développeur est ainsi tenté de coder lui-même ces derniers. Une telle démarche n'est pas souhaitable puisqu'elle requiert du temps et de l'énergie, elle est sujette à erreurs et la maintenance d'un tel code est reconnue difficile (Melab (2005)).

L'approche « *Seule réutilisation du code* » consiste à réutiliser un code dit de « tierce-partie », sous la forme de programmes indépendants et libres, ou de librairies. Dans le premier cas, il sera nécessaire à l'utilisateur d'examiner attentivement le code et de réécrire certaines sections spécifiques au problème traité. Il s'agit d'une tâche fastidieuse et sujette à erreurs. Les librairies constituent une solution alternative intéressante pour la réutilisation du code (Wall (1999)). Ces dernières s'avèrent bien souvent plus fiables, testées et documentées. La maintenance et l'efficacité sont grandement facilitées. Par ailleurs, l'avènement des langages orientés-objet tels que *Java* a conduit au développement de librairies, voire la réécriture de certaines librairies existantes, facilitant et encourageant ainsi la réutilisation. Cependant, les librairies permettent seulement la réutilisation de code, mais ne sont pas dédiées à faciliter la réutilisation de la conception (Melab (2005)).

[2]Francisation de l'anglais "From scratch"

L'objectif primordial de l'approche « *Réutilisation du code et de la conception* » est de pallier les limites de l'approche précédente en permettant à la fois la réutilisation du code et celle de la conception. En d'autres termes, elle s'attache à minimiser aux utilisateurs la quantité de code développé chaque fois qu'un nouveau problème d'optimisation est considéré. Dans un contexte de l'optimisation combinatoire, cette approche est fondée sur la séparation de la *partie invariante* des méthodes de résolution de la partie spécifique aux problèmes à traiter. La partie invariante, indépendante des problèmes, constitue un ensemble de composants pouvant être des modèles de conception (ou *design patterns*) (Gamma *et al.* (1995)). Ces composants sont implémentés et encapsulés au sein des plates-formes (Johnson et Foote (1988)).

Ainsi, une plate-forme peut être définie comme un ensemble de classes intégrant des modèles de solutions logicielles dédiées à une famille de problèmes apparentés (Fink *et al.* (1998)). Contrairement aux librairies, les plates-formes sont caractérisées par un mécanisme inverse de contrôle de l'interaction vis à vis du code de l'utilisateur selon le principe de Hollywood : *Ne nous appelez pas, nous vous appelons (Do not call us, we call you).* Ainsi, les plates-formes garantissent le contrôle de la partie invariante des algorithmes. L'utilisateur fournit seulement la partie spécifique à son problème. Par conséquent, l'approche plate-forme lui permet de passer plus de temps sur la modélisation de son problème que sur l'implémentation de la méthode de résolution appropriée. Dans la prochaine section, les critères d'évaluation d'une plate-forme seront présentés plus en details. Quelques uns de ces critères sont argumentés dans (MacDonald *et al.* (2002), Cahon (2005) et Melab (2005)) mais organisés différemment ici.

2.6.2 Les critères d'évaluation d'une plate-forme

Trois étapes constituent le « cycle de vie » d'une plate-forme logicielle : la conception, l'implémentation et son exploitation par l'utilisateur.

2.6.2.1 La conception

À l'étape de conception, l'objectif principal est de permettre une réutilisation aisée de la plate-forme. Pour cela, la conception doit satisfaire trois critères essentiels : (1) une séparation conceptuelle claire entre la solution logicielle et les problèmes ; (2) une conception globale facile à comprendre et à assimiler pour faciliter son exploitation, la hiérarchisation étant le moyen permettant de répondre à cet objectif ; (3) et enfin une conception de la plate-forme doit se montrer extensible (*scalable*) (Melab (2005), Cahon (2005)).

a) **La séparation**
La réutilisation maximale de la conception et du code nécessite une analyse approfondie du domaine d'application auquel la plate-forme est dédiée. Cette analyse doit permettre une séparation maximale entre la partie invariante de la partie variable ou adaptable. La partie constante est implémentée dans la plate-forme sous forme d'une collection de classes génériques pouvant être concrètes ou abstraites (appelées squelettes dans (Alba et Tomassini (2002))). La partie variable est spécifiée dans la plate-forme mais son implémentation est laissée à la charge de l'utilisateur. Roberts et Johnson (1996), recommandent l'utilisation de la composition au détriment de l'héritage pour mettre en œuvre cette séparation.

En effet, la réutilisation de classes entières s'avère souvent plus accessible que celle de méthodes individuelles. Cette dernière approche de réutilisation nécessite un examen plus détaillé des composants logiciels réutilisés (Melab (2005), Cahon (2005)).

b) **La hiérarchisation**
Une plate-forme est définie comme une hiérarchie de classes abstraites. La hiérarchie permet au programmeur le développement incrémental d'applications, et la visualisation des programmes à différents niveaux de détails. En outre, le remplacement d'une partie d'un programme par une autre est aisée, permettant une conception plus flexible et adaptable.

c) **L'extensibilité**
En considérant le critère d'extensibilité, on distingue deux catégories de plates-formes : les plate-formes dites « Boîte Noire » et celles dites « Boîte Blanche ». Les premières réutilisent les composants et se basent sur le couplage par composition et paramétrage statiques, sans se préoccuper de la manière dont ces derniers accomplissent leur rôle (Johnson et Foote (1988)). Au contraire, les plates-formes dites « Boîte Blanche » requièrent d'abord une compréhension des classes, une maîtrise à l'utilisation, pour ensuite mettre en œuvre, par le biais de l'héritage, des sous-classes valides. Par conséquent, elles autorisent une extensibilité meilleure. De nouvelles plates-formes sont généralement identifiées comme « Boîte Blanche » à la phase de démarrage. À l'exploitation, elles vont être progressivement adaptées et réutilisées par le mécanisme de spécialisation. Lorsque la partie variante est stabilisée, elles deviennent des plate-formes « Boîte Noire »(Fink *et al.* (1998)).

2.6.2.2 L'implémentation et le déploiement

L'objectif essentiel à ce niveau est de constituer une implémentation stable des composants logiciels afin d'encourager l'utilisation de la plate-forme. Une implémentation stable et réutilisable requiert la fiabilité du code, l'ouverture, le choix d'un langage de programmation adéquat, la portabilité, et enfin la performance et la robustesse (Melab (2005), Cahon (2005)).

a) **La fiabilité**
L'implémentation doit garantir la fiabilité des applications. Même si la plate-forme est bien testée, elle demeure bien souvent des bogues[3]. Cependant, la correction d'une telle erreur de programmation est effectuée une seule fois, préservant les autres utilisateurs. Par conséquent, les plates-formes permettent une meilleure fiabilité.

b) **L'ouverture du code**[4]
L'ouverture d'une plate-forme est l'opportunité pour l'utilisateur d'accéder aux détails de l'implémentation ou aspects de bas niveaux. On notera que ce critère est antagoniste avec le précédent. D'un autre côté, cela rend plus abordable l'objectif d'extensibilité décrit précédemment (Melab (2005), Cahon (2005)).

c) **Langage de programmation**
Il est important d'opter pour un langage de programmation facilitant la réutilisation et la séparation conceptuelle évoquées précédemment. Les langages objets, tels que *Java* ou *C++*, entrent dans cette catégorie. Le langage de programmation doit être également

[3]Francisation de l'anglais "bug"
[4]Francisation de l'anglais "Open Source"

adapté à la mise en œuvre d'applications parallèles et distribuées, et des mécanismes intrinsèques (multi-programmation, synchronisations, exceptions, etc.). Pour illustration, le langage *Java* intègre beaucoup de concepts intéressants, mais souffre d'une performance moindre. Au contraire, le langage *C++* s'avère plus performant à l'exécution, mais supporte moins les technologies appliquées au parallélisme (Melab (2005), Cahon (2005)).

d) **La portabilité**
La portabilité constitue un critère déterminant pour une large exploitation de la plate-forme. Elle doit être déployable sur des plates-formes matérielles variées (Réseaux de PCs, machines massivement parallèles, Clusters, machines parallèles à mémoires partagées, etc.) ainsi qu'aux systèmes d'exploitation associés (Linux, Windows, etc.). Par conséquent, il est important de développer à partir d'un langage portable tel que *Java*, de librairies standards pour la concurrence et la synchronisation(e.g. *Pthreads*[5]) et pour la communication (e.g. *MPI*[6], *PVM*[7], etc.).

e) **La performance et la robustesse**
Étant donné que les applications pour l'optimisation sont très gourmandes en temps CPU, le critère de performance à l'exécution est primordial. Le déploiement de métaheuristiques parallèles doit être robuste afin de garantir une fiabilité des résultats (Melab (2005), Cahon (2005)).

2.6.2.3 L'exploitation par utilisateur

a) **La facilité d'utilisation**
La plate-forme doit apparaître facile à l'utilisation pour le développeur. Cependant, cela est fortement dépendant du niveau de la maîtrise technique du programmeur. Ce critère est également lié au choix réalisé à la phase d'implémentation tel que le langage de programmation. En outre, la qualité d'implémentation, en termes de fiabilité et de lisibilité du code, peut se montrer d'une grande influence sur sa facilité d'utilisation. La plate-forme doit en effet être suffisamment testée afin d'identifier et de supprimer le plus de bogues potentiels (Melab (2005), Cahon (2005)).

b) **La transparence du parallélisme et de la distribution**
Le parallélisme et la distribution impliquent des mécanismes difficiles tels que la concurrence ou la gestion des synchronisations, la communication, etc. Ces technologies doivent être exploitées au niveau utilisateur de façon transparente. Il est important pour l'utilisateur de se focaliser sur le problème à résoudre, et non sur ces aspects.

c) **Les supports d'utilisation**
Le succès à l'exploitation d'une plate-forme dépend aussi des différents supports : une interface graphique, de visualisation, d'instrumentation, de suivi de l'exécution, d'une documentation en ligne, etc.

Nous présentons dans la section suivante deux catégories de plates-formes : *les plates-formes pour les AEs*[8] *parallèles et les plates-formes pour les méthodes de recherche locale parallèles.*

[5] Posix Threads
[6] Message Passing Interface
[7] Parallel Virtual Machine
[8] Algorithmes Évolutionnaires

2.6.3 État de l'art des plate-formes de métaheuristiques parallèles

De nombreuses plates-formes pour l'optimisation combinatoire ont été développées. Cependant, peu d'entre elles permettent d'atteindre tous les objectifs énoncés précédemment. Dans le tableau 2.1, nous présentons une étude comparative et non exhaustive de quatorze plates-formes logicielles de métaheuristiques, les plus représentatives à notre connaissance. Toutes ces plates-formes sont orientées-objet, excepté *CSM-PGA* et sont dédiées à la réutilisation de code et de conception.

Les différentes plates-formes sont classées selon les critères suivants : la classe des métaheuristiques supportées (Algorithmes Évolutionnaires (AE) et/ou métaheuristiques à base de Recherche Locale (RL)), les modèles parallèles implémentés (Topologies), le langage de programmation utilisé pour son développement, et enfin le support sous-jacent utilisé pour la mise en œuvre de la concurrence et/ou des communications (conc./com.). Le (-) dans le tableau indique que l'information n'est pas renseignée.

Plate-forme	Ref.	R.L	A.E	Topologies	Langage	Conc/Comm.
EasyLocal++	Di Gaspero et Schaerf (2000)	✓	×	-	C++	-
MAFRA	Krasnogor et Smith (2000)	✓	✓	-	Java	-
J-DEAL	Costa *et al.* (2001)	×	✓	M./E.	Java	Sockets TCP/IP
EO	Keijzer *et al.* (2001)	×	✓	-	C++	-
Localizer++	Michel et Van Hentenryck (2001)	✓	×	-	C++	-
DREAM	Arenas *et al.* (2002)	×	✓	Coop. insulaire	Java	Threads, Sockets TCP/IP
CoP	Adamidis et Petridis (2002)	✓	✓	Cellulaire	C++	MPI, PVM,
D-BEAGLE	Gagne *et al.* (2003)	×	✓	M./E.	C++	Sockets TCP/IP
ParadisEO	Cahon *et al.* (2004)	✓	✓	Toutes	C++	MPI, PVM, PThreads
MAGMA	Milano et Roli (2004)	✓	✓	-	C++	MPI, PVM, PThreads
ECJ	Luke *et al.* (2007)	×	✓	Coop. insulaire	Java	Threads, Sockets TCP/IP
MALLBA	Alba et MALLBA-Group (2007)	✓	✓	Toutes	C++	Netstream, MPI
CSM-PGA	Belkhelladi *et al.* (2007)	×	✓	Centralisée	C	Sockets TCP/IP
MAF-DISTA	Belkhelladi *et al.* (2010a)	×	✓	Coop. insulaire	Java	Java events, Java RMI

TAB. 2.1 – Taxonomie des plate-formes de métaheuristiques parallèles

Selon les deux premiers critères, les plate-formes de métaheuristiques sont divisées en trois catégories : celles dédiées aux AE, aux RL et celles supportant ces deux classes.

Easylocal++(An object-oriented plate-forme for flexible design of local search algorithms)

EasyLocal++ est une plate-forme C++ pour le développement de métaheuristiques à base de recherche locale (grimpeurs, récuit simulé et recherche tabou), conçue par Di Gaspero et Schaerf (2000). L'architecture pertinente de composants adoptée à la conception favorise une grande réutilisation de modèles et de code. Ainsi, différents schémas d'hybridation sont autorisés à la mise en œuvre (Melab (2005), Cahon (2005)).

MAFRA(A java Memetic Algorithms FRAmework)

MAFRA (Krasnogor et Smith (2000)) est une plate-forme Java pour le développement d'algorithmes mémétiques (i.e., algorithmes évolutionnaires hybridés avec une recherche locale se substituant généralement à la mutation). La conception de modèles co-évolutionnaires d'AEs est aussi supportée, mais le déploiement à l'exécution reste séquentiel. Les perspectives concernent deux aspects : l'intégration de fonctionnalités pour l'optimisation multi-objectif et l'implémentation d'une interface utilisateur de haut niveau pour la conception aisée et interactive d'algorithmes évolutionnaires (Melab (2005), Cahon (2005)).

J-DEAL (the Java Distributed Evolutionary Algorithms Library)

Cette plate-forme, due à Costa *et al.* (2001), est une autre plate-forme portable, développée en Java, pour la conception de méthodes évolutionnaires. Elle est libre de tout paradigme (algorithmes génétiques, stratégies évolutionnistes, etc.). *J-DEAL* permet le déploiement sur architectures distribuées de modèles reposant sur le paradigme « Maître/Esclave ». La réutilisation ou l'extension des composants existants est encouragée. On notera aussi la présence d'une documentation complète et d'un tutoriel. Ces deux caractéristiques montrent que les critères de flexibilité et de simplicité à l'utilisation ont été atteints. *J-DEAL* intègre des outils pour la génération automatique de statistiques à l'exécution et le repli/dépli de l'état d'exécution au niveau applicatif de façon transparente (Melab (2005), Cahon (2005)).

EO(Evolving Objects)

Conçue par Keijzer *et al.* (2001), *EO* est une plate-forme libre et portable en C++ pour le développement d'applications à base d'algorithmes évolutionnaires. Elle autorise différents paradigmes (algorithmes génétiques, programmation évolutionnaire, stratégies évolutionnistes, programmation génétique), et supporte la modélisation de problèmes multi-objectifs. Elle intègre différentes représentation prédéfinies. *EO* couvre une grande diversité d'opérateurs génétiques génériques (sélection, remplacement, etc.). Aucun support pour le parallélisme ou l'hybridation n'est assuré. La plate-forme *EO* assure différents services tels que la sauvegarde/reprise au niveau applicatif, la production de statistiques, l'analyse de paramètres en ligne caractérisant les aspects génériques du processus de l'évolution, etc. Enfin, notons que *EO* offre un tutoriel utilisateur assurant une maîtrise progressive de la plate-forme (Melab (2005), Cahon (2005)).

Localizer++(An open library for local search)

Développée par Michel et Van Hentenryck (2001), *Localizer++* constitue une autre plate-forme C++ pour l'implémentation de métaheuristiques à base de solution unique très proche de *EasyLocal++*. Comme cette dernière, elle permet la mise en œuvre de différentes formes d'hybridation mais ne supporte pas le déploiement de modèles coopératifs sur environnements parallèles ou distribués. *Localizer++* permet également la modélisation de plusieurs opérateurs de voisinage et leur intégration au sein d'une même métaheuristique (Melab (2005), Cahon (2005)).

DREAM(A framework for distributed evolutionary algorithms)

DREAM (Arenas *et al.* (2002)), désigne une infrastructure logicielle de type « pair à pair » et considère un « pool » virtuel de ressources de calcul distribuées, où les différentes étape d'un AE sont exécutés de manière automatique et transparente. La portabilité est assurée par le langage Java. *DREAM* est couplée avec l'interface utilisateur *GUIDE*, le langage de spécification *EASEA* et la plate-forme évolutionnaire *JEO* (cf. figure 2.3). L'originalité de *DREAM* réside dans une facilité d'utilisation variable à l'exploitation. Il est en effet possible pour l'utilisateur d'opter pour une interface de spécification adaptée à son niveau de maîtrise technique. L'interface graphique *GUIDE* constitue la plus simple d'entre elles, et permet l'exploitation de la plate-forme par des non-informaticiens. Différents panneaux sont respectivement dédiés à la modélisation du génome et des opérateurs de variation (à partir de structures prédéfinies), à la caractérisation du processus d'évolution (sélection, remplacement, etc.) et enfin à la définition de paramètres liés au déploiement sur environnements distribués. Seul le modèle coopératif insulaire est intégré dans la plate-forme. Aussi, *GUIDE* permet de définir les stratégies de sélection, remplacement, et topologies mis en œuvre (Melab (2005), Cahon (2005)).

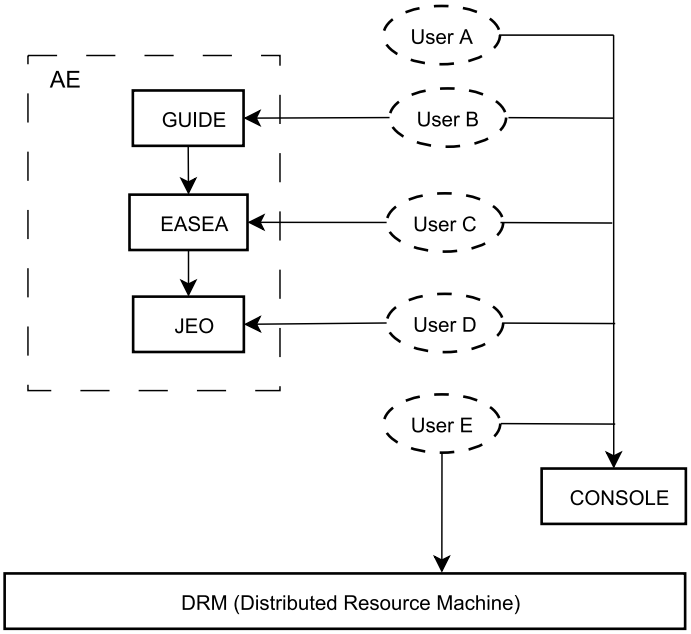


FIG. 2.3 – Architecture DREAM (Arenas *et al.* (2002))

CoP (Cooperating Populations)

CoP est une plate-forme pour la modélisation et l'implémentation séquentielle et parallèle des algorithmes évolutionnaires en tant qu'un ensemble de populations en coopération (Adamidis et Petridis (2002)). Il s'agit d'un modèle récursif multi-niveaux permettant la mise en œuvre de tout type d'AE en tout genre de matériel, *CoP* peut être décrite comme suit :

Le premier niveau est constitué de populations d'individus vivant dans des îlots qui sont en mesure de communiquer les uns avec les autres. À un deuxième niveau, les populations des îlots peuvent constituer une entité plus importante avec l'îlot considéré comme un individu. À un niveau plus élevé, ces grandes entités peuvent également former une population, chacune considérée comme un individu. La procédure peut continuer jusqu'au niveau requis. Le niveau initial de *CoP* (CoP^0) peut décrire à la fois les implémentations cellulaire et séquentielle de l'AE mis en œuvre, de sorte que chaque individu de (CoP^0) correspond à un gène. *CoPDEB* (Adamidis et Petridis (1996)) est au niveau 1, c'est-à-dire, (CoP^1). Dans (CoP^1), chaque individu est une (CoP^0) et correspond à un îlot. Chaque îlot a sa propre population, qui évolue en parallèle avec les populations des autres îlots (Melab (2005), Cahon (2005)).

D-BEAGLE (Distributed beagle : An environment for parallel and distributed evolutionary computations)

D-BEAGLE est conçue par Gagne *et al.* (2003). Elle se définit comme une version distribuée de la plate-forme portable C++ *Open BAEAGLE*. Elle couvre deux paradigmes que sont les algorithmes génétiques et la programmation génétique. Les modèles de conception de ces méthodes évolutionnaires reposent sur STL[9]. Le parallélisme à l'exécution est restreint au paradigme « Maître/Esclave », la communication repose sur le mécanisme des sockets TCP/IP. Le modèle coopératif insulaire avec migrations synchrones est supporté, mais son déploiement s'effectue sur un seul processeur. *Distributed BEAGLE* assure la régulation de charge et la sauvegarde au niveau applicatif de manière transparente (Melab (2005), Cahon (2005)).

MAGMA (Multi-AGent Metaheuristics Architecture)

MAGMA est une plate-forme introduite par Milano et Roli (2004). Les objectifs recherchés sont de pouvoir comparer les métaheuristiques existantes afin de faciliter leur hybridation et leur implantation. Dans cette approche, une métaheuristique est un système multi-agents composé de quatre niveaux. Chaque niveau dispose d'agents spécialisés dans le traitement d'une tâche générique. La figure 2.4 représente les différents niveaux identifiés dans l'approche *MAGMA*.

Le niveau 0 est composé d'agents chargés de générer de nouvelles solutions pour les niveaux supérieurs. Le niveau 1 permet d'améliorer des solutions en utilisant des procédures telles que la recherche locale. Les agents du niveau 2 servent à contrôler le rapport diversification/intensification. Enfin, si plusieurs stratégies de recherche coexistent déjà dans le niveau 2, les agents de niveau 3 permettent de les coordonner.

L'approche *MAGMA* permet d'identifier différentes tâches au sein d'une métaheuristique. La notion d'agent est utilisée ici pour décrire un élément capable de réaliser une tâche particulière du système. On retrouve dans *MAGMA* les différentes procédures de l'approche AMP. De plus, l'approche agent prend en compte l'aspect de distribution de l'exécution (Meignan (2008)).

[9]Standard Template Library

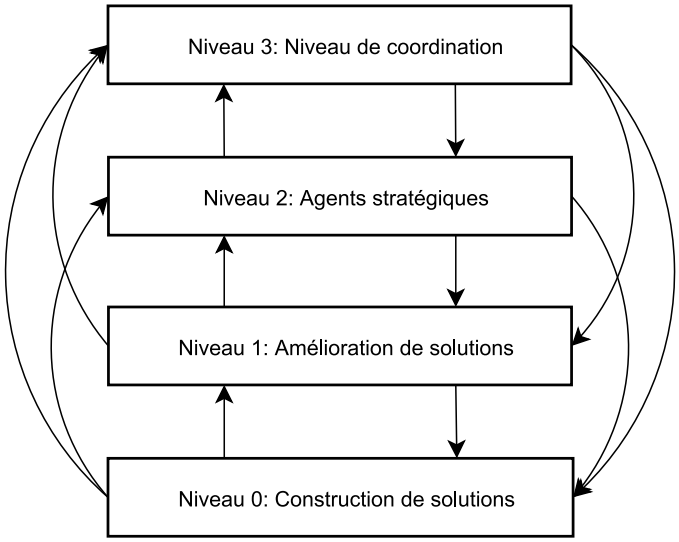


FIG. 2.4 – Architecture MAGMA (Milano et Roli (2004))

ParadisEO

ParadisEO est une plate-forme Open Source développée par l'INRIA de Lille. C'est une extension de la plate-forme EO (Evolving Objects) initialement conçue pour l'implantation d'algorithmes évolutionnaires (Cahon *et al.* (2004)). Les principaux objectifs de cette plate-forme sont tout d'abord, de pouvoir réutiliser du code pour faciliter et accélérer le développement de métaheuristiques. Ensuite, l'architecture de la plate-forme a été conçue de manière à permettre le développement de nouvelles métaheuristiques ou de métaheuristiques hybrides. Enfin, *ParadisEO* fournit des solutions pour l'exécution distribuée des algorithmes. L'architecture de *ParadisEO* est composée de trois couches : *Solvers, Runners* et *Helpers*. Les *Helpers* forment la « couche basse » de la librairie qui fournit les composants de base des métaheuristiques tels que les opérateurs évolutionnaires (mutation, croisement, sélection) ou les méthodes de déplacement dans un voisinage. Les *Runners* correspondent aux différentes métaheuristiques définies à partir des *Helpers*. Les *Solvers* correspondent aux processus chargés d'exécuter et éventuellement de coordonner la ou les métaheuristiques (Meignan (2008)).

ECJ (A java-based evolutionary computation research system)

ECJ est développée par Luke *et al.* (2007). Elle constitue une plate-forme Java très riche et portable couvrant les méthodes évolutionnaires et la programmation génétique. Elle apparaît très flexible et simple à l'utilisation puisque la majorité des classes et composants sont identifiés dynamiquement à l'exécution après l'analyse d'un fichier de paramètres. *ECJ* intègre des représentations prédéfinies mais autorise la définition de nouvelles structures. La plate-forme supporte la mise en œuvre du modèle coopératif insulaire d'AEs et son déploiement sur architectures parallèles ou distribuées par une utilisation des technologies « TCP/IP sockets » et « Java Threads ». D'autres aspects couvrent l'optimisation multi-objectif, la sauvegarde/reprise de l'exécution, des outils pour le traçage, etc (Melab (2005), Cahon (2005)).

Mallba(a software library to design efficient optimisation algorithms)

Mallba est développée par Alba *et al.* (2007). Elle intègre un ensemble de « squelettes » logiciels implémentant la plupart des métaheuristiques (algorithmes génétiques, recuit simulé, recherche Tabou, etc.) mais aussi les méthodes exactes (Branch & Bound, Programmation dynamique, Divide & Conquer). Les squelettes définissent les patrons d'algorithmes génériques. La performance à l'exécution est assurée par de nombreux modèles parallèles facilement déployables sur environnements d'exécution distribués de différentes échelles (LAN/WAN). *Mallba* repose sur l'interface de communication *ah hoc* Netstream pour la communication. *Mallba* est riche pour ces nombreuses formes d'hybridation : l'hybridation de métaheuristiques ou celle de méthodes approchées avec d'autres méthodes exactes (Melab (2005), Cahon (2005)).

MAF-DISTA (Mobile Agent Framework for Distributed Algorithms)

MAF-DISTA est une plate-forme conçue par Belkhelladi *et al.* (2010a) permettant de modéliser et d'implémenter les algorithmes à population sous forme de plusieurs agents de recherche. Dans une instance de *MAF-DISTA*, chaque algorithme de recherche est encapsulé dans un agent, une entité autonome qui doit garder la connaissance du problème d'optimisation à résoudre. Les agents sont coordonnés par un ensemble de règles stipulant les aspects topologiques et de communication (migration), ces règles pouvant être fixées a priori ou être changées lors de l'exécution par l'intermédiaire d'une entité de coordination dite « méta-agent ». Les détails de cette plate-forme sont abordés dans le chapitre 5.

MAF-DISTA se distingue des plates-formes existantes en introduisant les concepts permettant de traiter la distribution et l'adaptation dans les métaheuristiques parallèles à base de population. Dans cette plate-forme, les composants (agents) ne sont pas stationnaires et sont capables de migrer d'une machine à une autre pour une meilleure exploitation des ressources inutilisées.

2.7 Bilan et conclusion

Après avoir présenté quelques définitions du concept des métaheuristiques séquentielles et parallèles, nous nous sommes attachés à identifier les différents critères d'évaluation qui nous semblent pertinents au regard des techniques de conception des outils logiciels réutilisables. En effet, la première ambition d'une plate-forme est son exploitation par un maximum d'usagers, que ce soit informaticiens spécialistes réseaux ou non. Une étude comparative et une analyse critique de quelques plates-formes logicielles dédiées aux métaheuristiques parallèles ont été réalisées en tenant compte de quelques-uns de ces critères, parmi les plus importants ou aptes à être évalués. De cette étude, il ressort que la grande majorité des plates-formes se focalisent généralement sur une classe de métaheuristiques (Algorithmes évolutionnaires, métaheuristique à base de recherche locale) excepté la plate-forme *ParadisEO* qui se focalise sur les deux classes. On notera que peu de modèles de parallélisation, guidé principalement par des critères de simplicité et de portabilité, autorisent vraisemblablement un déploiement à large-échelle.

Dans cette thèse, nous proposons une plate-forme de métaheuristiques parallèles à base

de population permettant de remédier à certains inconvénients soulevés précédemment. Cette plate-forme devra donc permettre d'envisager les métaheuristiques parallèles sous l'angle de leur mise en œuvre et de leur modélisation. Elle devra intégrer des concepts liés à l'adaptation et à l'auto-adaptation dynamique de la stratégie de recherche. Elle devra également apporter des éléments méthodologiques permettant de faire le lien entre l'analyse, la modélisation et l'implantation. Nous pensons que l'application du paradigme multi-agents à la conception de métaheuristiques parallèles peut apporter une réponse à ces attentes.

Chapitre 3

Les systèmes d'agents mobiles et les métaheuristiques parallèles

Sommaire

3.1 Introduction

L'évolution des réseaux à grande échelle a permis la naissance d'un grand nombre de nouvelles applications qui se développent autour de ce type de réseau (Arcangeli *et al.* (2002)) : commerce électronique, recherche d'informations sur le web, plates-formes pour le calcul réparti, etc. Ces applications sont formées par des entités réparties et leur bonne fonctionnalité nécessite communication et échanges entre ces entités. Le modèle client/serveur où les échanges se font par des appels distants à travers le réseau est le modèle le plus utilisé. Dans ce modèle, seul le client représente une application au sens propre du terme et le rôle du serveur est de répondre aux demandes des clients. Le serveur construit ses réponses indépendamment du client, ainsi une partie des données envoyées est inutile augmentant ainsi le trafic sur le réseau. De plus ce modèle exige une connexion permanente entre le client et le serveur, ce qui n'est pas le cas des terminaux mobiles qui sont exposés à la perte de la connexion. Le concept d'agent mobile apparaît dans ce contexte comme une solution facilitant la mise en œuvre d'applications dynamiquement adaptables et il offre un cadre générique pour le développement d'applications réparties sur des réseaux de grande taille. L'envoi du code sur le serveur permet d'adapter les services distants aux exigences du client et de ne lui envoyer que les informations utiles.

Un agent mobile est un programme qui s'exécute dans un environnement. Les services nécessaires à l'agent mobile, qui sont fournis par l'environnement, doivent exister avec de bonnes propriétés. Ainsi, nous détaillons dans ce chapitre les principaux services qu'un environnement d'exécution doit fournir à ces agents. Nous abordons la création de l'agent, sa migration entre les différentes machines, sa localisation, sa protection, la protection de son site d'accueil et la communication entre les agents. Nous traitons le cycle de vie d'un agent mobile de sa création jusqu'à sa suspension et les différents points d'entrées qui vont aider le programmeur dans le contrôle de l'agent. Dans un souci de compatibilité entre les environnements d'exécution des agents mobiles, des efforts sont faits dans le domaine de la standardisation afin de faciliter l'interopérabilité : nous serons ainsi amenés à évoquer la standardisation d'intergiciels agent. Ensuite, nous mettons en évidence l'intérêt de l'approche d'agents mobiles utilisée dans le cadre des métaheuristiques parallèles. Enfin, nous concluons en défendant l'idée que la distribution dans les métaheuristiques parallèles peut être encouragée par l'adoption de l'approche d'agents mobiles (El-Falou (2006)).

3.2 Notions de base sur les systèmes multi-agents

3.2.1 Définition de la notion d'agent et de système multi-agent

Parmi les nombreuses définitions du terme « agent » certaines semblent faire l'objet d'un consensus au sein de la communauté multi-agent. Pour définir un agent, Ferber (1995) relève certaines propriétés clés s'appliquant aussi bien aux systèmes naturels qu'artificiels.

On appelle agent une entité physique ou virtuelle qui :

- est capable d'agir dans un environnement,
- peut communiquer directement avec d'autres agents,
- est mue par un ensemble de tendances (sous la forme d'objectifs individuels ou d'une fonction de satisfaction, voire de survie, qu'elle cherche à optimiser),

- possède des ressources propres,
- est capable de percevoir (mais de manière limitée) son environnement,
- ne dispose que d'une représentation partielle de cet environnement (et éventuellement aucune),
- possède des compétences et offre des services,
- peut éventuellement se reproduire,

Cette définition insiste sur la capacité d'agir des agents, et non pas seulement de raisonner. L'action est d'après Ferber (1995), un concept fondamental pour les systèmes multi-agents. Il repose sur le fait que les agents accomplissent des actions qui vont modifier l'environnement et donc leurs prises de décision futures. Une autre propriété essentielle des agents est l'autonomie, les agents ne sont pas dirigés par un utilisateur ou un autre agent, mais par un ensemble de tendances qui leur sont propres. Ces dernières peuvent prendre la forme de buts individuels à satisfaire ou de fonctions de satisfaction ou de survie que l'agent cherche à optimiser. Cette autonomie n'est pas seulement comportementale, mais porte aussi sur les ressources (énergie, CPU, quantité de mémoire, accès à certaines sources d'informations, etc.). Paradoxalement, ces ressources sont à la fois ce qui rend l'agent dépendant de son environnement, mais aussi, en étant capable de gérer ces ressources, ce qui lui donne une certaine indépendance vis-à-vis de lui (Meignan (2008)).

Le schéma de la figure 3.1 présente la structure d'un système multi-agents, selon la définition 1 donnée par Ferber (1995).

Définition 1 *On appelle système multi-agents (ou SMA), un système composé des éléments suivants :*

- *Un environnements E, c'est-à-dire un espace disposant généralement d'une métrique.*
- *Un ensemble d'objets O. Ces objets sont situés, c'est-à-dire que, pour tout objet, il est possible, à un moment donné, d'associer une position dans E. Ces objets sont passifs, c'est-à-dire qu'ils peuvent être perçus, créés, détruits et modifiés par les agents.*
- *Un ensemble A d'agents, qui sont des objets particuliers (A est contenu dans O), lesquels représentent les entités actives du système.*
- *Un ensemble R de relations qui unissent des objets (et donc des agents) entre eux.*
- *Un ensemble d'opérateurs Op. permettant aux agents de A de percevoir, produire, consommer, transformer et manipuler des objets de O.*
- *Des opérateurs chargés de représenter l'application de ces opérations et la réaction du monde à cette tentative de modification, que l'on pourrait appeler « les lois de l'univers ».*

Selon Jennings *et al.* (1998), le terme système multi-agents correspond à la définition donnée par Durfee *et al.* (1989) comme étant « un réseau faiblement couplé de solveurs de problèmes qui travaillent ensemble pour résoudre des problèmes qui sont au-delà des capacités individuelles ou des connaissances de chaque solveur de problème ». La principale caractéristique d'un SMA[1] réside dans son exécution par essence décentralisée : chaque agent agit de manière autonome mais aucun ne contrôle totalement la dynamique du système. Le comportement collectif d'un tel système résulte des interactions à l'exécution des différents agents qui permettent de construire des réponses plus complexes que leurs comportements

[1] Système Multi-agents

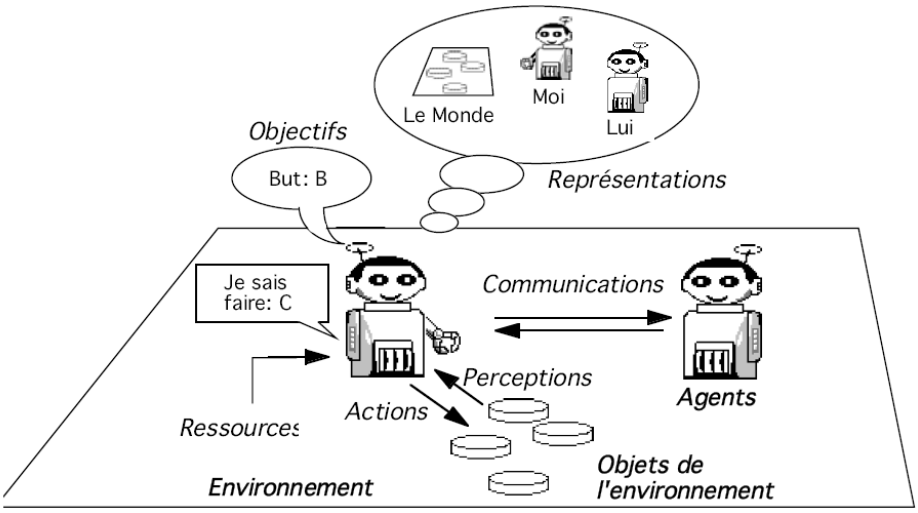


FIG. 3.1 – Structure d'un système multi-agents (Ferber (1995))

individuels (Thomas (2005)). Un système multi-agents est ainsi caractérisé par plusieurs niveaux d'observation :

- un niveau local ou individuel dans lequel les agents sont décrits et prennent leurs décisions. C'est à ce niveau que le comportement d'un agent par rapport au reste du système peut être observé ;
- un niveau global ou collectif dans lequel il est possible d'observer la dynamique du système et l'avancement de la résolution du problème.

3.3 Concepts de base des agents mobiles

Un processus informatique est constitué par une séquence d'instructions (code) qui s'exécute sur une machine et qui utilise les ressources de cette machine. Dans le contexte de la mobilité, il faut envisager la possibilité d'interrompre l'exécution d'un processus afin de la poursuivre sur une autre machine. Le processus est représenté, en plus de son code, par son état d'exécution. Par état d'exécution nous entendons la valeur du compteur ordinal, celle de la pile d'exécution et les différents registres du processeur. Selon le schéma d'exécution de ce code et la localisation des différentes entités du système (ressource, code et état d'exécution), nous pouvons distinguer les notions d'évaluation distante, de code à la demande et d'agents mobiles (El-Falou (2006)).

3.3.1 Évaluation distante

Dans une interaction par évaluation distante (cf. figure 3.2), un client envoie un code à un site distant. Le site récepteur utilise ses ressources pour exécuter le programme envoyé. Éventuellement, une interaction additionnelle délivre ensuite les résultats au client. Dans ce schéma, seul le code est transmis au serveur et l'exécution du code se déroule uniquement sur ce dernier. Les interactions avec les imprimantes *PostScript* sont réalisées par ce modèle. Le

code d'une requête *SQL* émis vers un serveur de base de données représente un autre exemple d'évaluation distante (El-Falou (2006).

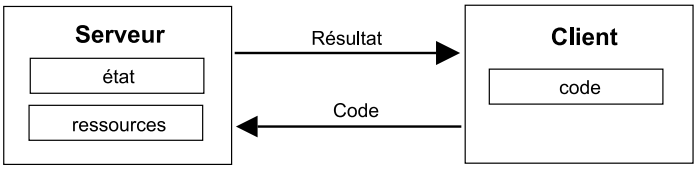


FIG. 3.2 – Évaluation distante

3.3.2 Code à la demande

Dans ce schéma, le processus client interagit avec un site distant afin de récupérer un savoir faire qui sera exécuté sur la machine cliente. Ainsi, le client télécharge le code nécessaire à la réalisation d'un service. Le rôle du site distant est de fournir le code du service qui sera exécuté sur le site client (voir figure 3.3). Les *Applets* Java reposent sur cette technologie du code mobile, il s'agit d'un programme chargé à partir d'une page Web pour être exécuté sur la machine du client (El-Falou (2006)).

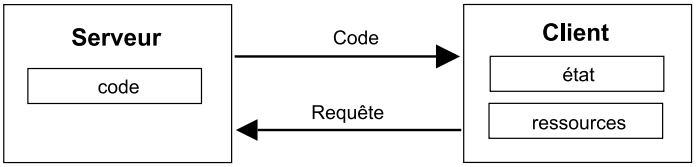


FIG. 3.3 – Code à la demande

3.3.3 Agents mobiles

Par comparaison avec les deux schémas précédents, l'exécution du processus débute sur le site client. Dans la mesure où le client a besoin d'interagir avec le serveur, ce même processus (code, état d'exécution et données) se déplace à travers le réseau pour continuer son exécution et pour interagir localement avec les ressources du serveur (cf. figure 3.4). Après exécution, l'agent mobile retourne éventuellement vers son client afin de lui fournir les résultats de son exécution (El-Falou (2006)).

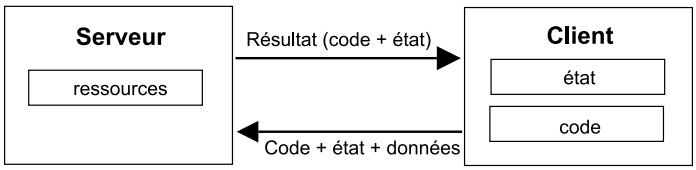


FIG. 3.4 – Agent mobile

Dans ce schéma, le savoir-faire appartient au client, l'exécution du code est initiée côté client et continuée sur les différentes machines visitées.

3.4 Définition d'un agent mobile

Le terme agent est très répandu en informatique. La notion d'agent a donné lieu à de très nombreuses définitions, notamment dans le domaine de l'intelligence artificielle. Un agent (Beale et Wood (1994)) est une entité logicielle (informatique ou électronique) plongée dans un environnement dans lequel elle est capable d'agir. Il possède un comportement autonome relié à ses observations, à sa connaissance et aux interactions qu'il entretient avec l'environnement et avec les autres agents. Nous définissons un agent mobile comme étant un agent répondant à cette définition et capable de se déplacer d'un site à un autre dans un réseau pour accomplir la tâche du client (El-Falou (2006)). De cette définition découlent les points essentiels suivants :

- ☞ **Comportement :** Un agent est pourvu d'un programme qui lui dicte la tâche à accomplir. On dit qu'il agit par délégation pour le client. Un agent peut ainsi venir avec une compétence particulière sur une machine hôte.
- ☞ **Autonomie :** Un agent agit indépendamment du client. Il décide lui-même là où il va se déplacer et ce qu'il doit y faire, en fonction du comportement qui lui a été donné. Cela implique que l'on ne peut pas toujours prévoir l'itinéraire des agents.
- ☞ **Mémoire :** Un agent mobile dispose d'une capacité de mémorisation lui permettant de récolter des informations sur les sites visités. Les informations mémorisées seront livrées par l'agent après son retour au client.
- ☞ **Environnement :** L'environnement dans lequel l'agent évolue est constitué par :
  - les machines qui vont accueillir l'agent lors de son exécution ; ainsi l'agent utilise les ressources (unité de calcul, mémoire, etc.) disponibles sur la machine afin d'accomplir la tâche demandée par le client,
  - les liens réseau que l'agent utilise pour se déplacer entre les differents sites,
  - les autres agents fixes et mobiles avec qui l'agent interagit afin de réaliser son but.

3.5 Environnement d'exécution d'agents mobiles

Un environnement d'exécution d'agents mobiles fournit une interface de programmation et un environnement d'exécution offrant des primitives pour créer, lancer et exécuter un agent mobile. Cet environnement est constitué d'un ensemble de programmes statiques, appelés places, s'exécutant sur les sites du système susceptibles d'accueillir des agents. Les environnements d'exécution d'agents mobiles offrent plusieurs services de base permettant l'exécution d'un agent mobile sur un site. Ces services sont les suivants : création et migration d'un agent mobile, communication et échange de messages entre les agents mobiles, accès aux ressources locales par les agents mobiles et traçage des agents mobiles en cours d'exécution. Il faut ajouter à ces services, fournis par l'environnement d'exécution d'agents mobiles, des mécanismes mettant en œuvre des notions de qualité de service requise par les applications comme par exemple la sécurité ou la tolérance aux pannes.

Les domaines d'application des agents mobiles sont multiples. Tout ce qui est fait en mode client/serveur pourra être fait avec des agents mobiles. Combien même une partie

des applications ne nécessitent pas ce nouveau modèle, certaines se révéleront plus efficaces en l'utilisant tel que le e-commerce, la recherche d'informations, le calcul parallèle et les télécommunications (Jha et Iyer (2001), Van Thanh (2001), El-Falou (2006)).

3.6 Services requis pour l'exécution d'agents mobiles

Dans cette section, nous allons décrire les fonctions nécessaires pour concevoir des applications réparties à base d'agents mobiles. Après une présentation de la structure d'un agent mobile, nous allons présenter en détail les services suivants :

- création d'un agent,
- transfert d'un agent,
- désignation d'un agent, c'est-à-dire offrir un nom globalement unique à un agent,
- localisation d'un agent,
- communication entre agents,
- exécution d'un agent,
- protection d'un agent.

3.6.1 Structure d'un agent mobile

Un agent mobile est un programme pouvant se comporter d'une façon autonome au profit de son client. Chaque agent a son flot d'exécution pour pouvoir prendre l'initiative d'exécuter des tâches. Un agent doit avoir la capacité de migrer d'une machine à une autre sur le réseau. Pour parler de la migration d'un agent, il est important de connaître les éléments à transférer avec l'agent. Une instance d'agent mobile comporte trois parties (Tanenbaum (2007)) :

- un code exécutable qui représente la suite d'instructions définissant le comportement statique de l'agent mobile,
- un contexte d'exécution qui reflète l'état d'exécution courant de l'agent mobile (valeurs des registres, pile d'exécution),
- les données ou les ressources utilisées par l'agent mobile ; ces ressources sont divisées en deux parties (Fuggetta *et al.* (1998)) :
  i. les ressources transférables qui sont les valeurs des attributs de l'agent et qui lui donnent un état global,
  ii. les ressources non transférables qui constituent l'environnement d'exécution fourni par le système (par exemple les fichiers ouverts, les connexions sockets, les registres, etc.) et les matériels physiques utilisés par l'agent (imprimante, écran, etc.). Dans la section 3.6.3, nous allons montrer comment l'agent migre avec son code, son contexte d'exécution et ses données.

3.6.2 Création d'agents mobiles

Dans la mesure où ces agents peuvent se déplacer, on peut se poser la question au moment de leur création, de l'endroit où l'on souhaite qu'ils démarrent leurs activités. Une fois créé, l'agent peut être actif, c'est-à-dire que l'exécution de son code est déclenchée. La création/lancement d'un agent peut prendre plusieurs formes :

- création locale,

– création et exécution à la demande (*code on demand*),
– création et exécution à distance (*remote execution*).

La création de l'agent n'est pas directement liée à la mobilité, mais à des mécanismes sur lesquels s'appuie la mobilité. À sa création, un nom globalement unique (cf. 3.6.4) doit être attribué à l'agent ce qui va permettre de localiser l'agent et de communiquer avec lui.

3.6.3 Migration d'un agent

La migration permet le transfert d'un agent en cours d'exécution d'un site à un autre à travers le réseau. Nous allons prendre le cas d'un agent qui désire migrer entre deux machines avec la possibilité de recevoir des messages pendant son déplacement (cf. figure 3.5). Cette migration comporte les étapes suivantes (Jain *et al.* (2000)) :

1. la sérialisation du contexte de l'agent et la production d'un message incluant le contexte sérialisé de l'agent et son code ;
2. le processus de l'agent étant suspendu sur sa machine d'origine, toutes les communications avec d'autres agents sont donc suspendues et l'agent est détruit ;
3. l'envoi du contexte et du code de l'agent vers la machine de destination ;
4. l'état de l'agent migrant est restauré sur la machine de destination, un nouveau processus léger est créé pour la poursuite de son exécution ;
5. les communications en cours sont redirigées vers la machine de destination, l'agent n'étant pas encore active, ces messages seront traités après sa réactivation ;
6. réactivation de l'agent sur la machine de destination, dès que la partie de son contexte nécessaire à son exécution est reçue ; la machine de destination s'occupe des liens dynamiques entre l'agent et son code. À partir de ce moment, la migration prend fin. Le processus sur la machine d'origine peut être définitivement effacé (El-Falou (2006)).

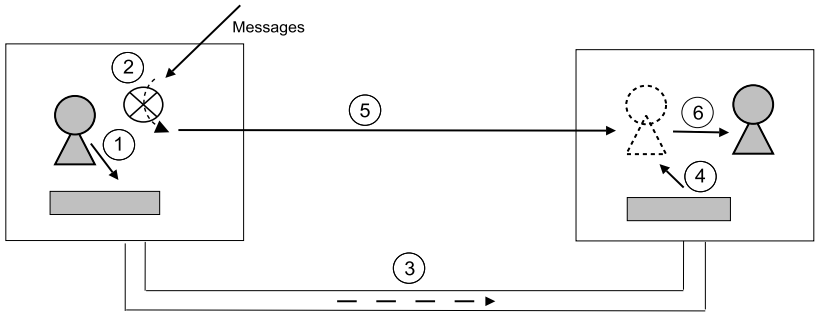


FIG. 3.5 – Migration d'un agent mobile

Les différentes stratégies de migration d'agents dépendent de la manière dont sont traitées ces diverses étapes et les données transférées avec l'agent lors de sa migration. Deux types de migrations ont été proposées dans les plates-formes existantes :

1. La migration forte permet à un agent de se déplacer quelque soit l'état d'exécution dans lequel il se trouve. Dans ce type de migration l'agent se déplace avec son code, son contexte d'exécution et ses données. L'agent reprend son exécution après la migration

exactement là où elle était avant son déplacement. La migration forte nécessite un mécanisme de capture instantanée de l'état d'exécution de l'agent. Elle peut être proactive ou réactive. Dans la migration proactive, la destination de l'agent est déterminée par l'agent lui-même. Dans la migration réactive, la migration de l'agent est dictée par une partie ayant une relation avec l'agent mobile.

2. La migration faible ne fait que transférer avec l'agent son code et ses données. Sur le site de destination, l'agent redémarre son exécution depuis le début en appelant la méthode qui représente le point d'entrée de l'exécution de l'agent, et le contexte d'exécution de l'agent est réinitialisé. Pour que l'agent se branche sur une instruction particulière de son code après sa migration, le programmeur doit inclure dans l'état de l'agent des moments privilégiés (explicites) dans le code de l'agent (point d'arrêt) pour pouvoir le relancer.

La migration forte, bien que beaucoup plus exigeante à implanter (Dillenseger *et al.* (2002)) que la migration faible (Bäumer et Magedanz (1999)), n'en est pas moins indispensable pour toutes les applications pour lesquelles les notions de fiabilité et de tolérance aux pannes sont primordiales.

3.6.4 Service de nommage

Dans un système à agents mobiles, les agents ont besoin de communiquer entre eux, ce qui nécessite de les nommer. Le nom donné à l'agent doit être globalement unique. Dans la plupart des systèmes à agents mobiles ce nom est construit à partir du nom de la machine (ou adresse IP) où l'agent s'exécute, d'un numéro de port et d'un identificateur localement unique (El-Falou (2006)).

3.6.5 Service de localisation

Afin de communiquer avec un agent mobile, il est nécessaire de le retrouver. La mobilité de l'agent introduit des contraintes sur les communications qui n'étaient pas présentées avec les systèmes classiques où les objets ne pouvaient pas changer de lieu d'exécution. En particulier, il faut que des entités mobiles puissent communiquer indépendamment de leur localisation et de leur mobilité (Alouf *et al.* (2002)). La fiabilité est un point très important car il faut assurer les communications à tout moment avec les objets mobiles. Pour ces raisons, un système à agents mobiles doit offrir un service de localisation des agents à travers un serveur de noms. Ce serveur de noms contient la localisation courante de l'agent ou bien suffisamment d'informations pour le localiser (El-Falou (2006)).

Les principaux schémas de localisation sont les suivants (Milojičić *et al.* (1999)) :

- **mise à jour sur le site d'origine :** le serveur de noms situé sur la machine d'origine des agents est mis à jour à chaque migration d'un agent. Ceci entraîne un nombre de communications avec le serveur d'origine égal au nombre de migrations de l'agent ($N_{com} = N_M$),
- **enregistrement :** les agents enregistrent leur mouvement auprès d'un serveur de noms défini et centralisé qui n'est pas situé sur la machine d'origine. Le nombre de communications est égal au nombre de mouvements de l'agent augmenté d'une communication de la machine d'origine vers le serveur de noms ($N_{com} = N_M + 1$),

- **recherche :** le serveur de noms cherche la localisation d'un agent selon un itinéraire bien défini. Cela suppose que l'itinéraire d'un agent doit être connu à l'avance. Le nombre de communications peut varier de 1 jusqu'au nombre maximum d'itinéraires ($1 \leqslant N_{com} \leqslant max(N_M)$),
- **poursuite :** suivre un lien de poursuite permet de localiser un agent. Le nombre de communications est au maximum égal au nombre de nœuds visités par l'agent ($1 \leqslant N_{com} \leqslant max(N_M)$).

3.6.6 Communications entre les agents

La communication entre deux entités peut se faire de deux façons ; la plus intuitive consiste à avoir un mécanisme permettant une communication directe entre les objets, ce qui correspond à la communication synchrone ou asynchrone entre les deux entités. Une deuxième manière est d'avoir des mécanismes de communication indirecte. Dans ce cas un objet voulant communiquer envoie son message à un objet tiers qui se charge de le faire suivre au destinataire.

Dans un système à agents mobiles, il faut différencier deux types de communication selon que la communication intervient entre deux agents ou entre un agent et un groupe d'agents. Les moyens de communication entre les agents sont multiples, nous pouvons citer la communication par message, session, tableau blanc, rendez-vous, par événement distribué ou local (El-Falou (2006)).

3.6.7 Exécution d'un agent

Un système à agents mobiles doit offrir la possibilité d'exécuter un agent sur les machines qui vont accueillir ce dernier, ainsi l'agent utilise les ressources disponibles sur la machine afin d'accomplir la tâche demandée par le client. Sur la machine d'accueil, les agents mobiles peuvent être amenés à effectuer des entrées/sorties, accéder à l'interface utilisateur, au système de gestion de fichiers, aux applications externes et à l'interface réseau. Les machines qui forment notre système d'exécution n'auront pas toutes le même type de système d'exploitation. Par conséquent, maintenir l'indépendance vis à vis du système d'exploitation d'un environnement d'agents mobiles tout en autorisant en même temps l'accès aux ressources est un problème important à surmonter dans les environnements d'exécution d'agents mobiles. C'est pour cette raison que la plupart des environnements d'agents mobiles utilisent *Java* comme langage d'implémentation du code mobile (Gomoluch et Schroeder (2001)). *Java* est un langage orienté objet développé par *Sun*[2] et présenté officiellement en 1995. Les applications codées en Java sont compilées en *bytecode* et exécutées sur une machine virtuelle, la Java Virtuel Machine (JVM). Le succès du langage Java se traduit, entre autre, par le fait qu'on peut raisonnablement considérer que la machine virtuelle Java constitue aujourd'hui un environnement d'exécution « universel », car elle est disponible sur toutes les machines d'un système réparti. Cette propriété en fait une plate-forme de développement de choix pour les applications mobiles ou distribuées car le programmeur n'a pas à gérer différents langages ou différentes versions d'un programme suivant les systèmes sur lesquels il va s'exécuter. En plus

[2] http://www.sun.com/

de cette propriété liée à la portabilité, Java propose toute une série de services qui permettent de construire des applications distribuées mobiles. On trouve notamment :

- la sérialisation/désérialisation d'un objet afin de le transmettre à travers le réseau,
- la communication entre deux objets situés sur des machines distantes (RMI, socket, etc.),
- le chargement dynamique de code à partir d'un site distant.

L'utilisation de Java permet de résoudre le problème lié à l'exécution de l'agent sur les différentes machines du système réparti. Mais avec la diversité des plates-formes disponibles, nous pouvons nous interroger sur la migration entre les différentes plates-formes. Chaque plate-forme possède sa propre façon pour définir le comportement des agents, ce qui pose un problème de compatibilité lors de la migration d'un agent entre les différentes plates-formes. Pour surmonter ce problème, Overeinder *et al.* (2004) proposent un langage générique, pour la description du comportement de l'agent, basé sur le langage *XML*[3]. Une projection de cette description permet à une plate-forme d'agents mobiles de reconstruire l'agent avec son propre langage.

L'exécution d'un agent mobile sur une machine hôte pose le problème du piratage des sites visités. Ce problème, non spécifique aux agents mobiles, est symétrique pour ces derniers. En effet, un agent mobile est exposé au problème du piratage par les logiciels qui fonctionnent sur les sites qu'il visite. Dans la section suivante, nous allons détailler les problèmes liés à la sécurité dans un environnement d'agents mobiles (El-Falou (2006)).

3.6.8 Sécurité de l'agent

Garantir la sécurité dans les environnements d'agents mobiles est important du fait de la nature même du modèle d'exécution des agents et du fait des interactions des agents mobiles avec plusieurs systèmes et ressources. Un système à agents mobiles doit offrir des mécanismes permettant une exécution sécurisée des agents au sein du système. Trois types de problèmes de sécurité ont été identifiés pour les environnements d'exécution d'agents mobiles (El-Falou (2006)) :

- **La sécurité du site d'accueil contre un agent.** Au cours de son exécution, un agent mobile peut avoir accès aux ressources du site sur lequel il est situé. Un agent malveillant peut donc profiter des accès aux ressources locales pour propager des virus (Chess *et al.* (1996)), des worms[4] ou des *chevaux de Troie*. Il peut masquer sa véritable identité et lancer une tâche lourde ce qui va entraîner un déni de service (Bellavista *et al.* (2001)). La machine qui va accueillir l'agent mobile doit être amenée à détecter le mauvais déroulement de l'exécution de l'agent. Dans (Galtier *et al.* (2001)), les auteurs présentent une méthode qui permet à l'agent de définir ses besoins d'une façon indépendante de la machine sur laquelle il s'exécute. Le site d'accueil peut détecter les abus commis par l'agent permettant ainsi de le contrôler. Une approche classique pour protéger les sites des agents malveillants est de limiter les accès des agents aux ressources locales du site (Hantz et Guyennet (2006)). Dans cette approche, le comportement de l'agent est écrit sous la forme d'un langage interprété. L'agent est exécuté dans un environnement qui à son tour s'exécute au dessus du site. Le contrôle de l'agent est ainsi

[3] eXtensible Markup Language
[4] Vers informatique

fait dans l'environnement d'exécution (Carvalho *et al.* (2004)). Une seconde approche consiste à ne laisser s'exécuter que les agents authentifiés (Ametller *et al.* (2004)). Une signature est alors attribuée à l'agent permettant au site d'accueil de l'authentifier avant son exécution.

- **La sécurité d'un agent contre le site d'accueil.** La protection des agents mobiles contre des sites hostiles est spécifique au domaine. Les agents sont à protéger à deux niveaux : la protection du code de l'agent (changement du comportement) et la modification de l'état de l'agent. Plusieurs approches ont été proposées : l'approche organisationnelle permet, seulement aux sites dignes de confiance, de gérer des systèmes d'agents mobiles ; l'approche de détection de manipulation offre des mécanismes fondés sur le traçage permettant de détecter les manipulations de données effectuées sur un agent (Díaz *et al.* (2001)) ; enfin l'approche de protection par boîte noire. La cryptographie mobile est une étape vers l'approche de protection par boîte noire. La spécification de l'agent est convertie en code exécutable et en un ensemble de données encryptées (Claessens *et al.* (2003)). Le cryptage de données empêche un éventuel attaquant de lire ou de modifier les données. Dans (Benachenhou et Pierre (2006)), les auteurs présentent une nouvelle approche qui consiste à exécuter un agent clone sur un site sûr afin de surveiller l'exécution du vrai agent. Cette solution engendre une augmentation dans le trafic réseau (échange entre l'agent et son clone).
- **La sécurité d'un agent contre un autre agent.** L'agent doit être protégé contre une éventuelle attaque par un autre agent (Hohlfeld *et al.* (2002)) situé sur le site hôte ou sur un autre site. Un agent doit avoir sa propre politique de sécurité qui peut être soit assurée par l'agent lui même, soit par le site d'accueil.

3.6.9 Tolérance aux pannes

Le modèle d'exécution de l'agent mobile implique son interaction avec plusieurs sites ce qui expose l'agent à une éventualité de disparition à cause de la défaillance ou de la déconnexion soudaine et imprévue d'un site sur lequel il s'exécute. La disparition d'un agent entraîne un dysfonctionnement de l'application basée sur ce dernier. Une application distribuée sûre doit pouvoir continuer de fonctionner en cas de défaillance d'une partie du système. Pour certains types d'applications, il est essentiel que les environnements d'exécution d'agents mobiles offrent des mécanismes de tolérance aux fautes.

Dans une architecture de services, les défaillances de sites peuvent conduire à un comportement défaillant d'un service et donc le rendre inutilisable par le client. Plusieurs types de défaillances sont à considérer :

- Arrêt (crash, panne) : un serveur ne rend pas de résultats suite à des invocations répétées.
- Omission : un serveur omet de répondre à son client.
- Temporelle : la réponse du serveur est fonctionnellement correcte mais n'est pas arrivée dans un intervalle de temps donné.
- Valeur : le serveur rend des résultats incorrects.

Pour une application distribuée basée sur le concept d'agents mobiles, la migration des agents engendre d'autres types de défaillance :

- Un agent peut disparaître après avoir visité plusieurs sites (machines). Si aucune pré-

caution n'est prise, les résultats de son exécution sur ces sites peuvent être perdus (Shao et Zhou (2006)).
- Il est important de détecter la disparition d'un agent pour en informer la machine qui l'a lancé.
- Un problème d'atomicité pour l'exécution globale d'un agent qui se déroule successivement sur plusieurs sites. Ce qui implique qu'il faut garantir que l'agent reprend son exécution exactement là où elle en était avant son déplacement et ceci avec le même contexte d'exécution. C'est au programmeur de garantir cette atomicité dans une migration faible (cf. la section 3.5).

Pour faire face à la défaillance, les environnements d'agents mobiles offrent un mécanisme de point de reprise (Bäumer et Magedanz (1999)). Les points de reprise sont conservés sur disque, support supposé fiable. Ils peuvent ainsi être utilisés ultérieurement pour restaurer l'agent en cas de défaillance. Cependant, lors de la restauration de l'agent il faut veiller à ne pas avoir deux agents actifs en même temps (problème rencontré en cas de défaillance liée à la perte de la connexion avec le site d'accueil de l'agent par exemple).

3.6.10 Traçabilité

Une fois lancé, l'agent mobile devient une entité autonome qui s'exécute d'une façon asynchrone. Pour la machine qui l'a lancé, il est utile de disposer d'informations sur l'exécution de l'agent, sur son état et sur la machine sur laquelle il se situe. Le mécanisme de traçabilité devient un moyen pour suivre et contrôler les déplacements des agents. Les agents mobiles peuvent disposer de toute l'autonomie leur permettant de choisir leurs itinéraires (espace) et les moments (temps) pour les explorer. Par exemple, il peut être judicieux pour un agent de changer l'ordre des sites qu'il doit visiter à cause d'une indisponibilité passagère d'un lien réseau ; il peut aussi décider de ne pas remettre en question l'ordre de son itinéraire, mais simplement d'attendre que la perturbation du lien réseau s'estompe. On peut aussi avoir des agents dont les missions sont soit exploratoires (robots sur Internet), soit font référence à des buts ou à des besoins (charge aux agents de trouver les bons interlocuteurs pour mener à bien leurs tâches). En d'autres termes, même si les agents peuvent être programmés pour donner des informations sur ce qu'ils font, il peut être utile de disposer de mécanismes de traçabilité qui leur sont externes, pour garder un aperçu de ce qu'ils font.

3.6.11 Cycle de vie et contrôle de l'agent

Un environnement d'exécution pour agents mobiles doit offrir à l'utilisateur la possibilité de contrôler les activités de l'agent. Depuis sa création jusqu'à sa terminaison, un agent peut passer par plusieurs étapes (cycle de vie d'un agent). Un langage de programmation d'agents mobiles doit offrir au programmeur des points d'entrée qui vont lui permettre de contrôler l'activité de son agent. Un agent passe par une partie ou la totalité des étapes suivantes :

1. **Création et initialisation :** Lors de sa création, l'agent peut être initialisé avec des informations nécessaires à son exécution telles qu'un itinéraire, des préférences de son utilisateur, etc. Par ailleurs, un agent est initialisé par le système avec des informations nécessaires à son interaction avec son environnement, comme par exemple l'identité de son utilisateur (El-Falou (2006)).

2. **Migration :** L'agent se déplace entre les machines du système. Le but de cette migration est souvent motivé par une coopération en local avec des agents fixes ou d'autres agents mobiles s'exécutant sur le même site ou pour exploiter des ressources supplémentaires (puissance de calcul, mémoire). Avant de reprendre ses activités, l'agent aura besoin des informations sur le nouveau site (El-Falou (2006)).
3. **Activation et désactivation :** Dans une application basée sur les agents mobiles, un agent se désactive en suspendant son exécution afin d'être sauvegardé sur un support non volatile (mécanisme de sérialisation). L'agent mobile est donc gelé. L'activation est l'opération inverse par laquelle l'agent est restauré afin de continuer son exécution (El-Falou (2006)).
4. **Terminaison :** Une fois que sa tâche est réalisée, l'agent se termine et son processus d'exécution est tué. Avant sa terminaison l'agent a besoin de livrer un bilan à son utilisateur.

Souvent, un environnement d'agents mobiles offre aux développeurs des méthodes relatives au cycle de vie d'un agent. Ces méthodes sont appelées des « call-back ». Un agent doit être informé chaque fois qu'un événement du cycle de vie commence, réussi ou échoue (voir table 3.1). Cela a pour but non seulement de réagir aux événements en question mais aussi d'être capable de refuser une création, une migration ou une réinitialisation après un déplacement.

Activité de l'agent	**Call-backs**
Création	afterBirth
Désactivation	beforeSuspend
Activation	afterResume
Migration	beforeMove, afterMove, afterMoveFailed
Terminaison	beforeDeath

Tab. 3.1 – Call-back dans un environnement d'agents mobiles

3.7 Le calcul distribué et les agents mobiles

Un programme parallèle peut être vu comme un ensemble de tâches réparties sur le réseau ; ces tâches utilisent le réseau pour communiquer et pour échanger des données. L'un des problèmes principaux de l'implantation d'un algorithme sur une architecture distribuée est celui de la répartition de charges sur les machines disponibles. Ainsi, le temps global d'exécution d'un algorithme parallèle peut être dégradé par une mauvaise utilisation des ressources (capacité de calcul et réseau de communication). La distribution des tâches sur les nœuds s'intéresse à ce problème en mettant en place un mécanisme permettant de décider du placement d'une tâche avant son exécution et de la migration de celle-ci au cours de l'exécution de l'application. Le placement des tâches peut s'effectuer à deux moments différents : soit au lancement de l'application (de manière statique), soit lors de l'exécution (de manière dynamique). Dans le cas du placement dynamique, les stratégies doivent disposer d'informations en provenance des différents nœuds (informations sur la charge de machines) pour ensuite prendre des décisions de localisation des tâches.

La distribution dynamique cherche à éviter qu'un nœud ne soit inactif alors que des tâches restent en attente sur d'autres nœuds. De manière plus forte, on cherche à ce que la charge de calcul soit bien équilibrée et que les ressources de calcul soient exploitées de façon optimale sur l'ensemble du système. Il est en effet assez intuitif de penser que le système est utilisé au mieux lorsque chaque ressource a la même charge de travail. Le placement dynamique des tâches nécessite la migration de ces dernières entre les différentes machines du réseau. Cette migration devient opérationnelle grâce à l'utilisation de la technologie des agents mobiles. Dans une application distribuée, une tâche est représentée par un agent et la terminaison de celle ci correspond à la fin de la vie de l'agent. Les échanges d'informations entre les tâches correspondent à la communication entre les agents. Quant au déplacement d'une tâche, il correspond à la migration d'un agent à travers le réseau, il s'agit de ce fait, de représenter une tâche par un agent mobile (El-Falou (2006)).

3.8 L'approche agent et les métaheuristiques parallèles

Un premier aspect des métaheuristiques qui les rapprochent des systèmes multi-agents est la distribution du calcul. Celle-ci peut intervenir à différents niveaux d'une métaheuristique. Ainsi, dans (Crainic et Toulouse (2003)), les auteurs distinguent trois niveaux de distribution dans les métaheuristiques. Le premier niveau, ou « bas niveau » consiste à exécuter en parallèle des opérations indépendantes dans une même itération de la métaheuristique. Par exemple, il est parfois possible d'effectuer une évaluation parallèle des solutions, ou encore il peut être possible d'appliquer des opérateurs de voisinage, de mutation ou de croisement, sur différentes solutions en parallèle. Ce niveau de parallélisme a uniquement pour but de permettre d'améliorer le temps d'exécution sans pour autant modifier la qualité du résultat par rapport à une version séquentielle du même algorithme. Dans la suite, nous ne nous intéresserons pas à ce type de distribution car il semble peu adapté à une interprétation multi-agents. Le second niveau de distribution consiste à répartir la recherche dans l'espace des solutions sur plusieurs processus de recherche. Cette seconde stratégie de distribution du calcul est généralement implantée sous la forme d'une architecture maître/esclave. Le processus maître est chargé de partitionner l'espace de recherche et les processus esclaves explorent de manière indépendante la partie qui leur a été assignée. Le dernier niveau de distribution correspond à la mise en concurrence de plusieurs processus de recherche heuristique. Un exemple répandu de ce type de distribution de l'exécution est l'approche des îlots pour les algorithmes évolutionnaires (Tanese (1989)). Cette approche de distribution consiste à former un ensemble de sous-populations séparées les unes des autres et n'interagissant entre elles que périodiquement au travers de migrations d'individus. Chaque sous-population est gérée par un algorithme évolutionnaire indépendant ayant éventuellement un paramétrage propre (Skolicki (2005)). Ce type de mise en œuvre distribuée permet, d'une part, d'améliorer l'exploration de l'espace de recherche, et d'autre part, de laisser ouverte la possibilité de mettre en concurrence différentes stratégies de recherche ou différents paramétrages de l'algorithme.

Le concept d'agent est parfois explicitement mentionné dans les métaheuristiques distribuées. C'est le cas dans la méthode *Co-search* (Bachelet et Talbi (2000)). L'architecture de *Co-Search* est donnée dans la figure 3.6. Le principe est d'utiliser plusieurs agents spécialisés pour faire évoluer un ensemble de solutions réunies dans une mémoire. Celle-ci se compose

d'une partie relative aux zones de l'espace de recherche explorées et d'une partie correspondant aux zones de recherche prometteuses. Trois agents utilisent et mettent à jour cette mémoire. Le premier agent est nommé *agent de diversification*. Il est chargé de la diversification et fournit de nouvelles solutions dans les zones non explorées de l'espace de recherche, en exploitant les informations concernant les zones explorées. Le second agent, *agent d'intensification*, a pour objectif de fournir des solutions prometteuses. Le dernier agent, *agent de recherche*, effectue une recherche à partir des solutions fournies par les deux agents précédents. L'heuristique développée dans (Bachelet et Talbi (2000)) utilise une recherche tabou pour l'*agent de recherche*, un algorithme génétique pour l'*agent de diversification* et un opérateur de perturbation pour l'*agent d'intensification*. L'objectif recherché dans *Co-search* est de faire coopérer plusieurs stratégies de recherche complémentaires (Meignan (2008)).

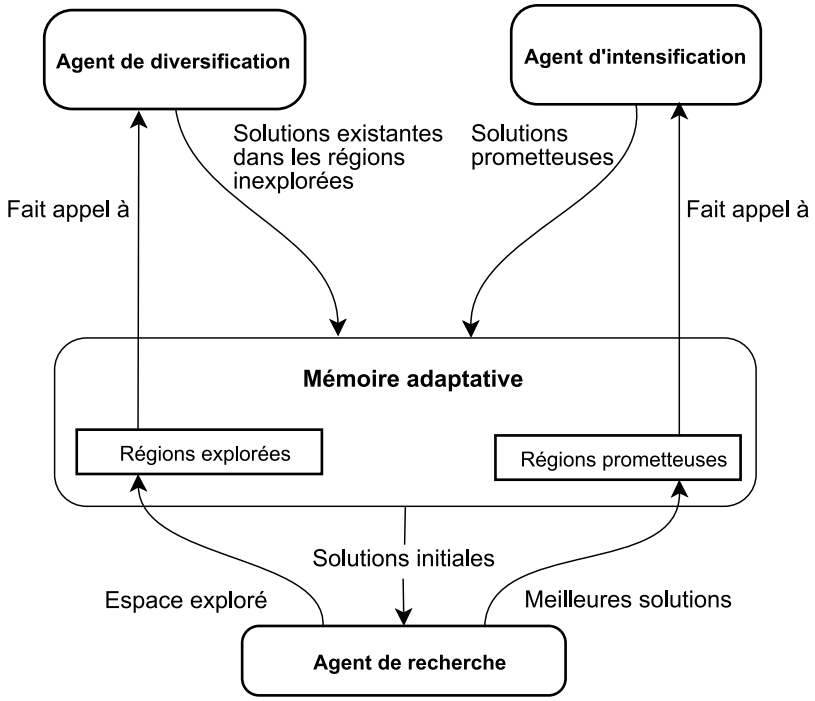


FIG. 3.6 – Principe de la métaheuristique Co-search (Bachelet et Talbi (2000))

3.9 Conclusion

Dans ce chapitre, nous avons présenté les différentes technologies utilisées pour la répartition et la communication entre les différentes entités d'une application répartie. Ainsi, nous avons situé la technologie d'agents mobiles par rapport aux différentes techniques présentées. Un agent mobile est une entité autonome qui se déplace d'une machine à l'autre sur le réseau, sans perdre son code ni son état. C'est l'environnement d'exécution qui se charge d'assurer cette fonctionnalité. Il permet la création et la migration d'un agent, la communication et l'échange de messages entre les agents mobiles, et il assure la sécurité de l'agent et de son site d'accueil.

À notre avis et d'après Meignan (2008), l'adoption du paradigme d'agents mobiles pour la conception de métaheuristiques séquentielles et parallèles repose essentiellement sur l'adoption d'une posture intentionnelle appliquée à des processus de recherche s'exécutant en parallèle et en interaction les uns avec les autres. C'est ce qui est sous-entendu lorsque l'on dit que le système est constitué d'agents autonomes possédant des buts et agissant collectivement en vue de la réalisation de ces buts.

En ce qui nous concerne, nous proposons d'approfondir l'application de la démarche orientée agent aux métaheuristiques parallèles en considérant cette perspective. Nous proposons de les analyser en adoptant certains outils méthodologiques généraux spécifiques à la conception de systèmes d'agents mobiles.

Partie 2 : Conception des plates-formes

Chapitre 4

Stratégies d'échange d'informations

Sommaire

4.1 Introduction

Classiquement, le parallélisme a été considéré comme un moyen d'augmenter la puissance de calcul en exécutant des tâches simultanément, c'est dans ce but qu'ont été développées des plates-formes SIMD[1], MIMD[2], réseaux ou fermes de stations de travail avec le sempiternel objectif de tirer parti du fait qu'exécuter des tâches en parallèle est plus rapide que les exécuter l'une après l'autre (Lee (2007)) .

Une perspective différente est cependant apparue depuis quelques années, moins technique, plus conceptuelle et que l'on peut relier à l'intelligence artificielle distribuée (IAD). L'objectif n'est plus d'accélérer les traitements ; mais d'avoir des activités concurrentes. L'implantation est reléguée au second ordre, elle peut être aussi bien parallèle que séquentielle. L'IAD simule des systèmes complexes composés de nombreux agents en interaction tels des colonies d'insectes dont s'inspirent les algorithmes en essaim (modèle des fourmis par exemple (Dorigo et Gambardella (1997))). Les réseaux d'automates cellulaires rentrent dans ce cadre de pensée également. Dans la même veine conceptuelle, quelques rares travaux en optimisation ont montré expérimentalement que la coopération d'activités permet d'obtenir des optima de meilleure qualité. Ainsi, Macready *et al.* (1995) montrent comment plusieurs activités concurrentes trouvent des états d'énergie plus bas dans des modèles de verre de spin ; Fonlupt *et al.* (1997), Belkhelladi *et al.* (2007), Belkhelladi *et al.* (2008), Lee (2007) montrent que des algorithmes génétiques coopérant par échange de solutions résolvent facilement un problème d'optimisation alors que chacun des algorithmes génétiques pris séparément en est incapable (la probabilité qu'il trouve l'optimum global est presque négligeable en pratique). Nous souhaitons ici poursuivre sur cette voie en nous attaquant à de nouvelles stratégies d'échange d'informations pour une meilleure coopération entre les entités de recherche.

Dans ce chapitre, nous présentons les différentes stratégies d'échange d'informations que nous mettons en œuvre. Trois stratégies d'échange d'informations ont été élaborées. La première et la seconde consistent en l'échange d'individus (solutions) entre les agents de recherche. Le concept de la première stratégie se base essentiellement sur l'adaptation des différentes stratégies d'échange d'informations au sein d'une colonie de fourmis parallèle au contexte des algorithmes génétiques. Le concept de la seconde stratégie s'inspire du phénomène d'immigration entre les pays du monde et est basé sur l'attribution des visas et des résidences. La troisième stratégie consiste en l'échange de jeux de paramètres entre les agents de recherche. Elle est considérée comme un *méta-algorithme* permettant d'adapter le paramétrage des agents de recherche.

4.2 La coopération dans les métaheuristiques parallèles

L'idée de base de la coopération est qu'un groupe d'agents coopérants engagés dans la résolution d'un problème peut résoudre ce dernier plus efficacement qu'un agent seul ou qu'un groupe d'agents opérant isolément les uns des autres. Plusieurs exemples tirés de la vie humaine et animale tendent à confirmer cette idée. D'un point de vue informatique, il peut être intéressant d'appliquer cette approche dans la conception d'algorithmes de résolution de

[1]Single Instruction on Multiple Data
[2]Multiple Instructions Multiple Data

problèmes complexes. L'idée de résoudre un problème à l'aide de plusieurs agents coopérants ou isolés qui opèrent en même temps présente un parallélisme naturel. Une architecture parallèle semble donc appropriée pour transposer cette idée dans un contexte informatique.

Un des principes fondamentaux des métaheuristiques est qu'elles mémorisent des solutions ou des caractéristiques des solutions visitées durant le processus de recherche passé et utilisent cette information pour les guider dans la recherche à venir (Taillard *et al.* (1998)). Un processus d'apprentissage est effectué durant l'exécution de l'algorithme et influence son comportement. Comme la qualité de l'algorithme est intimement liée à la pertinence de l'information conservée et utilisée, l'intégration de mécanismes de coopération peut créer de l'information nouvelle qui permettra à l'algorithme d'effectuer une meilleure recherche. La coopération permet donc une nouvelle forme d'apprentissage susceptible d'améliorer la performance des métaheuristiques.

Clearwater *et al.* (1992) définissent la coopération comme un processus impliquant un ensemble d'agents qui interagissent en se communiquant de l'information les uns aux autres durant la résolution d'un problème. La communication peut se faire, par exemple, par des diffusions entre les processeurs ou par l'accès à une source d'information centralisée. Selon ces mêmes auteurs, le comportement des agents impliqués dans un processus coopératif de recherche de solutions est déterminé par plusieurs éléments. Quelques uns de ceux-ci sont la nature du processus de recherche des agents (chaque agent peut avoir un processus semblable ou différent), la nature de l'information partagée, la façon dont elle est partagée et la structure organisationnelle du groupe d'agents.

Toulouse *et al.* (1995) ont cherché à identifier les principales questions qui devraient être posées lors de la conception de processus de recherche coopératifs et montrent que les questions reliées aux communications inter-agents sont de première importance. Selon ces auteurs, il faut déterminer principalement la nature de l'information à partager, le moment où elle devrait être partagée et entre quels processus elle devrait être partagée (Delisle (2002))
.

4.2.1 La nature des informations à partager

Quand vient le temps de déterminer quelles informations doivent être partagées, un facteur important à considérer est la quantité d'informations à partager parmi toutes celles qui ont été produites par les processus parallèles. Rendre disponible à tous les processus une grande quantité d'informations peut engendrer des coûts de communication et/ou d'accès à la mémoire très prohibitifs. De plus, il ne sera pas nécessairement intéressant pour les autres processus d'avoir accès à toute l'information disponible. Il est possible que certaines informations produites par un processus n'apportent rien et même nuisent aux autres. L'étude de la pertinence des données à partager est donc importante. Cependant, cette pertinence est généralement difficile à représenter dans un modèle quantitatif. La connaissance du problème et le jugement sont bien souvent les meilleures façons d'évaluer l'intérêt de partager une information plutôt qu'une autre (Delisle (2002)).

4.2.2 Le moment où l'information est partagée

Si les processeurs ont un accès sans restriction à l'information partagée, il peut se produire une détérioration de la performance de recherche de solutions. Par exemple, si l'information partagée fait trop souvent partie du processus de décision, il pourra alors se produire une certaine uniformisation des processus de recherche parallèles. Des comportements de recherche trop semblables pourraient impliquer un manque de diversité et une convergence prématurée de l'algorithme. Par conséquent, il est important de trouver un équilibre entre l'influence de l'information partagée et celle des paramètres locaux. Cela revient non seulement à déterminer la nature de l'information à partager, mais également le moment où elle est accessible par les processus de recherche. Ces conditions d'accès peuvent être déterminées de différentes façons. Elles peuvent être prédéfinies et fixées au début de l'exécution ou varier dynamiquement selon l'évolution de l'algorithme. Un exemple simple de condition d'accès pourrait être l'accès à la mémoire partagée toutes les t itérations (Delisle (2002)).

4.2.3 Les processus entre lesquels l'information est partagée

Il faut déterminer entre quels processeurs se feront les échanges d'informations. Autrement dit, il faut élaborer les liens logiques qui connectent les processus entre eux et qui contrôlent la propagation de l'information partagée. Cette structure de communication entre les processus doit être définie explicitement lors de la conception de l'algorithme. Par exemple, tous les processeurs pourraient participer à l'échange en diffusant certaines informations à un point de synchronisation donné et reprendre ensuite leur exécution. Une autre stratégie pourrait faire en sorte que tous les processeurs puissent écrire certaines informations dans une mémoire partagée qui ne serait lue que par certains processus sélectionnés. Mentionnons que l'échange d'informations peut être synchrone ou asynchrone. S'il est synchrone, les processus impliqués doivent tous avoir atteint un certain point de synchronisation avant que l'échange ne soit effectué et que l'exécution ne soit reprise par ceux-ci. S'il est asynchrone, l'échange s'effectuera selon la logique interne de chaque processus. Apporter des réponses aux trois questions présentées ci-haut permet de déterminer les principales règles d'interaction entre les processus et de définir le comportement coopératif de l'algorithme dans la résolution d'un problème (Delisle (2002)).

4.2.4 La co-évolution

Le concept de processus coopératifs prend diverses formes dans la littérature. Dans la famille des algorithmes évolutionnaires, dont font partie les algorithmes génétiques, la coopération peut prendre la forme d'un phénomène de co-évolution. Ce dernier représente l'évolution simultanée de plusieurs entités (par exemple des individus ou des populations) qui cherchent à s'adapter à leur environnement tout en interagissant entre elles (Koza (1991)). Un exemple d'algorithme co-évolutif est l'algorithme génétique parallèle en îles où plusieurs populations indépendantes (une par île/processeur) coopèrent en échangeant des individus qui migrent périodiquement d'une île à une autre (Calégari (1999)). Le choix des individus appelés à migrer peut se faire de différentes façons. Par exemple, il peut être fait de façon probabiliste parmi les individus de la population ou en sélectionnant le meilleur individu de chaque population pour remplacer le pire d'une autre population (Cantù-Paz (2000)). La

co-évolution peut également impliquer de la compétition plutôt que de la coopération. Le principe sous-jacent est que les entités peuvent lutter entre elles afin d'assurer leur subsistance dans l'environnement où elles évoluent. Juille et Pollack (1996) montrent qu'un modèle de co-évolution compétitive entre les individus dans un algorithme génétique peut apporter une certaine diversité dans une population, aider à prévenir la convergence prématurée de l'algorithme et permettre l'émergence de nouveaux comportements menant à de meilleures solutions (Delisle (2002)).

4.3 Stratégies d'échange d'informations

4.3.1 Les stratégies de Middendorf et *al.* (2000)

Middendorf *et al.* (2000) ont défini quatre stratégies d'échange d'informations. Dans cette section, nous avons adapté ces stratégies au contexte des algorithmes génétiques distribués. Les principes de fonctionnement de ces stratégies sont décrits comme suit :

4.3.1.1 Échange de la meilleure solution globale

Cette stratégie implique qu'à chaque étape d'échange d'informations, la meilleure solution trouvée par l'ensemble des processeurs sera diffusée à tous les processeurs où elle devient la meilleure solution locale. Il est important de spécifier que l'algorithme utilisé par Middendorf *et al.* (2000) effectue une mise à jour de la trace supplémentaire selon le principe d'élitisme, c'est-à-dire que la meilleure solution trouvée par l'algorithme depuis le début de son exécution est utilisée pour renforcer davantage la quantité de phéromone. La diffusion de la meilleure solution globale vient donc modifier les paramètres de cette mise à jour élitiste de la trace. Dans notre cas, nous utilisons la même idée d'élitisme pour renforcer les sous-populations, c'est-à-dire que les meilleures solutions trouvées par l'algorithme sont toujours préservées selon un taux d'élitisme (paramètre de l'algorithme). La Figure 4.1 illustre le fonctionnement de cette stratégie.

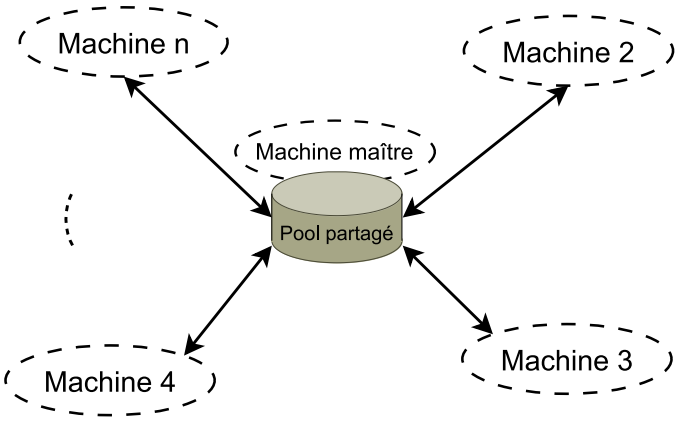


Fig. 4.1 – Diffusion de la meilleure solution globale

4.3.1.2 Échange circulaire des meilleures solutions locales

L'élaboration d'une stratégie d'échange circulaire des meilleures solutions locales nécessite qu'un voisinage virtuel soit établi entre les processeurs de façon à ce qu'ils forment un anneau. À chaque étape d'échange d'informations, chaque processeur envoie sa meilleure solution locale au processeur qui le succède dans l'anneau. L'information partagée est la même que dans le cas de la stratégie précédente, mais l'échange se fait entre des couples de processeurs plutôt que globalement. Un processeur fournit de l'information à son successeur et utilise celle de son prédécesseur.

Cette stratégie est mise en œuvre en réservant, un tableau de solutions T de dimension égale au nombre de processeurs. À chaque étape d'échange d'informations, le processeur i écrit à la case $T[i+1]$ la meilleure solution qu'il a trouvée localement. Comme chaque processeur écrit à un emplacement différent du tableau, cette procédure ne nécessite pas l'instauration d'une zone critique. Ensuite, après l'atteinte d'une barrière de synchronisation assurant que tous les processeurs ont écrit leur solution respective, chaque processeur i compare sa meilleure solution locale à celle présente à la case $T[i]$ du tableau (celle provenant de son prédécesseur). La Figure présente une représentation graphique du fonctionnement de cette stratégie.

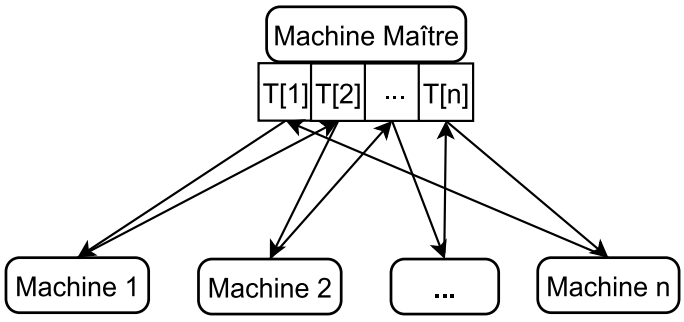


FIG. 4.2 – Échange circulaire des meilleures solutions locales

4.3.1.3 Échange circulaire de migrateurs

Comme dans le cas de la deuxième stratégie, les processeurs forment un anneau virtuel. À chaque étape d'échange, chaque processeur compare ses m meilleures solutions avec les m meilleures de son prédécesseur. La circulation des données se fait donc de la même façon que pour la stratégie précédente. Cependant, les meilleures solutions trouvées durant le cycle en cours sont échangées plutôt que les meilleures solutions locales. Les m meilleures solutions de ces deux groupes sont alors utilisées pour la mise à jour globale de la trace. Il est important de souligner ici l'importante différence entre l'échange de meilleures solutions et l'échange de migrateurs.

L'intégration de cette stratégie se fait également par la création d'un tableau de solutions. Ce tableau est noté T. Comme l'algorithme utilisé dans ce travail effectue la mise à jour de la sous-population à partir de la meilleure solution trouvée pendant le cycle présent, une seule solution *(m = 1)* sera échangée dans l'anneau, c'est-à-dire la meilleure solution trouvée par chaque processeur durant le cycle d'échange. La meilleure solution entre celle obtenue par

l'échange et celle trouvée localement durant le cycle sera retenue pour effectuer la mise à jour globale.

4.3.1.4 Échange circulaire des meilleures solutions locales et des migrateurs

Cette stratégie est tout simplement la combinaison des deux stratégies présentées dans les sections précédentes. Elle nécessite la création de deux tableaux de solutions de dimension p. Le tableau T sert à stocker la meilleure solution locale de chaque processeur et le tableau U contient la meilleure solution trouvée durant le cycle. Chaque processeur écrit, à l'emplacement approprié de chaque tableau, chacune de ces deux solutions. Ensuite, chacun utilise l'information qu'il a reçue de son prédécesseur pour effectuer la mise à jour de sa sous-population.

4.3.2 La stratégie d'immigration et intégration sélectives

L'objectif du maintien de la diversité génétique est d'empêcher la *convergence prématurée*. Les sous-populations séparées pourraient avoir leurs propres caractéristiques génétiques. Les échanges (migrations) entre les sous-populations sont nécessaires pour aider ces petites entités (sous-populations) de sortir de leurs minima locaux. Les combinaisons des meilleurs gènes de différentes espèces sont supposées d'engendrer de meilleurs enfants. Par conséquent, les meilleurs individus sont les candidats à être copiés sur d'autres environnements. Étant donné que les clones du meilleur individu global peut dominer toutes les sous-populations et réduire la diversité génétique, la migration doit être soigneusement réalisée. La figure 4.3 montre l'aspect fonctionnel de la plate-forme de calcul proposée. Elle illustre les flux de données et de contrôle de calcul avec un intervalle d'échange (c-à-d, k générations consécutives). Dans cette figure, chaque bloc en gris contient une sous-population d'individus sur laquelle le calcul évolutionnaire continue pour un certain nombre de générations avant que l'échange ait lieu. Au cours de cette période, le calcul de chaque sous-population et indépendant des autres, l'évolution des différentes sous-populations peut alors s'effectuer simultanément. Dans notre travail, nous avons élaboré une nouvelle stratégie d'échange d'individus utilisant les différentes méthodes de fenêtre dynamique de migration proposées par Kim (2002) pour contrôler la taille et la fréquence des migrations et le modèle de migration sélective proposé par Eldos (2006) pour la sélection des migrants.

Kim (2002) a proposé deux méthodes de fenêtre dynamique de migration pour définir le taux de migration (le nombre d'individus à migrer au moment de l'échange) :

1. *Fenêtre dynamique croissante :* la taille initiale de la fenêtre est fixée à 1. À chaque échange, la taille de la fenêtre d'échange augmente d'une taille fixe jusqu'à ce que celle-ci soit égale à 20% de la taille de la sous-population.
2. *Fenêtre dynamique aléatoire :* à chaque fois qu'un échange se produit, la taille de la fenêtre dynamique varie aléatoirement. La variation est de 1 à θ. La valeur de θ varie dans un intervalle entre 1 et 20% de la taille de la sous-population à chaque migration.

Eldos (2006) a également proposé un nouveau modèle dit de migration sélective. Ce modèle s'inspire du phénomène de la migration réelle entre les pays du monde. Dans ce modèle, le choix d'individus est effectué selon un ensemble de conditions (attribution des visas). Par

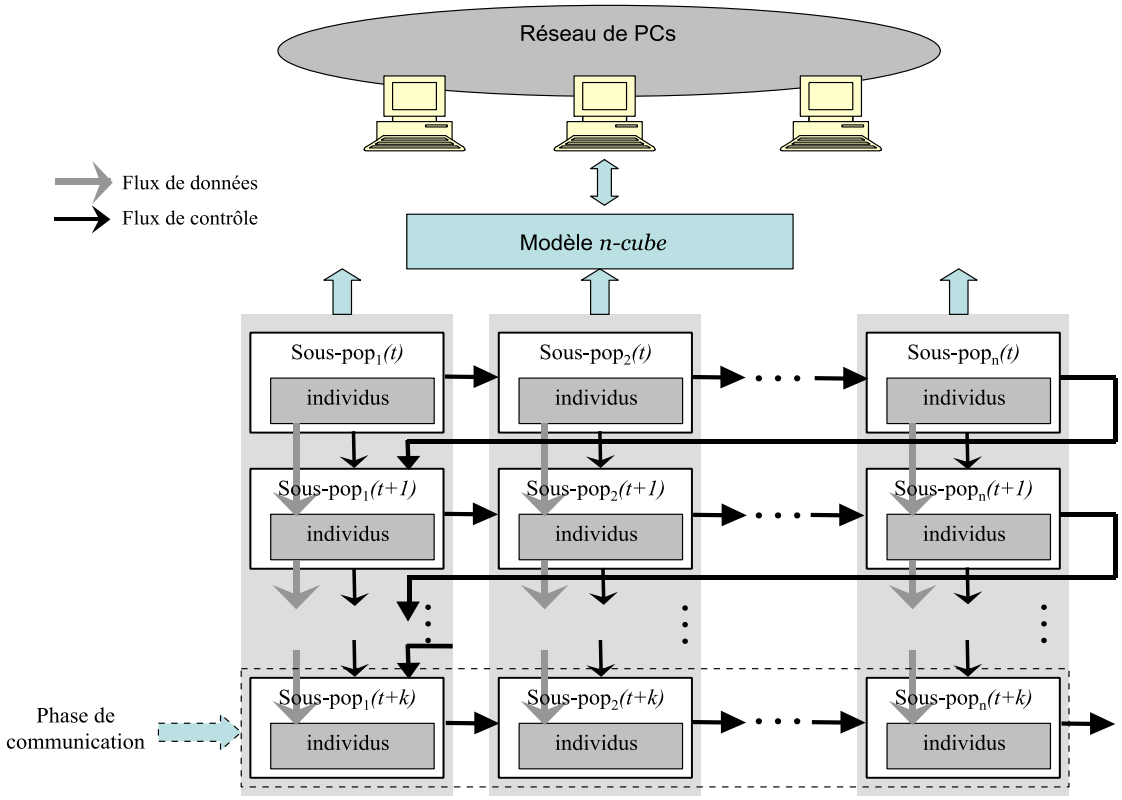


FIG. 4.3 – Coopération (adaptée à partir de Lee (2007))

exemple, les individus ayant migré à l'étape i seront marqués de manière à ne pas migrer à l'étape $i+1$.

La routine ***Migration*** dans le pseudo-code de l'algorithme 1 illustre notre stratégie d'échange d'individus.

L'assemblage technique ces différentes approches a donné lieu à la naissance d'une nouvelle stratégie d'échange d'informations dans le monde des algorithmes évolutionnaires distribués. Dans cette nouvelle stratégie, des nouveaux comportements ont été mis en place pour les différents types d'agents. Ces comportements sont définis en fonction de la stratégie du parallélisme utilisée.

4.3.3 Stratégie d'ajustement des paramètres d'un AE par le biais d'échange

4.3.3.1 Revue de la littérature

Par sa difficulté, le réglage des paramètres d'un algorithme évolutionnaire (AE) est souvent assimilé à de la magie noire (la littérature des algorithmes évolutionnaires utilise le terme de *Black Art*). Un algorithme évolutionnaire est contrôlé par plusieurs paramètres, e.g. la taille de la population, le taux de mutation, le taux de croisement, etc. Ces paramètres déterminent en grande partie le succès et l'efficacité de ce dernier pour résoudre un problème

spécifique. Malheureusement, ces paramètres interagissent les uns avec les autres de manière très compliquée. Beaucoup de praticiens trouvent des jeux de paramètres prometteurs pour des problèmes particuliers en essayant diverses combinaisons de paramètres de contrôle. Cette approche de sélection des paramètres exige évidemment un temps de calcul très important et qui peut être plus grand que le temps nécessaire pour résoudre un problème particulier par un AE lui-même. Schaffer *et al.* (1989) ont réalisé un test exhaustif de plusieurs combinaisons de paramètres. Ils ont soigneusement examiné la performance d'un AE en utilisant diverses combinaisons de paramètres de contrôle. Les expériences ont impliqué plusieurs fonctions de test de combinaisons de paramètres et ont duré environ 18 mois de temps CPU (Tongchim et Chongstitvatana (2002)).

Une autre étude a été réalisée par Deb et Agrawal (1998) pour focaliser les interactions entre les paramètres d'un AE. Les auteurs ont étudié l'effet de trois paramètres : la taille de la population, le taux de croisement et le taux de mutation. Les résultats ont montré que les approches basées sur la mutation et celles basées sur le croisement se comportent différemment entre les problèmes simples et complexes. Dans une autre approche, une taille correcte de la population est nécessaire pour atteindre de bonnes performances.

Pham (1995) a proposé une technique pour ajuster le paramètre de sélection en mettant en place une concurrence entre plusieurs sous-populations utilisant divers jeux de paramètres. Plusieurs sous-populations évoluaient de façon indépendante en utilisant leur propre jeu de paramètres. Ces sous-populations ont été maintenues par un seul processeur. Les sous-populations ayant de meilleurs jeux de paramètres avaient reçu un temps de traitement supplémentaire pour évoluer. Lis (1996) a introduit une technique permettant d'adapter le taux de mutation dans un modèle d'algorithme évolutionnaire parallèle. Plusieurs sous-populations évoluent séparément sur différents processeurs en utilisant différents taux de mutation. Après un intervalle prédéterminé, ces populations ont été comparés. Si le meilleur résultat a été obtenu par le processeur ayant le plus fort taux de mutation, les taux de mutation des autres processeurs ont été augmentés d'un niveau. Les taux de mutation ont également été réduits d'un niveau si le meilleur résultat a été obtenu par le processeur ayant le plus faible taux de mutation. Schlierkamp-Voosen et Mühlenbein (1994) ont utilisé une concurrence entre des sous-populations ayant chacune son propre jeu de paramètres pour sélectionner le meilleur jeu de paramètres. La taille de chaque groupe variait, tandis que la taille de la population totale a été fixée. Chaque groupe a concouru contre d'autres groupes pour obtenir sa taille. La taille du meilleur groupe a été augmentée, tandis que tous les autres groupes ont diminué. Une méthode similaire a été utilisée par Eiben *et al.* (1998) pour choisir le meilleur opérateur de croisement.

4.3.3.2 Un méta-algorithme pour ajuster les paramètres d'un AE parallèle

L'objectif primordial de l'approche proposée est de fournir une technique d'ajustement des paramètres pendant que la recherche est en cours. Cette approche se base sur l'observation de la performance de l'algorithme. En utilisant le concept de parallélisation à *gros grain*, la population est divisée en un nombre restreint de sous-populations. Ces sous-populations évoluent indépendamment et simultanément sur différents processeurs. Après un certain nombre de générations, certains individus sont échangés par le biais d'un processus de migration. Les sous-populations s'exécutent en utilisant différentes valeurs de paramètres.

Le méta-algorithme est appliqué à l'évolution des jeux de paramètres. Cet algorithme est basé sur la même topologie qu'un algorithme génétique dans lequel un chromosome illustre un jeu de paramètres. Les chromosomes sont représentés par des vecteurs d'entiers. Soit : $\overrightarrow{PS} = (ps_1, ..., ps_n)(ps_i \in [a, b], i = 1, .., n)$ un vecteur d'entiers. La valeur d'un élément particulier ps_i indique la valeur du i^{e} paramètre. Deux jeux de paramètres sont utilisés dans chaque sous-population. Le fitness moyen des individus traités par un jeu de paramètres est utilisé pour l'évaluation de l'adaptation du jeu de paramètres en question. Le meilleur jeu de paramètres sélectionné à partir des deux jeux de paramètres de chaque nœud est autorisé à se croiser avec un autre paramètre à partir d'autres nœuds. Chaque processeur décide si un jeu de paramètres local soit croisé avec un jeu de paramètres reçu depuis un nœud voisin. En particulier, chaque nœud envoie son meilleur jeu de paramètres avec son propre fitness aux autres nœuds. Si le fitness du jeu de paramètres du nœud voisin est meilleur que le meilleur jeu de paramètres local, deux nouveaux jeux de paramètres sont produits en appliquant une séquence d'opérateurs génétiques, tels que le croisement uniforme et la mutation (cf. l'annexe B), sur les deux jeux de paramètres. Chaque jeu de paramètres est utilisé pour produire la moitié de la nouvelle population. Par mutation, chaque champ dans les deux jeux de paramètres peut être remplacé par une valeur aléatoire en fonction d'une probabilité prédéterminée.

L'approche proposée est motivée par la méthode proposée par Pham (1995). L'auteur maintient plusieurs sous-populations en utilisant différents jeux de paramètres afin d'éviter un échec d'exécution causé par un mauvais jeu de paramètres initial. Cette méthode est pénalisée par l'augmentation des coûts de calcul puisque plusieurs sous-populations évoluent simultanément sur un seul processeur. Notre approche surmonte cet inconvénient en utilisant le modèle parallèle d'AE. En particulier, un modèle à *gros grain* est utilisé afin d'évaluer simultanément plusieurs jeux de paramètres. Pham (1995) utilise des jeux de paramètres statiques, alors que notre approche ajuste dynamiquement les jeux de paramètres en fonction de la performance observée. L'approche proposée peut être considérée comme un méta-niveau d'AE. Cette méthode se distingue des travaux réalisés par Grefenstette (1986) par le fait que les paramètres soient ajustés au cours de l'exécution de l'algorithme, Grefenstette (1986) déterminait les paramètres avant l'exécution de l'algorithme (Tongchim et Chongstitvatana (2002)).

Contrairement aux méthodes adaptatives définies par Pham (1995), Schlierkamp-Voosen et Mühlenbein (1994) et Eiben *et al.* (1998), notre méthode est conçue pour la parallélisation. Les techniques utilisaient dans ces études récompensent le succès d'une sous-population par l'augmentation de sa taille (Schlierkamp-Voosen et Mühlenbein (1994), Eiben *et al.* (1998)), ou en donnant un temps de traitement supplémentaire Pham (1995). Ces techniques peuvent ne pas être efficaces lorsqu'elles sont mises en œuvre en parallèle car elles sont susceptibles de déséquilibrer les charges de travail entre les processeurs.

La routine ***Controler_parametres*** dans le pseudo-code de l'algorithme 1 présente le fonctionnement du modèle d'ajustement automatique des paramètres.

Algorithme 1 Pseudo code

```
1: Initialiser la population, SP
2: tantque generation < max_generation faire
3:    Évaluer SP
4:    Sélectionner SP1' à partir de SP en utilisant la roue de la fortune biaisée
5:    sélectionner aléatoirement les parents dans SP1'
6:    appliquer les opérateurs génétiques pour créer le reste de la nouvelle population, SP2'
7:    fusionner SP1' et SP2' dans SP'
8:    remplacer SP par SP'
9:    si un intervalle de K générations est atteint alors
10:      Migration
11:      Controler_parametres
12:   generation = generation + 1
13: Routine Migration
14: Choisir aléatoirement un nombre (1 - pop_size) pour nommer un immigrant X
15: si le rang de X est dans les limites alors
16:   Envoyer X aux autres agents
17:   Marquer X comme "abandonner" à débarrasser dans (6)
18: si un immigrant Y est reçu alors
19:   si le rang de Y dépasse un seuil alors
20:      Ajouter Y dans la population
21:      Mettre à jour les fitness et trier la population
22:      Choisir aléatoirement un nombre (1 - pop_size) pour nommer une victime V
23:      Abandonner la victime V
24: Routine Controler_parametres
25: Envoyer le meilleur jeu de paramètres avec son fitness
26: Recevoir un jeu de paramètres avec son fitness
27: si le jeu de paramètres reçu est meilleur alors
28:   Produire deux nouveaux jeux de paramètres par le croisement et la mutation uniformes
```

4.4 Conclusion

Dans ce chapitre, nous avons présenté différentes stratégies d'échange d'informations et une stratégie d'adaptation automatique des paramètres des algorithmes. La première stratégie s'inspire des stratégies de communication définies par Middendorf *et al.* (2000) pour les colonies de fourmis parallèles. Nous avons adapté les quatres mécanismes inhérents au contexte des algorithmes génétiques. La seconde stratégie est basée sur les différentes méthodes de fenêtre dynamique de migration proposées par Kim (2002) pour contrôler la taille et la fréquence des migrations et le modèle de migration sélective proposé par Eldos (2006) pour la sélection des migrants. Une troisième et dernière stratégie consiste en l'ajustement automatique des paramètres des entités de recherche.

Ces différentes stratégies sont mises en œuvre sur différentes plates-formes décrites dans les chapitres suivants. Les performances de chacune de ces stratégies sont évoquées après chaque mise en œuvre.

Chapitre 5

CSM-PGA : Une approche client/serveur pour le CARP

Sommaire

5.1 Introduction

L'efficacité des métaheuristiques à traiter des problèmes d'optimisation combinatoire a été démontrée dans plusieurs travaux de recherche fondamentale et appliquée. Le succès de ces méthodes s'explique par un ensemble de facteurs, notamment par leur potentiel d'adaptation aux différentes contraintes associées à des problèmes spécifiques, par leur facilité d'implantation dans des programmes d'application divers et par la qualité des solutions qu'elles peuvent produire dans un temps relativement court. Le domaine des métaheuristiques est toutefois encore jeune et de nombreux efforts sont présentement mis en œuvre dans le but d'améliorer la performance de ces méthodes de résolution.

Le parallélisme n'est pas qu'une source de puissance de calcul, l'activité concurrente de plusieurs algorithmes peut accélérer des traitements par le biais d'une coopération entre ceux-ci. Le couplage du parallélisme avec le mécanisme de coopération dans le domaine des métaheuristiques fournit un cadre prometteur pour la réduction du temps de calcul et d'éventuelles améliorations de la qualité des recherches.

Le problème de tournées de véhicules consiste à déterminer, pour un ensemble donné de « clients », par quels véhicules de la flotte ils seront visités et à quel moment dans la tournée ils le seront. Deux grandes classes émergent en fonction de la position des « clients » : s'ils se trouvent sur des points précis d'un réseau, le problème étudié est un problème de tournées sur nœuds. En revanche, si les « clients » sont localisés sur des liens du réseau, il s'agit d'un problème de tournées sur arcs. La deuxième classe apparaît quand les « clients » se trouvent distribués le long des rues ou lorsque la rue elle-même nécessite un service.

Les tournées sur arcs ont été beaucoup moins étudiées, comparé aux tournées sur nœuds, bien qu'elles permettent de modéliser un grand nombre d'applications réelles comme la collecte des déchets, le déneigement des routes, le relevé de compteurs d'électricité, la distribution de courrier et l'inspection de lignes à haute tension. Vu la complexité des problèmes de tournées sur arcs, ils ont été abordés de façon progressive par la communauté scientifique, en commençant par des cas simplifiés et en ajoutant une à une les difficultés. Des recherches ont par ailleurs porté sur des applications particulières exigeant des modélisations et des techniques de résolution spécifiques. Il aurait fallu du temps avant de pouvoir proposer un modèle plus général nommé CARP (pour *Capacitated Arc Routing Problem*), regroupant ainsi la majorité des problèmes de tournées sur arcs.

Jusqu'à présent, les métaheuristiques publiées pour le CARP de base sont des méthodes tabou (Eglese et Li (1996), Hertz *et al.* (2000)), des recherches à voisinage variable (Mittaz (2000)), des méthodes de recuit simulé (Eglese (1994)), un algorithme hybride entre tabou et scatter search de Greistorfer (2003), une méthode de recherche locale guidée (GLS) de Beullens *et al.* (2003), un algorithme génétique (Lacomme *et al.* (2001)) et un algorithme mémétique (Lacomme *et al.* (2004)). L'algorithme le plus efficace est l'algorithme mémétique de Lacomme *et al.* (2004). Après avoir constaté l'absence de méthodes évolutionnistes parallèles, nous proposons dans ce chapitre le premier algorithme génétique parallèle (CSM-PGA) pour le CARP de base.

Les algorithmes génétiques parallèles sont efficaces pour la résolution de problèmes de grandes tailles. La plupart de ces algorithmes ont été mis en œuvre sur des machines parallèles massives et leur efficacité dépend du système de calcul parallèle.

Dans ce chapitre, nous proposons une approche « client/serveur » pour paralléliser un al-

gorithme génétique avec un modèle de gestion déléguée, qui gère l'échange d'informations (individus) entre les sous-populations à travers le serveur et élimine les communications directes entre les sous-populations. Nous avons utilisé cette approche pour optimiser des problèmes de tournées sur arcs (CARP). L'algorithme proposé dans ce chapitre s'inspire de l'algorithme proposé par Kojima *et al.* (2000).

Nous rappelons la définition du problème expérimental (CARP), nous décrivons le calcul du distancier et trois heuristiques constructives utilisées dans la population initiale. Enfin, nous effectuons une comparaison entre *CSM-PGA* et d'autres méthodes de résolution pour le CARP sur deux bibliothèques de jeux d'essai de la littérature.

5.2 Analyse du problème expérimental (CARP)

5.2.1 Définition du CARP

Le CARP[1] est défini dans la littérature sur un graphe non orienté valué *G=(V, E, C)*. L'ensemble V des nœuds comprend un dépôt d où est basée une flotte homogène de K véhicules identiques de capacité W, en nombre suffisant. Le nombre de véhicules K peut être imposé ou être une variable de décision. Un sous-ensemble d'arêtes $R(R \subseteq E)$ doit être obligatoirement traversé pour être traité. La traversée (avec ou sans traitement) d'une arête $[i, j]$ par un véhicule coûte c_{ij} et chaque arête de R a une demande r_{ij}.

Le CARP consiste à déterminer un ensemble de tournées de coût total minimal, tel que :
- chaque tournée est assurée par un véhicule qui part et revient au dépôt,
- la somme des demandes traitées par une tournée ne dépasse pas W,
- chaque arête de R est effectivement traitée, par une seule tournée et lors d'un seul passage (pas de traitement partiel).

Le CARP orienté ou non est *NP-difficile* (réduction du problème de *Bin-Packing*, cf. Dror (2000)). Signalons que le cas $R = E$ a été introduit dès 1973 par Christofides (1973), sous le nom de CCPP *(Capacitated Chinese Postman Problem)*.

La figure 5.1 montre un graphe représentant un problème de CARP. Les coûts liés aux collectes[2] sont représentés entre parenthèses suivis des coûts de traversées.

L'optimisation d'un problème de type CARP consiste à trouver l'ensemble des tournées qui permettent de traverser toutes les tâches, de respecter la capacité des véhicules et dont le coût total est minimum. La figure 5.2 montre une solution possible au problème pour des véhicules ayant une capacité $W = 27$.

Dans cet exemple, la solution représentée comporte trois tournées respectant chacune la capacité $W = 27$ des véhicules et dont le coût total de traversée est de 112[3].

5.2.2 Formulation mathématique du CARP

Nous rappelons ici la première formulation pour le CARP, introduite par Golden et Wong (1981). Elle concerne un graphe non orienté. Les nœuds sont indicés de 1 (le dépôt) à n. Soit A l'ensemble des arcs obtenus en remplaçant chaque arête par deux arcs opposés, x_{ijk} une

[1] Capacitated Arc Routing Problem
[2] Les coûts liés aux collectes peuvent, par exemple, correspondre à des masses ou des volumes.
[3] Somme des coûts de traversées de chaque tournée 43+18+51 = 112.

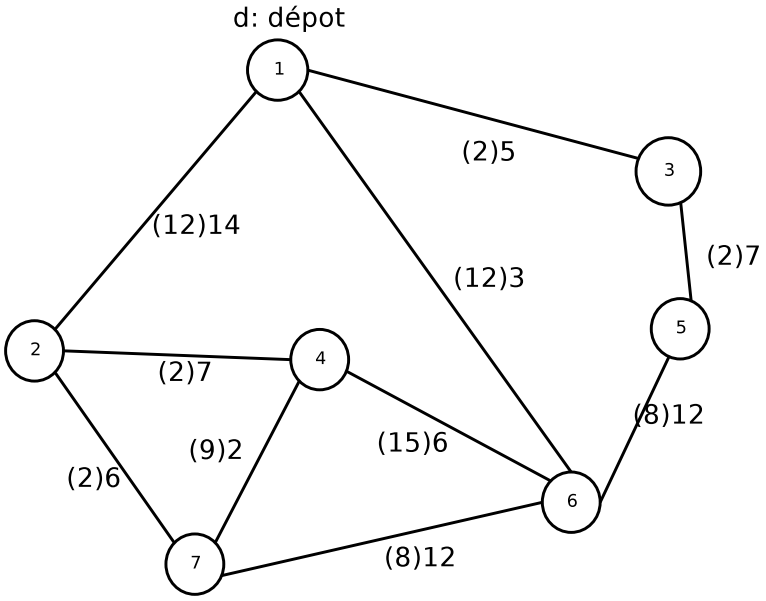


FIG. 5.1 – Représentation du graphe $G = (V, E, C)$

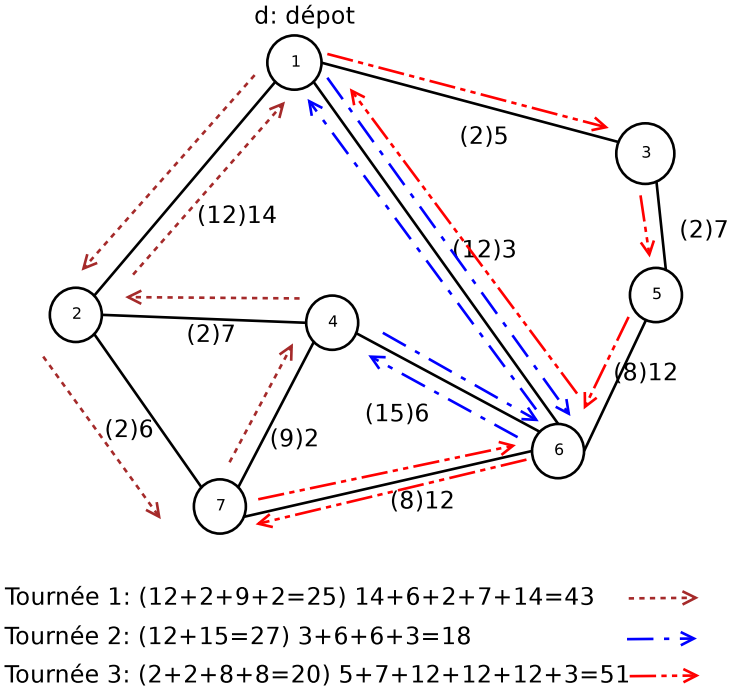


FIG. 5.2 – Représentation d'une solution possible du graphe $G = (V, E, C)$

variable binaire égale à 1 si et seulement si le véhicule k traverse l'arête $[i, j]$ de i vers j et l_{ijk} une variable binaire valant 1 si et seulement si le véhicule k traite $[i, j]$ de i vers j.

$$Minimiser \sum_{k \in I} \sum_{(i,j) \in A} c_{ij} x_{ijk} \tag{5.1}$$

$$\forall i \in V, \forall k \in I : \sum_{j \in \Gamma(i)} (x_{ijk} - x_{jik}) = 0 \tag{5.2}$$

$$\forall (i,j) \in A, \forall k \in I : x_{ijk} \geq l_{ijk} \tag{5.3}$$

$$\forall (i,j) \in R, \sum_{k \in I} (l_{ijk} + l_{jik}) = 1 \tag{5.4}$$

$$\forall k \in I, \sum_{(i,j) \in R} l_{ijk} r_{ij} \leq W \tag{5.5}$$

$$u_s^k, y_s^k, x_{ijk}, l_{ijk} \in \{0,1\} \tag{5.6}$$

$$\forall S \neq \emptyset, S \subset \{2,..,n\} \; and \; \forall k \in I : \begin{cases} \sum_{i,j \in S} x_{ijk} - n^2 y_s^k \leq \mid S \mid -1 \\ \sum_{i \in S} \sum_{j \in S} x_{ijk} + u_s^k \geq 1 \\ u_s^k + y_s^k \leq 1 \end{cases} \tag{5.7}$$

$\Gamma(i)$: ensemble des successeurs du sommet i.

L'objectif (5.1) consiste à minimiser la somme des coûts de parcours. Les contraintes (5.2) garantissent qu'un véhicule qui arrive en un nœud doit en repartir. Les contraintes (5.3) garantissent qu'un arc traité est aussi traversé. Les contraintes (5.4) imposent que chaque arête requise soit traitée, par un seul véhicule et dans un seul sens. Les contraintes (5.5) assurent le respect de la capacité des véhicules. Enfin l'interdiction des sous-tours illégaux est imposée par les contraintes (5.7) qui nécessitent des variables binaires supplémentaires y_s^k et u_s^k.

5.3 Algorithme génétique pour le CARP

Nous avons repris une partie des travaux effectués par Lacomme *et al.* (2001) pour mettre en œuvre un algorithme génétique pour le CARP. Certains aspects n'ont cependant pas été repris, comme la recherche locale. Cette hybridation (recherche génétique/recherche locale) risquerait de rendre plus difficile l'étude de l'apport lié à l'échange d'informations.

Nous avons choisi d'expérimenter nos outils sur des problèmes de CARP pour lesquels à chaque arête du graphe correspond une demande à traiter, ce qui signifie que chaque arête doit être parcourue et traitée. Dans les problèmes que nous allons traiter, il n'existe pas d'arête sans *demande*. Dans la suite du document, nous utiliserons uniquement le mot *tâche* pour définir les *arêtes*.

5.3.1 Modélisation du CARP

Soit un graphe non orienté $G = (V, E)$ où V est un ensemble de n *sommets* et E un ensemble de m *tâches*. Nous appellerons $Task_z$ chaque arête (a_z, b_z) de E où $a_z \in V$ est le sommet de départ de la tâche et $b_z \in V$ est le sommet d'arrivée avec $z \in \{1, 2, ..., m\}$. La

figure 5.3 montre une représentation de ce graphe. Le dépôt où sont stockés les véhicules de capacité W correspond au sommet 1.

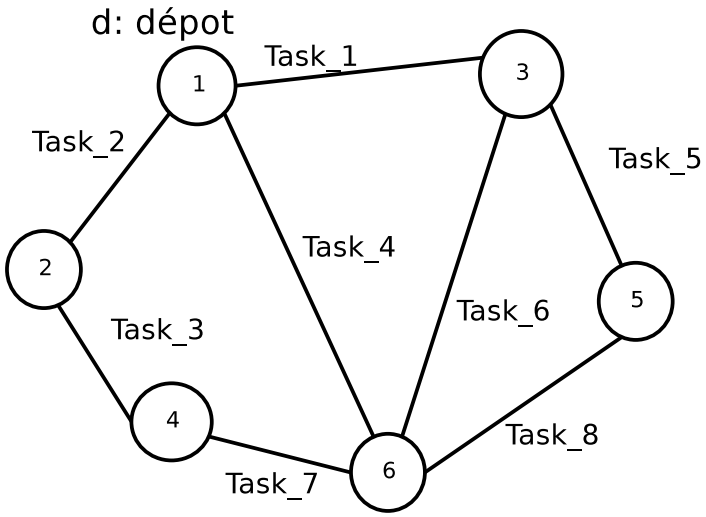


FIG. 5.3 – Graphe $G = (V, E)$ composé de six sommets et huit tâches

À chaque tâche du graphe est associé un coût de traversée $cost(Task_z)$ et un coût de demande $load(Task_z)$. Pour illustrer la partie suivante, nous prendrons comme exemple les valeurs présentées dans le tableau 5.1 correspondant au graphe de la figure 5.3.

TAB. 5.1 – Coût de traversée et de demande

tâche	sommets	cost	load
$task_1$	(1,3)	2	3
$task_2$	(1,2)	4	1
$task_3$	(2,4)	3	3
$task_4$	(1,6)	5	2
$task_5$	(3,5)	7	2
$task_6$	(3,6)	2	1
$task_7$	(4,6)	2	3
$task_8$	(5,6)	2	3

5.3.2 Codage des solutions

Le codage des solutions doit permettre de représenter les différentes tournées de collecte. Chaque tournée peut être représentée sous la forme d'une séquence de tâches à parcourir. Comme nous le montrent Lacomme *et al.* (2001), le codage des solutions ne prend pas en compte le chemin à parcourir entre deux tâches à traiter. En effet, il existe des algorithmes tels que l'algorithme de ***Dijkstra*** (Dijkstra (1959)) permettant de calculer un *distancier* entre tous les sommets d'un graphe en $O(m^3)$ ou $O(m^2.logm)$. Seul l'ordre des tâches à traiter est

donc nécessaire pour coder une solution.

Il n'est pas nécessaire de coder les séparations entre deux tournées successives.

Nous travaillons ici sur un graphe non-orienté où le coût d'une tâche est identique quelque soit le sens de traversée. Cependant le calcul de distance entre deux tâches diffère suivant le sommet par lequel on sort de la première tâche et celui par lequel on entre dans la seconde. Il est donc nécessaire de prendre en compte, dans le codage des solutions, le sens par lequel les tâches seront traversées.

Le codage d'une solution sera donc représenté par une séquence d'entiers représentant les indices z des tâches parcourues ou $(z + m)$ (m étant le nombre total de tâches) dans le cas d'une tâche parcourue dans le sens inverse. Cette séquence de tâches sera appelée *chromosome* dans la suite du manuscrit.

Prenons l'exemple de la tournée complète représentée par la séquence 5.8 :

$$\{task_1, task_2, task_5, inv(task_6), task_3, task_7, task_8, inv(task_4)\} \tag{5.8}$$

Cette séquence sera représentée par le chromosome suivant :

$$chrom = \{1; 2; 5; 14; 3; 7; 8; 12\} \tag{5.9}$$

La figure 5.4 nous montre, sur le graphe, la représentation de cette tournée complète. Il est bien évident à ce stade que les départs et retours au dépôt entre chaque tournée ne sont pas représentés.

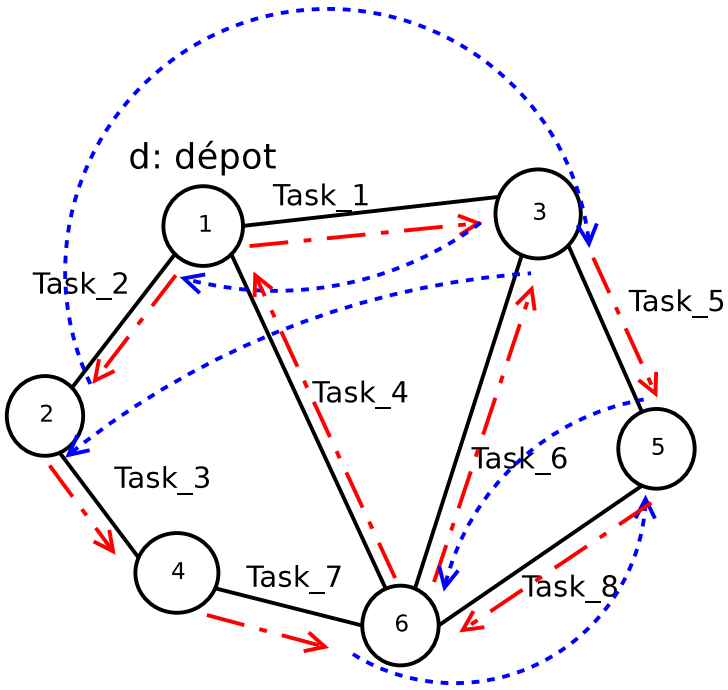


FIG. 5.4 – Représentation de la tournée complète

5.3.3 Fonction objectif

Le CARP consiste à trouver le plus court chemin permettant de traiter toutes les tâches du graphe. Notre fonction objectif consistera donc à calculer le coût du chemin représenté par notre chromosome.

Comme nous l'avons vu précédemment, nous n'avons pas pris en compte le découpage en tournées dans le codage de notre chromosome. Le chromosome représente donc une tournée complète que nous allons devoir découper pour être en mesure de calculer le coût de chaque tournée puis le coût total de la solution. Ce choix a été effectué afin de faciliter la recherche génétique, c'est-à-dire l'utilisation des opérateurs génétiques simples (cf. section 5.3.5). Ceci permettra également de diminuer le temps de calcul.

Le découpage de la tournée complète se fait par la construction d'un graphe intermédiaire représentant tous les découpages possibles du chromosome respectant les capacités des véhicules. Tous les retours possibles au dépôt sont représentés sur le graphe, même si les véhicules ne sont que partiellement remplis. Il ne restera ensuite qu'à trouver le plus court chemin sur ce graphe pour obtenir le découpage optimal.

Pour construire ce graphe, il est nécessaire de connaître les distances les plus courtes entre chaque sommet du graphe. Nous allons donc nous intéresser au calcul de ce *distancier*.

5.3.3.1 Calcul d'un distancier

Nous avons choisi d'utiliser l'algorithme de Dijkstra (1959) pour calculer le distancier. Cet algorithme permet de calculer les distances entre un sommet d'un graphe et tous les autres. Lacomme et Prins (2006) en détaillent les différents mécanismes.

La première phase consiste à créer une matrice D de dimension (nbs, nbs) où nbs représente le nombre de sommets du graphe et de l'initialiser de la façon suivante :

$$D_{i,j} = \begin{cases} 0 & \text{si} \quad i = j \\ \infty & \text{si} \quad i \neq j \quad \text{et} \quad (i,j) \notin E \\ cost(i,j) & \text{si} \quad i \neq j \quad \text{et} \quad (i,j) \in E \\ cost(j,i) & \text{si} \quad i \neq j \quad \text{et} \quad (j,i) \in E \end{cases} \tag{5.10}$$

Deux autres matrices de taille nbs sont crées. La premiere $Dist_i$ contiendra la distance minimum trouvée entre le sommet s pour lequel on recherche les distances minimums et les sommets i. La seconde $Pred_i$ contiendra le prédécesseur de i en provenance du sommet s par le plus court chemin. Ces matrices sont initialisées comme suit pour le calcul des plus courts chemins entre le sommet s et tous les sommets du graphe :

$$Dist_i = \begin{cases} \infty & \text{si} \quad i \neq s \quad \text{et} \quad (i,s) \notin E \\ cost(i,s) & \text{si} \quad i \neq s \quad \text{et} \quad (i,s) \in E \end{cases} \tag{5.11}$$

$$Pred_i = \begin{cases} -1 & \text{si} \quad i \neq s \quad \text{et} \quad (i,s) \notin F \\ s & \text{si} \quad i \neq s \quad \text{et} \quad (i,s) \in F \end{cases} \tag{5.12}$$

Algorithme 2 permettant de trouver, pour chaque sommet, un prédécesseur et le coût en distance entre ce prédécesseur et ce sommet :

Une fois que nous connaissons les prédécesseurs de chaque sommet pour les plus courts

Algorithme 2 Calcul du distancier

```
1: tantque tous les sommets n'ont pas été examinés faire
2:    choisir le sommet X non examiné tel que Dist(X,Pred[X])=min
3:    marquer le sommet X comme examiné
4:    pour Y=0 à Nb_Sommet faire
5:       si Dist(X,Y) existe et Y n'a pas été examiné alors
6:          si Dist(Y,Pred[y]) > Dist(X,Pred[X]) + Dist(X,Y) alors
7:             Pred[Y] =X
8:             Dist(Y,Pred[Y]) = Dist(X,Pred[X]) + Dist(X,Y)
```

chemins en provenance de s, il reste à définir le chemin complet entre chaque sommet et le sommet s puis d'en calculer le coût. L'algorithme 3 permet de définir le chemin le plus court. Le vecteur *Le_Chemin* contient tous les sommets successifs entre le sommet x et le sommet s.

Algorithme 3 Calcul du plus court le chemin

```
1: cpt=0
2: tantque source ≠ s et cpt<Nb_Sommet faire
3:    cpt=cpt+1
4:    Le_chemin(cpt)=pred(source)
5:    source = Pred(source)
```

À partir de la liste des sommets qui composent le chemin le plus court, il est possible de calculer le coût minimum entre un sommet et le sommet s (cf. Algorithme 4) :

Algorithme 4 Calcul du coût minimum entre un sommet et le dépôt s

```
1: distance = 0
2: pour i=cpt à i=1 faire
3:    distance = distance + D(Le_Chemin(i), Le_Chemin(i-1) )
```

Ces deux derniers algorithmes sont à répéter autant de fois qu'il existe de sommets. On obtient alors, pour chaque sommet, la distance minimum pour rejoindre le sommets s. L'ensemble de ces algorithmes sera donc à répéter autant de fois qu'il existe de sommets.

Cet algorithme nous donne le résultat (cf. le tableau 5.2) pour l'exemple défini dans le tableau 5.1[4] :

5.3.3.2 Découpage en tournées

À partir du *chromosome* (équation 5.9) et du distancier (tableau 5.2) il est possible de créer un graphe (voir figure 5.5) représentant la tournée complète avec des chemins optimaux entre chaque tâche et tous les aller-retours possibles au dépôt. Sur ce graphe, les coûts de ramassage sont indiqués entre parenthèses.

À partir de ce graphe, on définit un graphe intermédiaire (voir figure 5.6) représentant toutes les tournées respectant les capacités des camions. Dans notre exemple, les camions ont

[4]Le graphe étant non-orienté, le résultat obtenu est symétrique

TAB. 5.2 – Résultat du distancier sur le graphe du tableau 5.1

	1	2	3	4	5	6
1	0	4	2	6	6	4
2	4	0	6	3	7	5
3	2	6	0	4	4	2
4	6	3	4	0	4	2
5	6	7	4	4	0	2
6	4	5	2	2	2	0

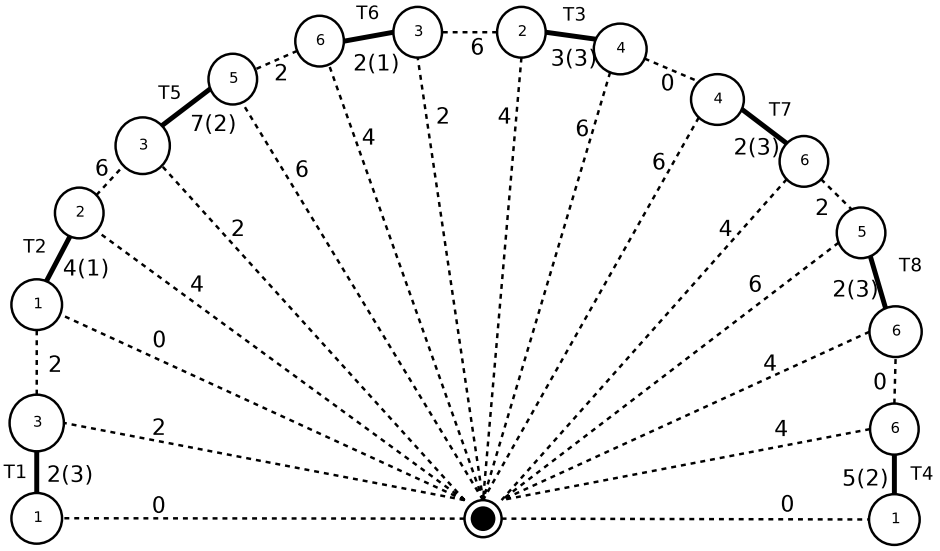


FIG. 5.5 – Représentation de la tournée complète avec les chemins optimaux

une capacité $W = 5$.

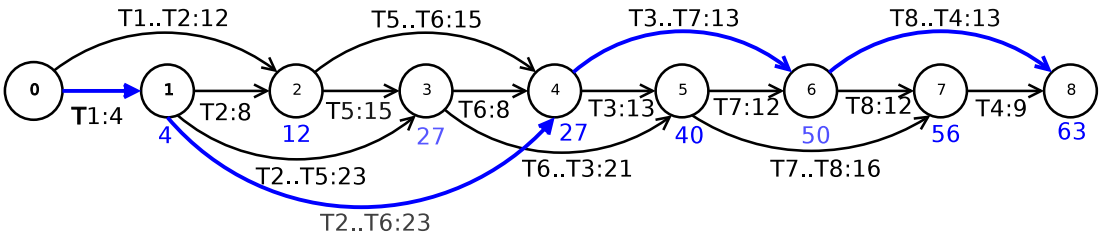


FIG. 5.6 – Graphe intermédiaire des tournées réalisables

L'application de l'algorithme de *Dijkstra* sur ce graphe intermédiaire permet de déduire

le chemin le plus court en respectant la capacité des camions. Une fois ce chemin déterminé, nous pouvons en calculer le coût. Dans l'exemple que nous avons choisi, le coût obtenu est égal à 63.

Maintenant que nous sommes capables d'évaluer chaque solution (ou *individu*), nous allons pouvoir nous intéresser à la sélection des *individus* qui participeront à la reproduction et nous verrons ensuite les différentes opérations génétiques.

5.3.4 Sélection

Nous avons choisi de mettre en œuvre la méthode de sélection proportionnelle *RWS* (cf. annexe B). L'algorithme 5 de cette méthode est détaillé ci-dessous :

Algorithme 5 La méthode de sélection proportionnelle *RWS*

```
1: pick = Random[0,1]  // Choix d'un nombre aléatoire entre 0 et 1
2: Stotal = Somme total des fonctions objectifs de tous les individus de la population
3: S = 0
4: i = 0  // individu choisi
5: répéter
6:    S = S + FonctionObjectif[i]
7:    i++
8: jusqu'à pick > S/Stotal
```

5.3.5 Opérateurs génétiques pour le CARP

Nous souhaitons étudier l'apport de l'échange d'informations entre des unités de calcul. L'échange d'informations n'a d'intérêt que si les unités de calcul ont la possibilité d'avoir des axes de recherche différents. Pour cela, il est nécessaire de pouvoir paramétrer différemment chaque unité de calcul. Outre le paramétrage des probabilités affectées à chaque unité de calcul, nous avons mis en œuvre différentes opérations génétiques pouvant être exécutées sur chaque algorithme génétique.

Nous avons donc repris les différents opérateurs de croisement et mutation étudiés par Lacomme *et al.* (2001). Nous en détaillons le fonctionnement ci-dessous :

5.3.5.1 Croisement

Linear Order Crossover : Soit deux parents $P1$ et $P2$, le croisement LOX[5] consiste à choisir deux positions p et q avec $p \leq q$. Les tâches comprises entre les deux positions p et q des parents $P1$ et $P2$ sont copiés aux mêmes emplacements dans les enfants $E1$ et $E2$.

Toutes les tâches de $P2$ sont ensuite balayées une à une, chaque tâche non encore présente dans $E1$ est ajoutée aux emplacements libres de gauche à droite. La figure 5.7 en montre le mécanisme.

[5]Linear Order Crossover

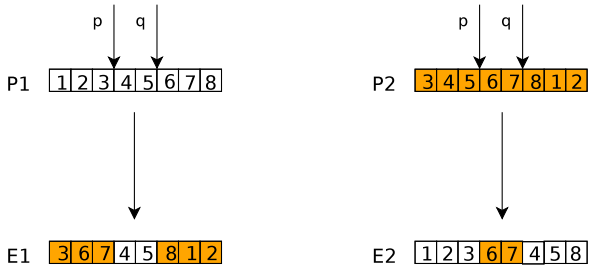


FIG. 5.7 – Linear Order Crossover

Order Crossover : Le croisement OX est identique au croisement LOX à la différence près que le balayage de $P1$ et le rangement dans $E1$ sont réalisés circulairement à partir de la position $q + 1$. Ce croisement est représenté dans la figure 5.8.

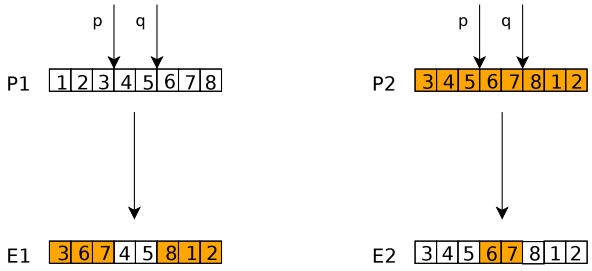


FIG. 5.8 – Order Crossover

Croisement à un point de coupure : Le croisement à un point de coupure est un cas particulier du croisement LOX avec $p = 1$ comme nous le montre la figure 5.9.

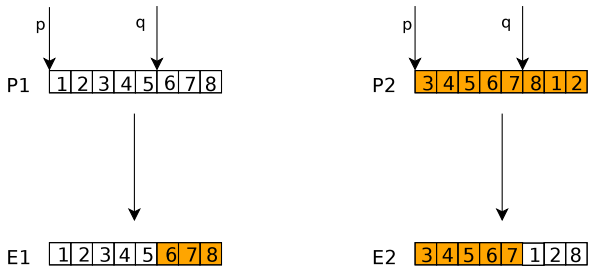


FIG. 5.9 – Croisement à un point de coupure

5.3.5.2 Mutation

En général dans les algorithmes génétiques, les mutations consistent à modifier un bit du chromosome. Dans le CARP, le chromosome représente une suite de tâches à traverser. Les mutations que nous allons mettre en œuvre consisteront donc à modifier cet ordre. Il sera également ajouté pour chaque type de mutation une opération consistant à inverser la ou les tâches qui auront été déplacées. Cette inversion sera appliquée avec une probabilité de 0,5.

Move : La mutation de type *move* consiste à choisir deux positions p et q tel que $1 \leqslant p \leqslant t$[6] , $0 \leqslant q \leqslant t$, $p \neq q$ et $p \neq q - 1$. La tâche de l'indice p est déplacée après la tâche de l'indice q[7] comme le montre la figure 5.10.

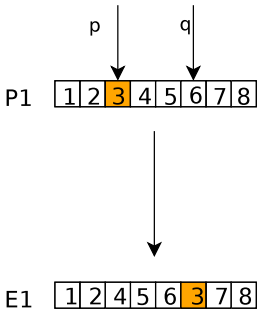


Fig. 5.10 – Move

Swap : Ici, la mutation consiste à échanger les tâches d'indices p et q. La figure 5.11 représente une mutation de type *swap* sans inversion.

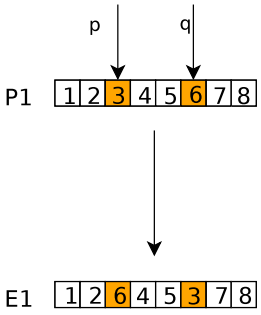


Fig. 5.11 – Swap

[6] t correspondant au nombre de tâches
[7] $q = 0$, correspond à une insertion en début de chromosome.

5.4 CSM-PGA pour le CARP

De nombreux modèles pour la parallélisation des algorithmes génétiques (AGs) ont été proposés dans la littérature depuis le début des recherches sur les AGs mais on recense principalement deux types de parallélisation. La première concerne la parallélisation de l'évaluation des individus. Elle a été introduite pour accélérer les traitements des calculs qui peuvent s'avérer coûteux dans une métaheuristique, sans essayer d'explorer d'avantage l'espace de recherche. Cette méthode correspond à une méthode séquentielle dont les traitements ont été accélérés. Dans l'autre cas, la recherche est divisée en plusieurs entités avec différents niveaux de synchronisation et de coopération. Le but de cette méthode est d'explorer plus activement l'espace de recherche en initialisant l'algorithme sur plusieurs points de cet espace. Les recherches s'effectuent ensuite simultanément à partir de ces points initiaux.

En effet, les algorithmes génétiques(AGs) parallèles à gros grains présentent un courant dominant dans le champ des études actuelles sur les modèles d'algorithmes génétiques parallèles et distribués. Cependant, lorsque nous réalisons un modèle en îlots dans un environnement de calcul à mémoire distribuée (réseau de machines), nous devons faire face aux problèmes suivants :

1. L'algorithme génétique a besoin de synchroniser les échanges d'informations entre les sous-populations. Cette synchronisation nécessite l'arrêt instantané des processus de calcul.
2. Les demandes d'informations arrivent des autres sous-populations de manière asynchrone et le choix des sous-populations candidates à l'échange est très compliqué. Par conséquent, le processus global pourrait être interrompu.
3. Les individus sont transmis et reçus pour empêcher la convergence prématurée. Par conséquent, la quantité des communications s'accroît en fonction de l'accroissement de la taille de la population et de la longueur des chromosomes.

Pour faire face à ces problèmes, nous proposons un modèle d'algorithme génétique parallèle dit *CSM-PGA*[8].

5.5 Implémentation de CSM-PGA

Dans cette étude, l'environnement de calcul distribué est réalisé par plusieurs machines reliées par un réseau et nous proposons un algorithme génétique parallèle distribué utilisant un modèle de gestion déléguée. Ce modèle facilite la mise en œuvre de l'algorithme génétique parallèle et permet de réduire les coûts de communication. La figure 5.12 nous montre l'architecture du modèle proposé.

Dans ce modèle, la population initiale est générée sur le serveur puis divisée en plusieurs sous-populations (autant de sous-populations que de clients). Ce modèle est mis en œuvre sous forme d'un client-serveur avec le serveur de gestion déléguée qui rassemble l'élite de chaque sous-population des clients travaillant sur ces sous-populations (cf. figure 5.13).

Les clients exécutent chacun un algorithme génétique (défini dans la section 5.3) sur une machine du réseau. Un client de recherche possède sa propre sous-population et ses propres

[8]Client-Server Model for Parallel Genetic Algorithms

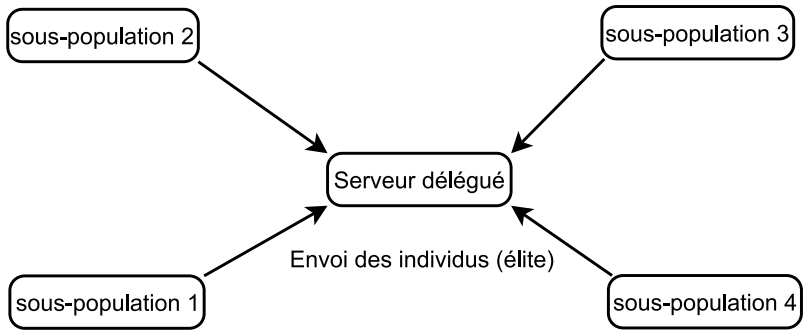


FIG. 5.12 – Modèle de gestion déléguée (adapté à partir de (Kojima *et al.* (2000)))

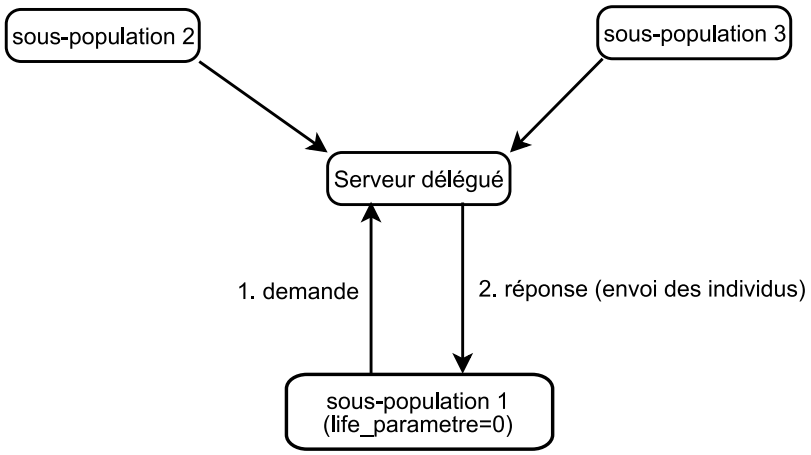


FIG. 5.13 – Modèle de la gestion déléguée au moment de la reconstruction (adapté à partir de (Kojima *et al.* (2000)))

paramètres de recherche. En outre, le modèle proposé utilise les règles suivantes en tant que stratégie de communication entre les clients.

- Quand l'élite est mise à jour, la sous-population envoie les informations d'une élite au serveur délégué.
- Chaque sous-population a un paramètre dit de "vie"(*life_parametre*) qui diminue en fonction de l'évolution de la recherche. Le paramètre de vie est diminué quand la sous-population a des difficultés de recherche (ex. convergence prématurée).
- Lorsque le paramètre de vie devient nul, la sous-population l'informe au serveur délégué, et la sous-population est restructurée en se basant sur les informations des délégués des autres sous-populations.
- Le serveur délégué gère les copies des élites ayant été envoyées par les sous-populations (clients). Lorsqu'une sous-population nécessite une reconstruction, le serveur délégué lui renvoie les élites nécessaires.

5.5.1 Contrôle des communications et stratégie de reconstruction

Le paramètre dit de vie est un indice pour décider à quel moment une sous-population reçoit des individus des autres sous-populations via le serveur. Lorsque la reconstruction n'est pas fréquente, le coût du à la reconstruction est très faible. Cependant, quand la fréquence de reconstruction s'approche de 0, la performance de la recherche décroît et le risque de la convergence prématurée devient important. En d'autres termes, il faut trouver un compromis entre la quantité de communication et la performance de la recherche. Par conséquent, nous mettons en place une règle qui contrôle l'état de la population. Un algorithme génétique s'exécutant sur une sous-population et qui a une difficulté de recherche due à une convergence prématurée diminue son paramètre de vie et la restructuration de la sous-population a lieu quand ce paramètre devient 0.

Différentes méthodes pour diminuer la valeur du paramètre de vie peuvent être proposées. Nous considérons « la génération quand une élite n'a pas été mise à jour » comme « indice de la mauvaise évolution de l'algorithme génétique » : le paramètre de vie est diminué d'un pas. Quand une élite est mise à jour, le paramètre de vie est remis à une valeur initiale. Par contre, quand l'évolution de la recherche de l'algorithme génétique se montre performante, la sous-population n'est pas restructurée, par conséquent, la nécessité de communication est diminuée.

Dans le modèle que nous proposons, quand l'algorithme génétique ait des difficultés d'évolution sur sa propre sous-population et le paramètre de vie devient 0, la sous-population est restructurée avec les élites des autres sous-populations qui sont obtenues du serveur de gestion. Plusieurs méthodes sont possibles pour le mettre en œuvre, mais nous avons besoin d'une technique la plus simple possible pour ne pas gêner l'algorithme génétique. Donc, nous avons utilisé une méthode où la sous-population est triée dans l'ordre de fitness et les individus dont les fitness sont les plus mauvais sont remplacés par les élites des autres sous-populations (cf. figure 5.14). En d'autres termes, nous avons utilisé une stratégie d'échange que nous avons décrite dans la section 4.3.1.2 du chapitre 4.

5.5.2 Les caractéristiques du modèle proposé

Quelques unes de ces caractéristiques ont été proposées par Kojima *et al.* (2000) mais argumentées différemment ici. Les caractéristiques du modèle proposé sont les suivantes :

1. Aucune synchronisation n'est nécessaire entre les sous-populations. Dans le modèle en îlot, la synchronisation entre les sous-populations est indispensable pour l'envoi et la réception des individus à un taux fixe. Par conséquent, un client doit attendre l'arrêt de l'algorithme génétique des autres clients pour échanger des individus. En outre, cet échange d'individus se produit probablement via des demandes de l'extérieur, indépendamment de la nécessité qui se pose dans la sous-population.

   Dans notre modèle, un client n'a pas besoin d'attendre les autres clients car il n'y a pas de communication directe entre les sous-populations. Les clients peuvent se consacrer uniquement à l'exécution de leur algorithme génétique et communiquent si cela est nécessaire. Par conséquent, les coûts de communication deviennent faibles.

2. Le nombre d'individus échangés est faible. En fait, c'est la communication des informations concernant les individus (fitness) qui occupe la majorité du volume de la com-

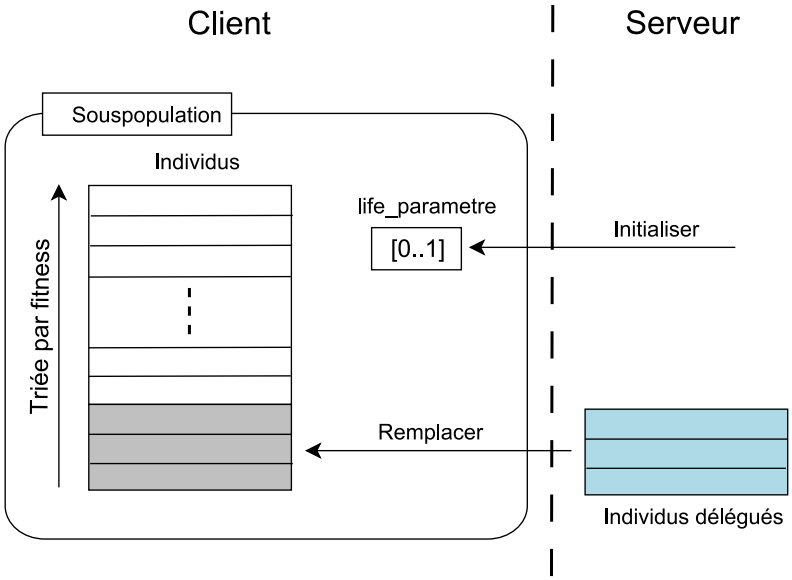


FIG. 5.14 – Processus de reconstruction (adapté à partir de (Kojima *et al.* (2000)))

munication dans le système de calcul parallèle. Dans notre modèle, un seul individu est communiqué, c'est le délégué de chaque sous-population. Ainsi, la quantité de communication est très faible par rapport au modèle d'AGs en îlots dans lequel les échanges d'individus sont effectués à un taux fixe.

3. La mise en œuvre et l'utilisation sur réseau sont simples. La synchronisation et la communication entre les sous-populations ne sont pas nécessaires parce que notre modèle parallèle est basé sur une stratégie *client-serveur*. Ce modèle peut être facilement réalisé sans avoir besoin d'une machine parallèle dédiée ou un langage de calcul parallèle spécifique. Le serveur est actif en permanence, le client nécessite un espace mémoire important, et la puissance de calcul sera aisée à faire varier en fonction de l'environnement de l'utilisateur. Il est difficile de confirmer les états de calcul parallèle mais facile de surveiller les états des clients dans ce modèle. Nous pouvons ainsi exécuter des algorithmes génétiques sur des machines puissantes et contrôler les états sur une machine à faible puissance.

Le diagramme de séquence de la figure 5.15 montre les flux de contrôle et le passage des messages dans notre environnement de calcul parallèle. Une fois la plate-forme de calcul activée, le serveur démarre en premier puis les clients sont démarrés par un message du serveur. Les clients demandent chacun une connexion au serveur. Le serveur transmet un message de démarrage de l'exécution des algorithmes génétiques à tous les clients, après s'être assuré que tous les clients sont connectés. Chaque client exécute un algorithme génétique sur sa propre sous-population après avoir reçu le message de démarrage du serveur. Quand un client satisfait le critère d'arrêt, il informe le serveur et termine son exécution. Le serveur confirme que tous les clients ont terminé leur exécution et le processus se termine.

5.6 Expérimentations et évaluation des performances

5.6.1 Une stratégie de tests

Nous avons mené deux séries d'expérimentation afin de comparer les performances des deux algorithmes génétiques distribués avec et sans échange d'informations. La première série examine si une implémentation parallèle sans échange d'informations est capable d'améliorer la qualité de la recherche. La seconde série évalue la performance et l'avantage de l'utilisation du modèle distribué de l'algorithme génétique avec échange d'informations, mettant en œuvre notre nouvelle approche d'échange définie dans la section 4.3.2 du chapitre 4 et les différents protocoles d'interaction fournis par l'approche *client-serveur*. Les algorithmes génétiques parallèles avec et sans échange d'informations ont été mis en œuvre sur un réseau de PCs. Dans ces tests, nous avons utilisé un réseau de 11 PCs ayant *Suse Linux* comme environnement de calcul réparti. Sur une machine du réseau, nous avons instancié un serveur et sur chacune des autres machines, nous avons instancié un client de recherche. Un client de recherche reçoit la charge de la recherche (exécution d'un algorithme génétique (défini dans la section 5.3) sur une sous-population pour participer à la résolution collective d'un problème CARP). En outre, le serveur s'occupe de l'initialisation du paramétrage et de l'activation synchrone des machines de recherche (clients), de la coordination des échanges d'informations et de la publication des résultats des calculs.

Notre objectif est de comparer les échantillons des résultats obtenus avec et sans échange d'informations entre les machines. Pour notre étude nous avons choisi de comparer des échantillons de même taille n. Cela signifie que pour chaque cas de test, nous exécuterons n fois nos outils sans échange d'informations et n fois avec échange d'informations.

Les différents cas de tests sont identifiables par :

- **L'instance du problème à optimiser**,
- **Le nombre de machines utilisées**,
- **Le paramétrage de chaque machine (client)**,
- **Le paramétrage du serveur**.

5.6.2 Paramétrage de l'application

Notre contribution a pour but d'étudier l'impact de l'échange d'informations entre différents algorithmes génétiques. Les opérations génétiques étudiées précédemment nous permettront de paramétrer différemment plusieurs unités de calcul. Aux différentes opérations génétiques, il est nécessaire d'ajouter les paramètres fixant les probabilités avec lesquelles ces opérations seront appliquées. D'autres paramètres seront également mis en œuvre tels que la taille de la sous-population, le nombre de générations, l'autorisation de clonage ou encore l'élitisme qui consiste à préserver les meilleurs individus d'une sous-population. Les différents paramètres de notre algorithme génétique sont listés dans le Tableau 5.3.

C'est dans un fichier de configuration que nous définissons le comportement de notre algorithme puisque nous y fixons tous les paramètres. Dans les différentes séries de tests que nous avons conduites, nous avons fixé les sous-populations à une taille identique *PopSize*(taille de la population/nombre de clients) sans clone (*ClonesForbiden*=yes). L'avantage des sous-populations sans clone est une meilleure dispersion des solutions, ce qui favorise une bonne exploration de l'espace de recherche tout en évitant une convergence prématurée. En

Tab. 5.3 – Paramètres de l'algorithme génétique parallèle

Paramètre	Description
MachineNumber	Numéro du client de recherche
Run	Nombre d'exécutions à effectuer par un client de recherche
PopSize	La taille de la sous-population
NbGenerations	Nombre de générations
ClonesForbiden	Autorisation/interdiction des clones
SurvivalProportion	Proportion de survie
Pcrossover	Probabilité de croisement [0..1]
Pmutation	Probabilité de mutation [0..1]
CrossType	Type de croisement (Lox, Ox, X1)
MutationType	Type de mutation (Move, Swap)
ExportationCycle	Cycle entre deux exportations d'individus
Synchronization	Démarrage synchrone des clients de recherche
MaxGenPlate	Nombre maximum de générations entre deux demandes d'individus sans amélioration
MinGenBetweenImport	Nombre minimum de générations entre deux demandes d'individus sans amélioration
ConvergenceImport	Importation d'individus si une convergence prématurée est détectée
Pexportation	Taux de migration
ImportWindowSize	Taille de la fenêtre de migration
IndividualWait	Attente des individus demandés
Life_parametre	Paramètre de vie

conséquence, on peut se permettre d'appliquer les mutations (*Move, Swap*) avec des taux relativement élevés (10 à 40%), sans que la convergence des algorithmes génétiques soit trop rapide. Les croisements (*LOX, OX et X1*) ont été appliqués avec des taux variant entre 10 et 90%. Une stratégie élitiste (avec un taux de *SurvivalProportion*=15% de la sous-population) a été mise en place pour sauvegarder les meilleures solutions obtenues après chaque génération de l'AG. Le nombre de générations a été fixé à *NbGenerations*=5000 pour tous les clients de recherche. Afin de pouvoir comparer les performances des deux versions parallèles avec et sans échange d'informations, nous avons conduit deux séries de tests (*Run*=2). La première série de test consiste à lancer notre algorithme génétique distribué sans utiliser notre stratégie d'échange d'informations (parallélisation pure). À l'inverse, la deuxième série de test utilise cette dernière.

Les paramètres relatifs à la stratégie d'échange d'informations sont identiques pour tous les clients de recherche. La durée d'un cycle entre deux exportations d'individus a été fixée à *ExportationCycle*=10% du nombre de générations. Le nombre maximum et le nombre minimum de générations entre deux demandes d'individus par client de recherche sont fixés respectivement à *MaxGenPlate*=15% et *MinGenPlate*=10% du nombre de générations. Ainsi, lorsqu'une convergence prématurée est détectée par *ConvergenceImport*=yes, une importation d'individus sera sollicitée par le client de recherche. Les paramètres *Pexportation* et *Import-*

WindowSize ont été fixés à 20% de la sous-population, comme dans l'expérience menée par Kim (2002). Enfin, le démarrage des clients de recherche est synchronisé par *Synchronization*=yes et l'attente des individus sollicités par un client de recherche n'est pas bloquante (*IndividualWait*=no). Enfin, le paramètre dit de vie (*Life_parametre*) est initialisé à une valeur selon l'expérience.

Dans la suite de ce chapitre, nous appelons la machine qui héberge le serveur par *Console* et les clients par *machine de recherche locale*.

5.6.3 Mise en œuvre d'un robot de test

5.6.3.1 Spécifications

Nous avons tout d'abord spécifié les différentes fonctions auxquelles notre robot devait satisfaire pour répondre à notre besoin :

- **Compatibilité avec un pare-feu :** afin de ne pas avoir à reconfigurer toutes les machines utilisées avant et après chaque campagne de tests, nous avons souhaité adapter notre application au réseau de machines utilisé. En effet, les premiers essais effectués nous obligeaient à régler le pare-feu de chaque machine du réseau.
- **Contrôle du système depuis la Console :** pour les mêmes raisons nous souhaitons que notre système puisse être contrôlé et lancé depuis la Console (aucune intervention sur les machines de recherche locale ne devra être nécessaire).
- **Distribution des paramètres :** les paramètres doivent être envoyés depuis la Console pour chaque cas de test.
- **Distribution des problèmes :** Comme pour les paramètres, les problèmes doivent être envoyés depuis la Console pour chaque cas de test.
- **Externalisation de la génération des sous-populations initiales :** la population initiale peut avoir un impact important sur la recherche de solutions. Nous prévoyons donc dès maintenant d'externaliser sa génération sur la Console, ce qui nous permettrait d'étudier des cas de test où l'on maîtriserait les populations initiales de chaque machine.
- **Envoi des statistiques à la Console :** les statistiques doivent être accessibles directement sur la Console.
- **Synchronisation des machines :** les machines doivent être synchronisées à chaque exécution.

5.6.3.2 Modélisation du robot de test

Chaque algorithme génétique doit être initialisé avec une sous-population, des paramètres génétiques et des paramètres relatifs à l'échange d'informations. La figure 5.16 nous montre cette nouvelle modélisation.

Les premiers travaux prévoyaient non seulement des serveurs sur la Console mais également un serveur sur chaque unité de calcul (voir figure 5.17). Les serveurs hébergés sur la Console avaient en charge la réception des individus envoyés par les unités de calcul et la réception des demandes d'individus. Les serveurs présents sur les unités de calcul avaient pour objectif de réceptionner les individus remarquables envoyés par la Console. Or, la mise en place de serveurs sur les unités de calcul nécessite un réglage spécifique du pare-feu de chaque machine.

Un pare-feu permet de protéger une machine contre des connexions non souhaitées. Il est en général configuré pour autoriser les connexions ouvertes à l'initiative de la machine qu'il protège. Par opposition, le pare-feu d'une machine hébergeant un serveur doit être configuré pour autoriser les machines clientes à se connecter. C'est pourquoi nous avons modifié la structure de notre application pour que seule la Console héberge des processus serveur (voir figure 5.18).

Nous allons maintenant décrire le fonctionnement des deux systèmes client-serveurs.

Système client-serveur pour l'exportation Le processus *interfMachine*, lance un *thread* qui attend la présence d'un message dans la file BOITEEXPORT. Dès la lecture d'un message, une demande de connexion est faite au serveur de récupération des exportations et les différentes informations sont envoyées comme nous le montre le diagramme de la figure 5.19.

Système client-serveur pour l'importation De la même façon, le processus *interfMachine* attend la présence d'un message dans la file *BOITESGADEMANDE*. Dès la lecture d'un message, une connexion est faite avec le serveur d'importation, le nombre d'individus souhaité et leur type sont envoyés au serveur qui, en retour, renvoie les individus demandés. La figure 5.20 nous en montre le déroulement.

5.6.3.3 Le chargement des paramètres

Comme nous l'avons vu précédemment, les paramètres sont gérés par la Console pour être ensuite envoyés aux différentes unités de calcul. Chaque unité de calcul envoie une demande de *nb* individus à la Console, en plaçant un message de type *PARAMETRE* dans la file *SGA DEMANDE*. Il récupère ensuite les paramètres un à un dans la file de messages *IMPORT*[9]. Le diagramme de séquence de cet échange est représenté par la figure 5.21.

5.6.3.4 Chargement des sous-populations initiales

Dans cette nouvelle configuration, les populations initiales sont générées sur la machine Console par le processus *totalPop*. Ce processus fournit au processus *ExtractTrait* des individus qui sont ensuite envoyés à chaque unité de calcul.

Chaque unité de calcul pourra faire une demande de population en précisant le nombre d'individus souhaités. Nous utilisons pour cela les mêmes files de messages que pour le chargement des paramètres, comme nous le montre la figure 5.22.

5.6.3.5 Le système d'échange d'individus

Comme nous l'avons vu dans la section 5.6.3.2, notre application est basée sur une architecture client-serveur où seule la machine Console héberge des serveurs. Tous les échanges d'individus sont donc déclenchés à l'initiative des machines de recherches locales[10]. La figure 5.23 nous montre le séquencement de notre système.

[9] Afin de limiter le nombre de files de messages utilisées, les paramètres sont formatés comme des individus

[10] nous verrons dans la suite du document qu'il est possible de déclencher des échanges à l'initiative de la Console

5.6.3.6 Processus : ExtractTrait

Nous avons conservé le même scénario d'échange que celui décrit à la section 5.5.1. C'est-à-dire que la Console conserve les meilleurs individus de chaque machine. Le choix de la machine qui fournit les individus est fait au hasard parmi les autres machines. Les individus sont quant à eux choisis par la méthode de *la roue de la fortune biaisée*.

Afin de pouvoir exécuter un certain nombre de *run* sans échange d'informations puis avec échange d'informations, notre application doit être en mesure de modifier les paramètres des machines de recherche locale. Pour cela, nous avons mis en œuvre dans le processus *ExtractTrait* une procédure permettant la mise à disposition de paramètres. Après chaque *run*, chaque machine de recherche locale envoie une requête de demande de paramètres de contrôle à la Console. Si des paramètres sont à disposition, la machine de recherche locale envoie alors une demande de paramètres. Le reste du traitement est identique à ce que nous avons vu précédemment. Cette nouvelle fonctionnalité permettra également le contrôle par la Console des paramètres des machines de recherche locale.

5.6.3.7 Processus : totalPop

Le rôle du processus *totalPop* est de générer des individus qui seront ensuite utilisés par les machines de recherche locale. Ce processus est étroitement lié au problème que l'on souhaite optimiser. En effet, la génération des individus doit obligatoirement suivre des règles définies pour chaque problème. Dans le cas du *CARP* les individus doivent représenter une suite de tâches à parcourir (cf. 5.3.2).

Pour initialiser la population dans nos algorithmes génétiques, nous avons sélectionné les heuristiques Path-Scanning (Golden et Wong (1981)), Augment-Merge (Golden *et al.* (1983)) et l'heuristique d'Ulusoy (Ulusoy (1985)). Ces heuristiques sont en effet assez efficaces tout en étant facilement extensibles pour traiter des contraintes supplémentaires. Nous proposons dans l'annexe C des implémentations détaillées et non ambigües, ainsi que des analyses de complexité, tâches omises par les auteurs de ces heuristiques dans les articles de référence.

5.6.4 Tests et résultats

CSM-PGA a été entièrement programmée en C et exécutée sur un réseau de 11 PCs équipés tous d'une Suse Linux comme environnement de calcul réparti. L'évaluation numérique utilise deux ensembles de problèmes-tests pour le CARP, qui peuvent être téléchargés à l'adresse web `http://www.uv.es/belengue/carp.html`.

Le premier ensemble (fichiers *gdb*) contient les 25 instances de la thèse de DeArmon (1981). Les instances 8 et 9 sont omises par tous les auteurs car elles contiennent des erreurs (graphes non connexes). Les 23 problèmes restants ont de 7 à 27 nœuds et 11 à 55 arêtes. Toutes les arêtes sont à traiter : il s'agit donc de postiers chinois avec contrainte de capacité (CCPP). Le deuxième ensemble (fichiers *val*) comprend 34 CCPP plus durs construits par Belenguer et Benavent (2003), avec 24 à 50 nœuds et 34 à 97 arêtes.

Dans tous les tableaux, *Pb* donne la référence du problème. *N* et *M* indiquent respectivement les nombres des nœuds et d'arêtes. *LBB* donne la borne inférieure de Belenguer et Benavent (2003), fournie avec les instances. La colonne *Carpet* donne la meilleure solution obtenue par la méthode de recherche Tabou de Hertz et *al.* Hertz *et al.* (2000) après avoir

testé divers réglages de paramètres. *GA* donne la solution obtenue par l'algorithme génétique hybride de Lacomme *et al.* (2001). Les résultats obtenus par nos algorithmes parallèles avec et sans échange sont donnés respectivement dans les colonnes *CSM-PGA* et *PGA*.

TAB. 5.4 – Résultats obtenus par PGA et CSM-PGA sur les instances de DeArmon

Pb	N	M	LBB	Best	Carpet	GA	PGA	CSM-PGA
gdb1	12	22	316	316	316	316	316	316
gdb2	12	26	339	339	339	339	339	339
gdb3	12	22	275	275	275	275	275	275
gdb4	11	19	287	287	287	287	287	287
gdb5	13	26	377	377	377	377	377	377
gdb6	12	22	298	298	298	298	298	298
gdb7	12	22	325	325	325	325	325	325
gdb10	27	46	344	348	348	356	356	352
gdb11	27	51	303	311	317	311	317	303
gdb12	12	25	275	275	275	275	275	275
gdb13	22	45	395	395	395	395	395	395
gdb14	13	23	448	458	458	458	458	458
gdb15	10	28	536	544	544	544	544	540
gdb16	7	21	100	100	100	100	100	100
gdb17	7	21	58	58	58	58	58	58
gdb18	8	28	127	127	127	127	127	127
gdb19	8	28	91	91	91	91	91	91
gdb20	9	36	164	164	164	164	164	164
gdb21	8	11	55	55	55	55	55	55
gdb22	11	22	121	121	121	123	123	123
gdb23	11	33	156	156	156	156	160	156
gdb24	11	44	200	200	200	200	200	200
gdb25	11	55	233	233	235	235	235	233

5.6.4.1 Résultats sur les instances de DeArmon

Observons d'abord que nous obtenons de bonnes solutions aussi bien avec *PGA* qu'avec *CSM-PGA*. Les 23 instances de DeArmon ont toutes leurs arêtes à traiter. Elles sont moins grandes que les instances de Belenguer et Benavent. En outre, pour moins de 10% des problèmes résolus, *CSM-PGA* a surpassé *PGA* par un avantage moyen de 0.48% en terme de qualité de solution. Comme il peut être observé dans la figure 5.24, le facteur d'accélération du temps d'exécution moyen de *MAS-DGA* est de l'ordre de 60% (1.7 fois plus rapide) par rapport à *PGA*, malgré les coûts associés à la communication. Les temps moyens d'exécution en utilisant *PGA* et *CSM-PGA* sont respectivement de 45.7 et 25.7 secondes.

Tab. 5.5 – Résultats obtenus par PGA et CSM-PGA sur les instances de DeArmon

Pb	N	M	LBB	Best	Carpet	GA	PGA	CSM-PGA
1A	24	39	173	173	173	173	178	173
1B	24	39	173	173	173	173	181	173
1C	24	39	245	245	245	245	254	245
2A	24	34	227	227	227	227	235	227
2B	24	34	259	259	260	259	270	259
2C	24	34	455	457	494	462	479	457
3A	24	35	81	81	81	81	82	81
3B	24	35	87	87	87	87	89	87
3C	24	35	137	138	138	138	144	137
4A	41	69	400	400	400	400	441	400
4B	41	69	412	412	416	412	474	416
4C	41	69	428	428	453	428	473	428
4D	41	69	520	541	556	541	595	536
5A	34	65	423	423	423	423	481	423
5B	34	65	446	446	448	446	492	446
5C	34	65	469	474	476	474	524	476
5D	34	65	571	581	607	581	649	577
6A	31	50	223	223	223	223	245	223
6B	31	50	231	233	241	233	245	233
6C	31	50	311	317	329	317	332	317
7A	40	66	279	279	279	279	302	279
7B	40	66	283	283	283	283	306	283
7C	40	66	333	334	343	334	358	333
8A	30	63	386	386	386	386	440	386
8B	30	63	395	395	401	395	443	401
8C	30	63	517	528	533	533	592	528
9A	50	92	323	323	323	323	370	323
9B	50	92	326	326	329	326	336	327
9C	50	92	332	332	332	332	338	332
9D	50	92	382	391	409	391	413	391
10A	50	97	428	428	428	428	430	428
10B	50	97	436	436	436	436	440	438
10C	50	97	446	446	451	446	458	446
10D	50	97	524	535	544	535	560	536

5.6.4.2 Résultats sur les instances de Belenguer et Benavent

Ces 34 instances de grande taille (jusqu'à 97 arêtes) ont toutes leurs arêtes à traiter. Les solutions que nous obtenons avec *CSM-PGA* sont meilleures que celles de *PGA*. En outre, sur tous les problèmes résolus, *CSM-PGA* a surpassé *PGA* par un avantage moyen de 7% en terme de *fitness*. Le facteur d'accélération du temps de résolution moyen de *CSM-PGA* est de l'ordre de 12% (cf. figure 5.25) par rapport à *PGA*. Ainsi, on peut conclure que la communication

et l'échange d'individus entre les clients mis en application dans notre système *CSM-PGA* peuvent améliorer le processus dans son ensemble pour trouver des meilleures solutions par rapport à *PGA*.

5.6.4.3 Comparaison des résultats de CSM-PGA à la littérature

Les tableaux 5.4 et 5.5 résument les résultats d'exécution des approches proposées (*PGA, CSM-PGA*) par rapport aux meilleures méthodes publiées pour le CARP, à savoir la méthode CARPET (Hertz *et al.* (2000)) longtemps considérée comme la méthode de référence, l'algorithme génétique hybride de Lacomme *et al.* (2001) et la borne inférieure proposée par Belenguer et Benavent (2003). Sur les instances de DeArmon (1981), les résultats obtenus par *CSM-PGA* sont comparables à ceux des meilleures méthodes de la littérature sur 15 des 23 problèmes (70%). Pour les 8 autres problèmes (30%), la déviation au *Best* est respectivement de 2.03%, 1.65%, 0.64%, 1.5% et 1.72% soit 1.51% en moyenne. De manière similaire, *CSM-PGA* a fourni des meilleures solutions par rapport à *PGA* sur les instances de Belenguer et Benavent. En moyenne, sur les 34 instances, *CSM-PGA* fournit des résultats à 0,73% de la borne inférieure contre 2,3% pour la méthode CARPET (Hertz *et al.* (2000)) et 0,75% pour l'approche génétique de Lacomme *et al.* (2001).

5.6.4.4 Performances de CSM-PGA et effet du nombre de clients

Nous avons effectué des expériences qui divisent une population initiale de 800 individus en sous-populations de tailles égales entre les clients. Les figures 5.26 et 5.27 montrent respectivement le temps de traitement et le fitness par rapport au nombre de clients.

Dans la figure 5.27, on peut noter que la qualité de la solution s'améliore progressivement avec l'augmentation du nombre de clients (sous-populations). En effet, le nombre de clients (sous-populations) a une influence directe sur la diversité génétique de ce modèle parallèle. Un grand nombre de sous-populations implique une réduction du nombre d'individus par sous-population (réduction du nombre de gènes). Comme les clones sont interdits, les sous-populations deviennent hétérogènes et permettent d'explorer des parties très importantes de l'espace des solutions.

D'autre part, le temps d'exécution diminue avec l'augmentation du nombre de clients (cf. figure 5.26).

5.6.4.5 Performance de calcul pour différentes valeurs initiales du paramètre de vie

La figure 5.28 montre une fluctuation du fitness en fonction du nombre de clients utilisés puisque la valeur initiale du paramètre de vie est incrémentée. La fréquence de reconstruction (restructuration des sous-populations) et la quantité de communications ont été réduites par l'augmentation de la valeur du paramètre de vie. Mais la performance de la recherche génétique diminue quand la fréquence de reconstruction est très faible de sorte que la contribution de chaque sous-population comme une partie d'une grande population est détruite. Les premières expériences avec une valeur de paramètre de vie égale à 1000 indiquent que les sous-populations ne sont pas restructurées. Dans ce cas, la taille de chaque sous-population

diminue en augmentant le nombre de clients, en conséquent, les performances de recherche se détériorent.

Dans cette expérience, nous pourrions confirmer que le fitness des solutions n'est sensiblement pas chuté quand le paramètre de vie était à 100 ou moins comparé avec la version séquentielle de l'algorithme génétique qui ne divise pas la population initiale. Donc, la valeur initiale du paramètre de vie dans le modèle proposé a une grande tolérance pour la performance de la recherche génétique.

Nous pouvons également confirmer que la performance de la recherche est meilleure quand la valeur initiale de vie était environ 10.

5.7 Conclusion

Soulignons que nous sommes les premiers à concevoir un algorithme génétique parallèle pour le CARP de base !

Dans ce chapitre, nous avons développé le premier algorithme génétique parallèle pour le CARP de base. Il s'agit d'un modèle client-serveur avec délégation d'algorithmes génétiques.

Notre algorithme génétique parallèle a été testé sur 57 instances de la littérature ayant jusqu'à 50 nœuds et 97 arêtes. Il est toujours au moins aussi bon que les meilleures métaheuristiques publiées. 6 meilleures solutions connues sont améliorées et 3 nouveaux optima ont été découverts.

Cependant, nous avons utilisé le modèle « client/serveur », où les échanges se font par envoi de messages à travers le réseau, ce modèle possède l'inconvénient d'augmenter le trafic sur le réseau et exige une connexion permanente. Un second inconvénient est que ces architectures *Client-Serveur*, bien adaptées à des réseaux homogènes de stations de travail, le sont beaucoup moins vis à vis d'architectures de réseaux hétérogènes, incluant des réseaux distants, des machines départementales et éventuellement des mainframes.

Les recherches de ce chapitre ont fait l'objet de deux publications. La première version complète de l'algorithme génétique parallèle a été présenté au congrès avec actes ECCS'2007 (Belkhelladi *et al.* (2007)). Elle était évaluée uniquement sur les fichiers *gdb* et *val.* L'algorithme a été ensuite amélioré par l'utilisation de notre nouvelle stratégie d'échange d'informations dite de migration sélective (cf. 4.3.2). Ceci a permis de tester les grosses instances *egl.* Les résultats ont fait l'objet d'une communication au congrès IEEE CEC'2008, dont les actes ont été publiés chez IEEE (Belkhelladi *et al.* (2008)).

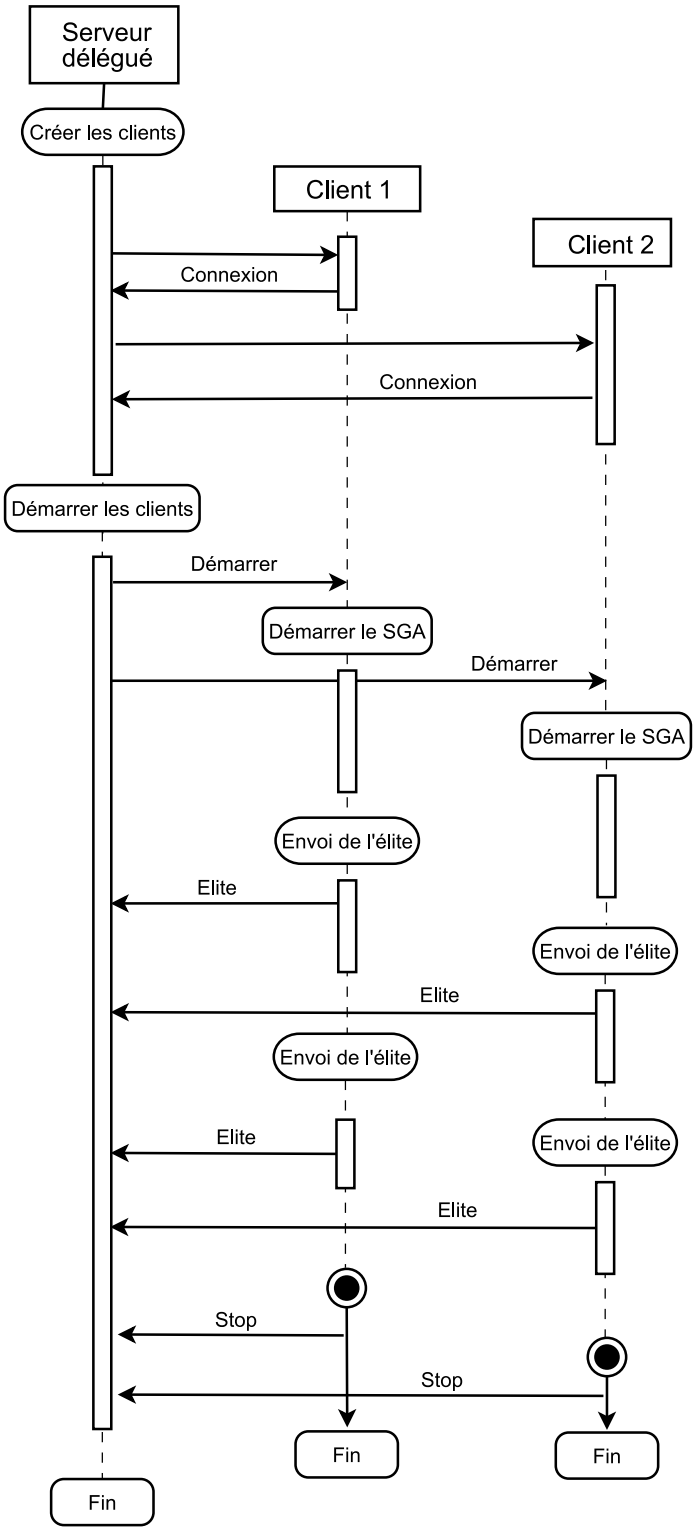


FIG. 5.15 – Diagramme de séquence du modèle de gestion déléguée (adapté à partir de (Kojima *et al.* (2000)))

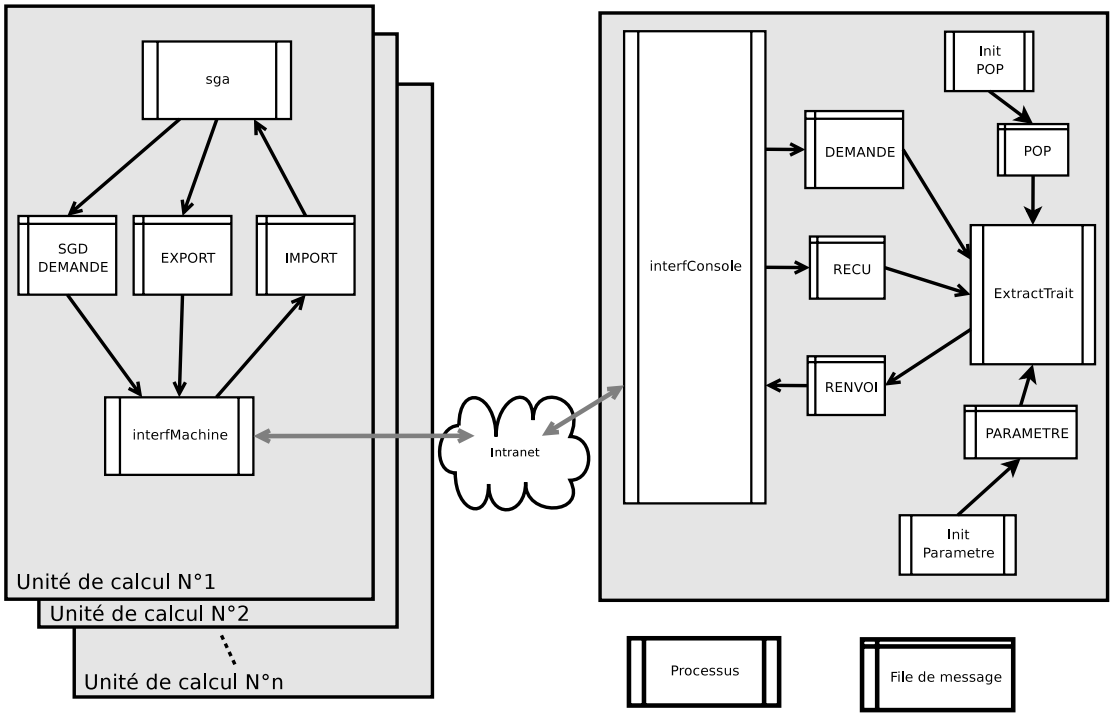


FIG. 5.16 – Structure de l'application de calcul réparti

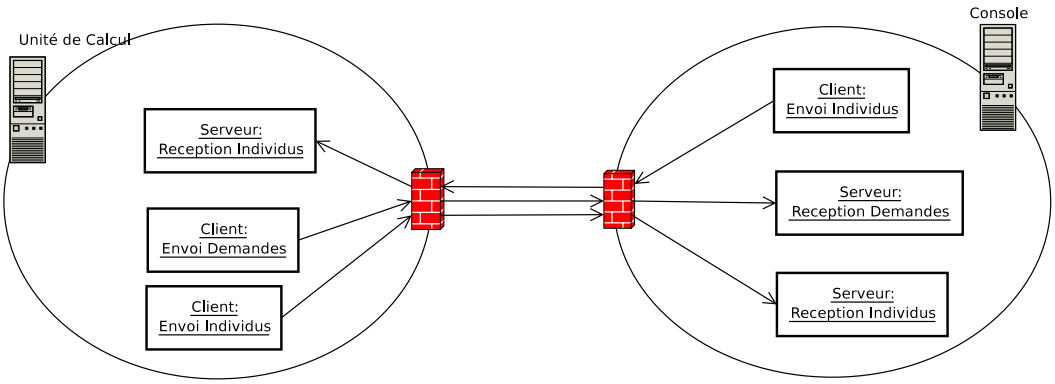


FIG. 5.17 – Système client-serveur avec serveur sur les unités de calcul

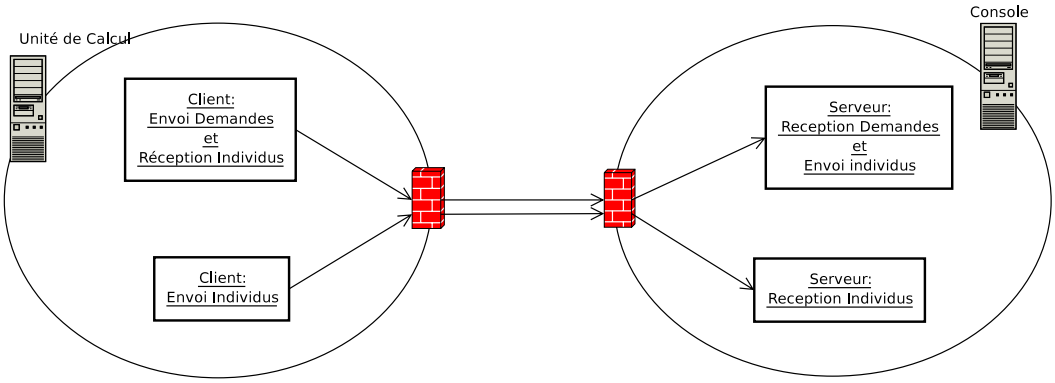


FIG. 5.18 – Système client-serveur sans serveur sur les unités de calcul

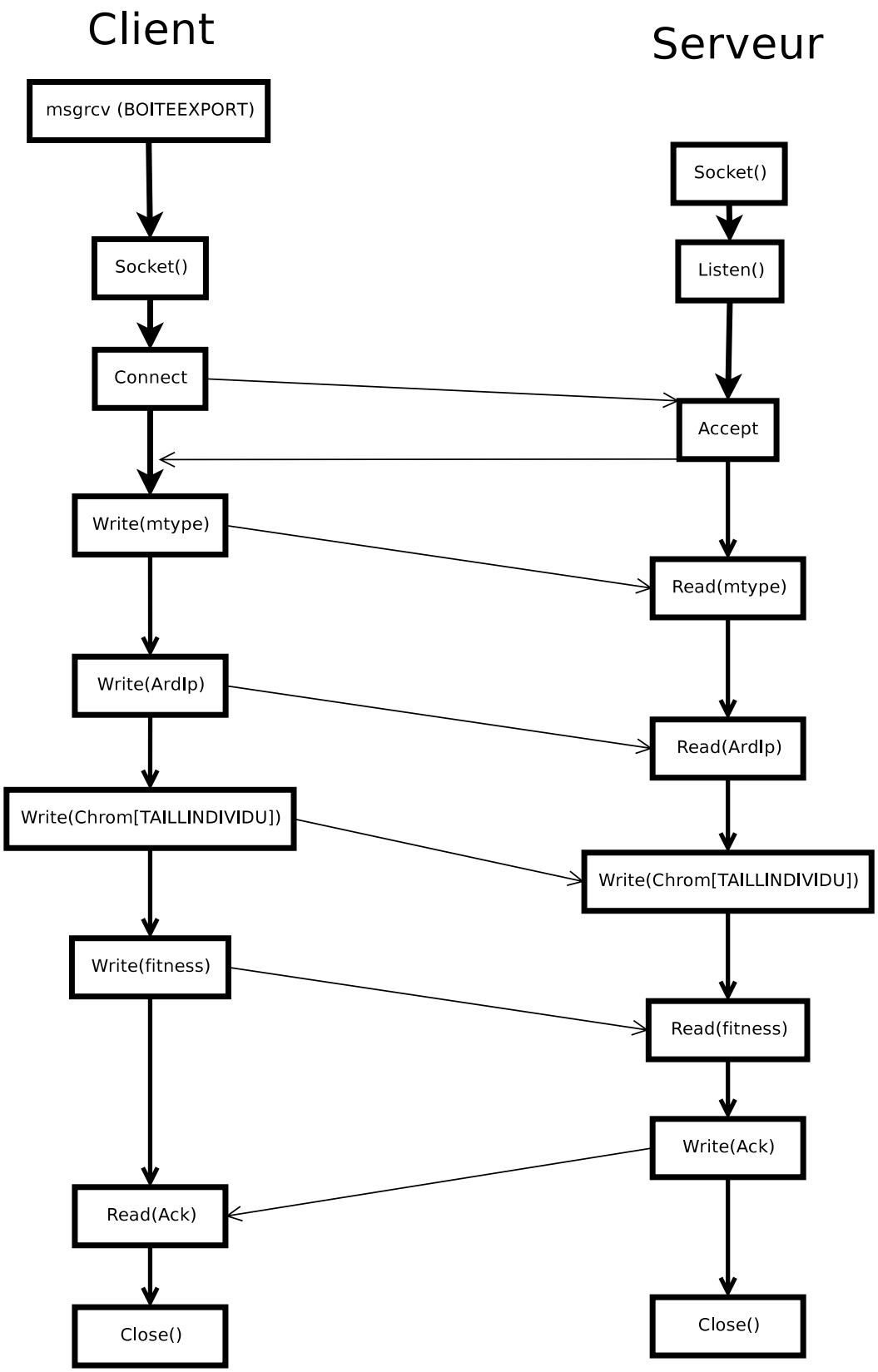


FIG. 5.19 – Communication entre processus lors d'une exportation

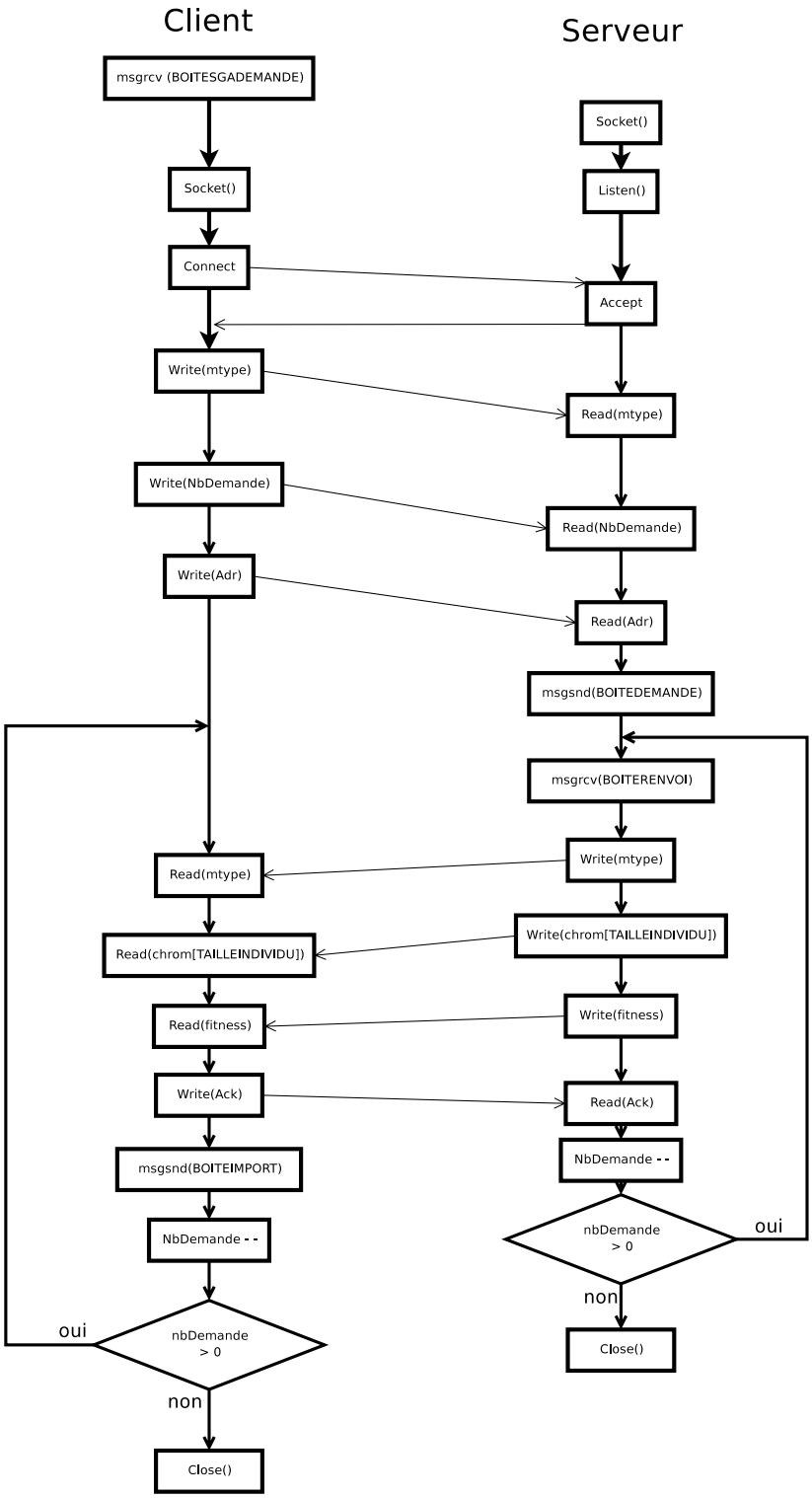


Fig. 5.20 – Communication entre processus lors d'une importation

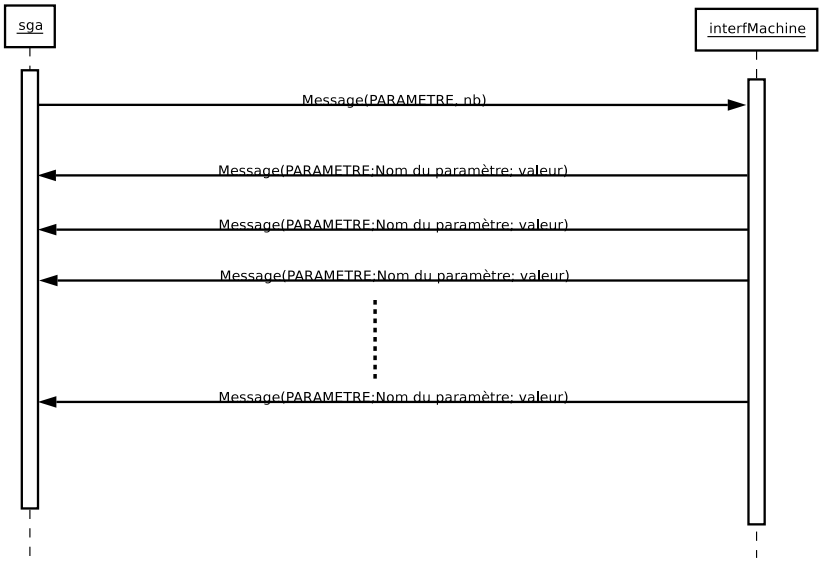


FIG. 5.21 – Diagramme de séquence du chargement des paramètres

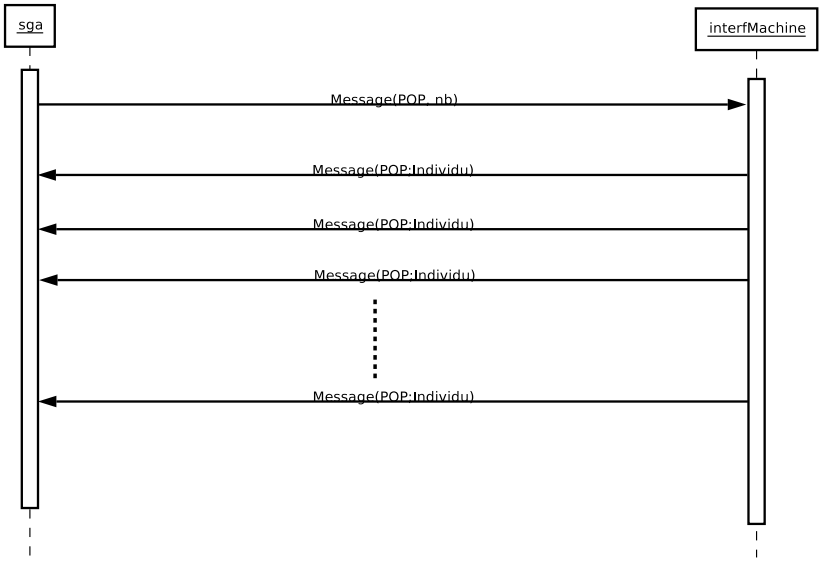


FIG. 5.22 – Diagramme de séquence du chargement de la population initiale

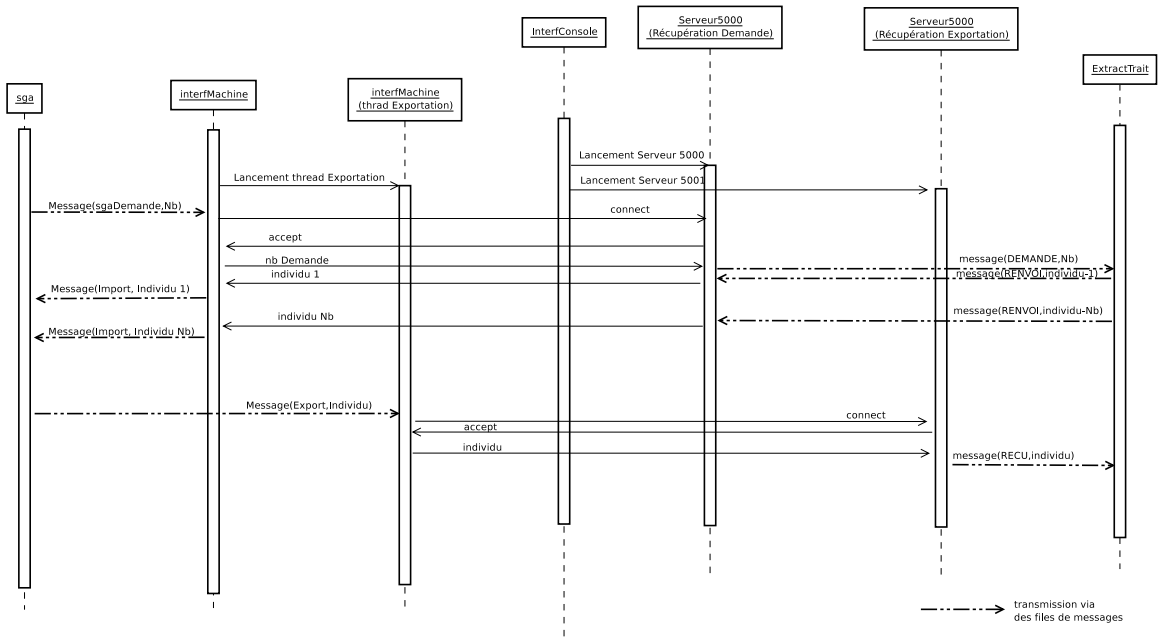


FIG. 5.23 – Diagramme de séquence de l'application

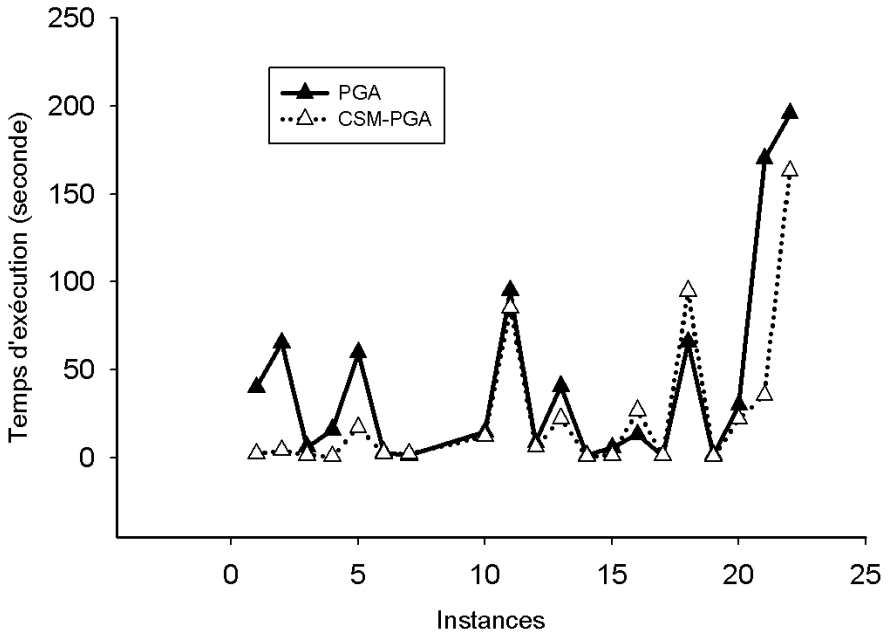


FIG. 5.24 – Temps d'exécution de PGA et de CSM-PGA sur les instances de DeArmon

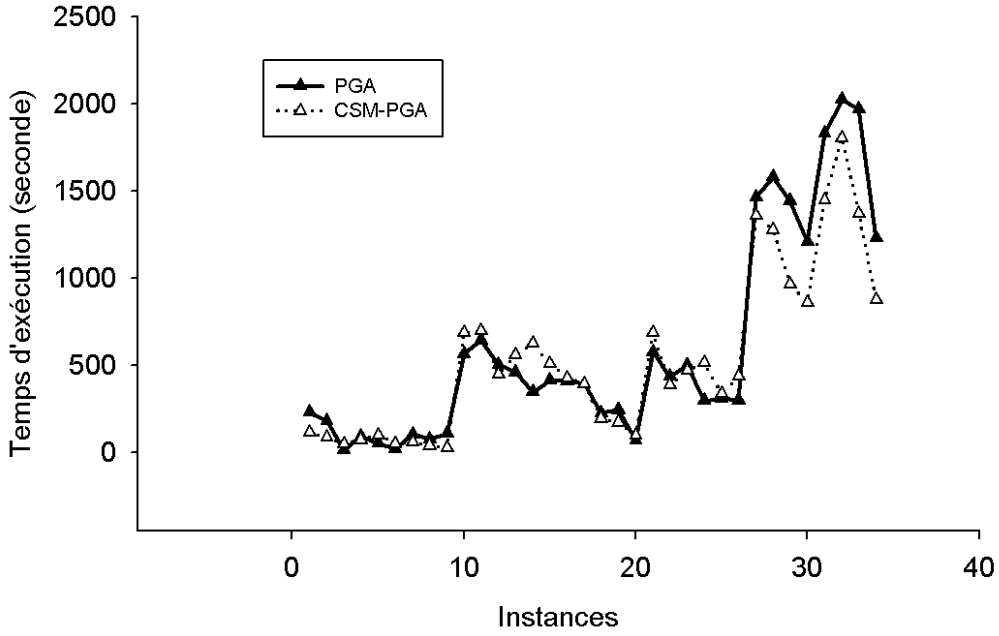


FIG. 5.25 – Temps d'exécution de PGA et de CSM-PGA sur les instances de Belenguer

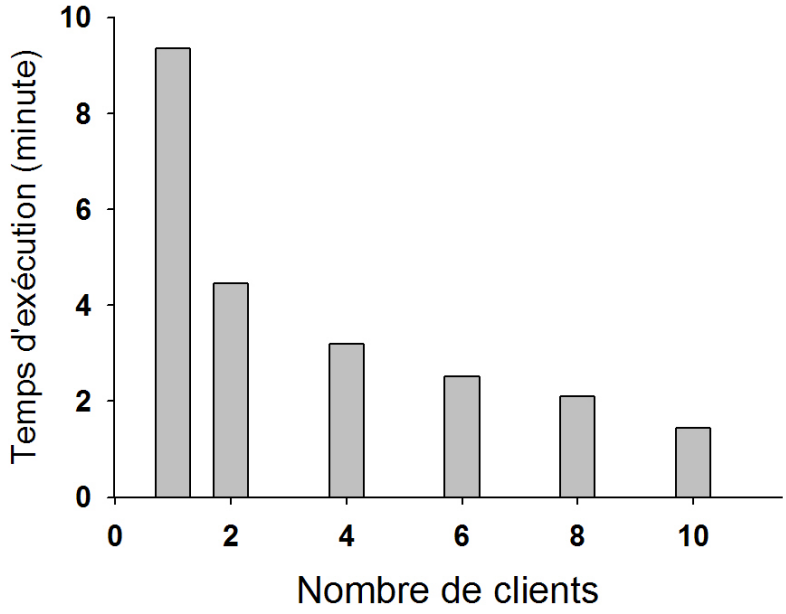


FIG. 5.26 – Effet du nombre de clients sur le temps d'exécution

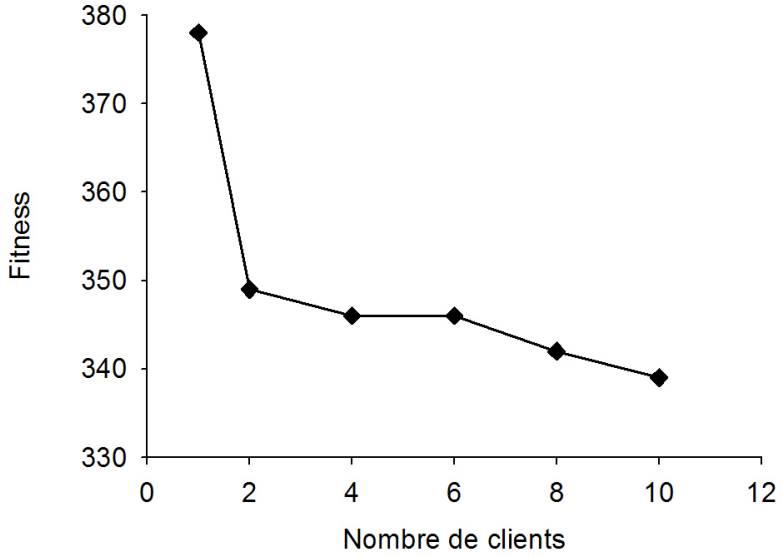


FIG. 5.27 – Effet du nombre de clients sur le fitness

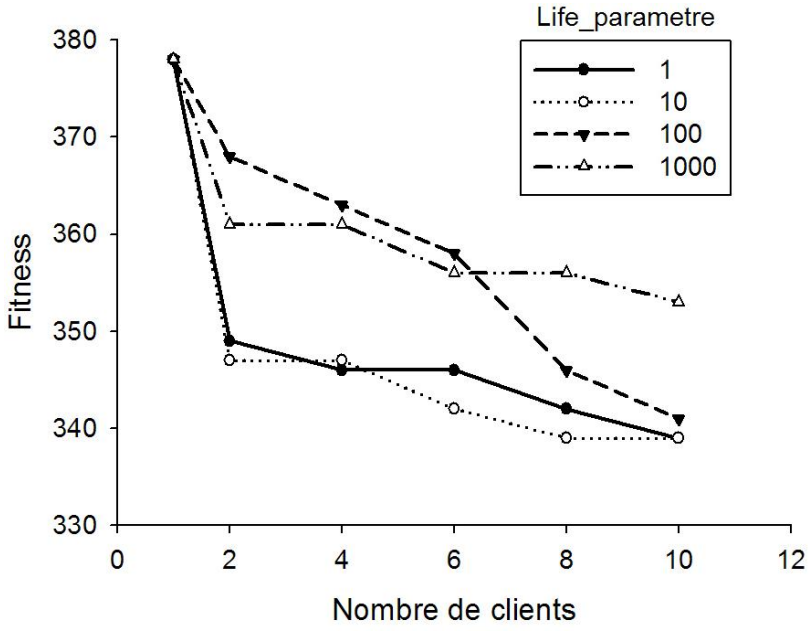


FIG. 5.28 – Effet de la valeur initiale de vie

Chapitre 6

MAF-DISTA : Une plate-forme d'agents mobiles pour le calcul distribué

Sommaire

6.1 Introduction

La mise en œuvre des algorithmes évolutionnaires parallèles nécessite des machines parallèles très coûteuses et qui ne peuvent pas être prévues dans tous les laboratoires de recherche et entreprises. Une solution alternative et prometteuse est de concevoir la plate-forme de calcul évolutionnaire parallèle sur un réseau classique de machines. Il existe quelques techniques pour faire fonctionner une telle plate-forme de calcul parallèle/distribué, par exemple un modèle client/serveur ou une approche basée sur des agents. Le modèle client/serveur où les échanges se font par des appels distants à travers le réseau est le modèle le plus utilisé à l'heure actuelle. Cependant, tous les composants sont stationnaires dans ce modèle par rapport à l'exécution. Ceci le rend inadapté, car aucune ressource spécifique de calcul n'est réservée et dédiée au calcul évolutionnaire parallèle, de sorte que les processus de calcul doivent se déplacer occasionnellement pour trouver des machines disponibles dans le réseau. Dans de telles circonstances, le paradigme d'agents mobiles offre un meilleur choix pour supporter le calcul parallèle (Lange et Oshima (1999), Macêdo et Silva (2005)). Les agents sont des processus possédant un contexte d'exécution, incluant du code et des données, pouvant se déplacer de machine en machine afin de réaliser les tâches qui leurs sont assignées. Ils ont été utilisés dans différentes applications qui se développent autour du réseau, notamment, dans le commerce électronique (Lee *et al.* (2003)), la recherche d'informations sur le web (Thati *et al.* (2002)), etc. De nombreuses plates-formes multi-agents sont publiquement accessibles telles que : AgentTCL, Jade, D'Agent, Tryllian, Tracy, etc. Les concepteurs d'applications peuvent utiliser ces plates-formes pour développer leurs logiciels dans le niveau dit d'« application » de ces dernières.

Dans ce chapitre, nous utilisons la plate-forme Jade et adoptons l'approche d'agents mobiles pour développer un modèle de calcul adaptatif pour paralléliser les algorithmes évolutionnaires. Nous présentons tout d'abord l'architecture adoptée à la conception. Nous décrivons ensuite les principales fonctionnalités de chacun des modules et les comportements des différents agents. Enfin, nous appliquons les résultats obtenus à l'optimisation du problème de tournées sur arcs décrit précédemment.

6.2 Choix de la plate-forme multi-agents

Le besoin de mettre en œuvre des systèmes à plusieurs composantes autonomes nécessite une infrastructure de logiciels utilisée comme environnement pour le déploiement et l'exécution d'un ensemble d'agents. Cette infrastructure est appelée *plate-forme de développement des systèmes multi-agents*. Cependant, l'implémentation de tels systèmes s'avère souvent difficile au niveau de la manipulation de structures de données complexes, de la distribution, de la communication ainsi qu'au niveau des contraintes matérielles imposées. De plus, l'intelligence artificielle est un domaine de recherche extrêmement riche et cette richesse induit une grande complexité et une grande multiplicité des approches proposées, ce qui conduit à de très nombreux modèles d'agents, d'environnement, d'interactions et d'organisations. Ces modèles sont souvent combinés au sein d'un même système multi-agents. Ainsi, le mieux est de choisir une plate-forme multi-agents adaptée aux contraintes du système à mettre en œuvre. Plusieurs plates-formes multi-agents existent, telles que MadKit, JADE, ZEUS, AgentBuilder, Jack, etc.

Pour la sélection de la plate-forme, nous avons souligné quelques critères qui nous semblent importants :

- La possibilité d'implémenter des systèmes relativement complexes ;
- *La flexibilité :* éviter les plates-formes qui supportent une méthodologie particulière ;
- L'accélération de développement grâce à la présence suffisamment importante de briques logicielles pour pouvoir produire une application aboutie ;
- Le traitement distribué et notamment la présence d'un support pour le paradigme d'agents mobiles ;
- L'existence d'une méthodologie couvrant les différentes étapes du développement ;
- *La facilité d'apprentissage et documentation :* la facilité d'apprentissage et de mise en œuvre des agents doit nous permettre de nous concentrer sur l'étude du rôle des différents agents que nous pourrons créer et sur l'étude de l'apport de l'échange d'informations ;
- *Licence d'utilisation :* nous souhaitons utiliser des logiciels *open-sources* pour leur gratuité et pour avoir accès aux sources (l'idéal étant une licence *open-source L-GPL*) ;
- *Déploiement :* le système doit prendre en compte le déploiement et la migration des agents dans un système distribué ;
- *Support pour la gestion du SMA :* il nous semble important que la plate-forme SMA possède des outils apportant, via une interface graphique par exemple, un support pour la gestion[1] des agents très utile lors des phases de mise au point ;
- *Facilité d'interconnexion avec d'autres plates-formes :* la plate-forme devra être compatible avec les spécifications de la FIPA, comme les *ACLmessage*[2].

Un résumé de certains éléments de comparaison des plates-formes est présenté dans le tableau 6.1. Dans ce tableau, nous évaluons pour chaque plate-forme les caractéristiques suivantes :

- ✣ sa philosophie de programmation des agents mobiles ;
- ✣ les principaux composants de la plate-forme ;
- ✣ si elle supporte le concept de proxy ou non,
- ✣ si les proxy continueront d'être valides lorsque les agents se déplacent ;
- ✣ si elle supporte les communications synchrones et/ou asynchrones ;
- ✣ si les agents peuvent communiquer par la transmission de messages entre eux mêmes ;
- ✣ si les appels distants (RMI-like) sont pris en charge ;
- ✣ si un agent peut spécifier une méthode de callback à exécuter en arrivant à sa destination (ou, au contraire, la même méthode prédéfinie est toujours exécutée) ;
- ✣ de savoir s'il est possible d'indiquer la cible d'un message/appel en spécifiant un nom convivial ;
- ✣ si elle permet d'indiquer la cible d'une circulation en spécifiant un nom convivial ;
- ✣ si elle est disponible en téléchargement (Licence) ;
- ✣ si elle est livrée avec des outils graphiques ;

[1] "Sniffer" de communication, outils de migration, envoi et réception de message
[2] ACL : *Agent Communication Language*

- le niveau d'activité associé à la plate-forme (sorti de nouvelles versions, mises à jour de son site Web, listes de diffusion, etc.) ;
- si elle offre des mécanismes de sécurité et, enfin,
- autres critères qui caractérisent également la plate-forme.

Les deux plates-formes qui ne spécifient aucune méthodologie et peuvent être considérées comme des « frameworks »[3], sont JADE et Aglets mais JADE l'emporte avec plusieurs autres caractéristiques intéressantes telles que la sécurité et l'existence d'un bon support de langages de contenu et d'ontologies.

Ainsi, pour le développement de *MAF-DISTA* et la simulation des résultats des approches de calcul distribué, nous avons choisi la plate-forme JADE[4] (Java Agent Development framework). JADE est un logiciel-médiateur[5] "middleware" qui permet une implémentation flexible des Systèmes Multi-Agents communiquant grâce à un transfert efficace des messages ACL (Agent Communication Language), conformes aux spécifications de la FIPA. JADE est codée en Java, supporte la mobilité, évolue rapidement et fait partie aujourd'hui des rares plates-formes multi-agents qui offrent la possibilité d'intégration des services pour les applications parallèles sur architecture à mémoire distribuée (agents des services réseau : RMA, DF et AMS). D'un autre côté, JADE tente de faciliter le développement des applications d'agents en optimisant les performances d'un système d'agents distribué.

Grâce à une conception des agents basée sur la notion de comportement, à une expressivité de la communication basée sur les ontologies et à ses outils d'aide au développement, JADE offre un environnement complet pour la réalisation d'applications réparties construites sur un ensemble d'agents mobiles.

Enfin, la méthodologie de développement du système choisi est celle conçue pour les systèmes multi-agents utilisant la plate-forme JADE ; il s'agit de la méthodologie proposée par Nikraz *et al.* (2006). Cette méthodologie, générique dans la phase de conception du système multi-agents, utilise fortement le formalisme UML et a l'avantage d'être clairement décrite et de proposer un exemple d'illustration.

6.3 Architecture générale de MAF-DISTA

MAF-DISTA est une plate-forme permettant de modéliser et de mettre en œuvre les algorithmes évolutionnaires parallèles sous forme d'un ensemble d'agents mobiles de recherche. Dans une instance de MAF-DISTA, chaque algorithme évolutionnaire est encapsulé dans un agent mobile, une entité autonome qui doit maintenir la connaissance de la recherche, l'apprentissage ou le problème à optimiser. Les agents se coordonnent entre eux selon un ensemble de règles stipulant les aspects topologiques et de communication, et ces règles peuvent être fixées a priori ou pendant l'exécution via une entité de coordination dite « agent de coordination ou de stratégie d'échange ».

[3] Un ensemble de bibliothèques permettant le développement rapide d'applications. Il fournit suffisamment de briques logicielles pour pouvoir produire une application

[4] http ://jade.tilab.com

[5] Permet la communication entre des clients et des serveurs ayant des structures et une implémentation différentes

TAB. 6.1 – Comparaison qualitative entre les plates-formes d'agents mobiles

Caractéristique	Aglets	Voyager	Grasshopper	Tryllian	JADE	Tracy	SPRINGS
Modèle	Événements	Procedurale	Procedurale	Tâches	Comportements	Procedurale	Procedurale
Éléments	Contextes, agents (aglets), Tahiti	Serveurs, Agents	Places, Régions, Agents	AFC, ADK, Habitats, Agents	Containers, Main-container, Plates-formes, Agents, DF, AMS, MTS	Agences, Agents, Plugins	Places, Régions (RNSs), Agents
Proxy	Oui	Oui	Oui	Non	Non	Non	Oui
Proxy dynamiques	Non	Oui (forwarding)	Oui (via le serveur de région)	Non	Non	Non	Oui (les mises à jour des emplacements)
Communications synchrones	Oui (impasse)	Oui	Oui	Oui (tâche d'envoi et de réception)	Non	Non	Oui
Communications asynchrones	Oui	Oui	Oui	Oui	Oui	Oui (avec la même agence)	Oui
Messages	Oui	Non	Oui (FIPA)	Oui (FIPA)	Oui (FIPA)	Oui (avec la même agence)	Oui
Appels distants	Non	Oui	Oui	Non	Non	Non	Oui
Rappels après les movements	Non	Oui	Non	Non	Non	Non	Oui
Appel/messages par nom	Non	Non	Non	Non	Oui (AID)	Oui (avec la même agence)	Oui
Movements par nom	Non	Non	Non	Oui (à travers des fichiers de configuration)	Oui (via AMS)	Oui (via TNS plugin)	Oui (dynamiquement)
Licence (téléchargement)	IBM Public License	Payant (version d'évaluation)	Non	LGPL	LGPL	Oui (binaires)	Oui (binaires)
Outils graphiques	Peu	Non	Oui	Oui	Oui	Non	Non
Niveau d'activité	Très faible	Élevé	Aucun	Moyen	Très élevé	Très faible	Élevé
Mécanisme de sécurité	Basique	Oui (gestionnaires de sécurité, etc.)	Basique	Oui (agents signés, etc.)	Oui (JAAS, etc.)	Oui (Bouncy Castle)	Basique (politiques)
Autres caractéristiques	ATP, Itinéraire	Multicast, Publish/subscribe, Aggregation dynamique	MASIF, FIPA, Multicast	FIPA, DNA	FIPA, Jess, JADEX, Support d'Ontologies	Lightweight, extensible, Kalong mobility model, Several migration strategies	No livelock, Schedule, Reliable, efficient

Sachant que l'objectif principal est d'élaborer une plate-forme de calcul évolutionnaire parallèle et sans utiliser des machines parallèles dédiées, nous avons choisi de mettre en œuvre un modèle parallèle à *gros-grain* sur un réseau d'ordinateurs dans notre laboratoire. Le réseau est considéré comme une machine de type MIMD[6] à mémoire distribuée (MIMD-DM) et sa disponibilité croissante en fait un environnement idéal et pratique pour paralléliser le calcul évolutionnaire. Nous avons élaboré une plate-forme basée sur des agents mobiles permettant de gérer de manière adaptative, l'exécution des différentes sous-populations ainsi que la communication entre elles. Cette plate-forme a deux principales caractéristiques : la topologie d'un système distribué sur une architecture réseau *pair-à-pair* et l'architecture de composants logiciels basée sur un paradigme d'agents mobiles. De cette façon, les utilisateurs finaux peuvent accéder aux ressources de calcul sans restrictions supplémentaires et les agents logiciels ne récupèrent que les machines libres pour effectuer leurs calculs.

Dans le chapitre 2, nous avons synthétisé une étude comparative des plates-formes logicielles dédiées à la résolution approchée de problèmes d'optimisation combinatoire. Cette étude nous a conduit à nous intéresser à l'élaboration d'une nouvelle plate-forme que nous avons appelée *MAF-DISTA*. *MAF-DISTA* se distingue des plates-formes existantes en introduisant les concepts permettant de traiter la distribution et l'adaptation dans les méta-heuristiques parallèles à base de population. L'environnement de calcul parallèle que nous proposons, offre plusieurs fonctionnalités et répond aux différentes contraintes qui sont les suivantes :

Exigences techniques

- La plate-forme devra être multi-plates-formes, c'est-à-dire portable sur n'importe quel système d'exploitation. On adoptera donc comme langage de programmation, le langage JAVA de Sun.
- La plate-forme devra être multi-agents, dans le but de lui donner de la fiabilité et de la flexibilité, et d'offrir au système de l'intelligence dans l'échange d'informations. On choisira donc la plate-forme JADE qui a été étudiée et préconisée dans la section 6.2.
- Le temps de calcul des solutions du problème du système sera au maximum égal à celui du système existant c'est-à-dire le CSM-PGA, décrit dans le chapitre précédent ; ainsi, le système réutilisera le code des l'algorithmes déjà codés.

Fonctionnalités générales de la plate-forme

La plate-forme MAF-DISTA devra :

- S'appuyer sur le format de fichier généré par l'initiateur de calcul pour résoudre le problème.
- Être modulaire, c'est-à-dire possède du code réutilisable pour des futures améliorations et modifications et dans d'autres environnement ;
- Posséder un module rapide de calcul des solutions du problème ;
- Disposer d'une *Console* de supervision des agents ;

Fonctionnalités spécifiques de la plate-forme

[6]Multiple Instruction stream, Multiple Data stream

- La plate-forme devra générer une population d'individus non identiques qui satisfait à la demande des agents de recherche ;
- La plate-forme devra gérer l'ensemble des machines pouvant accueillir les agents de recherche ;
- La plate-forme devra générer une liste de paramètres inhérents à l'algorithme de calcul, qui sera générée selon une certaine stratégie ;

Fonctionnalités offertes par la Console de supervision à l'utilisateur

La Console doit :

- Être unique à la plate-forme.
- Être sur la machine qui gère la plate-forme.
- Pouvoir créer de nouveaux agents et les détruire.
- Pouvoir suspendre l'activité des agents et la faire redémarrer.
- Pouvoir déplacer l'agent d'une machine à une autre.
- Pouvoir proposer différents modes de gestion des agents.

Fonctionnalités des agents

- Une catégorie d'agents doit remplir une seule tâche d'un service.
- Un ensemble distinct de catégorie d'agents différents doit remplir un seul service.
- Un agent doit pouvoir échanger des informations avec d'autres agents de manière autonome.
- Les échanges d'informations entre les différents agents devront être décentralisés.
- Les informations échangées entre les agents pourront être de nature diverses.
- Les agents seront autonomes, c'est-à-dire qu'aucune intervention extérieure à un agent ne sera nécessaire dans l'accomplissement de sa propre tâche.
- Les agents seront réactifs, c'est-à-dire qu'ils adopteront un certain comportement si un changement apparaît dans leur environnement.
- Les agents seront sociables, c'est-à-dire qu'ils coopéreront, donc échangeront des informations entre eux, pour accomplir leur service.
- Les agents exécutant des calculs devront stopper leur activité et éventuellement migrer vers une autre machine, si celle où ils sont exécutés devient utilisée.
- Les agents exécutant des calculs dont leur machine est déconnectée de la plate-forme, devront être stoppés jusqu'à ce qu'elle retrouve sa connexion. Après une trop longue attente, l'agent devra être détruit.

Fonctionnalités souhaitées par l'initiateur des calculs

L'utilisateur devra :

- Pouvoir lancer une campagne de calcul automatisée.
- Pouvoir consulter les statistiques de tous les agents (fitness, meilleures solutions, etc.).
- Disposer d'un affichage des solutions optimales (ex. sous la forme de nombre de tournées et conteneurs à vider lors de chaque tournée).
- Disposer d'une interface d'aide.
- Pouvoir gérer le parc des machines acceptant d'effectuer des calculs (ajout/retrait de machines, suspension du travail).

- Pouvoir visualiser l'état de toutes les machines du parc.
- Pouvoir visualiser l'état et la localisation de tous les agents.
- Pouvoir modifier la stratégie d'échange d'informations entre les agents mobiles qui effectuent les calculs.
- Pouvoir modifier les paramètres de l'algorithme de calcul sur chaque machine.
- Pouvoir demander au système de générer automatiquement pour chaque machine des paramètres de l'algorithme de calcul.

Les différentes contraintes définies ci-dessous nous ont conduit à établir un diagramme des cas d'utilisation (cf. figure 6.1),

Grâce à ce diagramme des cas d'utilisation, nous avons pu effectuer une identification des agents. À première vue, il s'agit de réunir les cas d'utilisation servant un même service, comme une responsabilité, et de les regrouper dans un agent.

Dans un premier temps, nous avons obtenu trois familles d'agents :

- **Les agents transversaux :** ce sont des agents uniques remplissant un seul service et qui servent à tous les autres agents. Ce sont en quelque sorte des serveurs. Nous verrons dans le point suivant les risques encourus par une trop grande centralisation des connaissances.
- **Les agents locaux :** ce sont des agents qui sont propres au conteneur.
- **Les agents mobiles :** ce sont des agents qui ont la faculté de migrer d'un conteneur à un autre de la plate-forme.

À l'issus du recoupement, nous avons identifié 9 types d'agents :
- Un agent qui lit un fichier de configuration du problème et de l'algorithme ;
- Un agent qui génère les paramètres des différentes instances de l'algorithme ;
- Un agent qui génère la population initiale et la met à disposition des autres agents ;
- Un agent qui gère le parc des machines ;
- Un agent qui gère la stratégie générale d'échange de connaissances ;
- Un agent qui surveille le statut des machines ;
- Un agent qui surveille le statut des autres agents de la plate-forme ;
- Un agent mobile qui effectue les calculs ;
- Un agent qui surveille le statut de la machine qui héberge le conteneur dans lequel il est situé.

À ce stade l'analyse, l'Agent Mobile de Calcul (AMC) était directement calqué de l'architecture de l'unité de calcul. Mais l'AMC s'avère de facto difficilement maintenable ; en effet, cet agent à lui seul a la charge de :

- Gérer une population locale, avec l'importation initiale auprès de l'Agent de Gestion de Population (AGPop) puis l'application des stratégies d'échange pour l'importation et l'exportation d'autres individus auprès des autres AMC ;
- Gérer les paramètres de l'algorithme, avec la demande de génération et de transfert des paramètres auprès de l'Agent de Gestion de Paramètres (AGParam) ;
- Récupérer auprès de l'Agent de Traitement du Fichier (ATF) la description du problème posé ;
- Récupérer le statut de la machine sur laquelle son conteneur est hébergé ;

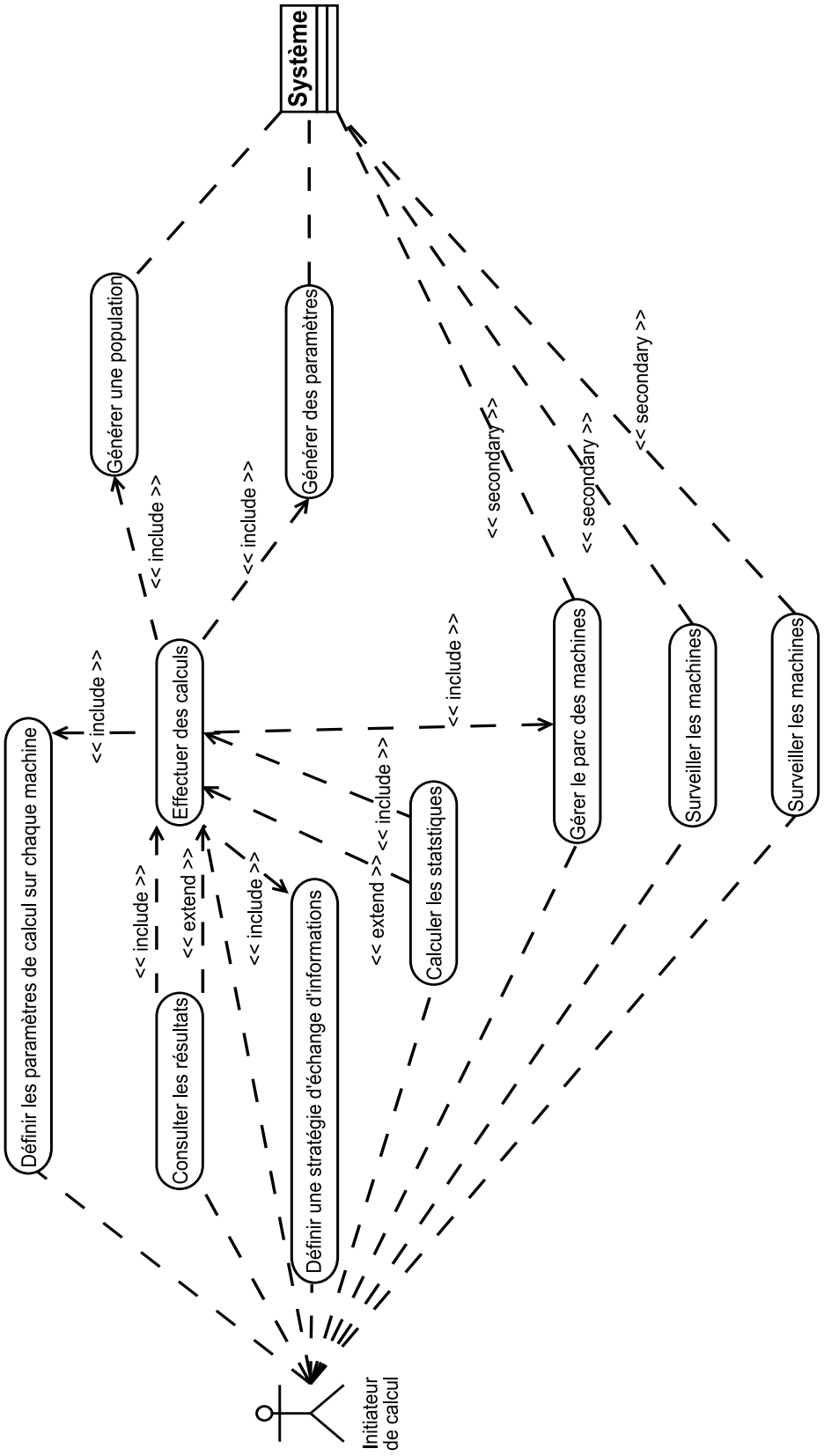


FIG. 6.1 – Diagramme des cas d'utilisation de MAF-DISTA

- Exécuter l'algorithme qu'il maintient en son sein ;
- Effectuer une action de post-traitement du résultat ;
- Maintenir en son sein un carnet d'adresse des autres AMC.

Selon l'API fourni par la plate-forme JADE, un agent est implémenté par un *thread JAVA*. Chacune des tâches ou responsabilités sont à implémenter sous JADE par des comportements (classe *Behaviour*) que doit exécuter l'agent ; ces comportements sont rangés dans une liste de comportements actifs, ce que la figure 6.2 illustre.

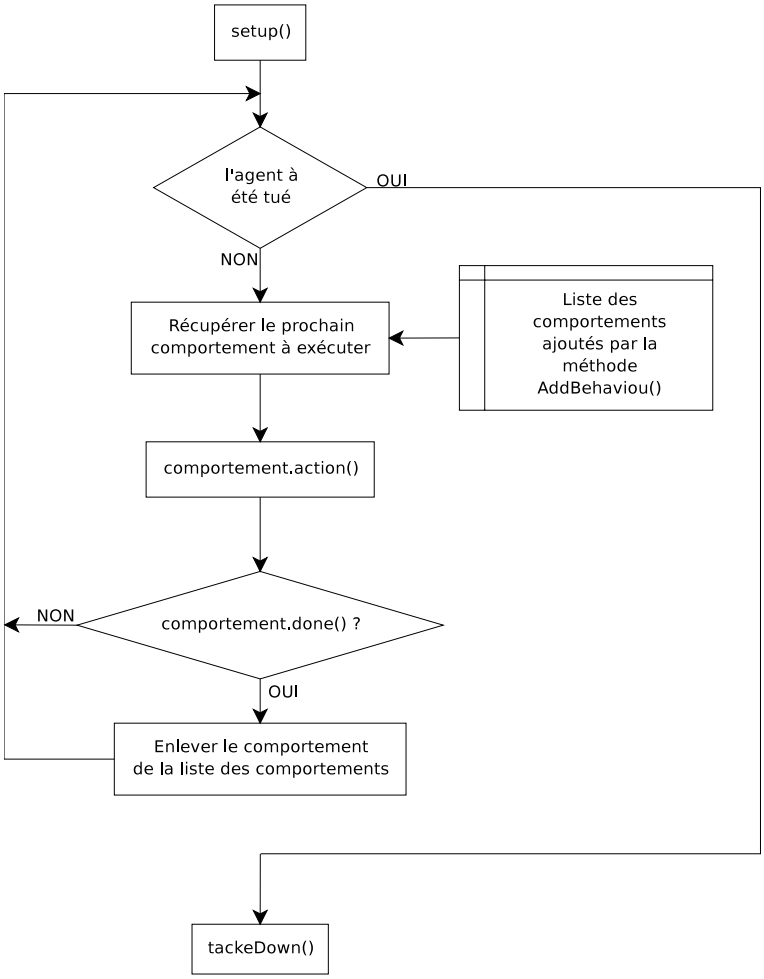


FIG. 6.2 – Exécution d'un agent Jade

Ainsi, un trop grand nombre de responsabilités implique :

- Une faible réactivité générale de l'agent ;
- Une complexité de conception de l'agent accrue ;
- Une évolutivité de l'agent très difficile à opérer ;
- Une modularité exécrable.

À contrario, si l'on décompose à outrance un agent, on s'expose à une complexification de conception générale du système et à un risque de lenteur de la plate-forme, d'une part

à cause de l'augmentation des agents, et d'autre part aux communications entre les agents induits par la décomposition.

Au vu des avantages et des inconvénients de la décomposition des agents, la modularité du système est primordiale ; il faut notamment que la plate-forme *MAF-DISTA* soit aisément adaptable à tous type de problème à résoudre et à tous type d'algorithme à employer. Nous avons donc opté pour la décomposition de l'agent mobile, initialement prévu, en trois agents mobiles et un agent propre au conteneur. Le résultat de cette décomposition est donné dans le tableau 6.2 :

Tab. 6.2 – Les agents mobiles

Agent	Rôles
Mobile de Calcul (AMC)	Récupérer des individus de sous-population gérée par son AMPop Récupérer les paramètres de son AMP Récupérer le statut de la machine Récupérer des pages jaunes auprès de l'AS situé sur sa machine Exécuter l'algorithme de résolution de problème Demander le déplacement dans un autre conteneur Gérer le déménagement de son AMP et de son AMPop Coordonner les activités de son AMP et de son AMPop Présenter les individus modifiés Présenter les résultats du calcul Présenter les statistiques Récupérer auprès de l'ATF la description du problème posé
Mobile de paramétrage (AMP)	Récupérer les paramètres de l'AGParam Permettre à l'utilisateur de modifier les paramètres avant son exécution Traiter les ordres de son AMC Présenter les paramètres
Mobile de Population (AMPop)	Récupérer une sous-population générée par l'AGPop Récupérer les stratégies d'échange d'informations de l'ASE Récupérer les individus modifiés de son AMC Récupérer des individus des autres AMPop Récupérer des pages jaunes les AMPop Effectuer une sélection des individus à modifier ou à importer Traiter les ordres de son AMC Présenter des individus selon la stratégie aux autres AMPop Présenter des individus à son AMC Demander de calculer de nouveaux individus à l'AGPop

6.3.1 Le problème de la centralisation des connaissances

Dans le système de calcul réparti présenté dans le chapitre précédent, le serveur, appelé aussi maître, est en charge de spécifier la politique d'échange des informations et de gérer ces échanges, ce qui rend le système vulnérable aux défaillances des unités de calcul (clients)

et du serveur. D'autant plus qu'une fois que les machines sur lesquelles les unités de calcul seront exécutées ont été définies, et que l'on a lancé le processus de résolution, on ne peut ni ajouter ni déplacer une unité de calcul, se qui rend le système statique dont les composants sont stationnaires.

C'est pourquoi nous avons choisi la plate-forme JADE pour faire évoluer le système *client-serveur*. Le choix est entièrement motivé par le fait de la décentralisation totale qu'offrent les systèmes multi-agents (SMA). Les agents communiquent entres-eux, par l'intermédiaire d'un média de communication procuré par la plate-forme. Mais les SMA sont des systèmes distribués asynchrones et répartis.

Ainsi, comme les agents n'ont qu'une connaissance partielle de leur environnement, ils sont incapables d'identifier les autres agents. C'est ainsi que sous JADE, et conformément aux normes de la FIPA, un service de pages jaunes (Directory Facilitator - DF) a été implémenté ; il est automatiquement lancé dès l'exécution de la plate-forme, en même temps que l'agent de gestion d'agents (Agent Management System - AMS) et un canal de communication entre agents (Agent Communication Channel - ACC).

6.3.1.1 L'agent des pages jaunes, une ressource critique

Le DF, unique dans notre cas à la plate-forme, est un agent indispensable de la plate-forme JADE qui remplit un service de pages jaunes. Chaque agent peut s'y inscrire, en fournissant à ce premier son nom, son adresse, le type de service qu'il offre, le langage de communication et l'ontologie.

L'agent est libre de s'y désengager à tout moment. Le DF concentre sur lui les services offerts par des agents du système. Ce service est sollicité à chaque fois qu'un agent recherche un service effectué par un autre pour converser.

Dans notre plate-forme, où l'échange d'informations, en particulier d'individus entre les agents mobiles de calcul entres-eux et les agents de traitement des résultats, est primordiale dans la résolution des problèmes, le DF devient une ressource critique, dans la mesure où :

- Il est soumis d'une part à un grand nombre de requêtes d'interrogation et d'enregistrement de service ;
- Et sans lui, les échanges d'informations entre les agents mobiles et locaux seraient impossibles.

C'est pourquoi il est stratégique et extrêmement important d'étudier les temps de réponses en fonctions de certains paramètres essentiels de la plate-forme, et pouvant à tout instant varier en fonction du choix de l'algorithme ou du problème, ou encore du nombre de machines constituant le réseau de résolution du problème. Ces paramètres sont :

- ❀ Le nombre d'entrées dans les pages jaunes ;
- ❀ La diversité des entrées dans les pages jaunes ;
- ❀ La fréquence d'interrogation d'un agent auprès du DF ;
- ❀ Le nombre d'interrogateur du DF.

Afin d'anticiper de mauvais résultats, une solution visant à pallier ce problème a été proposée : chaque agent se constitue un carnet de contacts pour chaque service dont il a besoin. L'agent des pages jaunes maintient ces différents carnets en diffusant les mises-à-jour nécessaires aux agents qui souscrivent ce service. Bien entendu, le service des pages jaunes

maintiendra toujours sa propre liste d'agents et leurs services offerts. Le DF dispose de ce service, implémenté de manière analogue ; il suffit d'y adhérer pour qu'il transmette à l'agent souscripteur les modifications sur les pages jaunes.

Le document d'étude de la charge du DF, consultable en annexe D, nous montre que :

- Le temps de réponse du DF augmente en moyenne linéairement. Le temps de réponse est en moyenne inférieur à une seconde pour un nombre d'entrée inférieur à 400 entrées, ce qui nous laisse une marche de manœuvre considérable.
- Le temps de réponse par nombre d'entrées diminue quand la diversité de celles-ci augmente. Ceci est dû au fait que le temps de recherche est constant pour un nombre d'entrées données ; le temps de réponse est alors conditionné par la constitution du message ACL de réponse.
- Le temps de réponse en fonction du nombre d'entrées reste quasiment inchangé quelque soit la période d'interrogation.
- Le temps de réponse augmente au fur et à mesure que le nombre des agents qui l'interroge augmente.

Ainsi, nous utiliserons le DF dans sa manière la plus classique ; un agent qui souhaite obtenir ou rafraîchir son carnet d'adresses local fera une demande au DF.

6.3.1.2 Les agents transversaux

Le même problème se pose pour les agents transversaux ; à ceci près qu'ils ne sont sollicités qu'une seule et unique fois, c'est-à-dire à chaque lancement de la plate-forme MAF-DISTA et à chaque nouveau calcul. Et du fait que ces agents remplissent un service unique de présentation ou de génération à la demande, comme le traitement d'un fichier de configuration, la génération des paramètres ou de population, il y a très peu de chance que ces agents connaissent une défaillance.

Néanmoins, comme l'architecture générale du SMA est une hiérarchie d'agents, la défaillance des agents transversaux peut mettre en échec tout le système.

La figure 6.3 montre l'architecture détaillée de MAF-DISTA. Dans cette figure, les différents groupement d'agents sont colorés différemment.

6.3.2 Vers un système standard

Une première version de MAF-DISTA a été conçue pour l'optimisation du problème CARP (cf. section 6.5 (tests et résultats)). Cependant, l'objectif à terme était de proposer une plate-forme multi-agents de calcul réparti pour l'optimisation de problèmes combinatoires, et qu'il serait plus rentable de proposer une librairie de connaissances génériques et d'agents standards afin de ne pas à avoir à concevoir de nouveau un nouveau SMA pour un nouveau type de problème ou pour un nouvel algorithme employé.

Par conséquent, il a fallut veiller à ce qu'elle soit entièrement paramétrable selon le type d'algorithme employé et le type de problème à résoudre. Le déploiement de cette dernière devra être donc rapide et simple à mettre en œuvre. Afin de rendre ce travail le plus naturel possible, nous nous sommes attaqués aux abstractions des connaissances du système actuel. Avant de déterminer quels agents et quels comportements allaient être génériques, nous devions déterminer un modèle abstrait des connaissances des agents.

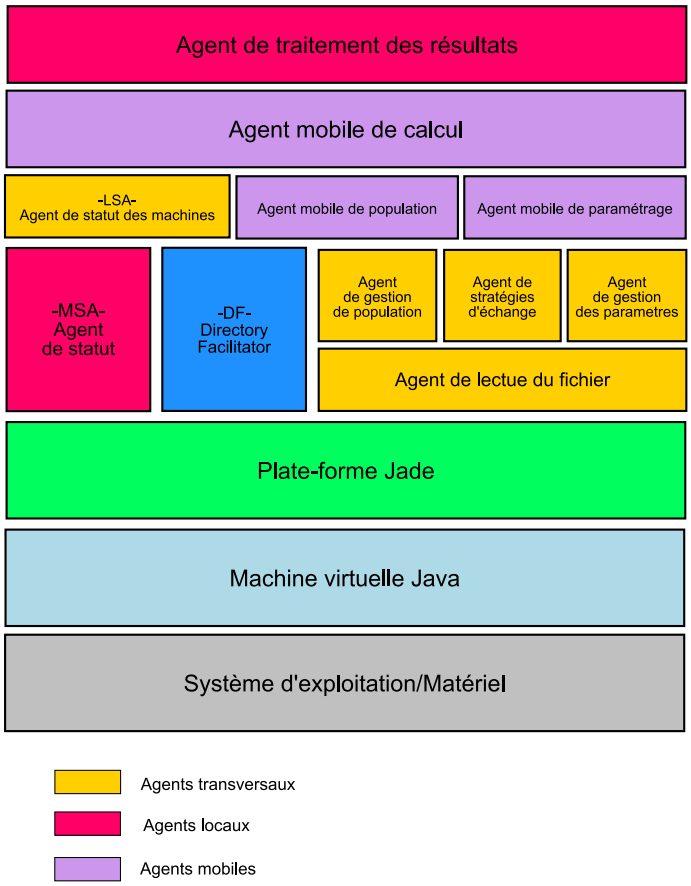


FIG. 6.3 – Architecture en couches de MAF-DISTA

6.3.2.1 Abstraction des connaissances

Comme dans tout processus d'abstraction de connaissances, il faut bien définir les objets sur lesquels nous travaillons ; Qu'est-ce exactement un algorithme ? Qu'entend-on par problème ? C'est ainsi que nous avons travaillé pour pouvoir abstraire notre modèle des connaissances actuel.

1. **Définitions des ressources**

   Un problème d'optimisation est communément défini comme un ensemble de tâches à réaliser, dans des conditions définies. Par ailleurs, on ne connaît pas de solution ou de méthode systématique de résolution : on sait quel est le but ou l'objectif à atteindre et le contexte. La résolution du problème consiste alors autant à inventer, ou définir, la méthode à appliquer, autrement dit l'algorithme, qu'à la mettre en œuvre, c'est-à-dire l'implémentation de l'algorithme.

2. **Analyse de la représentation des connaissances actuelles**

   L'analyse des connaissances du système dans sa phase de développement actuelle a donné le diagramme de classes de la figure 6.4. Bien que ce modèle ne crée pas d'entorse à la définition précédente, de nombreuses critiques peuvent être émises à la vue de ce

diagramme.

D'une part, une tâche est un concept très abstrait et nécessite qu'elle soit spécialisée lors de l'implémentation. De même pour le problème, compte tenu du fait que les tâches sont des notions abstraites. De plus, les conditions de réalisabilité d'une solution et son évaluation (ou niveau d'adaptation) sont propres au type de problème.

Quant à la classe algorithme, elle doit être nécessairement abstraite de part le fait qu'il faille systématiquement implémenter une méthode de résolution du problème. Outre ce fait, tous les algorithmes de résolution de problèmes ne sont pas à base de population ; il est donc conceptuellement faux d'associer un algorithme général à une population de travail. De plus, il n'est aucunement mentionné qu'un algorithme dispose d'une stratégie d'échange d'informations. Les échanges d'informations doivent être donc pris en charge par un composant dédié pour.

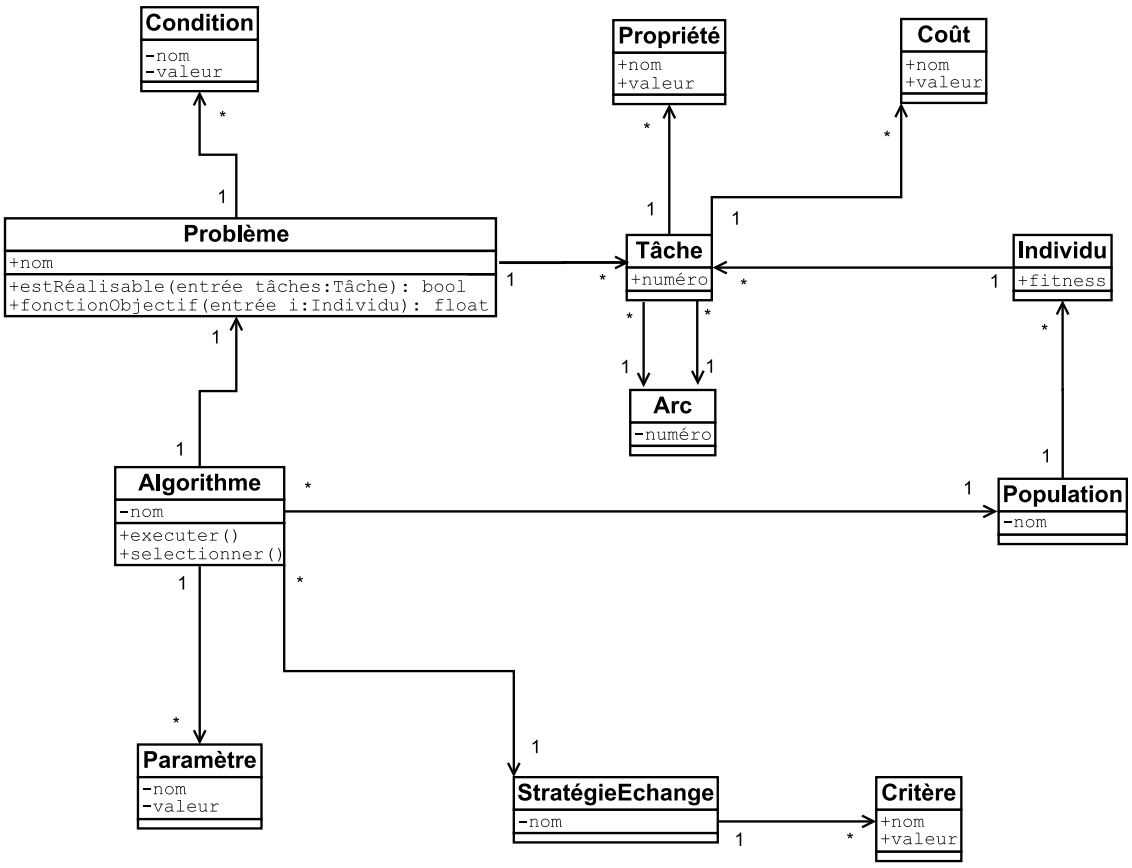


FIG. 6.4 – Diagramme de classes de MAF-DISTA

3. **Modélisation abstraite générale des connaissances**

À l'issue de notre réflexion, nous proposons le diagramme de classes donné dans la figure 6.5 ; il respecte à la lettre la définition donnée auparavant, tout en reprenant les idées du dernier.

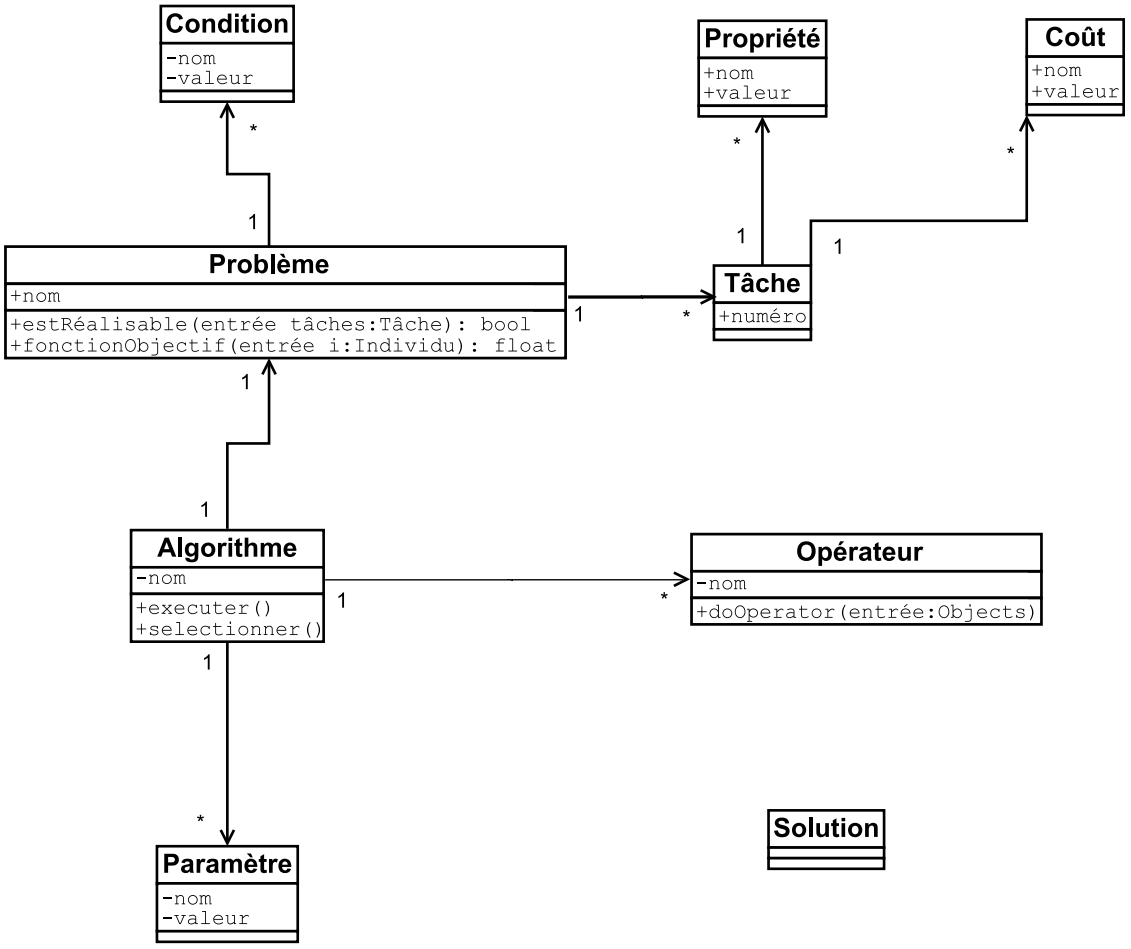


FIG. 6.5 – Diagramme de classes de MAF-DISTA après abstraction

6.3.2.2 Les agents et leurs rôles

Nous identifions trois types d'agents de manière générale dans notre plate-forme MAF-DISTA :

- Les agents standards
- Les agents spécifiques
- Les agents personnalisés

La standardisation des agents et de leurs comportements a été rendue possible grâce à la phase d'abstraction des connaissances échangées et maintenues par les agents. Cette standardisation concerne plus les agents au niveau des échanges d'informations, plus qu'au traitement. Ainsi, un agent standard possède un modèle de données sur lequel il travaille ; ce travail est à implémenter.

Agents standards

Les agents standards sont nécessaires au fonctionnement général du système et doivent

TAB. 6.3 – Les agents standards

Agents standards	Responsabilités
Gestion du Parc (AGP)[1]	Initialiser les machines spécifiées par l'utilisateur Présenter une liste de machines
Surveillance des Machines (ASM)[1]	Récupérer les statuts des machines des AS Récupérer tous les AS du FDA Présenter une liste des statuts des machines
Statut de la machine (AS)[2]	Surveiller la charge du processeur de la machine Surveiller la connectivité de la machine Présenter le statut de la machine
Mobile de paramétrage (AMP)[3]	Récupérer les paramètres de l'AGParam Permettre à l'utilisateur de modifier les paramètres avant son exécution Présenter les paramètres

[1] Agents uniques clairement identifiés, situés dans le conteneur principal de la plate-forme JADE
[2] Agents statiques uniques à une machine, i.e. à un conteneur
[3] Agents mobiles (plusieurs peuvent résider dans un conteneur)

être intégrés au futur SMA. Dans le tableau 6.3, vous trouverez la liste des agents standards fournis par la plate-forme. Ces agents étant standards, il n'y a, à priori, aucune modification à effectuer.

6.3.2.3 Agents spécifiques

Les agents spécifiques sont en général des agents qui sont soit algorithme dépendants, soit problème dépendants. La librairie propose des agents abstraits avec des comportements par défaut. Les futures personnes déployant la plate-forme *MAF-DISTA* auront donc la tâche de personnaliser ces agents en reprenant les comportements par défaut, autrement dit de spécialiser ces agents. Le tableau 6.4 présente les différents agents spécifiques de *MAF-DISTA*.

6.3.2.4 Agents personnalisés

Il se peut que les agents standards et spécifiques ne remplissent pas entièrement les fonctionnalités dont les futures personnes déployant la plate-forme auraient voulu disposer. Ainsi, la plate-forme met à disposition la librairie fournie par JADE, afin qu'ils puissent développer les agents souhaités. Il suffit de mettre en œuvre les comportements et les interactions entre les agents, comme le guide méthodologique fourni par JADE le montre. Ces agents ont aussi la possibilité de correspondre avec ceux de la plate-forme.

6.4 MAF-DISTA pour le CARP

Dans la suite de ce chapitre, nous nous intéressons à l'application de *MAF-DISTA* à l'optimisation des problèmes de tournées sur arcs. Nous reprenons donc les différentes définitions déjà fournies dans le chapitre précédent concernant le CARP. Chaque agents mobile de calcul dans *MAF-DISTA* encapsule une instance de l'algorithme génétique dont les différents composants sont donnés dans la section 5.3 du chapitre précédent. Il s'agit de l'ensemble des

TAB. 6.4 – Les agents spécifiques

Agent de Lecture du Fichier (ALF)	
Services principaux	Instancier le problème à résoudre à partir d'un fichier Instancier d'autres objets nécessaires
Dépendance	Problème et organisation du fichier
Responsabilités par défaut	Présenter la description du problème aux AMC en début de calcul
À implémenter	Lire et interpréter un fichier Format du fichier du problème
Agent de Génération de Paramétrage (AGParam)	
Services principaux	Gérer les paramètres de l'algorithme des AME
Dépendance	Algorithme
Responsabilités par défaut	Présenter des paramètres
À implémenter	Génération des paramètres Interface d'entrée des paramètres
Agent d'Exploitation des Résultats (AER)	
Services principaux	Traiter et formater les résultats des AMC Instancier d'autres objets nécessaires
Dépendance	Problème
Responsabilités par défaut	Récupérer les résultats de l'AME situé sur la même machine Récupérer du FDA l'AME situé sur sa machine Présenter les résultats du calcul traités Récupérer auprès de l'ATF la description du problème posé
À implémenter	Formatage des résultats
Agent Mobile d'Exécution (AME)	
Services principaux	Exécuter l'algorithme de résolution du problème
Dépendance	Algorithme
Responsabilités par défaut	Récupérer les paramètres de son AMP Récupérer le statut de la machine Récupérer du FDA l'AS situé sur sa machine Demander le déplacement dans un autre conteneur Gérer le déménagement de son AMP Présenter les résultats du calcul Récupérer auprès de l'ATF la description du problème posé
À implémenter	Formatage des résultats

opérateurs de croisement (LOX, OX et X1), des opérateurs de mutation (MOVE et SWAP), de la sélection par la roue de la fortune biaisée. Ainsi, les différentes heuristiques définies dans l'annexe C ont été utilisées pour la génération de la population initiale. La section suivante décrit l'ensemble des résultats obtenus sur des instances de CARP.

En ce qui concerne l'échange d'informations entres les agents, nous avons utilisé les deux stratégies de communication définies dans le chapitre 4. Il s'agit de la troisième stratégie d'immigration et intégration sélectives et de la quatrième stratégie consistant en l'ajustement des paramètres d'un AE[7] par le biais d'échange.

La figure 6.6 montre l'interface graphique d'un agent mobile de calcul de *MAF-DISTA*.

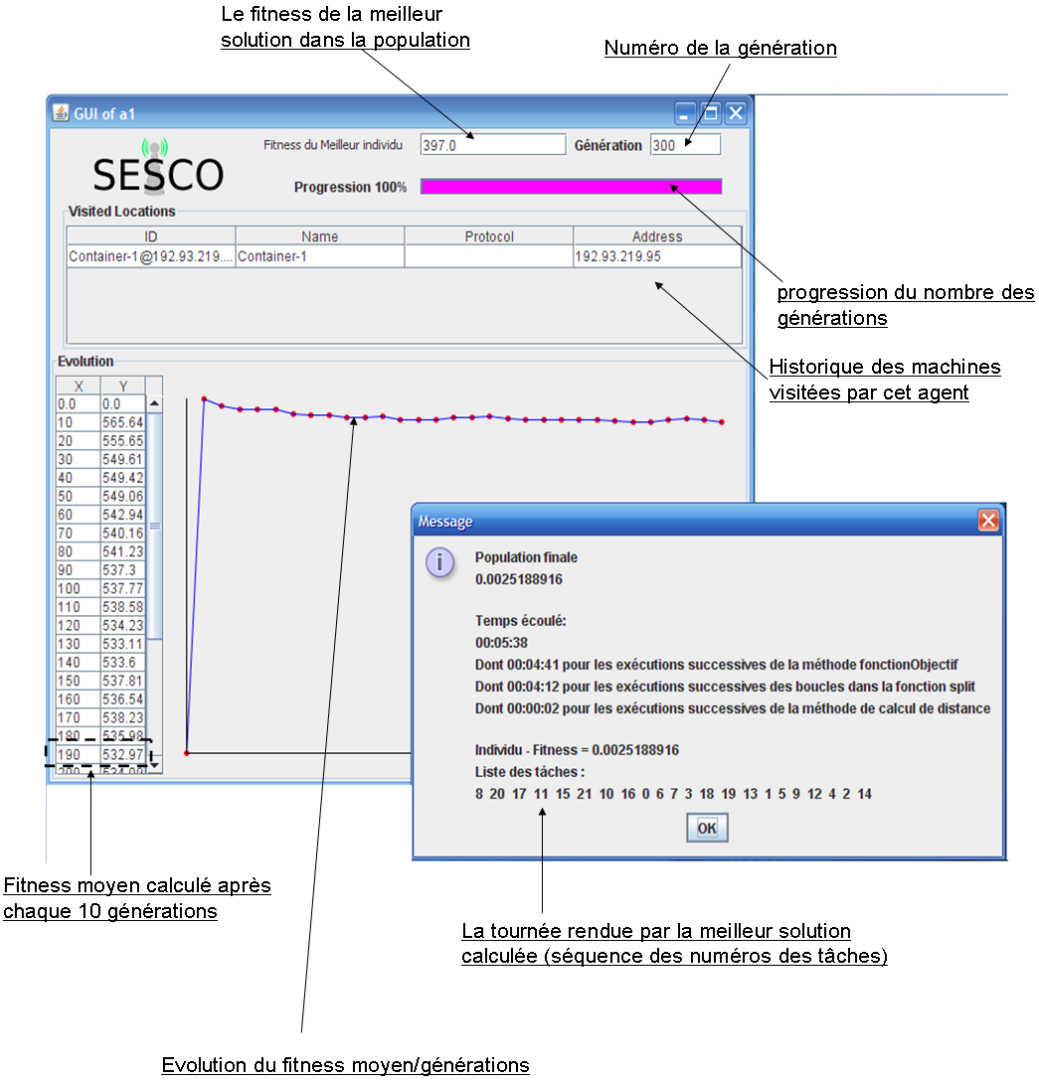


FIG. 6.6 – Interface graphique de l'agent mobile

[7] Algorithme Évolutionnaire

6.5 Simulations et résultats

MAF-DISTA a été entièrement programmée en Java et exécutée sur un réseau de 11 PCs équipés tous d'une Suse Linux comme environnement de calcul réparti. L'évaluation numérique utilise trois ensembles de problèmes-tests pour le CARP, qui peuvent être téléchargés à l'adresse web `http://www.uv.es/belengue/carp.html`.

Le premier ensemble (fichiers *gdb*) contient les 25 instances de la thèse de DeArmon (1981). Les instances 8 et 9 sont omises par tous les auteurs car elles contiennent des erreurs (graphes non connexes). Les 23 problèmes restants ont de 7 à 27 nœuds et 11 à 55 arêtes. Toutes les arêtes sont à traiter : il s'agit donc de postiers chinois avec contrainte de capacité (CCPP). Le deuxième ensemble (fichiers *val*) comprend 34 CCPP plus durs construits par Belenguer et Benavent (2003), avec 24 à 50 nœuds et 34 à 97 arêtes. Le troisième et dernier ensemble, dit « d'Eglese » (fichiers *egl*) contient 24 instances de grande taille conçues en fait par Belenguer et Benavent (2003), avec 77 à 140 nœuds et 98 à 190 arrêtes. Elles sont basées sur des portions du réseau routier rural du comté de Lancaster (Royaume-Uni), étudié par Eglese (1994) pour la planification des opérations de sablage en cas de verglas. Belenguer et Benavent (2003) en ont tiré 24 fichiers en faisant varier la capacité des véhicules et le pourcentage d'arêtes à traiter. Certaines de ces instances sont de vrais CARP, dans lesquelles certaines arêtes ne sont pas à traiter.

Dans ces expériences, onze machines étaient connectées à la plate-forme JADE dont une seule machine a été réservée pour l'initialisation des calculs évolutionnaires. Cette dernière est utilisée pour héberger le conteneur principal « Main-Container » pour permettre la création des agents par défaut AMS, DF et RMA. Ainsi, les utilisateurs finaux sont autorisés à utiliser les dix autres machines. Chacune de ces machines est un conteneur « Container » qui comprenait les différents agents mobiles et un agent de statut local (LSA).

Nous avons conduit trois séries d'expériences. La première série consiste à examiner si la plate-forme *MAF-DISTA* peut être utilisée pour exploiter la puissance de calcul du réseau pour accélérer le calcul évolutionnaire. Par conséquent, les agents sont stationnaires, ils ne migrent pas entre les différentes machines. Pour comparer l'effet d'utiliser différents nombres de machines, nous avons mené des expériences sur une sous-population de 800 individus, deux sous-populations de 400 individus, quatre sous-populations de 200 individus, six sous-populations de 133 individus, huit sous-populations de 100 individus et dix sous-populations de 50 individus. La population initiale est la même pour toutes les expériences. Le nombre de générations a été fixé à 5000 pour tous les agents mobiles de calcul.

La deuxième série d'expériences consiste à montrer l'intérêt de l'utilisation du parallélisme adaptatif de la plate-forme *MAF-DISTA*. Dans cette expérience, les agents mobiles ne sont pas stationnaires, ils peuvent migrer d'une machine à l'autre pour exploiter les ressources de calcul de façon optimale.

Enfin, la troisième série d'expériences permet d'examiner l'effet de l'utilisation de nos stratégies d'échange d'informations. Pour Ceci, nous avons utilisé les différents opérateurs génétiques définis pour le CARP. Il s'agit des trois opérateurs de croisement (LOX, OX et X1) et des deux opérateurs de mutation (MOVE et SWAP). Dix valeurs pour le taux de croisement ont été utilisées, allant de 0.1 à 0.99 par incrément de 0.1. En outre, dix valeurs pour le taux de mutation ont été utilisées variant de 0.1 à 0.55 en incrément de 0.05. Les clones (solutions identiques) ont été interdits dans chaque sous-population, pour avoir une meilleure

dispersion des solutions et diminuer le risque de la *convergence prématurée*. Chaque fois que la migration se produit, la taille de la fenêtre dynamique de migration varie aléatoirement entre 1 et θ. La valeur de θ a été générée aléatoirement dans les 20% de la taille de la sous-population. Le seuil de qualification au visa et résidence a été fixé au fitness moyen de la sous-population.

Dans tous les tableaux, *Pb* donne la référence du problème. N et M indiquent respectivement les nombres des nœuds et d'arêtes. *LBB* donne la borne inférieure de Belenguer et Benavent (2003), fournie avec les instances. La colonne *Carpet* donne la meilleure solution obtenue par la méthode de recherche Tabou de Hertz *et al.* (2000) après avoir testé divers réglages de paramètres. Dans les tableaux 6.5 et 6.6, *GA* donne la solution obtenue par l'algorithme génétique hybride de Lacomme *et al.* (2001). Dans le tableau 6.7, *MA* donne la solution obtenue par l'algorithme mémetique de Lacomme *et al.* (2004). Les résultats obtenus par nos algorithmes parallèles avec et sans échange sont donnés respectivement dans les colonnes *MAF-DISTA* et *PGA*.

6.5.1 Résultats sur les instances de DeArmon

Observons d'abord que nous obtenons de bonnes solutions aussi bien avec *PGA* qu'avec *MAF-DISTA*. Les 23 instances de DeArmon ont toutes leurs arrêtes à traiter. Elles sont moins grandes que les instances de Belenguer et Benavent. En outre, pour moins de 10% des problèmes résolus, *MAF-DISTA* a surpassé *PGA* par un avantage moyen de 0.48% en terme de qualité de solution.

6.5.2 Résultats sur les instances de Belenguer et Benavent

Ces 34 instances de grande taille (jusqu'à 97 arêtes) ont toutes leurs arêtes à traiter. Les solutions que nous obtenons avec *MAF-DISTA* sont meilleures que celles de *PGA*. En outre, sur tous les problèmes résolus, *MAF-DISTA* a surpassé *PGA* par un avantage moyen de 7% en terme de *fitness*.

6.5.3 Résultats sur les instances d'Eglese

Contrairement aux instances de DeArmon et de Belenguer et Benavent, pour certains fichiers, toutes les arêtes ne sont pas requises. Une colonne *M/2* a donc été ajoutée au tableau pour indiquer le nombre d'arêtes. La colonne T indique les arrêtes requises. Les résultats de *Carpet*, non publiés, ont été calculés par Mittaz (2000) à la demande de Belenguer et Benavent.

Les fichiers *egl* sont encore plus durs que les *val* : la borne *LBB* (qui n'est jamais atteinte). La raison pour la non-atteinte de la borne est due à la propriété du graphe. Selon Belenguer et Benavent (2003), le graphe partiel des arêtes à servir est parfois non connexe et leur borne, qui n'exploite pas cette propriété, pourrait probablement être raffinée.

Les solutions que nous avons obtenues avec *MAF-DISTA* sont meilleures que celles de *PGA*. En outre, sur tous les problèmes résolus, *MAF-DISTA* a surpassé *PGA* par un avantage moyen de 5.6% en terme de *fitness*.

TAB. 6.5 – Résultats obtenus par PGA et MAF-DISTA sur les instances de DeArmon

Pb	N	M	LBB	Best	Carpet	GA	PGA	MAF-DISTA
gdb1	12	22	316	316	316	316	316	316
gdb2	12	26	339	339	339	339	339	339
gdb3	12	22	275	275	275	275	275	275
gdb4	11	19	287	287	287	287	287	287
gdb5	13	26	377	377	377	377	377	377
gdb6	12	22	298	298	298	298	298	298
gdb7	12	22	325	325	325	325	325	325
gdb10	27	46	344	348	348	356	356	352
gdb11	27	51	303	311	317	311	317	303
gdb12	12	25	275	275	275	275	275	275
gdb13	22	45	395	395	395	395	395	395
gdb14	13	23	448	458	458	458	458	458
gdb15	10	28	536	544	544	544	544	540
gdb16	7	21	100	100	100	100	100	100
gdb17	7	21	58	58	58	58	58	58
gdb18	8	28	127	127	127	127	127	127
gdb19	8	28	91	91	91	91	91	91
gdb20	9	36	164	164	164	164	164	164
gdb21	8	11	55	55	55	55	55	55
gdb22	11	22	121	121	121	123	123	123
gdb23	11	33	156	156	156	156	160	156
gdb24	11	44	200	200	200	200	200	200
gdb25	11	55	233	233	235	235	235	233

6.5.4 Comparaison des résultats de MAF-DISTA à la littérature

Le tableau 6.8 compare les résultats de MAF-DISTA avec les meilleures solutions publiées pour le CARP. Rappelons aussi que, sur toutes les instances de la littérature que nous avons testées, MAF-DISTA a donné des résultats remarquables.

Ainsi, on peut conclure que la communication et l'échange d'individus entre les agents mis en application dans notre système *MAF-DISTA* peuvent améliorer le processus dans son ensemble pour trouver des meilleures solutions par rapport à *PGA*.

6.5.5 Performance de MAF-DISTA et effet du nombre d'agents

Nous avons effectué des expériences qui divisent une population initiale de 800 individus en sous-populations de tailles égales entre les agents. Les figures 6.7 et 6.8 montrent respectivement le temps de traitement et le fitness par rapport au nombre d'agents.

Dans la figure 6.8, on peut noter que la qualité de la solution s'améliore progressivement avec l'augmentation du nombre d'agents (sous-populations). En effet, le nombre d'agents (sous-populations) a une influence directe sur la diversité génétique de ce modèle parallèle. Un grand nombre de sous-populations implique une réduction du nombre d'individus par

TAB. 6.6 – Résultats obtenus par PGA et MAF-DISTA sur les instances de Belenguer et Benavent

Pb	N	M	LBB	Best	Carpet	GA	PGA	MAF-DISTA
1A	24	39	173	173	173	173	178	173
1B	24	39	173	173	173	173	181	173
1C	24	39	245	245	245	245	254	245
2A	24	34	227	227	227	227	235	227
2B	24	34	259	259	260	259	270	259
2C	24	34	455	457	494	462	479	457
3A	24	35	81	81	81	81	82	81
3B	24	35	87	87	87	87	89	87
3C	24	35	137	138	138	138	144	137
4A	41	69	400	400	400	400	441	400
4B	41	69	412	412	416	412	474	416
4C	41	69	428	428	453	428	473	428
4D	41	69	520	541	556	541	595	536
5A	34	65	423	423	423	423	481	423
5B	34	65	446	446	448	446	492	446
5C	34	65	469	474	476	474	524	476
5D	34	65	571	581	607	581	649	577
6A	31	50	223	223	223	223	245	223
6B	31	50	231	233	241	233	245	233
6C	31	50	311	317	329	317	332	317
7A	40	66	279	279	279	279	302	279
7B	40	66	283	283	283	283	306	283
7C	40	66	333	334	343	334	358	333
8A	30	63	386	386	386	386	440	386
8B	30	63	395	395	401	395	443	401
8C	30	63	517	528	533	533	592	528
9A	50	92	323	323	323	323	370	323
9B	50	92	326	326	329	326	336	327
9C	50	92	332	332	332	332	338	332
9D	50	92	382	391	409	391	413	391
10A	50	97	428	428	428	428	430	428
10B	50	97	436	436	436	436	440	438
10C	50	97	446	446	451	446	458	446
10D	50	97	524	535	544	535	560	536

sous-population (réduction du nombre de gènes). Comme les clones sont interdits, les sous-populations deviennent hétérogènes et permettent d'explorer des parties très importantes de l'espace des solutions.

D'autre part, le temps d'exécution diminue avec l'augmentation du nombre d'agents (cf. figure 6.7.

TAB. 6.7 – Résultats obtenus par PGA et MAF-DISTA sur les instances d'Eglese

Pb	N	M/2	T	LBB	Carpet	MA	PGA	MAF-DISTA
egl-e1-A	77	98	51	3515	3625	3548	3752	3548
egl-e1-B	77	98	51	4436	4532	4498	5054	4498
egl-e1-C	77	98	51	5453	5663	5595	6166	5595
egl-e2-A	77	98	72	4994	5233	5018	5716	5018
egl-e2-B	77	98	72	6249	6422	6340	6540	6340
egl-e2-C	77	98	72	8114	8603	8415	9338	8385
egl-e3-A	77	98	87	5869	5907	5898	6523	5898
egl-e3-B	77	98	87	7646	7921	7822	8113	7816
egl-e3-C	77	98	87	10019	10805	10433	10459	10369
egl-e4-A	77	98	98	6372	6489	6461	6461	6461
egl-e4-B	77	98	98	8809	9216	9021	9216	9021
egl-e4-C	77	98	98	11276	11824	11779	11878	11779
egl-s1-A	140	190	75	4992	5149	5018	5148	5018
egl-s1-B	140	190	75	6201	6641	6435	7086	6435
egl-s1-C	140	190	75	8310	8687	8518	9572	8518
egl-s2-A	140	190	147	9780	10373	9995	9995	9995
egl-s2-B	140	190	147	12886	13495	13174	13174	13174
egl-s2-C	140	190	147	16221	17121	16795	16825	16715
egl-s3-A	140	190	159	10025	10541	10296	11213	10296
egl-s3-B	140	190	159	13554	14291	14053	15163	14053
egl-s3-C	140	190	159	16969	17789	17297	19071	17297
egl-s4-A	140	190	190	12027	13036	12442	13280	12442
egl-s4-B	140	190	190	15933	16924	16531	18212	16531
egl-s4-C	140	190	190	20179	21486	20832	21127	20832

TAB. 6.8 – MAF-DISTA vs littérature

Critère	*gdb*		*val*		*egl*	
	Autres	MAF-DISTA	Autres	MAF-DISTA	Autres	MAF-DISTA
Écart en % par rapport à LBB	0.32	0.31	0.75	0.73	2.8	2.7
Nombre d'optima prouvés	17	18	21	22	0	0
Meilleures solutions	21	23	28	34	24	24

6.5.6 L'effet de la charge CPU sur l'accélération des calculs

Dans cette expérience, onze machines étaient connectées à la plate-forme JADE dont une seule a été réservée pour l'initialisation des calculs évolutionnaires. Cette dernière est utilisée pour héberger le conteneur principal « Main-Container » pour permettre la création des agents par défaut AMS, DF et RMA. Ainsi, les utilisateurs finaux sont autorisés à utiliser les dix autres machines. Chacune de ces machines est un conteneur « Container » qui comprenait les différents agents mobiles et un agent de statut local (LSA). En raison des différents accès

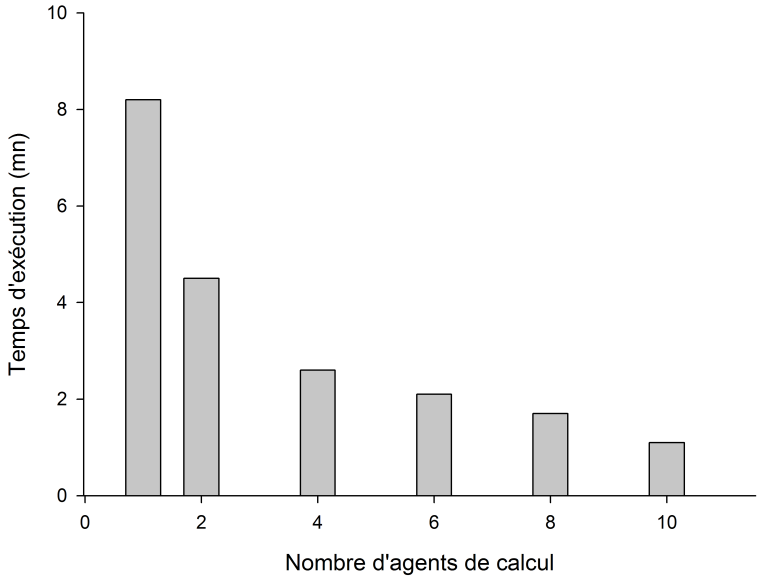


FIG. 6.7 – Effet du nombre d'agents sur le temps d'exécution

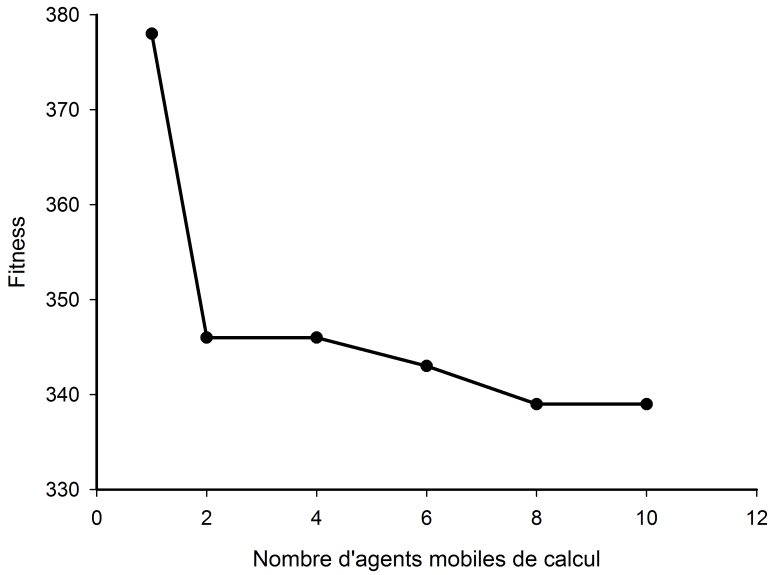


FIG. 6.8 – Effet du nombre d'agents sur le fitness

effectués par les utilisateurs, les expériences ont été conduites cinq fois dans la même journée. La figure 6.9 montre les résultats obtenus. Comme l'on peut constater, tous les calculs ont été accélérés en utilisant l'approche proposée. Elle montre l'efficacité de notre approche basée sur les agents mobiles. Bien que le coût du calcul parallèle adaptatif est plus élevé que celui du

calcul parallèle statique dans lequel les agents sont stationnaires, le premier est néanmoins une façon plus pratique pour paralléliser le calcul évolutionnaire dans un environnement réseau.

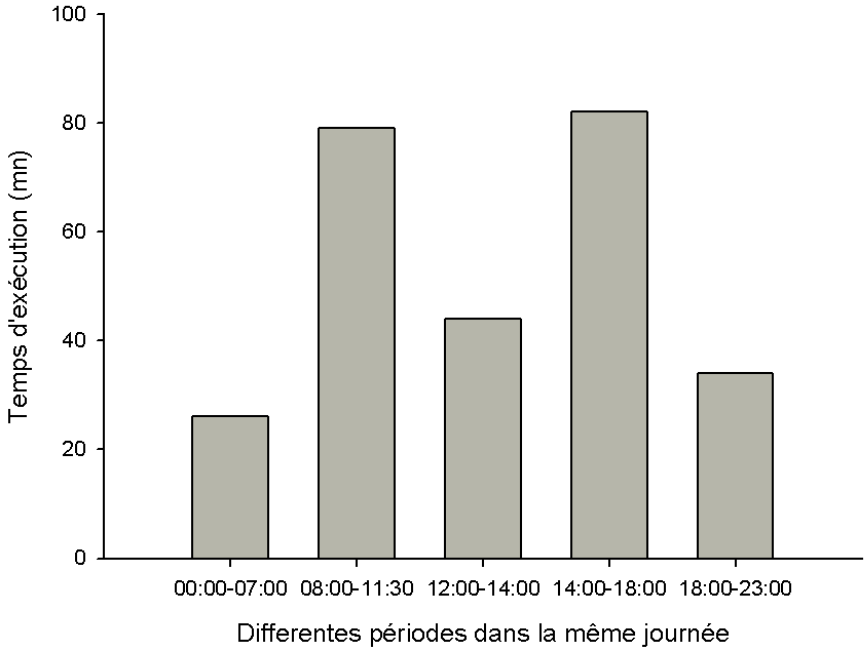


FIG. 6.9 – Effet de la charge CPU sur le temps de calcul

6.6 Les agents mobiles et leur efficacité dans MAF-DISTA

Dans le cadre des applications réparties dans les réseaux à grande envergure, l'apparition du paradigme Agent Mobile en 1994 fut une invention bouleversante qui promettait aux industriels des solutions performantes pour un meilleur accès à une information répartie et volumineuse. Au fil des années, les avantages de ce paradigme ont longtemps été discutés et il a été enfin prouvé qu'il peut être intéressant dans certains cas (Baldi et Picco (1998), Buse *et al.* (2003)). En effet, l'efficacité du paradigme Agent Mobile dépend considérablement de l'architecture du système adopté (Lu et Mori (2003), Buse *et al.* (2003)).

En outre, la plate-forme MAF-DISTA est un intergiciel permettant le développement des algorithmes parallèles à base de population. Elle présente différents avantages qui ont été hérités de la plate-forme d'agents mobiles JADE, comprenant :

- La capacité à simplifier le développement des métaheuristiques parallèles à base de population composées d'entités autonomes ayant besoin de communiquer et de collaborer pour un meilleur fonctionnement du système de calcul.
- L'interopérabilité : les agents mobiles peuvent coopérer avec d'autres agents à condition qu'ils soient conformes à la norme FIPA.
- L'uniformité et portabilité : *MAF-DISTA* fournit un ensemble homogène d'APIs qui

sont indépendants du réseau et de la version du Java.
– La réutilisation maximal du code : *MAF-DISTA* fournit à l'utilisateur une conception complète de l'architecture de sa méthode de résolution. En outre, le programmeur doit refaire le moins de code possible. Elle fournit également une séparation conceptuelle maximale entre les méthodes de résolution et le problème à résoudre. L'utilisateur peut donc développer que le code spécifique au problème à résoudre.

6.7 Conclusion

Dans ce chapitre, nous avons introduit la plate-forme *MAF-DISTA* destinée à l'analyse et à la conception de métaheuristiques parallèles à base de population. Le modèle organisationnel de *MAF-DISTA* adopte les concepts du méta-modèle JADE et présente ainsi une métaheuristique parallèle sous la forme d'une organisation composée de plusieurs agents en interaction. Ce modèle peut être considéré comme une trame ou un schéma qui doit être raffiné en fonction des caractéristiques particulières de la métaheuristique retenue et du problème à traiter.

Afin d'évaluer *MAF-DISTA*, nous avons mené différentes séries d'expériences. Dans un premier temps, nous avons utilisé une version statique des agents de *MAF-DISTA* pour exploiter la puissance de calcul offerte par le réseau. Ensuite, nous avons utilisé les agents mobiles de *MAF-DISTA* pour paralléliser de manière dynamique l'exécution d'une métaheuristique à population sur différentes machines du réseau. Enfin, une dernière série d'expériences a consisté en l'examen de la mise en œuvre des différentes stratégies d'échange d'informations pour l'amélioration de la qualité de recherche.

La première version complète de *MAF-DISTA* a fait l'objet d'un chapitre de livre (Belkhelladi *et al.* (2010a)). Elle était évaluée sur les des instances du CARP et quelques instances de UWCOP[8] (défini dans le prochain chapitre).

L'efficacité de *MAF-DISTA* et de ces différents composants sont évalués en traitant un problème d'optimisation de tournées de véhicules dans le chapitre 7 puis un problème d'optimisation des avances et des retards de production dans le chapitre 8.

[8] Underground Waste Collection Optimization problem

Partie 3 : Applications

Chapitre 7

Application de MAF-DISTA à la résolution du problème de collecte des conteneurs enterrés

Sommaire

7.1 Introduction

Dans le contexte d'une économie moderne qui doit être compétitive et mieux prendre en compte les coûts environnementaux, l'optimisation des ressources est devenue un enjeu majeur. Le partenariat public/privé en matière de recherche devient alors un enjeu économique et sociétal majeur. Les outils d'aide à la décision et d'optimisation des ressources deviennent à présent incontournables pour tous les professionnels de la logistique qui sont à la recherche de réactivité et de gains de productivité.

À l'heure où les gains de productivité sont recherchés à tous les échelons de l'entreprise, l'optimisation de tournées est un atout reconnu, tant pour les prestataires logistiques et les professionnels du transport que pour les industriels grands comptes, les PME ou encore les bureaux d'études. Or, sous l'impulsion de la législation française et européenne, de vastes programmes de collecte sélective ont été lancés : des centres de tri ont été créés ou mis aux normes et le réseau de déchetteries n'a cessé de se densifier. Mais si beaucoup d'entreprises innovantes sont présentes sur les aspects recyclage et valorisation énergétique, peu d'entreprises des TIC[1] proposent leurs services aux collectivités locales pour améliorer le rendement du processus de valorisation des déchets au niveau de la collecte. Les gains à en attendre devraient être aussi important que dans le domaine industriel et représenter un défi majeur pour les collectivités locales. En particulier les collectes des déchets sont coûteuses en terme énergétique et matériel, et leur optimisation permettrait d'en diminuer le coût économique et environnemental.

La direction Déchets-Environnement d'Angers-Loire Métropole met en œuvre un nouveau système de collecte des déchets triés. Le concept est le suivant : les ordures sont déposées par les usagers dans des conteneurs souterrains en fonction de leur nature (papiers, verres, ordures ménagères), puis collectées par des véhicules spécifiquement conçus pour leur vidage automatique. Testé avec succès dans différents quartiers de la communauté d'agglomérations, ainsi que dans d'autres villes (Antibes, Pau, etc.), ce système est amené à se généraliser.

Les outils d'aide à la décision pour l'optimisation de tournées mettent en relation des points de livraison ou de collecte avec une ou plusieurs plates-formes d'où partent et reviennent les véhicules, tout en tenant compte des coûts (fixes, kilométriques et horaires) et de contraintes internes et externes :

- jours et plages horaires possibles de livraison ou collecte,
- capacités de chargement de chaque type de véhicules ou spécifiquement de chaque véhicule,
- accessibilité sur le site du client,
- incompatibilité éventuelle entre les produits,
- disponibilité des véhicules,
- mono ou multi-dépôts, retour au même dépôt ou non,
- contraintes sociales, contraintes géographiques ponctuelles,
- etc.

L'objectif de ce chapitre est de montrer l'application de l'approche *MAF-DISTA* sur le problème de tournées de collecte des conteneurs enterrés. Ce problème consiste à déterminer un ensemble optimal de circuits afin de collecter un ensemble de conteneurs de déchets.

[1] Technologies de l'Information et de la Communication

À travers cette application, nous souhaitons tout d'abord illustrer la spécialisation dans *MAF-DISTA* et montrer l'intérêt du choix d'une architecture multi-agents. Ensuite, l'un des objectifs de cette application au *UWCOP*[2] est d'évaluer de manière expérimentale l'apport des mécanismes d'auto-adaptation et de coopération.

Ce chapitre débute par une présentation du *UWCOP* et des approches de résolution existantes. Ensuite, la section 7.3 détaille la spécialisation de *MAF-DISTA* pour traiter le *UWCOP*. Les résultats expérimentaux de *MAF-DISTA* sont analysés dans la section 7.4. Un bilan de ces expérimentations est dressé dans la dernière section du chapitre.

7.2 UWCOP - Problème de tournées de collecte des conteneurs enterrés

7.2.1 Présentation du problème

Le système de stockage des déchets de la ville d'Angers, ou le programme TOM[3], est un système de gestion des déchets novateur. Il est composé de deux éléments principaux : d'une part d'un conteneur de déchets enfoui, dont la capacité est de 4 ou 5 mètres cubes, et d'autre part d'un dépôt spécialement conçu. Le système de ramassage des véhicules a été spécifiquement développé ; ces véhicules disposent d'un système de chargement latéral qui permet à un seul conducteur (ou opérateur) de vider les conteneurs de manière automatique. Le temps pour vider un conteneur est d'environ 7 minutes. Cette durée est calculée entre le moment où le véhicule se stationne et le moment où il repart. Le véhicule a un volume de 20 mètres cubes, offrant une capacité de charge équivalente à environ 9,5 tonnes.

Comme dans d'autres villes de France, la majorité de la population d'Angers est concentrée dans des zones urbaines à très forte densité. De ce fait, la technique de collecte a évolué. Il y a dans chaque point de collecte un ou plusieurs conteneurs. Différents camions recueillent chaque type de déchets et les transportent à leur destination (une décharge, un incinérateur ou un centre de recyclage). Le facteur déterminant pour assurer le bon fonctionnement du système de collecte est la conception d'itinéraires appropriés pour chaque type de déchets, qui doivent être collectés séparément. À Angers chaque type de déchets est collecté séparément avec une périodicité fixée par la municipalité. Dans ce cas, la collecte de chaque type de déchets (ordures urbaines, déchets génériques comme le papier, les plastiques, les emballages, le verre, etc.) peut être considérée comme un problème indépendant mais équivalent. En outre, les itinéraires doivent se conformer au code de la route, tout en réduisant au minimum le coût de la collecte.

Par rapport à la collecte à domicile (porte-à-porte), de nouvelles contraintes sont à considérer. Tout d'abord, un conteneur doit être vidé si son taux de remplissage dépasse un certain seuil (20% dans notre cas). Ensuite, certains conteneurs sont considérés comme des points noirs et doivent être vidés, même si leur taux de remplissage est inférieur à ce seuil. Ces conteneurs sont situés dans des lieux sensibles, généralement dans des structures publiques comme les marchés. Enfin, deux coûts par route sont considérés : le coût de parcours et le coût de ramassage des conteneurs.

[2] Underground Waste Collection Optimization Problem
[3] Traitement des Ordures Ménagères

Notons que le problème considéré est un problème de tournées sur arcs puisque dans les circonstances réelles, les routes sont complètement traversées même si les points de collecte sont situés dans des endroits précis de la route. En outre, la capacité des camions est limitée. Ainsi, la nature fondamentale du problème est celle d'un problème de tournées sur arcs avec capacités (CARP[4]). Même si ils n'ont pas été étudiés de façon extensive comme leurs homologues de tournées sur nœuds, il existe une littérature assez vaste sur les problèmes de tournées sur arcs (Dror (2000)) et sur le CARP en particulier (ex. Golden et Wong (1981), Belenguer et Benavent (1998)). Néanmoins, le problème que nous traitons présente plusieurs caractéristiques qui le rendent différent des problèmes de tournées sur arcs classiques. Ces caractéristiques sont essentiellement issues des contraintes de circulation. D'une part, le graphe du problème est un graphe mixte contenant des arcs et arêtes, car en réalité, certaines rues peuvent être parcourues dans les deux sens alors que d'autres ont une seule direction. D'autre part, à la jonction des rues, quelques interdictions de tourner, comme des demi-tours interdits à certains carrefours, alors que d'autres sont autorisés. Ces questions ont été abordées théoriquement par Lacomme *et al.* (2001) et par Belenguer *et al.* (2006). Pour des problèmes sans capacités, les demi-tours interdits ont été considérés pour le calcul et l'ordonnancement des tournées pour le balayage des rues dans (Bodin et Kursh (1978), Bodin et Kursh (1979)) et pour le problème du postier rural sur des graphes mixtes avec application des pénalités sur les demi-tours interdits dans (Corberán *et al.* (2002)), où les auteurs proposent une approche de résolution basée sur la pénalisation des demi-tours interdits et le post-traitement des solutions irréalisables. Dans (Roy et Rousseau (1989)), la formulation du problème de postier chinois avec capacité a été adaptée au contexte de la société canadienne des postes et une heuristique générale a été présentée pour résoudre le problème sous plusieurs hypothèses où, là encore, des sanctions ont été affectées aux demi-tours interdits.

Dans ce travail, nous construisons un modèle pour le problème de tournées sur arcs mixte où les demi-tours interdits sont pris en compte dans l'ensemble des contraintes au lieu de les pénaliser par l'intermédiaire de la fonction objectif. Nous pensons que cela est plus approprié lorsque le nombre de demi-tours interdits est élevé, comme dans le vrai problème que nous traitons. Dans la section 7.2.2, nous décrivons la formulation du problème et à la section 7.2.3, nous présentons le modèle proposé, qui résulte de différentes transformations du problème.

7.2.2 Formulation du problème

Le problème de collecte des conteneurs enterrés que nous avons nommé *UWCOP* (Underground Waste Collection Optimization Problem), est défini sur un réseau de rues découpé en secteurs de collecte. Par « rue », on veut dire en fait un tronçon d'une rue réelle, reliant deux carrefours. Une rue peut être à sens unique ou à double sens, étroite ou large. Il peut y avoir des interdictions de tourner, comme des demi-tours interdits à certains carrefours ou coûteux en terme de temps. Chaque rue requise a un ou plusieurs conteneurs avec une quantité de déchets à collecter, qui peuvent être répartis d'un seul côté ou des deux côtés. Selon le cas, cette répartition peut nécessiter un passage ou deux.

Une tournée de collecte est assurée par un seul camion. Le camion part du dépôt, effectue un ou plusieurs tours se terminant à un lieu de vidage, puis revient au dépôt. En pratique, il peut y avoir plusieurs lieux de vidage, équivalents ou réservés à certains types de déchets,

[4]Capacitated Arc Routing Problem

avec éventuellement des capacités de stockage limitées pour chaque type. Les contraintes de capacité des véhicules s'exercent sur chaque tour, tandis que les contraintes d'autonomie concernent la tournée entière.

Les camions doivent collecter toutes les rues ayant un nombre non nul de conteneurs. Toute rue à collecter doit être traitée par un seul véhicule et lors d'un seul tour. Par contre, toute rue peut être traversée plusieurs fois en *haut-le-pied* (appelé par la profession pour désigner le coût de traversée sans collecte), par un même camion ou par des camions différents. L'objectif est de calculer un ensemble de tournées collectant toutes les rues requises(conteneurs), en minimisant le coût total. Le nombre de camions utilisés peut être imposé (plein emploi des chauffeurs), être plafonné à une valeur maximale ou être une variable de décision, éventuellement à minimiser.

UWCOP est défini par un graphe $G = (V, E, A)$, les coûts de traversé dc : $E \rightarrow \mathbb{Z}^+$, les coûts de collecte $cc : E \rightarrow \mathbb{Z}^+$, les demandes $w : E \rightarrow \mathbb{Z}^+$, une flotte de K véhicules avec une capacité W est basée dans un dépôt s. $R = \{(i, j) \in E | w((i, j)) > 0\}$ est l'ensemble de rues requises (à collecter). Nous supposons que F est l'ensemble de tournées qui débutent et se terminent au dépôt, dans lesquelles les rues peuvent être *servies* ou *traversées*. En outre, il existe un ensemble de règles issues de la réglementation de la circulation qui déterminent quand il est possible de connecter deux rues voisines par un sommet commun ainsi que des interdictions de tourner associées aux rues adjacentes lorsque l'angle commun est trop étroit. Nous nous référerons à ces règlements comme des contraintes de tourner. Chaque rue $a \in (E \cup A)$ a une longueur l_a. Chaque rue requise $a \in R$ a une demande $w(a)$.

L'ensemble F est une solution faisable pour *UWCOP* si et seulement si :

- Chaque rue requise est servie par une seule tournée dans F ;
- La somme des demandes des rues requises dans une tournée de F ne doit pas dépasser la capacité W du camion ;
- Les règles de circulation sont respectées.

Nous souhaitons donc trouver une solution qui minimise la somme des coûts des tournées de collecte en respectant les règles de circulation.

UWCOP est définit par le programme linéaire suivant :

Notations

E : Ensemble des arêtes.
R : Ensemble des rues requises.
V : Ensemble des nœuds du réseau.
I : Ensemble des véhicules.
W : Capacité d'un véhicule.
K : Nombre de véhicules.
C : Capacité d'un conteneur.
Q_c : Quantité des déchets disponibles dans un conteneur c.
M_{ij} : Nombre de conteneurs enterrés placés sur la rue *(i,j)*.
T_{cc} : Cycle total de collecte d'un conteneur.
v_c : Une variable binaire qui vaut 1 si $Q_c \geq 20\% \times C$ et 0 sinon.
b_c : Une variable binaire qui vaut 1 si le conteneur c est *point noir* et 0 sinon.
dc_{ij} : Coût de traversée d'une arête *(i,j)*.
cc_{ij} : Coût de collecte d'une rue *(i,j)*.

w_{ij} : Demande de la rue *(i,j)* ou les demandes totales entre les nœuds i et j.
x_{ijk} : Une variable binaire qui vaut 1 si le véhicule k traverse l'arête *(i,j)* de i vers j et 0 sinon.
l_{ijk} : Une variable binaire qui vaut 1 si l'arête *(i,j)* est servie par le véhicule k de i vers j et 0 sinon.

$$Minimiser \sum_{k\in I}\sum_{(i,j)\in E} dc_{ij}x_{ijk} + \sum_{(i,j)\in R} cc_{ij} \tag{7.1}$$

$$\forall i \in V, \forall k \in I : \sum_{j\in\Gamma(i)} (x_{ijk} - x_{jik}) = 0 \tag{7.2}$$

$$\forall (i,j) \in E, \forall k \in I : x_{ijk} \geq l_{ijk} \tag{7.3}$$

$$\forall (i,j) \in R, \sum_{k\in I}(l_{ijk} + l_{jik}) = 1 \tag{7.4}$$

$$\forall k \in I, \sum_{(i,j)\in R} l_{ijk}w_{ij} \leq W \tag{7.5}$$

$$\forall (i,j) \in R, cc_{ij} = \sum_{m=1}^{M_{ij}} T_{cc} \times (v_c \vee b_c) \tag{7.6}$$

$$u_s^k, y_s^k, x_{ijk}, l_{ijk}, v_c, b_c \in \{0,1\} \tag{7.7}$$

$$\forall S \neq \emptyset, S \subset \{2,..,n\} \; et \; \forall k \in I : \begin{cases} \sum_{i,j\in S} x_{ijk} - n^2 y_s^k \leq \mid S \mid -1 \\ \sum_{i\in S}\sum_{j\in S} x_{ijk} + u_s^k \geq 1 \\ u_s^k + y_s^k \leq 1 \end{cases} \tag{7.8}$$

L'objectif (7.1) consiste à minimiser la somme des coûts de parcours et de collecte. Les contraintes (7.2) garantissent qu'un véhicule qui arrive en un nœud doit en repartir. Les contraintes (7.3) garantissent qu'une rue traitée est aussi traversée. Les contraintes (7.4) imposent que chaque rue requise soit traitée, par un seul véhicule et dans un seul sens. Les contraintes (7.5) assurent le respect de la capacité des véhicules. Un conteneur doit être vidé si son taux de remplissage dépasse un certain seuil (20% dans notre cas) ou il est point noir, ceci est assuré par la contrainte (7.6). Enfin l'interdiction des demi-tours illégaux est imposée par les contraintes (7.8) qui nécessitent des variables binaires supplémentaires y_s^k et u_s^k.

Remarque : On dit qu'un chemin est faisable pour les règles dans le graphe G, s'il ne viole aucune des contraintes et respecte toutes les règles de la circulation. Notons que pour toute paire de sommets $(i,j) \in V^2$, il est possible de savoir s'il existe un chemin allant de i vers j en respectant les règles et pour déterminer la valeur du plus court chemin en fonction des longueurs d'arcs l. Ceci est possible en utilisant une procédure basée sur l'algorithme de *Dijkstra* qui a été proposée par Boroujerdi et Uhlmann (1998). Nous notons donc par $P_{i,j}$, le plus court chemin entre i et j dans le graphe G, et pour valeur $l(P_{i,j})$ (Bautista *et al.* (2008)).

7.2.3 Stratégie de résolution

Les problèmes de tournées sur nœuds comme le VRP (Vehicle Routing Problem) sont plus étudiés que les problèmes de tournées sur arcs, probablement à cause de leurs applications précoces en logistique de distribution. De nombreux modèles et méthodes de résolution ont été proposés, citons les articles de Bodin *et al.* (1983), Toth et Vigo (2001), Laporte (1992) et de Laporte (1997). Il est donc tentant de transformer les problèmes de tournées sur arcs en problèmes de tournées sur nœuds, afin de réutiliser les algorithmes déjà disponibles.

Pearn *et al.* (1987) sont les premiers à avoir proposé une telle transformation, pour des graphes non orientés. Elle consiste à remplacer chaque arête par trois nœuds et à répartir la quantité de l'arête sur ces trois nœuds. Laporte (1997) a détaillé une autre transformation pour certains problèmes de tournées sur arcs dans un graphe non orienté. Le point commun entre les problèmes traités par Laporte est l'absence des contraintes de capacité. La méthode consiste à créer deux nœuds à partir de chaque arête, en ne multipliant ainsi que par deux la taille du problème, au lieu de trois. Il reste ensuite à résoudre un problème de tournées généralisé (appelé GRP ou General Routing Problem), qui consiste à déterminer un tour de coût minimal passant au moins une fois sur chaque arête et sur chaque nœud (Bautista *et al.* (2008)).

Dror et Langevin (1997) ont converti le problème du postier rural orienté en un problème du voyageur de commerce généralisé, dans lequel les nœuds du graphe sont répartis en groupes et où il suffit de visiter un nœud par groupe. Cette approche est intéressante mais ne permet pas de gérer des contraintes de capacité. Mullaseril et Dror (1996) ont travaillé sur un problème de tournées sur arcs avec contraintes de capacité et fenêtres horaires, dans un graphe orienté. Ils ont proposé une méthode transformant tout arc du graphe en un nœud à visiter dans un problème de tournées sur nœuds, en conservant ainsi la taille du problème. Gueguen (1999) a étendu cette technique aux graphes non orientés et aux graphes mixtes, avec ou sans contraintes de capacité (Bautista *et al.* (2008)).

Dans la suite de cette étude, nous allons transformer notre problème de tournées sur arcs en un problème de tournées sur nœuds. Un cadre général pour ce type de transformation pour les problèmes de tournées sur arcs sans contraintes de capacité et sans pénalités sur les interdictions de tourner a été défini par Laporte (1997), et une autre transformation qui prend en compte les demi-tours a été proposée par Corberán *et al.* (2002). Notre transformation est donc basée sur l'une des transformations proposées par Laporte (1997), même si elle intègre quelques nouvelles fonctionnalités pour gérer les pénalités sur les demi-tours et les contraintes de capacité. En outre, il convient de mentionner que la façon dont nous traitons les sanctions se base sur les distances de parcours résultant de la remarque donnée dans la section 7.2.2, qui est plus simple que la transformation utilisée par Corberán *et al.* (2002).

Nous définissons le nouveau graphe issu de la transformation par $G_{UWCOP} = (V, A)$. Ce graphe est orienté dans lequel les nœuds correspondent aux rues requises (nécessitant une collecte) dans le graphe G : un nœud si la rue requise est un arc (rue à sens unique ou bien la collecte se fait sur un seul côté de la rue), et deux nœuds si la rue requise est une arête (c-à-d, un nœud est associé à chacune des deux directions possibles de l'arête). En particulier, pour chaque arc $(i, j) \in R$ dans le graphe G, on définit un nœud $p(i, j) \in V$ dans G_{UWCOP} et pour chaque arête requise $(i, j) \in E$ dans G, on définit deux nœuds $p(i, j), p(j, i) \in V$ dans G_{UWCOP}. En outre, le nœud $0 \in V$, correspond au dépôt. Pour des fins de notation, nous

allons partitionner l'ensemble des sommets en ensembles disjoints (nous allons les représenter comme des groupes de nœuds), tel que $V = T_0 \cup T_1 \cup T_2 \cup ... \cup T_r$ avec $T_0 = 0$. Pour $t = 1, ..., r$, chaque ensemble T_t de nœuds est un singleton, quand le sommet est associé à un arc (rue requise) de R dans G, ou contient les deux sommets $p(i,j), p(j,i)$ associés aux deux arcs opposés de l'arête requise (i,j). Pour chaque nœud $p \in V$, C_p désigne l'indice du groupe auquel il appartient. Notons que $C_p = r \Leftarrow : p \in T_r$. L'ensemble d'arcs (A) contient un arc (p,q) pour chaque paire de nœuds $p = p(i,j), q = q(k,h) \in V$, avec $C_p \neq C_q$, tel que, dans G il existe un chemin respectant les règles reliant i et h. Notons que le graphe G_{UWCOP} ne doit pas être un graphe complet. Le coût d'un arc $(p,q) \in A$, avec $p = p(i,j), q = q(k,h)$, est donné par la longueur du plus court chemin reliant i et k dans G et respectant les règles. Lorsque $j = k$ et aucun demi-tour n'est autorisé depuis (i,j) vers (k,h), les longueurs de tous les arcs supplémentaires qui doivent être parcourus pour relier (i,j) et (k,h) d'une manière réaliste doivent être pris en compte. C'est-à-dire $c_{pq} = l(P_{ih}) - l_{kh}$. Notons que la matrice de coût qui en résulte est asymétrique. Notons également que le coût c_{pq} représente les longueurs de tous les arcs dans le chemin faisable P_{ih}. En principe, les longueurs des rues requises ne doivent pas être calculées, car toute solution réalisable desservira toutes les rues requises exactement une fois et, par conséquent, la longueur totale des arêtes servies est une constante qui ne modifie pas la solution du problème d'optimisation. Toutefois, lorsque la réglementation des demi-tours doit être prise en compte, la longueur du chemin entre deux arcs voisins ne peut pas être nulle si une interdiction de tourner sur un nœud commun est imposée. Au contraire, elle doit inclure les longueurs de tous les arcs utilisés pour définir le chemin d'accès possible (cf. figure 7.1).

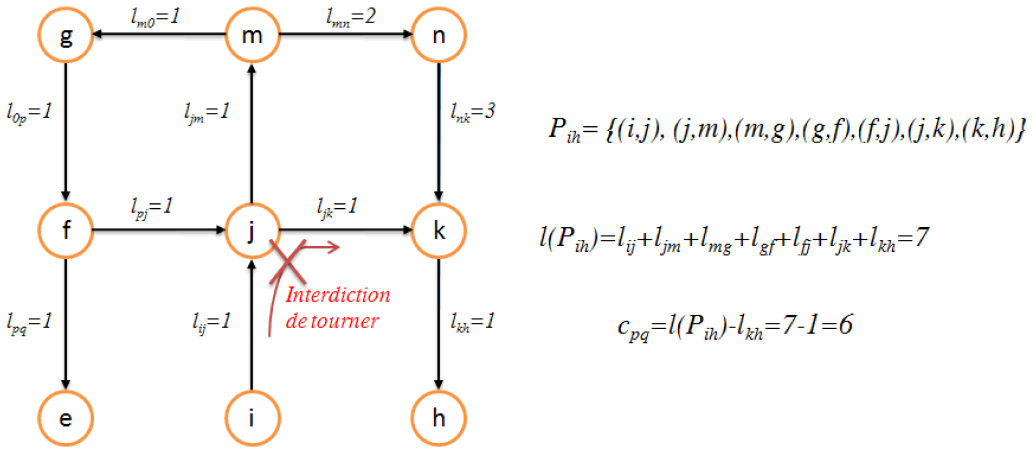


FIG. 7.1 – Calcul de la distance d'un chemin avec des interdictions de tourner (Bautista *et al.* (2008))

Nous définissons une fonction de demande dans G_{UWCOP}, qui est héritée de la fonction de demande originale du graphe G. Nous simplifions ainsi la notation, si $p = p(i,j) \in V$, on note d_p la demande de la rue requise (i,j).

UWCOP est donc équivalent à un problème de tournées sur nœuds asymétrique sur le graphe G_{UWCOP} qui consiste à trouver un ensemble de tournées tel que :

– chaque groupe est exactement visité par une tournée,
– pour chaque groupe, l'un de ses sommets est exactement visité par une tournée,
– la demande totale des sommets visités par chaque tournée ne dépasse pas la capacité du véhicule.

La transformation du $UWCOP$ a donné un problème qui est une extension du problème de tournées de véhicules avec des groupes de nœuds connu sous le nom de problème de tournées de véhicules généralisé (GVRP[5]). Pour résoudre ce problème, nous avons utilisé la transformation proposée par Dimitrijević et Šarić (1997). Avec cette nouvelle transformation du $GVRP$ nous avons obtenu un problème de tournées de véhicules de type VRP. Dans la suite de ce chapitre, nous présentons une adaptation de notre plate-forme de calcul ($MAF-DISTA$) pour la résolution du VRP résultant de la double transformation de $UWCOP$.

7.3 MAF-DISTA pour UWCOP

Après avoir effectué une double transformation sur $UWCOP$, nous avons obtenu un problème de tournées sur nœuds (VRP). Pour résoudre ce problème, nous avons utilisé un algorithme génétique comme approche de résolution. Dans cette section, nous détaillons les différentes parties de l'algorithme génétique développé pour le VRP. Puis nous adaptons notre plate-forme *MAF-DISTA* pour l'exécution distribuée de l'algorithme génétique développé ici.

7.3.1 Algorithme génétique pour le VRP

Nous avons repris une partie des travaux effectués par Baker et Ayechew (2003) pour mettre en œuvre un algorithme génétique pour le VRP.

7.3.1.1 Génération de la population initiale

Nous avons utilisé une méthode de génération des solutions structurées. Cette méthode est inspirée de l'approche d'affectation généralisée de Fisher et Jaikumar (1981). Ces auteurs ont présenté une méthode pour générer automatiquement un emplacement de base pour chaque véhicule. En utilisant une approximation des distances parcourues correspondant à toutes les localisations des rues requises, un problème d'affectation généralisé est résolu pour parvenir à une répartition des rues requises aux véhicules qui satisfont les contraintes de capacité. Cependant, pour l'algorithme génétique, nous générons une population initiale de solutions dans laquelle la structure des tournées peut être assez brute et un certain degré de violation de contraintes peut survenir. Par conséquent, nous avons légèrement simplifié la méthode de génération des emplacements. Au lieu de résoudre un problème d'affectation généralisé, nous utilisons une attribution aléatoire pour chaque rue requise à l'un de ses deux meilleurs emplacements.

L'ensemble des emplacements est généré comme suit : les cônes des rues requises sont définies par le dessin des rayons à partir du dépôt qui coupent les angles entre les rues adjacentes selon l'ordre dans lequel elles sont triées, tels que décrits par Fisher et Jaikumar (1981). Ensuite, les cônes de véhicules se forment par l'agrégation des cônes des rues requises et des fractions de cônes des rues requises pour obtenir une répartition de la demande égale

[5]Generalized Vehicle Routing Problem

entre les cônes des véhicules. Chaque emplacement est situé le long d'un rayon qui coupe en deux l'angle du cône du véhicule correspondant. La distance de chaque emplacement à partir du dépôt est égale au maximum des distances des rues requises pour ce cône de véhicule. En supposant une matrice de distance symétrique, d_{ij}, avec 0 représentant le dépôt, le coût de l'affectation d'une rue requise j à un emplacement i est calculé par $c_{ij} = d_{0j} + d_{ij} - d_{0i}$.

Ensuite, si a_j et b_j sont respectivement le premier et le deuxième plus petits coûts d'affectation d'un emplacement pour la rue requise j, chaque rue requise est affectée à l'un de ses deux meilleurs emplacements, avec les probabilités données respectivement par $b_j/(a_j + b_j)$ et $a_j/(a_j + b_j)$.

Les tournées de chaque véhicule sont alors calculées en utilisant la méthode 2-optimale (Croes (1958)), suivie d'un raffinement 3-optimale (Lin (1965)). Aucune tentative n'est faite pour satisfaire les contraintes de capacité ou de distance au cours du processus d'affectation. D'autres membres de la population peuvent être générés en répétant le processus d'affectation aléatoire.

7.3.1.2 La sélection

Nous avons choisi de mettre en œuvre la méthode de sélection proportionnelle *RWS*. L'algorithme 6 de cette méthode est détaillé ci-dessous :

Algorithme 6 La méthode de sélection proportionnelle *RWS*

```
pick = Random[0,1]  // Choix d'un nombre aléatoire entre 0 et 1
Stotal = Somme total des fonctions objectifs de tous les individus de la population
S = 0
i = 0  // individu choisi
répéter
    S = S + FonctionObjectif[i]
    i=i+1
jusqu'à pick > S/Stotal
```

7.3.1.3 Croisement

Les enfants sont produits à partir des deux solutions parents à l'aide d'une procédure de croisement standard. Les meilleurs résultats ont été obtenus en utilisant le croisement à 2 points de coupure, dans lequel deux points dans le chromosome sont choisis de manière aléatoire. Un premier enfant est composé des valeurs de gènes d'un parent qui sont à gauche du premier point et à droite du deuxième point, avec les valeurs de gènes du deuxième parent qui sont entre les deux points choisis dans le chromosome. Un deuxième enfant est produit en échangeant autour des parents et puis en utilisant la même procédure.

Comme nous l'avons déjà mentionné, l'objectif est de numéroter les véhicules de telle sorte que, pour tous les individus de la population, des véhicules portant le même numéro fonctionnent à peu près dans la même région. Puis, dans un cas typique du *VRP*, si le dépôt est situé au centre d'une clientèle uniformément répartie, le croisement à 2 points de coupure est équivalent à diviser le plan en deux secteurs avec deux rayons tirés au hasard à partir du dépôt. Ceci est possible en utilisant les affectations des véhicules à partir de l'un des parents

au sein d'un secteur et les affectations de véhicules de l'autre parent dans le deuxième secteur. Pour atteindre la numérotation du véhicule, la moyenne des coordonnées x et la moyenne des coordonnées y sont calculées pour les rues requises visitées par chaque véhicule, donnant des points $(\overline{x}_i, \overline{y}_i)$, $i = 1, .., m$. Puis les angles polaires des points $(\overline{x}_i, \overline{y}_i)$ sont utilisés pour ordonner les véhicules, avec le véhicule 1 correspondant à l'angle le plus proche de zéro (ce qui peut être un angle un peu moins de 360°) et, par la suite, les $m-1$ véhicules restants étant numérotés dans l'ordre croissant de l'angle. Avant le croisement des deux parents choisis, si l'angle pour le véhicule 1 dans l'un des deux parents est plus proche de l'angle du véhicule 2 dans l'autre parent qu'il ne l'est à l'angle du véhicule 1, alors les véhicules d'un parent seront numérotés de nouveau pour rendre des angles pour les premiers véhicules plus proches (Baker et Ayechew (2003)).

La figure 7.3 illustre l'application du croisement à 2 points de coupure à un problème avec 20 rues requises, dans lequel une solution avec quatre véhicules est demandée. Les rues requises et les véhicules ont été numérotés dans l'ordre croissant de la taille de l'angle polaire, tel que décrit précédemment. Les figures 7.3(a) et 7.3(b) montrent deux parents bien structurés, dont les représentations sont les suivantes (cf. figure 7.2) :

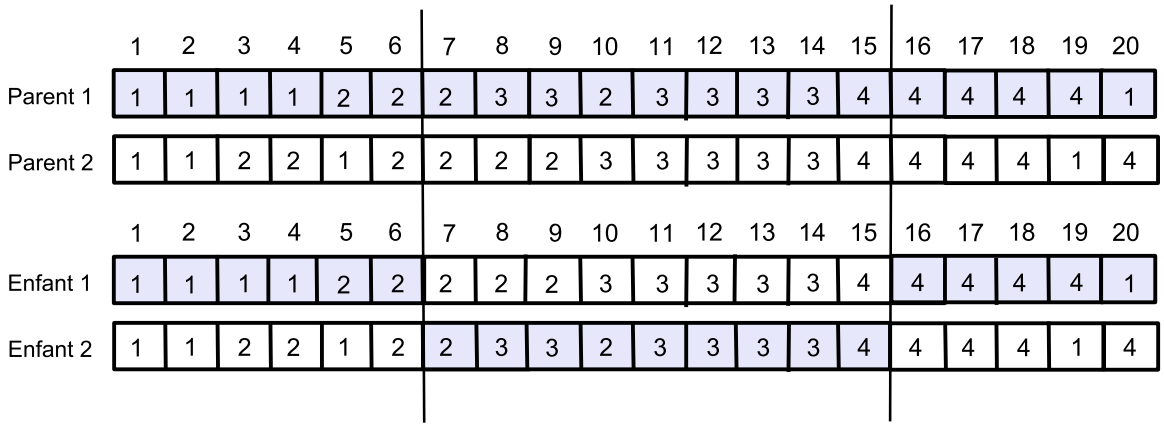


Fig. 7.2 – Croisement à 2 points de coupure

Les points de croisement ont été générés entre les rues requises 6 et 7, et entre les rues requises 15 et 16. Les rayons sont indiqués par des lignes en pointillés et ont été insérés dans les emplacements correspondants dans la figure 7.3, divisant la région en deux secteurs. L'application du croisement à 2 points de coupure a donné les deux chromosomes (*Enfant 1* et *Enfant 2*, cf. figure 7.2), chacun ayant des affectations de véhicules du parent $P1$ au sein d'un secteur, et du parent $P2$ dans l'autre secteur.

La figure 7.3(c) montre l'enfant ($Enfant1$), qui a la meilleure structure de tournées parmis les quatre solutions. La possibilité qu'une tournée d'un véhicule est positionnée avec une autre tournée n'est pas prévue. Si la distance à partir du dépôt de $(\overline{x}_{i+1}, \overline{y}_{i+1})$ est inférieur à la moitié de la distance à partir du dépôt de $(\overline{x}_i, \overline{y}_i)$ pour tout i, et la différence entre les angles des véhicules est inférieure à $180^\circ/m$, alors les deux numéros de véhicules sont échangés, de sorte que la route intérieure vient toujours avant la route extérieure (Baker et Ayechew (2003)).

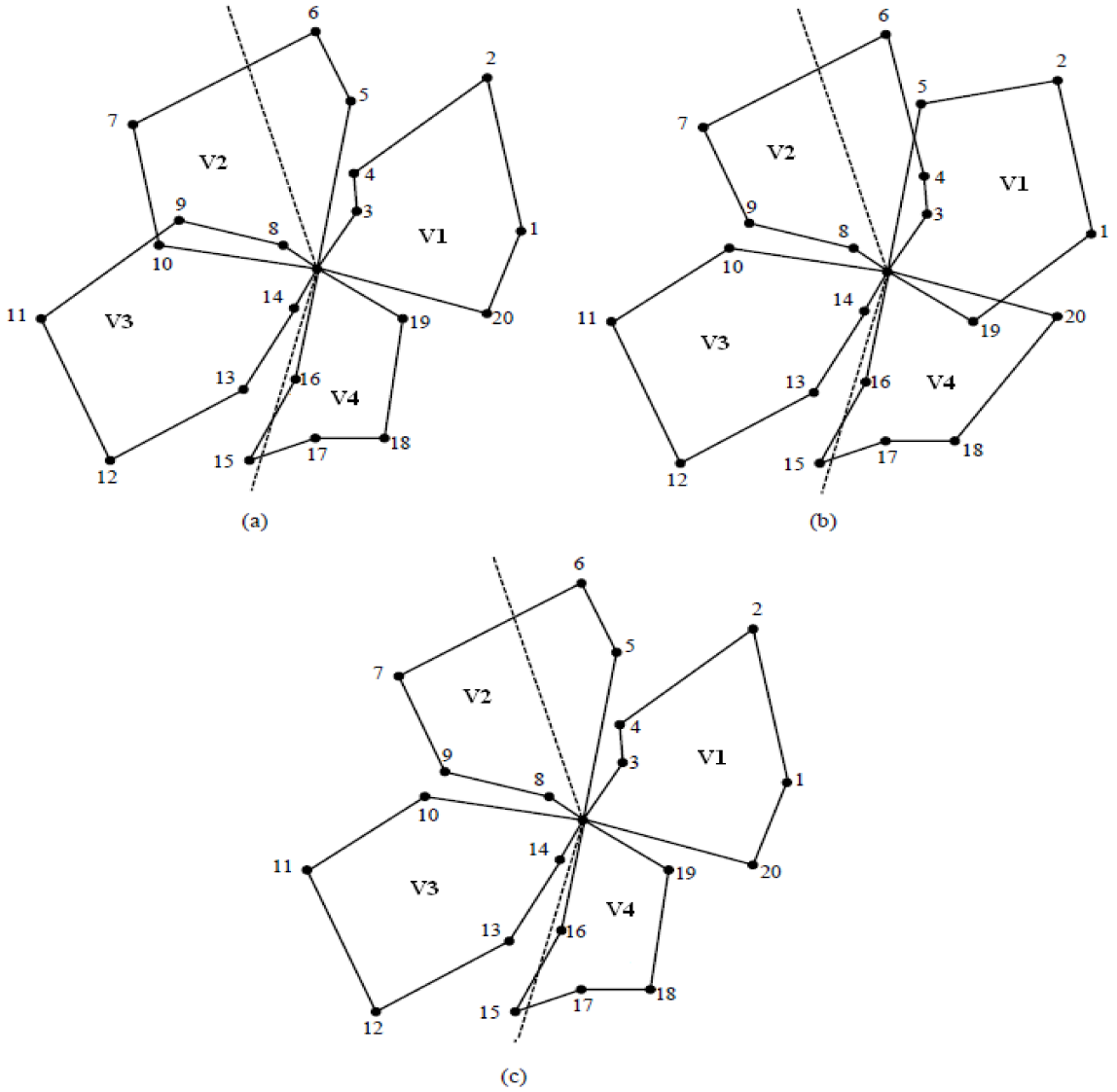


FIG. 7.3 – (a) Parent 1, (b) Parent 2, (c) Enfant 1. (Baker et Ayechew (2003))

7.3.1.4 Mutation

La performance de notre algorithme génétique peut être améliorée en appliquant une mutation simple aux chromosomes issus du croisement, dans laquelle deux gènes sont sélectionnés au hasard et leurs valeurs sont échangées. Ainsi, deux rues requises choisies au hasard sont échangées entre les véhicules, sauf dans les cas où les deux rues requises se trouvent sur le même véhicule.

Dans la suite de ce chapitre, nous nous intéressons à l'adaptation de *MAF-DISTA* pour l'optimisation de *UWCOP*. Nous reprenons donc les différentes définitions déjà fournies dans le chapitre précédent. Chaque agent mobile de calcul dans *MAF-DISTA* encapsule une instance de l'algorithme génétique dont les différents composants sont donnés dans les sections

précédentes. Il s'agit de l'ensemble des opérateurs de croisement de mutation, de la sélection par la roue de la fortune biaisée.

En ce qui concerne l'échange d'informations entres les agents, nous avons utilisé les deux stratégies de communication définies dans le chapitre 4. Il s'agit de la stratégie d'immigration et intégration sélectives et de la stratégie consistant en l'ajustement des paramètres d'un AE par le biais d'échange.

La section suivante décrit l'ensemble des résultats obtenus sur des instances de *UWCOP* que nous avons générées à partir des informations fournies par les services de la direction Déchet-Environnement d'Angers-Loire Métropole.

7.4 Simulations et résultats

Nous avons généré une instance de *UWCOP* à partir des informations fournies par les services de la direction Déchets-Environnement d'Angers-Loire Métropole. Il s'agit du problème défini dans la section 7.2. C'est un problème avec un seul dépôt, 2 véhicules d'une capacité de 9,5 tonnes, des conteneurs pour chaque type de déchet. Le temps de collecte d'un conteneur est estimé à 7 minutes.

Les résultats obtenus par *MAF-DISTA* pour cette instance de *UWCOP* sont détaillés dans les tableaux 7.1, 7.2, 7.3, 7.4, 7.5, 7.6 et 7.7.

Dans tous les tableaux, *N* donne le numéro du point de collecte. *Commune* indique le nom de la commune dans laquelle se situe le point de collecte. *Point de collecte* donne l'adresse exacte du point de collecte. *N° conteneur* donne la référence du conteneur à collecter. La quantité des déchets de chaque conteneur est donnée dans la colonne *Tonnage*. La dernière ligne du tableau indique le nombre total de conteneurs collectés par tournée, la durée totale de la tournée et la quantité totale des déchets collectée lors de cette tournée.

7.4.1 Planification des tournées

Contrairement aux tournées sur nœuds en distribution, les jours de collecte d'une rue donnée doivent être les mêmes chaque semaine. Il y a à cela deux raisons principales. Premièrement, la production de déchets est relativement stable, ce qui autorise le calcul des tournées sur une semaine-type. Deuxièmement, les équipages mettent du temps pour bien connaître les rues d'un quartier et préfèrent donc des tournées stables d'une semaine à l'autre.

La figure 7.4 illustre une planification des tournées obtenues par *MAF-DISTA* pour l'instance du *UWCOP*. Cette planification a permis de répartir les différentes tournées sur les jours de la semaine en respectant le nombre de camions et le temps de travail des agents.

7.4.2 Tests sur le terrain

Les résultats obtenus par *MAF-DISTA* ont fait l'objet d'un test sur le terrain en collaboration avec les agents (ingénieurs et chauffeurs) de la direction Déchets-Environnement d'Angers-Loire Métropole. Ce test a tenu sur une semaine comme horizon de planification. Le plan de la figure 7.5 résume les résultats finaux que nous avions obtenus lors de cette première compagne de test. Cette première expérience de nos outils d'aide à la décision a permis de réduire le nombre de tournées. Ce nombre est passé de 8 tournées à 7 tournées par

TAB. 7.1 – Tournée n° 1

N	Commune	Point de collecte	N° conteneur	Tonnage
1	ANGERS	Place Lafayette	181284	0,22
2	ANGERS	Place Lafayette	181285	0,23
3	ANGERS	Place Imbach / Pocquet de Livonnières	181270	0,22
4	ANGERS	Place Olivier Giran	181 274	0,28
5	ANGERS	Place Olivier Giran Bas	181 276	0
6	ANGERS	Place Olivier Giran Haut	181 275	0,22
7	ANGERS	bvd Beaussier (n°8)	181213	0,56
8	ANGERS	bvd Beaussier (n°10)	181225	0,2
9	ANGERS	Rue Quémard (n°61)	181227	0,31
10	ANGERS	Rue Quémard (n°64)	181247	0,34
11	ANGERS	Rue Quémard (n°66)	181234	0,3
12	ANGERS	Rue Quémard (n°68)	181248	0,3
13	ANGERS	Rue Quémard 2ème (n°68)	181249	0,2
14	ANGERS	rue Pierre Blandin (n°37 - 39)	181250	0,24
15	ANGERS	Rue Pierre Blandin (n°56)	181254	0,34
16	ANGERS	rue Pierre Blandin (n°58)	181252	0,34
17	ANGERS	rue Pierre Blandin 2ème (n°58)	181253	0,34
18	ANGERS	rue Pierre Blandin 2ème (n°37 - 39)	181251	0,3
19	ANGERS	Rue de la Lande (n° 30 - 34)	181226	0,3
20	ANGERS	Tours gaubert, T1 pierre gaubert	181239	0,3
21	ANGERS	Tours gaubert, T1 pierre gaubert 2eme	181240	0,3
22	ANGERS	Tours gaubert, T5 pierre gaubert	181241	0,25
23	ANGERS	Tours gaubert, T7 (2ème) pierre gaubert	181244	0,34
24	ANGERS	Tours gaubert, T7 pierre gaubert	181243	0,34
25	ANGERS	Tours gaubert, T5 (2ème) pierre gaubert	181242	0,3
26	ANGERS	Tours gaubert, T3 (2ème)Bd beaussier	181246	0,26
27	ANGERS	Tours gaubert, T3 Bd beaussier	181245	0,53
28	ANGERS	bvd Beaussier (n°37)	181224	0,26
29	ANGERS	rue de la Barre	181362	0,2
30	ANGERS	salle festive du lac de maine	181328	0,15
	TOTAL	**30 conteneurs**	**4h20mn**	**8,47**

semaine avec un gain de 10% au niveau du coût par rapport à l'ancien système de collecte des conteneurs enterrés. Ces résultats nous semblent très intéressants et nous motivent pour effectuer une étude plus approfondie.

7.5 Bilan

Dans ce chapitre a été présentée l'application de *MAF-DISTA* à un problème de tournée de véhicules pour la collecte des conteneurs enterrés. L'approche *MAF-DISTA* semble pouvoir être adaptée aisément à la résolution de ce type de problème. La complexité du problème justifiant une approche évolutionnaire, l'accent est mis dans *MAF-DISTA* sur une utilisation adéquate des opérateurs génétiques pouvant être définis et les différentes transformations ayant eu lieu pour résoudre un problème de tournées avec des exigences particulaires, no-

TAB. 7.2 – Tournée n° 2

N	Commune	Point de collecte	N° conteneur	Tonnage
1	ANGERS	Place Lafayette	181127	0,3
2	ANGERS	Place Lafayette	181284	0,32
3	ANGERS	Place Lafayette	181285	0,27
4	ANGERS	place sémard (4m3)	181257	0,5
5	ANGERS	Place de la Visitation (4m3)	181046	0,52
6	ANGERS	place st eloi (4m3)	181256	0,4
7	ANGERS	place maurice saillant (4m3)	181255	0,52
8	ANGERS	Place Hérault	181 281	0,5
9	ANGERS	Place Imbach / Guitton (Eglise)	181269	0,48
10	ANGERS	Place Imbach / Pocquet de Livonnières	181270	0,31
11	ANGERS	rue de rennes (parking miterrand)	181345	0,2
12	ANGERS	rue de rennes/Avenue de la Constitution	181346	0,28
13	ANGERS	Place Olivier Giran	181 274	0,4
14	ANGERS	Place Olivier Giran Bas	181 276	0,38
15	ANGERS	Place Olivier Giran Haut	181 275	0,33
16	ANGERS	30 A rue saint Exupéry	181354	0,1
17	ANGERS	30 A rue saint Exupéry supplémentaire	181355	0,3
18	ANGERS	22 D rue Saint Exupéry	181353	0,5
19	ANGERS	2 rue du Petit prince	181350	0,34
20	ANGERS	2 rue du Petit prince supplémentaire	181351	0,3
21	ANGERS	6 rue du Petit Prince	181352	0,41
22	ANGERS	10 rue Saint exupery	181348	0,4
23	ANGERS	10 rue Saint exupery supplémentaire	181349	0,15
24	ANGERS	Angle rue louis/Gain st Exupery	181347	0,3
25	ANGERS	Béjonnières	181265	0,23
26	ANGERS	Square Luther King	181131	0,25
27	ANGERS	71 av. Jean XXIII (accès par Rue des Bonnelles)	181309	0,25
28	ANGERS	75 av. Jean XXIII (accès par Rue des Bonnelles)	181310	0,25
	TOTAL	**28 conteneurs**	**4h00mn**	**9,49**

tamment, le respect des règles de la circulation. Une instance de test a été générée à partir des données réelles fournies par les services de la direction Déchets-Environnement d'Angers-Loire Métropole. Un ensemble d'expérimentations ont été menées afin de fournir les différentes tournées de collecte respectant la capacité des véhicules, les règles de circulation ainsi que le temps de travail des agents. Un plan interactif a été mis en place pour faciliter le déploiement des résultats obtenus par les opérateurs (chauffeurs). Enfin, des tests sur le terrain ont confirmé l'efficacité de *MAF-DISTA* dans la résolution de ce type de problème. En effet, l'utilisation de notre approche a permis de réduire significativement le nombre de tournées. Ce nombre est passé de 8 tournées à 7 tournées par semaine avec un gain de 10% au niveau du coût par rapport à l'ancien système de collecte des conteneurs enterrés.

À travers cette application, nous avons tout d'abord souhaité illustrer la démarche de spécialisation dans *MAF-DISTA*.

L'objectif recherché avec $MAF - DISTA$ n'est pas seulement de proposer une approche efficace sur un problème particulier, mais de fournir aussi une plate-forme flexible qui exploite

Tab. 7.3 – Tournée n° 3

N	Commune	Point de collecte	N° conteneur	Tonnage
1	PONTS DE CE	Rue de l'Amiral Chauvin (n°3 - 9)	19124	0,38
2	PONTS DE CE	Rue de l'Amiral Chauvin (11 - 21)	19125	0,38
3	PONTS DE CE	Rue du Petit Pouillé (n°12)	19123	0,32
4	PONTS DE CE	Rue du Petit Pouillé (n°10)	19107	0,31
5	TRELAZE	Rue du 14 juillet - Les Campanules	17128	0,5
6	TRELAZE	Rue du 14 juillet - La Tour	17127	0,32
7	TRELAZE	Rue du 14 juillet - Les Fougères	17126	0,35
8	TRELAZE	bd de la république - Les Primevères	17125	0,3
9	TRELAZE	Bd de la république - Les Canaris	17124	0,38
10	TRELAZE	Rue Chevrollier n°6	17129	0,35
11	TRELAZE	Rue Chevrollier n°14 (place)	17102	0,52
12	TRELAZE	Avenue Mendès France (2ème)	17134	0,2
13	TRELAZE	Avenue Mendès France (33-35 Chevrollier)	17133	0,2
14	TRELAZE	Place Picasso	17135	0,25
15	TRELAZE	Rue Elysée reclus (n°29 - 37)	17101	0,45
16	TRELAZE	Rue Elysée reclus (n°1 -3)	17130	0,45
17	ANGERS	32 bd jacques Millot	181319	0,2
18	ANGERS	32 bd jacques Millot 2ème	181320	0,28
19	ANGERS	93 boulevard Bédier (La Retraite)	181295	0,35
20	ANGERS	65 rue de la Morellerie (La Retraite)	181296	0,3
21	ANGERS	51 rue de la Morellerie (La Morellerie)	181297	0,25
22	ANGERS	49 rue de la Morellerie (La Morellerie)	181298	0,3
23	ANGERS	49 rue de la Morellerie (La Morellerie) 2ème	181299	0,32
24	ANGERS	45 rue de la Morellerie (La Morellerie)	181300	0,34
25	ANGERS	43 rue de la Morellerie (La Morellerie)	181302	0,32
26	ANGERS	41 rue de la Morellerie (La Morellerie)	181301	0,3
	TOTAL	**26 conteneurs**	**3h30mn**	**8,62**

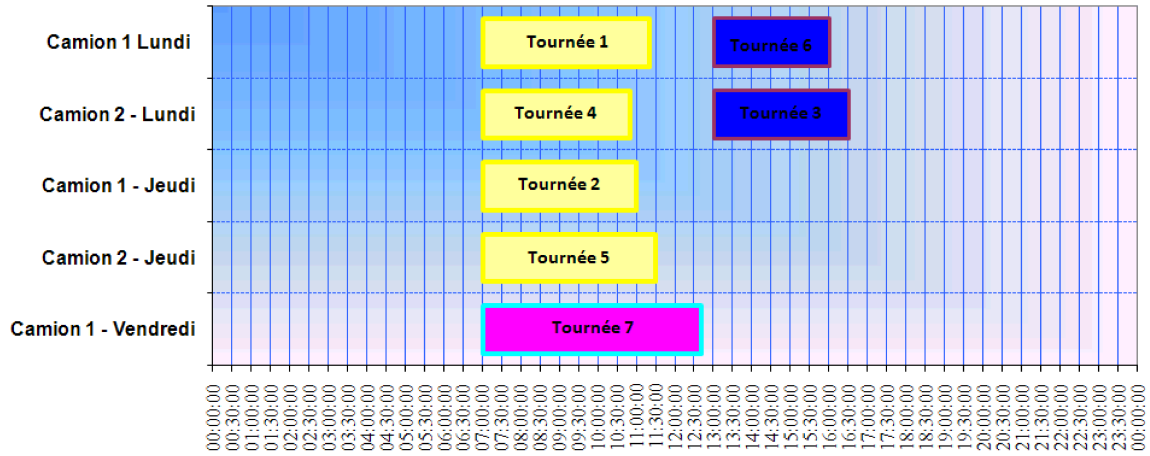


Fig. 7.4 – Planification des tournées

TAB. 7.4 – Tournée n° 4

N	Commune	Point de collecte	N° conteneur	Tonnage
1	ST BARTHELEMY	Centre Technique	10120	0,3
2	ANGERS	petit papillai 1 Daguenet	181335	0,26
3	ANGERS	petit papillai 2 Daguenet	181336	0,33
4	ANGERS	petit papillai 3 Daguenet	181337	0,31
5	ANGERS	petit papillai 4 Daguenet	181338	0,33
6	ANGERS	Marché Grand Pigeon	181323	0,3
7	ANGERS	Boulevrad des 2 croix	181357	0,2
8	ANGERS	Rue Levavasseur	181324	0,26
9	ANGERS	Rue Levavasseur	181325	0,3
10	ANGERS	2 rue Comte de tourvil	181342	0,25
11	ANGERS	2 rue Comte de tourvil	181343	0,25
12	ANGERS	comte de tourvil	181340	0,2
13	ANGERS	Villeroy/Mareuil	181341	0,2
14	ANGERS	Grand Nozay Trémolières	181332	0,35
15	ANGERS	Grand Nozay Trémolières	181333	0,32
16	ANGERS	Grand Nozay Trémolières	181334	0,31
17	ANGERS	Marché Monplaisir	181294	0,28
18	ANGERS	Marché Monplaisir	181321	0,3
19	ANGERS	Marché Monplaisir	181322	0,31
20	ANGERS	Place de l'Europe	181291	0,32
21	ANGERS	Place de l'Europe (2ème)	181292	0,32
22	ANGERS	Boulevard Schuman (Mimosas)	181289	0,4
23	ANGERS	Boulevard Schuman (Pavot)	181288	0,35
24	ANGERS	Square Schuman (Volubilis)	181290	0,3
25	ANGERS	Boulevard Monplaisir (Pensées)	181287	0,54
26	ANGERS	Boulevard Monplaisir (Résédas)	181286	0,3
	TOTAL	**26 conteneurs**	**3h50mn**	**7,89**

les avantages des mécanismes d'adaptation des choix de méthodes de résolution et de la distribution des calculs. Pour renforcer la présentation de l'aspect générique de $MAF - DISTA$, nous proposons dans le chapitre suivant une deuxième application portant sur un problème de planification et d'ordonnancement des avances et des retards de production.

Les travaux de ce chapitre ont fait l'objet d'une publication dans un journal international (Belkhelladi *et al.* (2010b)).

TAB. 7.5 – Tournée n° 5

N	Commune	Point de collecte	N° conteneur	Tonnage
1	ANGERS	place de la poissonnerie	181356	0,25
2	ANGERS	Boulevard du Ronceray (en face de l'ENSAM)	181215	0,25
3	ANGERS	Capitainerie	181293	0,26
4	ANGERS	Place Bordillon Marché	181326	0,28
5	ANGERS	Place Bordillon	181126	0,5
6	ANGERS	Place de la Laiterie	181027	0,52
7	ANGERS	Place bichon Marché	181327	0,3
8	ANGERS	Rue des Petites Pannes	181317	0,25
9	ANGERS	Rue des Petites Pannes 2éme	181318	0,2
10	ANGERS	petit rocher rue raoul ponchon	181329	0,51
11	ANGERS	petit rocher rue raoul ponchon	181330	0,2
12	ANGERS	Rue Raoul Ponchon,à l'interieur	181331	0,38
13	ANGERS	Général Lizé (n°18-22)	181219	0,5
14	ANGERS	Rue Jean Bourré	181 280	0,5
15	ANGERS	Rue Yvette (n°13-17)	181223	0,26
16	ANGERS	Rue Renée (n°105 -115)	181221	0,26
17	ANGERS	Rue Renée 2ème (n°105 -115)	181259	0,26
18	ANGERS	Rue Yvette (n°1-11)	181222	0,27
19	ANGERS	Rue Yvette (n°1-11) 2ème	181260	0,25
20	ANGERS	Rue Pelluaud / Rue Thérèse	181278	0,27
21	ANGERS	Rue Général Lizé / Rue Pelluaud	181277	0,27
22	ANGERS	Rue Pelluaud / Rue Tranchant	181 279	0,27
23	ANGERS	René Tranchant (n°8)	181220	0,27
24	ANGERS	René Tranchant (n°1)	181258	0,25
25	ANGERS	Abel Chantereau (Angers Habitat)	181268	0,27
26	ANGERS	bvd Beaussier (n°10)	181225	0
27	ANGERS	Rue Quémard (n°61)	181227	0,2
28	ANGERS	Rue Quémard (n°66)	181234	0,24
29	ANGERS	rue Pierre Blandin (n°58)	181252	0,22
30	ANGERS	rue Pierre Blandin 2ème (n°58)	181253	0
31	ANGERS	Rue de la Lande (n° 30 - 34)	181226	0,25
32	ANGERS	Tours gaubert, T1 pierre gaubert	181239	0,2
33	ANGERS	Tours gaubert, T1 pierre gaubert 2eme	181240	0,29
34	ANGERS	Tours gaubert, T5 pierre gaubert	181241	0
35	ANGERS	bvd Beaussier (n°37)	181224	0,26
	TOTAL	**35 conteneurs**	**4h30mn**	**9,46**

Tab. 7.6 – Tournée n° 6

N	Commune	Point de collecte	N° conteneur	Tonnage
1	ANGERS	Béjonnières	181265	0,27
2	ANGERS	Rue Mal Juin / Bd Portet	181264	0,38
3	ANGERS	Square Luther King	181131	0,32
4	ANGERS	rue Jan pallach	181305	0,28
5	ANGERS	rue Jan pallach (2ème)	181306	0,32
6	ANGERS	67 et 69 av. Jean XXIII (accès par Rue des Bonnelles)	181307	0,32
7	ANGERS	67 et 69 av. Jean XXIII (accès par Rue des Bonnelles)	181308	0,32
8	ANGERS	71 av. Jean XXIII (accès par Rue des Bonnelles)	181309	0,32
9	ANGERS	75 av. Jean XXIII (accès par Rue des Bonnelles)	181310	0,32
10	ANGERS	75 av. Jean XXIII (accès par Rue des Bonnelles) 2ème	181311	0,16
11	ANGERS	2 square dumontd'urville (rouge + vert)	181312	0,35
12	ANGERS	4 square dumontd'urville (bleu + jaune)	181313	0,35
13	ANGERS	6 square dumontd'urville (jaune + rouge)	181314	0,35
14	ANGERS	8 square dumontd'urville (vert + bleu)	181315	0,35
15	ANGERS	10 square dumontd'urville (bleu + jaune)	181316	0,35
16	ANGERS	les pleïades, bat A 9 rue bergson	181235	0,3
17	ANGERS	les pleïades, bat A 9 rue bergson 2eme	181261	0,3
18	ANGERS	les pleïades, bat B 11 bd arbrissel	181236	0,6
19	ANGERS	les pleïades, bat c 13 bd arbrissel	181237	0,22
20	ANGERS	les pleïades, bat c 13 bd arbrissel 2eme	181238	0,52
21	ANGERS	rue Gagarine	181361	0,2
22	ANGERS	rue Paul Claudel	181360	0,2
23	ANGERS	6 rue Charles Beaudelaire	181358	0,2
24	ANGERS	6 rue charles beaudelaire supplémentaire	181359	0,2
25	ANGERS	Agathides	181262	0,34
26	ANGERS	Agathides (2ème)	181263	0,4
	TOTAL	**26 conteneurs**	**3h00mn**	**8,24**

TAB. 7.7 – Tournée n° 7

N	Commune	Point de collecte	N° conteneur	Tonnage
1	PONTS DE CE	Rue du Petit Pouillé (n°12)	19123	0,28
2	TRELAZE	Rue du 14 juillet - Les Campanules	17128	0,3
3	TRELAZE	Rue du 14 juillet - La Tour	17127	0,28
4	TRELAZE	Rue du 14 juillet - Les Fougères	17126	0,26
5	TRELAZE	Bd de la république - Les Canaris	17124	0,28
6	TRELAZE	Rue Chevrollier n°6	17129	0,24
7	TRELAZE	Rue Elysée reclus (n°29 - 37)	17101	0,3
8	ANGERS	Boulevard Monplaisir (Résédas)	181286	0,24
9	ANGERS	Boulevard Schuman (Mimosas)	181289	0,25
10	ANGERS	Place de l'Europe (2ème)	181292	0,28
11	ANGERS	Place de l'Europe	181291	0,26
12	ANGERS	Marché Monplaisir	181322	0,27
13	ANGERS	Marché Monplaisir	181321	0,1
14	ANGERS	Marché Monplaisir	181294	0,1
15	ANGERS	Grand Nozay Trémolières	181333	0,28
16	ANGERS	Grand Nozay Trémolières	181332	0,28
17	ANGERS	Abel Chantereau (Angers Habitat)	181268	0,23
18	ANGERS	René Tranchant (n°1)	181258	0,1
19	ANGERS	René Tranchant (n°8)	181220	0,27
20	ANGERS	Rue Général Lizé / Rue Pelluaud	181277	0,23
21	ANGERS	Rue Pelluaud / Rue Thérèse	181278	0,22
22	ANGERS	Rue Yvette (n°1-11)	181222	0,25
23	ANGERS	Rue Renée 2ème (n°105 -115)	181259	0,24
24	ANGERS	Rue Renée (n°105 -115)	181221	0,1
25	ANGERS	Rue Yvette (n°13-17)	181223	0,22
26	ANGERS	93 boulevard Bédier (La Retraite)	181295	0,25
27	ANGERS	65 rue de la Morellerie (La Retraite)	181296	0,25
28	ANGERS	49 rue de la Morellerie (La Morellerie)	181298	0,25
29	ANGERS	49 rue de la Morellerie (La Morellerie) 2ème	181299	0,26
30	ANGERS	45 rue de la Morellerie (La Morellerie)	181300	0,24
31	ANGERS	43 rue de la Morellerie (La Morellerie)	181302	0,25
32	ANGERS	41 rue de la Morellerie (La Morellerie)	181301	0,1
33	ANGERS	2 square dumontd'urville (rouge + vert)	181312	0,3
34	ANGERS	4 square dumontd'urville (bleu + jaune)	181313	0,3
35	ANGERS	6 square dumontd'urville (jaune + rouge)	181314	0,3
36	ANGERS	8 square dumontd'urville (vert + bleu)	181315	0,3
37	ANGERS	10 square dumontd'urville (bleu + jaune)	181316	0,3
38	ANGERS	les pleïades, bat A 9 rue bergson	181235	0,26
39	ANGERS	les pleïades, bat A 9 rue bergson 2eme	181261	0,25
40	ANGERS	Agathides	181262	0,22
	TOTAL	**40 conteneurs**	**5h40mn**	**9,69**

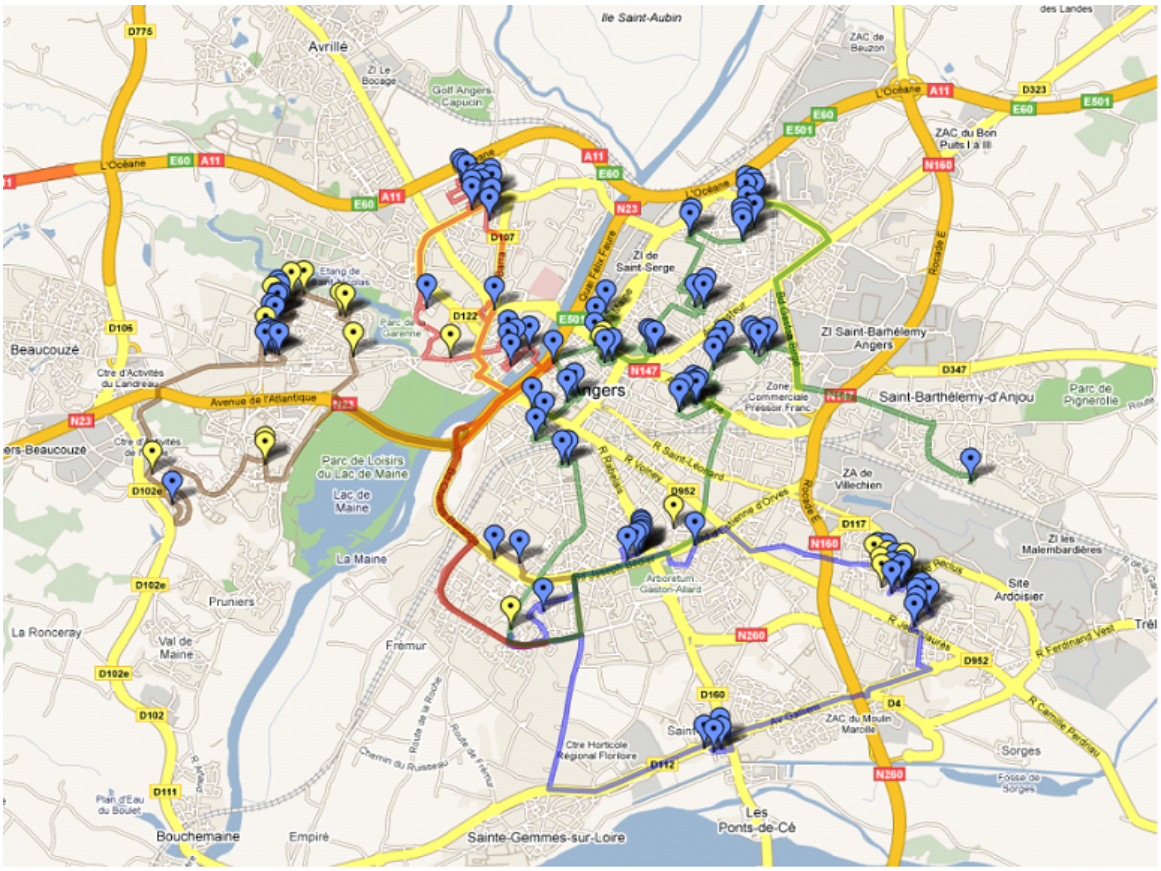


Fig. 7.5 – Plan des tournées (source : Google Maps)

Chapitre 8

Application de MAF-DISTA à la résolution du problème de planification et ordonnancement multi-produit

Sommaire

8.1 Introduction

Dans le contexte actuel de compétition internationale croissante, l'efficacité et la productivité sont devenues des données critiques de l'évaluation de la santé d'une entreprise. La gestion de la production et des ressources humaines constitue un moyen d'action important, pour une société, d'influer sur ces facteurs.

Les économies conséquentes réalisables par une organisation de bonne qualité de ces fonctions poussent à y affecter une grande part du temps de travail de leurs décideurs (chefs d'équipe, responsables de services, etc.), car l'agencement de ces éléments met souvent en jeu des problématiques complexes soumises à un ensemble de règles de plus en plus fourni, qu'elles soient physiques, législatives ou sociales. Devant ce constat, il devient clairement utile de développer des stratégies d'automatisation et d'optimisation de ces processus, afin, d'une part, de soulager et d'assister le décideur et, d'autre part, d'exploiter la puissance sans cesse grandissante des nouvelles technologies pour proposer des solutions d'efficacité accrue.

Ce chapitre est consacré à l'étude de l'application de *MAF-DISTA* à la résolution d'un troisième problème d'optimisation. Le problème traité est celui de la planification des avances/retards de production (*Earliness-Tardiness Planning and Scheduling Problem*), issu de la politique industrielle du *juste-à-temps*. Le but recherché en expérimentant à nouveau la démarche de spécialisation est d'insister sur la possibilité offerte d'un développement générique, mettant en jeu des composants simples et produisant au final une métaheuristique distribuée, robuste, et compétitive. C'est la conjonction de ces divers facteurs considérés suivant le paradigme multi-agents et pris ensemble qui constituent le cœur de la méthode et qui doivent justifier et motiver le choix de son utilisation.

En nous focalisant tout d'abord sur la démarche de spécialisation, nous précisons comment est introduite la connaissance du problème via la conception d'opérateurs génétiques. Ensuite, des évaluations comparatives avec des approches connues les plus performantes sur le problème sont présentées. En nous appuyant sur l'utilisation de jeux de tests de grande taille, considérés comme représentatifs de la difficulté du problème, nous établirons un panorama comparatif des approches prenant en compte la qualité des solutions et le temps d'exécution. Nous chercherons à montrer, qu'en dépit de sa simplicité et de sa généricité, l'approche *MAF-DISTA* n'en est pas moins compétitive vis-à-vis des approches les plus performantes.

8.2 ETPSP

8.2.1 Présentation du problème

Le problème de planification et d'ordonnancement des avances/retards de production (*ETPSP*) a attiré beaucoup d'attention ces dernières années. En pratique, il fournit un moyen efficace d'intégrer la planification des ressources de production (*MRP-II*) avec l'ordonnancement *juste-à-temps*.

Nous considérons un système de production qui fabrique n types de produits pour satisfaire les demandes des clients externes dans un horizon fini de périodes T. Chaque produit doit passer par M process (étapes d'assemblage) à partir des matières premières jusqu'au produit final.

Données :

N : Le nombre de produits (items).
i : Indice des produits, i=1..N.
M : Le nombre de process (étapes d'assemblage).
j : Indice des process, j= 1..M.
T : Horizon de planification.
K : Indice des périodes, k=1..T.
$Produit_i$: Le nom du i^e produit.
$Process_j$: Le nom du j^e process.
$Periode_k$: Le nom de la k^e période dans l'horizon de planification.
$d_i(k)$: La demande externe du produit i à la période k.
$c_j(k)$: La capacité disponible du process j à la période k.
w_{ij} : Nombre constant d'unités de produit i nécessaire pour le process j.
l_i : Le stock initial du produit i.
α_i : Pénalité d'avance du produit i.
β_i : Pénalité de retard du produit i. En général, $\alpha_i > \beta_i$, α_i et β_i peuvent être déterminés par les coûts de stockage et la compensation des retards en pratique.
s_i : La taille du lot du produit i.
$p_i(k)$: La quantité de produit i à fabriquer à la période k.

Le *ETPSP* consiste à trouver un ordonnancement optimal des tailles de lot dans un horizon de planification et d'ordonnancement, tel que le coût total des pénalités d'avance et de retard est minimisé tout en respectant les contraintes de capacité des ressources. En raison des différences dans les tailles de lot, le problème *ETPSP* est discret et est décrit par le modèle suivant (Wang (1995)) :

$$(P)min_p = \sum_{i=1}^{N}\sum_{k=1}^{T}\{\alpha_i[l_i + \sum_{t=1}^{k} p_i(t) - \sum_{t=1}^{k} d_i(t)]^+ + \beta_i[\sum_{t=1}^{k} d_i(t) - \sum_{t=1}^{k} p_i(t) - l_i]^+\} \quad (8.1)$$

$$\sum_{i=1}^{n} w_{ij}p_i(k) \leq c_j(k), j = 1..M, k = 1..T \quad (8.2)$$

$$0 \leq p_i(k) \in S_i, S_i = \{r.s_i, r \geq 0\}, i = 1..N, k = 1..T \quad (8.3)$$

avec $(x)^+ = max\{0, x\}$.

Notons que la fonction objectif (8.1) du modèle (P) est non linéaire et les contraintes (8.2) et (8.3) sont des fonctions discrètes, le modèle (P) ne peut pas être résolu par les méthodes de la programmation linéaire telles que le *Simplex*.

8.3 MAF-DISTA pour ETPSP

Après avoir formulé et cité les difficultés du *ETPSP*, nous avons décidé d'utiliser un algorithme génétique comme approche de résolution. Dans cette section, nous détaillons les différentes parties de l'algorithme génétique développé pour le *ETPSP*. Puis, nous adaptons

notre plate-forme *MAF-DISTA* pour l'exécution distribuée de l'algorithme génétique développé ici.

8.3.1 Algorithme génétique pour le ETPSP

Nous avons repris une partie des travaux effectués par Li *et al.* (1998) pour mettre en œuvre un algorithme génétique pour le *ETPSP*.

8.3.1.1 Codage d'une solution

Nous avons utilisé un codage réel pour représenter les individus. Ceci est justifié par le nombre important des paramètres mis en jeu pour l'optimisation. Le chromosome est représenté par une chaîne de nombres réels des quantités de production comme suit :

$$p_1(1)p_1(2)p_1(3)...p_1(T)p_2(1)p_2(2)p_2(3)...p_2(T)...p_N(1)p_N(2)p_N(3)...p_N(T)$$

Comme déjà évoqué par Ip *et al.* (2000), la valeur de la fonction objectif est considérée comme la règle d'évaluation de chaque individu. Parfois, le meilleur et le pire des chromosomes produisent presque les mêmes enfants après le croisement, ce qui provoque *la convergence prématurée* de la population. Dans ce cas, l'effet de la sélection naturelle n'est donc pas évident. Pour résoudre ce problème, nous avons effectué une normalisation linéaire, qui convertit les évaluations des chromosomes en un fitness. Ceci est décrit dans l'algorithme 7.

Algorithme 7 Evaluation des chromosomes ($Fitness$)

1: Initialiser les paramètres de contrôle : une valeur constante c et un taux de décrementation r, l'évaluation $E = \{e_i, 1 \leqslant i \leqslant s\}$ avec s pour la taille de la population
2: Trier les individus par ordre décroissant des évaluations, $E' = \{e_i^{'}, \forall 1 \leqslant i \leqslant j \leqslant s, e_i^{'} \leqslant e_j^{'}, \forall e_i^{'} \in E\}$
3: Calculer le fitness F_i correspondant à E', F_i commence par c, et décroît linéairement avec le taux r, $F = \{f_i; f_i = c + (i-1) \times r, i = 1, 2, .., s\}$

La valeur d'évaluation est remplacée par la valeur du fitness pour la sélection des parents.

8.3.1.2 Génération de la population initiale

Nous avons utilisé une méthode de génération aléatoire des chaînes de nombres réels. Cette approche est donnée par l'algorithme 8.

Dans cet algorithme, la satisfaction des contraintes du problème peut être déterminée par l'algorithme 9.

8.3.1.3 Sélection

Nous avons choisi de mettre en œuvre la méthode de sélection proportionnelle *RWS* (cf. annexe B). L'algorithme 10 de cette méthode est détaillé ci-dessous :

Algorithme 8 Génération de la population initiale

1: Initialiser les paramètres : indice $j = 1$, la taille de la population s et la population $P = \emptyset$;
2: Générer aléatoirement une chaîne de nombre réels,
$P_j = p_1(1)p_1(2)..p_1(T)p_2(1)p_2(2)...p_2(T)...p_N(1)p_N(2)..p_N(T)$
$p_i(k)$ représente la quantité du produit fabriquée à la période k.
3: Si P_j est réalisable (algorithme 9), aller à 4, sinon aller à 2,
4: Si P_j est différent de tous les individus de la population alors $P = P \cup \{P_j\}; j = j + 1$; sinon aller à 2,
5: Si $(j > s)$ alors $P = \{P_1, P_2, .., P_s\}$ et fin ; sinon aller à 2.

Algorithme 9 Test de satisfaction des contraintes

1: $P_t = \{p_{t1}, p_{t2}, .., p_{ts}\}$ un nouveau individu ;
2: Si $\sum_{i=1}^{n} w_{ij} p_i(k) \leq c_j(k), (j = 1..M, k = 1..T)$ et $p_{ti}(k) \geqslant 0, (i = 1..N, k = 1..T)$ alors P_t est réalisable, sinon P_t n'est pas réalisable.

Algorithme 10 La méthode de sélection proportionnelle *RWS*

```
pick = Random[0,1]  // Choix d'un nombre aléatoire entre 0 et 1
Stotal = Somme total des fonctions objectifs de tous les individus de la population
S = 0
i = 0  // individu choisi
répéter
    S = S + FonctionObjectif[i]
    i=i+1
jusqu'à pick > S/Stotal
```

8.3.1.4 Croisement

Nous avons utilisé trois opérateurs de croisement. Il s'agit du croisement simple (SX)(Wright (1991), Michalewicz (1996)), le croisement linéaire (LX)(Wright (1991)) et le croisement à 2 points de coupure $(X2)$.

Croisement simple : Soit deux parents $P_1 = (p_1^1...p_1^n)$ et $P_2 = (p_1^2...p_n^2)$, le croisement SX[1] consiste à choisir aléatoirement une position $i \in \{1, 2, ..., n-1\}$ et les deux enfants E_1 et E_2 sont produits de la façon suivante :

$$E_1 = (p_1^1, p_2^1, ..., p_i^1, p_{i+1}^2, ..., p_n^2) \tag{8.4}$$

$$E_2 = (p_1^2, p_2^2, ..., p_i^2, p_{i+1}^1, ..., p_n^1) \tag{8.5}$$

Croisement Linéaire : Dans le croisement linéaire trois enfants $E_k = (e_1^k, ..., e_i^k, ..., e_n^k), k = 1, 2, 3$ sont produits tels que :

$$e_i^1 = \frac{1}{2} p_i^1 + \frac{1}{2} p_i^2 \tag{8.6}$$

[1] Simple Crossover

$$e_i^2 = \frac{3}{2}p_i^1 - \frac{1}{2}p_i^2 \tag{8.7}$$

$$e_i^3 = \frac{1}{2}p_i^1 + \frac{3}{2}p_i^2 \tag{8.8}$$

Enfin, parmis les trois enfants produits par le croisement, nous choisissons les deux meilleurs en fonction de leurs fitness pour remplacer les parents dans la population.

Croisement à deux points de coupure : Le principe du croisement à deux points de coupure est représenté dans la figure 8.1.

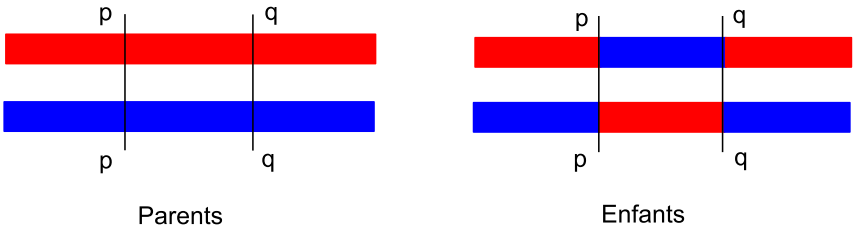


FIG. 8.1 – Croisement à 2 points de coupure

8.3.1.5 Mutation

En général, dans les algorithmes génétiques, les mutations consistent à modifier un bit du chromosome. Nous avons utilisé deux opérateurs de mutation pour le *ETPSP*. Il s'agit de la mutation non uniforme (NUM[2]) et la mutation par glissement réel (RCM[3]).

Soit $P = (p_1, p_2, ..., p_i, ..., p_n)$ un chromosome et $p_i \in [a_i, b_i]$ le gène à muter dans P. Ensuite, le gène $p_i^{'}$ résultant de l'application des différents opérateurs de mutation.

Mutation non uniforme : Si la mutation non uniforme est appliquée à un chromosome à la génération g et g_{max} est le nombre maximum de générations, alors :

$$p_i^{'} = \begin{cases} p_i + \triangle\,(g, b_i - p_i)\ si\ \tau = 0 \\ p_i + \triangle\,(g, p_i - a_i)\ si\ \tau = 1 \end{cases} \tag{8.9}$$

τ un nombre binaire généré aléatoirement et

$$\triangle\,(g, y) = y[1 - r^{(1-\frac{g}{g_{max}})^b}] \tag{8.10}$$

Où r est un nombre aléatoire dans l'intervalle [0,1] et b est un paramètre choisi par l'utilisateur, qui détermine le degré de dépendance sur le nombre d'itérations. Cette fonction donne une valeur dans l'intervalle $[0, y]$ telle que la probabilité de retourner un nombre proche de zéro augmente au fur et à mesure du nombre de générations de l'algorithme. La taille de l'intervalle de génération de gènes doit être inférieure au fil des générations. Cette propriété

[2]Non-Uniform Mutation
[3]Real Creep Mutation

de l'opérateur permet d'effectuer une recherche uniforme dans l'espace de recherche initial lorsque g est petit (Herrera *et al.* (1998)).

Mutation par glissement réel : La mutation par glissement réel (RCM[4]) est définie dans l'algorithme 11.

Algorithme 11 La mutation par glissement réel *(RCM)*

1: Choisir un individu p_0,
2: Calculer le gradient de la direction négative au point $p = p_0$
$-\nabla F(p)|_{p=p_0} = -\nabla F(p_0)$,
3: Sélectionner un nombre réel aléatoire r,
4: Faire un glissement de r pas,
$p_{new} = p_0 + r \cdot (-\nabla F(p_0))$
5: Si p_{new} est réalisable alors fin ; sinon aller à 3.

Dans la suite de ce chapitre, nous nous intéressons à l'adaptation de *MAF-DISTA* pour l'optimisation de *ETPSP*. Nous reprenons donc les différentes définitions déjà fournies dans le cinquième chapitre. Chaque agent mobile de calcul dans *MAF-DISTA* encapsule une instance de l'algorithme génétique dont les différents composants sont donnés dans les sections précédentes. Il s'agit de l'ensemble des opérateurs de croisement, de mutation et de la sélection par la roue de la fortune biaisée. Enfin, le principe de fonctionnement de *MAF-DISTA* est résumé par le diagramme de séquence de la figure 8.2.

En ce qui concerne l'échange d'informations entres les agents, nous avons utilisé les deux stratégies de communication définies dans le chapitre 4. Il s'agit de la stratégie d'immigration et intégration sélectives et de la stratégie d'ajustement des paramètres d'un *AE* par le biais d'échange.

La section suivante décrit l'ensemble des résultats obtenus sur des instances de *ETPSP*.

8.4 Simulations et résultats

L'efficacité de la plate-forme *MAF-DISTA* dans la résolution du *ETPSP* est démontrée par le calcul sur les instances citées par Ip *et al.* (2000). Les données utilisées pour 2, 3 et 4 produits sont résumées dans les tableaux 8.1, 8.2, 8.3 et 8.4. Les deux premières lignes de chaque table sont pour le cas de 2 produits et les trois premières lignes sont pour les 3 produits.

Le tableau 8.1 fournit les pénalités d'avance (α_i), les pénalités de retard (β_i), les niveaux du stock initial ($l_i(0)$) et les tailles de lot (s_i) pour les produits (i). Le tableau 8.2 donne les demandes ($d_i(t)$) du produit i à la période t. Le tableau 8.3 présente la capacité (w_{ij}) du produit i nécessaire au process j. Le tableau 8.4 donne la capacité disponible ($c_j(t)$) du process j à la période t.

Les solutions obtenues sont données dans les tableaux 8.5, 8.6 et 8.7 respectivement pour le cas de 2 produits, le cas de 3 produits et le cas de 4 produits. Pour le cas du 2 produits, la valeur du fitness obtenue par *MAF-DISTA* est 479.1 contre $640,5$ obtenue par l'approche

[4]Real Creep Mutation

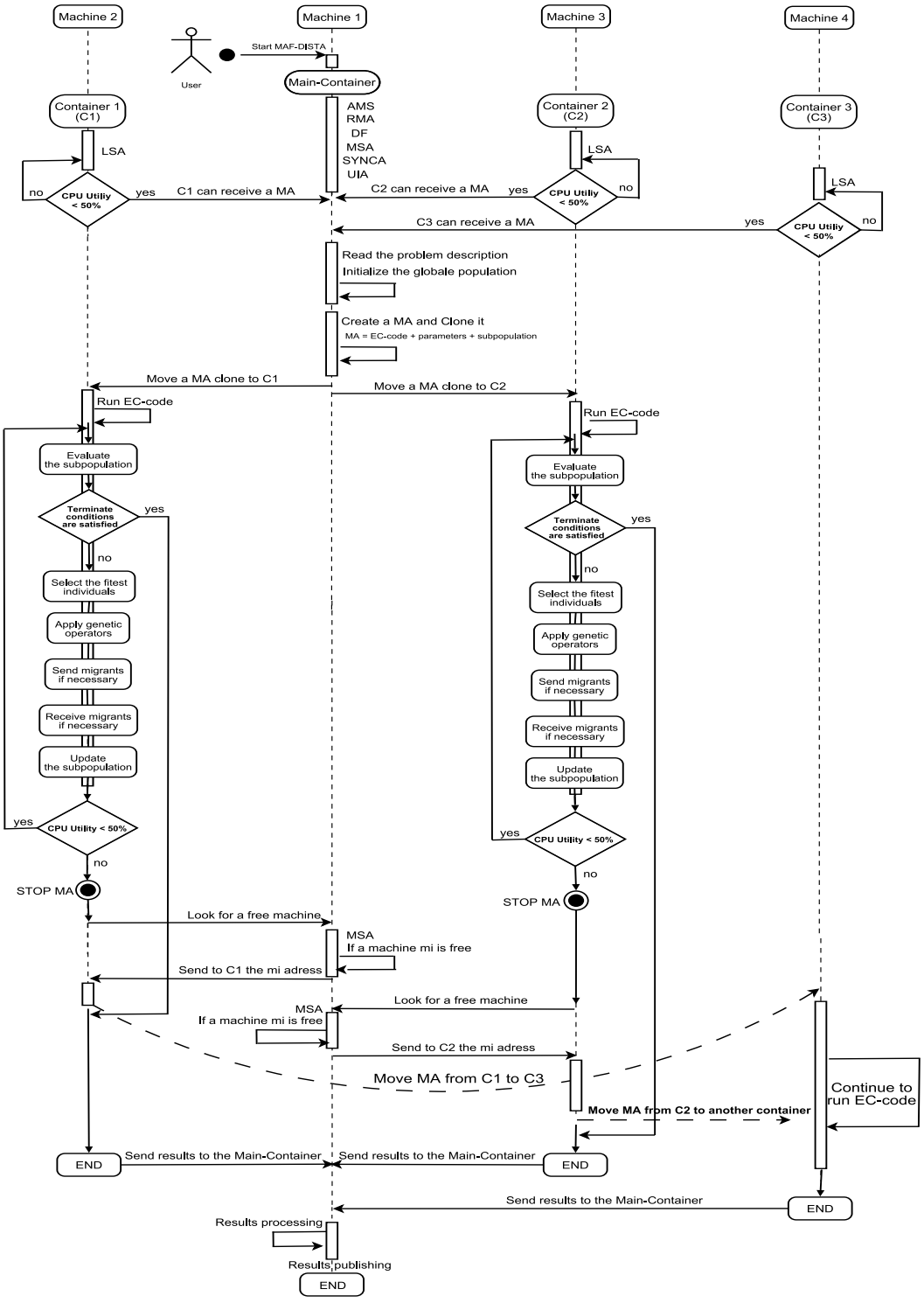


FIG. 8.2 – Diagramme de séquences de MAF-DISTA

TAB. 8.1 – Les pénalités d'avance et de retard, le stock initial et la taille du lot des produits

Produit	α_i	β_i	$l_i(0)$	s_i
1	5	15	0	5
2	10	20	0	10
3	5	10	0	5
4	5	15	0	5

TAB. 8.2 – La demande du produit i à la période j

Produit	Période 1	Période 2	Période 3	Période 4	Période 5	Période 6
1	0	0	20	0	0	30
2	10	0	0	50	0	0
3	20	0	20	0	0	5
4	0	0	10	40	0	10

TAB. 8.3 – la quantité nécessaire du produit i au process j

Produit	Process 1	Process 2	Process 3	Process 4	Process 5
1	1.0	0.6	0.8	0.3	0.7
2	0.6	0.8	1.3	2.0	0.7
3	0.1	0.2	0.2	0.3	0.1
4	0.3	0.2	0.1	0.4	0.2

TAB. 8.4 – Capacité du process j disponible à la période t

Process	Période 1	Période 2	Période 3	Période 4	Période 5	Période 6
1	30	30	30	30	30	30
2	18	28	18	18	18	18
3	34	44	34	34	34	34
4	24	34	24	44	19	24
5	26	36	26	46	26	26

génétique (AG) proposée par Ip *et al.* (2000). Pour le cas de 3 produits, le fitness obtenu par *MAF-DISTA* est $598,7$ contre $1057,5$ par l'AG. Pour le cas de 4 produits, le fitness obtenu par *MAF-DISTA* est 1178,67, contre 1455,5 par l'AG.

TAB. 8.5 – Solution obtenue par *MAF-DISTA* pour le cas de 2 produits

Variable	Produit	Période 1	Période 2	Période 3	Période 4	Période 5	Période 6
$p_i(t)$	1	0	2.500	17.500	0	0	30.000
	2	10.000	17	10	22.000	2.000	0
$l_i(t)$	1	0	2.500	0	0	0	0
	2	0	17	26.000	2.000	0	0

TAB. 8.6 – Solution obtenue par *MAF-DISTA* pour le cas de 3 produits

Variable	Produit	Période 1	Période 2	Période 3	Période 4	Période 5	Période 6
$p_i(t)$	1	0	7	14	0	2	29
	2	9.000	17	7	22.000	6.000	0
	3	20.000	0	20.000	0	0	5.000
$l_i(t)$	1	0	5.833	0	0	1.5	0
	2	1.000	15.125	22.000	6.000	0	0
	3	0	0	0	0	0	0

TAB. 8.7 – Solution obtenue par *MAF-DISTA* pour le cas de 4 produits

Variable	Produit	Période 1	Période 2	Période 3	Période 4	Période 5	Période 6
$p_i(t)$	1	0	8	15	0	17	13
	2	9.000	16.000	5.000	12.500	8	7
	3	20.000	0	20.000	0	0	5.000
	4	0	0	10.000	40.000	0	10.000
$l_i(t)$	1	0	7	0	0	14.5	0
	2	1.000	15.000	20.000	16	10	0
	3	0	0	0	0	0	0
	4	0	0	0	0	0	0

8.4.1 Performance du modèle d'ajustement des paramètres

La figure 8.3 montre les performances de notre approche d'adaptation automatique des paramètres. Nous remarquons que l'exécution de MAF-DISTA avec le modèle d'adaptation automatique des paramètres a trouvé une meilleure solution plus vite qu'une exécution sans ce modèle. Notons que l'utilisation du modèle d'adaptation des paramètres a permis de tester plusieurs jeux de paramètres sur chaque sous-population en cours d'exécution. Dans cette approche, les meilleurs jeux de paramètres sont souvent échangés entre les sous-populations. Ainsi, l'application du croisement et de la mutation a permis de produire de nouveaux jeux de paramètres.

8.4.2 L'effet de la charge CPU sur l'accélération des calculs

Dans cette expérience, onze machines étaient connectées à la plate-forme JADE dont une seule a été réservée pour l'initialisation des calculs évolutionnaires. Cette dernière est utilisée pour héberger le conteneur principal « Main-Container » pour permettre la création des agents par défaut AMS, DF et RMA. Ainsi, les utilisateurs du réseau sont autorisés à utiliser les dix autres machines. Chacune de ces machines est un conteneur « Container » qui comprenait les différents agents mobiles et un agent de statut local (LSA). En raison des différents accès effectués par les utilisateurs, les expériences ont été conduites cinq fois dans la même journée. La figure 8.4 montre les résultats obtenus. Comme on peut le constater, tous les calculs ont été accélérés par l'approche proposée. Elle montre l'efficacité de notre

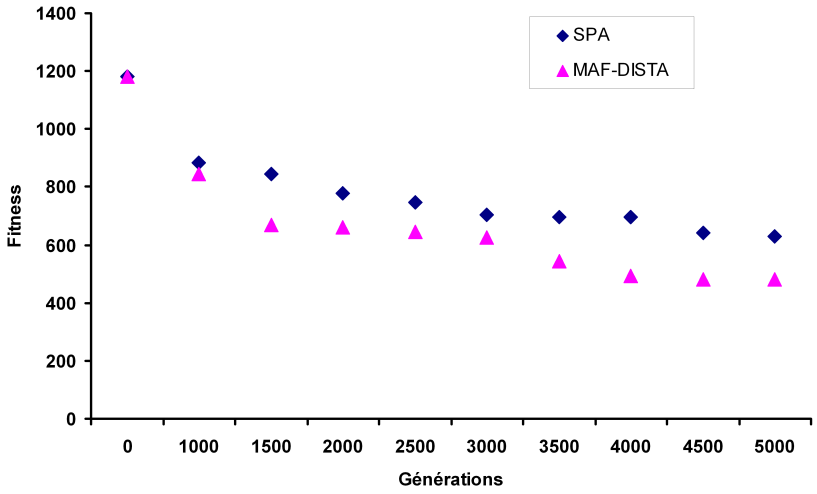


FIG. 8.3 – Performances du modèle d'adaptation des paramètres

approche basée sur les agents mobiles. Bien que le coût du calcul parallèle adaptatif est plus élevé que celui du calcul parallèle statique dans lequel les agents sont stationnaires, le premier est néanmoins une façon plus pratique pour paralléliser le calcul évolutionnaire dans un environnement réseau.

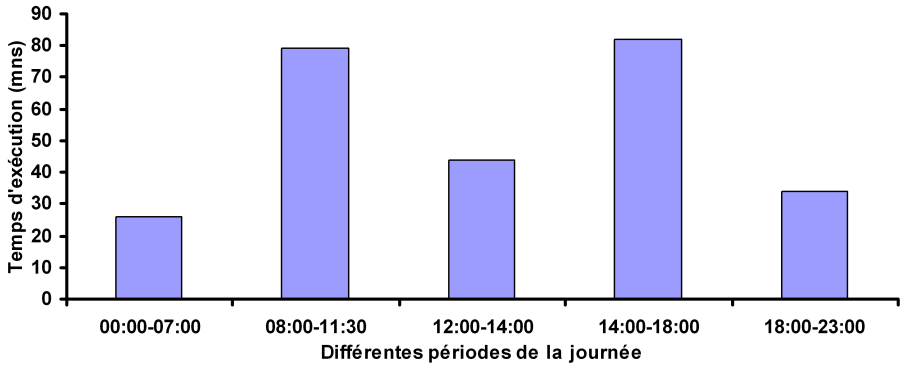


FIG. 8.4 – Effet de la charge CPU sur le temps de calcul

8.5 Conclusion

Nous avons présenté dans ce chapitre l'application de *MAF-DISTA* sur un problème de planification et d'ordonnancement des avances et des retards de production. L'objectif est d'illustrer le caractère générique de notre plate-forme.

L'approche *MAF-DISTA* semble pouvoir être adaptée aisément à la résolution de ce type de problème. La complexité du problème justifiant une approche évolutionnaire, l'accent est

mis dans *MAF-DISTA* sur une utilisation adéquate des opérateurs génétiques pouvant être définis pour résoudre un problème de planification et d'ordonnancement de production.

Les travaux de ce chapitre ont fait l'objet de deux communications (Belkhelladi *et al.* (2009a) ; Belkhelladi *et al.* (2009b)).

Partie 4 : Bilans

Chapitre 9

Conclusion générale et perspectives

9.1 Bilan

Tout au long de cette thèse, nous avons proposé un ensemble d'outils pour l'analyse et la conception de métaheuristiques parallèles à base de population pour l'optimisation combinatoire en les formulant dans le cadre des systèmes multi-agents. D'une part, nous avons souhaité proposer des outils qui encouragent la modularité et la réutilisation des modèles. D'autre part, l'accent a été mis sur la technologie d'agents mobiles et à son utilisation dans le domaine du calcul réparti. Cette technologie permet la conception d'applications flexibles et "dynamiquement" adaptables aux contraintes de l'environnement d'exécution ou de l'application elle-même. La mobilité permet d'adapter les services distants aux besoins des clients ; c'est une forme d'application répartie adaptable. La mobilité du code réduit la dépendance entre le programme et le site d'accueil et offre un premier degré de décentralisation des activités et une forte flexibilité pour le système (Arcangeli *et al.* (2002)).

Cette thèse réunit différentes notions de parallélisme, de paradigme multi-agents et de métaheuristiques afin d'apporter des méthodes de résolution performantes (robustes et auto-adaptatives) à des problèmes d'optimisation combinatoire réels. Il démontre que l'introduction de stratégies de parallélisation et d'échange d'informations dans un algorithme à population permet à ce dernier d'améliorer considérablement ses facultés de recherche de solutions. En outre, l'utilisation des agents mobiles permet une exploitation optimale des ressources de calcul inutilisées dans un organisme (laboratoire, entreprise) et favorise aussi l'autonomie des processus de calcul pour pouvoir gérer les éventuelles pannes dans un réseau.

Pour valider nos plates-formes de calcul réparti, nous avons traité trois problèmes d'optimisation : un problème de tournées sur arcs (CARP), un problème de tournées de véhicules pour la collecte des conteneurs enterrés (UWCOP) et un problème de planification et d'ordonnancement des avances et des retards de production (ETPSP). Pour ces trois applications, nous avons détaillé dans un premier temps la démarche de spécialisation qui consiste essentiellement en la définition des opérateurs génétiques à utiliser. Ensuite, nous avons mis en évidence un apport positif des mécanismes de coopération et d'auto-adaptation. Enfin, les résultats ont été comparés avec ceux obtenus avec plusieurs autres méthodes heuristiques existantes considérées dans la littérature comme étant parmi les plus puissantes sur ces problèmes. Cette évaluation comparative réalisée en termes de qualité des solutions et de temps d'exécution semble indiquer que le plate-forme *MAF-DISTA* est effectivement compétitive vis-à-vis des approches existantes.

Ces applications ont permis d'illustrer concrètement une démarche de conception de métaheuristiques parallèles fondées sur le paradigme d'agents mobiles.

Les différents résultats obtenus au cours de cette thèse nous permettent de conclure sur les contributions que nous apportons au domaine des métaheuristiques parallèles. Nous souli-

gnons aussi les apports significatifs de la technologie d'« agents mobiles » dans l'amélioration des performances d'une application distribuée. Ensuite, nous présentons les différents axes d'étude qui s'ouvrent à la suite de ce travail. Les travaux présentés dans cette thèse ont fait l'objet de publications dans des conférences nationales et internationales (Belkhelladi *et al.* (2007), Belkhelladi *et al.* (2008), Belkhelladi *et al.* (2009a), Belkhelladi *et al.* (2009b)), d'un chapitre de livre (Belkhelladi *et al.* (2010a)) et article de journal international (Belkhelladi *et al.* (2010b)).

9.2 Perspectives

L'architecture que nous avons proposée ne dépend pas du réseau utilisé ; ce qui nous amène à avancer que son utilisation sur un réseau dynamique soit possible. Un réseau dynamique est un réseau dans lequel certains liens de communication peuvent être temporairement perdus ou surchargés. Nous pensons que les algorithmes asynchrones seront mieux adaptés dans ce type du réseau.

À court terme, nous envisageons d'améliorer la partie interactive de MAF-DISTA pour faciliter son déploiement par différents types d'utilisateurs d'une part, et d'autre part, de tester *MAF-DISTA* sur d'autres problèmes d'optimisation réels. Des contacts sont en cours avec la direction des transports urbains de la ville de Dijon pour optimiser les tournées des bus.

À moyen terme, nous allons étendre *MAF-DISTA* pour prendre en compte d'autres méta-heuristiques à base de population telles que : les algorithmes à colonie de fourmis et à essaims de particules. D'autre part, de nouvelles expérimentations sont prévues sur des réseaux de grande taille pour traiter des instances de taille bien plus importante des problèmes traités dans cette thèse.

À long terme, nous envisageons de constituer une bibliothèque de problèmes modélisés et l'intégrer dans *MAF-DISTA*. Cette bibliothèque pourrait être exploitée à travers une interface Web.

Partie 5 : Annexes

Annexe A

Les algorithmes parallèles

A.1 Mesures de performance des algorithmes parallèles

annexe1-section1 Il existe plusieurs critères permettant d'évaluer la performance d'un algorithme parallèle. Cette section en présente quelques uns parmi les plus utilisés, comme l'accélération, l'efficacité, le nombre de processeurs et l'iso-efficacité (Delisle (2002)).

A.1.1 L'accélération

L'efficacité des algorithmes séquentiels est généralement mesurée par leur temps d'exécution en fonction de la taille de leur entrée (Cormen *et al.* (2002)). Considérons P comme étant un problème de calcul et n la taille de son entrée. On note $T^*(n)$ comme le temps d'exécution du meilleur algorithme séquentiel connu qui résout ce problème. La raison principale justifiant la conception d'algorithmes parallèles est en fait la réduction du temps d'exécution. Soit A, un algorithme parallèle qui résoud P dans un temps $T_p(n)$ sur un ordinateur parallèle avec p processeurs, l'accélération $S_p(n)$ obtenue par A est définie comme étant :

$$S_p(n) = \frac{T^*(n)}{T_p(n)} \tag{A.1}$$

$S_p(n)$ mesure donc le facteur d'accélération obtenu par un algorithme A quand p processeurs sont disponibles (Jaja (1992)). Il y a cependant une limite d'accélération qu'un algorithme peut atteindre. En effet, certaines parties d'un programme sont facilement parallélisables, alors que d'autres sont strictement séquentielles. Quand un grand nombre de processeurs est disponible, les parties parallélisables sont exécutés plus rapidement, mais les sections qui doivent demeurer séquentielles ne gagneront pas de vitesse. Cette observation est connue sous le nom de la loi d'Amdahl (2000) et s'exprime comme suit : si un programme consiste en deux sections, une qui est strictement séquentielle et une qui est pleinement parallélisable, et si la section strictement séquentielle utilise une fraction f du temps total de calcul, alors l'accélération est limitée par :

$$S_p(n) < \frac{1}{f + \frac{1-f}{p}} < \frac{1}{f}, \forall p \tag{A.2}$$

A.1.2 L'efficacité

Une autre mesure de performance d'un algorithme parallèle est l'efficacité (efficiency) définie par :

$$E_p(n) = \frac{T_1(n)}{p \times T_p(n)} \tag{A.3}$$

$T_1(n)$ représente le temps d'exécution de l'algorithme parallèle quand le nombre de processeurs p est égal à 1 (qui n'est pas nécessairement le même que le temps d'exécution séquentiel $T^*(n)$). Cette mesure donne une indication sur l'efficacité de l'utilisation des p processeurs dans l'algorithme parallèle. Une valeur de $E_p(n)$ approximativement égale à 1, pour un p donné, indique que l'algorithme A s'exécute p fois plus vite en utilisant p processeurs qu'il ne le fait avec 1 processeur. Chaque processeur effectue donc un « travail utile » durant chaque étape de l'algorithme (Jaja (1992)).

Cette mesure est souvent utilisée à des fins de comparaison avec les algorithmes séquentiels. Dans ce cas, $T_1(n)$ est remplacé par le temps d'exécution du meilleur algorithme séquentiel connu (ou possible). Plus grande est l'efficacité, meilleure est la solution parallèle (Bazewicz *et al.* (2000)).

Le nombre de processeurs est une mesure de performance étroitement liée à l'efficacité. De plus, pour des raisons économiques, le nombre de processeurs dont a besoin un algorithme parallèle est d'une importance considérable. Si deux algorithmes conçus pour résoudre un problème quelconque sur un modèle parallèle donné ont des temps d'exécution identiques, celui qui nécessite le moins de processeurs sera le moins coûteux à exécuter, donc sera le plus intéressant.

A.1.3 L'iso-efficacité

L'accélération et l'efficacité dépendent de la taille du problème et du nombre de processeurs. En fixant la taille du problème, il est possible d'étudier l'évolution de ces indicateurs en fonction du nombre de processeurs. L'inverse est également possible, c'està- dire qu'en fixant le nombre de processeurs, on peut étudier l'évolution de l'accélération et de l'efficacité en fonction de la taille du problème à résoudre. L'iso-efficacité représente la relation qu'il doit y avoir entre le nombre de processeurs p et la taille du problème n pour qu'une certaine efficacité donnée soit atteinte. Cette relation peut être utile pour déterminer le nombre idéal de processeurs pour une instance de problème donnée ou pour estimer la taille minimale que devrait avoir un problème pour être efficace avec un nombre déterminé de processeurs (Gengler *et al.* (1996)).

La performance des algorithmes parallèles est influencée par un ensemble de facteurs qui sont dépendants les uns des autres. La section suivante traitera de certains de ces éléments.

Annexe B

Sélection, croisement et mutation

B.1 Croisement et mutation uniformes

B.1.1 Croisement uniforme

Il peut être vu comme un croisement multi-points dont le nombre de points de coupure n'est pas connu a priori. En pratique on utilise un masque binaire de la même longueur que les génotypes. Si à la n^e position, il y a un 0, on conserve les symboles sinon on les échanges (cf. figure B.1). Le masque est généré aléatoirement pour chaque couple.

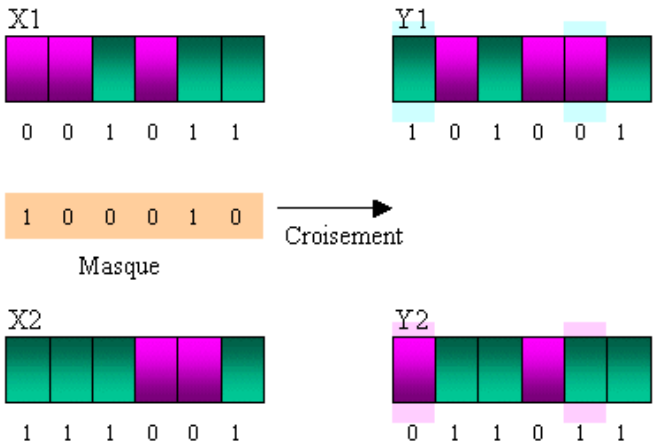


Fig. B.1 – Croisement uniforme (Ackley (1987))

B.1.2 Mutation uniforme

consiste à tirer *uniformément* la nouvelle valeur de la composante x_i de l'individu x dans un intervalle Min_i, Max_i, c'est un opérateur plus « brutal », mais qui peut être efficace malgré tout lorsqu'il convient de maintenir une une bonne diversité (en complément, avec d'autres opérateurs de mutation, par exemple).

B.2 La sélection

Dans un algorithme génétique, la *sélection* a pour but de choisir les individus qui participeront à la phase de reproduction. Les individus sont sélectionnés sur la base de leur *fonction d'adaptation*. Les meilleurs individus se reproduisent alors que les moins bons sont éliminés

. Cette fonction est aussi appelée *fonction objectif* ou *forme physique*[1]. Il existe plusieurs méthodes de sélection comme la *sélection proportionnelle* ou la *sélection par tournoi*.

B.2.1 Sélection proportionnelle

La sélection proportionnelle consiste à choisir les individus avec une probabilité proportionnelle à une fonction de performance f'_i calculée à partir de l'adaptation f_i de chaque individu i. Considérons une population de 10 individus et leurs adaptations f_i telles que :

Tab. B.1 – Résultat d'une fonction d'adaptation f_i

i	1	2	3	4	5	6	7	8	9	10
f_i	0.45	0.36	1.25	4.55	4.34	2.24	1.23	0.03	1.99	3.02

La figure B.2 représente sous la forme d'une roue l'ensemble des individus. Chaque individu étant représenté par un secteur proportionnel à son adaptation.

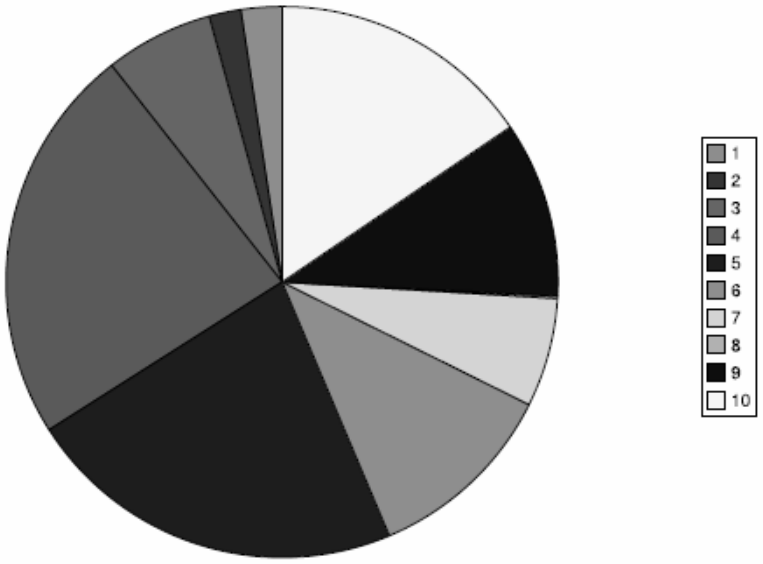


Fig. B.2 – Représentation des individus en fonction de leurs adaptations

Il existe, à partir de cette représentation, plusieurs méthodes de sélection. Les deux méthodes les plus répandues étant : La méthode de *la roulette* (RWS[2]) initiée par Holland (1992a) et *la méthode d'échantillonnage stochastique universel* (SUS[3]). La méthode RWS consiste à dérouler la roue sur un segment de longueur 1 et de tirer un nombre aléatoire dans le domaine $[0, 1]$ représentant une position sur ce segment (cf. : figure B.3). L'individu

[1] fitness en anglais
[2] Roulette wheel Selection
[3] Stochastic Universal Sampling

correspondant à cette position sera sélectionné. Ce processus de sélection doit être répété autant de fois que l'on souhaite d'individus.

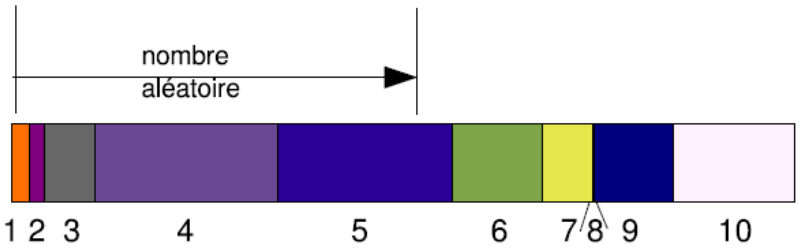


FIG. B.3 – Méthode RWS

L'inconvénient de cette méthode est la variance importante dans le processus de sélection. La méthode SUS garantit une variance minimale comme nous le montre J.DREO Dréo *et al.* (2003). De la même façon que dans la méthode précédente, on part d'un segment représentant les individus proportionnellement à leurs adaptations mais cette fois ci, un seul nombre aléatoire est défini pour sélectionner tous les individus. Ce nombre correspond à un offset appliqué à une série de points situés à égale distance sur le segment. Si l'on souhaite sélectionner n individus, on détermine un nombre aléatoire dans le domaine $[0, \frac{1}{n}]$ et on place, à partir de cet offset, n points équidistants de $\frac{1}{n}$. Chaque point sur le segment correspond à un individu sélectionné. La figure B.4 nous montre un exemple de ce type de sélection.

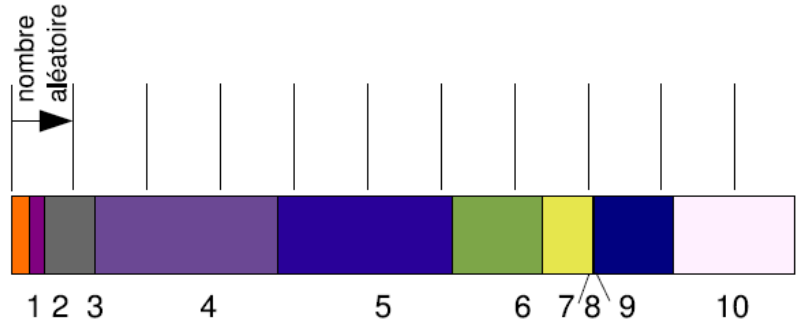


FIG. B.4 – Méthode SUS

Le tableau B.3 nous montre le nombre de sélections de chaque individu avec ma méthode SUS.

TAB. B.2 – Nombre de sélections de chaque individu

i	1	2	3	4	5	6	7	8	9	10
nb sélection	0	0	1	2	3	1	0	0	2	1

L'avantage de cette méthode est qu'un individu ind_j suffisamment adapté tel que : $f_j >$

$\frac{\sum_{i=1}^{m} f_i}{n}$, sera obligatoirement sélectionné[4].

De la même façon que la sélection d'individus proportionnellement à leurs fonctions d'adaptations f_i, il existe d'autres méthodes consistant à sélectionner les individus proportionnellement à leurs classements par rapport aux autres individus de la population. L'*ordonnancement* (ou la sélection selon le rang) consiste à classer les individus i suivant leurs *adaptations* f_i. Le rang 1 étant affecté au meilleur individu, on calcule ensuite $f'_i(r_i)$ où r_i est le rang de l'individu i dans le classement précédent et μ le nombre d'individus. Les individus sont ensuite sélectionnés selon une probabilité proportionnelle à f'_i. La formule suivante en est une représentation usuelle selon Dréo *et al.* (2003).

$$f'_i = \left(1 - \frac{r_i}{\mu}\right)^p \tag{B.1}$$

Les figures B.5 et B.6 montrent cette méthode appliquée à l'exemple précédent respectivement pour $p = 1$ et $p = 2$. Le tableau B.3 récapitule le nombre de sélections de chaque individu.

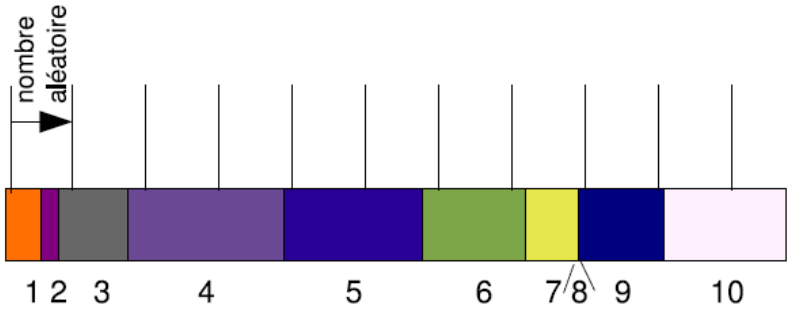


FIG. B.5 – Méthode SUS appliquée à la fonction f'_i avec $p = 1$

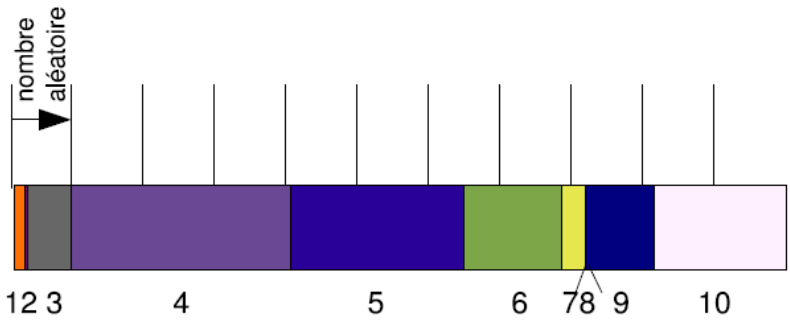


FIG. B.6 – Méthode SUS appliquée à la fonction f'_i avec $p = 2$

[4] m représente le nombre d'individus dans la population et n le nombre d'individus à sélectionner. Dans notre exemple on a $m = n$.

TAB. B.3 – Nombre de sélections de chaque individu (SUS avec ordonnancement

i	1	2	3	4	5	6	7	8	9	10
nb sélection avec $p = 1$	0	0	1	2	2	2	0	0	2	1
nb sélection avec $p = 2$	0	0	1	3	2	1	1	0	1	1

La connaissance de la fonction objectif n'est pas nécessaire pour cette méthode. Seul le classement des individus est indispensable. Elle est donc adaptée à des fonctions objectifs signées. Elle est adaptée à la maximisation ou la minimisation d'un problème sans profonde modification. Cependant, elle ne prend pas en compte la différence d'adaptation entre deux individus. Seul leur classement est pris en compte.

Il existe également d'autres méthodes d'ajustement de la fonction d'adaptation comme l'ajustement linéaire ou exponentiel. Ces méthodes sont décrites par Dréo *et al.* (2003)

B.2.2 Sélection par tournoi

Dans la méthode de sélection par tournoi, nous présélectionnons aléatoirement k individus indépendamment de leurs adaptations. Parmi ces k individus, le mieux adapté est sélectionné pour participer à la phase de reproduction. Ce processus est répété autant de fois que l'on souhaite d'individus.

Annexe C

Path-Scanning, Augment-Merge et l'Algorithme d'Ulusoy

C.1 Introduction

Quelques uns de ces algorithmes sont argumentés dans la thèse de Ramdane-Chérif (2002) mais organisés différemment ici.

C.2 Path-Scanning

C.2.1 Principe

Path-Scanning de Golden et Wong (1981) est décrite de façon littéraire par ses auteurs, sans vrai algorithme ni mention de la complexité. On construit les tournées une par une. La tournée courante θ part du dépôt σ et est prolongée à chaque itération vers une arête à traiter compatible avec la capacité résiduelle du véhicule. Pour une tournée parvenue en un nœud i, on donne la priorité aux arêtes non encore traitées partant de i. S'il n'y en a plus, on va au plus proche nœud ayant des arêtes non traitées. Les arêtes possibles $[i,j]$ sont départagées par un critère F. Les auteurs en proposent 5, expliques plus loin. L'algorithme est exécuté 5 fois (une fois par critère) et on sort à la fin la meilleure solution.

C.2.2 Algorithme

Pour un critère F donné, nous donnons une description concise de Path-Scanning dans l'algorithme 12. Les variables propres à l'algorithme sont la solution S, le nombre d'arcs traités *ndone* et *rmin(i)*, la quantité minimale sur les arcs à traiter partant de i. Par convention, $rmin(i) = \infty$ si i n'a pas d'arcs à traiter.

Rappelons qu'une solution S est une liste de tournées, qui sont elles-mêmes des listes d'arcs requis reliés implicitement par des plus courts chemins. S cache aussi des indicateurs $done(u)$ indiquant les arcs déjà traités, un coût $cost(S)$ et une charge $load(S)$. Coût et charge sont automatiquement mis à jour quand on ajoute ou on enlève des arcs.

L'heuristique commence par un $clear(S)$ qui prepare une solution vide S et initialise le statut des arcs à « non traite » ($done(u) = false$). Elle calcule les $rmin(i)$ et met *ndone* à 0. Puis, chaque itération du *repeat* construit une tournée courante θ qui est concaténée à S. Partant d'une tournée vide θ au nœud-dépôt, chaque itération du *loop* prolonge θ d'un arc.

On calcule d'abord le nœud j le plus proche de i ayant encore des arcs à traiter compatibles avec la charge du camion (si $rmin(j)$ ne rentre pas dans le camion, aucun arc partant de j ne convient). Notez que l'algorithme trouve $j = i$ si i a encore des arcs à traiter, puisque dans

ce cas $dist(i,j) = 0$. Si j n'est pas trouvé, la tournée est terminée. Sinon, on cherche un arc u non encore traité partant de j, compatible avec la charge du camion, et minimisant le critère F. Cet arc existe d'après notre définition des $rmin$. On l'ajoute en fin de tournée et on note qu'il est traité, ainsi que son inverse. On met à jour $rmin(i)$ car on peut repasser plus tard en i. Le nœud courant devient l'extrémité de u, $e(u)$ avec nos notations.

Quand la tournée courante θ est finie, on la concatène à la solution avec la procedure en queue et on compte dans *ndone* son nombre de tâches. L'algorithme se termine quand toutes les τ tâches sont traitées (Ramdane-Chérif (2002)).

Algorithme 12 Heuristique Path-Scanning

clear(S)
pour *touti* $\in V$ **faire**
  $rmin(i) := min\{r(u), u \in \delta_R^+(i)\}$
ndone :=0
répéter
  $\theta := \emptyset$
  $i := \sigma$
  boucler
    $j := argmin\{dist(i,k) : k \in Vandrmin(k) \leq (Q - load(\theta))\}$
    sortie si j=0
    $u := argmin\{F(v) : v \in \delta_R^+(j)anddone(v) = false\}$
    $done(u), done(inv(u)) := true$
    $enqueue(\theta, u)$
    $rmin(j) := min\{r(v) : v \in \delta_R^+(j)anddone(v) = false\}$
    $i := e(u)$
  $enqueue(S, \theta)$
  $ndone := ndone + |\theta|$
jusqu'à ndone=τ

Cet algorithme doit être exécuté en instanciant F par les fonctions simples suivantes :
$F_1(v) = -d(e(v), \sigma)$, maximisation de la distance au dépôt,
$F_2(v) = d(e(v), \sigma)$, minimisation de la distance au dépôt,
$F_3(v) = -r(v)/w(v)$, maximisation du rendement (quantité / coût de traitement),
$F_4(v) = r(v)/w(v)$, minimisation du rendement,
$F_5(v) =$ **si** $load(\theta) \leq Q/2$ **alors** $F_5(v) := F_1(v)$ **sinon** $F_5(v) := F_2(v)$ **finsi**

C.2.3 Complexité

La fonction *clear* qui met *done* à *false* coûte *O(n)*. Le calcul de tous les *rmin* est possible en $O(\tau)$. La boucle *boucler* range les tâches dans les tournées et fait donc en tout $O(\tau)$ itérations. Les calculs de j est en $O(n)$. Celui de u et la mise à jour des $rmin(i)$ sont aussi en $O(n)$ car pour tout nœud j on a $|\delta_R^+(j) \leq n|$. Le corps du « boucler » est donc en $O(n)$, d'où un algorithme en $O(\tau.n)$. Ces complexités deviennent respectivement $O(m.n)$ si tous les arcs sont requis, $O(n^2)$ si G est de type routier et $O(n^3)$ si G est complet.

C.3 Augment-Merge

C.3.1 Principe

Cette heuristique ressemble à la fameuse méthode de la marguerite proposée par Clarke et Wright (1964) pour le VRP. Il s'agit encore d'un algorithme de Golden et Wong, pour lequel les auteurs donnent une description laborieuse et sans complexité. Voici son principe pour le CARP de base, non orienté (voir aussi la figure C.1). Elle part d'une solution triviale de τ tournées réduites chacune à une tâche.

Dans la phase dite *Augment*, on cherche à éliminer les petites tournées dont l'unique tâche est traversée par une tournée plus grande : si la capacité résiduelle du camion le permet, la grande tournée peut collecter cette tâche en passant, absorbant ainsi la petite tournée. Pour de meilleurs résultats, Golden et Wong préconisent de trier les tournées initiales par coûts décroissants puis, pour toute tournée S_i dans l'ordre du tri, de tenter d'absorber $S_{i+1}, S_{i+2}, ..., S_\tau$, si S_i passe sur l'unique tâche-arête de ces tournées et si $load(S_i)$ le permet.

Ces absorptions ne changent pas le coût de S_i puisque les coûts de traitement et les coûts de traversée sont égaux. Notons que pour une tournée en cours d'augmentation S_i, les tournées S_1 à S_i peuvent avoir plusieurs tâches, tandis que celles à partir de S_{i+1} n'en ont toujours qu'une.

C.3.2 Algorithme

Nous proposons une version conceptuelle de l'heuristique dans l'algorithme 13. Nous rappelons qu'avec nos notations de listes, $S(i, j)$ désigne la tâche de rang j dans la tournée i de la solution S. Pour écrire simplement les coûts des fusions, on utilise un distancier arc à arc. Les tournées gardent leurs rangs pendant l'algorithme : quand une tournée $S(k)$ est supprimée dans la solution, on la met donc à vide plutôt que de faire un *remove* qui décrémenterait le rang des tournées suivantes. Cette façon de faire est importante pour une bonne complexité. Le 1^{e}bloc (initialisation) construit la marguerite avec une tournée par arête puis trie les tournées par coûts décroissants. Notons que c'est la liste-tournée (u) et non pas l'arc u qui est ajouté à S. Le 2^{e}bloc (phase *Augment*) considère chaque tournée $S(i)$ en partant de la plus grande et tente d'absorber les tournées suivantes.

Chaque itération du 3^{e} bloc (*Merge*) considère les couples (i, j) de tournées distinctes et évalue dans v les variations de coût des fusions possibles. On peut concaténer $S(i)$ puis $S(j)$ en inversant ou pas chaque tournée, d'où 4 cas. Il suffit de traiter les couple (i, j) avec $i < j$ car les coûts d'une tournée et de son inverse sont égaux : ainsi, concaténer $S(i)$ et $S(j)$ équivaut à concaténer l'inverse de $S(j)$ et l'inverse de $S(i)$. Les calculs de variations sont plus lisibles en introduisant les arcs b_i et e_i (resp. b_j et e_j) débutant et terminant la tournée i (resp. j).

La meilleure fusion de variation négative est stockée avec $vmin$ et le couple k_{ind} des indices des deux tournées, multipliés par -1 quand elles doivent être inversées : par exemple, $k_{ind} = (i, -j)$ signifie une fusion concaténant la tournée i puis l'inverse de la tournée j. Après évaluation de toutes les paires, chaque itération de $Merge$ effectue la meilleure fusion trouvée, en réutilisant l'indice i pour stocker la nouvelle tournée. L'algorithme se termine si aucune fusion n'est possible à cause des charges ou si les fusions possibles ne procurent plus de gains (Ramdane-Chérif (2002)).

Algorithme 13 Heuristique Augment-Merge

{Initialisation}
clear(S)
pour $u = 1$ à m **faire**
  si $done(u) = false$ **alors**
    $enqueue(S, (u))$
    $done(u), done(inv(u)) := true$
trier S en ordre décroissant des coûts des tournées
{Phase Augment}
pour $i = 1$ à $|S|$ **faire**
  si $S(i) \neq \emptyset$ **alors**
    pour $j := i + 1$ à $|S|$ **faire**
      si $S(j) \neq \emptyset \wedge load(S(i)) + load(S(j)) \leq Q$ **alors**
        si $S(j, 1)$ est insérable à coût nul dans $S(i)$ à un certain rang k **alors**
          $insert(S(i), k, S(j, 1))$
          $S(j) := \emptyset$
{Phase Merge}
boucler
  $vmin := 0$
  pour $i = 1$ à $|S|$ **faire**
    si $S(i) \neq \emptyset$ **alors**
      pour $j := i + 1$ à $|S|$ **faire**
        si $S(j) \neq \emptyset \wedge load(S(i)) + load(S(j)) \leq Q$ **alors**
          $b_i := S(i, 1); e_i := last(S(i))$
          $b_j := S(j, 1); e_j := last(S(j))$
          $var := d(ei, b_j) - d(e_i, s) - d(s, b_j)$
          si $v < vmin$ **alors**
            $vmin := v; k_{ind} := (+i, +j)$
          $var := d(e_i, inv(e_j)) - d(e_i, s) - d(e_j, s)$
          si $v < vmin$ **alors**
            $vmin := v; k_{ind} := (+i, -j)$
          $var := d(inv(b_i), b_j) - d(s, b_i) - d(s, b_j)$
          si $v < vmin$ **alors**
            $vmin := v; k_{ind} := (-i, +j)$
          $var := d(inv(b_i), inv(e_j)) - d(s, b_i) - d(e_j, s)$
          si $v < vmin$ **alors**
            $vmin := v; k_{ind} := (-i, -j)$
  Sortie si $vmin = 0$
  $i := abs(k_{ind}(1)); j := abs(k_{ind}(2))$
  si $k_{ind}(1) < 0$ **alors**
    $S(i) := inverse(S(i))$
  si $k_{ind}(2) < 0$ **alors**
    $S(j) := inverse(S(j))$
  $S(i) := S(i) + S(j)$
  $S(j) := \emptyset$

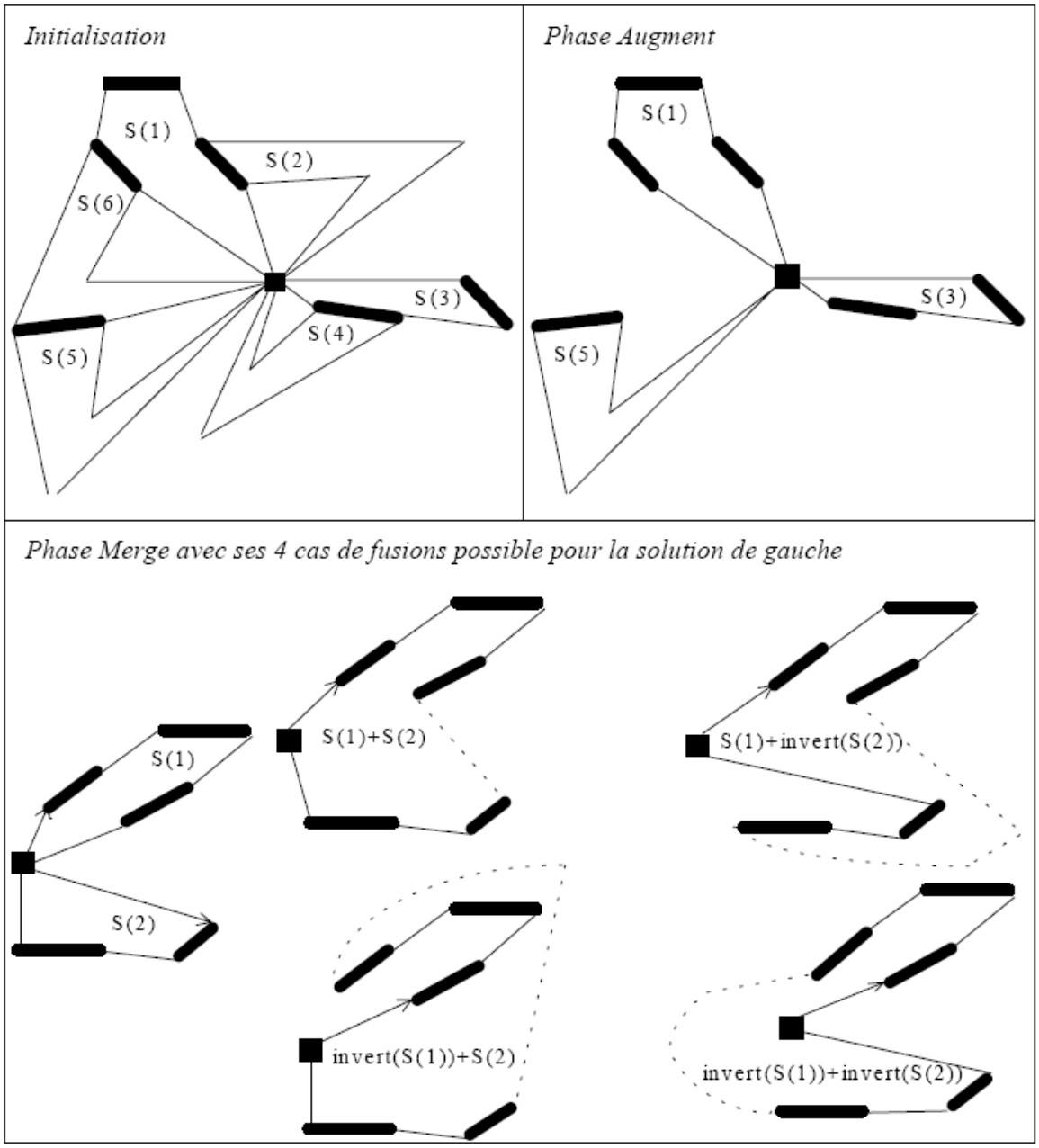


FIG. C.1 – Heuristique Augment-Merge

C.3.3 Complexité

Golden *et al.* (1983) ne mentionnent pas la complexité. Pearn (1988) pretend sans aucune justification qu'elle est en $O(n^3)$. En fait, seuls τ arcs parmi m (un par tâche) forment les tournées initiales. L'initialisation (tri) peut se faire en $O(\tau.log\tau)$. Dans *Augment*, pour $S(i)$ et $S(j)$ fixées, la tournée $S(i)$ peut vite contenir plusieurs tâches et il faut tester $|S(i)|+1 = O(\tau)$ positions d'insertion. La phase *Augment* balaie $O(\tau^2)$ paires de tournées et coûte donc $O(\tau^3)$.

La phase *Merge* effectue au pire $\tau - 1$ fusions et calcule pour chacune les variations de coût pour $O(\tau^2)$ paires de tournées, elle est aussi en $O(\tau^3)$. La complexité totale est donc $O(\tau^3)$, c'est-a-dire $O(m^3)$ si tous les arcs sont requis, $O(n^6)$ si G est complet et $O(n^3)$ si G est routier (Ramdane-Chérif (2002)).

C.4 Algorithme d'Ulusoy

C.4.1 Principe

Comme Golden *et al.* (1983), Ulusoy donne une description littéraire et sans complexité de son algorithme, avec une évaluation numérique réduite à quelques exemples. Nous proposons donc notre propre analyse de cette heuristique qui, nous le verrons, est très efficace. L'heuristique d'*Ulusoy* nécessite une séquence θ contenant les τ tâches, qui peut être vue comme un « tour complet » effectué par un véhicule de capacité infinie. Ce tour complet est ensuite découpé optimalement en tournées réalisables pour le CARP.

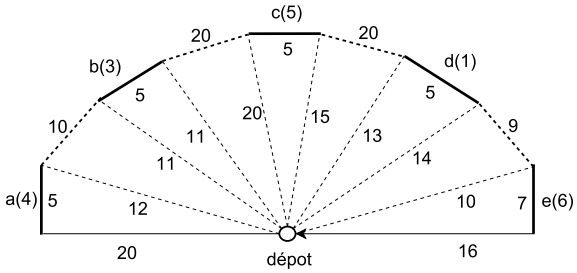
Si toutes les arêtes sont à traiter, le calcul du tour complet revient à résoudre un problème de postier chinois non orienté (UCPP, cf. chapitre 1), polynomial. Sinon, on a à faire à un problème de postier rural non orienté (URPP), NP-difficile. Comme une solution optimale du URPP ne donne pas nécessairement une solution optimale du CARP, une résolution avec n'importe quelle heuristique pour le URPP suffit. Nous appelons en fait *Path-Scanning* avec une capacité infinie et ses 5 critères pour construire 5 séquences différentes. Nous découpons ensuite ces 5 séquences avec la méthode d'*Ulusoy* et retournons la meilleure solution. Cette technique donne de très bons résultats.

La méthode d'Ulusoy découpe optimalement θ en calculant un plus court chemin dans un graphe auxiliaire value $H = (X, U, Z)$. X comprend $\tau + 1$ nœuds indexés de 0 à τ. Le nœud 0 est un nœud de départ. Tout arc (i,j) de U représente une tournée réalisable traitant les tâches $\theta(i+1)$ à $\theta(j)$ incluses, et la valeur $z(i, j)$ sur l'arc est le coût d'une telle tournée. Un plus court chemin entre les nœuds 0 et τ dans H fournit un découpage de θ en tournées réalisables minimisant le coût total. On peut détailler les tournées en remontant le chemin et en remplaçant chaque arc par la sous-sequence correspondante de θ.

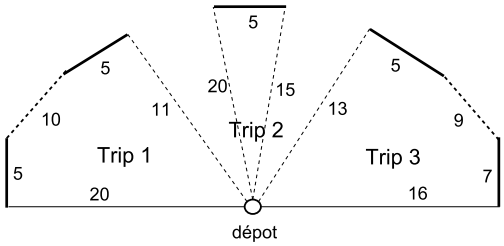
La figure 3.2 donne un exemple avec un tour complet initial θ de 5 tâches et des quantités entre parenthèses, en supposant une capacité $Q = 9$. Le plus court chemin dans le graphe auxiliaire est calcule avec l'algorithme de Bellman. Il donne un découpage optimal (sous contrainte de la sequence imposée par θ) en trois tournées et un coût total de 141.

C.4.2 Algorithme

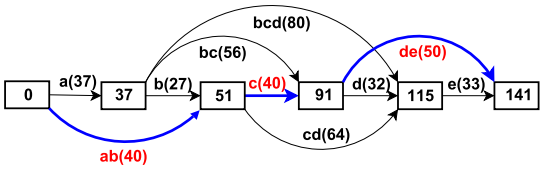
Notre algorithme 3.4 donne une procédure *Split* qui évite une génération explicite du graphe auxiliaire H et minimise en plus le nombre de véhicules en critère secondaire. Trois labels sont utilisés pour chaque nœud j de H : V_j, le coût du plus court chemin de 0 à j dans H, N_j, le nombre d'arcs de ce chemin (nombre de tournées ou de véhicules) et B_j, le prédécesseur de j sur ce chemin. Pour un tour complet θ, l'algorithme énumère toutes les tournées faisables $(\theta_i, ..., \theta_j)$ et calcule leurs charges et leurs coûts comme expliqué en 2.5.4. Au lieu de créer un arc $(i-1, j)$ pour chaque tournée comme dans 3.3.3.1, les labels de j



Etape 1: Tour complet avec 5 tâches.



Etape 3:Tournées résultantes



Etape 2: Plus court chemain dans le graphe auxiliaire pour W=9

FIG. C.2 – Principe de l'heuristique d'Ulusoy.

sont directement mis à jour. À la fin, le coût total et le nombre de véhicules utilisés nvu sont définis par V_τ et N_τ (Ramdane-Chérif (2002)) .

Algorithme 14 Procédure de découpage Split pour l'heuristique d'Ulusoy.

$V(0), N(0) := 0$
pour $i = 1$ à τ **faire**
  $V(i) := \infty$
pour $i = 1$ à τ **faire**
  $load, cost := 0; j := i$
  répéter
    $load := load + r(\theta(j))$
    si $i = j$ **alors**
      $cost := D(s, \theta(i)) + w(\theta(i)) + D(\theta(i), s)$
    sinon
      $cost := cost - D(\theta(j-1), s) + D(\theta(j-1), \theta(j)) + w(\theta(j)) + D(\theta(j), s)$
    si $load \leq Q$ **alors**
      $VNew := V(i-1) + cost$
      si $(VNew < V(j))or((VNew = V(j))and(N(i-1)+1 < N(j))$ **alors**
        $V(j) := VNew$
        $N(j) := N(i-1) + 1$
        $B(j) := i - 1$
      $j := j + 1$
  jusqu'à $(j > \tau) \vee (load > Q)$

Une solution S pour le CARP peut être obtenue avec l'algorithme 14 qui remonte les arcs du plus court chemin grace aux labels B_j. La tournée ξ associée à l'arc courant est

extraite de θ puis insérée $entetedeS$ pour respecter l'ordre du tour complet θ (procédure $push$). Remarquons la grande compacité du processus grâce à nos notations de listes.

Algorithme 15 Heuristique d'Ulusoy : extraction de la solution S.

1: clear(S)
2: $j := \tau$
3: **répéter**
4: $\quad i := B(j)$
5: $\quad \xi := copy(\theta, i+1, j)$
6: $\quad j := i$
7: $\quad push(S, \xi)$
8: **jusqu'à** $i = 0$

C.4.3 Complexité

La complexité du calcul du tour complet dépend de l'heuristique utilisée. Nous analysons donc seulement la phase de découpage. Pour un tour complet θ de τ tâches, le graphe H a $O(\tau^2)$ arcs. L'algorithme de $Bellman$ a pour complexité le nombre d'arcs du graphe, il est donc aussi en $O(\tau^2)$. L'extraction de la solution coûte $O(\tau)$. L'algorithme global est donc en $O(\tau^2)$, c'est-à-dire $O(m^2)$ si tous les arcs sont requis, $O(n^4)$ si G est complet, et $O(n^2)$ si G est de type routier (Ramdane-Chérif (2002)).

Annexe D

Étude de la charge du DF

D.1 Introduction

L'agent Directory Facilitator (D.F.) est un agent optionnel de la plateforme JADE qui remplit un service de pages jaunes. Chaque agent peut s'y inscrire, en fournissant à ce premier son nom, son adresse, le type de service qu'il offre, le langage de communication et l'ontologie. L'agent est libre de s'y désengager à tout moment.

Dans notre plateforme, où l'échange d'information, en particulier d'individus entre les agents mobile de calculs entres-eux et les agents de traitement des résultats, est primordial dans la résolution des problèmes, le DF devient une ressource critique, dans la mesure où :

- Il est soumis d'une part à un grand nombre de requêtes d'interrogation et d'enregistrement de service ;
- Et sans lui, les échanges d'information entre les agents mobiles et locaux seraient impossibles.

C'est pourquoi il nous semble stratégique et extrêmement important d'étudier les temps de réponses en fonction de certains paramètres essentiels de notre plateforme, et pouvant à tout instant varier en fonction du choix de l'algorithme ou du problème, ou encore du nombre de machines constituant le réseau de résolution du problème. Ces paramètres sont :

- Le nombre d'entrées dans les pages jaunes ;
- La diversité des entrées dans les pages jaunes ;
- La fréquence d'interrogation d'un agent auprès du DF ;
- Le nombre d'interrogateurs du DF.

D.2 Étude de l'influence des entrées du DF sur le temps de réponse

D.2.1 Protocole de test

Nous considérons deux types d'agents dans ce test. Un premier qui s'enregistre auprès du DF ; aucun comportement ne lui est associé et un second dont le seul comportement est de créer des agents d'enregistrement de service et d'interroger ensuite le DF.

Pour mesurer le temps de réponse nous avons utilisé deux classes de l'API fournies par JAVA : la classe Timer qui exécute de manière périodique de la méthode implémentée *run* de la classe TimerTask. Ce timer est lancé avant l'exécution de la demande de recherche auprès du DF et arrêtée après.

La plateforme MAF-DISTA s'exécutera typiquement sur un réseau de 30 machines. Dans notre application de cette plateforme, quatre types de service seront présents dans le service

des pages jaunes du DF, portant ainsi le nombre d'entrées à 120 ; afin de disposer d'une plus grande marche de manœuvre, nous avons choisi d'exécuter le test sur 200 entrées identiques dans un premiers temps. Nous étudierons l'influence de la diversité des entrées dans la section suivante.

D.2.2 Résultats

Le temps de réponse du DF devrait augmenter avec le nombre croissant d'entrées dans l'agent. Le graphe de la figure D.1 consigne la moyenne des mesures du temps de réponse du DF en fonction du nombre d'entrée.

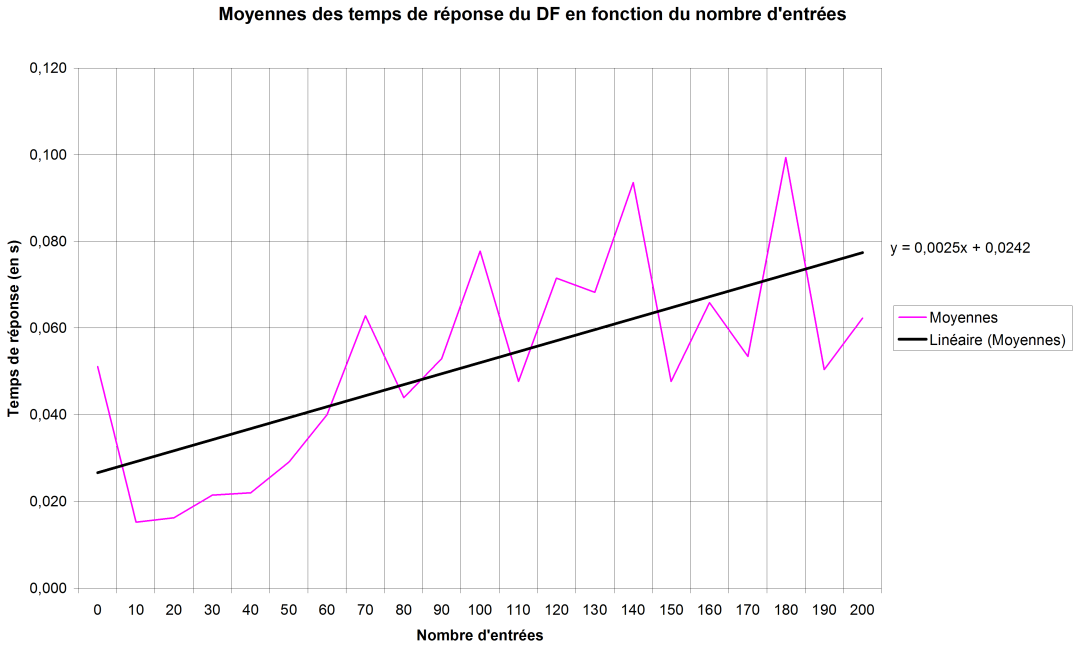


Fig. D.1 – Temps de réponse du DF

Sur ce graphique (cf. figure D.1), nous constatons que le temps de réponse du DF augmente en moyenne linéairement, ce qui affirme notre hypothèse de départ. Le temps de réponse est en moyenne inférieur à une seconde pour un nombre d'entrée inférieur à 400 entrées, ce qui nous laisse une marche de manœuvre considérable.

D.3 Étude de l'influence de la diversité des entrées du DF sur le temps de réponse

D.3.1 Protocole de test

Comme dans la campagne de mesure précédente, nous utilisons les mêmes principes et les mêmes agents. La différence ici ce fait sur la diversité des types de services enregistrés au niveau du DF.

Comme précisé précédemment, nous avons dans notre application un nombre d'entrées de service borné à 200 et au maximum quatre type de service.

D.3.2 Résultats

Le temps de réponse du DF devrait augmenter avec la diversité croissante des entrées dans l'agent. Le graphe de la figure D.2 consigne la moyenne des mesures du temps de réponse du DF en fonction du nombre d'entrées et de sa diversité.

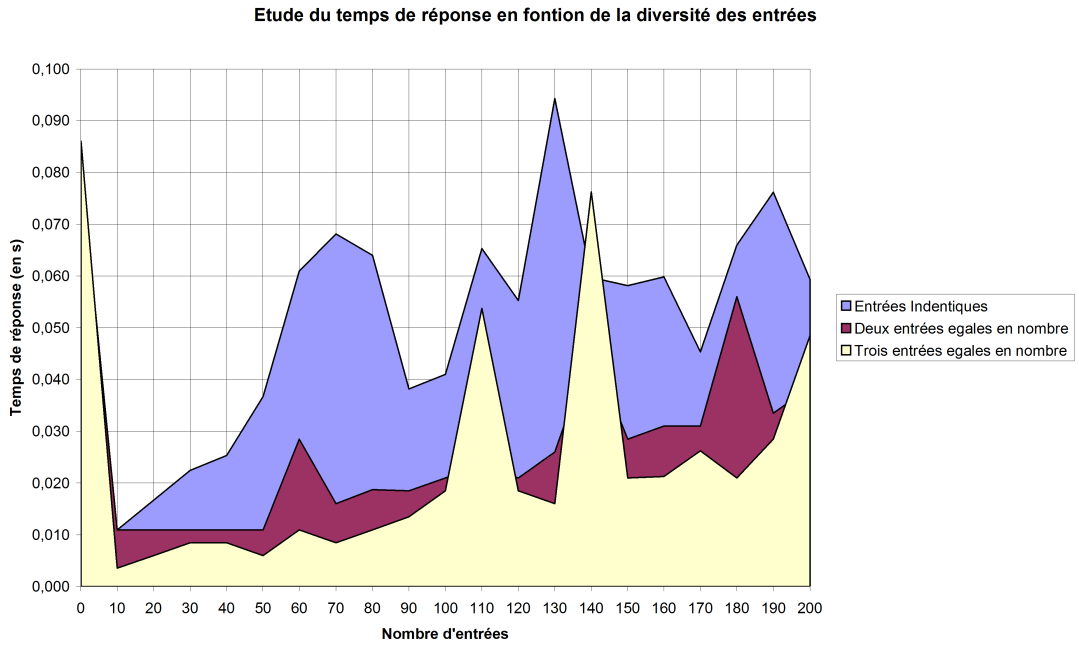


FIG. D.2 – Temps de réponse du DF / diversité d'entrées

Au vu des résultats obtenus, nous nous limiterons à étudier l'influence de cette diversité sur trois types différents. En effet, nous observons que le temps de réponse par nombre d'entrées diminue quand la diversité de celles-ci augmente. Ceci est dû au fait que le temps de recherche est constant pour un nombre d'entrées données ; le temps de réponse est alors conditionné par la constitution du message ACL de réponse.

D.4 Étude de l'influence de la fréquence des demandes sur le temps de réponse

D.4.1 Protocole de test

Comme dans la première campagne de mesure, nous utilisons les mêmes principes et les mêmes agents. La différence ici ce fait sur la fréquence d'interrogation du DF ; on fait varier la période d'interrogation de l'agent auprès du DF.

D.4.2 Résultats

Le temps de réponse du DF devrait être le même quelque ce soit la période d'interrogation. Le graphe de la figure D.3 consigne la moyenne des mesures du temps de réponse du DF en fonction du nombre d'entrées et de la fréquence d'interrogation.

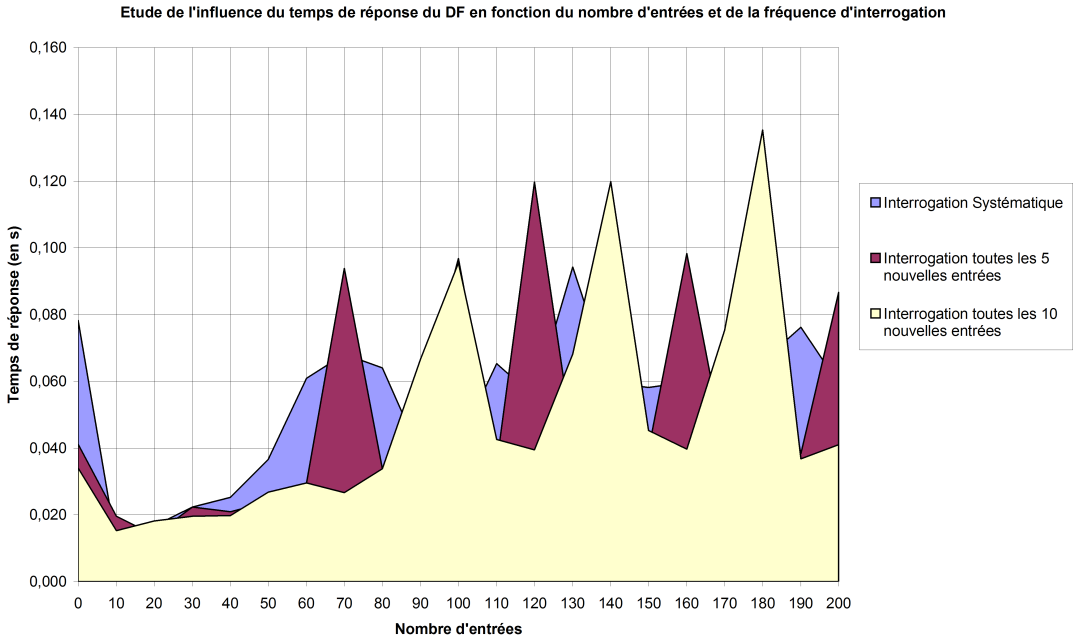


FIG. D.3 – Influence de la fréquence des demandes sur le temps de réponse

Nous constatons que les temps de réponse du DF fluctuent grandement. Néanmoins, nous pouvons observer que les temps de réponse en fonction du nombre d'entrées restent quasiment inchangées quelque soit la période d'interrogation.

D.5 Étude de l'influence du nombre d'interrogateurs sur le temps de réponse

D.5.1 Protocole de test

Comme dans la première campagne de mesure, nous utilisons les mêmes principes et les mêmes agents. La différence ici ce fait sur le nombre d'agent qui interrogent le DF. Nous fixons ici le nombre d'entrées à 100, toutes identiques. Les agents qui interrogent le DF le font toutes les secondes et l'on augmente progressivement leur nombre.

D.5.2 Résultats

Le temps de réponse du DF devrait augmenter avec le nombre croissant des agents qui interrogent le DF. Le graphe de la figure D.4 consigne la moyenne des mesures du temps de

réponse du DF en fonction du nombre d'entrées et de sa diversité.

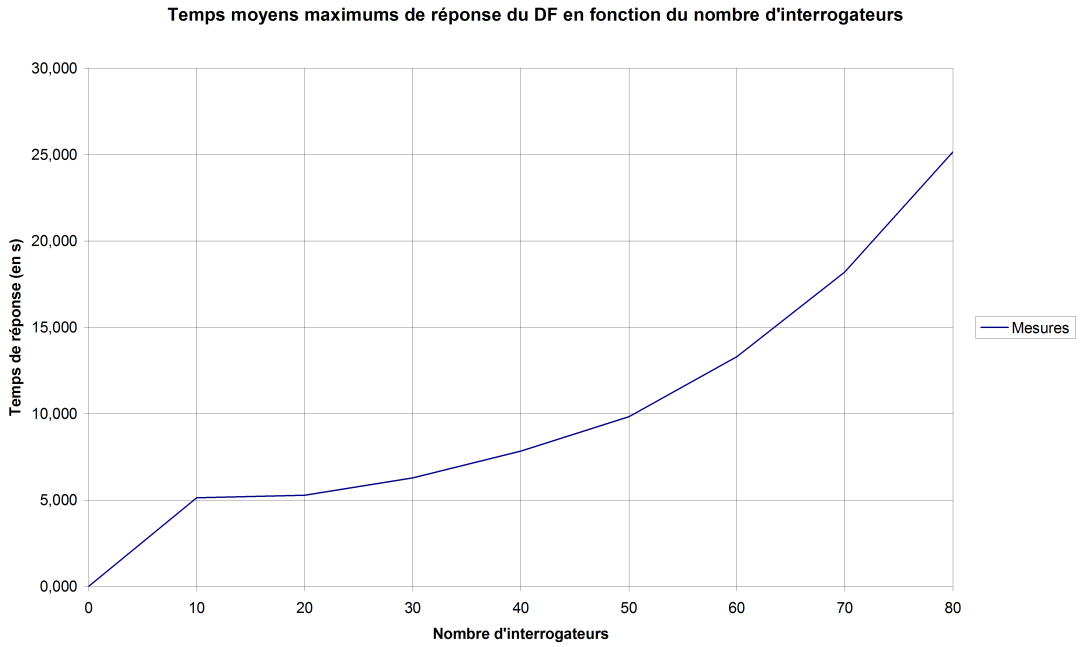


FIG. D.4 – Influence du nombre d'interrogateurs sur le temps de réponse

Nous observons que le temps de réponse augmente au fur et à mesure que le nombre des agents qui l'interrogent augmente, ce qui valide notre hypothèse de départ.

Bibliographie

Ackley, D. H. (1987). *A connectionist machine for genetic hillclimbing*. Kluwer Academic Publishers, Norwell, MA, USA. 179

Adamidis, P. et Petridis, V. (1996). Co-operating populations with different evolution behaviours. *Dans International Conference on Evolutionary Computation*, pages 188–191. 32

Adamidis, P. et Petridis, V. (2002). On modelling evolutionary algorithm implementations through co-operating populations. *Dans PPSN VII : Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, pages 321–330, London, UK. Springer-Verlag. 29, 31

Alba, E., Luque, G. et Luna, F. (2007). Parallel metaheuristics for workforce planning. *Journal of Mathematical Modelling and Algorithms*. 2, 34

Alba, E. et MALLBA-Group (2007). Mallba : a software library to design efficient optimisation algorithms. *Int. J. Innov. Comput. Appl.*, 1(1):74–85. 29

Alba, E. et Tomassini, M. (2002). Parallelism and evolutionary algorithms. *Evolutionary Computation, IEEE Transactions on*, 6(5):443–462. 26

Alouf, S., Huet, F. et Nain, P. (2002). Forwarders vs. centralized server : an evaluation of two approaches for locating mobile agents. *Perform. Eval.*, 49(1-4):299–319. 46

Amdahl, G. M. (2000). Validity of the single processor approach to achieving large scale computing capabilities. pages 79–81. 177

Ametller, J., Robles, S. et Ortega-Ruiz, J. A. (2004). Self-protected mobile agents. *Dans AAMAS '04 : Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems*, pages 362–367, Washington, DC, USA. IEEE Computer Society. 49

Arcangeli, J.-P., Hameurlain, A., Bernard, G. et Monin, J.-F. (2002). Agents et code mobiles. *Revue des sciences et technologies de l'information, série Technique et science informatiques (RSTI-TSI), Hermès Science Publications*, 21(6). 39, 173

Arenas, M. G., Collet, P. et al. (2002). A framework for distributed evolutionary algorithms. *Dans PPSN VII : Proceedings of the 7th International Conference on Parallel Problem Solving from Nature*, pages 665–675, London, UK. Springer-Verlag. 29, 31

Bachelet, V. et Talbi, E.-G. (2000). Cosearch : a co-evolutionary metaheuristic. *Dans Proc. of the 2000 Congress on Evolutionary Computation*, pages 1550–1557, Piscataway, NJ. IEEE Service Center. 52, 53

Baker, B. M. et Ayechew, M. A. (2003). A genetic algorithm for the vehicle routing problem. *Comput. Oper. Res.*, 30(5):787–800. 144, 146, 147

Baldi, M. et Picco, G. P. (1998). Evaluating the tradeoffs of mobile code design paradigms in network management applications. *Dans ICSE '98 : Proceedings of the 20th international conference on Software engineering*, pages 146–155, Washington, DC, USA. IEEE Computer Society. 131

Bäumer, C. et Magedanz, T. (1999). Grasshopper - a mobile agent platform for active telecommunication. *Dans IATA '99 : Proceedings of the Third International Workshop on Intelligent Agents for Telecommunication Applications*, pages 19–32, London, UK. Springer-Verlag. 46, 50

Bautista, J., Fernández, E. et Pereira, J. (2008). Solving an urban waste collection problem using ants heuristics. *Comput. Oper. Res.*, 35(9):3020–3033. 141, 142, 143

Bazewicz, J., Trystram, D., Ecker, K. et Plateau, B., éditeurs (2000). *Handbook on Parallel and Distributed Processing*. Springer-Verlag New York, Inc., Secaucus, NJ, USA. 178

Beale, R. et Wood, A. (1994). Agent-based interaction. *Dans HCI '94 : Proceedings of the conference on People and computers IX*, pages 239–245, New York, NY, USA. Cambridge University Press. 43

Belenguer, J. et Benavent, E. (May 1998). The capacitated arc routing problem : Valid inequalities and facets. *Computational Optimization and Applications*, 10:165–187(23). 139

Belenguer, J. M. et Benavent, E. (2003). A cutting plane algorithm for the capacitated arc routing problem. *Comput. Oper. Res.*, 30(5):705–728. 91, 94, 125, 126

Belenguer, J.-M., Benavent, E., Lacomme, P. et Prins, C. (2006). Lower and upper bounds for the mixed capacitated arc routing problem. *Comput. Oper. Res.*, 33(12):3363–3383. 139

Belkhelladi, K., Chauvet, P., Daya, B. et Schaal, A. (2009a). A mobile agent framework to support parallel computing - application to multi-product planning and scheduling problems. *Dans Developments in E-Systems Engineering*, pages 335–342. Abu Dhabi, UAE. 169, 174

Belkhelladi, K., Chauvet, P., Péridy, L. et Schaal, A. (2009b). Un système d'agents pour la planification et l'ordonnancement multi-produits, multi-modes avec approvisionnements. *Dans ROADEF*, pages 300–301. Nancy, France. 169, 174

Belkhelladi, K., Chauvet, P., Réveillère, F. et Schaal, A. (2007). An intelligent distributed genetic algorithm for the capacitated arc routing problem. *European Conference on Complex Systems (ECCS 2007), Dresden, Germany.* 2, 29, 59, 95, 174

Belkhelladi, K., Chauvet, P. et Schaal, A. (2008). An agent framework with an efficient information exchange model for distributed genetic algorithms. *Dans IEEE Congress on Evolutionary Computation*, pages 848–853. IEEE. 59, 95, 174

Belkhelladi, K., Chauvet, P. et Schaal, A. (2010a). *Cooperation Control in Distributed Population-based Algorithms using a Multi-agent Approach - Application to a real-life Vehicle Routing Problem.* Accepted to appear in the book "Robotics, Automation and Control". 29, 34, 132, 174

Belkhelladi, K., Chauvet, P. et Schaal, A. (2010b). Maf-dista : A mobile agent framework as a tool for implementing parallel population-based algorithms. *En révision, Multiagent and Grid Systems.* 152, 174

Bellavista, P., Corradi, A. et Stefanelli, C. (2001). How to monitor and control resource usage in mobile agent systems. *Dans DOA '01 : Proceedings of the Third International Symposium on Distributed Objects and Applications*, page 65, Washington, DC, USA. IEEE Computer Society. 48

Benachenhou, L. et Pierre, S. (2006). Protection of a mobile agent with a reference clone. *Computer Communications*, 29(2):268–278. 49

Beullens, P., Muyldermans, L., Cattrysse, D. et Oudheusden, D. V. (2003). A guided local search heuristic for the capacitated arc routing problem. *European Journal of Operational Research*, 147(3):629–643. 71

Birattari, M. (2005). *The Problem of Tuning Metaheuristics.* PhD thesis, Université Libre de Bruxelles. 1

Blum, C. et Roli, A. (2003). Metaheuristics in combinatorial optimization : Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308. 1

Bodin, L., Golden, B., Assad, A. et Ball, M. (1983). Routing and scheduling of vehicles and crews - the state of the art. volume 10, pages 63–212. Computers and Opertions Research. 142

Bodin, L. D. et Kursh, S. J. (1978). A Computer-Assisted System for the Routing and Scheduling of Street Sweepers. *OPERATIONS RESEARCH*, 26(4):525–537. 139

Bodin, L. D. et Kursh, S. J. (1979). A detailed description of a computer system for the routing and scheduling of street sweepers. *Computers & OR*, 6(4):181–198. 139

Boroujerdi, A. et Uhlmann, J. (1998). An efficient algorithm for computing least cost paths with turn constraints. *Inf. Process. Lett.*, 67(6):317–321. 141

Buse, D. P., Feng, J. Q. et Wu, Q. H. (2003). Mobile agents for data analysis in industrial automation systems. *Dans IAT '03 : Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, page 60, Washington, DC, USA. IEEE Computer Society. 131

Cahon, S. (2005). *ParadisEO : Une plate-forme pour la conception et le déploiement de métaheuristiques parallèles hybrides sur clusters et grilles.* Thèse de doctorat, Université de Lille, France. 25, 26, 27, 28, 29, 30, 31, 32, 33, 34

Cahon, S., Melab, N. et Talbi, E. G. (2004). Paradiseo : A framework for the reusable design of parallel and distributed metaheuristics. *Journal of Heuristics*, 10(3):357–380. 2, 29, 33

Calégari, P. R. (1999). *Parallelization of population-based evolutionary algorithms for combinatorial optimization problems*. Thèse de doctorat, Lausanne. 24, 61

Cantù-Paz, E. (2000). *Efficient and Accurate Parallel Genetic Algorithms*. Kluwer Academic Publishers, Norwell, MA, USA. 25, 61

Carvalho, M., Cowin, T., Suri, N., Breedy, M. et Ford, K. (2004). Using mobile agents as roaming security guards to test and improve security of hosts and networks. *Dans SAC '04 : Proceedings of the 2004 ACM symposium on Applied computing*, pages 87–93, New York, NY, USA. ACM. 49

Chess, D., Harrison, C. et Kershenbaum, A. (1996). Mobile agents : Are they a good idea ? *Dans* Vitek, J. et Tschudin, C., éditeurs : *Proceedings of the Second International Workshop on Mobile Object Systems*, volume 1222 de *Lecture Notes in Computer Science*, pages 25–47, Linz, Austria. Springer-Verlag. 48

Christofides, N. (1973). The optimum traversal of a graph. *Omega*, 1(6):719–732. 72

Claessens, J., Preneel, B. et Vandewalle, J. (2003). (how) can mobile agents do secure electronic transactions on untrusted hosts ? a survey of the security issues and the current solutions. *ACM Trans. Internet Technol.*, 3(1):28–48. 49

Clarke, G. et Wright, J. W. (1964). Scheduling of Vehicles from a Central Depot to a Number of Delivery Points. *OPERATIONS RESEARCH*, 12(4):568–581. 187

Clearwater, S. H., Hogg, T. et Huberman, B. A. (1992). Cooperative problem solving. *Dans Computation : The Micro and the Macro View*, pages 33–70. World Scientific. 23, 60

Corberán, A., Martí, R., Martínez, E. et Soler, D. (2002). The rural postman problem on mixed graphs with turn penalties. *Comput. Oper. Res.*, 29(7):887–903. 139, 142

Cormen, T., Leiserson, C., Rivest, R. et Stein, C., éditeurs (2002). *Introduction à l'algorithmique*. Dunod, France. 177

Costa, J., Lopes, N. et Silva, P. (2001). Jdeal. the java distributed evolutionary algorithms library. 29, 30

Crainic, T., Gendreau, M. et Potvin, J.-Y. (2005). *Parallel Tabu Search*. Parallel Metaheuristics, E. Alba (Ed.), John Wiley & Sons. 2

Crainic, T. et Toulouse, M. (2003). Parallel strategies for meta-heuristics. *State-of-the-Art Handbook in Metaheuristics, F. Glover, G. Kochenberger (Eds.), Kluwer Academic Publishers, Norwell, MA, Kluwer Academic Publishers, Norwell, MA*, pages 475–513. 2, 52

Crainic, T. G. et Toulouse, M. (1998). Parallel metaheuristics. *Dans Fleet Management and Logistics. , T. G. Crainic and G. Laporte. Norwell, MA.*, pages 205–251. Kluwer Academic. 15, 18, 19, 20, 21, 22

Croes, G. A. (1958). A Method for Solving Traveling-Salesman Problems. *OPERATIONS RESEARCH*, 6(6):791–812. 145

Cung, V.-D., Martins, S. L., Ribeiro, C. C. et Roucairol, C. (2002). Strategies for the parallel implementation of metaheuristics. *Dans Essays and Surveys in Metaheuristics*, pages 263–308. Kluwer Academic Publishers. 2, 15, 18, 19, 22

DeArmon, J. (1981). *A Comparison of heuristics for the capacitated Chinese Postman problem*. Thèse de doctorat, University of Maryland, College Park, MD. 91, 94, 125

Deb, K. et Agrawal, S. (1998). Understanding interactions among genetic algorithm parameters. *Dans in Foundations of Genetic Algorithms 5*, volume 5, pages 265–286. 66

Delisle, P. (2002). *Parallélisation d'un algorithme d'Optimisation par Colonies de Fourmis pour la résolution d'un problème d'ordonnancement industriel*. Thèse de doctorat, Université du Québec, Canada. 15, 18, 22, 23, 24, 25, 60, 61, 62, 177

Derman, C. (1970). *Finite State Markov Decision Processes*, volume 67 de *Mathematics in Science and Engineering*. Academic Press, Inc., 111 Fifth Avenue, New York, NY 10003, first édition. 16

Di Gaspero, L. et Schaerf, A. (2000). Easylocal++ : An object-oriented framework for flexible design of local search algorithms. 29

Díaz, J. A. P., Gutiérrez, D. Á. et Sobrado, I. (2001). A fast data protection technique for mobile agents against malicious hosts. *Electr. Notes Theor. Comput. Sci.*, 63. 49

Dijkstra, E. W. (1959). *Communication with an Automatic Computer*. Thèse de doctorat, University of Amsterdam. 75, 77

Dillenseger, B., Tagant, A.-M. et Hazard, L. (2002). Programming and executing telecommunication service logic with moorea reactive mobile agents. *Dans MATA '02 : Proceedings of the 4th International Workshop on Mobile Agents for Telecommunication Applications*, pages 48–57, London, UK. Springer-Verlag. 46

Dimitrijević, V. et Šarić, Z. (1997). An efficient transformation of the generalized traveling salesman problem into the traveling salesman problem on digraphs. *Inf. Sci.*, 102(1-4):105–110. 144

Dorigo, M. et Gambardella, L. M. (1997). Ant colony system : A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66. 59

Dréo, J. (2007). Différentes classifications des métaheuristiques. `http://fr.wikipedia.org/`. 17

Dréo, J., Pétrowski, A., Siarry, P. et Taillard, E. D. (2003). *Métaheuristiques pour l'optimisation difficile*. Eyrolles. 181, 182, 183

Dror, M. (2000). Arc routing : Theory, solutions and applications. kluwer academic publishers. 72, 139

Dror, M. et Langevin, A. (1997). A Generalized Traveling Salesman Problem Approach to the Directed Clustered Rural Postman Problem. *TRANSPORTATION SCIENCE*, 31(2): 187–192. 142

Durfee, E. H., Lesser, V. R. et Corkill, D. D. (1989). Trends in cooperative distributed problem solving. *IEEE Trans. on Knowl. and Data Eng.*, 1(1):63–83. 40

Eglese, R. W. (1994). Routing winter gritting vehicles. *Discrete Appl. Math.*, 48(3):231–244. 71, 125

Eglese, R. W. et Li, L. Y. O. (1996). A tabu search based heuristic for arc routing with a capacity constraint and time deadline. in : I. h. osman and j. p. kelly (ed.), metaheuristics : Theory and applications, kluwer, 633-650. 71

Eiben, A. E., kuyper, I. G. S. et Thijssen, B. A. (1998). Competing crossovers in an adaptive ga framework. *Dans In Proceedings of the 5th IEEE Conference on Evolutionary Computation*, pages 787–792. IEEE Press. 66, 67

Eksioglu, S. D., Pardalos, P. M. et Resende, M. G. C. (2001). Parallel metaheuristics for combinatorial optimization. advanced algorithmic techniques for parallel computation with applications. *R. Correa et al. (Eds.), Kluwer Academic Publishers.* 2

El-Falou, S. (2006). *Programmation répartie, optimisation par agent mobile.* Thèse de doctorat, Université de Caen, FRANCE. 3, 39, 41, 42, 43, 44, 45, 46, 47, 48, 50, 51, 52

Eldos, T. (2006). A new migration model for distributed genetic algorithms. *Dans CSC*, pages 128–134. 64, 68

Feo, T. et Resende, M. G. C. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, 8(2):67–71. 16

Ferber, J. (1995). *Les systèmes multi-agents, vers une intelligence collective.* InterEditions, Paris (France). 3, 39, 40, 41

Ferber, J. (1997). Les systèmes multi agents : un aperçu général. *TSI techniques et sciences Informatiques.* 3

Fink, A., Vo, S. et Woodruff, D. (1998). Building reusable software components for heuristic search. 25, 26, 27

Fisher, M. L. et Jaikumar, R. (1981). Generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124. cited By (since 1996) 162. 144

Fonlupt, C., Robillard, D. et Preux, P. (1997). A comparison of the 2-opt-move and the city-swap operators for the tsp. *Dans Evolution Artificielle.* 59

Fuggetta, A., Picco, G. P. et Vigna, G. (1998). Understanding code mobility. *IEEE Transactions on Software Engineering*, 24(5):342–361. 44

Gagne, C., Parizeau, M. et Dubreuil, M. (2003). Distributed beagle : An environment for parallel and distributed evolutionary computations. *Dans Proceedings of the 17 th Annual International Symposium on High Performance Computing Systems and Applications (HPCS*. 29, 32

Galtier, V., Mills, K., Carlinet, Y., Bush, S. et Kulkarni, A. (2001). Predicting and controlling resource usage in a heterogeneous active network. *Dans AMS '01 : Proceedings of the Third Annual International Workshop on Active Middleware Services*, page 35, Washington, DC, USA. IEEE Computer Society. 48

Gamma, E., Helm, R., Johnson, R. et Vlissides, J. (1995). *Design patterns : elements of reusable object-oriented software*. Addison-Wesley Professional. 26

Garey, M. S. et Johnson, D. S. (1979). *Computer and Intractability : A Guide to the Theory of NP-Completeness*. New York, W.H. Freeman and Co. 1

Gengler, M., Ubéda, S. et Desprez, F., éditeurs (1996). *Initiation au Parallélisme Concepts, Architectures et Algorithmes*. Manuels Informatiques MASSON, France. 178

Glover, F. (1989). Tabu search - part i. *ORSA Journal on Computing*, 1(3):190–206. 16

Glover, F. (1990). Tabu search - part ii. *ORSA Journal on Computing*, 2(1):4–32. 16

Golden, B., Dearmon, J. et Baker, E. (1983). Computational experiments with algorithms for a class of routing problems. *Computers and Operations Research*, 10(1):47–59. cited By (since 1996) 45. 91, 189, 190

Golden, B. L. et Wong, R. T. (1981). Capacitated arc routing problems. *Networks*, 11(3):305–315. 72, 91, 139, 185

Gomoluch, J. et Schroeder, M. (2001). Information agents on the move : A survey on load-balancing with mobile agents. 47

Gras, R., Hernandez, D., Hernandez, P., Zangge, N., Mescam, Y., Frey, J., Martin, O., Nicolas, J. et Appel, R. D. (2003). Cooperative metaheuristics for exploring proteomic data. *Artif. Intell. Rev.*, 20(1-2):95–120. 18, 20, 21

Grefenstette, J. (1986). Optimization of control parameters for genetic algorithms. *IEEE Trans. Syst. Man Cybern.*, 16(1):122–128. 67

Greistorfer, P. (2003). A tabu scatter search metaheuristic for the arc routing problem. *Comput. Ind. Eng.*, 44(2):249–266. 71

Gueguen, C. (1999). *Méthode de résolution exacte pour les problèmes de tournées de véhicules*. Thèse de doctorat, Paris. 142

Hantz, F. et Guyennet, H. (2006). A p2p platform using sandboxing. *Dans WSHPCS (Workshop on Security and High Performance Computing Systems) In conjunction with the 20th European Conference on Modelling and Simulation (ECMS 2006)*, pages 736–739, Bonn, Germany. 48

Herrera, F., Lozano, M. et Verdegay, J. L. (1998). Tackling real-coded genetic algorithms : Operators and tools for behavioural analysis. *Artif. Intell. Rev.*, 12(4):265–319. 164

Hertz, A. (2005). *Les Métaheuristiques*. ORANGE Lab. 16, 17

Hertz, A., Laporte, G. et Mittaz, M. (2000). A tabu search heuristic for the capacitated arc routing problem. *Oper. Res.*, 48(1):129–135. 71, 91, 94, 126

Hohlfeld, M., Ojha, A. et Yee, B. (2002). Security in the sanctuary system λ. 49

Holland, J. H. (1992a). *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA. 180

Holland, J. H. (1992b). *Adaptation in Natural and Artificial Systems : An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press. 16

Honavar, V., Miller, L. et Wong, J. (1998). Distributed knowledge networks. *Dans In : Proceedings of the IEEE Information Technology Conference*, pages 87–90. IEEE Press. 3

Huberman, B. A. (1990). The performance of cooperative processes. *Dans CNLS '89 : Proceedings of the ninth annual international conference of the Center for Nonlinear Studies on Self-organizing, Collective, and Cooperative Phenomena in Natural and Artificial Computing Networks on Emergent computation*, pages 38–47, Amsterdam, The Netherlands, The Netherlands. North-Holland Publishing Co. 22

Ip, W. H., Li, Y., Man, K. F. et Tang, K. S. (2000). Multi-product planning and scheduling using genetic algorithm approach. *Comput. Ind. Eng.*, 38(2):283–296. 161, 164, 166

Jain, R., Anjum, F. et Umar, A. (2000). A comparison of mobile agent and client-server paradigms for information retrieval tasks in virtual enterprises. *Dans AIWORC '00 : Proceedings of the Academia/Industry Working Conference on Research Challenges*, page 209, Washington, DC, USA. IEEE Computer Society. 45

Jaja, J. (1992). *An Introduction to Parallel Algorithms*. Addison-Wesley Professional. 177, 178

Jennings, N. R., Sycara, K. et Wooldridge, M. (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1(1):7–38. 40

Jha, R. et Iyer, S. (2001). Performance evaluation of mobile agents for e-commerce applications. *Dans HiPC '01 : Proceedings of the 8th International Conference on High Performance Computing*, pages 331–340, London, UK. Springer-Verlag. 44

Johnson, R. E. et Foote, B. (1988). Designing reusable classes. *Journal of Object-Oriented Programming*, 1(2):22–35. 26, 27

Juille, H. et Pollack, J. B. (1996). Dynamics of co-evolutionary learning. *Dans* Maes, P., Mataric, M. J., Meyer, J.-A., Pollack, J. et Wilson, S. W., éditeurs : *Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior : From animals to animats 4*, pages 526–534, Cape Code, USA. MIT Press. 25, 62

Keijzer, M., Merelo, J. J., Romero, G. et Schoenauer, M. (2001). Evolving objects : a general purpose evolutionary computation library. *Dans EA-01, Evolution Artificielle, 5th International Conference in Evolutionary Algorithms*, pages 231–244. 29, 30

Kim, J.-S. (2002). Distributed genetic algorithm with multiple populations using multi-agent. *Dans ISHPC '02 : Proceedings of the 4th International Symposium on High Performance Computing*, pages 329–334, London, UK. Springer-Verlag. 64, 68, 89

Kojima, K., Kawamata, W., Matsuo, H. et Ishigame, M. (2000). Network based parallel genetic algorithm using client-server model. *Dans Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, volume 1, pages 244 –250 vol.1. 72, 84, 85, 86, 96

Koza, J. R. (1991). Genetic evolution and co-evolution of computer programs. *Dans* Langton, C. T. C., Farmer, J. D. et Rasmussen, S., éditeurs : *Artificial Life II*, volume X de *SFI Studies in the Sciences of Complexity*, pages 603–629. Addison-Wesley, Santa Fe Institute, New Mexico, USA. 24, 61

Krasnogor, N. et Smith, J. (2000). Mafra : A java memetic algorithms framework. *Dans Proceedings of the 2000 International Genetic and Evolutionary Computation Conference (GECCO 2000)*, The Rivera Hotel and Casino, Las Vegas, Nevada, USA. 29, 30

Lacomme, P. et Prins, C. (2006). Algorithmes de découpage pour les problèmes de tournées. *Dans 6^{e} Conférence Francophone de MOdélisation et SIMulation - MOSIM'06*, Rabat, Maroc. 77

Lacomme, P., Prins, C. et Ramdane-Chérif, W. (2001). A genetic algorithm for the capacitated arc routing problem and its extensions. *Dans EvoWorkshops*, pages 473–483. 71, 74, 75, 80, 92, 94, 126, 139

Lacomme, P., Prins, C. et Ramdane-Chérif, W. (2004). Competitive memetic algorithms for arc routing problems. *Annals of Operations Research*, 131(5):159–185. 71, 126

Lange, D. B. et Oshima, M. (1999). Seven good reasons for mobile agents. *Commun. ACM*, 42(3):88–89. 107

Laporte, G. (1992). The vehicle routing problem : An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358. 142

Laporte, G. (1997). Modeling and solving several classes of arc routing problems as traveling salesman problems. *Comput. Oper. Res.*, 24(11):1057–1061. 142

Le Bouthillier, A. et Crainic, T. G. (2005). A cooperative parallel meta-heuristic for the vehicle routing problem with time windows. *Comput. Oper. Res.*, 32(7):1685–1708. 1

Lee, K., Yun, J. et Jo, G. (February 2003). Mocaas : auction agent system using a collaborative mobile agent in electronic commerce. *Expert Systems with Applications*, 24:183–187(5). 107

Lee, W.-P. (2007). Parallelizing evolutionary computation : A mobile agent-based approach. *Expert Systems with Applications*, 32(2):318 – 328. 59, 65

Li, Y., Ip, W. H. et Wang, D. W. (1998). Genetic algorithm approach to earliness and tardiness production scheduling and planning problem. *International Journal of Production Economics*, 54(1):65–76. 161

Lin, S. (1965). Computer solutions of the traveling salesman problem. *Bell Syst. Tech.*, 44:2245–2269. 145

Lis, J. (1996). Parallel genetic algorithm with dynamic control parameter. *Dans Proceedings of the 1996 IEEE Conference on Evolutionary Computation*, pages 324–329. IEEE Press, Piscataway, NJ. 66

Lu, X. et Mori, K. (2003). Autonomous information services integration and allocation in agent-based information service system. *Dans IAT '03 : Proceedings of the IEEE/WIC International Conference on Intelligent Agent Technology*, page 290, Washington, DC, USA. IEEE Computer Society. 131

Luke, S., Panait, L. et al. (2007). Ecj 16 : A java-based evolutionary computation research system. 29, 33

MacDonald, S., Anvik, J., Bromling, S., Schaeffer, J., Szafron, D. et Tan, K. (2002). From patterns to frameworks to parallel programs. *Parallel Comput.*, 28(12):1663–1683. 26

Macêdo, R. J. A. et Silva, F. M. A. (2005). The mobile groups approach for the coordination of mobile agents. *J. Parallel Distrib. Comput.*, 65(3):275–288. 107

Macready, W. G., Siapas, A. G. et Kauffman, S. A. (1995). Criticality and parallelism in combinatorial optimization. Working Papers 95-06-054, Santa Fe Institute. 59

Meignan, D. (2008). *Une approche organisationnelle et multi-agent pour la modélisation et l'implantation de métaheuristiques, Application aux problèmes d'optimisation de réseaux de transports.* Thèse de doctorat, Université de Technologie de Belfort-Montbéliard et Université de Franche-Comté, FRANCE. 3, 4, 32, 33, 40, 53, 54

Melab, N. (2005). *Contribution à la résolution de problèmes d'optimisation combinatoire sur grilles de calcul.* Thèse de doctorat, Lille. 25, 26, 27, 28, 29, 30, 31, 32, 33, 34

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. et Teller, E. (1953). Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092. 16

Michalewicz, Z. (1996). *Genetic algorithms + data structures = evolution programs (3rd ed.)*. Springer-Verlag, London, UK. 162

Michel, L. et Van Hentenryck, P. (2001). Localizer++ : An open library for local search. Rapport technique, Providence, RI, USA. 29, 30

Middendorf, M., Reischle, F. et Schmeck, H. (2000). Information exchange in multi colony ant algorithms. *Dans IPDPS '00 : Proceedings of the 15 IPDPS 2000 Workshops on Parallel and Distributed Processing*, pages 645–652, London, UK. Springer-Verlag. 62, 68

Milano, M. et Roli, A. (2004). Magma : a multiagent architecture for metaheuristics. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 34(2):925–941. 29, 32, 33

Milojičić, D. S., LaForge, W. et Chauhan, D. (1999). Mobile objects and agents (moa). pages 595–610. 46

Mittaz, M. (2000). *Problèmes de cheminements optimaux dans les réseaux avec contraintes associées aux arcs*. Thèse de doctorat, Lausanne. 71, 126

Mladenović, N. et Hansen, P. (1997). Variable neighborhood search. *Comput. Oper. Res.*, 24(11):1097–1100. 16

Mullaseril, P. et Dror, M. (1996). *A set-covering approach for directed node and arc routing problems with split delivery and time windows*. Working Paper, MIS department, University of Arizona, Tucson, Arizona. 142

Nikraz, M., Caire, G. et Bahri, P. (2006). A methodology for the analysis and design of multi-agent systems using jade. 109

Osman, I. et Laporte, G. (1996). Metaheuristics : A bibliography. *Annals of Operations Research*, 63(5):511–623. 15

Overeinder, B. J., Brazier, F. M. T. et de Groot, D. R. A. (2004). Cross-platform generative agent migration. *Dans Proceedings of the Fourth European Symposium on Intelligent Technologies, Hybrid Systems and their implementation on Smart Adaptive Systems*, pages 356–363. EUNITE 2004, June 10-12, 2004, Aachen, Germany, http ://www.eunite.org. 48

Pearn, W. L. (1988). New lower bounds for the capacitated arc routing problem. *Networks*, 18(3):181–191. cited By (since 1996) 7. 189

Pearn, W.-L., Assad, A. et Golden, B. L. (1987). Transforming arc routing into node routing problems. *Comput. Oper. Res.*, 14(4):285–288. 142

Pham, Q. T. (1995). Competitive evolution : A natural approach to operator selection. *Dans AI '93/AI '94 : Selected papers from the AI'93 and AI'94 Workshops on Evolutionary Computation, Process in Evolutionary Computation*, pages 49–60, London, UK. Springer-Verlag. 66, 67

Ramdane-Chérif, W. (2002). *Problèmes d'optimisation en tournées sur arcs*. Thèse de doctorat, Université de technologie, Troyes, FRANCE. 185, 186, 187, 190, 191, 192

Roberts, D. et Johnson, R. (1996). Evolving frameworks : A pattern language for developing object-oriented frameworks. *Dans Proceedings of the Third Conference on Pattern Languages and Programming*. Addison-Wesley. 26

Roch, J. L., Villard, G. et Roucairol, C. (1995). Parallélisme et applications irrégulières. *Algorithmes Irréguliers et Ordonnancement, Hermès*, pages 73–88. 19

Roy, S. et Rousseau, J. (1989). The capacitated canadian postman problem. *INFOR*, 27(1):58–73. 139

Schaffer, J. D., Caruana, R. A., Eshelman, L. J. et Das, R. (1989). A study of control parameters affecting online performance of genetic algorithms for function optimization. *Dans Proceedings of the third international conference on Genetic algorithms*, pages 51–60, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. 66

Schlierkamp-Voosen, D. et Mühlenbein, H. (1994). Strategy adaptation by competing subpopulations. *Dans Parallel Problem Solving from Nature (PPSN III*, pages 199–208. 66, 67

Shao, M.-H. et Zhou, J. (2006). Protecting mobile-agent data collection against blocking attacks. *Computer Standards & Interfaces*, 28(5):600–611. 50

Skolicki, Z. (2005). An analysis of island models in evolutionary computation. *Dans GECCO '05 : Proceedings of the 2005 workshops on Genetic and evolutionary computation*, pages 386–389, New York, NY, USA. ACM. 52

Stützle, T. et Hoos, H. (1998). Improvements on the ant system : Introducing the max - min ant system. *Artificial Neural Networks and Genetic Algorithms. N. C. S. R.F. Albrecht G.D. Smith. New York, Springer Verlag*, pages 245–249. 22

Taillard, E., Gambardella, L. M., Gendreau, M. et Potvin, J.-Y. (1998). Adaptive memory programming : a unified view of metaheuristics. 23, 60

Talbi, E. G. et Bachelet, V. (2006). Cosearch : A parallel cooperative metaheuristic. *Journal of Mathematical Modelling and Algorithms*, pages 5–22. 2

Tanenbaum, A. S. (2007). *Modern Operating Systems*. Prentice Hall Press, Upper Saddle River, NJ, USA. 44

Tanenbaum, A. S. et Steen, M. V. (2001). *Distributed Systems : Principles and Paradigms*. Prentice Hall PTR, Upper Saddle River, NJ, USA. 2

Tanese, R. (1989). Distributed genetic algorithms. *Dans Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 434–439, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc. 52

Thati, P., Chang, P.-H. et Agha, G. (2002). Crawlets : Agents for high performance web search engines. *Dans MA '01 : Proceedings of the 5th International Conference on Mobile Agents*, pages 119–134, London, UK. Springer-Verlag. 107

THOMAS, V. (2005). *Proposition d'un formalisme pour la construction automatique d'interactions dans les systèmes multi-agents réactifs*. Thèse de doctorat, Université Henri Poincaré Nancy 1, France. 41

TONGCHIM, S. et CHONGSTITVATANA, P. (2002). Parallel genetic algorithm with parameter adaptation. *Inf. Process. Lett.*, 82(1):47–54. 66, 67

TOTH, P. et VIGO, D. (2001). An overview of vehicle routing problems. pages 1–26. 142

TOULOUSE, M., CRAINIC, T. G. et GENDREAU, M. (1995). Communication issues in designing cooperative multi-thread parallel searches. *Dans Meta-Heuristics : Theory and Applications*, pages 501–522. Kluwer Academic Publishers. 23, 60

ULUSOY, G. (1985). The fleet size and mix problem for capacitated arc routing. *European Journal of Operational Research*, 22(3):329–337. 91

VAN THANH, D. (2001). Using mobile agents in telecommunications. *Dans DEXA '01 : Proceedings of the 12th International Workshop on Database and Expert Systems Applications*, page 685, Washington, DC, USA. IEEE Computer Society. 44

VERHOEVEN, M. G. A. et AARTS, E. H. L. (1995). Parallel local search. *Journal of Heuristics*, pages 43–65. 19

WALL, M. (1999). Galib : A c++ library of genetic algorithm components. http ://lancet.mit.edu/ga/dist/. 25

WANG, D. (1995). Earliness/tardiness production planning approaches for manufacturing systems. *Comput. Ind. Eng.*, 28(3):425–436. 160

WANG, X. et TANG, L. (2009). A population-based variable neighborhood search for the single machine total weighted tardiness problem. *Computers & Operations Research*, 36(6):2105 – 2110. 1

WHITE, J. E. (1997). Mobile agents. 3

WRIGHT, A. H. (1991). Genetic algorithms for real parameter optimization. *Dans* RAWLINS, G. J., éditeur : *Foundations of genetic algorithms*, pages 205–218. Morgan Kaufmann, San Mateo, CA. 162

Résumé

Ce manuscrit décrit les travaux de recherche effectués au cours de ma thèse, au sein de l'équipe informatique et recherche opérationnelle du laboratoire CREAM[1], en collaboration avec le laboratoire LISA[2], et avec le soutien du Conseil Général de la ville d'Angers. Ces travaux de recherche se situent à l'intersection des domaines de l'optimisation combinatoire et des systèmes multi-agents. Ils s'inscrivent dans la continuité des propositions de modèles ou de plates-formes pour les métaheuristiques parallèles.

Ce rapport réunit différentes notions du parallélisme, du paradigme multi-agents et des métaheuristiques afin d'apporter des méthodes de résolution performantes (robustes et auto-adaptatives) à des problèmes d'optimisation combinatoire réels. Il démontre que l'introduction de stratégies de parallélisation et d'échange d'informations à un algorithme à population permet à ce dernier d'améliorer considérablement ses facultés de recherche de solutions. En outre, l'utilisation des agents mobiles permet une exploitation optimale des ressources de calcul inutilisées dans un organisme (laboratoire, entreprise) et de favoriser ainsi l'autonomie des processus de calcul pour pouvoir gérer les éventuelles pannes dans un réseau. Le succès de cette approche dans la résolution d'un problème de tournées de véhicules et d'un problème d'ordonnancement de production, montre l'intérêt pratique de ces méthodes et leurs retombées économiques potentielles.

Ce travail de recherche représente l'une des premières explorations des possibilités offertes par deux domaines fort prometteurs de l'intelligence artificielle distribuée et de la recherche opérationnelle. L'union de méthodes auto-adaptatives et d'une puissance de calcul imposante pourrait fort bien se révéler un outil performant pour la résolution de problèmes d'une telle envergure.

Mots clés : Optimisation combinatoire, agents, échange d'informations.

[1] Centre de Recherche et d'Étude des Applications de Mathématiques, Université Catholique de l'Ouest

[2] Laboratoire d'Ingénierie des Systèmes Automatisés, Université d'Angers

Information exchange strategies in distributed computing systems

Abstract : This thesis describes my *PHD* work carried out within the computer science and operations research group of the CREAM[3] laboratory, in collaboration with the LISA[4] laboratory, and the support of Angers's General Council. The reported work belongs at the same time to the multi-agent system (MAS), parallelism and metaheuristics domains, and more precisely to develop an intelligent distributed computing system and to compare different information exchange strategies in a distributed memory architecture for parallel computing (Networked PCs).

This thesis combines different notions of parallelism, multi-agent paradigm and metaheuristics to provide a high performance method to solve real combinatorial problems. It proves that including information exchange and parallelisation strategies to a population-based algorithm can improve it ability. Instead of using expensive parallel computing facilities, in this work we propose to implement parallel computation model on easily available networked PCs, and present a multi-agent framework to support parallelism. In this model, agents can easily be programmed to initiate the execution of actions autonomously, and they can dynamically move from machine to machine to discover services and communicate with other agents. To evaluate the proposed approach, different kinds of experiments have been conducted to assess the developed system mainly on a vehicule routing problem and an industrial scheduling problem. Results obtained show the efficiency of our approach.

Keywords : Combinatorial optimization, agents, information exchange strategies.

[3]Centre de Recherche et d'Étude des Applications de Mathématiques, Université Catholique de l'Ouest
[4]Laboratoire d'Ingénierie des Systèmes Automatisé, Université d'Angers