THESE

pour obtenir le grade de

HABILITATION A DIRIGER DES RECHERCHES DE L'USTL
DISCIPLINE : INFORMATIQUE

présentée et soutenue publiquement,

le 28 Novembre 2005, par

NOUREDINE MELAB

Contributions à la résolution de problèmes d'optimisation combinatoire sur grilles de calcul

Jury

Président :	Jean-Marc Geib	Professeur, USTL, Lille
Rapporteurs :	Franck Cappello	Directeur de Recherche INRIA Futurs
	Denis Caromel	Professeur, Université de Nice - Sophia Antipolis
	Catherine Roucairol	Professeur, Université de Versailles
Examinateurs :	Michel Daydé	Professeur, INP - ENSEEIHT, Toulouse
	Jean-Louis Pazat	Professeur, INSA de Rennes
Directeur :	El-Ghazali Talbi	Professeur, USTL, Lille

Universite des Sciences et Technologies de Lille
LIFL - UMR 8022 - Cité Scientifique, Bât. M3 Ext. - 59655 Villeneuve d'Ascq Cedex

A la mémoire de mes deux frères Madjid et Mouloud
décédés respectivement en mars et juin 2005

A Carole et notre fille Clara

A ma mère, mes frères et soeurs

Remerciements

Je tiens à remercier vivement El-Ghazali Talbi, responsable de l'équipe CNRS/OPAC et du projet INRIA DOLPHIN, pour la confiance qu'il m'a témoignée en m'accueillant dans son équipe. Cette confiance a été également appuyée par le LIFL, que ce laboratoire trouve ici toute ma reconnaissance.

Mes vifs remerciements vont également à Jean-Marc Geib, Directeur du LIFL, pour son soutien et ses encouragements pour la préparation de mon HDR.

La synthèse des travaux de recherche présentés dans ce document sont le résultat d'un travail d'équipe réalisé principalement avec Sébastien Cahon (thèse soutenue en juillet 2005) et Mohand Mezmaz (thèse en cours). Je tiens donc à les remercier avec intensité pour le sérieux, la curiosité et la grande capacité de travail dont ils ont fait preuve.

Je remercie les membres de l'équipe OPAC pour tous les échanges et discussions constructifs que nous avons eus depuis mon arrivée et pour la bonne ambiance de travail qui règne au sein de l'équipe.

Je remercie également mes anciens collègues de l'équipe "Modélisation et Évolution du Logiciel (MODEL)" du Laboratoire d'Informatique du Littoral (LIL), en particulier Henri Basson, Mourad Bouneffa et Laurent Deruelle.

Le projet national Grid5000 a représenté une partie importante de mes activités en matière d'animation de la recherche. Je remercie toute la communauté Grid5000, notamment le comité de pilotage et le comité technique dont je fais partie. Avec le projet Grid5000, j'ai beaucoup appris sur la gestion de projets d'envergure nationale et sur la mise en place d'une grille. Ce projet nous a également permis d'avancer dans nos recherches sur l'algorithmique des grilles de calcul pour la réalisation de défis dans la résolution de problèmes de grande taille.

Je tiens à remercier nos partenaires des projets de collaboration, notamment DOC-G de l'ACI GRID, GGM de l'ACI Masse de données, DOCK et CHOC de l'ANR "Calcul Intensif et Grilles de Calcul".

J'adresse mes remerciements à Jean-Marc Geib (Professeur, Université de Lille1) d'avoir accepté de présider le jury de soutenance de mon HDR.

Je tiens à remercier les rapporteurs des travaux présentés dans ce document : Franck Cappello (Directeur de Recherche INRIA Futurs), Denis Caromel (Professeur, Université de Nice - Sophia Antipolis) et Catherine Roucairol (Professeur, Université de Versailles).

Je remercie également Michel Daydé (Professeur, INP - ENSEEIHT, Toulouse) et Jean-Louis Pazat (Professeur, INSA de Rennes) d'avoir chaleureusement accepté de participer au jury.

Je tiens enfin à remercier vivement Carole et notre fille Clara d'avoir compris et accepté ma faible disponibilité pendant la préparation de mon HDR. Merci à Carole pour son précieux soutien et ses encouragements.

Mon parcours depuis le Doctorat

Travaux de Doctorat et Post-Doctorat (1992-1998)

Depuis 1992, mes activités de recherche s'inscrivent principalement dans le domaine des systèmes distribués et parallèles. Dans le cadre de ma thèse de Doctorat, j'ai travaillé sur les problèmes de granularité et de régulation de charge dans le contexte des langages fonctionnels. A ce titre, j'ai proposé une approche statique de grossissement de la granularité et un mécanisme de régulation de charge auto-adaptative à l'environnement. Ce mécanisme m'a particulièrement valu un prix IEEE à la conférence *Parallel and Distributed Computing Systems'1996*.

Au milieu des années 1990, le domaine du calcul parallèle distribué a connu l'émergence des méta-systèmes intégrant des réseaux de stations de travail, des machines parallèles, etc. La caractéristique principale de tels environnements est l'hétérogénéité et la volatilité des machines due, d'une part, aux pannes matérielles et/ou logicielles de celles-ci, et, d'autre part, à leur libération/réquisition fréquente par leurs propriétaires. Ce caractère adaptatif ne pouvait alors être pris en compte sans revoir la méthodologie de programmation parallèle ainsi que la manière d'aborder les problématiques telles que la régulation de charge, la tolérance aux pannes, etc. Ces questions ont fait l'objet de mes activités de recherche menées après ma thèse. Celles-ci ont porté sur le calcul scientifique parallèle adaptatif et ont été réalisées dans le cadre d'une collaboration avec les équipes MAP (S. Petiton) et GOAL (E-G. Talbi) du LIFL. Ces travaux constituent une première étape des travaux réalisés aujourd'hui sur les grilles de calcul.

Travaux de recherche au Laboratoire d'Informatique du Littoral (LIL) (1998-2002)

En 1998, j'ai été nommé à l'Université du Littoral - Côte d'Opale sur un poste MCF 27^e. J'ai alors intégré l'équipe *Modélisation et Evolution du Logiciel (ModEL)* du LIL. J'ai, en même temps, continué à travailler en collaboration avec l'équipe Optimisation PArallèle Coopérative (OPAC) (Responsable : E-G. Talbi).

Le projet de l'équipe ModEL porte sur la modélisation et la gestion de l'évolution des applications distribuées. Ce type d'applications est vu comme un ensemble de composants (*e.g.* une fonction) reliés par des relations (*e.g.* un appel de fonction). Le contrôle de l'évo-

lution consiste à déterminer, *a priori*, l'impact du changement effectué sur un composant d'une application distribuée, sur les autres composants de cette application. Le savoir-faire acquis sur les systèmes distribués et leur performance m'a permis de participer (25%) à l'encadrement de la thèse de Laurent Deruelle et de deux projets de DEA. Ce travail a abouti à la conception d'une plate-forme logicielle libre pour l'analyse de code distribué et la gestion de son évolution. Parallèlement à cette activité, j'ai mis en place avec la collaboration de mes collègues une formation de DESS sur l'ingénierie des systèmes informatiques distribués, et en ai assuré la responsabilité jusqu'à mon départ (pendant 2 ans). Ce DESS a été très utile à un laboratoire qui manquait d'une formation de DEA et de thésards.

La collaboration avec l'équipe OPAC a porté sur l'optimisation parallèle coopérative appliquée à la fouille de données, et a fait l'objet de 2 co-encadrements (50%) de projets de DEA. Il s'agit de la modélisation de certaines tâches de fouille de données en problèmes d'optimisation combinatoire et leur résolution avec des algorithmes génétiques parallèles. Cette collaboration m'a permis de rester en contact avec le LIFL.

Travaux de recherche au LIFL (depuis 2002)

En septembre 2002, j'ai été muté à Polytech'Lille (Université de Lille1), ce qui m'a permis de rejoindre l'équipe OPAC. Depuis mon arrivée dans l'équipe, j'ai participé à la rédaction du projet DOLPHIN de l'INRIA Futurs portant sur l'optimisation multi-objectif parallèle coopérative. Dans le cadre de ce projet, une délégation INRIA à mi-temps m'a été accordée pour l'année 2005-2006. J'ai également participé à la rédaction du projet GGM (Grille Géno-Médicale) de l'ACI "Masse de données". Il s'agit d'une collaboration entre l'équipe OPAC du LIFL et des laboratoires LIRIS (INSA de Lyon) et IRIT (Toulouse). Notre contribution porte sur l'application de l'optimisation combinatoire parallèle à l'extraction de connaissances à partir d'un entrepôt de données réparti sur une grille. J'ai aussi participé au montage d'autres projets autour de l'optimisation sur grilles, notamment les projets CHOC (CHallenges en Optimisation Combinatoire) et DOCK (Conformation sampling and docking on grids) de l'ANR "Calcul intensif et Grilles de Calcul".

Depuis 2002, j'ai participé au projet ParadisEO (thèse de S. Cahon) de conception d'une plate-forme logicielle d'aide à la mise en oeuvre de métaheuristiques multi-objectifs parallèles hybrides sur grappes et grilles de calcul. Cette plate-forme s'inscrit dans le cadre du projet DOC-G (Défis en Optimisation Combinatoire sur Grilles) de l'ACI GRID (collaboration avec les équipes OPALE du PRiSM et P3 de l'ID-IMAG) et d'un contrat avec France Telecom R&D. Le projet nécessite, en plus de l'optimisation parallèle, des compétences en modélisation logicielle et méthodologie de mise en place de plates-formes. Mon passage à l'Université du Littoral m'a permis d'apporter le savoir-faire nécessaire. Un couplage de la plate-forme avec les méthodes exactes multi-objectifs est en cours (thèse de M. Mezmaz). Ce couplage, initié dans le projet DOC-G, a pour objectif de produire efficacement des solutions exactes qui serviront d'étalons pour l'évaluation de la qualité des métaheuristiques.

Avec le soutien de Jean-Marc Geib, Directeur du LIFL, j'ai monté et porté le dossier de réponse à l'appel d'offres du Programme national GRID5000 de l'ACI GRID. En tant que

membre à la fois du comité de pilotage (CP) et du comité technique (CT) de GRID5000, j'ai assuré la gestion technique et administrative du projet sur le site de Lille : rédaction du CCTP, analyse technique des réponses des soumissionnaires avec rédaction d'un rapport, réunions des CP et CT, demandes d'aides financières, recrutement d'un ingénieur INRIA expert pour GRID5000, etc. Autour de GRID5000, l'activité des grilles est aujourd'hui fortement soutenue par le LIFL et l'Université de Lille1.

Table des matières

Chapitre 1

Introduction

Les travaux présentés dans ce document s'incrivent dans le cadre de mes activités de recherche au sein de l'équipe *Optimisation PArallèle Coopérative (OPAC)* du laboratoire CNRS/LIFL de l'Université de Lille1 et du projet DOLPHIN de l'INRIA Futurs.

En optimisation combinatoire, qu'ils soient académiques (génériques) ou industriels (souvent spécifiques), les problèmes sont en général complexes et reconnus NP-difficiles. En pratique, leur modélisation est souvent en constante évolution en termes de contraintes et d'objectifs. En outre, malgré l'importance de leur taille les délais souhaités voire imposés pour leur résolution sont de plus en plus courts. La qualité, en matière de robustesse et d'efficacité, et la viabilité des méthodes de résolution de ces problèmes se mesureront alors à leur capacité à prendre en compte ces caractéristiques. La conception de telles méthodes doit s'appuyer sur l'utilisation conjointe d'approches avancées issues de l'optimisation combinatoire, du parallélisme à grande échelle et du génie logiciel.

Selon la qualité et les délais d'obtention exigés des solutions, on distingue deux grandes familles de méthodes d'optimisation : les méthodes *exactes* et les méthodes *approchées* ou *heuristiques*. Parmi les heuristiques, on distingue les *métaheuristiques* s'appliquant potentiellement à une variété de types de problèmes. Il existe également des algorithmes hybrides, ayant permis d'obtenir les meilleurs résultats [Tal02], combinant différentes méthodes aux comportements complémentaires.

Les méthodes exactes permettent de trouver des solutions exactes avec preuve d'optimalité mais s'avèrent vite inutilisables sur des problèmes de grande taille. Les métaheuristiques permettent de produire en temps raisonnable des solutions de bonne qualité, parfois proches de l'optimum, mais s'appliquent à des instances de taille nettement supérieure. Cependant même si, par définition, elles réduisent de manière significative la taille de l'espace de recherche exploré, il n'empêche que leur combinatoire demeure importante sur des problèmes de très grande taille. Les algorithmes hybrides apportent à la fois plus de robustesse et une meilleure qualité des solutions. Cependant, leur coût en temps CPU s'avère exorbitant, ce qui les rend souvent inexploitables sur des problèmes de grande taille. Il en résulte que très souvent, faute de ressources de calcul suffisantes, les problèmes sont simplifiés pour réduire leur taille et pouvoir les traiter, au risque de fausser leur définition et les résultats associés.

Le parallélisme à grande échelle, basé sur l'utilisation de grilles informatiques, s'avère un moyen indispensable pour supporter le coût des méthodes d'optimisation et de leur hybridation. Il permet de traiter efficacement la combinatoire importante associée aux problèmes de taille réelle en produisant des solutions de meilleure qualité. L'exploitation du parallélisme à grande échelle nécessite la gridification des modèles parallèles et des mécanismes d'hybridation en vue de leur adaptation aux caractéristiques des grilles de calcul. Ces dernières diffèrent d'une définition de la grille à une autre [Fos02, Gri02, BLDGS03]. Dans nos travaux, nous considérons les caractéristiques présentées dans [BBL02]. Il s'agit d'une grille composée d'un ensemble de ressources de calcul hétérogènes et volatiles, et réparties sur plusieurs domaines d'administration autonomes et inter-connectés par un réseau à grande échelle.

La gridification des méthodes d'optimisation parallèles hybrides nécessite la prise en compte des caractéristiques des grilles, c'est à dire la résolution des problèmes inhérents, notamment la sécurité (traversée de pare-feu), la tolérance aux pannes, la minimisation des coûts de communication et la mesure de performances dans un contexte hétérogène. Même si l'utilisation d'intergiciels permet de gérer une partie de ces problèmes de manière transparente, ces derniers ne répondent pas totalement aux besoins des applications. Par exemple, certains intergiciels ne sont pas adaptés au parallélisme en ce sens qu'ils ne supportent pas la génération dynamique de nouvelles tâches et la communication entre elles. Ces intergiciels peuvent être également limités au regard des solutions qu'ils proposent aux problèmes évoqués. Par exemple, pour traiter le problème de volatilité, on peut préférer pour des raisons d'efficacité la solution de sauvegarde/restauration (ou *checkpointing*) à la reprise de l'exécution depuis le début (*from scratch*) proposée par la plupart des intergiciels.

Il existe aujourd'hui peu de travaux proposant des implémentations sur grilles de calcul réalisées avec des intergiciels tels que Condor-MW [LKGY00], Globus [FK97], NinfG [TNS+03] ou des simulateurs de grilles tels que Parsec [BMT+98]. Cependant, même si ces implémentations apportent quelques enseignements sur la gridification elles restent isolées et ne sont pas réutilisables. La meilleure réutilisation logicielle passe par l'utilisation de librairies d'algorithmes d'optimisation, ou encore mieux par les plates-formes (ou *frameworks*). En plus de la réutilisation de code offerte par les librairies, les plates-formes permettent la réutilisation de conception. Plusieurs librairies et plates-formes ont été proposées et sont disponibles sur le Web. Cependant, non seulement la plupart d'entre elles ne sont plus maintenues, elles sont souvent limitées en termes de méthodes fournies, et surtout en matière d'accès transparent au parallélisme en particulier à grande échelle. C'est pourquoi la plate-forme logicielle ParadisEO que nous avons proposée constitue une contribution palliant ces limites.

Ce document est structuré en trois grandes parties identifiant chacune un front d'étude de l'optimisation combinatoire sur grilles de calcul. Dans la première partie, nous proposons une analyse des méthodes d'optimisation combinatoire dans le contexte des grilles. Cette analyse est un conjugaison de travaux publiés dans la littérature et de notre propre expérience détaillée dans les deux parties suivantes. Le deuxième front d'étude concerne la proposition, dans le cadre de la thèse de Sébastien Cahon soutenue le 1^{er} juillet 2005, d'une plate-forme logicielle d'aide à la conception de métaheuristiques parallèles sur grilles.

La troisième partie porte sur les méthodes exactes parallèles et leur coopération avec les métaheuristiques parallèles sur grilles de calcul. Cette partie est issue principalement des travaux menés depuis 2 ans dans le cadre de la thèse de Mohand Mezmaz.

Optimisation combinatoire parallèle hybride sur grilles

En optimisation combinatoire parallèle, quelques travaux ont porté sur l'étude des modèles parallèles pour les métaheuristiques [CP97, CMRR02, AT02a, TAML05] ou les méthodes exactes [GC94, CDC+94]. Les études réalisées comportent souvent une taxinomie, un état de l'art et une analyse des différents modèles au regard des problèmes du parallélisme tels que la synchronisation, la régulation de charge, la granularité de parallélisme, l'évaluation de performances, etc. D'autres études ont été également dédiées aux mécanismes d'hybridation de méthodes d'optimisation [Tal02]. Le modèle d'évaluation parallèle d'une solution ou borne est souvent ignoré dans ces études. Cependant, son utilisation conjointe avec d'autres modèles (pour former ensemble un modèle hiérarchique) peut grandement améliorer l'efficacité d'exécution sur une grille de calcul. D'autre part, même si ces études sont riches en enseignements sur la conception parallèle et/ou hybride de méthodes d'optimisation, elles ne sont pas totalement adaptées aux grilles de calcul. En effet, les caractéristiques évoquées précédemment ne sont souvent pas prises en compte dans l'analyse, cela étant dû principalement à la jeunesse du domaine des grilles informatiques. Pour pallier cette limite, nous avons proposé une première étude des modèles parallèles et des mécanismes d'hybridation dans le contexte des grilles de calcul. Cette analyse pourrait servir d'ébauche d'aide à la gridification des méthodes d'optimisation.

Notre étude souligne, entre autres, l'importance du modèle asynchrone, notamment du vol de cycles dans un environnement d'exécution volatile, hétérogène et multi-domaine d'administration. En outre, elle permet d'identifier pour chaque modèle l'information à stocker/restaurer pour la mise en oeuvre du mécanisme de sauvegarde/restauration pour la gestion de la volatilité liée à la disponibilité variable et aux pannes des machines. Conjuguées à la nature stochastique des métaheuristiques et au non déterminisme des méthodes exactes, les caractéristiques hétérogène et volatile des grilles rendent plus complexe le problème d'évaluation de performances. Une étude, inspirée des travaux existants [GC94, LKGY00, AT02b], sur les indicateurs de performance et la manière de les évaluer est proposée. Pour les méthodes exactes, la nature irrégulière de l'arborescence explorée rend la tâche de régulation de charge fortement sollicitée. Dans un contexte à grande échelle, il est important de minimiser le coût des communications inhérentes en proposant un codage adapté des unités de travail. Enfin, pour assurer une réutilisabilité des différents modèles nous avons identifié pour chacun d'eux la partie invariante et la partie spécifique au problème traité. Cette séparation conceptuelle est une étape importante dans la démarche méthodologique de conception d'une plate-forme logicielle facilitant la réutilisation de code et de conception.

Une plate-forme d'aide à la conception de métaheuristiques parallèles sur grilles

Il existe peu de travaux proposant un état de l'art des plates-formes logicielles pour les métaheuristiques [VW02]. De plus, la plupart de ces travaux ne prennent pas en compte le parallélisme, encore moins le parallélisme sur grilles. Nous avons donc proposé un état de l'art des plates-formes existantes et une étude comparative basée sur différents critères qui nous semblent majeurs. Parmi ces critères, on distingue notamment le support des modèles parallèles et d'hybridation ainsi que leur gridification. Nous avons considéré également la capacité de réutilisation de code et de conception, la facilité et la flexibilité d'adaptation pour le traitement de nouveaux problèmes et enfin l'utilité et la portabilité. L'utilité d'une plate-forme est évaluée à sa capacité à fournir différents types de méthodes d'optimisation mono et multi-objectifs, de modèles parallèles et mécanismes d'hybridation, etc.

L'étude comparative réalisée a montré les limites des plates-formes d'optimisation existantes surtout en matière de parallélisme en particulier à grande échelle. Ces limites ont motivé la proposition d'une nouvelle plate-forme dans le cadre du projet ACI DOC-G [1], appelée ParadisEO, d'aide à la conception de métaheuristiques pour différents types d'architectures parallèles et notamment les grilles de calcul. Comparée aux autres plates-formes, ParadisEO apparaît comme l'une des plus complètes au regard des critères évoqués ci-dessus. Cette plate-forme a été validée sur deux problèmes réels : la conception (ou *design*) de réseaux cellulaires (Contrat France Telecom R&D) et l'extraction de connaissances en spectroscopie proche infrarouge (PIR) (Collaboration avec le laboratoire LASIR, Université de Lille1).

Méthodes exactes parallèles et leur coopération avec les métaheuristiques sur grilles

La nature irrégulière de l'arborescence explorée par les méthodes exactes et le caractère volatile des grilles exigent un nombre important d'opérations de régulation de charge à l'exécution. Ces opérations se traduisent par un coût exorbitant de transfert et de stockage des unités de travail (ensembles de noeuds) dynamiquement généré(e)s. Il est donc primordial de disposer d'un codage des unités de travail qui minimiserait le coût de communication sur une grille volatile et à grande échelle. Nous avons proposé une gridification de l'algorithme *Branch-and-Bound* (*B&B*) basée sur un codage particulier de ces unités de travail. L'information communiquée et/ou stockée est un descripteur se résumant principalement à un couple d'entiers désignant les numéros de deux noeuds délimitant une unité de travail. Ce codage permet d'une part de minimiser le coût des communications. D'autre part, le coût de chaque communication étant constant, cela facilite l'évaluation des coûts globaux de communication et de stockage induits par la régulation de charge et la gestion des reprises sur erreurs.

Comme il a été évoqué précédemment, l'utilisation d'intergiciels pour la programmation des grilles rend transparente la gestion de certains problèmes difficiles. Cependant, ils ne

[1] DOC-G : Défis en Optimisation Combinatoire sur Grilles (partenaires du projet : PRiSM (Université de Versailles), ID-IMAG (ENSIMAG-INPG) et LIFL (Université de Lille1)).

sont pas totalement adaptés au parallélisme coopératif à granularité fine ou moyenne. Par exemple, les intergiciels de type *Coordinateur-Travailleurs* (ou *Dispatcher-Worker*) ne prennent souvent pas en compte la génération dynamique de tâches et leur coopération à l'exécution. Nous avons donc proposé pour ce type d'intergiciels un modèle de coopération inspiré de Linda [Gel85] et adapté à l'optimisation multi-objectif sur grilles. Ce modèle peut être intégré comme couche de coordination dans ces intergiciels. Une première intégration de ce modèle a été réalisée dans la version *v1r2-rc6* de XtremWeb [Fed03].

Les premières expérimentations ont été réalisées sur une grille de 412 machines réparties sur 4 domaines d'administration appartenant à différents établissements de l'Université de Lille1. L'approche proposée a été appliquée au problème Flow-Shop [TB02] bi-objectif. Le problème consiste à ordonnancer N travaux sur M machines suivant le principe de la chaîne de production en : (1) minimisant la date de fin du dernier travail sur la dernière machine (*makespan*) ; (2) minimisant la somme des retards des travaux. Les expérimentations ont été réalisées sur l'instance 0 du problème 50 travaux sur 5 machines (une des instances générées par E. Taillard [2]). Les résultats montrent que le codage proposé des unités de travail permet une régulation de charge efficace et que le coût de la sauvegarde/restauration est presque négligeable. Ils ont également montré l'apport de l'hybridation des métaheuristiques entre elles et avec les méthodes exactes sur des problèmes de grande taille.

D'autres résultats expérimentaux ont été obtenus après la soutenance d'Habilitation à Diriger des Recherches de l'auteur de ce document. Ces résultas n'étaient donc pas présentés dans la version de ce document soumise aux rapporteurs de ce travail. Deux types d'expérimentations ont été réalisés : d'abord, l'exploitation de l'approche proposée pour la résolution sur 412 processeurs de l'instance 1 du problème *bi-objectif 50 travaux sur 5 machines* beaucoup plus gourmande en temps CPU que l'instance 0 ; ensuite, l'application de l'approche à l'instance $TA056$ du problème *mono-objectif* (*makespan* seulement) *50 travaux sur 20 machines*. Cette instance n'a jamais été résolue de manière exacte depuis 1989. L'approche a permis de produire la solution optimale de cette instance en *7 semaines* en utilisant en moyenne 500 processeurs appartenant à Grid5000 et aux réseaux d'enseignement de l'Université de Lille1. Un pic de 1245 processeurs a été atteint pendant une nuit.

Plan du document

Ce document est composé de trois chapitres résumant le bilan de nos travaux et les contributions apportées. Le chapitre 1 décrit les principes de l'optimisation combinatoire mono et multi-objectif, ses modèles parallèles et mécanismes d'hybridation. Nous y présentons une analyse issue de notre première expérience de ces modèles et mécanismes dans le contexte des grilles de calcul. Nous y identifions pour chaque méthode d'optimisation et pour chaque modèle parallèle la partie générique et réutilisable et la partie spécifique au problème traité. Cette séparation est la base de la conception d'une manière générale d'une plate-forme logicielle d'aide à la réalisation de métaheuristiques parallèles hybrides

[2] `http://ina2.eivd.ch/Collaborateurs/etd/problemes.dir/ordonnancement.dir/ordonnancement.html`

sur grilles. Plus particulièrement, elle représente le fondement de la conception de notre plate-forme *ParadisEO* présentée dans le chapitre suivant. La fin du chapitre 1 est consacrée aux problèmes d'évaluation de performances liés aux caractéristiques des méthodes d'optimisation et des grilles.

Le chapitre 2 présente une taxonomie, un état de l'art et une étude comparative des plates-formes de métaheuristiques, en particulier parallèles hybrides, en expliquant les motivations de la plate-forme ParadisEO. La conception et l'architecture de celle-ci sont ensuite présentées en tenant compte de l'analyse décrite dans le chapitre précédent. Pour valider la plate-forme, deux applications sont présentées : la conception (ou *design*) de réseaux cellulaires et l'extraction de connaissances en spectroscopie proche infrarouge (PIR).

Le chapitre 3 est consacré aux méthodes exactes parallèles et leur coopération avec les métaheuristiques parallèles sur grilles. Nous y présentons l'approche proposée pour la gridification de l'algorithme $B\&B$ et le mécanisme de coopération proposé et intégré à la première version de l'intergiciel XtremWeb pour l'implémentation de cet algorithme. Une évaluation expérimentale est enfin présentée pour mesurer l'apport du parallélisme et de la coopération avec les métaheuristiques sur grilles.

Les annexes de ce document contiennent mon curriculum vitae et quelques publications significatives.

Chapitre 2

Optimisation combinatoire parallèle hybride sur grilles

Un problème d'optimisation combinatoire (POC) peut être formulé comme une optimisation (minimisation ou maximisation) d'une fonction coût (optimisation mono-objectif) ou d'un vecteur de fonctions coûts (optimisation multi-objectif). Cette (ces) fonction(s) est (sont) appelées *fonction(s) objectif(s)*. Dans ce document, nous allons considérer les problèmes de minimisation. Les problèmes de maximisation peuvent être définis de manière similaire. Nous pouvons formuler un POC de la manière suivante :

$$\text{POC} \left\{ \begin{array}{ll} Min & F(x) = (f_1(x), f_2(x), ..., f_n(x)) \\ S.C & x \in C \end{array} \right.$$

où n est le nombre de fonctions objectifs (dimension de l'espace objectif), $\vec{x} = (x_1, x_2, ..., x_K)$ représente le vecteur (de variables) de décision (espace de décision de dimension K), C désigne l'ensemble des solutions réalisables associées à des contraintes d'égalité ou d'inégalité, et $F(\vec{x}) = (f_1(x), f_2(x), ..., f_n(x))$ est le vecteur des objectifs à optimiser. Dans le cas où $n = 1$ (resp. $n \geq 2$), on parle d'optimisation mono-objectif (resp. multi-objectif). Dans le cas multi-objectif, les objectifs sont contradictoires i.e. l'amélioration de la valeur d'un objectif entraîne la dégradation des valeurs des autres et inversement.

En considérant un seul objectif (le i^{eme} par exemple), une solution x_1 est dite *préférée* à une autre solution x_2 si tout simplement $f_i(x_1) < f_i(x_2)$. Par contre, dans le cas multi-objectif la notion de préférence est plus complexe. Dans ce cas, nous utiliserons une relation d'ordre partiel appelée *relation de dominance* au sens Pareto pour comparer deux solutions. Nous dirons alors qu'une solution $y = (y_1, y_2, ..., y_n)$ domine une solution $z = (z_1, z_2, ..., z_n)$ si et seulement si : $\forall i \in [1..n], yi \leq z_i$ *et* $\exists i \in [1..n]/y_i < z_i$.

Comme les différents objectifs sont contradictoires, un vecteur est rarement optimum pour tous les objectifs. Une autre définition de la notion d'optimalité est alors nécessaire. Le concept généralement utilisé est la notion d'*optimalité Pareto*. Une solution $x^* \in C$ est dite *Pareto optimale* si et seulement si il n'existe pas de solution $x \in C$ telle que $F(x)$ domine $F(x^*)$. La résolution d'un problème d'optimisation combinatoire mono-objectif consiste donc à trouver la solution réalisable qui minimise la fonction objectif. Dans le

contexte multi-objectif, la résolution d'un problème consiste à trouver un ensemble PO de solutions Pareto optimales, appelé *frontière ou front Pareto*.

Selon la qualité exigée des solutions, deux familles de méthodes de résolution peuvent être utilisées : les *méthodes exactes* et les *méthodes approchées* ou *heuristiques* (figure 2.1). Les méthodes exactes (Branch-and-X, programmation dynamique, programmation par contraintes ou PPC, etc.) permettent de trouver des solutions exactes avec preuve d'optimalité, cependant elles restent inutilisables en pratique sur des instances de grande taille. A l'inverse, les heuristiques produisent de bonnes solutions, mais sont applicables à des instances de taille plus importante. Les heuristiques peuvent être *spécifiques* à un problème donné, mais elles peuvent également être génériques i.e. applicables à différents types de problèmes dans quel cas elles sont appelées *métaheuristiques*. Ces dernières peuvent être *à solution unique* (méthodes de descente, recherche tabou, récuit simulé, ...) ou *à population de solutions* (algorithmes évolutionnaires, colonies de fourmis, ...). Une troisième famille dérivant les deux grandes familles de méthodes regroupe les *algorithmes hybrides*, permettant de combiner différentes méthodes pour tirer profit des avantages de chacune d'elles. Les méthodes hybridées peuvent appartenir à la même famille ou à différentes familles.

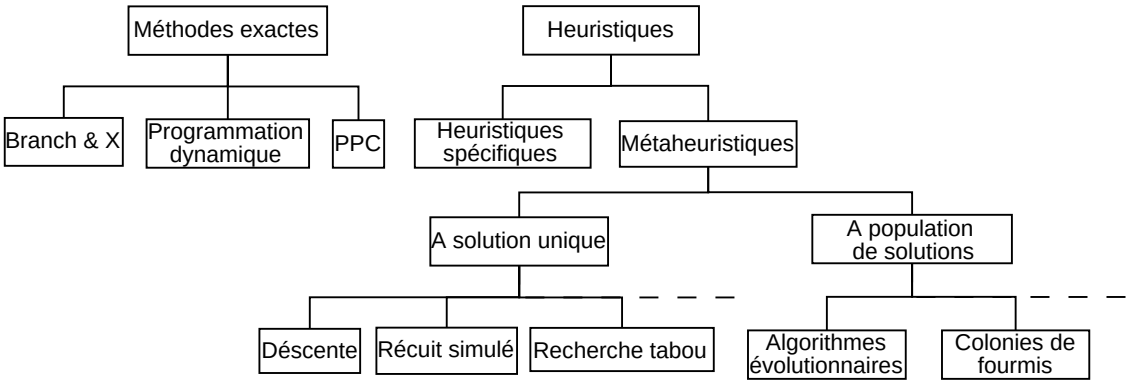

FIG. 2.1 – Une taxinomie des méthodes de résolution en optimisation combinatoire.

Même si les heuristiques permettent de réduire de manière significative l'espace de recherche, l'utilisation du parallélisme à grande échelle reste un moyen indispensable pour résoudre des problèmes de très grande taille. Ce moyen est également nécessaire pour repousser les limites en termes de ressources matérielles afin de supporter le besoin intensif des méthodes exactes et le coût des mécanismes d'hybridation qui s'avère souvent exorbitant.

Dans la suite de ce chapitre, nous allons analyser les méthodes d'optimisation combinatoire parallèles hybrides sur grilles de calcul. Nous allons d'abord définir le contexte des grilles dans lequel nous nous plaçons en donnant les caractéristiques de celles-ci. Nous décrirons les rares travaux sur l'optimisation combinatoire réalisés dans ce contexte. Ensuite, nous donnerons les principes des algorithmes d'optimisation mono et multi-objectifs. Nous détaillerons surtout les modèles parallèles et mécanismes d'hybridation associés en analysant leur gridification. Au cours de l'analyse, nous nous attacherons à définir pour chaque modèle et mécanisme les données (ou la mémoire) nécessaires au mécanisme de sauvegarde/restauration pour gérer la volatilité de la grille. Dans la perspective de proposer

une plate-forme logicielle d'optimisation sur grilles (décrite au chapitre 2), nous identifierons pour chaque méthode et modèle parallèle la partie générique et la partie spécifique au problème.

2.1 Grilles de calcul

Au cours de ces deux dernières décennies, les systèmes distribués et parallèles ont connu une évolution significative sur les plans matériel et logiciel. En si peu de temps, ce domaine est passé des super-calculateurs aux architectures variées (vectorielles, à mémoire partagée ou distribuée) aux grilles informatiques, en passant par les réseaux (*Networks of Workstations ou NOW*) et grappes (*Clusters of Workstations ou COW*) de stations de travail et les méta-systèmes. L'engouement suscité aujourd'hui par les grilles est dû à plusieurs facteurs, notamment à : l'émergence et la popularité grandissante d'Internet et des technologies associées ; et l'évolution technologique et économique des ressources informatiques en termes de puissance de calcul, de capacité de stockage, de bande passante, et de prix de plus en plus abordables. Cet engouement a déjà conduit à la prolifération de projets sur les grilles avec différentes incarnations traduisant différentes approches : méta-calcul ou *metacomputing*, super-calcul virtuel ou *virtual supercomputing*, calcul global ou *global/Internet computing*, systèmes pair-à-pair ou *peer-to-peer computing*, etc.

Historiquement, l'idée des grilles est apparue en 1995 avec le projet I-WAY [DFP+96] visant à construire un méta-système multi-site et une infrastructure logicielle permettant son exploitation pour le traitement d'applications haute-performance. Le méta-système inter-connecte des périphériques de visualisation (caves de réalité virtuelle immersive et instruments d'observation) et des super-calculateurs répartis sur 17 sites. L'idée a été étendue à tous types de ressources informatiques incluant les applications, le stockage et les masses de données. Cette extension se retrouve dans la définition donnée dans [FKT01], selon laquelle la grille vise le partage coordonné de ressources et la résolution de problèmes dans des organisations virtuelles dynamiques et multi-institutionnelles. Suivant les objectifs visés par l'exploitation de ressources réparties, plusieurs autres définitions ont été proposées (e.g. [FKT01, KBM02]), ce qui justifie d'ailleurs la publication d'articles tels que [Fos02, Gri02, BLDGS03] dédiés à l'analyse de ces définitions dans le but de clarifier le concept de grille. Il nous semble donc important de préciser le contexte dans lequel on se place lorsqu'on aborde des travaux sur grilles informatiques.

Dans le cadre de nos travaux sur l'optimisation combinatoire parallèle sur grilles, nous considérons une grille comme un ensemble de ressources de calcul réparties ayant les caractéristiques présentées dans [BBL02] : multi-domaine d'administration, hétérogène, extensible et dynamique (à disponibilité spatio-temporelle variable).

- *La grille est multi-domaine d'administration* : les ressources sont réparties sur plusieurs domaines d'administration et gérées par différentes organisations. Les utilisateurs et fournisseurs de ressources sont clairement identifiés atténuant ainsi les problèmes de sécurité. Néanmoins, la traversée de pare-feu (ou *firewalls*) demeure un problème crucial qu'il convient de résoudre. Dans les systèmes de calcul global tels que XtremWeb [Fed03] basés sur le vol de cycles étendu à Internet, ce problème est

résolu de manière naturelle puisque la communication est initiée par les machines volontaires "de l'intérieur d'un domaine d'administration". La résolution du problème n'est pas toujours aussi simple, et cela constitue souvent le défi des systèmes de *metacomputing*. Par exemple, pour Condor [LLM88], différentes approches ont été envisagées pour traiter le problème : *flocking*, *Condor-G* et *Condor Glide-in* [TTL02]. La première solution consiste à mettre en place un gestionnaire de ressources par domaine d'administration et de permettre à chacun de soumettre à d'autres gestionnaires les travaux qu'il ne peut traiter. Les deux autres approches sont basées sur le couplage avec Globus [FK97].

- *La grille est hétérogène* : l'hétérogénéité des ressources matérielles et logicielles est accentuée par le nombre important de machines dans la grille et appartenant à différentes organisations. La résolution de ce problème en utilisant les standards d'échange récents tels que XML et SOAP constitue, entre autres, un facteur ayant fortement encouragé l'émergence des grilles. Néanmoins, l'hétérogénéité rend particulièrement plus difficile l'évaluation de performances des applications déployées, et constitue un thème de recherche à part entière [GWB+04, NGB04]. Plusieurs colloques sont dédiés à ce thème, en particulier *Automatic Performance Analysis Tools and Performance Tools for the GRID* (*APART Workshop*).

- *La grille est extensible* : la grille a une échelle importante en termes du nombre de machines potentiellement disponibles et de la taille du réseau d'interconnexion de ces machines. Suivant l'échelle visée, on distingue souvent les systèmes de calcul global et pair-à-pair des grilles de calcul. Les premiers supportent pour le moment bien plus de machines que les grilles de calcul. Ils sont destinés en particulier aux applications basées sur le volontariat de milliers voire de millions de machines d'Internet. Le plus représentatif et populaire des intergiciels gérant ce type de systèmes *de calcul* est SETI@home [ACK+02]. XtremWeb [Fed03] en fait également partie mais, à la différence de SETI@home, il n'est pas dédié à un seul projet. Les applications gourmandes en ressources doivent être conçues de manière à supporter le passage à l'échelle. Il faut prendre en compte notamment les délais de communication importants.

- *La grille est dynamique* : la disponibilité variable des ressources due à leurs pannes ou aux départs/retours de leurs propriétaires n'est pas une exception mais une règle dans la grille. En effet, avec un nombre si important de ressources, la probabilité de "disparition" des ressources est plus importante. Cette volatilité des ressources pose des problématiques de découverte dynamique de ressources, de tolérance aux pannes, de sauvegarde/restauration des données, de synchronisation, etc. Ces problématiques sont souvent difficiles à gérer de manière transparente et efficace dans les intergiciels. C'est pourquoi, leur prise en compte est parfois nécessaire au niveau applicatif.

Le développement d'applications sur grilles de calcul nécessite la prise en compte des caractéristiques ci-dessus et les problèmes associés. Afin de faciliter l'accès aux grilles, ces problèmes sont en général gérés au sein d'intergiciels masquant aux utilisateurs la difficulté inhérente à leur gestion. Ces intergiciels permettent différents types d'utilisation de la grille

identifiés dans [FK99, KBM02, BLDGS03]. Il s'agit principalement du super-calcul virtuel distribué à grande échelle (ou *distributed virtual supercomputing*, du calcul "haut débit" (ou *high-throughput computing*), du traitement intensif de masses de données (e.g. *data mining*), du traitement à la demande (ou *on-demand computing*) permettant l'accès à des ressources distantes, du travail coopératif, et du multimédia (*QoS*, Vidéo, etc.). Les deux premiers types d'utilisation permettent de réduire le temps d'exécution respectivement d'une seule application et d'un ensemble (ou *pool*) d'applications. Le traitement intensif de données a pour objectif de faire la synthèse ou la découverte de connaissances à partir de référentiels et bases de données et de librairies numériques. Le projet européen *DataGrid* [HJMS+00] est le plus représentatif de grilles dédiées à ce type d'applications. Les trois derniers types d'utilisations permettent l'accès à des services sur la grille tels que l'exploitation d'une ressource distante matérielle ou logicielle, l'interaction entre utilisateurs présents sur la grille, ou l'utilisation d'un support multimédia tel que la vidéo-conférence.

Dans le cadre de l'optimisation combinatoire parallèle, la plupart des travaux se focalisent sur le super-calcul viruel distribué et le calcul "haut débit". Il s'agit souvent d'implémentations parallèles d'algorithmes d'optimisation en utilisant un intergiciel grille. Par exemple, dans [AN04] on présente une étude comparative des implémentations de métaheuristiques parallèles en utilisant différents types d'intergiciels notamment Globus [FK97], Ninf [SNS+], Condor [LLM88], Legion [GW96]. Il ressort de cette étude que l'utilisation de Globus et Condor s'avère plus performante. Dans [AO05, GKYL00, IF00], les auteurs présentent des implémentations parallèles gridifiées d'un algorithme *Branch-and-Bound*. Dans [AO05], les auteurs considèrent une grille composée de 348 PC appartenant à 4 grappes réparties sur le territoire japonais. L'implémentation, basée sur le modèle *Fermier-Travailleurs* hiérarchique, est réalisée en utilisant Ninf-G [TNS+03]. Les communications intra-grappe et inter-grappes sont assurées respectivement par Ninf [SNS+] (appels RPC) et Ninf-G (appels GridRPC [SNM+02]). Le couplage de Ninf avec Globus permet la gestion de la sécurité inter-domaines d'administration. Les résultats expérimentaux obtenus montrent que le déploiement sur grilles d'applications de granularité fine (moins d'une seconde d'exécution) peut améliorer grandement l'efficacité de l'exécution. Dans [GKYL00], les auteurs présentent une étude de cas réalisée dans le cadre du projet MetaNEOS [The00]. Il s'agit d'un algorithme B&B appliqué au problème de l'affectation quadratique (ou *QAP*) sur une grille de 502 PC composée de plusieurs grappes réparties aux États-Unis. L'implémentation a été réalisée avec Condor [LLM88] et MW [LKGY00] en utilisant les techniques de *flocking* et *glidding* [TTL02] pour l'ordonnancement inter-domaines d'administration.

Dans le même esprit, nous proposons au chapitre 2 une plate-forme d'optimisation parallèle hybride couplée avec Condor-MW utilisant le *flocking* pour gérer plusieurs domaines d'administration. Une étude d'un deuxième niveau de couplage avec Globus pour répondre au même objectif mais avec en plus la gestion de la sécurité est une des perspectives de nos travaux. Dans [IF00], le travail présenté considère une grille ayant les caractéristiques présentées ci-dessus en mettant l'accent sur la disponibilité variable et la fréquence des pannes des ressources, et l'extensibilité. Un mécanisme de tolérance aux pannes distribué à grande échelle est présenté. Le simulateur Parsec [BMT+98] a été utilisé pour expérimenter ce mécanisme sur un algorithme B&B en utilisant 100 processeurs. Les résultats obtenus démontrent la performance de l'algorithme en termes d'efficacité d'exécution et de coûts de communication et de stockage. Cependant, l'échelle utilisée reste insuffisante

pour juger de la réelle performance du mécanisme proposé. En outre, la description des unités de travail est basée sur un codage de l'arbre exploré. Ce codage ne permettrait pas le passage à l'échelle. Nous proposerons dans le chapitre 3 une gridification de l'algorithme B&B basé sur codage plus efficace et plus extensible. L'algorithme B&B a été déployé et expérimenté sur une grille pouvant contenir jusqu'à 1245 processeurs.

Ces différentes implémentations, basées sur des intergiciels offrant les services nécessaires à l'exploitation de la grille, démontrent que la gridification des algorithmes d'optimisation combinatoire est grandement facilitée par ces intergiciels. Cependant, leur utilisation exige la connaissance des concepts du parallélisme notamment sur grilles de calcul (ordonnancement, tolérance aux pannes, granularité, etc.) souvent non maîtrisés par les utilisateurs non avertis. Il est donc nécessaire de disposer d'un intergiciel supplémentaire entre l'utilisateur (l'application) et l'intergiciel grille. Le rôle de ce nouvel intergiciel dédié est d'assurer le développement et le déploiement de composants d'optimisation combinatoire parallèle sur la grille en masquant à l'utilisateur l'intergiciel permettant l'exploitation de celle-ci.

Très peu d'intergiciels dédiés à l'optimisation combinatoire ont été proposés dans la littérature [ACE+02, WA04]. Le projet européen DREAM [ACE+02] propose une plate-forme éponyme permettant l'utilisation du modèle insulaire pour l'implémentation parallèle pair-à-pair des algorithmes évolutionnaires. La plate-forme est destinée à différents types d'utilisateurs allant des novices aux experts en programmation et en optimisation combinatoire et parallèle. Elle est basée sur un intergiciel de calcul global pair-à-pair *ad hoc* développé en Java. HeuristicLab Grid [WA04] est un intergiciel similaire à SETI@home dédié à la résolution de problèmes d'optimisation combinatoire. Basée sur les services Web, cette plate-forme encapsule des algorithmes d'optimisation et la représentation des problèmes à résoudre déployés sur des agents travailleurs. Ces derniers exécutent, à la demande des utilisateurs clients, ces algorithmes (sous forme de *plug-ins*) en fonction des paramètres indiqués. Les résultats sont renvoyés à un serveur de base de données et sont mis à la disposition des utilisateurs propriétaires. Un couplage de la bibliothèque BOB++ [1] de méthodes exactes avec Athapascan/Inuktitut [2] a été proposé [3]. L'expérimentation de ce couplage sur le problème d'affectation quadratique réalisée sur le réseau eToile [PGS02] et sur le ICluster de Grenoble a démontré l'efficacité de ce couplage à traiter des instances de grande taille [ea03].

Dans le cadre de thèse de S. Cahon, notre objectif a été de proposer une plate-forme d'aide à la conception et réalisation de métaheuristiques parallèles hybrides sur grilles. Cette plate-forme permet l'utilisation transparente du parallélisme sur environnements dédiés. Un couplage avec Condor-MW et la mise en place d'un mécanisme de sauvegarde/restauration a permis son extension aux environnements non dédiés à grande échelle. La thèse de M. Mezmaz porte sur le parallélisme à grande échelle des méthodes exactes (notamment B&B) et leur hybridation avec les métaheuristiques. L'exploitation des grilles est basée sur l'utilisation de XtremWeb avec intégration de la coopération entre pairs et de mécanismes de sauvegarde/restauration (ou *checkpointing*).

[1] http ://www.prism.uvsq.fr/~blec/Research/BOBO/index.html
[2] http ://www-id.imag.fr/Logiciels/inuktitut/
[3] http ://www-id.imag.fr/~revire/HtmlA1DocG/

2.2 Métaheuristiques parallèles

Dans cette étude, nous analyserons les modèles parallèles en considérant les métaheuristiques à solution unique évoquées dans la taxinomie présentée précédemment. Par contre, nous considérerons seulement les algorithmes évolutionnaires (AE) et en particulier les algorithmes génétiques (AG) dans la classe des métaheuristiques à population de solutions. Néanmoins, le principe de fonctionnement des modèles reste valable pour les autres métaheuristiques à population de solutions.

2.2.1 Modèles parallèles pour les métaheuristiques à population de solutions

Les AE [Hol75] sont des techniques de recherche stochastique ayant été appliquées avec succès à différents problèmes réels et complexes. Leur principe de fonctionnement (Algorithme 2.2.1) est basé sur l'application itérative d'opérateurs stochastiques à une population de solutions (ou individus). A chaque génération du processus d'évolution, des individus de la population sont sélectionnés (*politique de sélection*) et recombinés (*opérateurs de variation* tels que le *croisement* ou *crossover* et la *mutation*) de manière à générer de nouvelles solutions évaluées pouvant en remplacer d'autres (*stratégie de remplacement*). Le processus d'évolution s'arrête lorsqu'un critère est avéré (*Critère d'arrêt*).

Algorithme 2.2.1. Pseudo-code d'un AE.

Générer($P(0)$) ;
$t := 0$;
Evaluer($P(t)$) ;
Tant que Terminaison_non_avérée($P(t)$) **Faire**
  $P'(t)$:= Sélectionner($P(t)$) ;
  $P'(t)$:= Appliquer_op_variation($P'(t)$) ;
  Evaluer($P(t)$) ;
  $P(t+1)$:= Remplacer($P(t)$, $P'(t)$) ;
  $t := t+1$;
Fin TantQue

La population initiale est souvent générée de manière *aléatoire ou gloutonne*. La sélection peut être *aléatoire* ou *élitiste* utilisant une politique basée sur le *tournoi*, la *roulette*, la *sélection par rang*, etc. Le tournoi consiste à tirer uniformément T individus de la population et en sélectionner le meilleur. La roulette consiste à donner à chaque individu une probabilité d'être sélectionné proportionnelle à sa performance (ou *fitness*). La sélection par rang est une roulette dans laquelle la performance d'un individu est mesurée non pas par sa qualité mais par son rang dans la population (compris entre 1 et la taille de la population). Trois stratégies de remplacement sont souvent utilisées : *plus*, *générationnelle* et *stationnaire* ou *steady-state*. La stratégie *plus* consiste à fusionner les individus parents

avec les enfants issus de leur reproduction, et éliminer de manière aléatoire ou élitiste une partie suffisante de la population résultante pour la ramener à la taille constante fixée. Dans l'approche générationnelle, les parents sont tous remplacés par les enfants générés. Dans le schéma stationnaire, un individu est sélectionné, généralement par tournoi, un second si le croisement doit être appliqué, et l'enfant résultant (après croisement et mutation éventuels) est réinséré dans la population en remplacement d'un parent sélectionné par tournoi inversé (le moins performant est retenu). Le critère d'arrêt peut être simplement un nombre fixé de générations, ou basé sur la progression de la qualité des solutions pendant un certain nombre de générations. Par exemple, le processus d'évolution s'arrête si aucune progression de la qualité des solutions n'est avérée après un seuil fixé de générations.

Dans le contexte multi-objectif, la structure de l'algorithme reste inchangée. Par contre, des adaptations sont nécessaires dans les phases d'évaluation et de sélection. L'étape d'évaluation comprend en plus du calcul des valeurs associées aux différents objectifs l'obtention à partir de celles-ci d'une valeur d'efficacité. Une fonction (*e.g. d'agrégation*) de transformation du vecteur de fonctions objectifs en une valeur scalaire doit être définie. Cette dernière est utilisée dans les autres phases de l'algorithme, notamment la sélection (*e.g. ranking*). Le processus de sélection est basé sur deux mécanismes appelés *élitisme* et *partage* ou *sharing*. Ces derniers permettent respectivement de converger vers le meilleur front Pareto [OTT98, PM98, Meu02] et de maintenir une diversité des individus dans la population et dans le front Pareto. L'élitisme consiste à maintenir une population (appelée *archive Pareto*) autre que la population courante en archivant toutes les solutions Pareto optimales générées pendant le processus d'évolution. Cette archive, mise à jour à chaque itération, est utilisée dans le processus de sélection. En effet, différentes stratégies de sélection peuvent être envisagées selon que les individus sont choisis à partir de la population courante, à partir de l'archive ou à partir des deux. L'opérateur de partage maintient la diversité en considérant le degré de similitude d'un individu vis à vis des autres individus, souvent basé sur une distance à définir dans l'espace objectif.

Pour une conception et une mise en oeuvre réutilisable d'un AE, on doit distinguer deux parties : une *partie invariante* et une *partie spécifique au problème*. Cette dernière comprend principalement le codage des solutions, la définition de la fonction d'évaluation ou objectif, et la spécification des opérateurs de variation notamment les opérateurs de recombinaison et de mutation. Par contre, la partie invariante est complètement indépendante du problème traité. Elle comporte essentiellement la stratégie de sélection des individus (la roulette, le rang, le tournoi, etc.), la procédure de remplacement (générationnelle, élitiste, par tournoi, etc.), et enfin le critère de décision d'arrêt/continuation (l'atteinte d'un nombre prédéfini d'itérations, la notification de convergence exprimée par un nombre d'itérations sans amélioration, etc.).

L'information définissant la progression du processus d'évolution à une itération donnée se résume à la génération courante (une population de solutions) ainsi qu'à l'état du critère de décision d'arrêt/continuation. Par exemple, il peut s'agir pour ce dernier d'un compteur d'itérations contrôlant l'avancement de la recherche. Dans le cas multi-objectif, il faut ajouter l'archive qui stocke le front Pareto courant.

On distingue trois modèles parallèles pour les métaheuristiques à population de solutions : le *modèle insulaire*, le *modèle d'évaluation parallèle de la population* et le *modèle d'évaluation parallèle d'une solution.*

2.2.1.1 Le modèle insulaire

Dans le modèle insulaire [CHMR87], inspiré de certains comportements observés dans les niches écologiques, plusieurs AE sont déployés pour faire évoluer simultanément différentes populations (îles) de solutions. Dans la suite de ce document, on confondra parfois une île avec son AE ou sa population. Les AE peuvent être *homogènes ou hétérogènes* selon qu'ils utilisent des paramètres (opérateurs de variation et autres) identiques ou différents. Comme le montre la figure 2.2, ces AE échangent de manière *synchrone ou asynchrone* du matériel génétique. Cet échange a pour objectif de diversifier l'espace de recherche et de retarder la convergence du processus d'évolution. Cela permet l'obtention de solutions de meilleure qualité et une exécution *plus robuste* i.e. minimisant la déviation en terme de qualité d'une exécution à une autre.

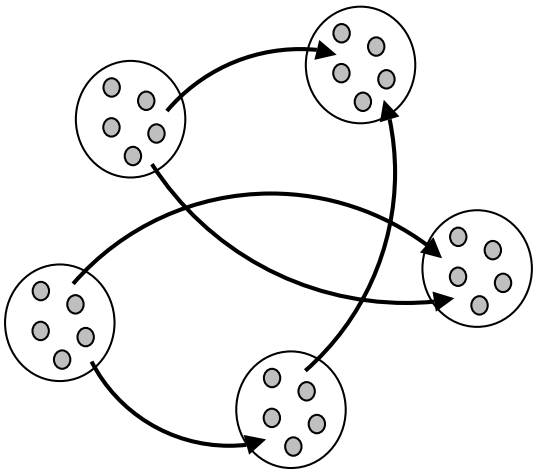

FIG. 2.2 – Le modèle coopératif insulaire d'AE.

Pour chacune des populations, le processus de gestion des migrations synchrones ou asynchrones intervient au terme de chaque génération de l'AE standard, succédant à la phase de remplacement. La politique de migration d'individus entre îles est définie par les paramètres suivants : le critère de décision de migration, la topologie des échanges d'individus, le nombre d'émigrants intervenant dans une opération de migration et la stratégie de leur sélection, la politique d'intégration des immigrants. Tous ces paramètres sont indépendants du problème traité, ce qui rend le modèle générique.

- **Critère de décision de migration** : La migration d'individus d'une île vers une autre peut être décidée suivant un critère aveugle ou intelligent. La migration aveugle peut être périodique ou probabiliste. La première intervient sur chaque AE après un nombre de générations fixé par le programmeur (fréquence de migration). La seconde consiste à effectuer la migration après chaque génération avec une probabilité défi-

nie par l'utilisateur. A l'inverse, la migration intelligente est dirigée par des critères d'amélioration de la qualité des solutions. Un seuil d'amélioration de cette qualité doit être fourni, et si l'amélioration entre deux générations est inférieure à ce seuil, une opération de migration est déclenchée. Ces différents paramètres (fréquence, probabilité, seuil de pression) doivent être fixés en fonction de la puissance des machines pour prendre en compte le caractère hétérogène de la grille.

- **La topologie des échanges d'individus** : La topologie indique pour chaque île ses voisins au regard de la migration i.e. l'île (ou les îles) de destination/provenance de ses émigrants/immigrants. De nombreux travaux ont été menés pour étudier l'impact de la topologie sur la qualité des résultats obtenus [CHMR87, SWM91]. Il ressort de ces études que les graphes cycliques sont préférables, les modèles en anneau et hypercube sont d'ailleurs largement utilisés. Il faut noter qu'un nombre trop élevé de jonctions entre îles se révèle inefficace, l'ensemble des populations distribuées se comportant alors comme une seule population globale. Sur une grille de calcul (caractère volatile), notons que la disparition d'une île requiert une reconfiguration dynamique de la topologie. Cette reconfiguration peut s'avérer très coûteuse et rendre le processus de migration inefficace. Belding [Bel95] a expérimenté une coopération d'AE sans topologie prédéfinie, signifiant que la population cible de chaque migration est choisie de manière aléatoire. Les résultats ont montré que ce type de topologie contribue à améliorer de manière significative la robustesse. Une topologie aléatoire est donc envisageable dans le contexte des grilles de calcul. Celle-ci est utilisée dans nos travaux présentés au chapitre 3, les résultats expérimentaux obtenus confirment le constat de Belding.

- **Le nombre d'émigrants** : Ce paramètre peut être défini comme un nombre fixé ou variable d'individus, ou comme un pourcentage d'individus de la population et/ou de l'archive Pareto (dans le contexte multi-objectif). Le choix de ce paramètre est crucial. En effet, s'il est trop faible les îles auront tendance à évoluer de manière indépendante et la migration aura moins d'impact sur le retard de la convergence et donc sur la qualité des solutions obtenues. Si le nombre d'émigrants est trop élevé le coût de communication sera plus important surtout sur une grille, et les îles auront tendance à converger vers les mêmes solutions. Un compromis est donc à trouver pour allier à la fois l'exploration et l'intensification de la recherche.

- **Politique de sélection des émigrants** : La politique de sélection des émigrants indique pour chaque île de manière *élitiste* ou *aléatoire* les individus à migrer. La stratégie aléatoire ne garantit pas la sélection des meilleurs individus, mais elle a un coût de calcul plus faible. La stratégie élitiste, souvent basée sur la roulette, le rang, le tournoi ou l'échantillonnage uniforme, tente de sélectionner les meilleurs individus de la population. Dans le contexte multi-objectif, les individus peuvent être sélectionnés de la population courante, de l'archive Pareto ou des deux.

- **Politique de remplacement/intégration des immigrants** : De manière symétrique, la politique de remplacement/intégration des immigrants indique de manière

élitiste ou aléatoire les individus de la population de destination à remplacer par les nouveaux arrivants. Dans le modèle insulaire multi-objectif, plusieurs politiques peuvent être envisagées. Par exemple, les solutions Pareto immigrantes remplacent de manière aléatoire les individus de la population n'appartenant pas au front Pareto local. On peut également faire un classement de tous les individus de la population en fronts Pareto en utilisant la relation de dominance. Le plus mauvais front est remplacé par les solutions immigrantes. Une autre solution consiste à fusionner le front Pareto immigrant avec le front local pour constituer la nouvelle archive locale.

L'implémentation du modèle insulaire peut être *asynchrone* ou *synchrone*. Le modèle asynchrone associe à chaque algorithme insulaire un critère de décision d'immigration. Il est réévalué au terme de chaque itération à partir de l'état de la population courante et/ou de l'archive Pareto. S'il est avéré, des requêtes sont émises vers les populations voisines, lesquelles les traiteront dans un temps indéterminé. La réception et intégration de nouvelles solutions interviendra lors des itérations ultérieures. Cependant, en raison de l'hétérogénéité logicielle et/ou matérielle, il se peut que les populations soient à différents stades d'évolution, entraînant le problème de *non-effet* ou du *super individu*. L'arrivée de solutions de mauvaise qualité dans une population à un stade d'évolution avancé n'apporte aucune contribution. Dans la situation opposée, la coopération insulaire se traduit par une convergence prématurée. Le modèle est non bloquant et donc plus performant et plus tolérant aux pannes jusqu'à un certain seuil autorisé de pertes de migrations.

Dans le modèle synchrone, les algorithmes insulaires procèdent à une opération de synchronisation à chaque itération. Cette opération se traduit par un échange d'individus entre les différentes îles. La synchronisation garantit que ces îles sont au même stade d'évolution évitant ainsi les problèmes évoqués ci-dessus. Par contre, elle pose des problèmes de performance en terme de temps de calcul sur les grilles en raison de leur hétérogénéité. En effet, le processus d'évolution est souvent suspendu sur les machines les plus puissantes en attente des machines les moins puissantes. En outre, le modèle synchrone est non tolérant aux pannes puisque la perte d'une île entraîne le blocage du modèle. Ceci le rend difficile à exploiter dans un environnement volatile. En résumé, le modèle synchrone est plus complexe à mettre en oeuvre sur grilles de calcul de manière performante.

Les données (ou la mémoire) du modèle coopératif insulaire nécessaires au mécanisme de sauvegarde/restauration comprennent (comprend) uniquement les individus en cours de migration d'une population vers une autre. On note que, dans le mode asynchrone, cette information n'est pas critique à l'exécution du modèle parallèle puisque les AE sont des processus stochastiques.

2.2.1.2 Le modèle d'évaluation parallèle de la population

L'évaluation de la population est la phase la plus coûteuse en temps CPU d'un AE, en particulier lorsqu'il s'agit de résoudre des problèmes réels. Succédant à la phase de transformation (par application des opérateurs de variation), l'évaluation consiste à déterminer la qualité (ou *fitness*) de chaque nouvelle solution produite. Les différentes évaluations étant indépendantes les unes des autres, leur parallélisation est donc naturelle et est basée sur le partitionnement des individus suivant le modèle *fermier-travailleurs* (Fig. 2.3).

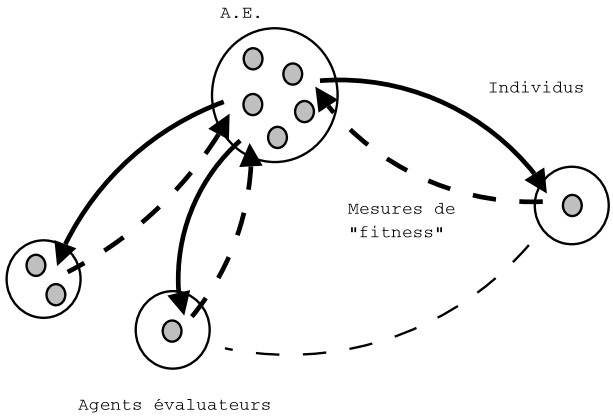

FIG. 2.3 – Illustration du modèle d'évaluation parallèle.

Dans le contexte multi-objectif, l'approche d'implémentation souvent utilisée est celle qui consiste à distribuer les individus de la population entre les différents travailleurs. Chaque travailleur évalue le vecteur de toutes les fonctions objectifs sur les individus reçus. Une autre approche, également envisageable, consiste à répartir les fonctions objectifs entre les travailleurs. Chacun d'entre eux évalue sa propre fonction sur tous les individus de la population. Le fermier se chargera de l'agrégation des différents résultats sur toute la population. Cette approche offre un degré de parallélisme et une extensibilité limités par le nombre d'objectifs considérés dans la fonction multi-objectif, c'est à dire souvent 2 à 3.

Selon l'ordre d'exécution de la phase d'évaluation par rapport au reste de l'AE, le modèle a deux modes : *synchrone* et *asynchrone*. Dans le mode synchrone, le processus *fermier* gère l'évolution de la population, exécute en séquence les phases de sélection, transformation et remplacement. A chaque itération, il distribue l'ensemble des nouvelles solutions générées entre différents agents évaluateurs (travailleurs), et *se met en attente des résultats*. Après collecte des résultats, le processus d'évolution se poursuit à nouveau. Le modèle n'altère en rien le comportement de l'AE original.

L'exécution du modèle synchrone est toujours synchronisée avec le retour de la dernière solution évaluée. Il est donc bloquant sur une grille en raison de l'hétérogénéité de celle-ci. De plus, le modèle n'est pas tolérant aux pannes, aussi la notification de disparition d'un agent évaluateur nécessite une redistribution des solutions lui ayant été affectées vers d'autres agents évaluateurs. Il convient donc de mémoriser l'ensemble des solutions non encore évaluées. Par ailleurs, l'extensibilité de ce modèle est limitée par la taille de la population.

Dans le mode asynchrone, la phase d'évaluation n'est pas synchronisée avec le reste de l'AE. Le processus fermier n'attend pas le retour de toutes les évaluations pour exécuter les phases de sélection, transformation et remplacement. L'AE *Steady-State* est le meilleur exemple illustrant le modèle asynchrone. Il offre divers avantages importants en comparaison du modèle synchrone. Il est naturellement non bloquant et tolérant aux pannes (à un taux de perte raisonnable) car il n'y a pas de contrôle quant au retour des solutions émises pour évaluation. De plus, ce modèle est bien adapté, en terme d'efficacité, à l'ensemble

des applications caractérisées par un coût d'évaluation irrégulier. En outre, il n'est pas nécessaire de mémoriser les solutions traitées par les processus évaluateurs, ce qui réduit le taux d'occupation de la mémoire. Enfin, l'extensibilité de ce modèle n'est pas limitée par la taille de la population.

Par ailleurs, le modèle d'évaluation parallèle (a)synchrone, étant indépendant du problème traité, sa conception est générique. Cependant, son efficacité à l'exécution dépend fortement de la granularité des évaluations i.e. le nombre de solutions qu'un travailleur doit évaluer à chaque envoi. Les délais de communication étant importants, le problème de granularité de parallélisme est accentué sur une grille (grande échelle). Le choix de la taille du grain i.e. du nombre pertinent de solutions affectées à chaque travailleur est un paramètre de performance crucial pour le modèle.

2.2.1.3 Le modèle d'évaluation parallèle d'une solution

L'évaluation de chaque solution de la population est effectuée de manière parallèle centralisée. Dans ce modèle de type *Fermier-Travailleurs*, une même solution est répliquée sur les différents sites évaluateurs (travailleurs). Les valeurs de qualité partielles retournées par ces travailleurs seront ensuite agrégées par le fermier.

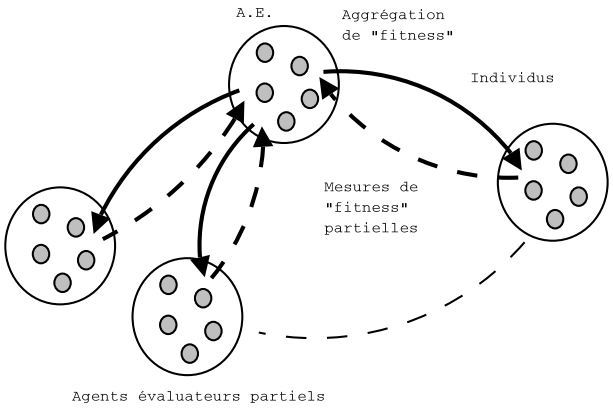

FIG. 2.4 – L'évaluation parallèle d'une solution unique.

Ce modèle est nécessaire à la résolution de problèmes réels dont la fonction objectif requiert l'accès à des bases de données volumineuses ne pouvant être manipulées sur un seul site. A cause de cette contrainte matérielle, elles sont distribuées entre différents sites. Le parallélisme de données est exploité lors de l'évaluation de la fonction objectif. La parallélisation de la fonction objectif est également intéressante lorsque l'évaluation d'une seule solution s'avère coûteuse. Même si son extensibilité est limitée, son exploitation conjointe avec le modèle d'évaluation parallèle en particulier synchrone de la population permet d'étendre le degré de parallélisme de celui-ci et améliorer son extensibilité. En optimisation multi-objectif, toutes les fonctions objectifs peuvent être parallélisées simultanément.

Le modèle nécessite de nouveaux composants spécifiques au problème traité notamment le vecteur de fonctions d'évaluation partielles ainsi qu'une fonction d'agrégation de celles-ci.

L'implémentation de ce modèle étant toujours synchrone, il est indispensable de mémoriser les valeurs de qualité partielles de la solution évaluée pour gérer la tolérance aux pannes et la disponibilité variable des machines sur une grille.

Le modèle d'évaluation parallèle d'une solution est souvent ignoré dans les taxinomies [CP97]. Cependant, son utilisation conjointe avec d'autres modèles améliore de manière siginificative le degré de parallélisme. En effet, considérons l'application de conception de réseaux mobiles sur laquelle nous avons travaillé dans le cadre d'un contrat avec *France Telecom R&D*. Il s'agit de sélectionner parmi un ensemble de sites candidats les sites devant accueillir les stations de base du réseau, y configurer et paramétrer des antennes. Le paramétrage d'antennes doit répondre aux enjeux économiques de l'entreprise. Il faut minimiser le coût de déploiement et de maintenance du réseau en minimisant le nombre de sites, en offrant en même temps la meilleure qualité de service aux clients i.e. en minimisant les interférences entre les communications et en maximisant le trafic écoulé. Sur certaines instances du problème, le degré de parallélisme du modèle d'évaluation parallèle d'une solution (une conception d'un réseau mobile) peut atteindre 20. En considérant un modèle insulaire à 10 îles de 100 individus, la prise en compte du modèle d'évaluation parallèle d'une solution fait passer le nombre d'évaluations pouvant être effectuées simultanément de 1000 à 20000.

2.2.2 Modèles parallèles pour les métaheuristiques à solution unique

Les métaheuristiques à solution unique peuvent être vues comme des marches à travers des voisinages i.e. des trajectoires dans l'espace de recherche du problème traité [CG02]. Ces marches sont conduites par des procédures itératives (Algorithme 2.2.2) permettant de passer d'une solution à une autre dans l'espace de recherche. Nous appellerons *mouvement* l'opération de base permettant de transformer une solution en une autre solution se trouvant dans son voisinage. Par exemple, dans le problème du Voyageur de Commerce le mouvement peut être une permutation de deux arcs du circuit représentant la solution courante. La sélection d'un mouvement acceptable peut être basée sur la génération d'un voisinage partiel ou total, déterministe (e.g. élitiste) ou stochastique.

Algorithme 2.2.2. Pseudo-code d'une recherche locale.

Générer($s(0)$) ;
$t := 0$;
Tant que Terminaison_non_avérée($s(t)$) **Faire**
    m(t) := SélectionnerMouvementAcceptable($s(t)$) ;
    $s(t+1)$:= AppliquerMouvement($m(t), s(t)$) ;
    $t := t+1$;
Fin TantQue

La conception et réalisation d'une métaheuristique à solution unique comporte deux parties : une partie invariante et une partie spécifique au problème traité. Dans la partie

spécifique, il convient de définir le codage d'une solution et le(s) voisinage(s) directement lié(s) au problème traité. Au contraire, les composants énoncés ci-après sont génériques, et diffèrent selon la métaheuristique utilisée : le choix de la solution initiale, l'ensemble des données mémorisées au cours de l'exploration (meilleure solution courante, liste de mouvements tabous, divers paramètres guidant la trajectoire, etc.), la stratégie de génération du voisinage candidat (partiel ou complet, déterministe ou non, etc.), la stratégie de sélection d'une solution voisine (élitiste ou stochastique), et enfin le critère d'arrêt (identification d'un optima local, nombre d'itérations fixé atteint, etc.).

La configuration des différents composants génériques ou spécifiques ont une grande influence sur l'efficacité des métaheuristiques à solution unique en termes de temps d'exécution et de qualité de la solution produite. Un autre moyen d'améliorer cette efficacité est d'utiliser le parallélisme. Il existe trois modèles parallèles principaux pour les métaheuristiques à solution unique : le *modèle parallèle multi-départ*, le *modèle d'évaluation parallèle du voisinage* et le *modèle d'évaluation parallèle d'un mouvement*.

2.2.2.1 Le modèle parallèle multi-départ

Le modèle parallèle multi-départ consiste à déployer simultanément plusieurs marches afin d'obtenir des solutions meilleures et plus robustes. Les algorithmes à base de recherche locale déployés peuvent être de nature *homogènes ou hétérogènes* (démarrant d'une même solution ou non, associés aux mêmes paramètres ou non guidant la trajectoire.), *indépendants ou coopératifs* (Fig. 2.5). Les marches indépendantes n'échangent pas d'information durant leur exécution, les résultats sont donc identiques à ceux que l'on obtiendrait par de mêmes algorithmes séquentiellement exécutés. Dans ce contexte particulier, le parallélisme est trivial, facile à implémenter, et totalement indépendant du problème traité. Cela justifie les nombreuses implémentations proposées et expérimentées dans la littérature.

Dans le modèle multi-départ parallèle coopératif, des informations sont échangées entre les différentes méthodes de recherche locale. Contrairement au modèle multi-départ indépendant, l'implémentation est généralement spécifique au problème. En effet, la nature des informations échangées ne peut être définie de manière générique. Ces informations peuvent être les bonnes solutions visitées, des mouvements effectués et/ou d'autres paramètres exprimant la trajectoire accomplie dans l'espace de recherche, etc.

La communication peut être mise en oeuvre par le biais d'un processeur central contrôlant les échanges entre les différentes méthodes, ou selon un modèle totalement décentralisé de type pair à pair. Le choix de la topologie adoptée sera déterminante à l'extensibilité du modèle déployé. L'échange centralisé se montre généralement moins extensible que celui totalement distribué. D'autre part, le mode de communication peut-être synchrone ou non. Le premier se distingue par une fiabilité accrue mais une efficacité moindre à l'exécution.

2.2.2.2 Le modèle d'évaluation parallèle du voisinage

Le modèle d'évaluation parallèle du voisinage est un modèle de type *Fermier-Travailleurs* (Fig. 2.6) qui n'altère en rien le comportement de l'heuristique parallélisée. Il se caractérise

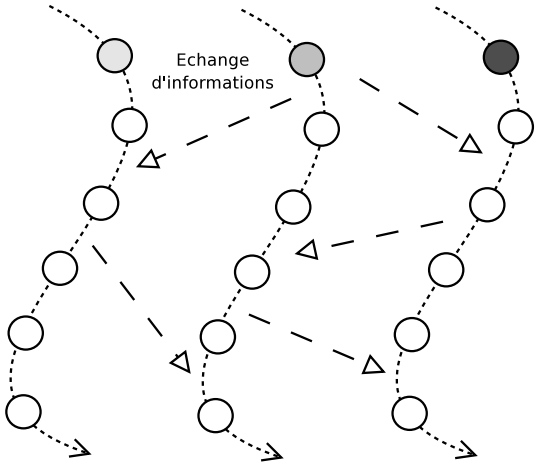

FIG. 2.5 – Coopération de recherches locales concurrentes.

principalement par une distribution des calculs, une même recherche séquentielle produirait les mêmes résultats. Au début de chaque itération, le fermier duplique la solution courante qui sera émise vers différents noeuds travailleurs distribués, lesquels considéreront un voisinage partiel de solutions candidates. Le fermier se met en attente des résultats avant de poursuivre sa marche.

L'implémentation parallèle de ce modèle peut être réalisée de manière indépendante. Il convient cependant de définir les mécanismes afin de partitionner le voisinage d'une solution. Ce modèle de parallélisation de *bas-niveau* se montre efficace si le temps d'évaluation d'une solution voisine est important et/ou le nombre de solutions voisines candidates à considérer est élevé. Selon la stratégie de sélection du prochain voisin à évaluer (*e.g.* le choix du premier meilleur voisin), l'efficacité du modèle n'est pas toujours avérée. En effet, le nombre de solutions voisines à parcourir avant d'en générer une améliorant la fonction objectif se montre très variable.

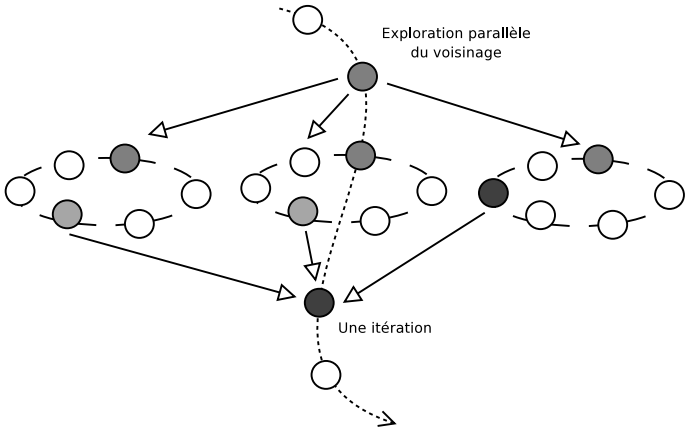

FIG. 2.6 – La décomposition et l'exploration parallèle du voisinage d'une solution.

Dans le contexte multi-objectif, l'exploration parallèle du voisinage est souvent appliquée à un front Pareto entier. Dans ce cas, le modèle d'exploration parallèle du voisinage est combiné avec le modèle multi-départ. Le modèle hiérarchique obtenu consiste à appliquer simultanément une exploration parallèle du voisinage à chaque solution du front Pareto. Ce modèle s'est révélé efficace dans nos travaux sur les applications multi-objectifs de conception de réseaux mobiles et d'ordonnancement de type Flow-Shop.

2.2.2.3 Le modèle d'évaluation parallèle d'un mouvement

Le principe du modèle est analogue à celui de l'évaluation parallèle de la fonction objectif décrite précédemment dans le cadre des AE. Ce modèle s'avère particulièrement intéressant lorsque l'évaluation incrémentale d'un seul mouvement est coûteuse.

La mise en oeuvre de ce modèle nécessite l'implémentation d'un ensemble de nouvelles fonctions d'évaluation partielles du mouvement appliqué à une solution donnée. Il convient également de définir un opérateur d'agrégation de telles valeurs de qualité.

Ce modèle est caractérisé par sa granularité très fine. C'est pourquoi la plupart de ses implémentations ont été réalisées sur architectures parallèles à mémoire partagée [KR87].

2.3 Méthodes exactes parallèles

Les méthodes de résolution exacte utilisées en optimisation combinatoire sont en général basées sur le parcours arborescent de type *Branch-and-X* (X pouvant être *Bound*, *Cut* ou *Price*). La racine de l'arbre désigne le problème à traiter, les noeuds intermédiaires représentent des sous-problèmes obtenus par ajout de contraintes aux parents, et les feuilles contiennent des solutions réalisables. L'exploration de l'arbre étant de complexité exponentielle, une technique d'élagage (le X du *Branch-and-X*) est utilisée pour réduire la taille de l'espace de recherche. Dans la suite de cette section, nous restreindrons notre étude aux méthodes de type *Branch-and-Bound* (ou $B\&B$). Dans ce cas, la technique d'élagage est basée sur l'utilisation de bornes (*Bounds*) (inférieure et supérieure) correspondant à une estimation de la valeur de la meilleure solution issue de la branche parcourue. En plus de la fonction objectif f, les méthodes $B\&B$ utilisent une autre fonction g de calcul de la borne inférieure de la fonction objectif.

Le principe d'un algorithme $B\&B$, illustré par la figure 2.3 ([Cla99]), est simple. Il s'agit d'explorer l'espace de recherche en construisant dynamiquement un arbre constitué initialement d'un seul noeud (racine) désignant le problème à traiter. Deux variables majeures sont utilisées : la meilleure solution courante trouvée (*meil_sol*) et une liste $\mathcal{P}$ de sous-problèmes à traiter. La valeur initiale de *meil_sol* est générée par une heuristique si cela est possible, sinon elle est fixée à ∞. Initialement, cette liste contient le problème à traiter et la borne inférieure associée.

Algorithme 2.3. Pseudo-code d'un algorithme $B\&B$.

Initialisation : $meil\_sol := \infty$; $BI(P_0) := g(P_0)$; $\mathcal{P} := \{(P_0, BI(P_0))\}$;
Tant que $\mathcal{P} \neq \emptyset$ **Faire**
Sélection : sélectionner un noeud P de $\mathcal{P}$; $\mathcal{P} := \mathcal{P} \setminus \{P\}$;
Séparation : décomposer P en P_1, P_2, ..., P_k ;
Pour $i := 1$ à k **Faire**
Evaluation : $BI(P_i) := g(P_i)$;
Si ($BI(P_i) = f(X)$, X réalisable) et ($f(X) <$ $meil\_sol$) **Alors**
$meil\_sol := f(X)$; $Solution := X$;
Sinon Si $BI(P_i) \geq meil\_sol$ **Alors**
Elagage : Elaguer P_i ;
Sinon $\mathcal{P} := \mathcal{P} \bigcup \{(P_i, BI(P_i))\}$
Fin Faire
Fin TantQue
Résultat : $SolutionOptimale := Solution$; $ValeurOptimale :=$ $meil\_sol$.

A chaque itération de l'algorithme, un sous-problème est sélectionné de la liste $\mathcal{P}$ suivant une *stratégie de sélection*. Une *opération de séparation* (ou *branching*) est alors appliquée (par ajout de contraintes) au sous-problème sélectionné pour produire de nouveaux sous-problèmes. Autrement dit, le sous-espace d'exploration associé au sous-problème est subdivisé en sous-espaces de taille réduite. Pour chaque sous-problème P_i (ou noeud) généré, on évalue la borne inférieure $BI(P_i)$ (*opération d'évaluation ou bounding*). Si le noeud correspond à une solution réalisable X et la borne associée $BI(P_i)$ est la valeur $f(X)$ d'une solution optimale, cette dernière est comparée à la valeur de *meil_sol*. Si elle lui est inférieure alors X sera la nouvelle solution optimale courante et sa valeur sera la nouvelle valeur de *meil_sol*. Si la borne inférieure n'est pas meilleure que la valeur de *meil_sol* (*test d'élagage*) alors le sous-problème P_i est élagué. Dans le cas contraire, le sous-problème est ajouté à la liste avec la borne inférieure associée. Le processus est réitéré tant qu'il y a des sous-problèmes à traiter. La solution optimale correspond à la solution dont *meil_sol* est la valeur de la fonction objectif.

Dans le contexte multi-objectif, les tests d'inégalité (notamment le test d'élagage) deviennent des tests de dominance. La variable *meil_sol* correspond au meilleur front Pareto trouvé. Ce front est mis à jour chaque fois qu'une nouvelle solution Pareto optimale est trouvée. Du point de vue de la généricité, la partie spécifique au problème à traiter est composée du codage des solutions du problème, des fonctions objectifs et de calcul de la borne inférieure, et de l'opérateur de séparation. La conception de l'algorithme et les opérateurs de sélection et d'évaluation constituent la partie invariante de l'algorithme.

Nous présentons ici une taxinomie des modèles parallèles pour les méthodes exactes basées sur les classifications proposées dans [GC94, CDC+94]. Quatre modèles sont identifiés : le *modèle multi-paramétrique parallèle*, le *modèle d'exploration arborescente parallèle*, le *modèle d'évaluation parallèle des bornes* et le *modèle d'évaluation parallèle d'une borne*.

2.3.1 Le modèle multi-paramétrique parallèle

Le modèle multi-paramétrique parallèle (Figure 2.7), très peu étudié dans la littérature, consiste à considérer plusieurs algorithmes $B\&B$. A l'instar des modèles insulaire et multi-départ, il s'agit d'un modèle de grosse granularité. Plusieurs variantes de ce modèle peuvent être envisagées en fixant différemment un ou plusieurs paramètre(s) des algorithmes. Les algorithmes diffèrent seulement par l'opération de séparation dans [MP93]. Ils sont différents uniquement par l'opération de sélection dans [JAM88] où une variante de la stratégie *profondeur d'abord* est utilisée. Chaque algorithme sélectionne de manière aléatoire le prochain noeud à traiter parmi les derniers noeuds générés. Dans [KK84], les algorithmes utilisent chacun une borne supérieure différente dans leurs tests. L'idée est qu'un seul algorithme utilise la meilleure borne supérieure trouvée tant dis que les autres (ϵ-approchés) font usage de cette borne diminuée d'une valeur $\epsilon > 0$. Une autre variante de ce modèle parallèle consiste à décomposer l'intervalle défini par la borne inférieure et la borne supérieure du problème à traiter en sous-intervalles. Chaque sous-intervalle est affecté à un des algorithmes. Dans le contexte multi-objectif, le front Pareto (composé de solutions appelées *solutions supportées*) peut être calculé par un algorithme $B\&B$. Ensuite, plusieurs algorithmes $B\&B$ peuvent coopérer pour calculer les solutions non supportées situées dans le voisinage du front Pareto. Une telle variante a été proposée dans [LDT04].

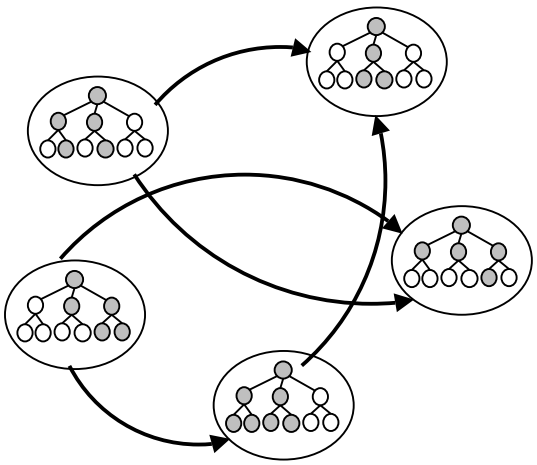

FIG. 2.7 – Illustration du modèle multi-paramétrique.

Le principal avantage du modèle multi-paramétrique parallèle est sa généricité permettant son exploitation de manière transparente à l'utilisateur. Son inconvénient est le surcoût d'exploration qu'il engendre puisque certains noeuds de l'arborescence sont explorés de manière redondante. Toutefois, ce surcoût a une moindre conséquence lorsque le modèle est déployé sur une grille de calcul puisque l'on dispose de suffisamment de ressources. Par ailleurs, le nombre d'algorithmes mis en concurrence n'étant pas important, son exploitation sur une grille ne peut être justifié que s'il est combiné avec d'autres modèles parallèles.

2.3.2 Le modèle d'exploration arborescente parallèle

Le modèle d'exploration arborescente parallèle (Figure 2.8) consiste à parcourir en parallèle différents noeuds racines de sous-arbres définissant des sous-espaces de recherche du problème. Cela signifie que les opérations de séparation, sélection, évaluation et élagage sont exécutées en parallèle de manière (a)synchrone par différents processus explorant ces sous-espaces. Dans le mode synchrone, l'algorithme $B\&B$ comporte différentes phases. Durant chaque phase, les processus de l'algorithme effectuent leur exploration de manière indépendante. Entre les phases, les processus se synchronisent pour s'échanger de l'information, par exemple la meilleure solution trouvée. Dans le mode asynchrone, les processus de l'algorithme communiquent de manière imprévisible.

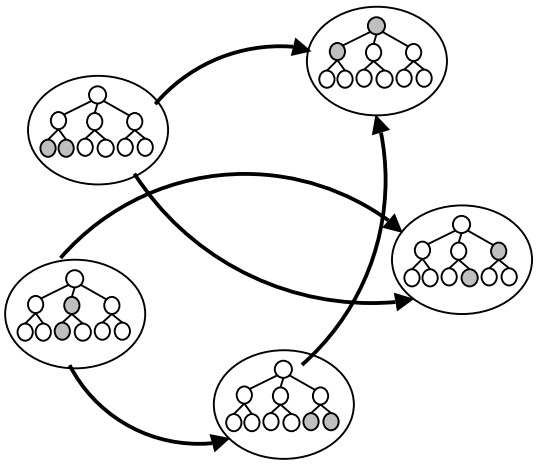

FIG. 2.8 – Illustration du modèle d'exploration arborescente parallèle.

Comparé aux autres modèles, le modèle d'exploration arborescente parallèle a suscité beaucoup d'intérêt et a fait l'objet de beaucoup de travaux de recherche pour deux raisons majeures. D'une part, le degré de parallélisme de ce modèle peut être très important sur les instances de grande taille justifiant à lui tout seul l'intérêt d'une grille de calcul. D'autre part, l'implémentation du modèle pose plusieurs problèmes constituant des défis de recherche dans le domaine de l'algorithmique parallèle. Parmi ces problèmes, on peut citer le placement et la gestion de la liste des sous-problèmes à résoudre, la répartition et le partage de la charge (des sous-problèmes générés), la communication de la meilleure solution trouvée, la détection de la terminaison de l'algorithme, et la tolérance aux pannes.

Dans [GC94], une analyse des algorithmes $B\&B$ synchrones et asynchrones a été présentée. Il ressort de cette analyse que la synchronisation s'avère inutile et inefficace même sur des machines homogènes à passage de messages ne dépassant pas 32 processeurs. Son exploitation sur les grilles de calcul (de nature hétérogène et volatile) ne peut donc être recommandée. Dans le même article, une étude a été également menée sur le problème du placement et de la gestion de la liste des sous-problèmes à résoudre. Deux classes d'algorithmes ont été identifiées : ceux utilisant une seule liste de sous-problèmes partagée par tous les processus et ceux dans lesquels plusieurs listes sont utilisées. Dans les conclusions de cette étude, il est stipulé que l'utilisation d'une seule liste peut être intéressante pour

les problèmes dont le calcul de la borne est non trivial, et pour les machines (notamment à mémoire partagée) ayant un petit nombre de processeurs. On ne peut donc pas recommander l'utilisation d'une seule liste dans le contexte de grilles.

Dans notre étude des problèmes évoqués précédemment, nous considérons seulement les algorithmes asynchrones utilisant plusieurs listes pour stocker les sous-problèmes générés car il nous semble qu'ils sont mieux adaptés aux grilles. Il existe trois approches pour la répartition de ces listes : *collégiale*, *groupée* et *mixte* [GC94]. Dans l'approche collégiale, chaque processus a sa propre liste où il stocke les sous-problèmes qu'il génère. L'organisation groupée consiste à définir différentes partitions de processus, et une liste est associée à chaque partition pour stocker les sous-problèmes générés par ses processus. L'approche mixte est une organisation collégiale dans laquelle chaque processus a sa propre liste mais partage également une liste globale avec tous les autres processus. L'approche collégiale étant totalement distribuée, elle doit assurer un meilleur passage à l'échelle. Par contre, l'approche mixte peut s'avérer adéquate et efficace sur une grille à plusieurs domaines d'administration. Les processus appartenant au même domaine d'administration partageront la même liste de sous-problèmes à traiter.

Le problème de régulation de charge consiste à minimiser l'occurrence de situations dans lesquelles des processus ont beaucoup de sous-problèmes à traiter pendant que d'autres ont des listes vides. Une politique de régulation de charge est alors nécessaire, celle-ci aura comme objectif d'initier aux moments opportuns des transferts de sous-problèmes des machines en surcharge vers les machines en sous-charge. Trois agents définissent une politique de régulation de charge : un agent d'information, un agent de transfert et un agent de localisation [MDLT96, Mel97]. Le premier agent a pour rôle de collecter l'information sur la charge des processeurs. Il comporte un indicateur de charge (nombre de sous-problèmes, taux d'utilisation CPU, etc.) permettant de définir ce qu'est la charge d'une machine, et d'une stratégie d'échange d'information (centralisée, distribuée ou hiérarchique). L'agent de transfert utilise une stratégie passive, active ou mixte. Dans la stratégie passive (resp. active), les transferts sont à l'initiative des processeurs en sous-charge (resp. surcharge). La stratégie mixte est une combinaison des deux. L'agent de localisation permet de déterminer le processeur destinataire (souvent le moins chargé) des sous-problèmes en trop dans les machines surchargées. Dans un environnement de type grille de calcul, pour tenir compte de l'hétérogénéité des machines la puissance de celles-ci devrait être prise en compte dans la définition de l'indicateur de charge. Une collecte d'information à grande échelle centralisée peut entraîner un goulot d'étranglement lorsque la charge globale évolue trop vite. Dans le modèle de vol de cycles à grande échelle, exploité dans nos travaux, l'hétérogénéité est prise en compte de manière naturelle : les machines les plus puissantes demandent plus de travail que les machines les moins puissantes.

La meilleure solution trouvée par un processus peut être communiquée aux autres processus soit au travers d'un espace partagé soit directement par diffusion. La première approche consiste à utiliser une variable globale pour le stockage de la meilleure solution trouvée. Cette variable est mise à jour chaque fois qu'un processus a trouvé une solution meilleure que celle trouvée jusqu'alors. La communication de la nouvelle valeur de la meilleure solution aux autres processus peut être faite par diffusion ou suivant une autre approche basée sur la propagation de celle-ci de voisin en voisin. La communication par

diffusion dans un environnement à grande échelle ne peut être envisagée. Le problème de détection de la terminaison de l'exécution n'est pas spécifique aux algorithmes $B\&B$. Dans notre approche proposée dans le chapitre 3, la terminaison est détectée sur chaque processeur suite à un échec à une demande de travail. Sa résolution est souvent difficile en absence d'un élément de contrôle central.

Contrairement aux métaheuristiques où une certaine perte du traitement est autorisée puisqu'on recherche une solution approchée, le problème de tolérance aux pannes pour les méthodes exactes est plus crucial. En effet, une perte d'un ou plusieurs sous-problème(s) risque d'empêcher l'obtention de la solution optimale. Sur une grille de calcul volatile, le problème peut être traité soit au niveau de l'intergiciel soit au niveau de l'application. Au niveau intergiciel, les solutions proposées sont souvent indépendantes des applications. Le processus en panne est souvent relancé depuis le début, ce qui peut s'avérer inefficace pour les processus à longue durée de vie. Au niveau applicatif, le traitement du problème repose sur le mécanisme de sauvegarde/restauration (ou *checkpointing*). Ce mécanisme permet de relancer le processus en panne à partir du dernier point de sauvegarde. Il est donc important d'identifier les données à sauvegarder. L'information sauvegardée par chaque processus contient principalement la meilleure solution trouvée, le sous-problème en cours de traitement et la liste des sous-problèmes en attente. D'autres informations indiquées précédemment peuvent également être sauvegardées si le modèle parallèle est combiné avec les modèles d'évaluation parallèle des bornes et d'évaluation parallèle de chaque borne.

2.3.3 Le modèle d'évaluation parallèle des bornes

Par analogie aux métaheuristiques parallèles, le modèle d'évaluation parallèle des bornes (Figure 2.9) correspond au modèle d'évaluation parallèle d'une population. Le modèle s'apparente au parallélisme de données, et permet la parallélisation de la phase d'évaluation des noeuds générés par l'opérateur de séparation. Ce modèle est exploitable dans le cas où la phase d'évaluation des bornes est entièrement exécutée à la suite de l'opération de séparation. Il ne change pas la conception de l'algorithme i.e. il est identique à la version séquentielle sauf que la phase d'évaluation est plus rapide. Le principal avantage de ce modèle est sa généricité.

Par contre, dans un environnement de type grille de calcul il peut s'avérer inefficace pour les raisons suivantes : (1) le modèle est de nature synchrone, ce qui le rend coûteux en temps CPU dans un contexte hétérogène et volatile; (2) sa granularité (coût de la fonction de calcul de la borne) peut s'avérer fine et donc pénalisante dans un environnement large échelle. Par exemple, dans le cas du problème du Flow-Shop le coût CPU de l'évaluation d'une borne n'est pas suffisant pour justifier son exploitation. (3) le degré de parallélisme de ce modèle est dépendant du problème traité. Il est souvent limité et décroît au fur et à mesure de l'exploration car le nombre de sous-problèmes générés diminue au fur et mesure qu'on rajoute des contraintes à l'opérateur de séparation. Sa combinaison avec le modèle d'exploration arborescente parallèle peut générer un parallélisme massif ne pouvant être exploité de manière efficace que sur grille de calcul.

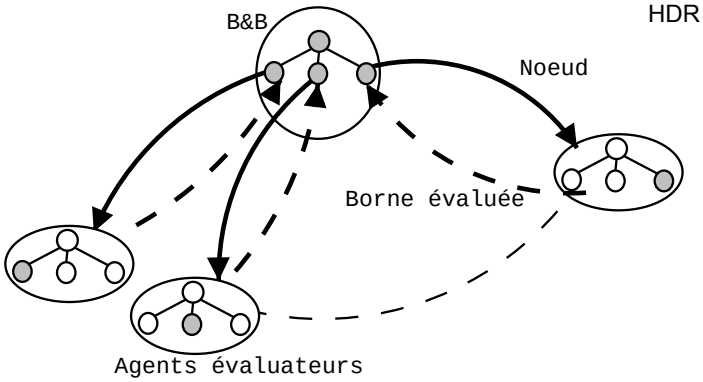

FIG. 2.9 – Illustration du modèle d'évaluation parallèle des bornes.

2.3.4 Le modèle d'évaluation parallèle d'une borne

Le modèle d'évaluation parallèle d'une borne est similaire au modèle d'évaluation parallèle d'une solution ou d'un mouvement pour les métaheuristiques parallèles. La même analyse peut donc être faite sur ce modèle. Brièvement, le modèle ne change pas la conception de l'algorithme i.e. il est identique à la version séquentielle sauf que la phase d'évaluation est plus rapide. En outre, le modèle est dépendant du problème traité, synchrone et de type *Fermier-Travailleurs*. Son extensibilité étant limitée, son exploitation sur une grille est plus efficace lorsqu'il est combiné à d'autres modèles.

2.4 Mécanismes d'hybridation

2.4.1 Motivations de l'hybridation

Afin de tirer avantage des bénéfices apportés par les différentes méthodes d'optimisation, il est souvent nécessaire de les combiner. Aujourd'hui, les méthodes hybrides permettent d'obtenir les meilleurs résultats sur la plupart des problèmes, qu'ils soient académiques (voyageur de commerce, affectation quadratique, etc.) ou pratiques (issus du monde réel) [WM95]. Nous nous sommes intéressés à deux types d'hybridation : l'hybridation entre métaheuristiques (à population de solutions et à solution unique) et l'hybridation entre métaheuristiques et méthodes exactes.

La motivation du premier type d'hybridation est d'exploiter à la fois le pouvoir d'exploration des métaheuristiques à population de solutions et le pouvoir d'intensification des métaheuristiques à solution unique pour produire des solutions diversifiées et de meilleure qualité. D'autre part, comme on le verra dans le chapitre 3, le deuxième type d'hybridation permet d'exploiter la complémentarité entre métaheuristiques et méthodes exactes : (1) les méthodes exactes permettent de prouver l'optimalité des solutions pour des instances de taille raisonnable, alors que les méthodes approchées trouvent de bonnes solutions pour des instances de taille nettement supérieure et/ou avec un nombre de contraintes beaucoup plus important. (2) Les solutions optimales trouvées par les méthodes exactes servent sou-

vent d'étalon pour mesurer l'efficacité des méthodes approchées. (3) Les bonnes solutions obtenues par les méthodes approchées sont quant à elles d'excellentes bornes pour élaguer des branches des méthodes exactes pendant le parcours arborescent et réduire ainsi considérablement le temps de calcul.

2.4.2 Mécanismes d'hybridation

Une taxinomie hiérarchique et à plat des différents schémas d'hybridation est proposée dans [Tal02] et illustrée par la figure 2.10. Dans la classification à plat, trois critères sont considérés : la composition des méthodes d'optimisation hybridées (homogène ou hétérogène), leur fonction (généraliste ou spécifique) et la portée du domaine du problème considéré (globale ou partielle). Les hybridations dites homogènes, par exemple le modèle insulaire, se basent sur le couplage de métaheuristiques de même type, mais généralement associées à des paramètres différents. Les hybridations hétérogènes, comme la méthode GRASP, utilisent des métaheuristiques de types différents. Les hybridations générales visent à résoudre un même problème d'optimisation. Au contraire, les hybridations spécialistes regroupent des méthodes dédiées chacune à un problème différent. Les algorithmes impliqués dans des hybridations globales considèrent tout l'espace décisionnel dans sa totalité. Dans des hybridations partielles, le problème à résoudre est préalablement décomposé en sous-problèmes. Chacune des méthodes de résolution hybridées est associée à l'exploration d'un sous-espace.

Dans la classification hiérarchique, deux niveaux (*Haut* et *Bas*), et deux modes (*Relais* et *Co-évolutionnaire*) d'hybridation sont considérés permettant d'identifier quatre classes : *Hybride Bas-niveau Relais (HBR)*, *Hybride Bas-niveau Co-évolutionnaire (HBC)*, *Hybride Haut-niveau Relais (HHR)* et *Hybride Haut-niveau Co-évolutionnaire (HHC)*.

2.4.2.1 La classe *Hybride Bas-niveau Relais (HBR)*

La classe des hybrides de bas niveau en mode relais regroupe les hybrides souvent constitués d'une métaheuristique à solution unique dans laquelle est insérée une autre autre méthode de résolution. Il existe très peu de méthodes hybrides appartenant à cette classe. Par exemple, dans [MOF92] une méthode de récuit simulé est combinée avec une méthode de descente. A chaque itération du récuit simulé, la solution localement optimale courante est perturbée pour obtenir la solution initiale de la méthode de descente. Celle-ci remplace la solution localement optimale courante du récuit simulé si elle lui est meilleure.

2.4.2.2 La classe *Hybride Bas-niveau Co-évolutionnaire (HBC)*

La classe des hybrides de bas niveau en mode co-évolutionnaire regroupe les hybrides constitués en général d'une méthode de résolution insérée dans une métaheuristique à population de solutions. La meilleure illustration de cette classe est la substitution d'un opérateur génétique par une métaheuristique à solution unique. En effet, un opérateur de recombinaison ou de mutation pourrait être remplacé par une méthode de recherche locale ou une méthode exacte considérant le(s) individu(s) en entrée comme des (une)

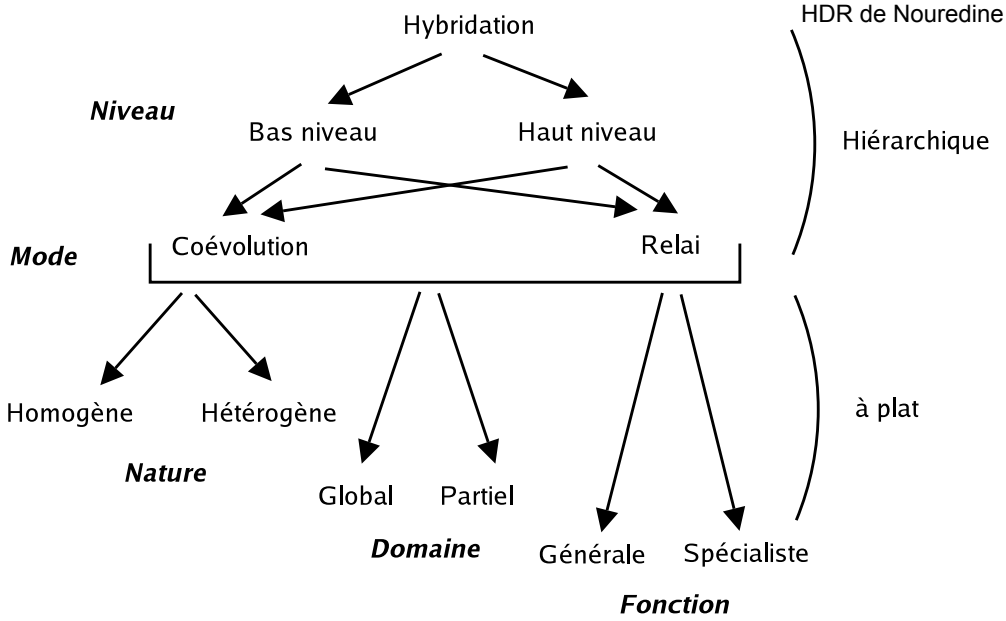

FIG. 2.10 – Classifications *hiérarchique* et *à plat* des schémas d'hybridation de méthodes d'optimisation combinatoire.

solution(s) initiale(s). Les nouvelles solutions générées intégreraient ensuite la population de la métaheuristique à population de solutions. Par exemple, dans [CANT95], un algorithme $B\&B$ explore les solutions potentielles issues de la recombinaison de deux parents afin d'en déduire la plus intéressante.

Ce type d'hybridation transforme des opérateurs de granularité fine en opérateurs de moyenne voire de grosse granularité. La phase de transformation devient ainsi coûteuse en temps d'exécution, ce qui rend l'utilisation du parallélisme inévitable. Pour avoir une idée du coût, considérons à nouveau l'application de conception de réseaux mobiles. En pratiquant l'hybridation qui consiste à remplacer un opérateur (mutation par exemple) par une méthode de recherche locale on fait passer le coût global de l'opérateur de quelques secondes à plus d'un mois ! L'hybridation parallèle peut être assimilée au modèle parallèle multi-départ sans coopération si plusieurs recombinaisons sont appliquées simultanément. Les mêmes considérations peuvent donc être appliquées quant à son implémentation sur un support de type grille de calcul. Les résultats expérimentaux présentés au chapitre 3 montrent que, combinées aux métaheuristiques suivant ce schéma, les méthodes exactes sont nettement plus efficaces.

2.4.2.3 La classe *Hybride Haut-niveau Relais (HHR)*

Dans les hybrides de haut niveau en mode relais, les méthodes de résolution conservent leur intégrité et sont exécutées en séquence. Dans ce schéma d'hybridation, la sortie d'une méthode de résolution est utilisée comme entrée d'une autre méthode de résolution. Par exemple, on peut utiliser un ou plusieurs algorithme(s) glouton(s) afin de générer une bonne

solution initiale ou une population initiale de bonnes solutions, exploitée(s) ensuite par une métaheuristique. On peut également appliquer une métaheuristique à solution unique (un récuit simulé [MG95] ou une recherche tabou [TMS94]) ou une méthode exacte sur l'ensemble des solutions d'une population obtenue par une métaheuristique à population de solutions pour améliorer leur qualité. Dans le contexte multi-objectif, le front Pareto obtenu par une métaheuristique (un AG par exemple) est amélioré par une autre métaheuristique à solution unique (la recherche tabou par exemple) ou une méthode exacte. Le schéma HHR peut être également assimilé au modèle parallèle multi-départ sans coopération. Par conséquent, son étude dans le contexte de grilles de calcul se ramène à l'étude du modèle multi-départ dans le même contexte. Les résultats expérimentaux présentés au chapitre 3 montrent que, combinées aux métaheuristiques en mode HHR, les méthodes exactes sont nettement plus efficaces.

2.4.2.4 La classe *Hybride Haut-niveau Co-évolutionnaire (HHC)*

Dans les hybridations de haut niveau co-évolutionnaires, la structure interne des méthodes de résolution hybridées n'est pas modifiée. Ces dernières sont exécutées simultanément et coopèrent pour résoudre un problème donné. Le meilleur exemple illustratif de ce schéma d'hybridation est le modèle insulaire. Dans ce cas, nos travaux dédiés à l'évaluation du schéma HHC montrent que celui-ci permet d'améliorer la robustesse des métaheuristiques et la qualité des solutions produites. Ces travaux montrent également que l'utilisation de ce schéma pour faire coopérer les métaheuristiques et les méthodes exactes est source d'efficacité mais pose des problèmes de gestion de la concurrence de ces méthodes.

Les schémas d'hybridation généralistes sont indépendants du problème traité. Pour assurer leur généricité à l'implémentation, il faut faire en sorte que la sémantique des opérateurs substitués soit identique à celle des substituants. Par exemple, la sémantique d'un opérateur de mutation doit être la même que celle d'une métaheuristique à solution unique. Cette approche a été utilisée pour la mise en oeuvre de l'hybridation dans la plate-forme ParadisEO.

2.5 Evaluation des performances

L'évaluation des performances des méthodes d'optimisation parallèle sur grilles de calcul est complexe. Cette complexité est liée à la fois à la nature particulière de ces méthodes et aux caractéristiques des grilles de calcul. En effet, d'une part, les méthodes exactes ont un comportement non déterministe à l'exécution, les métaheuristiques ont un caractère stochastique. D'une exécution à une autre, les statistiques produites (qualité des solutions, temps d'exécution, etc.) sont souvent différentes. C'est pourquoi, il est recommandé de calculer la moyenne et l'écart type des statistiques générées par les différentes exécutions. D'autre part, les caractéristiques des grilles rendant l'évaluation des performances complexe sont essentiellement l'hétérogénéité (notamment des machines, du réseau, et des systèmes d'exploitation) et la disponibilité variable des machines et/ou du réseau.

L'évaluation des performances des méthodes d'optimisation parallèle est basée sur des paramètres spécifiques aux méthodes utilisées ou généraux. Les paramètres spécifiques sont par exemple la qualité (*fitness*) des solutions produites souvent utilisée pour mesurer la qualité des composants des métaheuristiques. Dans le contexte multi-objectif, cette qualité peut être évaluée en utilisant des métriques particulières telles que la contribution, l'entropie, la distance générationnelle, la S-métrique, etc. Une étude de ces métriques est proposée dans [KC99]. Dans [Bas05], on distingue les métriques supposant la connaissance préalable d'un front Pareto optimal et celles basées sur le contraire. Le premier type de métriques est souvent basé sur le calcul de la distance absolue entre le front approché et le front optimal. Par exemple, la distance générationnelle [VL00b] est la distance (par exemple euclidienne dans le cas bi-objectif) moyenne entre les deux fronts. Le deuxième type de métriques utilise souvent une distance relative entre deux fronts approchés pour mesurer leur qualité en termes de convergence et de diversification. Par exemple, la contribution [MTR00] permet de mesurer la convergence relative d'un front par rapport à un autre en calculant la proportion dans ce front de solutions non dominées obtenues par la fusion des deux fronts. L'entropie [MTR00] permet de mesurer la dispersion des solutions d'un front en comparaison de celle d'un autre front. Une autre métrique mesurant à la fois la convergence et la diversité d'un front par rapport à un autre est la S-métrique [ZT99]. Celle-ci permet de mesurer le volume de l'espace dominé par un front Pareto. Tous ces indicateurs sont intégrés à la plate-forme GUIMOO (Graphical User Interface for Multi Objective Optimization) développée dans le cadre de la thèse de Sébastien Cahon. La plate-forme est disponible sur le Web à l'adresse : http ://www.nongnu.org/guimoo/.

Pour les méthodes exactes, le paramètre souvent évalué est le nombre de sous-problèmes générés. Ce paramètre n'est en général pas révélateur du réel coût de traitement des sous-problèmes. En effet, d'une part, le temps de traitement est variable d'un sous-problème à un autre. D'autre part, le coût de gestion du parallélisme n'est pas pris en compte.

Les paramètres généraux sont communs non seulement aux différentes classes de méthodes d'optimisation mais aux algorithmes parallèles en général. Parmi ces paramètres, on distingue les métriques communément utilisées en calcul haute performance notamment l'*accélération* (ou *speed-up*), l'*efficacité* (ou *efficiency*) et l'*extensibilité* (ou *scalability*). L'accélération et l'efficacité mesurent respectivement le rapport entre le temps d'exécution du meilleur algorithme séquentiel connu et le temps d'exécution parallèle, et le rapport entre l'accélération et le nombre de machines utilisées. La mesure de l'extensibilité, importante dans le contexte des grilles, permet de montrer comment l'accélération ou l'efficacité se comporte en augmentant le nombre de processeurs.

Le caractère naturellement hétérogène et dynamique des grilles rend difficile la mesure de ces paramètres généraux à l'exécution. Les mesures traditionnellement utilisées ne peuvent pas convenir ici puisqu'elles ne s'appliquent qu'à un ensemble de ressources homogènes et dédiées. Une nouvelle approche de l'évaluation des performances proposée dans [GKYL00] est basée sur la normalisation du temps CPU nécessaire à l'exécution d'un processus avec la performance de la machine correspondante. Les temps d'exécution sont agrégés et des statistiques viables sur différentes exécutions sont établies. Ce facteur de normalisation peut être basé sur une information matérielle si celle-ci est disponible (par exemple MIPS, FLOPS, etc.).

Dans [GKYL00], différentes métriques de performance ont été définies et tout particulièrement l'efficacité parallèle sur environnements composés de ressources de calcul hétérogènes et dynamiques dans l'espace et dans le temps. Ces métriques sont intégrées dans ParadisEO-CMW puisqu'elles sont intégrées à MW. Considérons une application parallèle composée d'un ensemble $\mathcal{J}$ de tâches déployées sur une grille composée d'un ensemble de $\mathcal{I}$ ressources CPU déclarées (Noeuds Travailleurs ou NT). Avant de définir les différentes métriques, considérons les définitions suivantes :

- $\alpha(i)$: facteur de normalisation de la performance du NT $i \in \mathcal{I}$.
- $U(i)$: temps de disponibilité cumulée du NT i.
- $w(j)$, l'index du NT i ayant accompli la tâche $j \in \mathcal{J}$.
- $t(j)$: temps utilisateur passé par le NT $w(j)$ à accomplir la tâche j.
- $\mathcal{W}$: temps total nécessaire à l'obtention des résultats.
- T : temps CPU cumulé de tous les NT.

Les mesures de performances dans le contexte de grilles dynamiques et hétérogènes peuvent être définies de la manière suivante :

- $\mathcal{T}$: temps CPU cumulé normalisé consacré à l'accomplissement de toutes les tâches déployées.

$$\mathcal{T} = \sum_{j \in \mathcal{J}} \alpha(w(j)) \times t(j)$$

  Cette information exprime le temps séquentiel qu'il serait nécessaire à une machine de performance « moyenne » au sein d'une grille afin d'exécuter toutes les tâches.

- $\mathcal{P}$: performance équivalente du groupe de ressources connectées.

$$\mathcal{P} = \frac{\sum_{i \in \mathcal{I}} \alpha(i) \times U(i)}{\sum_{i \in \mathcal{I}} U(i)}$$

  Cette métrique définit la puissance maximale potentiellement exploitable. Sa formalisation tient compte de la performance associée à chacune des ressources CPU participantes, mais aussi de sa durée cumulée de disponibilité durant le temps d'exécution.

- $\mathcal{N}$: nombre moyen de Noeuds Travailleurs participants.

$$\mathcal{N} = \frac{\sum_{i \in \mathcal{I}} U(i)}{\mathcal{W}}$$

Ce paramètre est défini par le rapport de la durée de disponibilité cumulée de chaque ressource participant au calcul par le temps nécessaire à l'obtention des résultats depuis l'instant de soumission du travail.

- μ : efficacité parallèle.

$$\mu = \frac{\sum_{j \in \mathcal{J}} t(j)}{\sum_{i \in \mathcal{I}} U(i)}$$

L'efficacité sur environnement d'exécution hétérogène et non dédié se définit par le rapport du temps d'accomplissement cumulé de toutes les tâches (quelque soit leur affectation à un NT donné) par celui de disponibilité totale et cumulée des ressources CPU durant le temps d'exécution.

2.6 Conclusion

Dans ce chapitre, nous avons proposé une analyse des méthodes d'optimisation combinatoire parallèles hybrides en vue de leur gridification et leur conception et mise en oeuvre réutilisables. La gridification signifie la prise en compte des caractéristiques des grilles lors de la conception des méthodes d'optimisation, notamment l'hétérogénéité, la volatilité, la nature multi-domaine d'administration et la grande échelle.

- Dans les taxinomies proposées dans la littérature sur l'optimisation parallèle, le modèle d'évaluation parallèle d'une solution ou d'une borne est souvent ignoré. Cependant, lorsque la fonction objectif ou le calcul de la borne est parallélisable l'exploitation de ce modèle sur grilles s'avère importante. En effet, son utilisation conjointe avec d'autres modèles dans un mode hiérarchique augmente le degré de parallélisme de manière à rendre indispensable l'utilisation de grilles.

- Pour supporter le coût exorbitant des mécanismes d'hybridation, il est nécessaire de les combiner avec les modèles parallèles. Ainsi, certains schémas d'hybridation, notamment HBC et HHR, devraient être exploités suivant le modèle parallèle multi-départ. Par exemple, sur chacune des solutions d'une population finale ou du front Pareto produits par un AG on peut appliquer simultanément une recherche locale.

- L'utilisation du mode asynchrone et du vol de cycles s'avère importante pour le déploiement des différents modèles parallèles et mécanismes d'hybridation.

- Pour chaque modèle parallèle, nous avons défini les données (ou mémoire) caractérisant l'état courant de l'exécution pour la gestion transparente de la volatilité. Cet état permet de minimiser le coût de gestion de la volatilité en évitant de reprendre

l'exécution depuis le début à l'instar de ce qui est fourni par les intergiciels. Certains intergiciels tels que MW offrent des primitives de sauvegarde/restauration, et il suffira de leur fournir cet état pour mettre en place le mécanisme.

- Les modèles parallèles pour les méthodes exactes, notamment le modèle d'exploration arborescente parallèle, nécessitent la gestion du problème de régulation de charge. Cette gestion induit un coût de communication important dans un contexte à grande échelle. Il est donc indispensable de proposer un codage des noeuds de l'arbre permettant une description des unités de travail minimisant le coût de communication.

- Des études du problème de l'évaluation des performances des méthodes d'optimisation parallèle ont été proposées dans différents travaux [TAML05, CP97]. Cependant, ces études ne sont pas adaptées aux grilles. Il faudrait donc ajouter aux recommandations de ces travaux la prise en compte des caractéristiques des grilles en considérant par exemple les métriques proposées dans MW [LKGY00].

La réutilisation des méthodes d'optimisation parallèle hybride doit être basée sur la séparation conceptuelle de la partie invariante et la partie spécifique aux problèmes. L'analyse de ces méthodes nous a permis de réaliser cette séparation. Celle-ci pourrait servir pour la conception de plates-formes d'aide au développement de méthodes d'optimisation réutilisables. Elle nous a permis en particulier de concevoir et réaliser la plate-forme ParadisEO présentée dans le chapitre suivant.

Chapitre 3

ParadisEO : une plate-forme d'aide à la conception de métaheuristiques sur grilles

Le travail présenté dans ce chapitre rentre principalement dans le cadre de la thèse de Sébastien Cahon [Cah05] soutenue le 1^{er} juillet 2005. Cette thèse a été menée au sein du projet DOLPHIN de l'INRIA Futurs, et s'est inscrite dans le cadre du projet ACI-GRID DOC-G (Défis en Optimisation Combinatoire sur Grille) et d'un contrat avec France Telecom R&D.

Dans ce chapitre, nous allons montrer l'intérêt et les objectifs des plates-formes logicielles et proposerons une taxinomie et un état de l'art de celles dédiées aux métaheuristiques parallèles hybrides. Ensuite, nous présenterons la plate-forme ParadisEO en donnant son architecture. Nous nous attacherons aussi à montrer comment ParadisEO se distingue des autres plates-formes en fournissant les différents modèles parallèles et mécanismes d'hybridation pouvant être exploités de manière transparente sur des architectures dédiées. Nous présenterons également le couplage de la plate-forme avec Condor-MW pour permettre l'exploitation transparente de ces modèles et mécanismes sur des grilles de calcul. Nous expliquerons le mécanisme de sauvegarde/restauration que nous avons proposé et son intégration dans MW [LKGY00]. Nous terminerons le chapitre par deux applications réelles en guise de validation de ParadisEO. Après formulation du problème traité dans chaque application, nous décrirons leur résolution parallèle hybride et en évaluerons les performances.

3.1 Taxinomie et état de l'art des plates-formes de métaheuristiques parallèles hybrides

Nous allons d'abord présenter les approches de réutilisation logicielle en mettant en évidence l'approche *plate-forme* (ou *framework*) adoptée pour la conception de ParadisEO. Ensuite, nous présenterons les objectifs conceptuels et les moyens utilisés pour la mise en

oeuvre de la plate-forme. Ces objectifs constitueront les critères d'une taxinomie et les aspects abordés dans un état de l'art que nous proposerons autour des plates-formes de métaheuristiques parallèles hybrides.

3.1.1 Approches de réutilisation logicielle et intérêt des plates-formes

La réutilisabilité de composants logiciels, en particulier des métaheuristiques parallèles hybrides, se définit par leur capacité à être exploités au sein de plusieurs différentes applications sans effort significatif de la part du développeur [FVW98]. Cette définition nous a amené à identifier trois approches principales dédiées à la réutilisation de métaheuristiques parallèles hybrides : *Aucune réutilisation ou From scratch*, *Seule réutilisation du code* et enfin *Réutilisation du code et de la conception*. La motivation de l'approche *Aucune réutilisation* est l'apparente simplicité des algorithmes mis en oeuvre incitant le développeur à les implémenter lui-même. Néanmoins, outre la difficulté de maintenance du code qu'elle implique, une telle approche requiert du temps et de l'énergie en étant sujette à erreurs.

L'approche *Seule réutilisation du code* consiste à réutiliser un code dit de "tierce-partie", souvent disponible sur le Web sous la forme de *programmes indépendants* et libres, ou de *librairies*. La réutilisation de programmes indépendants nécessite un examen détaillé du code et la réécriture de certaines sections spécifiques au problème traité. Cette tâche s'avère longue, fastidieuse et sujette à erreurs. C'est pourquoi, les librairies constituent une solution plus efficace et plus fiable pour la réutilisation de code existant [Wal99]. De plus, étant souvent testées et documentées, leur maintenance est ainsi grandement facilitée. Par ailleurs, l'avènement de nouveaux langages orientés-objet tels que Java a conduit au développement de librairies, voire la réécriture de certaines librairies existantes, facilitant et encourageant ainsi leur réutilisation. Cependant, les librairies permettent seulement la réutilisation de code, mais ne sont pas dédiées à faciliter la réutilisation de conception.

L'objectif 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 (et donc l'effort de développement) chaque fois qu'un nouveau problème d'optimisation est considéré. Dans le 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* [GHJV94]). Ces composants sont implémentés et encapsulés au sein de plates-formes [JF88].

Ainsi, une plate-forme peut être définie comme un ensemble de classes intégrant des modèles génériques de solutions logicielles dédiées à une famille de problèmes apparentés [FVW98]. 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. Dans une plate-forme, le code encapsulé fait appel à celui développé par 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 total de la partie invariante des algorithmes. L'utilisateur fournit seulement la partie spécifique à son problème. Par consé-

quent, 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 associée.

3.1.2 Objectifs conceptuels et moyens de mise en oeuvre

3.1.2.1 Réutilisation maximum de conception et de code

La réutilisation maximum de conception et de code nécessite une analyse approfondie du domaine d'application auquel la plate-forme est dédiée. Cette analyse doit permettre une séparation maximum entre la partie invariante et 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 [AT02a]). La partie variable est spécifiée dans la plate-forme mais son implémentation est laissée à la charge de l'utilisateur. Dans [RJ97], on recommande l'utilisation de la composition au détriment de l'héritage pour mettre en oeuvre 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. Nous avons appliqué cette recommandation pour la conception et la réalisation de ParadisEO. Dans [BLTZ03], une autre méthode non orientée-objet est utilisée pour réaliser cette séparation dans la plate-forme *Pisa*. En effet, deux modules sont considérés séparément pour les parties variable et invariante, et ceux-ci communiquent via des fichiers textes.

3.1.2.2 Adaptation minimale et flexible

La partie variable d'une plate-forme constitue un ensemble de « trous » (*holes* ou *hot spots*) [PPSS95] au sein des squelettes de la partie invariante. Nous appellerons ces "trous" des *points d'adaptation*. L'adaptation à un problème donné consiste à "remplir ces trous" en fournissant des classes obtenues soit par spécialisation de classes existantes ou par simple création. Ces classes vont être attachées aux points d'adaptation pour être utilisées par la plate-forme. L'objectif décrit dans la section précédente permet de minimiser le nombre de ces points, réduisant ainsi l'effort de l'utilisateur. Un autre moyen facilitant grandement la réutilisation consiste à fournir à l'utilisateur un *guide d'adaptation* [PPSS95]. Ce guide contient l'objectif et la procédure d'adaptation, et ne doit en aucun cas contenir les détails d'implémentation de la partie invariante de la plate-forme. La procédure doit décrire pas à pas le processus d'adaptation. Pour ParadisEO, ce guide consiste en une série de tutoriels disponibles sur le Web.

3.1.2.3 Large utilité

L'objectif ultime d'une plate-forme est d'être exploitée par la plus large communauté d'utilisateurs possible. Il doit donc offrir aux programmeurs une grande variété de méta-heuristiques mono et multi-objectifs, différents types de modèles parallèles distribués en particulier gridifiés et de mécanismes d'hybridation. En outre, le succès d'une plate-forme

dépend aussi des différents supports qu'il offre : une interface graphique, un outil d'instrumentation et de suivi de l'exécution, des métriques pour la mesure de performance, une documentation en ligne, etc. Dans ParadisEO, cet objectif s'est traduit par toutes les extensions appliquées à la plate-forme EO et présentées dans la section 3.2.2.

3.1.2.4 Accès transparent à la performance et à la robustesse

Sachant que les métaheuristiques sont stochastiques et très gourmandes en consommation CPU, le critère de performance et de robustesse à l'exécution est primordial. Pour rendre les modèles parallèles et les mécanismes d'hybridation fournis par une plate-forme accessibles aux utilisateurs, même novices en parallélisme, ceux-ci doivent être exploitables de manière transparente. Dans ParadisEO, les modèles parallèles sont encapsulés dans des classes exploitables par simple instanciation. L'exploitation de l'hybridation est naturelle puisque la sémantique des composants mis en jeu dans le mécanisme est identique, et les types et formats des informations échangées entre métaheuristiques coopératives sont également identiques.

3.1.2.5 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 et grappes de PC, machines massivement parallèles, machines parallèles à mémoire partagée, etc.) utilisant différents systèmes d'exploitation (différentes distributions de Linux, Windows, etc.). Par conséquent, il est important de développer la plate-forme en utilisant un langage portable, des librairies standard pour la concurrence et la synchronisation (*e.g.* Pthreads) et pour la communication (*e.g.* MPI, PVM, etc.). Dans ParadisEO, ces trois dernières librairies ainsi que le langage C++ sont utilisés pour assurer la portabilité.

3.1.3 Taxinomie et état de l'art des plates-formes

De nombreuses plates-formes [MTC04] pour l'optimisation combinatoire ont été développées. Cependant, peu d'entre elles permettent d'atteindre tous les objectifs énoncés précédemment. Dans le tableau 3.1, nous présentons une étude comparative et non exhaustive de dix plates-formes logicielles "libres" de métaheuristiques, les plus représentatives à notre connaissance. Toutes ces plates-formes sont orientées-objet 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 mécanismes d'hybridation et les modèles parallèles implémentés, le langage de programmation utilisé pour son développement, et enfin le support sous-jacent utilisé pour la mise en oeuvre de la concurrence et/ou des communications.

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

	RL	AE	Hybrid.	Parall.	Langage	Support. Para.
ECJ	-	+	AE / AE	Coop. insulaire	Java	Threads Sockets TCP / IP
D-BEAGLE	-	+	AE / AE	F-T	C++	Sockets TCP / IP
J-DEAL	-	+	AE / AE	F-T	Java	Sockets TCP / IP
EasyLocal ++	+	-	RL / RL	-	C++	-
Localizer ++	+	-	RL / RL	-	C++	-
MAFRA	+	+	AE / AE RL / RL	-	Java	-
DREAM	-	+	AE / AE	Coop. insulaire	Java	Threads Sockets TCP / IP
MALLBA	+	+	AE / RL AE / AE RL / RL	Toutes	C++	Netstream MPI
EO	-	+	-	-	C++	-
ParadisEO	+	+	AE / RL AE / AE RL / RL	Toutes	C++	Threads MPI / PVM Condor / MW

TAB. 3.1 – Revue des plates-formes majeures de métaheuristiques.

plates-formes évolutionnaires sont les suivantes : *DREAM* [1] [ACE+02], *ECJ* [2] [LPB+02], *JDEAL* [3] [CLS00], *Distributed BEAGLE* [4] [GPD03] et *EO* [5] [KMRS01]. Le développement des trois premières en Java facilite leur portabilité sur différents systèmes d'exploitation. Cependant, elles se montrent souvent limitées au regard des modèles parallèles. En effet, *DREAM* et *ECJ* proposent uniquement le modèle coopératif insulaire en utilisant "Java Threads" et TCP/IP. *JDEAL* et *D-BEAGLE* implémentent le modèle distribué *Fermier-Travailleurs* pour une distribution des calculs associés aux phases de transformation et/ou d'évaluation de la population.

Dans le cadre des métaheuristiques à base de solution unique, la plupart des plates-formes existantes comme *EasyLocal++* [GS01] et *Localizer++* [MH01] ne proposent pas d'implémentations parallèles. Toutefois, il existe des plates-formes supportant le parallélisme mais souvent dédiées à une seule métaheuristique. Nous citerons pour illustration la plate-forme présentée dans [BhX01] comportant deux « squelettes » de recherche Tabou parallèles implémentant respectivement le modèle multi-départ indépendant et celui de l'évaluation parallèle du voisinage.

En pratique, peu de plates-formes logicielles sont dédiées au développement d'applications à base d'AE et de RL et leur hybridation. *MALLBA* [AtMG02], *MAFRA* [6] [KS00] et *ParadisEO* [CTM03, CMT04] en font partie. *MAFRA*, basée sur les modèles de conception (ou *design patterns*) et développée en Java, est fortement orientée hybridation mais s'avère très limitée au regard du parallélisme et de la distribution. *MALLBA* et *ParadisEO* ont de nombreux points communs : ce sont des plates-formes libres et développées en C++. Elles implémentent les différents modèles parallèles et mécanismes d'hybridation. Cependant, utilisant des classes de granularité fine, *ParadisEO* offre plus de flexibilité. D'autre part, ParadisEO offre en plus de MPI la communication à base de PVM et le multi-threading basé sur la librairie PThreads. En outre, contrairement à MALLBA, ParadisEO permet le déploiement sur grilles en utilisant Condor et MW.

3.2 ParadisEO sur environnements parallèles et grilles dédiés

Evolving Objects (EO) est une plate-forme libre *GPL* développée en C++ dans le cadre d'une collaboration européenne [KMRS01] impliquant principalement M. Keijzer [7], J.J. Merelo et G. Romero [8] et M. Schoenauer [9]. La plate-forme est dédiée à la conception d'algorithmes évolutionnaires notamment les algorithmes génétiques, les stratégies évolutionnistes, la programmation évolutionnaire et génétique. Elle est basée sur une re-

[1] Distributed Resource Evolutionary Algorithm Machine : `http://www.world-wide-dream.org`
[2] Java Evolutionary Computation : `http://www.cs.umd.edu/projects/plus/ec/ecj/`
[3] Java Distributed Evolutionary Algorithms Library : `http://laseeb.isr.ist.utl.pt/sw/jdeal/`
[4] Distributed Beagle Engine Advanced Genetic Learning Environment : `http://www.gel.ulaval.ca/~beagle`
[5] Evolving Objects : `http://www.sourceforge.org/EO`
[6] Java Memetic Algorithms Framework : `http://www.cs.nott.ac.uk/~nxk/MAFRA/MAFRA.html`
[7] Danish Hydraulic Institute
[8] GeNeura Team, Depto. Arquitectura y Tecnologia de Computadores, Universidad de Granada (Spain)
[9] Projet Fractales INRIA, LRI Paris-Sud

présentation des solutions lui conférant une flexibilité au regard des problèmes traités. Cette représentation est définie par :

- Une métrique d'évaluation associant à la solution représentée une mesure de qualité (*fitness*) pouvant être non scalaire. Dans le cadre de l'optimisation multi-objectif, la qualité d'une solution est un vecteur de plusieurs valeurs réelles. Il convient alors de définir l'opérateur de comparaison (au sens lexicographique, Pareto, etc.) à considérer.

- Au moins un opérateur dit de variation (recombinaison et/ou mutation, etc.) permettant de *transformer* une solution ou un ensemble de solutions.

Quelques représentations prédéfinies sont intégrées dans la plate-forme modélisant des structures de données élémentaires (chaînes, arbres, etc.). Pour chacune d'elles, un certain nombre d'opérateurs de recombinaison ou de mutation sont déjà développés et donc directement réutilisables. Une collection de classes implémentant la partie invariante des AE (différentes stratégies de sélection, de remplacement et d'arrêt) est également fournie.

Dans la suite, nous allons présenter l'architecture de ParadisEO et ses paquetages constituant les différentes extensions apportées à *EO*.

3.2.1 Architecture de ParadisEO

ParadisEO a une architecture multi-couche (Fig. 3.1) constituant une hiérarchie de classes à quatre niveaux de granularité décroissante : *Solvers*, *Runners*, *Helpers fournis* et *Helpers requis*.

- Les *Helpers* sont des classes de bas-niveau accomplissant des actions spécifiques en considération du processus d'évolution ou de recherche. On distingue deux catégories : *Evolutionary Helpers* (EH) et *Local Search Helpers* (LSH). Les EH regroupent essentiellement les composants d'un AE i.e. les opérateurs de sélection, de transformation et de remplacement, la fonction objectif et le critère de continuation. Certains LSH peuvent être génériques comme la stratégie d'exploration du voisinage, ou spécifiques à la métaheuristique de recherche locale comme la notion de liste Tabou dans la recherche Tabou. D'autre part, on notera quelques *Helpers* particuliers, dédiés à la gestion de modèles parallèles et distribués. Les *Helpers* coopèrent entre eux et interagissent avec les composants de la couche supérieure i.e. les *Runners*. Ces derniers invoquent les *Helpers* par le passage de paramètres aux méthodes. En effet, les *Helpers* ne sont pas associés à des données, mais au contraire accèdent à celles intégrées au sein des *Runners*.

- Les *Runners* caractérisent les classes implémentant les métaheuristiques elles-mêmes. Elles reposent sur l'exécution des métaheuristiques partant d'une solution ou d'un ensemble de solutions initiale(s) vers un état final. On distingue les *Evolutionary Runners* (ER) tels que les algorithmes génétiques, les stratégies évolutionnistes, etc. et les *Local Search Runners* (LSR) tels que la recherche Tabou ou le recuit simulé. Les *Runners* invoquent les *Helpers* afin d'accomplir des actions spécifiques sur leurs propres données. Pour illustration, un ER requiert l'accès à la fonction objectif afin

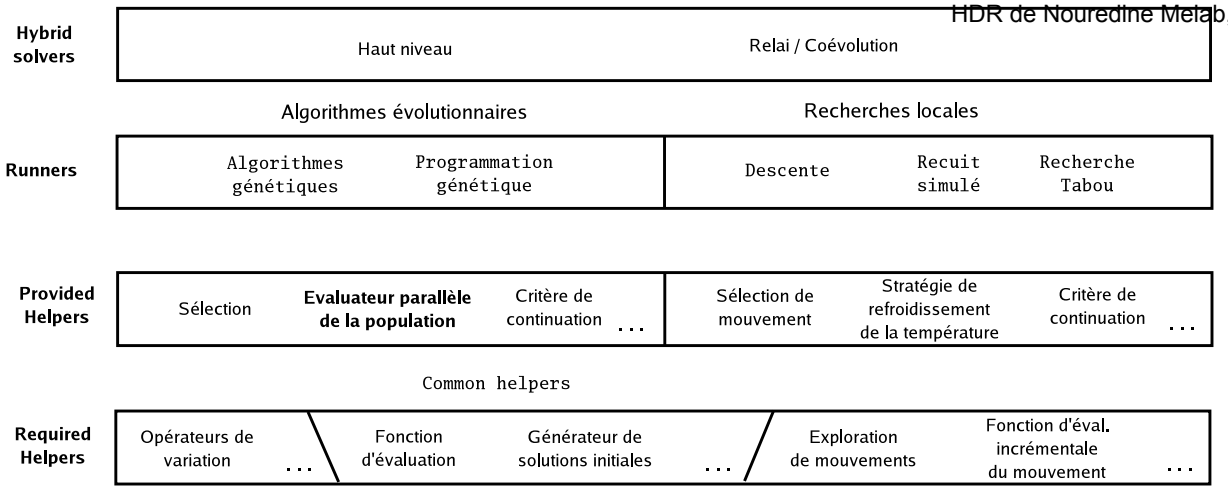

FIG. 3.1 – L'architecture de ParadisEO

d'évaluer une population donnée de solutions. Un LSR invoque le *Helper* de d'application d'un mouvement sur la solution courante. De même que précédemment, les *Runners* peuvent être implémentés de manière séquentielle ou parallèle.

– Les *Solvers* sont dédiés au contrôle du processus d'évolution ou de recherche. Ils génèrent l'état initial (une solution ou population d'individus) et définissent la stratégie combinant et ordonnançant les différentes métaheuristiques. On distingue deux types de *Solvers* : ceux basés sur une métaheuristique seule *Single Metaheuristic Solvers* (SMS) et d'autres reposant sur plusieurs métaheuristiques *Multiple Metaheuristic Solvers* (MMS). Les MMS sont plus complexes étant donné qu'ils contrôlent et assurent le contrôle de différentes métaheuristiques pouvant être hétérogènes. Ils font usage de différents mécanismes d'hybridation. Les *Solvers* interagissent avec l'utilisateur par la saisie des données d'entrée et la délivrance de sorties (*e.g.* l'affichage de la meilleure solution, la production de statistiques, etc.).

La séparation conceptuelle de la partie invariante de la partie spécifique aux problèmes est traduite à l'implémentation par la considération de deux types de classes : les classes fournies *Provided Classes* (*PC*) et les autres non développées et donc requises *Required Classes* (*RC*). Les premières encapsulent la partie invariante des métaheuristiques. Elles sont donc génériques, implémentées dans la plate-forme, et garantissent le contrôle à l'exécution. Les secondes sont identifiées et spécifiées dans ParadisEO mais sont à implémenter par le programmeur par utilisation du mécanisme de spécialisation.

3.2.2 Extensions apportées à EO

Quatre extensions majeures ont été apportées à *EO* constituant différents paquetages pouvant être utilisés séparément : extension au multi-objectif (*Multi-Objective EO* ou *MOEO*), aux métaheuristiques à solution unique (*Moving Objects* ou *MO*), au parallélisme et hybridation sur environnements dédiés (*Parallel Distributed EO* ou *ParadisEO*),

et au parallélisme et hybridation sur environnements non dédiés (*ParadisEO-Condor MW* ou *ParadisEO-CMW*). Les deux premières extensions sont présentées dans la suite de cette section. Les deux suivantes seront présentées respectivement dans les sections 3.2.3 et 3.3.

Dans la plate-forme *EO*, seule l'étape de sélection a été considérée dans le contexte multi-objectif. Aussi, nos contributions *MOEO* consistent en l'intégration de *Helpers* fournis implémentant des opérateurs d'élitisme et de partage, des opérateurs génétiques adaptatifs et enfin des métriques pour l'évaluation des performances.

Un opérateur adaptatif est un opérateur complexe encapsulant des opérateurs de croisement, de mutation, ou même de recherche locale. Durant le processus d'évolution, il se comporte comme un de ses opérateurs internes choisi en fonction de sa performance récente (*i.e.* sa contribution à faire progresser le front Pareto) [Bas05]. Les opérateurs adaptatifs ont été intégrés dans *MOEO* sous forme d'opérateurs de recombinaison ou de mutation standard encapsulant chacun un ensemble d'opérateurs du même type. Sont également intégrées dans *MOEO* diverses métriques notamment la contribution et l'entropie.

Moving Objects (ou *MO*) est une extension de la plate-forme *EO* fournissant principalement trois *Runners* implémentant les métaheuristiques à solution unique : la descente, le recuit simulé et la recherche Tabou. Celles-ci sont basées sur l'exploration d'un voisinage défini par la notion de mouvement et trois opérateurs principaux. Ceux-ci permettent d'initialiser un mouvement, de générer le mouvement suivant de manière déterministe, et de calculer de manière incrémentale (donc efficace) la valeur de *fitness* associée à la solution qui serait obtenue en lui appliquant un mouvement. Ces opérateurs sont spécifiques au problème traité mais peuvent être réutilisés d'une métaheuristique à une autre. Une implémentation est fournie (sur le problème du Voyageur de Commerce) sous forme de *Helpers* pour lesquels le type de mouvement (paramètre de type *template*) doit être instancié pour être utilisés. Tous les autres opérateurs invariants communs ou spécifiques aux trois métaheuristiques sont implémentés sous forme de *Helpers* fournis. En outre, les interfaces des parties dépendantes des problèmes traités sont fournies dans la plate-forme et constituent des *Helpers* requis.

3.2.3 Parallélisme et hybridation dans ParadisEO

Différents modèles parallèles et mécanismes d'hybridation ont été identifiés dans le chapitre précédent. Nous allons montrer ici comment ils ont été implémentés dans ParadisEO. Pour les modèles parallèles, nous distinguerons les implémentations suivant le mode de déploiement synchrone ou asynchrone et le support matériel utilisé i.e. machines à mémoire partagée ou distribuée. Ces modèles sont exploitables de manière transparente car ils sont implémentés dans ParadisEO sous forme de *Helpers* fournis. La communication et la synchronisation sur machines à mémoire partagée est assurée par accès multi-threadé à la mémoire commune en utilisant la librairie Posix Threads. Sur les machines à mémoire distribuée, la communication est assurée par ajout (par composition) de canaux de communication aux *Helpers* fournis implémentant les différents modèles. Ces canaux constituent des *Helpers* fournis dédiés soit à l'échange d'individus ou de valeurs de *fitness* de ces derniers. Ils sont implémentés par des zones mémoire (ou *buffers*) manipulées de manière transparente, au gré de l'utilisateur, par des primitives de PVM ou MPI.

3.2.3.1 Parallélisme des métaheuristiques à population de solutions

Nous rappelons que trois modèles parallèles ont été identifiés : le modèle insulaire, le modèle d'évaluation parallèle de la population, et le modèle d'évaluation parallèle d'une solution.

- L'implémentation asynchrone du modèle insulaire consiste à intégrer dans un AE un gestionnaire de migrations. Dans ParadisEO, celui-ci est obtenu par spécialisation du *Helper* fourni implémentant le critère de continuation car le processus de migration intervient lors de l'évaluation de ce critère. La spécialisation de ce *Helper* consiste à y intégrer par composition les différents paramètres du modèle (critère de décision de migration, politique de remplacement, etc.) et un canal de communication d'individus (sur machines à mémoire distribuée). Pour exploiter le modèle insulaire, l'utilisateur a juste à l'instancier en lui passant ses différents paramètres qui sont eux-mêmes des *Helpers* fournis instanciés auparavant. La version synchrone du modèle est presqu'identique à la version asynchrone. La différence se situe au niveau du critère de migration i.e. un intervalle de migration est fourni à la place du critère de migration.

- Le modèle d'évaluation parallèle de la population est implémenté dans ParadisEO selon le paradigme *Fermier-Travailleurs* en mode synchrone et asynchrone. Le fermier contrôle le processus d'évolution et gère une file d'individus à évaluer. Dans le mode asynchrone, les évaluations sont effectuées suivant le modèle de *vol de cycles*. Les demandes d'individus à évaluer sont effectuées par simples copies mémoire ou par échange de messages sur les machines à mémoire respectivement partagée et distribuée. Dans le dernier cas, différents canaux doivent être instanciés notamment les canaux nécessaires à l'échange d'individus et leurs valeurs de qualité. Dans ParadisEO, l'implémentation du modèle est réalisée sous forme de *Helper* fourni nécessitant à l'instanciation un *Helper* requis en l'occurrence la fonction d'évaluation.

- Le modèle d'évaluation parallèle d'une solution est également basé sur le paradigme *Fermier-Travailleurs*. Une solution est émise vers différents sites évaluateurs générant des valeurs de qualité partielles, ensuite collectées et agrégées. Le modèle est implémenté sous forme de *Helper* fourni nécessitant la fonction d'évaluation partielle fonction et la fonction d'agrégation de qualités partielles.

3.2.3.2 Parallélisme des métaheuristiques à solution unique

Nous rappelons que trois modèles parallèles de métaheuristiques à solution unique ont été identifiés : le modèle parallèle multi-départ, le modèle d'évaluation parallèle du voisinage, et le modèle d'évaluation parallèle d'un mouvement. Les trois modèles sont implémentés et fournis dans ParadisEO. Par conséquent, ils sont exploitables de manière transparente à l'utilisateur. L'implémentation du premier modèle est triviale dans sa version indépendante. La conception du troisième modèle est similaire à celle du modèle d'évaluation parallèle d'une solution présenté précédemment.

Le modèle d'évaluation parallèle du voisinage est implémenté dans ParadisEO suivant le paradigme *Fermier-Travailleurs*. Le fermier assure le contrôle de la recherche locale. A chaque itération, il duplique la solution courante entre les différents travailleurs explorant chacun un voisinage partiel et se met en attente des résultats d'exploration. Les communications entre le fermier et les travailleurs sont assurées par différents canaux destinés entre autres à l'échange de mouvements et leurs valeurs de coût associées.

3.2.3.3 Hybridation de métaheuristiques

Nous rappelons que quatre formes d'hybridation ont été identifiées dans le chapitre précédent : *Hybride Bas-niveau Relais (HBR)*, *Hybride Bas-niveau Co-évolutionnaire (HBC)*, *Hybride Haut-niveau Relais (HHR)* et *Hybride Haut-niveau Co-évolutionnaire (HHC)*. L'hybridation HBR consiste à encapsuler une métaheuristique dans une autre métaheuristique à base de solution unique. Son implémentation a nécessité l'introduction d'un nouveau composant dédié à l'application de traitements particuliers à la solution courante au terme de chaque itération d'une recherche locale. Ce composant intègre un opérateur de transformation de solution appliqué à la solution courante si un critère d'hybridation est avéré.

L'hybridation HBC se traduit par le remplacement des opérateurs de variation standard par des heuristiques. Ce type d'hybridation peut être exploité simplement dans ParadisEO car toutes les métaheuristiques à solution unique ont la même sémantique que les opérateurs de mutation. Elles peuvent donc se substituer naturellement à ces opérateurs. Le schéma HHR est également simple d'utilisation car le solveur hybride peut être construit par simple indication de l'ordonnancement des métaheuristiques hybridées. L'hybridation HHC est basée sur le déploiement simultané de plusieurs métaheuristiques coopératives à l'instar du modèle insulaire. Sa mise en oeuvre nécessite l'instanciation de plusieurs canaux de communication pour la gestion de la coopération.

3.3 ParadisEO sur grilles

Une plate-forme d'optimisation pour environnements non dédiés à grande échelle doit comporter, en plus de la couche algorithmique pour l'optimisation, une couche intergicielle pour la gestion des communications et des ressources volatiles. Cette couche logicielle (ou intergiciel) doit fournir à la fois une interface pour l'exploitation de l'infrastructure et une interface de programmation des grilles. Deux approches peuvent être utilisées pour disposer de cet intergciel : en développer un nouveau [Alb01] ou réutiliser un intergiciel existant. Nous avons adopté la deuxième approche pour la gridification de la plate-forme ParadisEO en la couplant avec Condor [LLM88] et MW [LKGY00]. Ces derniers fournissent les services et interfaces nécessaires pour la programmation d'applications sur environnements non dédiés à grande échelle. Le résultat du couplage est appelé *ParadisEO-CMW*. Avant de présenter l'architecture de ParadisEO-CMW, nous allons décrire succinctement Condor et la plate-forme MW.

3.3.1 Le système Condor

Condor [LLM88] est un intergiciel de type *High-Throughput Computing* ou *HTC* pour la gestion d'environnements non dédiés hétérogènes et multi-utilisateurs. Il permet de gérer des ressources volatiles en décidant de leur disponibilité sur la base de leur charge CPU moyenne et de l'information sur l'utilisation récente de certains périphériques comme le clavier et la souris. Cette information sur la disponibilité variable des ressources permet à Condor de fournir un ordonnancement adaptatif. En effet, les tâches d'une application sont réparties entre les machines disponibles et sont migrées vers d'autres machines dès que celles-ci ne sont plus disponibles en raison de leur panne ou de leur réquisition par leurs propriétaires. En outre, Condor fournit deux services sophistiqués et importants : le *matchmaking* et le *checkpointing* [LBRT97]. Le premier service permet de mettre en correspondance les besoins des applications ordonnancées et les politiques d'allocation des ressources. Le second est dédié à la sauvegarde de l'état des tâches en cours d'exécution et à leur redémarrage après restauration de cet état après une éventuelle panne.

La figure 3.2 illustre comment Condor déploie et gère les tâches sur un réseau de stations. Différents démons sont déployés sur chacune des ressources selon leur nature. Le super démon *Master* démarre les démons *Startd*, *Collector*, *Negotiator*, et *Schedd* permettant respectivement de caractériser la ressource locale, de collecter les descriptions en résultant, d'assurer la correspondance d'une requête avec une des ressources disponibles (*matchmaking*), et enfin d'accepter la soumission de nouveaux travaux. Dans cette figure, on montre les étapes menant de la soumission d'une tâche à son accomplissement.

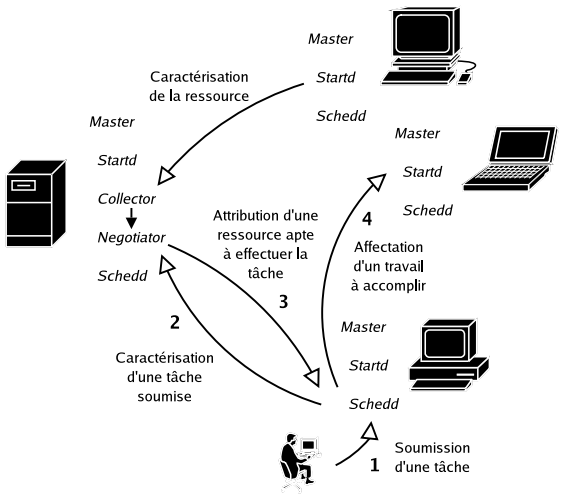

FIG. 3.2 – L'organisation du système Condor sur un réseau de stations.

Condor permet la gestion de grilles de calcul englobant des ressources réparties sur plusieurs domaines administratifs. La technologie *Flocking* autorise la coopération de plusieurs infrastructures où ce système est installé. Condor associé à Globus intègre aussi la plupart des méthodologies et protocoles émergeant des grilles. Ainsi, Condor-G [TTL02] est totalement inter-opérable avec des ressources distribuées à grande échelle et administrées par Globus.

3.3.2 La plate-forme MW

Master/Worker (ou *MW*) [LKGY00] est une plate-forme dédiée à la conception d'applications parallèles sur grilles reposant sur le paradigme *Fermier-Travailleurs*. Elle a été développée dans le cadre du projet *Metacomputing Environment for Optimization (MetaNEOS)* [The00], associant des chercheurs en optimisation numérique et quelques membres des équipes impliquées dans le développement des plates-formes Globus et Condor. MW s'inscrit dans la lignée des environnements tels que CARMI [PL95], Piranha [GK91] ou MARS [TGHK99]. A l'instar de ces environnements, MW vise à offrir un environnements de programmation parallèle adaptative, complet mais simple d'utilisation, tenant compte des caractéristiques intrinsèquement liées aux environnements non dédiés à grande échelle, tels que ceux exploités par le système Condor.

MW fournit un ensemble de classes abstraites C++ aidant à la programmation et la gridification d'applications (*Interface de programmation ou API*) et au développement d'infrastructures de type grilles de calcul (*Interface d'infrastructure*). Gridifier une application avec MW consiste à réécrire un nombre réduit de fonctions virtuelles. L'interface d'infrastructure permet l'accès à la gestion des communications entre fermier et travailleurs et des ressources (identification et caractérisation des ressources, détection des pannes, exécution d'appels à distance, etc.).

Indépendemment du choix de l'infrastructure sous-jacente, la gridification d'une application avec MW requiert de compléter trois classes : *MWDriver*, *MWTask* et *MWWorker* illustrées en UML par la figure 3.3. *MWDriver* joue le rôle de fermier et gère l'ensemble des travailleurs participant au (ou se retirant du) calcul distribué, et (ré)assigne les travaux à accomplir par ces derniers. *MWTask* désigne une unité de travail et encapsule ses données et les résultats renvoyés par un travailleur. Dans la classe dérivée, les fonctions d'échange des données et résultats entre fermier et travailleurs doivent être ré-implémentées. *MWWorker* boucle principalement en attente d'une tâche *MWTask* à exécuter. Pour chaque tâche reçue, il exécute le traitement requis et renvoie les résultats au fermier avant de se mettre à nouveau en attente. Le diagramme UML *MWRMComm* désigne une abstraction des mécanismes de communication et de gestion de ressources. Différentes implémentations sont fournies utilisant une librairie d'E/S (*MWFileRC*), une librairie de sockets (*MWSocketRC*), Globus (*MWGlobusRC* en perspective) et Condor-PVM (*MWCondorPVMRC*).

3.3.3 Architecture de ParadisEO-CMW

Un des objectifs conceptuels majeurs de MW est d'assurer à ses utilisateurs une programmabilité maximum i.e. leur faciliter le portage de leur code existant sur la plate-forme. Le couplage de ParadisEO avec Condor-MW est en effet grandement facilité par les interfaces d'infrastructure et de programmation fournies par MW. De plus, ce couplage est également rendu plus simple par le fait que les deux plates-formes sont implémentées en C++. L'architecture multi-couche de ParadisEO-CMW est illustrée par la figure 3.4. De haut en bas, le premier niveau désigne l'application faisant usage de plate-forme afin de résoudre des problèmes d'optimisation combinatoire. Le deuxième niveau représente l'ensemble des solveurs offerts par la plate-forme encapsulant des métaheuristiques mono et

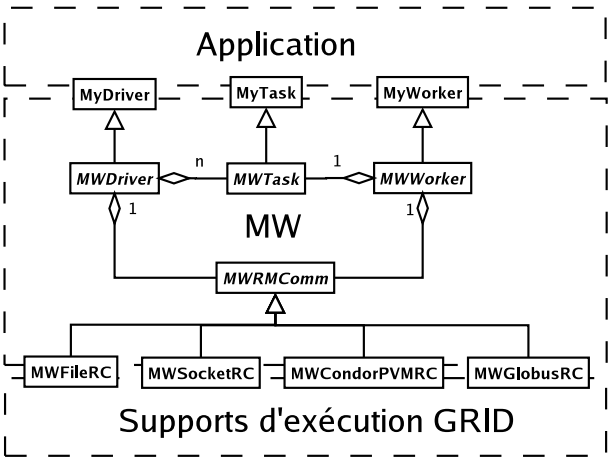

FIG. 3.3 – Représentation UML des principaux composants impliqués dans MW.

multi-objectifs. Le troisième niveau fournit des interfaces pour la programmation sur grilles et l'accès à l'infrastructure Condor. Le quatrième et dernier niveau offre des services de communication et de gestion de ressources.

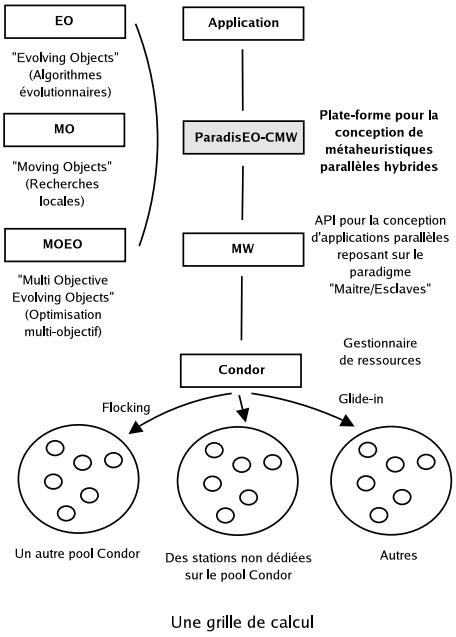

FIG. 3.4 – Architecture de ParadisEO-CMW.

Du point de vue de l'implémentation, les différents modèles parallèles ont été repensés et ré-implémentés en utilisant le seul paradigme offert par MW i.e. Fermier-Travailleurs. En effet, les classes abstraites fournies dans MW notamment *MWDriver*, *MWTask* et *MWWorker* ont été implémentées et sont respectivement désignées par *eoDriver*, *eoTask* et *eoWorker*. L'utilisateur de ParadisEO-CMW n'aura pas à les ré-implémenter, et aura juste à les instancier. Les solveurs, pouvant être parallèles, sont encapsulés dans une classe appelée

eoAlgo. Les instances de cette classe représentent les travaux (ou *jobs*) soumis à Condor. Nous décrirons ci-dessous succinctement le rôle de chacune des classes ré-implémentées dans le déploiement des solveurs d'optimisation.

- *eoTask* désigne une tâche à accomplir, et se définit simplement par un ensemble de données amenées à être communiquées et par son propriétaire (*i.e.* une instance *eoAlgo*). Deux classes caractérisent une tâche selon que cette dernière est traitée par le fermier ou par un travailleur : *eoDriverTask* et *eoWorkerTask*. Comme le montre la figure 3.5, elles se distinguent simplement par les opérations élémentaires de gestion des données empaquetées.

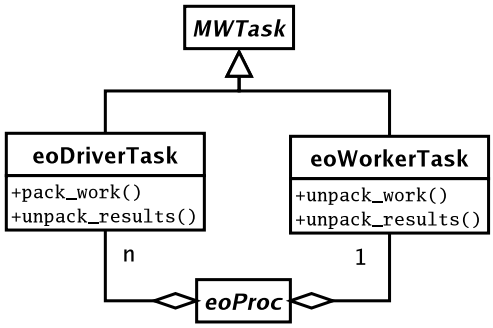

FIG. 3.5 – Une représentation UML des tâches *eoTask*.

- *eoDriver* constitue le gestionnaire principal, exécuté par le fermier. Son rôle est de gérer l'ensemble des solveurs déclarés, d'ordonnancer les tâches déployées par ces derniers sur un ensemble fluctuant de travailleurs. Il assure également la sauvegarde périodique de l'état du processus principal au niveau applicatif, et la production de statistiques au terme de l'exécution.

- *eoWorker* constitue le coeur de l'application exécutée sur chaque travailleur. Une instance de cette classe se met en attente d'une tâche à accomplir, extrait de celle-ci la clé (le numéro identifiant unique) d'un solveur (*job*) donné puis les données encapsulées, active cette procédure compte tenu de ces informations, et enfin retourne au fermier les résultats.

3.3.4 Tolérance aux pannes

Un problème crucial inhérent à l'exécution parallèle sur une grille (en raison de son caractère volatile) est la tolérance aux pannes. MW gère la volatilité des travailleurs en redéployant automatiquement les tâches interrompues sur de nouveaux travailleurs disponibles. Cette gestion de niveau système a l'avantage d'être transparente, cependant elle reste coûteuse en temps CPU et en espace de stockage. Par contre, MW prévoit un couple de primitives (à fournir) de repli/dépli de l'application permettant cette gestion au niveau applicatif et améliorant ainsi considérablement les performances. Ces primitives sont fournies dans ParadisEO-CMW et permettent de gérer la volatilité de manière transparente.

Ni cette solution ni celle offerte par Condor-MW ne peuvent cependant être appliquées au noeud central dans lequel le fermier supervise le déroulement de l'exécution. L'approche adoptée est l'implémentation d'un mécanisme de sauvegarde/restauration (ou *checkpointing*) transparent à l'utilisateur pour les deux classes de métaheuristiques. Il s'agit d'enregistrer essentiellement la ou les solution(s) courante(s), la ou les meilleure(s) solution(s) trouvée(s), le critère de continuation, et quelques paramètres additionnels contrôlant le processus d'évolution ou la recherche. Différentes politiques indiquant pour chaque métaheuristique la mémoire constituant sa progression à tout instant sont fournies. Celles-ci sont utilisées par défaut, néanmoins l'utilisateur peut fournir sa propre politique définissant les données à sauvegarder/restaurer.

3.4 Application et validation sur des problèmes industriels

3.4.1 Conception de réseaux cellulaires en téléphonie mobile

La conception (ou *design*) de réseaux cellulaires est l'un des problèmes majeurs de l'ingénierie des réseaux de télécommunications mobiles (GSM, UMTS, etc.). Le problème consiste à sélectionner un ensemble de sites parmi une liste d'emplacements géographiques candidats, à y placer et configurer des stations de bases (ou antennes) de manière à en optimiser à la fois le coût et la qualité de service (*QoS*) [GZBS86]. Nous allons d'abord décrire les données du problème et en donner une formulation. Ensuite, nous illustrerons sa résolution efficace avec ParadisEO en utilisant les différents modèles parallèles et mécanismes d'hybridation.

3.4.1.1 Formulation du problème

On considère une zone géographique discrétisée en un nombre fini d'emplacements potentiels de sites, définis par leurs coordonnés et leurs hauteurs au sol. On distingue trois types de points dits de *réception test* (*RTP*), de *service test* (*STP*) ou de *trafic test* (*TTP*). Les RTP désignent les points pour lesquels la puissance du champ radio est calculable. Les STP représentent les points pour lesquels un service minimum doit être assuré afin de permettre l'établissement de la communication (contrainte de couverture [FMB96]). Les points TTP sont associés à une quantité de trafic potentiel (exprimée en Erlang) estimée par l'opérateur. D'autre part, les antennes à placer sur les sites peuvent être *sectorielles* ou *omnidirectionnelles* pouvant émettre respectivement dans un secteur seulement ou dans toutes les directions. Les principaux paramètres d'une antenne sont : le type (omnidirectionnel, sectoriel large ou étroit), la puissance d'émission, l'inclinaison verticale (*tilt*), et enfin la direction horizontale (*azimut*) dans le cas d'une antenne sectorielle. Chaque antenne (station de base ou *BTS*) est caractérisée par sa cellule qui désigne l'ensemble des RTP ayant cette antenne comme meilleure serveuse.

Le problème de conception de réseaux mobiles est un problème à trois objectifs et deux contraintes. Ces objectifs sont : (1) la minimisation du coût d'installation du réseau désigné par le nombre de ses sites ; (2) la maximisation du trafic écoulé estimé à partir des TTP ; et, (3) la minimisation des interférences dans les zones de chevauchement des cellules. Les

contraintes considérées sont la couverture et le *relais* (ou *handover*). La première consiste à assurer que tous les STP reçoivent un champ radio supérieur au seuil de sensibilité du mobile. La deuxième contrainte permet d'assurer la continuité de la communication d'une cellule à une autre d'un mobile en déplacement. Le calcul des objectifs et contraintes est basé sur un modèle de propagation des ondes électromagnétiques de type *espace-libre* [RC98, LGT99]. Le modèle prédit par simulation l'affaiblissement du signal émis par une antenne et reçu par un mobile représenté par un RTP.

Une présentation plus formelle du problème est donnée dans les thèses [Meu02, Cah05]. Le problème est NP-difficile et complexe en pratique [PS82, VH00]. Pour avoir une idée de la complexité, considérons le tableau 3.2 et les instances fournies par France Telecom R&D *Arno 1.0* (bande autoroutière de 250 sites), et *Arno 3.0* et *Arno 3.1* (zones urbaines de 568 et 747 sites). Pour un nombre $|\mathcal{L}|$ de sites candidats, la taille de l'espace de recherche pour une instance donnée est de :

$$(|P_s| \times |\beta| + (2 \times |P_s| \times |\beta| \times |\delta|)^3)^{|\mathcal{L}|}$$

i.e. $\backsimeq 4,8 \times 10^{2558}$, $8,4 \times 10^{6494}$ et $5,5 \times 10^{8541}$ pour les instances respectives *Arno 1.0, 3.0* et 3.1 ! Cette complexité exponentielle justifie l'utilisation du parallélisme pour la résolution du problème.

Paramètre	**Notation**	**Domaine**	**Discrétisation**	**Dimension**
Puissance	P_s	[26, 55]	2 dBm	15
Azimuth	δ	[0^o, 360^o]	10^o	36
Tilt	β	[-15^o, 0^o]	3^o	6

TAB. 3.2 – Paramètres d'ingénierie discrétisés d'une antenne.

3.4.1.2 Résolution parallèle hybride avec ParadisEO

Nous avons proposé un AG multi-objectif utilisant une archive Pareto pour la résolution du problème dont les individus sont des réseaux représentés par des listes de sites. Pour chaque site, la configuration de la (ou des) antenne(s) est codée. Différents types de crossover et de mutation ont été proposés en jouant sur les paramètres à croiser ou à muter. Différentes stratégies de sélection et remplacement ont été également définies. D'autre part, les différents modèles parallèles et un mécanisme d'hybridation avec la recherche locale ont été adaptés au problème. Toute la modélisation du problème ainsi que sa résolution parallèle hybride ont été intégrées dans ParadisEO pour constituer une plate-forme dédiée appelée *DEMARNO* livrée à France Telecom R&D.

Le modèle parallèle hybride hiérarchique implémenté avec ParadisEO est illustré par la figure 3.6.

– *Le modèle insulaire* : Le modèle insulaire comprend 4 AG coopératifs en anneau, constitués chacun de 100 réseaux générés aléatoirement. La technique d'élitisme a été mise en oeuvre en sauvegardant les solutions Pareto générées au cours du processus d'optimisation dans une archive locale. Dans la phase de sélection, l'AG considère

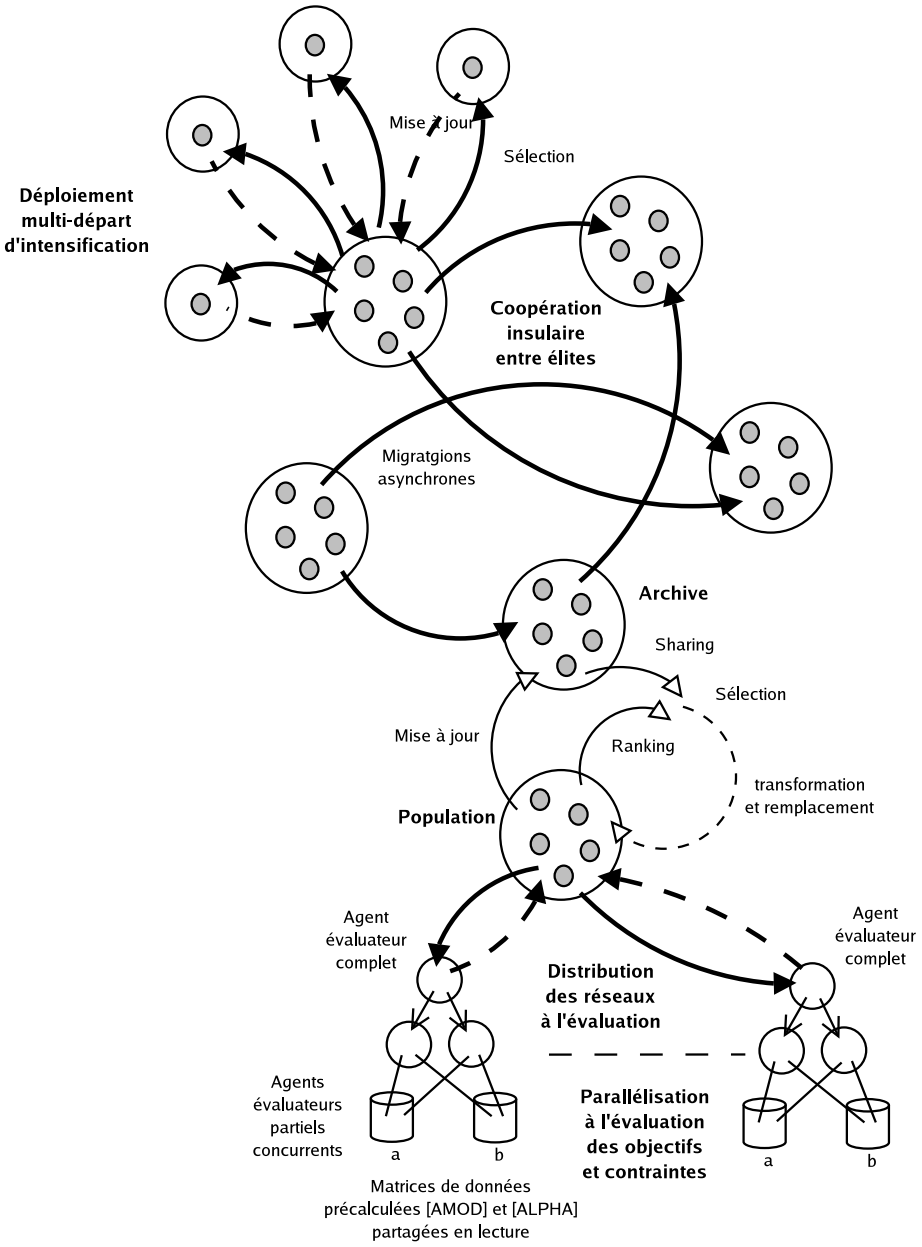

FIG. 3.6 – Parallélisme hiérachique et hybride.

à proportion égale les solutions de la population courante (NSGA-II) et celles de l'archive (*Sharing*). Le remplacement des nouvelles solutions est générationnel (*i.e.* substitution de la population courante par la nouvelle génération). Des migrations asynchrones sont déclenchées toutes les 10 générations, et se composent de 10 réseaux sélectionnés depuis l'archive locale et déterminés par le partage. Les émigrants seront intégrés dans l'archive associée à l'île cible.

- *L'hybridation HHR* : Chaque AG est hybridé avec une méthode de recherche locale suivant le schéma *HHR*. Au terme de chaque génération, un ensemble de 100 réseaux est sélectionné depuis l'archive locale parmi les plus *isolés* sur la frontière Pareto. Un déploiement multi-départ est réalisé, basé sur l'exécution concurrente et indépendante d'une recherche locale (déterminée aléatoirement parmi les méthodes implémentées) appliquée à chacune de ces solutions. A leur terminaison, un ensemble de solutions Pareto optimales est ainsi localement constitué, ensuite réintégré dans l'archive Pareto originale.

- *L'évaluation parallèle hiérarchique des réseaux* : Deux modèles ont été mis en oeuvre afin de distribuer les calculs : la distribution synchrone des réseaux et la parallélisation de leur évaluation. L'évaluation de la fonction multi-objectif et des contraintes associés à chaque réseau est effectuée en parallèle synchrone, en partitionnant le domaine géographique (espace opérationnel). Chaque noeud décompose la zone de travail, générant les champs radio dans sa partition, puis calculant des valeurs partielles de trafic, interférences, couverture, etc. Ces résultats sont alors collectés puis agrégés en une évaluation complète.

La figure 3.7 illustre la convergence progressive de l'union des frontières Pareto composant les archives associées aux AG du modèle coopératif insulaire. En effet, les courbes représentent le front agrégé après 0, 10, 50, 100, 800 et 1600 génération(s).

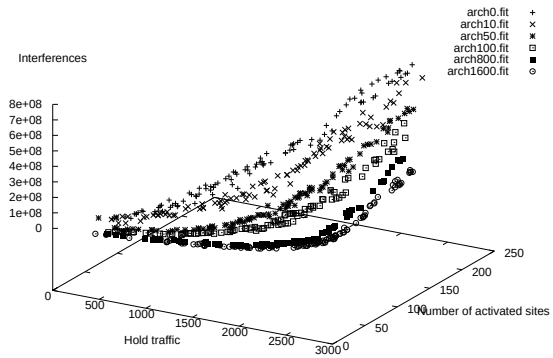

FIG. 3.7 – Illustration du phénomène de convergence de l'archive Pareto (Instance Arno 1.0).

Pour évaluer plus finement la convergence de ce modèle, la notion de *distance générationnelle* (définie dans le chapitre précédent) est utilisée pour mesurer la distance entre

un front et un autre qui le domine complètement [VL00a]. Cette mesure ne peut donc être utilisée pour mesurer la distance entre deux fronts quelconques. Elle permet d'évaluer la progression d'un algorithme en fonction du nombre d'itérations. Dans ce cas, le front ne peut que s'améliorer. La figure 3.8 mesure au terme de chaque génération la distance générationnelle de l'union des archives associées aux AG coopératifs avec celle des archives obtenues à partir des populations initiales. La convergence a nécessité l'accomplissement de 1700 générations. Afin d'aboutir à ce résultat, une durée cumulée de 7200 heures CPU ont été nécessaires sur la grappe Icluster2 de Grenoble.

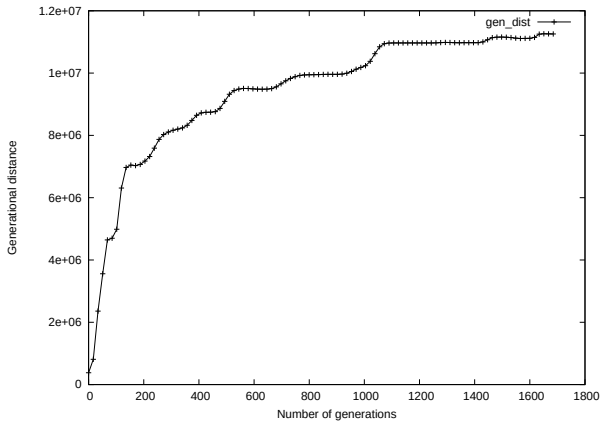

FIG. 3.8 – Distance générationnelle de l'archive Pareto avec l'archive initiale.

3.4.2 Sélection d'attributs en fouille de données spectroscopiques

La fouille de données est un processus de découverte dans les bases de données de modèles de prédiction nouveaux, précis, utiles et faciles à comprendre. Ces bases de données sont souvent denses signifiant que leurs schémas sont constitués d'un grand nombre d'attributs. La prise en compte de tous les attributs pour la construction de modèles de prédiction pose deux problèmes majeurs : des études théoriques et expérimentales [Zak99] ont montré que l'exploration des données est de complexité temporelle exponentielle en nombre d'attributs, donc très coûteuse en temps CPU pour des bases de données denses ; Un nombre important d'attributs est *non pertinent* et/ou *redondant* rendant plus difficile la compréhension des modèles de prédiction extraits. Dans la littérature, différentes définitions de la pertinence ont été proposées. Dans ce travail, nous considérons qu'un attribut est pertinent si son élimination provoque une perte significative de précision du modèle extrait. D'autre part, un attribut est dit redondant s'il est corrélé avec d'autres attributs. Sa prise en compte dans le modèle n'apporte aucune information supplémentaire aidant à la compréhension de celui-ci.

Dans [MCTD02], nous avons proposé une taxinomie et un état de l'art des approches utilisées dans la littérature pour résoudre ces deux problèmes de performance. Notre approche de résolution est basée sur la conjugaison de plusieurs approches, notamment la sélection d'attributs, l'optimisation combinatoire, le multi-threading et le parallélisme sur grilles de calcul. La sélection d'attributs [LM98] est une approche permettant de réduire

la dimension du problème. Il s'agit de sélectionner M attributs pertinents et non redondants parmi les N attributs du problème sans dégrader la précision du modèle extrait. Le problème de sélection optimale d'attributs a été montré NP-difficile [NF77].

Plusieurs techniques ont été proposées dans la littérature pour la résolution du problème de sélection d'attributs [NF77, VDK92, DL97, LM98]. Ces techniques sont basées sur trois catégories d'approches : les approches *filtrantes* [AT91, LS96], les approches *enveloppantes* [KJ97], et les approches hybrides [SN02]. Les premières considèrent deux étapes distinctes dans le processus de construction du modèle : d'abord la sélection d'attributs, et ensuite la construction du modèle. Dans ce cas, la sélection n'est pas basée sur la précision du modèle extrait mais sur d'autres critères tels que la corrélation entre attributs. Au contraire, dans les approches enveloppantes la sélection et la construction du modèle sont mêlées. Le processus global se déroule en plusieurs étapes, et à chaque étape une sélection est effectuée, un modèle prédictif est construit et sa qualité de prédiction est calculée. La qualité de la sélection est améliorée d'étape en étape, et le processus s'arrête lorsqu'aucune amélioration n'est avérée. Nous nous sommes intéressés aux algorithmes génétiques appliqués à la sélection d'attributs en spectroscopie proche infrarouge (PIR). Une modélisation de la sélection d'attributs sous forme de problème d'optimisation combinatoire est présentée dans la suite de ce chapitre. Une approche de résolution et sa parallélisation sur grilles avec ParadisEO-CMW est décrite. Les résultats expérimentaux obtenus sont également présentés et interprétés.

3.4.2.1 Présentation et données du problème

Le problème d'analyse de spectres auquel nous nous sommes intéressés consiste à construire un modèle de prédiction de la concentration d'un composant donné (e.g. le sucre) dans une matière organique (e.g. la betterave). Comme le montre la figure 3.9, une source lumineuse envoie des radiations PIR (N longueurs d'ondes) sur un échantillon de la matière organique en question. Un spectre d'absorption est ainsi recueilli à l'aide de capteurs. En outre, la concentration du composant étudié dans l'échantillon est mesurée par analyse chimique. L'expérience est renouvelée un certain nombre de fois permettant ainsi d'obtenir les données du problème. Il s'agit d'un lot d'échantillons, chacun constitué de N absorbances (valeurs réelles) et une valeur (réelle) de concentration. Pour vérifier la précision du modèle (une fois construit), les échantillons sont répartis en deux lots : un lot de calibration et un lot de validation, permettant respectivement de construire et valider le modèle.

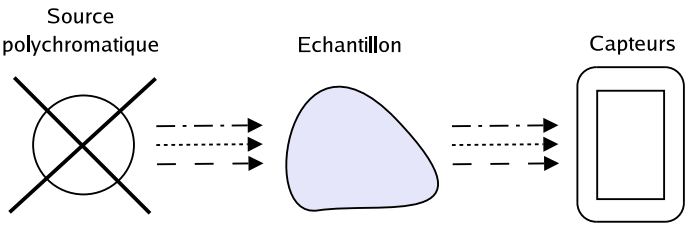

FIG. 3.9 – Constitution d'un spectre par mesure de l'absorbance lumineuse sur le domaine PIR.

Nous avons en particulier considéré une instance du problème où la matière organique et le composant désignent respectivement la betterave et le sucre. Le jeu de données nous a été fourni par le *Laboratoire de spectroscopie et raman de l'Université de Lille1 (LASIR)*. Les lots de validation et de calibration contiennent respectivement 957 et 1024 échantillons. Chaque échantillon est constitué de 1020 absorbances (correspondant à 1020 longueurs d'onde) et une valeur de concentration mesurée par analyse chimique.

Selon la loi de Beer Lambert, la concentration d'un composant est fonction (combinaison linéaire) des absorbances correspondant aux longueurs d'onde qu'il peut absorber. Le problème de prédiction consiste donc à estimer cette fonction (modèle). Comme le problème est linéaire, il est souvent traité avec des méthodes statistiques telles que la méthode des moindres carrés partiels ou *Partial Least Square (PLS)* [WRWD84]. La métrique permettant de calculer la précision d'un modèle est l'erreur *RMS*, obtenue à partir des concentrations prédites (calculées en utilisant le modèle de prédiction) et de celles réelles qui nous sont connues. L'erreur *RMS*, correspondant à l'écart-type des erreurs de prédiction peut être formulée de la manière suivante :

$$Erreur\ RMS = \sqrt{\frac{\sum_{i=1}^{n}(\widehat{c_i} - c_i)^2}{n}}$$

n désigne le nombre d'échantillons de validation. Pour un échantillon i, c_i et $\widehat{c_i}$ désignent respectivement ses concentrations réelle et prédite.

3.4.2.2 Résolution parallèle sur grilles avec ParadisEO-CMW

L'analyse statistique des données de notre problème a montré une forte corrélation d'une grande partie des longueurs entre elles, et avec la concentration associée. Des courbes illustrant ces mesures de corrélation sont présentées dans [Cah05]. Ce constat nous a conduit à penser que bien que la méthode PLS soit efficace, sa combinaison avec une sélection d'attributs peut davantage améliorer sa performance. L'idée est de sélectionner les longueurs d'onde les moins corrélées entre elles et les plus corrélées avec la concentration. Le problème peut être ainsi formulé comme suit : un échantillon de données peut être vu comme un vecteur comprenant les valeurs d'absorbance associées à N longueurs d'onde (ω_1, ω_2, ..., ω_N) et une valeur de concentration réelle c. Si f désigne le modèle alors $c = f(\omega_1, \omega_2, ..., \omega_N)$. Le problème de sélection d'attributs consiste alors à choisir M (tel que $M \leq N$) longueurs d'onde pertinentes et non redondantes parmi les N longueurs d'onde du problème.

L'approche que nous avons proposée est enveloppante combinant un AG avec la méthode statistique PLS. Les individus de la population de l'AG sont des chaînes binaires de longueur $N = 1020$. Le i^{eme} bit d'un individu est mis à 1 (resp. à 0) si la i^{eme} longueur d'onde est sélectionnée (resp. n'est pas sélectionnée). A chaque génération, la méthode PLS est appliquée à chaque individu en considérant la sélection d'attributs correspondante. Pour chaque individu, un modèle prédictif est ainsi construit et une erreur de prédiction RMS est calculée. Cette erreur représente la valeur de qualité (ou *fitness*) associée à l'individu. Les opérateurs de variation sont les opérateurs classiques, c'est à dire le crossover et la mutation.

Le choix d'une sélection d'attributs s'avère délicat en considérant d'autres critères que la qualité du modèle de prédiction lui-même. L'approche filtrante est donc difficilement envisageable. L'approche enveloppante est plus adaptée à ce type de problème et s'avère plus fiable, cependant celle-ci pose un problème de faisabilité. En effet, le nombre d'attributs est élevé (1020) et donc le nombre de sélections candidates (2^{1020}) l'est également. De plus, le coût d'évaluation de chaque sélection est exorbitant car celle-ci est réalisée par régression PLS, elle-même basée sur un calcul matriciel intensif. L'utilisation du parallélisme à grande échelle s'avère indispensable pour supporter le coût de "l'hybridation" de l'AG avec la méthode PLS.

Deux modèles parallèles fournis par la plate-forme ParadisEO-CMW ont été exploités simultanément constituant un parallélisme hiérarchique à deux niveaux : le modèle insulaire et le modèle d'évaluation parallèle de la population (voir Figure 3.10). A l'exception de la fonction d'évaluation (ici l'erreur RMS calculée après application de la PLS) car elle est spécifique au problème, toutes les classes nécessaires à l'application sont fournies. L'effort de codage a été ainsi minimisé car il a suffi d'instancier ces classes et de coder la méthode PLS pour obtenir une application complète.

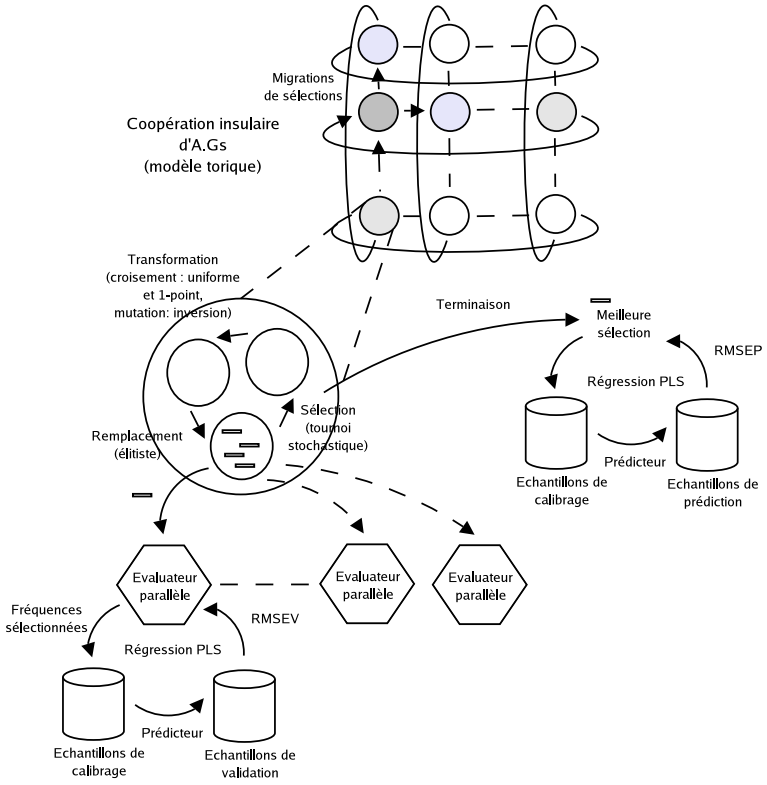

FIG. 3.10 – Un modèle coopératif insulaire en tore avec phase d'évaluation distribuée.

L'application a été déployée et expérimentée sur le réseau d'enseignement de Polytech'Lille durant les jours de travail. Ce réseau est hétérogène et contient 122 PC (sous Debian) dont la disponibilité est variable et offre une puissance de calcul de 29, 4 GFLOPS. Les résultats expérimentaux obtenus permettent de tirer plusieurs enseignements. D'abord,

l'utilisation de la sélection d'attributs (avec un seul AG) a permis d'améliorer la précision du modèle prédictif construit d'environ 42%. Le déploiement du modèle insulaire a permis de porter cette amélioration à environ 45% avec 4 îles en anneau, et à environ 48% avec 16 îles coopérant suivant une topologie torique. La sélection obtenue est composée de 122 attributs sur 1020 longueurs d'onde candidates. De plus, cette élimination d'environ 88% des fréquences permet une accélération de l'analyse de tout nouvel échantillon de betterave i.e. le temps de traitement est divisé par un facteur de 7.5.

Enfin, le tableau 3.3 montre quelques statistiques mesurées pendant l'exécution de l'AG coopératif insulaire torique. On peut constater que ParadisEO-CMW permet l'utilisation conjuguée des différents modèles parallèles (ici 2) générant suffisamment de travail (1600 évaluations simultanées pour 16 AG de 100 individus) pour justifier l'utilisation d'une grille de calcul. Ce parallélisme est exploité avec une efficacité de 82% pour une disponilbilité moyenne de 78 sur 115 noeuds travailleurs. Comme indiqué dans le chapitre précédent, l'efficacité mesure le rapport entre la durée d'exécution cumulée et celle de la disponibilité totale des machines durant le calcul. De plus, 10 heures ont suffi pour exécuter l'application pour une durée d'exécution cumulée de 27 jours.

Nombre de NT	122
Durée d'exécution réelle	36953 s. (10 heures)
Durée d'exécution cumulée	2363571 s. (27 jours)
Nombre moyen de NT actifs	115
Nombre moyen de NT disponibles	78
Efficacité parallèle	0.82

TAB. 3.3 – Statistiques mesurées à l'exécution de l'A.G. coopératif torique

3.5 Conclusion

L'étude comparative des plates-formes logicielles dédiées aux métaheuristiques parallèles hybrides réalisée dans le chapitre 1 a révélé les limites de celles-ci en termes d'utilité, en particulier en matière de parallélisme et d'hybridation sur grilles. En outre, la majorité de ces plates-formes ne sont plus maintenues. C'est pourquoi, nous avons proposé la plate-forme ParadisEO qui fournit un large éventail de composants réutilisables et extensibles : les algorithmes évolutionnaires mono et multi-objectifs, les méthodes de recherche locale, les différents modèles parallèles et mécanismes d'hybridation associés dédiés à différents types d'architectures notamment les grilles de calcul. La réutilisabilité et l'extensibilité de ces composants proviennent à la fois de la séparation réalisée à leur conception entre la partie générique et la partie dépendante du problème traité, et à l'approche objet utilisée pour les coder.

L'utilisation de librairies standard telles que Pthreads, PVM et MPI pour l'implémentation des différents modèles parallèles rend ces modèles portables et déployables sur des architectures parallèles aussi bien à mémoire distribuée qu'à mémoire partagée. Le couplage de la plate-forme avec Condor-MW rend leur déploiement possible sur des grilles de

calcul. Pour gérer la disponibilité variable des machines, nous avons proposé un mécanisme sauvegarde/restauration efficace et indépendant des problèmes traités. A notre connaissance, un tel mécanisme n'a jamais été proposé. L'exploitation des différents modèles et du mécanisme de sauvegarde/restauration associé est totalement transparente à l'utilisateur.

ParadisEO a été validée sur différents problèmes académiques tels que le problème du voyageur de commerce et le problème de coloration de graphes, mais aussi sur des applications réelles. Dans ce chapitre, nous avons présenté deux applications : la conception de réseaux cellulaires et la fouille de données en spectroscopie proche infrarouge. Ces applications nous ont permis de proposer une modélisation des problèmes associés et une approche pour leur résolution. Elles nous ont également permis de valider tous les composants fournis par ParadisEO en prouvant expérimentalement leur efficacité. L'application de conception de réseaux mobiles a montré qu'on peut exploiter simultanément tous les modèles parallèles identifiés en combinaison avec les mécanismes d'hybridation. Le modèle hiérarchique obtenu, nécessitant une grille, peut être facilement et efficacement déployé avec ParadisEO-CMW. Par ailleurs, le modèle multi-départ est particulièrement intéressant pour la mise en oeuvre de l'hybridation HHR dans le contexte multi-objectif.

L'utilité, la réutilisabilité et l'extensibilité, la portabilité, l'exploitation transparente du parallélisme et de l'hybridation sur grilles et la disponibilité de ParadisEO sur le Web ont contribué à encourager le télé-chargement et l'utilisation de ParadisEO dans différents pays et dans différents domaines d'application (voir Annexe). Pour assurer la viabilité de la plate-forme, il faut en réaliser un guide d'adaptation. Les tutoriels disponibles sur le Web constituent une base de ce guide.

Pour augmenter l'utilité de ParadisEO, il est nécessaire de compléter la plate-forme avec des méthodes exactes mono et multi-objectifs parallèles et les mécanismes nécessaires pour leur coopération avec les métaheuristiques. Ces méthodes exactes et les modèles parallèles et schémas d'hybridation associés peuvent être directement intégrés à ParadisEO. Ils peuvent être également fournis par une autre plate-forme et accessibles à ParadisEO via une interface. Cette perspective constitue l'objet du chapitre suivant.

Chapitre 4

Vers la coopération sur grilles entre méthodes exactes et métaheuristiques parallèles

Le travail présenté dans ce chapitre rentre principalement dans le cadre de la thèse de Mohand Mezmaz. Cette thèse fait partie des activités du projet DOLPHIN de l'INRIA Futurs, et sera soutenue fin 2006. Elle porte sur les méthodes exactes parallèles et leur coopération avec les métaheuristiques parallèles dans le contexte des grilles de calcul.

Dans ce chapitre, nous présenterons d'abord un algorithme B&B parallèle basé sur le modèle de vol de cycles. Nous proposerons ensuite une approche de gridification de cet algorithme minimisant le coût de communication inhérent aux opérations de régulation de charge et au mécanisme de sauvegarde/restauration. Deux implémentations seront également présentées. La première est basée sur l'intergiciel XtremWeb, nous montrerons les limites de celui-ci pour l'optimisation multi-objectif sur grilles et proposerons un modèle de coopération inspiré du modèle Linda et intégré à l'intergiciel. La deuxième implémentation est basée sur le modèle RPC. Celle-ci a été utilisée pour résoudre pour la première fois de manière optimale sur une grille de plus de 2000 processeurs une instance mono-objectif du problème du Flow-Shop.

Nous proposerons également une métaheuristique parallèle hybride combinant le modèle insulaire parallèle, le modèle multi-départ parallèle et le mécanisme d'hybridation en mode HHC entre un algorithme évolutionnaire et une méthode de recherche locale. Les deux méthodes d'optimisation (exacte et métaheuristique) sont appliquées au problème du Flow-Shop bi-objectif. Elles sont déployées sur une grille de plus de 400 machines d'abord séparément, ensuite de manière hybride. Deux schémas d'hybridation ont été utilisés : haut-niveau relais (HHR) et haut-niveau co-évolutionnaire (HHR). Les résultats obtenus permettent de montrer l'apport du parallélisme et de l'hybridation sur grille.

4.1 Méthodes exactes parallèles sur grilles

Nous nous intéresserons au modèle d'exploration arborescente parallèle. Dans un premier temps, nous considérerons un algorithme B&B parallèle basé sur le modèle de *vol de cycles à grande échelle* de type *Coordinateur-Travailleurs*. Dans ce modèle, les communications sont toujours initiées par les travailleurs, ce qui permet la prise en compte naturelle de la caractéristique multi-domaine d'administration des grilles. L'initiative du déploiement de tâches à travers les pare-feu vient de l'intérieur des domaines d'administration. Après identification des limites de l'algorithme, nous présenterons l'amélioration apportée en vue de sa gridification. Les résultats expérimentaux obtenus seront présentés et analysés.

4.1.1 Un algorithme B&B parallèle basé sur le vol de cycles

Dans le modèle *Coordinateur-Travailleurs*, l'algorithme est composé d'un processus coordinateur et d'une liste de processus travailleurs (un processus par machine). Dans la suite, nous confondrons les processus coordinateur et travailleurs avec les machines sur lesquelles ils s'exécutent. Le rôle du coordinateur est de créer les processus travailleurs sur la grille et de répondre à leurs requêtes de demande de tâches d'exploration. D'autre part, le coordinateur maintient et met à jour la meilleure solution trouvée et la communique aux travailleurs. Chaque travailleur explore de manière *asynchrone* la liste des noeuds lui ayant été attribuée. Il retourne au coordinateur la solution courante si celle-ci est meilleure que la solution trouvée (stockée localement). Une fois sa liste de noeuds entièrement explorée, il renouvelle sa demande de travail au coordinateur. Après vérification, le coordinateur met à jour la meilleure solution trouvée. Dans le contexte multi-objectif, cette mise à jour est effectuée en fusionnant l'ancien front Pareto désignant la meilleure solution avec le nouveau front représentant la nouvelle solution trouvée. L'algorithme s'arrête lorsqu'il n'y a plus de travail en attente au sein du coordinateur.

Dans cet algorithme, une approche *collégiale* de répartition des sous-problèmes (noeuds de l'arbre) est utilisée. Chaque travailleur dispose localement de sa liste de noeuds reçue du coordinateur. La stratégie de placement des noeuds est mixte i.e. à la fois active et passive. En effet, périodiquement (la période est un paramètre utilisateur), un travailleur envoie (stratégie *active*) la moitié de sa liste courante au coordinateur, qui l'enverra ensuite à un travailleur disponible en réponse à sa demande (stratégie *passive*). La tolérance aux pannes des travailleurs et la détection de la terminaison de l'algorithme sont laissées à la charge de l'intergiciel.

Comme dans la majorité des implémentations parallèles de l'algorithme B&B, l'unité de travail est une liste de noeuds (pouvant être réduite à un seul noeud). Chaque opération de répartition de charge se traduit par deux communications : du travailleur (partageant son travail) au coordinateur et du coordinateur au travailleur disponible et demandeur de travail. Ces deux messages contiennent une liste de descripteurs (contraintes) de sous-problèmes. La taille de ces messages s'agrandit au fur et à mesure que l'exploration avance (on descend dans l'arbre). Ce constat induit un coût de communication et de stockage (au sein du coordinateur) exorbitant et pénalisant sur une grille de calcul (caractéristique liée

à la grande échelle). Il est donc primordial de réduire ce coût en proposant une autre approche de description des unités de travail. L'objectif de cette approche est non seulement de réduire le coût de stockage et de régulation de charge mais aussi de réduire le coût de gestion du problème de tolérance aux pannes. Cette nouvelle approche est présentée dans la section suivante.

4.1.2 Gridification de l'algorithme B&B

La nouvelle approche est basée sur le codage illustré par la figure 4.1. Ce codage consiste à numéroter les chemins de *l'arbre de base* suivant un parcours en profondeur d'abord. Un arbre de base est un arbre obtenu en appliquant à ses noeuds l'opérateur de branchement sans appliquer l'opérateur d'élagage. Le chemin le plus à gauche porte le numéro 0 et tous les noeuds d'un même chemin portent le même numéro. Une unité de travail (ou d'allocation) est alors décrite par un intervalle $[X, Y]$, où X et Y désignent les numéros associés à deux chemins donnés de l'arbre. Elle désigne l'ensemble des noeuds dont les numéros sont compris entre X et Y. Cette nouvelle description du travail nous a permis de proposer de nouvelles solutions pour la régulation de la charge, la tolérance aux pannes et la détection de la terminaison de l'exploration parallèle.

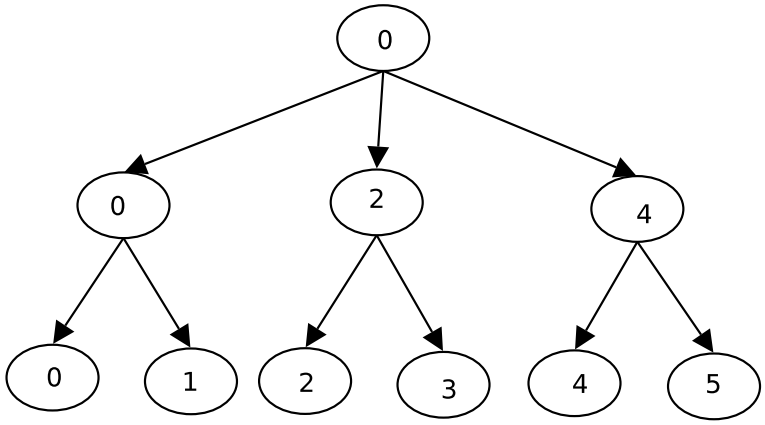

FIG. 4.1 – Codage de l'arbre basique

Initialement, le processus commençant l'exploration envoie au coordinateur l'intervalle $[0, F(n)]$ représentant la totalité de l'espace de recherche. $F(n)$ désigne le numéro du chemin le plus à droite de l'arbre associé à un problème de taille n. Cette fonction est spécifique au problème et doit donc être fournie par l'utilisateur. Par exemple, pour les problèmes de permutation de taille n, $F(n) = n!$. Au fur et à mesure de l'exploration, l'intervalle est découpé en intervalles de plus petite taille (unités de travail) à la demande des processeurs travailleurs suivant une stratégie *passive* de régulation de charge. Un seuil de découpe est défini pour éviter l'allocation d'unités de granularité trop fine. Le coordinateur garde une trace de tout intervalle alloué ou non. Lorsqu'un processeur i explorant l'intervalle $[X_i, Y_i]$ n'a plus de travail (i.e. Y_i devient supérieur à X_i après exploration), il adresse une requête de demande de travail au coordinateur. Ce dernier lui renvoie le plus grand intervalle non encore alloué s'il en existe. Dans le cas contraire, il "coupe" un intervalle $[X, Y]$ déjà alloué

en deux intervalles $[X, Y_1]$ et $]Y_1, Y]$. Le processeur demandeur aura la deuxième partie de l'intervalle car le processeur détenteur de cet intervalle a déjà commencé son exploration en partant de X. Cela permet de minimiser l'exploration redondante de mêmes noeuds de l'arbre.

Cette approche de description des unités de travail simplifie et réduit le coût de gestion de la disponibilité variable des machines par sauvegarde/restauration. Périodiquement, chaque processeur travailleur envoie au coordinateur un compte rendu de l'avancement de son exploration. Ce compte rendu contient les nouvelles extrémités de l'intervalle en cours d'exploration et un identifiant désignant de façon unique cet intervalle au sein du coordinateur. Si $[X_1, Y_1]$ et $[X_2, Y_2]$ désignent respectivement l'intervalle avant exploration (version du coordinateur) et en cours d'exploration (version locale du travailleur), le coordinateur met à jour son intervalle en appliquant la règle : $[X_1, Y_1] \wedge [X_2, Y_2] \Rightarrow [Max(X_1, X_2), Min(Y_1, Y_2)]$. La restauration d'un intervalle perdu est naturelle puisqu'il sera alloué à d'autres processeurs à leur demande. Un intervalle $[X, Y]$ disparaît du coordinateur dès que la condition $X > Y$ est vérifiée. Lorsque le coordinateur n'a plus d'intervalle, les demandes des travailleurs se soldent par des échecs synonymes de terminaison. Ainsi, la détection de la terminaison de l'exploration parallèle se fait de manière naturelle.

L'approche permet de réduire de manière significative le coût des communications et de stockage puisque seulement quelques octets suffisent à transférer ou stocker des unités de travail. Cependant, elle engendre un coût supplémentaire nécessaire à construire l'arbre à explorer à partir de son descripteur i.e. de l'intervalle associé. Dans notre approche, la reconstruction de l'arbre consiste à régénérer l'arbre en partant de sa racine (i.e. le problème à traiter à l'origine) en appliquant l'opérateur de décomposition et un opérateur d'élagage particulier. Cet opérateur est défini de la manière suivante : un noeud N de numéro n de l'arbre associé à un intervalle $[X, Y]$ est élagué s'il vérifie la condition : $n + F(N) <= X$. Cet élagage s'arrête lorsque le noeud de numéro X est atteint. Les noeuds de l'arbre régénéré sont placés dans la liste locale de sous-problèmes à traiter. La figure 4.2 illustre la procédure de régénération de l'arbre.

Les noeuds de l'arbre contiennent leurs numéros associés obtenus en appliquant le codage proposé. Les valeurs se trouvant à gauche de l'arbre désignent les valeurs de la fonction F associées à chaque niveau de l'arbre. Les noeuds gris, blancs et hachurés désignent respectivement des noeuds décomposés, élagués et placés dans la liste des noeuds à traiter. Dans l'exemple illustré par la figure, il s'agit de régénérer l'arbre associé à un intervalle dont la première valeur est 167. Par exemple, le noeud 155 a été coupé parce qu'il vérifie la condition $155 + 5 < 167$. Ce qui n'est pas le cas du noeud 165 puisqu'on a $165 + 5 > 167$.

4.1.3 Implémentation sur grilles basée sur un intergiciel Coordinateur-Travailleurs

Pour implémenter les deux versions parallèles de l'algorithme B&B, nous nous sommes intéressés aux intergiciels de type *Coordinateur-Travailleurs* tels que XtremWeb [Fed03], SETI@home [ACK+02] ou JNGI/JXTA [VNRS02]. Dans ces intergiciels (voir figure 4.3), à

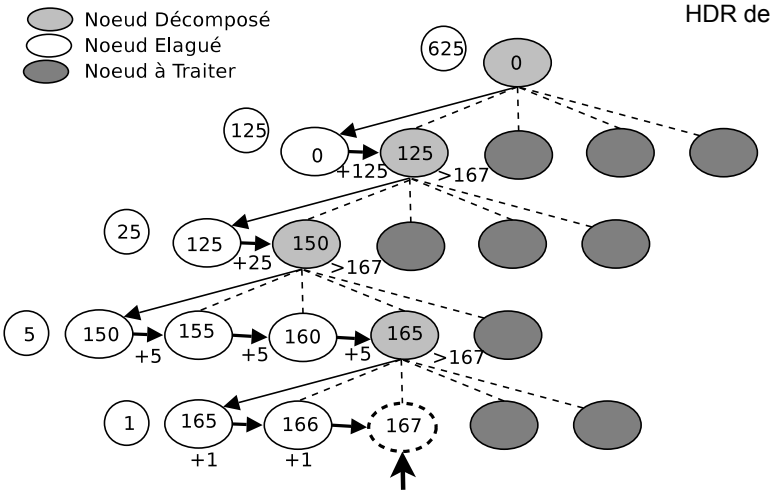

FIG. 4.2 – Régénération de l'arbre à partir du descripteur associé

l'exception de SETI@home qui est dédié à une seule application, des machines dites *clients* peuvent soumettre leurs applications à un coordinateur. Ces applications seront ensuite exécutées sous le contrôle du coordinateur par des machines autonomes dites *travailleurs* (ou *workers*). Les travailleurs demandent du travail au coordinateur suivant le modèle de vol de cycles à grande échelle. Ces intergiciels sont souvent dédiés aux applications dites *multi-paramètres* composées de tâches parallèles indépendantes. Les communications sont de type Coordinateur-Travailleurs et sont initiées par les travailleurs. De plus, très souvent, aucune communication n'est prévue entre les travailleurs. D'autre part, les soumissions de tâches sont à l'initiative des clients. Par conséquent, la prise en compte de tâches dynamiques (générées pendant l'exécution d'une application) n'est souvent pas prévue. Ces limites constituent un handicap au déploiement d'applications parallèles coopératives, en particulier l'optimisation combinatoire parallèle. En effet, dans le cas des métaheuristiques le modèle insulaire est basé sur la coopération entre îles. Le modèle d'exploration arborescente des méthodes exactes parallèles nécessite la communication entre les différents processus pour le partage du travail et l'échange de la meilleure solution trouvée.

Pour résoudre le problème de prise en compte dynamique de tâches, il suffit d'intégrer dans les processus travailleurs l'interface des clients. Cette interface leur permettra de soumettre au coordinateur les tâches qu'ils génèrent au cours de l'exécution. Cette solution nécessite la modification du code de l'intergiciel, ce qui peut être fastidieux pour les utilisateurs non avertis. Elle constitue la première extension que nous avons apportée à XtremWeb (version v1r2-rc6). Le deuxième problème consiste à mettre en place un mécanisme de coopération entre travailleurs. Ce mécanisme peut être associé à l'application elle-même, dans quel cas sa conception et sa réalisation sont du ressort de l'utilisateur. Encore une fois, cette solution s'avère non triviale pour l'utilisateur non averti car elle nécessite la maîtrise du code de l'intergiciel. L'approche qui nous semble meilleure et que nous avons adoptée est celle qui consiste à concevoir un mécanisme de coopération et à l'intégrer dans l'intergiciel. L'avantage de cette solution est la possibilité d'exploiter le mécanisme de manière transparente à travers son interface.

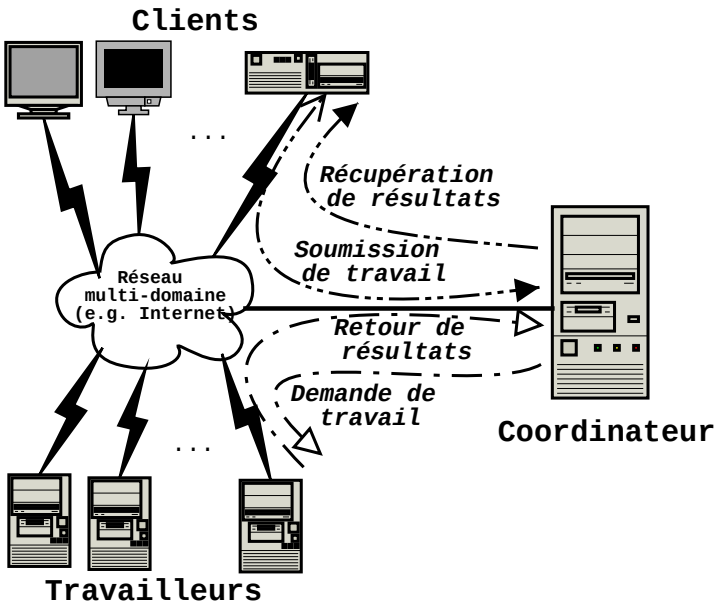

FIG. 4.3 – Architecture d'un intergiciel Coordinateur-Travailleurs

4.1.3.1 Un mécanisme de coopération pour les intergiciels de type *Coordinateur-Travailleurs*

Plusieurs modèles de coopération ont été proposés dans la littérature [GC92, PA98], et Linda [Gel85] est certainement le plus populaire d'entre eux. Notre mécanisme de coopération est basé sur une extension du modèle Linda. Dans ce modèle, la coordination est assurée au travers d'une mémoire associative appelée *espace de tuples* ou *tuple space ou TS*. Chaque tuple est un ensemble ordonné de valeurs souvent de types hétérogènes. La création, synchronisation et communication de processus sont réalisées par un jeu réduit de quatre opérations élémentaires : *rd*, *in*, *out* et *eval*. Les deux premières primitives permettent de lire un tuple de TS respectivement sans le détruire et en le détruisant. La primitive *out* sert à écrire un tuple dans TS. L'opération *eval* est dédiée à la création de tuples actifs synonymes de création de processus dont les résultats sont stockés dans ces tuples. Le modèle Linda n'est pas adapté à l'optimisation multi-objectif sur grilles pour trois raisons majeures. D'abord, il n'autorise pas la réécriture atomique (lecture suivie d'une écriture). Celle-ci est réalisée en deux opérations séparées *in* ou *rd* suivie de *out*. Elle nécessite donc deux (longues) communications, une synchronisation et la gestion des situations de perte des données lues. Ensuite, il ne supporte pas les opérations de groupe nécessaires à la gestion de fronts (de solutions) Pareto. Enfin, les opérations non bloquantes et importantes dans un contexte volatile ne sont pas non plus supportées. Nous avons donc proposé une extension du modèle Linda pour pallier ces limites. L'espace des tuples contient, entre autres, une collection de fronts (Pareto). Des opérations de groupe, non bloquantes et de réécriture sont définies sur ce nouvel espace de tuples. Une description plus détaillée et plus formelle du modèle est présentée dans [MMT05b, MMT05c].

4.1.3.2 Implémentation du modèle et son intégration dans XtremWeb

Le modèle Linda étendu proposé, illustré par la figure 4.4, pourrait être intégré à n'importe quel intergiciel de type Coordinateur-Travailleurs. L'implémentation du modèle est composée de deux parties : une interface de coordination et son implémentation au sein des processus travailleurs, et un médiateur de coordination ou un bus logiciel de requêtes de coordination (ou *BRC*). L'espace des tuples est une partie de la base de données de l'intergiciel, et chaque tuple est un enregistrement de la base.

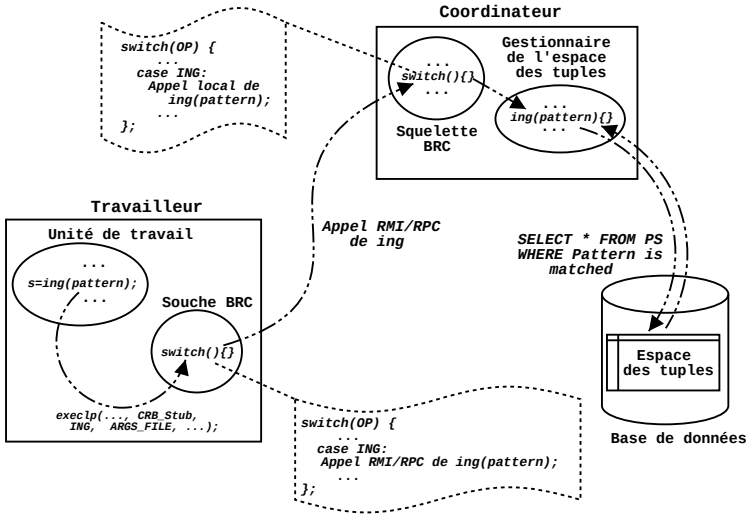

FIG. 4.4 – Intégration du modèle Linda étendu dans un intergiciel de type Coordinateur-Travailleurs

L'interface de coordination contenant les opérations du modèle Linda étendu est implémentée sous forme de librairie en C/C++ et en Java. Cette librairie doit être incluse dans les applications déployées au niveau des travailleurs. Au niveau du coordinateur, l'interface de coordination est implémentée en Java et C++ comme un gestionnaire de l'espace des tuples. Le BRC permet le transport vers le coordinateur des appels des travailleurs aux opérations de coordination. Il a deux composants : un squelette du côté du coordinateur et une souche du côté des travailleurs. Le rôle de la souche est de transformer en appels RMI ou RPC les appels locaux aux opérations de coordination effectués par les unités de travail exécutées par les travailleurs. Le rôle du squelette est de transformer les appels RMI ou RPC en appels locaux aux opérations de coordination exécutées par le gestionnaire de l'espace des tuples. Ces appels locaux sont traduits en requêtes SQL adressées à l'espace Pareto stocké dans la base de données de l'intergiciel.

XtremWeb [Fed03] est un intergiciel de calcul global P2P développé en Java à l'Université de Paris-Sud. Il s'agit d'un intergiciel de type *Coordinateur-Travailleurs* et son architecture est similaire à celle de la figure 4.3. Le coordinateur utilise une BD MySql pour stocker notamment les résultats des applications et des informations relatives aux travailleurs et au déploiement de ces applications. Le modèle de coordination proposé a été intégré à XtremWeb. La version obtenue a été utilisée pour implémenter et expérimenter des métaheuristiques et méthodes exactes parallèles multi-objectifs et leur coopération.

4.1.3.3 Application au problème du Flow-Shop bi-objectif

Le problème du Flow-Shop est un problème d'ordonnancement [TB02] qui a suscité un intérêt particulier dans le domaine industriel. Le problème consiste à ordonnancer N travaux (ou *jobs*) $J_1, J_2, \ldots, J_N$ sur M machines. Les machines sont des ressources critiques car chaque machine ne doit pas être affectée à deux travaux simultanément. Chaque travail J_i est composé de M tâches consécutives $t_{i1}, \ldots, t_{iM}$, où t_{ij} représente la j^{eme} tâche du travail J_i ayant besoin de la machine m_j. A chaque tâche t_{ij} est associé un temps de traitement p_{ij}, et chaque travail J_i doit être fini avant une date buttoir d_i.

Nous avons considéré en particulier le problème de permutation Flow-Shop (*BPFSP*) où les travaux peuvent être ordonnancés dans le même ordre sur toutes les machines (voir figure 4.5). Par conséquent, deux objectifs sont à optimiser (ici à minimiser) :

- C_{max} : temps d'exécution complète des travaux (ou *Makespan*),
- T : retard total (ou *Tardiness*).

Si s_{ij} désigne le temps de passage de la tâche t_{ij}, les deux objectifs peuvent être formulés de la manière suivante :

$$f_1 = C_{max} = Max\{s_{iM} + p_{iM} | i \in [1 \ldots N]\}$$

$$f_2 = T = \sum_{i=1}^{N} [max(0, s_{iM} + p_{iM} - d_i)]$$

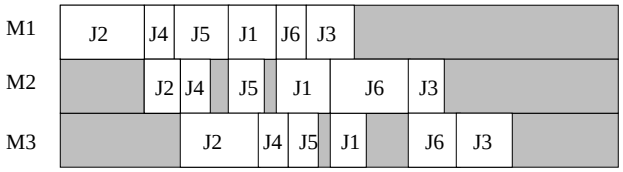

FIG. 4.5 – Instance du problème du Flow-Shop avec 6 travaux et 3 machines

Le front Pareto PF associé au problème du Flow-Shop peut être formulé de la manière suivante :

$$\forall y, \exists x \in PF, (m(x) \leq m(y)) \ or \ (t(x) \leq t(y))$$

où x et y sont des solutions du problème, et $m(x)$ (resp. $t(x)$) est la valeur de x correspondant à l'objectif *makespan* (resp. *tardiness*).

4.1.3.4 Expérimentation et analyse

Dans les premières expérimentations de l'algorithme $B\&B$ gridifié et appliqué au problème BPFS, nous avons considéré les deux instances 0 et 1 du problème d'ordonnancement de 50 travaux sur 5 machines. Ces instances ont été générées par Eric Taillard [Tai93] [1] avec le *makespan* comme seul objectif. Ces instances ont été étendues avec le retard ou *tardiness* comme deuxième objectif [2]. Ces instances n'ont jamais été résolues en prenant en compte les deux objectifs simultanément.

[1] http ://www.eivd.ch/ina/Collaborateurs/etd/default.htm
[2] http ://www.lifl.fr/OPAC/

L'application a été expérimentée sur la grille illustrée par la figure 4.6 et composée de 412 machines réparties sur 4 réseaux (domaines d'administration) appartenant à 4 établissements de l'Université de Lille1 : les réseaux Gigabit Ethernet d'enseignement (students.deule.net) et de recherche (rech-info.yser.net) de Polytech'Lille, le réseau Ethernet rapide de l'IUT "A" (iut-info.univ-lille1.fr), et le réseau Gigabit Ethernet des FIL (Formations Informatiques de Lille) (fil.univ-lille1.fr). Ces réseaux de PC sont interconnectés par le réseau Gigabit Ethernet du campus de l'Université.

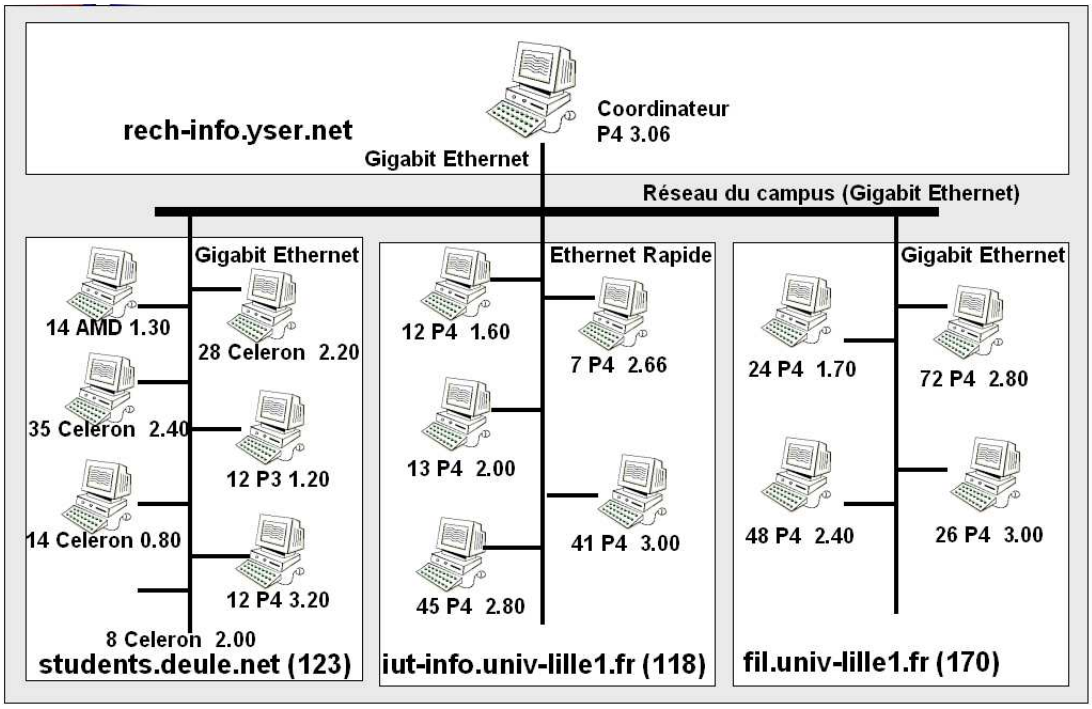

FIG. 4.6 – Illustration de la grille de calcul

Le tableau 4.1 contient les détails de la grille : le type de processeur de chaque PC et son système d'exploitation, le domaine d'administration auquel il appartient et son rôle (coordinateur ou travailleur).

Le tableau 4.2 contient quelques statistiques collectées pendant la résolution des instances 0 et 1 du problème bi-objectif 50 travaux sur 5 machines. Les travailleurs ont effectué un nombre important d'opérations de sauvegarde de leurs unités de travail. Ces sauvegardes ont permis plusieurs opérations de restauration après indisponibilité d'autant de machines. Ces deux types d'opérations montrent que le mécanisme de sauvegarde/restauration a bien fonctionné. D'autre part, un nombre également important d'opérations de régulation de charge (requêtes de demande de travail satisfaites) ont été effectuées. Pour chaque unité de travail allouée, une trace (un tuple ou descripteur) est sauvegardé(e) dans l'espace des tuples. Tous ces tuples sont détruits une fois les unités de travail correspondantes ont été traitées. Les travailleurs ont mis à jour la meilleure solution trouvée plusieurs milliers de fois. Les duplications d'unités de travail entre travailleurs sont effectuées pendant la phase de terminaison de l'application. Ce travail spéculatif permet

CPU (GHz)	OS	Domaine	Rôle	Nbr
P4 3.06	Redhat.9	rech-info.yser.net	Coordinateur	1
P4 1.70	Mandrake.10	fil.univ-lille1.fr	Travailleur	24
P4 2.40				48
P4 2.80				72
P4 3.00				26
AMD 1.30	Debian	students.deule.net		14
Celeron 2.40				35
Celeron 0.80				14
Celeron 2.00				8
Celeron 2.20				28
P3 1.20				12
P4 3.20				12
P4 1.60		iut-info.univ-lille1.fr		12
P4 2.00				13
P4 2.80				45
P4 2.66				7
P4 3.00				41
Total				412

TAB. 4.1 – Détails de la grille

de mieux exploiter l'hétérogénéité de la grille. Enfin, le temps d'obtention avec la grille d'un front exact est de $15h05$ (resp. 6 jours et 8 heures) pour l'instance 0 (1) du problème alors que sur une seule machine ce front n'est toujours pas obtenu après deux semaines d'exécution.

	Instance 0	Instance 1
Temps d'exécution	15h05	152h03 (6j8h)
Nombre de processeurs dans la grille	357	412
Opérations de sauvegarde	96 604	958 718
Opérations de restauration	210	2 441
Unités de travail (UT) allouées	92 870	335 862
UT dupliquées	588	2 523
Tuples détruits (UT terminées)	92 870	335 862
Mises à jour de la meilleure solution	8 264	726

TAB. 4.2 – Quelques statistiques sur la résolution des instances 0 et 1 du problème bi-objectif 50 travaux sur 5 machines.

4.1.4 Implémentation sur grilles basée sur RPC

L'implémentation présentée dans les sections précédentes est basée sur un intergiciel de type Coordinateur-Travailleurs (en particulier XtremWeb) étendu avec le modèle de coopération proposé. Parmi les caractéristiques de l'intergiciel, nous en soulignons trois : (1) il fournit notamment des interfaces facilitant son exploitation ; (2) il offre également une base de données dans laquelle sont stockées des informations relatives au déroulement des exécutions sur la grille, concernant les travailleurs mais aussi les applications en cours d'exécution ; (3) l'intergiciel a été développé en Java, ce qui facilite sa portabilité. Ces caractéristiques présentent certes les avantages cités mais elles sont pénalisantes en terme d'efficacité d'exécution. C'est pourquoi, nous avons proposé une autre implémentation en C++ plus simple utilisant des appels RPC. Celle-ci est basée sur le modèle de vol de cycles à grande échelle et intègre le modèle de coopération illustré par la figure 4.4. La base de données est remplacée par un simple fichier.

Cette nouvelle implémentation a été utilisée pour expérimenter et valider l'algorithme B&B gridifié appliqué au problème du Flow-Shop mono-objectif (*makespan*) d'ordonnancement de 50 travaux sur 20 machines (instance $Ta056$ de Taillard). Une borne inférieure ($Cmax = 3667$) a été proposée pour cette instance en 1995 dans [Vae95] et celle-ci a été résolue de manière approchée [RS04] en 2004 ($Cmax = 3681$). Cependant, faute de ressources et d'algorithmes capables de résoudre le problème en exploitant les grilles de calcul, cette instance n'a jamais été résolue de manière exacte. Notre algorithme B&B gridifié a permis la résolution exacte de cette instance sur une grille composée des 412 processeurs de la grille universitaire (présentée précédemment) et des processeurs de Grid5000 [3]. Le projet Grid5000 vise à construire une grille expérimentale nationale hautement reconfigurable. Cette grille est constituée de 9 grappes interconnectées par le réseau haut débit Renater [4] et réparties en France sur les sites suivants : Bordeaux, Grenoble, Nancy, Lille, Lyon, Orsay (Paris), Rennes, Sophia Antipolis et Toulouse. L'approche a permis de produire avec preuve d'optimalité la solution optimale de cette instance en *7 semaines* en utilisant en moyenne 500 processeurs. Un pic de 1245 processeurs a été atteint pendant une nuit. La meilleure solution trouvée a été initialisée avec la meilleure solution (approchée) connue ayant une valeur de *makespan* $Cmax = 3681$ [RS04] :

Permutation = (14, 37, 3, 5, 18, 13, 33, 20, 8, 21, 42, 49, 50, 40, 43, 28, 19, 32, 46, 30, 6, 45, 4, 39, 36, 47, 24, 22, 1, 2, 44, 31, 17, 25, 10, 16, 11, 26, 15, 48, 7, 41, 23, 27, 29, 34, 9, 35, 38, 12).

La solution optimale trouvée par notre algorithme, avec un *makespan* de $Cmax = 3679$, est la suivante :

Permutation = (14, 37, 3, 18, 8, 33, 11, 21, 42,5, 13, 49, 50, 20, 28, 45, 43, 41, 46, 15, 24, 44, 40, 36, 39, 4, 16, 47, 17, 27, 1, 26, 10, 19, 32, 25, 30, 7, 2, 31, 23, 6, 48, 22, 29, 34, 9, 35, 38, 12).

Pour s'assurer de la validité du résultat, la résolution a été effectuée une nouvelle fois en initialisant la meilleure solution trouvée à la meilleure solution connue diminuée de 1,

[3] `https://www.grid5000.fr`
[4] `http://www.renater.fr`

Durée totale d'exécution	25 jours 46 minutes
Durée totale d'exécution agrégée	22 ans 185 jours 16 heures
Nombre moyen de processeurs utilisés	328
Pic de processeurs atteint sur toute la grille	1 195
Nombre de fois qu'un processeur a rejoint la grille	11 802
Pic de processeurs atteint sur chaque site	Bordeaux (88), Orsay (360), Sophia (190), Lille (98), Toulouse (112), Rennes (456), Université de Lille1 (304)
Noeuds explorés	6,50874 e+12
Noeuds redondants	0,39%
Unités de travail (UT) allouées	129 958
Opérations de sauvegarde/restauration	4 094 176
Efficacité parallèle	97%
Taux d'utilisation CPU du coordinateur	1,7%

TAB. 4.3 – Quelques statistiques sur la résolution de l'instance $Ta056$ du problème mono-objectif (*makespan*) d'ordonnancement de 50 travaux sur 20 machines.

c'est à dire à $Cmax = 3680$. Le tableau 4.3 montre les statistiques collectées pendant la résolution de l'instance sur la grille de plus de 2000 processeurs.

La résolution n'a duré cette fois-ci que 25 jours et 46 minutes. En moyenne, 328 processeurs ont été utilisés avec un pic de 1195 processeurs. Parmi les 9 sites de Grid5000, 6 ont été impliqués dans le calcul. La résolution en utilisant un seul processeur durerait 22 ans 185 jours et 16 heures. Le nombre d'évènements "un processeur a rejoint la grille" constatés pendant la résolution est de 11802. Cela témoigne de la nature volatile de la grille. La volatilité est liée soit aux pannes des processeurs ou à leur disponibilité variable. Cette dernière peut être due à la libération/réquisition des processeurs de la grille universitaire ou au système de réservation appliqué dans Grid5000. Les processeurs peuvent être réservés pour une durée limitée, ne dépassant souvent pas 2 jours.

4.2 Coopération avec les métaheuristiques parallèles sur grilles

Nous avons étudié différents schémas de coopération parallèle : d'abord des métaheuristiques entre elles, ensuite, entre méthodes exactes et métaheuristiques parallèles. Ces différents schémas ont été appliqués au problème du Flow-Shop bi-objectif expérimentés sur la grille universitaire illustrée par la figure 4.6.

4.2.1 Coopération entre métaheuristiques parallèles

Le but de cette coopération parallèle est d'utiliser la grille de calcul pour exploiter de manière efficace à la fois les pouvoirs d'exploration et d'exploitation apportés par les deux

classes de métaheuristiques. Il s'agit, comme le montre la figure 4.7, d'une combinaison hiérarchique de deux types d'hybridation et de deux modèles parallèles de métaheuristiques. Le modèle insulaire parallèle permet de déployer différentes populations (îles) en mode d'hybridation *HHC*. Dans le cadre particulier du problème du Flow-Shop bi-objectif, l'algorithme évolutionnaire instancié sur chaque île est l'algorithme hybride génétique mimétique (AGMA) proposé dans le cadre d'une thèse de l'équipe OPAC [Bas05]. Les deux modèles d'évaluation parallèle de la population et de chaque solution ne sont pas exploités car leur granularité n'est pas suffisante pour justifier leur exploitation. En effet, le coût CPU de la fonction objectif est faible (quelques dizaines de microsecondes).

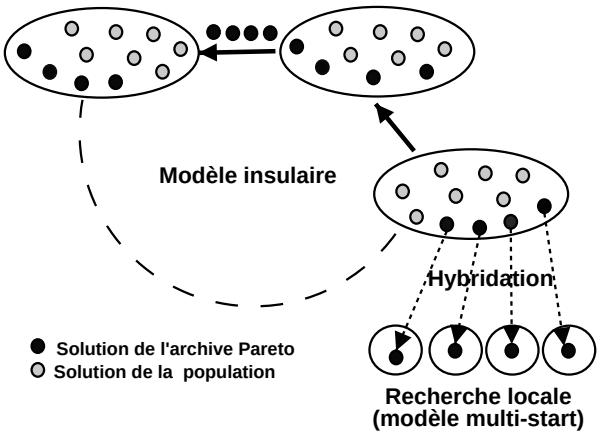

FIG. 4.7 – Coopération parallèle entre métaheuristiques

Chaque AGMA fait évoluer la population de son île et stocke dans une archive Pareto les solutions non dominées. Il échange avec les autres AGMA son archive Pareto de manière asynchrone. A chaque itération de l'algorithme AGMA, l'archive Pareto est exploitée et intensifiée par un processus de recherche locale suivant le schéma d'hybridation *HHR*. La recherche locale est effectuée suivant le modèle multi-départ parallèle. Chaque solution de l'archive Pareto constitue la solution initiale d'une recherche locale dont le but est de calculer le voisinage de celle-ci. Les solutions non dominées de ce voisinage sont intégrées dans l'archive Pareto de l'AGMA.

L'analyse présentée dans le chapitre 1 nous a permis d'identifier les informations à stocker pour chaque modèle parallèle pour gérer la disponibilité variable des machines les déployant. Concernant le modèle insulaire, à chaque AGMA est associé un tuple particulier dans l'espace Pareto dédié au stockage d'informations nécessaires au mécanisme de sauvegarde/restauration. Ce tuple est de la forme : $[agmaId,\ gen,\ pop,\ fp]$, où $agmaId$, gen, pop et fp désignent respectivement l'identifiant unique de l'AGMA, le nombre de générations écoulées, la population et le front Pareto au dernier point de sauvegarde. Les individus émigrants sont stockés dans l'espace Pareto puisqu'ils y transitent. D'autre part, la granularité des recherches locales étant fine, la gestion de la disponibilité variable des machines déployant le modèle multi-départ est laissée à la charge de XtremWeb.

L'application de la métaheuristique parallèle hybride au problème BPFS a été expérimentée en considérant les mêmes instances 0 et 1 du problème (50 travaux sur 5 machines

et la même grille expérimentale que précédemment. Les paramètres du modèle insulaire sont les suivants : la fréquence de migration et de sauvegarde (paramètre du mécanisme de sauvegarde/restauration) est de 2 minutes, le nombre d'émigrants à chaque opération de migration est fixé à 20 si la taille de l'archive Pareto est supérieure ou égale à 20 et à la totalité de l'archive sinon, la taille de la population dans chaque AGMA est de 100, les émigrants sont choisis de manière aléatoire si la taille de l'archive est supérieure à 20, à l'arrivée les immigrants sont fusionnés avec l'archive locale. La figure 4.8 illustre l'évolution de la S-métrique en fonction du temps en faisant varier le nombre d'AGMA pour l'instance 0. Nous rappelons que cette métrique permet de mesurer l'hyper volume des solutions dominées par le front Pareto obtenu par rapport à une solution de référence dominée, ici la solution $(9380, 2865)$ de l'espace objectif. Ces premiers résultats montrent d'une part que les différentes exécutions permettent d'obtenir le même front Pareto, ce qui démontre la robustesse de la métaheuristique parallèle hybride. D'autre part, la figure 4.8 nous permet de trouver le nombre d'îles nécessaire et suffisant pour obtenir, en 23 minutes, la meilleure solution approchée possible i.e. 20 îles pour l'instance 0. En ce qui concerne l'instance 1, la meilleure solution approchée est obtenue en $1h43$ avec un nombre idéal de 60 îles.

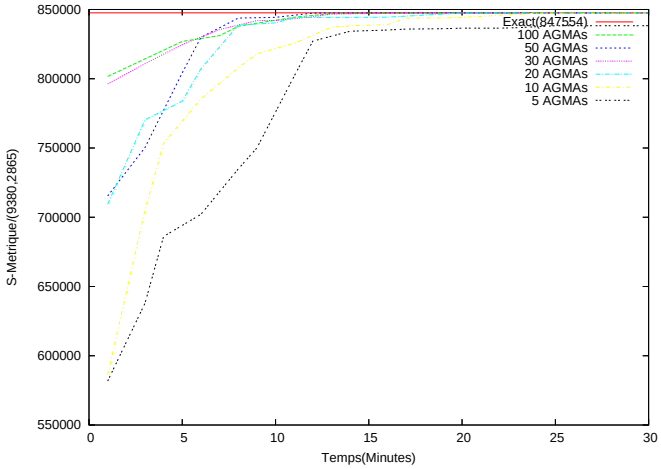

FIG. 4.8 – Évolution de la S-métrique en fonction du nombre d'AGMA sur l'instance 0 du problème 50 travaux sur 5 machines

4.2.2 Coopération entre méthodes exactes et métaheuristiques parallèles

Un critère majeur ayant un impact considérable sur l'efficacité des méthodes exactes est le choix de la valeur initiale de la meilleure solution trouvée. En effet, ce paramètre est fondamental et critique pour les opérations de séparation et du test d'élagage. Le choix judicieux de celui-ci permet de réduire de manière significative la taille de l'espace de recherche exploré. Cependant, à l'instar des expérimentations présentées dans la section 4.1.3.4, la meilleure solution est souvent initialisée à la pire des valeurs i.e. l'infini. C'est pourquoi, nous avons proposé une approche dans laquelle la valeur initiale de la

meilleure solution trouvée est fournie à la méthode exacte par une métaheuristique. L'intérêt de cette approche sera avéré si le coût d'exécution de la métaheuristique est largement inférieur au coût d'exploration des noeuds élagués grâce à la valeur initiale qu'elle fournit. Autrement dit, le coût cumulé d'exécution de la métaheuristique et de la méthode exacte est largement inférieur au coût de la même méthode exacte dont la meilleure solution est initialisée à l'infini. Il faudrait donc que la métaheuristique produise la meilleure solution possible en moins de temps possible. La production de la meilleure solution possible permettra à la méthode exacte d'élaguer le plus de noeuds possible. La minimisation du temps d'exécution de la métaheuristique est primordial pour ne pas compromettre l'intérêt de celle-ci. L'utilisation d'une métaheuristique parallèle hybride telle que présentée ci-dessus s'avère donc une approche prometteuse. Nous avons envisagé deux schémas d'hybridation entre métaheuristiques et méthodes exactes : *HHR* (figure 4.9.(a)) et *HHC* (figure 4.9.(b)).

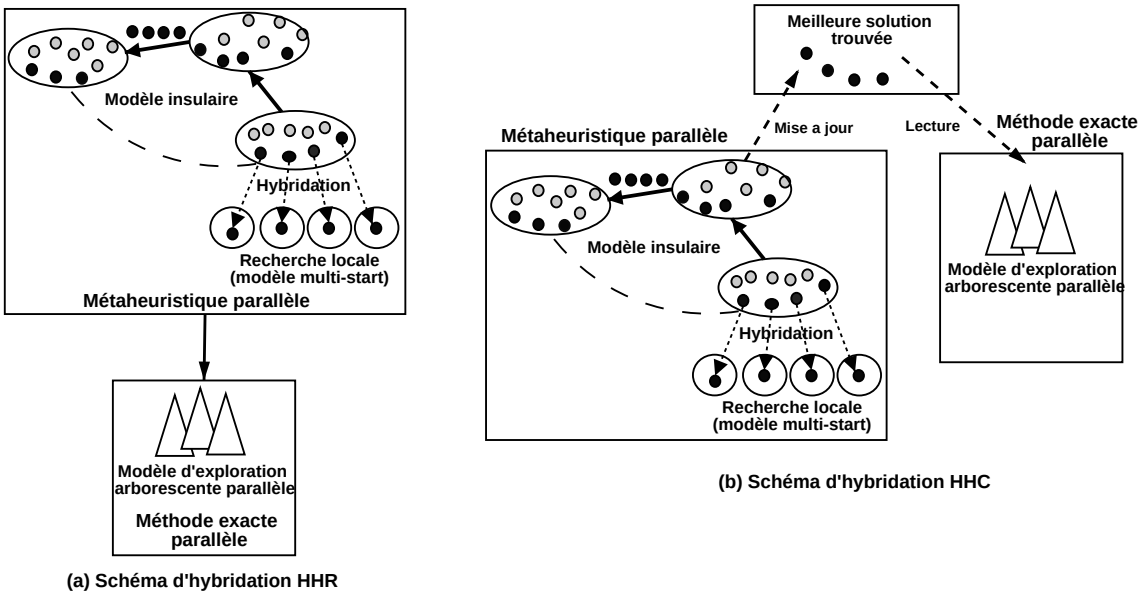

FIG. 4.9 – Coopération parallèle entre métaheuristiques et méthodes exactes

Dans le mode *HHR*, la métaheuristique et la méthode exacte sont exécutées en séquentiel. Dans le schéma *HHC*, les deux méthodes sont exécutées en mode co-évolutionnaire. La meilleure solution trouvée est partagée entre la métaheuristique et la méthode exacte. La méthode exacte initialise sa meilleure solution trouvée à l'infini. Périodiquement, elle récupère la meilleure solution trouvée par la métaheuristique pour mettre à jour la sienne. L'exécution s'arrête lorsque la méthode exacte n'a plus de noeuds à explorer. Ces deux schémas d'hybridation permettent de produire des fronts Pareto exacts contrairement à l'approche de coopération (entre métaheuristiques seulement) décrite dans la section précédente.

L'application de la métaheuristique parallèle coopérative hybridée avec une méthode exacte parallèle au problème BPFS a été expérimentée en considérant toujours les mêmes instances 0 et 1 du problème 50 travaux sur 5 machines et la grille universitaire. Les résultats, illustrés par les tableaux 4.4 et 4.5, montrent que les métaheuristiques permettent aux

Schéma d'hybridation	Méta.	B&B	Méta.+B&B	S-métrique
Métaheuristique parallèle hybride	0h23		0h23	847536
B&B parallèle		15h05	15h05	847554
Hybridation HHR	0h23	3h28	3h51	847554
Hybridation HHC	0h27	3h55	4h22	847554

TAB. 4.4 – Temps d'exécution obtenus avec et sans hybridation pour l'instance 0.

Schéma d'hybridation	Méta.	B&B	Méta.+B&B	S-métrique
Métaheuristique parallèle hybride	1h43		1h43	
B&B parallèle		152h03	152h03 (6j8h)	
Hybridation HHR	1h43	116h26	118h09	(5j8h)
Hybridation HHC	1h44	128h40	128h40	(4j22h)

TAB. 4.5 – Temps d'exécution obtenus avec et sans hybridation pour l'instance 1.

méthodes exactes d'obtenir des fronts Pareto exacts en réduisant de manière significative leurs temps d'exécution.

Dans l'autre sens, comme il a été souligné dans le chapitre 1, les méthodes exactes peuvent servir d'étalon pour mesurer l'efficacité des métaheuristiques. Dans le cas du problème BPFS, la différence en terme de S-métrique entre le front approché et le front exact est minime puisqu'elle est de 0.00002%. Plus précisemment, le front exact contient une solution Pareto optimale de plus que le front approché. Ce résultat démontre l'efficacité de la métaheuristique parallèle hybride sur l'instance étudiée. Encore mieux, pour l'instance 1 le front Pareto produit par la métaheuristique parallèle hybride est exact. C'est pourquoi, dans le tableau 4.5 la valeur de S-métrique n'est pas indiquée. Dans ce cas, l'intérêt de la méthode exacte est de prouver l'optimalité du résultat fourni par la métaheuristique.

D'autre part, le mode d'hybridation HHR paraît plus efficace que le mode HHC en terme de temps d'exécution. Le mode HHC fournit le même front que le mode HHR mais la méthode exacte explore plus de noeuds puisqu'elle ne démarre pas directement du meilleur front fourni par la métaheuristique. Il faudrait également confirmer ce résultat en expérimentant d'autres instances du problème de plus grande taille. D'autre part, le gain en temps d'exécution apporté par la métaheuristique dans le cas de l'instance 0 est plus important que dans le cas de l'instance 1 même si la solution produite par la métaheuristique n'est pas exacte. L'efficacité apportée dépend fortement de la position de la solution optimale dans l'espace de recherche.

L'étude expérimentale des deux schémas d'hybridation a révélé une question ouverte importante : à quel moment arrêter la métaheuristique en mode relais ? Son arrêt prématuré permet de fournir une relativement bonne solution de départ en moins de temps mais laisserait beaucoup plus de travail à la méthode exacte. En effet, les résultats illustrés par la figure 4.4 montrent que pour faire progresser le front Pareto d'un seul point il fallait environ $3h28$ sur un temps total de résolution de $3h51$. A l'inverse, l'arrêt tardif de la métaheuristique (après convergence) permettrait de produire une meilleure solution

de départ éventuellement optimale ou très proche de l'optimum. L'intérêt est de réduire de manière significative le nombre de noeuds explorés et donc le temps d'exécution de la méthode exacte. Cependant, la convergence de la métaheuristique pourrait survenir après plusieurs jours voire semaines. En effet, après 10 jours 6 heures et 40 minutes la convergence n'a pas encore été atteinte sur une instance 200 travaux sur 10 machines du problème BPFS [MMT05a]. Par ailleurs, l'étude a également montré qu'il est important de définir pour le schéma HHC une politique de répartition de ressources entre métaheuristiques et méthodes exactes en trouvant un compromis entre l'avancement de l'exploration de l'arbre (plus de processeurs à la méthode exacte) et la production d'une meilleure solution approchée (plus de processeurs à la métaheuristique).

4.3 Conclusion

Dans ce chapitre, nous avons proposé une gridification de l'algorithme B&B en prenant en compte l'analyse présentée dans le chapitre 1 sur les méthodes exactes parallèles et leur coopération avec les métaheuristiques sur grilles. Cette gridification est basée sur un codage de l'arbre de base et des unités de travail permettant de minimiser le coût des communications nécessaires à la régulation de charge et le coût de stockage inhérent au mécanisme de sauvegarde/restauration. Ensuite, une métaheuristique multi-objectif parallèle hybride a été proposée. Il s'agit d'un algorithme coopératif insulaire hybridé avec la recherche locale parallèle multi-départ. Les deux modèles parallèles sont asynchrones et basés sur le vol de cycles à grande échelle. Enfin, nous avons proposé une hybridation de l'algorithme B&B avec la métaheuristique parallèle hybride.

Pour l'implémentation des différents algorithmes, nous avons choisi dans un premier temps un intergiciel dédié au vol de cycles i.e. un intergiciel de type Coordinateur-Travailleurs. Cependant, très souvent les intergiciels de ce type ne supportent pas le parallélisme coopératif. Ce qui nous a amené à proposer un mécanisme de coopération basé sur Linda pour l'optimisation combinatoire multi-objectif parallèle sur grilles et dédié à ces intergiciels. Ce mécanisme a été intégré à XtremWeb et a servi au déploiement des algorithmes multi-objectifs proposés.

Ces algorithmes ont été appliqués au problème du Flow-Shop bi-objectif et expérimentés sur une grille de plus de 400 machines réparties sur 4 domaines d'administration. Les résultats préliminaires ont montré d'une part l'efficacité et la robustesse apportée aux métaheuristiques par les modèles parallèles combinés aux mécanismes d'hybridation. En effet, la métaheuristique parallèle hybride permet de fournir des fronts Pareto approchés proches des fronts exacts. D'autre part, l'hybridation de la méthode exacte avec la métaheuristique permet à la deuxième méthode de fournir à la première une solution de départ de nature à réduire de manière importante son temps d'exécution. L'analyse des résultats et du comportement des deux schémas d'hybridation HHR et HHC à l'exécution a soulevé certains problèmes notamment : à quel moment doit être arrêtée la métaheuristique en mode relais, comment gérer la répartition des ressources entre les deux types de méthodes dans le mode co-évolutionnaire.

Nous avons également proposé une implémentation simplifiée et efficace de l'algorithme

gridifié basée sur les appels RPC et sur notre modèle Linda étendu de coopération à grande échelle. Cette implémentation a permis l'exploitation d'un nombre plus important de processeurs appartenant à la grille universitaire et à la grille expérimentale nationale Grid'5000 pour la résolution exacte pour la première fois de l'instance de grande taille $Ta056$ du problème de Flow-Shop mono-objectif ($makespan$) de 50 travaux sur 20 machines. La résolution a duré 7 semaines impliquant en moyenne 500 processeurs avec un pic de 1245 pendant une nuit de calcul.

Chapitre 5

Conclusions et perspectives

Dans ce document, nous avons présenté nos contributions à la résolution parallèle hybride sur grilles de calcul de problèmes d'optimisation combinatoire mono et multi-objectifs. Nous avons d'abord proposé une *analyse* des méthodes d'optimisation combinatoire parallèles hybrides. Cette analyse est en soit une contribution car, contrairement aux autres études proposées dans la littérature, *elle est placée dans le contexte des grilles de calcul*. En outre, elle s'inscrit dans une démarche méthodologique de *mise en place d'une plate-forme* d'aide à la mise en oeuvre de méthodes d'optimisation réutilisables en termes de conception et de code. Pour chaque méthode d'optimisation, nous avons en particulier identifié les modèles parallèles et les mécanismes d'hybridation. L'analyse effectuée dans le contexte des grilles de calcul a pour objectif, d'une part, de *repenser l'algorithmique* de ces modèles et mécanismes en vue de sa gridification, et d'autre part, de *mieux choisir les intergiciels, le cas échéant les adapter*, pour les supporter. Dans nos travaux, la gridification signifie la prise en compte des caractéristiques des grilles lors de la conception des méthodes d'optimisation, notamment l'hétérogénéité, la volatilité, la nature multi-domaine d'administration et la grande échelle.

Sur le plan algorithmique, il s'avère que le modèle de vol de cycles à grande échelle est particulièrement adapté aux grilles à plus d'un titre : il permet une meilleure exploitation de l'hétérogénéité et une meilleure répartition de la charge puisque les machines les plus puissantes demandent plus de travail que les machines les moins puissantes ; il est plus tolérant aux pannes en particulier pour les métaheuristiques ; et, il permet de passer les pare-feu. Pour gérer la volatilité liée aux pannes et à la disponibilité variable des machines, nous avons proposé pour chaque méthode les données nécessaires au mécanisme de sauvegarde/restauration. Ce mécanisme a l'avantage d'être plus efficace que celui proposé par les intergiciels d'exploitation des grilles puisque, d'une part, la reprise se fait depuis le dernier point de sauvegarde et non pas depuis le début. D'autre part, son coût de mise en oeuvre en termes de stockage et de communication est négligeable en comparaison du temps total d'exécution des applications sur les grilles. En outre, il est indépendant du matériel utilisé et des problèmes traités.

Pour assurer le passage à l'échelle, il faut optimiser entre autres les communications générées pendant la résolution des problèmes en minimisant leur nombre et leurs volumes

d'information. Pour les métaheuristiques, il s'agirait de proposer un codage des individus à faible coût de communication. Il faudrait en particulier trouver un paramétrage (notamment le nombre d'émigrants et la fréquence de migration) efficace du processus de migration pour le modèle insulaire. Pour les méthodes exactes, un intérêt particulier devrait être porté à la description des unités de travail. A l'instar de celle que nous avons proposée, une bonne description doit permettre de minimiser à la fois le coût de régulation de charge et du mécanisme de sauvegarde/restauration.

Les méthodes exactes génèrent souvent une quantité de travail de nature à justifier l'intérêt d'une grille de calcul pour la traiter. Ce constat n'est pas toujours avéré pour les métaheuristiques. En effet, le modèle insulaire est souvent limité à quelques îles. Le modèle d'évaluation parallèle de la population est limité à la taille de la population, souvent fixée à 100. Combiné à ces deux modèles, le modèle d'évaluation parallèle d'une solution, souvent non cité dans la littérature sur les métaheuristiques parallèles, s'avère important pour justifier l'utilisation d'une grille de calcul. Par exemple, sur l'application de conception de réseaux mobiles la granularité de ce modèle peut atteindre 15 sur certaines instances. Le modèle peut donc améliorer le degré de parallélisme de manière significative et contribuer à améliorer grandement les performances de l'exécution.

L'hybridation entre métaheuristiques sur grilles est source de robustesse et produit des solutions de meilleure qualité, mais son coût peut s'avérer exorbitant. Pour supporter ce coût, il est nécessaire de combiner les mécanismes d'hybridation avec les modèles parallèles. Ainsi, certains schémas d'hybridation, notamment HBC et HHR, devraient être exploités suivant le modèle parallèle multi-départ. Par exemple, sur chacune des solutions d'une population finale ou du front Pareto produits par un AG on peut appliquer simultanément une recherche locale. L'hybridation sur grilles est source d'efficacité pour les méthodes exactes lorsqu'elles sont combinées avec les métaheuristiques. En effet, les métaheuristiques fournissent de bonnes solutions de départ aux méthodes exactes pour réduire le nombre de noeuds à explorer et donc leur temps d'exécution. En effet, sur le problème du Flow-Shop le temps d'exécution passe de $15h05$ ($152h03$) à $3h51$ ($118h09$) sur l'instance 0 (1) du problème 50 travaux sur 5 machines. Cependant, cette hybridation soulève certains questions notamment : à quel moment doit être arrêtée la métaheuristique dans le mode relais ? comment gérer la répartition des ressources entre les deux types de méthodes dans le mode co-évolutionnaire ?

En se basant sur l'analyse proposée, nous avons contribué à la résolution parallèle hybride sur grilles de calcul de problèmes académiques et industriels. Pour les problèmes académiques, il s'agit du problème du Voyageur de Commerce (ou TSP) et surtout du problème d'ordonnancement Flow-Shop. Pour ce dernier, nous avons proposé deux approches gridifiées de résolution mono et multi-objectif. En mono-objectif, il s'agit d'un algorithme B&B gridifié utilisant un codage particulier de l'arbre à explorer et des unités de travail minimisant le coût de régulation de charge et de checkpointing pendant la résolution du problème sur une grille. Cet algorithme a été utilisé pour réaliser un défi. Il s'agit de la résolution exacte pour la première fois d'une instance de grande taille du Flow-Shop (50 travaux sur 20 machines). La résolution a duré 7 semaines sur une grille de plus de 2000 processeurs, exploitant en moyenne 500 processeurs avec un pic de 1245 pendant une nuit et une efficacité de 97% d'exploitation du temps de disponibilité des machines.

En multi-objectif, il s'agit d'une métaheuristique parallèle hybride combinant un AG avec la recherche locale. La fonction objectif ayant un faible coût CPU à l'exécution, nous avons proposé un AG parallèle hiérarchique limité à deux niveaux : le modèle insulaire et le modèle d'évaluation parallèle de la population. Cet algorithme est combiné avec la recherche locale dans un schéma HHR parallèle multi-départ. Pour le problème du Flow-Shop, le front Pareto obtenu est souvent proche du front exact. Pour obtenir efficacement le front exact, nous avons proposé une hybridation de cette métaheuristique parallèle hybride avec l'algorithme B&B gridifié.

Pour les problèmes industriels, il s'agit de la conception multi-objectif de réseaux mobiles (Contrat France Telecom R&D) et la fouille de données en spectroscopie PIR (Collaboration avec le LASIR, Université de Lille1). Pour chaque problème, nous avons proposé une modélisation en problème d'optimisation combinatoire et une métaheuristique parallèle hybride pour sa résolution sur la grille. Les résultats expérimentaux obtenus sur l'application de fouille de données en spectroscopie PIR montrent, d'une part, l'intérêt en analyse de spectres de précéder la construction de modèles prédictifs d'une sélection d'attributs. D'autre part, ils montrent l'efficacité des algorithmes génétiques pour ce type de problèmes dans une approche enveloppante. L'application de conception de réseaux en téléphonie mobile est certainement la plus complète et complexe. Elle nécessite la mise en oeuvre conjointe (mode hiérarchique) de tous les modèles parallèles de métaheuristiques et certains mécanismes d'hybridation.

Sur le plan intergiciel, les intergiciels existants supportent facilement les modèles d'optimisation parallèle de type *Fermier-Travailleurs*. Par contre, les modèles coopératifs tels que le modèle insulaire ou le modèle d'exploration parallèle d'une arborescence nécessitent une adaptation des intergiciels pour prendre en compte la génération dynamique de tâches et les communications entre elles. Ce besoin constitue une limite des intergiciels, par exemple de type *Coordinateur-Travailleurs*. Ces intergiciels sont intéressants car ils sont basés sur le vol de cycles à grande échelle permettant le passage de pare-feu, mais sont plutôt mieux adaptés aux applications multi-paramétriques. C'est pourquoi nous avons proposé un mécanisme de coopération dédié à ces intergiciels. Celui-ci est basé sur une extension du modèle Linda et adapté à l'optimisation multi-objectif coopérative sur grilles. Ce mécanisme a été intégré à XtremWeb et utilisé pour la mise en oeuvre de modèles parallèles coopératifs multi-objectifs.

En matière de plate-forme logicielle, l'étude comparative des plates-formes dédiées aux métaheuristiques parallèles hybrides réalisée dans l'analyse proposée a révélé les limites de celles-ci en termes d'utilité, en particulier en matière de parallélisme et d'hybridation sur grilles. En outre, la majorité de ces plates-formes ne sont plus maintenues. Nous avons donc proposé la plate-forme ParadisEO qui fournit un large éventail de composants réutilisables et extensibles : les algorithmes évolutionnaires mono et multi-objectifs, les méthodes de recherche locale, les modèles parallèles et mécanismes d'hybridation associés sur différents types d'architectures notamment les grilles de calcul. La réutilisabilité et l'extensibilité de ces composants proviennent à la fois de la séparation réalisée à leur conception entre la partie générique et la partie dépendante du problème traité, et à l'approche objet utilisée pour les coder. La démarche méthodologique utilisée pour la conception de ParadisEO pourrait être réutilisée pour la construction d'autres plates-formes logicielles.

L'utilisation de librairies standards telles que Pthreads, PVM et MPI pour l'implémentation des différents modèles parallèles rend ces modèles portables et déployables sur des architectures parallèles aussi bien à mémoire distribuée qu'à mémoire partagée. Le couplage (ParadisEO-CMW) de la plate-forme avec Condor-MW rend leur déploiement possible sur des grilles de calcul. Pour gérer la disponibilité variable des machines, nous avons utilisé le mécanisme de sauvegarde/restauration évoqué précédemment. L'exploitation des différents modèles parallèles et du mécanisme de sauvegarde/restauration associé est totalement transparente à l'utilisateur. Les métriques pour l'évaluation de performances sur grilles proposées dans Condor-MW sont de fait intégrées dans ParadisEO-CMW.

L'utilité, la réutilisabilité et l'extensibilité, la portabilité, l'exploitation transparente du parallélisme et de l'hybridation sur grilles et la disponibilité de ParadisEO sur le Web ont contribué à encourager le téléchargement et l'utilisation de ParadisEO dans différents pays et dans différents domaines d'application. Pour assurer la viabilité de la plate-forme, il faut en réaliser un guide d'adaptation. Les tutoriels disponibles sur le Web constituent une base de ce guide.

Le domaine des grilles informatiques étant récent, les *perspectives* de nos travaux sur l'optimisation combinatoire parallèle hybride sur grilles de calcul sont nombreuses. La première perspective portera sur le paramétrage automatique des modèles parallèles des métaheuristiques sur grilles. En effet, les modèles parallèles et mécanismes d'hybridation et les paramètres associés sont toujours choisis par l'utilisateur. Ces choix ne sont pas toujours faciles à faire et peuvent s'avérer inefficaces à l'exécution. Il faudrait donc proposer un mécanisme permettant d'effectuer ces choix de manière automatique en prenant en compte les caractéristiques de la grille et du problème à traiter notamment sa fonction coût. Une étude d'un tel mécanisme a déjà été entamée dans les travaux de Cantu-Pàz [CP97]. Cependant, d'une part, l'environnement parallèle considéré est homogène, non volatile et mono-domaine d'administration. D'autre part, seuls le modèle insulaire et le modèle d'évaluation parallèle de la population ont été étudiés.

Une étude sur la description des unités de travail est nécessaire pour l'optimisation des communications, des opérations de régulation de charge et de sauvegarde/restauration pour les méthodes exactes parallèles sur grilles de calcul. L'approche que nous avons proposée constitue une ébauche de cette étude. Il faudrait l'étendre à d'autres approches telles que celle proposée dans [IF00], en faire une comparaison basée sur une analyse de complexité et une expérimentation sur une grille.

La métaheuristique parallèle hybride combinée à une méthode exacte parallèle présentée dans ce document a été implémentée avec XtremWeb. Cette hybridation devrait être réalisée au travers d'une coopération entre XtremWeb et ParadisEO-CMW qui fourniraient respectivement la méthode exacte et la métaheuristique. Ce travail s'inscrit dans la perspective de coupler ParadisEO-CMW avec une plate-forme dédiée aux méthodes exactes sur grilles. Un tel couplage entre ParadisEO-CMW et la plate-forme BOB++ de l'équipe OPALE du PRiSM constitue un des objectifs du projet *CHOC* (*CHallenges en Optimisation Combinatoire*) de l'ANR "Calcul Intensif et Grilles de Calcul". Il s'agit d'un projet de collaboration entre les équipes OPAC (LIFL, Université de Lille1), OPALE (PRiSM, Université de Versailles), MOAIS (ID-IMAG) et GILCO (ENSGI, INP Grenoble).

Les travaux présentés dans ce document considèrent une grille volatile. Par exemple, le couplage de ParadisEO avec Condor-MW a pour principal objectif de gérer cette volatilité. Cependant, il existe des grilles telles que Grid5000 dites dédiées en ce sens que leurs machines sont toujours disponibles (en réservation) sauf quand elles sont en panne. Pour pouvoir exploiter Grid5000 et les grilles dédiées nous envisageons de coupler ParadisEO avec Globus. Ce couplage sera utilisé dans le cadre du projet *GGM* (*Grille Géno-Médicale*) de l'ACI "Masse de données". Ce projet est une collaboration avec deux partenaires : LIRIS (INSA de Lyon) et IRIT (Université Paul Sabatier à Toulouse). Il servira également à la résolution de problèmes de calcul intensif nécessitant une grille de la taille de Grid5000. A l'instar de l'instance Nug30 du QAP ayant été résolue avec MW [LKGY00] ou du problème des 25 reines ayant été résolue avec ProActive [HCB04], ces problèmes peuvent être des problèmes académiques pour lesquels on voudrait résoudre des instances n'ayant jamais été résolues. Les problèmes peuvent être industriels tels que le *Protein Folding* ou le *docking* qui rentrent dans le cadre du projet DOCK (*Conformation sampling and docking on grids*) de l'ANR "Calcul Intensif et Grilles de Calcul" et de la thèse de Alexandru Tantar qui vient de démarrer sous la direction de E-G. Talbi et moi-même.

A long terme, nous envisageons de constituer une bibliothèque de problèmes modélisés et l'intégrer dans ParadisEO. Les problèmes et les méthodes de résolution fournis par la plate-forme pourront être exploités à distance à travers une interface Web. Cette interface et le moteur de résolution fourni par ParadisEO constitueront un environnement complet (*Problem Solving Environment* ou *PSE*) de résolution de problèmes d'optimisation combinatoire sur grilles.

Bibliographie

[ACE+02] M.G. Arenas, P. Collet, A.E. Eiben, M. Jelasity, J.J. Merelo, B. Paechter, M. Preuss, and M. Schoenauer. A framework for distributed evolutionary algorithms. In *Proceedings of PPSN VII*, september 2002.

[ACK+02] D.P. Anderson, J. Cobb, E. Korpela, , M. Lepofsky, and D. Werthimer. SETI@home : An Experiment in Public-Resource Computing. *Communications of the ACM*, Vol. 45(No. 11) :56–61, Nov. 2002.

[Alb01] E. Alba. NetStream : a Flexible and Simple OOP Message Passing Service for LAN/WAN Utilization. Technical report, Université de Málaga, 2001.

[AN04] E. Alba and A.J. Nebro. *Solving Optimization Problems with Grid-Enabled Technologies*, volume 59. ERCIM News, Oct. 2004.

[AO05] K. Aida and T. Osumi. A Case Study in Running a Parallel Branch and Bound Application on the Grid. In *IEEE Intl. Symp. on Applications and the Internet*, pages 164–173, 2005.

[AT91] H. Almuallim and T.Dietterich. Learning with many irrelevant features. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, 1991.

[AT02a] E. Alba and M. Tomassini. Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5) :443–462, October 2002.

[AT02b] E. Alba and M. Tomassini. Parallelism and Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation*, 6(5) :443–462, October 2002.

[AtMG02] E. Alba and the MALLBA Group. MALLBA : A library of skeletons for combinatorial optimisation. In R.F.B. Monien, editor, *Proceedings of the Euro-Par*, volume 2400 of LNCS, pages 927–932, Paderborn, 2002. Springer-Verlag.

[Bas05] M. Basseur. *Conception d'algorithmes coopératifs pour l'optimisation multi-objectif : Application aux problèmes d'ordonnancement de type Flow-Shop.* PhD thesis, Université de Lille1, France, 2005.

[BBL02] M. Baker, R. Buyya, and D. Laforenza. Grids and grid technologies for wide-area distributed computing. *Software - Practice & Experience*, 32(15) :1437–1466, 2002.

[Bel95] T. Belding. The distributed genetic algorithm revisited. In D. Eshelmann editor, editor, *Sixth Int. Conf. on Genetic Algorithms*, San Mateo, CA, 1995. Morgan Kaufmann.

[BhX01] M.J. Blesa, L. hernandez, and F. Xhafa. Parallel Skeletons for Tabu Search Method. In *8th Int. Conference on Parallel and Distributed Systems*, pages 23–28, Kyongju City, Korea, 2001. IEEE Computer Society Press.

[BLDGS03] Miguel L. Bote-Lorenzo, Yannis A. Dimitriadis, and Eduardo Gómez-Sánchez. Grid Characteristics and Uses : A Grid Definition. In *European Across Grids Conference, LNCS 2970*, Lecture Notes in Computer Science, pages 291–298, 2003.

[BLTZ03] S. Bleuler, M. Laumanns, L. Thiele, and E. Zitzler. Pisa : A Platform and Programming Language Independent Interface for Search Algorithms. In *EMO*, pages 494–508, 2003.

[BMT+98] R. Bagrodia, R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song. Parsec : A Parallel Simulation Environment for Complex Systems. *IEEE Computer October 1998*, 31(10) :77–85, 1998.

[Cah05] Sébastien Cahon. *ParadisEO : Une plate-forme pour la conception et le déploiement de métaheuristiques parallèles hybrides sur clusters et grilles*. PhD thesis, Université de Lille1, 2005.

[CANT95] C. Cotta, J.F. Aldana, A.J. Nebro, and J.M. Troya. Hybridizing Genetic Algorithms with Branch and Bound Techniques for the Resolution of the TSP. In *Artificial Neural Nets and Genetic Algorithms 2, D.W. Pearson, N.C. Steele, R.F. Albrecht (eds.), Springer-Verlag*, pages 277–280, Wien New York, 1995.

[CDC+94] V. D. Cung, S. Dowaji, B. Le Cun, T. Mautor, and C. Roucairol. Parallel and distributed branch-and-bound/A* algorithms. Technical Report 94/31, Laboratoire PRISM, Université de Versailles, 1994.

[CG02] T.G. Crainic and M. Gendreau. Cooperative Parallel Tabu Search for Capacited Network Design. *Journal of Heuristics*, pages 601–627, 2002.

[CHMR87] J.P. Cohoon, S.U. Hedge, W.N. Martin, and D. Richards. Punctuated equilibria : A parallel genetic algorithm. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, page 148. Lawrence Erlbaum Associates, 1987.

[Cla99] Jens Clausen. Branch and Bound Algorithms. Principles and Examples. Technical report, University of Copenhagen, Denmark, March 1999.

[CLS00] J. Costa, N. Lopes, and P. Silva. Jdeal : The Java Distributed Evolutionary Algorithms Library. 2000.

[CMRR02] V.-D. Cung, S.L. Martins, C.C. Ribeiro, and C. Roucairol. Strategies for the Parallel Implemetation of Metaheuristics. *Essays and Surveys in Metaheuristics, C.C. Ribeiro, P. Hansen (Eds.), Kluwer Academic Publishers, Norwell, MA*, pages 263–308, 2002.

[CMT04] S. Cahon, N. Melab, and E-G. Talbi. ParadisEO : A Framework for the Reusable Design of Parallel and Distributed Metaheuristics. *Journal of Heuristics*, Vol. 10 :353–376, May 2004. Kluwer Academic Publishers `http ://www.lifl.fr/~cahon/paradisEO/`.

[CP97] E. Cantú-Paz. A Survey of Parallel Genetic Algorithms, 1997.

[CTM03] S. Cahon, E-G. Talbi, and N. Melab. ParadisEO : A Framework for Parallel and Distributed Biologically Inspired Heuristics. In *IEEE NIDISC'03 Nature Inspired Distributed Computing Workshop, (in conjunction with IEEE IPDPS2002)*. Nice, France, April 2003.

[DFP+96] T.A. DeFanti, I. Foster, M.E. Papka, R. Stevens, and T. Kuhfuss. Overview of the I-WAY : Wide-Area Visual Supercomputing. *The Intl. Journal of Supercomputer Applications and High Performance Computing*, 10(2/3) :123–131, Summer/Fall 1996.

[DL97] M. Dash and H. Liu. Feature selection for classification. *Intelligent Data Analysis*, 1(3), 1997.

[ea03] P. Primet et al. Réflexions et propositions pour une grille haute performance. In *Rapport scientifique sur le projet RNTL e-Toile*, 15 Déc. 2003.

[Fed03] Gilles Fedak. *XtremWeb : une plate-forme pour l'étude expérimentale du calcul global pair-à-pair*. PhD thesis, Université Paris XI, 2003.

[FK97] I. Foster and C. Kesselman. Globus : A Metacomputing Infrastructure Toolkit. *Intl. J. of Supercomputer Applications*, 11(2) :115–128, 1997.

[FK99] I. Foster and C. Kesselman. *The Grid : Blueprint for a New Computing Infrastructure*, chapter Chapter 2 : Computational Grids. Morgan-Kaufman, San Francisco, CA, 1999.

[FKT01] I. Foster, C. Kesselman, and S. Tuecke. The anatomy of the Grid : Enabling Scalable Virtual Organizations. *Int. J. High Perform. Comput. Appl.*, 15(3) :200–222, 2001.

[FMB96] T. Fruhwirth, J.R. Molwitz, and P. Brisset. Planning cordless buisiness communication systems. *IEEE Expert Magazine*, pages 662–673, 1996.

[Fos02] Ian Foster. What is the Grid ? A Three Point Checklist. *Grid Today*, 1(6), July 22 2002.

[FVW98] A. Fink, S. Vo, and D. Woodruff. Building reusable software components for heuristc search, 1998.

[GC92] D. Gelernter and N. Carriero. Coordination languages and their significance. *Communications of the ACM*, 35(2) :97–107, Feb. 1992.

[GC94] B. Gendron and T.G. Crainic. Parallel Branch and Bound Algorithms : Survey and Synthesis. *Operations Research*, 42 :1042–1066, 1994.

[Gel85] D. Gelernter. Generative Communication in Linda. *ACM Transactions on Programming Languages and Systems*, Vol. 7(No. 1) :80–112, jan. 1985.

[GHJV94] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns, Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[GK91] D. Gelernter and D.L. Kaminsky. Supercomputing out of recycled garbage : Preliminary experience with Piranha. In *6th ACM Int. Conf. on Supercomputing*, 1991.

[GKYL00] J-P. Goux, S. Kulkarni, M. Yoder, and J. Linderoth. An Enabling Framework for Master-Worker Applications on the Computational Grid. In *HPDC'00 :*

Proc. of the 9^{th} *IEEE Intl. Symposium on High Performance Distributed Computing (HPDC'00)*, page 43, Washington, DC, USA, 2000. IEEE Computer Society.

[GPD03] Christian Gagné, Marc Parizeau, and Marc Dubreuil. Distributed BEAGLE : An Environment for Parallel and Distributed Evolutionary Computations. In *Proc. of the 17th Annual International Symposium on High Performance Computing Systems and Applications (HPCS) 2003*, May 11-14 2003.

[Gri02] A.S. Grimshaw. What is a Grid ? *Grid Today*, 1(26), 2002.

[GS01] L. Di Gaspero and A. Schaerf. Easylocal++ : An object-oriented framework for the design of local search algorithms and metaheuristics. In *MIC'2001 4th Metaheuristics International Conference*, pages 287–292, Porto, Portugal, July 2001.

[GW96] A.S. Grimshaw and W.A. Wulf. Legion–A View From 50,000 Feet. In *Proc. of the* 5^{th} *IEEE Intl. Symp. on High Performance Distributed Computing*, Los Alamitos, CA, Aug. 1996. IEEE Computer Society Press.

[GWB+04] M. Gerndt, R. Wismüller, Z. Balaton, G. Gombás, Z. Németh, N. Podhorski, H.-L. Truong, T. Fahringer, M. Bubak, E. Laure, and T. Margalef. *Performance Tools for the Grid : State of the Art and Future*, chapter Volume 30 of LRR-TUM Research Report Series35. Shaker Verlag, Aachen, 2004.

[GZBS86] A. Gamst, E-G. Zinn, R. Beck, and R. Simon. Cellular radio network planning. *IEEE AES Magazine*, pages 8–11, Feb. 1986.

[HCB04] F. Huet, D. Caromel, and H.E. Bal. A High Performance Java Middleware with a Real Application. In *Proc. of the Supercomputing Conference*, Pittsburgh, Pensylvania, USA, November 2004.

[HJMS+00] W. Hoscheck, J. Jaen-Martinez, A. Samar, H. Stockinger, and K. Stockinger. Data management in an international data Grid project. In *Proc. of the* 1^{st} *IEEE/ACM Intl. Work. on Grid Computing (Grid 2000)*, pages 77– ?90, Déc. 2000.

[Hol75] J.H. Holland. *Adaptation in natural and artificial systems.* Ann Arbor, MI, USA, The University of Michigan Press, 1975.

[IF00] A. Iamnitchi and I. Foster. A Problem-Specific Fault-Tolerance Mechanism for Asynchronous, Distributed Systems. In *Intl. Conf. on Parallel Processing*, pages 4–14, 2000.

[JAM88] V.K. Janakiram, D.P. Agrawal, and R. Mehrotra. A Randomized Parallel Branch-and-bound Algorithm. In *in Proc. of Int. Conf. on Parallel Processing*, pages 69–75, Aug. 1988.

[JF88] R. Johnson and B. Foote. Designing Reusable Classes. *Journal of Object-Oriented Programming*, 1(2) :22–35, June/July 1988.

[KBM02] K. Krauter, R. Buyya, and M. Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software - Practice & Experience*, 32(2) :135–164, 2002.

[KC99] J.D. Knowles and D.W. Corne. On Metrics for Comparing Non-Dominated Sets. In *In Proc. of the 2002 Congress on Evolutionary Computation (CEC02), IEEE Press*, pages 711–716, 1999.

[KJ97] R. Kohavi and G.H. John. Wrappers for Feature Subset Selection. *Artificial Intelligence*, 97(1–2) :273–324, 1997.

[KK84] V. Kumar and L. Kanal. Parallel Branch-and-Bound Formulations For And/Or Tree Search. *IEEE Trans. Pattern. Anal. and Machine Intell.*, PAMI–6 :768–778, 1984.

[KMRS01] M. Keijzer, J.J. Morelo, G. Romero, and M. Schoenauer. Evolving Objects : A General Purpose Evolutionary Computation Library. *Proc. of the* 5^{th} *Intl. Conf. on Artificial Evolution (EA'01), Le Creusot, France*, October 2001.

[KR87] S.A. Kravitz and R.A. Rutenbar. Placement by simulated annealing on a multiprocessor. *IEEE Transactions in Computer Aided Design*, 6 :534–549, 1987.

[KS00] Natalio Krasnogor and Jim Smith. MAFRA : A Java memetic algorithms framework. In Alex A. Freitas, William Hart, Natalio Krasnogor, and Jim Smith, editors, *Data Mining with Evolutionary Algorithms*, pages 125–131, Las Vegas, Nevada, USA, 8 2000.

[LBRT97] M. Livny, J. Basney, R. Raman, and T. Tannenbaum. Mechanisms for High Thoughput Computing. *SPEEDUP Journal*, 11(1), Jun. 1997.

[LDT04] J. Lemesre, C. Dhaenens, and E.G. Talbi. A parallel exact method for a bicriteria permutation flow-shop problem. In *In Proc. of Project Management and Scheduling (PMS'04)*, pages 359–362, Jul. 2004.

[LGT99] X. Lagrange, P. Goldlewski, and S. Tabbane. *Réseaux GSM-DCS*. Hermes Science, 4e édition revue et augmentée, 1999.

[LKGY00] J. Linderoth, S. Kulkarni, J.P. Goux, and M. Yoder. An Enabling Framework for Master-Worker Applications on the Computational Grid. In *Proc. of the* 9^{th} *IEEE Symposium on High Performance Distributed Computing (HPDC9)*, pages 43–50, Pittsburgh, PA, Aug. 2000.
`http ://www.cs.wisc.edu/condor/mw/`.

[LLM88] M. Litzkow, M. Livny, and M. Mutka. Condor - A Hunter of Idle Workstations. In *Proc. of the* 8^th *Intl. Conf. of Distributed Computing Systems*, Jun. 1988.

[LM98] H. Liu and H. Motoda. *Feature Selection for Knowledge Discovery and Data Mining*. Kluwer Academic Publishers, 1998.

[LPB+02] S. Luke, L. Panait, J. Bassett, R. Hubley, C. Balan, and A. Chicop. ECJ : A Java-based Evolutionary Computation and Genetic Programming Research system, 2002. http ://www.cs.umd.edu/projects/plus/ec/ecj/.

[LS96] H. Liu and R. Setiono. A Probabilistic Approach to Feature Selection - a Filter Approach. In *Proc. of the* 13^{th} *Intl. Conf. on Machine Learning*, 1996.

[MCTD02] N. Melab, S. Cahon, E-G. Talbi, and L. Duponchel. Parallel genetic algorithm based wrapper feature selection for spectroscopic data mining. In

IEEE BioSP3 Workshop on Biologically Inspired Solutions to Parallel Processing Problems (in conjunction with IEEE IPDPS2002). Fort-Lauderdale, USA, April 2002.

[MDLT96] N. Melab, N. Devesa, M-P. Lecouffe, and B. Toursel. Adaptive Load Balancing of Irregular Applications - A Case Study : IDA* Applied to the 15-Puzzle Problem. In *IRREGULAR'96, Springer Verlag, LNCS 1117*, pages 327–338, 1996.

[Mel97] Nouredine Melab. *Gestion de la granularité et régulation de charge dans le modèle P^3 d'évaluation parallèle des langages fonctionnels.* Université de Lille1, France, 1997.

[Meu02] H. Meunier. *Algorithmes évolutionnaires parallèles pour l'optimisation multi-objectif de réseaux de télécommunications mobiles.* PhD thesis, Université des Sciences et Technologies de Lille, 2002.

[MG95] S.W. Mahfoud and D.E. Goldberg. Paralell recombinative simulated annealing : A genetic algorithm. *Parallel Computing*, 21 :1–28, 1995.

[MH01] Laurent Michel and Pascal Van Hentenryck. Localizer++ : An open library for local search. Technical Report CS-01-02, Brown University, Computer Science, 2001.

[MMT05a] N. Melab, M. Mezmaz, and E-G. Talbi. Parallel Cooperative Meta-Heuristics on the Computational Grid. *Parallel Computing (under revision)*, 2005.

[MMT05b] M. Mezmaz, N. Melab, and E-G. Talbi. Optimisation parallèle hybride sur système pair à pair. In *RENPAR'16, Le Croisic, France*, 14-16 Feb. 2005.

[MMT05c] M. Mezmaz, N. Melab, and E-G. Talbi. Towards a Coordination Model for Parallel Cooperative P2P Multi-objective Optimization. In *In Springer Verlag LNCS 3470, Proc. of European Grid Conf. (EGC'2005), Amsterdam, The Netherlands*, pages 305–314, 2005.

[MOF92] O. Martin, S.W. Otto, and E.W. Felten. Large-step markov chains for the tsp : Incorporating local search heuristics. *Operation Research Letters*, 11 :219–224, 1992.

[MP93] D.L. Miller and J.F. Pekny. The Role of Performance Metrics for Parallel Mathematical Programming Algorithms. *ORSA J. Computing*, 5(1) :26–28, 1993.

[MTC04] N. Melab, E-G. Talbi, and S. Cahon. *Frameworks for the Design of Reusable Parallel and Distributed Metaheuristics*, chapter 35, Handbook of Bioinspired Algorithms and Applications. Edited by S. Olariu and A. Y. Zomaya, CRC Press, USA, 2004.

[MTR00] H. Meunier, El-Ghazali Talbi, and P. Reininger. A Multiobjective Genetic Algorithm for Radio Network Optimization. In *Congress on Evolutionary Computation*, volume 1, pages 317–324. IEEE Service Center, July 2000.

[NF77] P.M. Narendra and K. Fukunanga. A branch and bound algorithm for feature subset selection. In *IEEE Trans. on computer*, Septembre 1977.

[NGB04] Z. Németh, G. Gombás, and Z. Balaton. Performance Evaluation on Grids : Directions, Issues and Open Problems. In *12th Euromicro PDP*, pages 290–. IEEE Service Center, Feb. 2004.

[OTT98] S. Obayashi, S. Takahashi, and Y. Takeguchi. Niching and Elitist Models for MOGAs. In *PPSN'98, Springer Verlag, LNCS 1498*, pages 260–269, 1998.

[PA98] G.A. Papadopoulos and F. Arbab. Coordination models and languages. *Advances in Computers : The Engineering of Large Systems, Academic Press*, 46, 1998.

[PGS02] P. Primet, G.Romier, and M. Soberman. Le projet e-toile : Développement et mise en ¡uvre d ?une grille haute performance. In *Ateliers RNTL-ASTI, JOURNEES 2002 du RNTL*, Oct. 2002.

[PL95] J. Pruyne and M. Livny. Parallel processing on dynamic resources with CARMI. In *Proceedings of the Workshop on Job Sheduling for Parallel Processing IPPS'95*, 1995.

[PM98] G.T. Parks and I. Miller. Selective Breeding in a Multiobjective Genetic Algorithm. In A. E. Eiben, M. Schoenauer, and H.-P. Schwefel, editors, *Parallel Problem Solving From Nature — PPSN V*, pages 250–259, Amsterdam, Holland, 1998. Springer-Verlag.

[PPSS95] W. Pree, G. Pomberger, A. Schappert, and P. Sommerlad. Active Guidance of Framework Development. *Software - Concepts and Tools*, 16(3) :136–, 1995.

[PS82] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity.* Prentice-Hall, 1982.

[RC98] P. Reininger and A. Caminada. Model for gsm radio network optimization. In *2nd International Workshop on Discrete Algorithms and Methods for Mobile Computing and Communications. In conjunction with ACM/IEEE Mobicom'98*, 1998.

[RJ97] D. Roberts and R.E. Johnson. Evolving Frameworks : A Pattern Language for Developing Object-Oriented Frameworks. In *Pattern Languages of Program Design 3*. Addison Wesley, 1997.

[RS04] R. Ruiz and T. Stutzle. A Simple and Effective Iterative Greedy Algorithm for the Flowshop Scheduling Problem. *Technical Report, submitted to European Journal of Operational Research*, 2004.

[SN02] M. Sebban and R. Nock. A Hybrid Filter/Wrapper Approach of Feature Selection using Information Theory. *Journal of Pattern Recognition*, 35(4) :835–846, 2002.

[SNM+02] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova. Overview of GridRPC : A Remote Procedure Call API for Grid Computing. In Manish Parashar, editor, *Proc. of the 3rd Intl. Work. on Grid Computing (GRID 2002)*, pages 274–278, Baltimore,MD, USA, Nov. 2002. Springer.

[SNS+] M. Sato, H. Nakada, S. Sekiguchi, S. Matsuoka, U. Nagashim, and H. Takagi. Ninf : a n.

[SWM91] T. Starkweather, D. Whitley, and K. Mathias. Optimization using distributed genetic algorithms. In H.-P. Schwefel and R. Manner, editors, *Parallel Problem Solving from Nature*, volume 496, page 176. Lecture Notes in Compuer Science, Springer-Verlag, 1991.

[Tai93] E. Taillard. Parallel iterative search methods for vehicle routing problem. *Networks*, 23 :661–673, 1993.

[Tal02] E-G. Talbi. A Taxonomy of Hybrid Metaheuristics. *Journal of Heuristics, Kluwer Academic Publishers*, Vol.8 :541–564, 2002.

[TAML05] E-G. Talbi, E. Alba, N. Melab, and G. Luque. *Metaheuristics and Parallelism*, chapter 4, pages 79–103. In Wiley Book on Parallel Metaheuristics : A New Class of Algorithms, 2005.

[TB02] V. T'Kindt and J-C. Billaut. *Multicriteria Scheduling - Theory, Models and Algorithms*. Springer-Verlag, 2002.

[TGHK99] E-G. Talbi, J-M. Geib, Z. Hafidi, and D. Kebbal. MARS : An Adaptive Parallel Programming Environment. In R. Buyya, editor, *High Performance cluster computing*, volume 1, chapter 4. Prentice Hall PTR, 1999.

[The00] The MetaNEOS Project. Metacomputing Environments for Optimization. 2000. `http ://www.mcs.anl.gov/metaneos`.

[TMS94] E-G. Talbi, T. Muntean, and I. Samarandache. Hybridation des algorithmes génétiques avec la recherche tabou. In *Evolution Artificielle EA '94*, Toulouse, Sept. 1994.

[TNS+03] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka. Ninf-G : A Reference Implementation of RPC-based Programming Middleware for Grid Computing. *Journal of Grid Computing*, 1(1) :41–51, 2003.

[TTL02] D. Thain, T. Tannenbaum, and M. Livny. Condor and the Grid. In F. Berman, G. Fox, and T. Hey, editors, *Grid Computing : Making the Global Infrastructure a Reality*. John Wiley & Sons Inc., Dec. 2002.

[Vae95] R.J.M. Vaessens. Permutation flow shop bound found by branch and bound technique. *Personnal Ccommunication*, 1995.

[VDK92] H. Vafaie, J. De, and A. Kenneth. Genetic Algorithms as a Tool for Feature Selection in Machine Learning. In *Proc. of the Intl. Conf. on Tools with AI*, pages 200–204, Arlington, VA, 1992. IEEE Computer Society Press.

[VH00] M. Vasquez and J-K. Hao. A heuristic approach for antenna positioning in cellular networks. *Submitted to Journal of Heuristics*, 2000.

[VL00a] D. A. Van Veldhuizen and G. B. Lamont. On measuring multiobjective evolutionary algorithm performance. In *Proceedings of the 2000 Congress on Evolutionary Computation*, volume 1, Piscataway, New Jersey, july 2000. IEEE Service Center.

[VL00b] D.A. Van Veldhuizen and G.B. Lamont. On Measuring Multiobjective Evolutionary Algorithm Performance. In *In 2000 Congress on Evolutionary Computation, volume 1, IEEE Service Center*, pages 204–211, Piscataway, New Jersey, Jul. 2000.

[VNRS02] J. Verbeke, N. Nadgir, G. Ruetsch, and I. Sharapov. Framework for Peer-to-Peer Distributed Computing in a Heterogeneous, Decentralized Environment. *In Proc. of the Third Intl. Workshop on Grid Computing (GRID'2002), Baltimore, MD*, pages 1–12, Jan. 2002.

[VW02] S. Voss and D. Woodruff. *Optimization Software Class Libraries*. Kluwer Academic Publishers, 2002.

[WA04] S. Wagner and M. Affenzeller. Heuristiclab Grid - A Flexible And Extensible Environment For Parallel Heuristic Optimization. Technical report, Johannes Kepler University, Austria, 2004.

[Wal99] M. Wall. GAlib : A C++ Library of Genetic Algorithms Components., 1999. http ://lancet.mit.edu/ga/.

[WM95] D.H. Wolpert and W.G. Macready. No free lunch theorems for search. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[WRWD84] S. Wold, A. Ruhe, H. Wold, and W. J. Dunn, III. The collinearity problem in linear regression. the partial least squares (PLS) approach to generalized inverses. *SIAM Journal on Scientific and Statistical Computing*, 5(3) :735–743, Sep 1984.

[Zak99] Mohammed Javeed Zaki. Parallel and distributed data mining : An introduction. In *Large-Scale Parallel Data Mining*, pages 1–23, 1999.

[ZT99] E. Zitzler and L. Thiele. Multiobjective Evolutionary Algorithms : A Comparative Case Study and the Strength Pareto Approach. *IEEE Trans. on Evolutionary Computation*, 3 :257–271, 1999.