Université de Montréal

Modélisation UML pour une architecture coopérative appliquée au problème de tournées de véhicules avec fenêtres de temps

Par
Alexandre Le Bouthillier
Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Mémoire présenté à la Faculté des études supérieures
En vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en informatique.

Décembre 1999

Identification du jury

Université de Montréal

Faculté des études supérieures

Ce mémoire intitulé :

Modélisation UML pour une architecture coopérative appliquée au problème de tournées de véhicules avec fenêtres de temps

Présenté par :

Alexandre Le Bouthillier

A été évalué par un jury composé des personnes suivantes :

Jean-Yves Potvin	Président-rapporteur
Teodor Gabriel Crainic	Directeur de recherche
Rudolf K. Keller	Codirecteur de recherche
Nadia El-Mabrouk	Membre du jury

Mémoire accepté le :_____

Sommaire

Le problème de construction de tournées de véhicules avec fenêtres de temps (*VRPTW*) modélise bien un problème de transport très répandu qui est la livraison (ou la cueillette) de produits auprès de divers clients. Une façon de réduire les coûts de service est de trouver l'ordre de parcours des clients qui, pour un nombre minimal de véhicules, minimisera la distance totale parcourue tout en respectant les contraintes de temps des clients et de capacités des véhicules. Ce problème classique apparaît dans de nombreux problèmes réels de distributions : livraisons bancaires et postales, routes d'autobus scolaires, transport de passagers et de marchandises…

La résolution de ce problème d'ordonnancement NP-DUR par des méthodes exactes nécessite des temps de calculs prohibitifs pour les problèmes de taille réelle. L'utilisation de méthodes approximatives qui tentent d'exploiter une certaine structure dans l'espace de recherche de solutions, en occurrence les métaheuristiques, fournissent parfois les seuls résultats de qualité acceptable dans des temps raisonnables. La résolution de ces problèmes réels permet aux industries et aux gouvernements d'économiser des millions de dollars chaque année en frais de transport tout en minimisant les impacts environnementaux.

Par contre, aucune métaheuristique ne fournit à elle seule tous les meilleurs résultats des problèmes présentés dans la littérature. Il n'y a donc pas de métaheuristique universelle mais plutôt un sous-ensemble de metaheuristiques qui, prises individuellement, fonctionnent très bien pour certaines instances de *VRPTW*. En plus d'être sensibles aux méthodes de résolution, la qualité des solutions produite à l'aide de métaheuristiques est aussi influencée par les paramètres utilisés. Ainsi, un ensemble de paramètres pour une métaheuristique donnée fonctionnera bien pour un sous-ensemble d'instance de *VRPTW* mais moins bien pour d'autres. Les solutions aux *VRPTW* sont ainsi fortement biaisées par ces paramètres et il est souvent nécessaire d'effectuer un changement dynamique de ces derniers pour s'adapter à chaque problème.

Nous proposons une combinaison de diverses métaheuristiques dans un environnement coopératif distribué où chaque métaheuristique utilise différentes configurations de paramètres. Cette combinaison est modélisée sous UML (Unified Modelling Language), conceptualisant ainsi le mécanisme de parallélisation asynchrone de diverses métaheuristiques prometteuses (recherches avec tabous et algorithmes évolutifs). Cette modélisation nous permettra d'identifier clairement la méthode de coopération entre les diverses métaheuristiques impliquées et ainsi d'en faciliter l'implantation. Nous démontrerons ainsi que cette combinaison de métaheuristiques et de paramètres, en plus d'offrir une plus grande rapidité d'exécution, permet d'offrir une plus grande robustesse de qualité de solution de *VRPTW* en comparaison avec les métaheuristiques prises individuellement.

Finalement, nous proposons une modification d'une mesure tridimensionnelle pour le calcul et la représentation des distances en combinant les mesures de distances et de fenêtres de temps en une seule métrique. Il est ainsi possible d'obtenir une visualisation plus naturelle du *VRPTW* ainsi qu'une méthode pour catégoriser les problèmes et les solutions selon leurs niveaux de contraintes.

Table des matières

IDENTIFICATION DU JURY II

SOMMAIRE III

LISTE DES TABLEAUX X

LISTE DES FIGURES XI

LISTE DES FIGURES XI

Liste des équations *XII*

LISTE DES SIGLES, ABREVIATIONS ET DEFINITIONS XIV

DEDICACE XV

REMERCIEMENTS XVI

AVANT-PROPOS XVII

Chapitre 1 Introduction 1

Chapitre 2 Revue de la littérature sur la représentation et la modélisation d'architectures réparties par UML 5

2.1 DEFINITION SUCCINCTE DE UML 5

2.2 UML DISTRIBUE TEMPS REEL 6

2.3 REVUE DES MODELES ET DIAGRAMMES DE BASE DE UML 7

2.3.1 Modèle de classe (exemple d'un cas général) *7*

2.3.2 Modèle de rôle (exemple d'un cas général) *9*

2.3.3 Modèles d'instance (exemple d'un cas général) *10*

2.3.4 Diagramme d'état (exemple d'un cas général) *11*

2.3.5 Diagramme d'activité *12*

2.3.6 Diagramme de composantes *13*

2.3.7 Diagramme de déploiement *14*

2.4 CONSTATATIONS 15

Chapitre 3 Formulation du VRPTW 16

3.1 FORMULATION 16

3.2 FORMULATION 18

3.2.1 Détail des équations *19*

3.2.2 Modification de la métrique de distance *20*

3.2.2.1 Implication de la métrique 21
3.2.3 Visualisation 2D du VRPTW *22*
3.2.4 Visualisation 3D de deux clients *23*
3.3 COMPLEXITE 24
Chapitre 4 Revue de la littérature VRPTW 25
4.1 VRP 25
4.1.1 Heuristiques *26*
4.1.1.1 Heuristiques de construction 26
4.1.1.2 Algorithmes de gain 26
4.1.1.2.1 Clarke et Wright 26
4.1.1.3 Méthode d'insertion séquentielle 27
4.1.1.4 Algorithme groupe en premier, route en second 27
4.1.1.5 Algorithme route en premier, groupe en second 28
4.1.1.5.1 Algorithme de balayage 28
4.1.1.6 Algorithme en pétale 28
4.1.2 Heuristiques d'amélioration *28*
4.1.3 Méthodes hybrides *29*
4.1.3.1 CCAO 29
4.1.3.2 GENIUS 30
4.1.3.2.1 GENI 30
4.1.3.2.2 US 32
4.1.3.2.2.1 Retrait de type I 33
4.1.3.2.2.2 Retrait de type II 33
4.1.4 Métaheuristiques *34*
4.1.4.1 Recherches avec tabou (TS) 34
4.1.4.1.1 Taburoute 34
4.1.4.2 Algorithme de Taillard 35
4.1.4.3 Algorithme de Xu et Kelly 35
4.1.4.4 Algorithme de Rego et Roucairol 36
4.1.4.5 Recherche granulaire de Toth et Vigo 36
4.1.4.6 Algorithmes évolutifs 36
4.2 VRPTW 37
4.2.1 Construction de routes *37*
4.2.1.1 Génération par villes (DENN, Double-Ended Nearest Neighbour) 38
4.2.1.2 Génération par arc (MF, Multiple Fragment) 38
4.2.1.3 Génération par arc hybride (SAH, Shortest Arcs Hybridation) 38
4.2.1.4 Génération aléatoire 39
4.2.2 Amélioration de routes *39*

4.2.3 Combinaison d'heuristiques 39
4.2.4 Métaheuristiques 40
4.2.4.1 Algorithmes évolutifs 41
4.2.4.1.1 Encodage 41
4.2.4.1.2 Solutions initiales 42
4.2.4.1.3 Opérateur de sélection 42
4.2.4.1.4 Opérateur de croisement 42
4.2.4.1.4.1 OX, Order crossover 43
4.2.4.1.4.2 ER, Edge recombination crossover 43
4.2.4.1.5 Opérateur de réparation 44
4.2.4.1.6 Opérateur de mutation 45
4.2.4.1.7 Descentes locales 45
4.2.4.2 Recherche avec tabous 46
4.2.4.2.1 Recherche avec tabous pour le VRPSTW (Vehicule Routing Problem with Soft Time Windows) 48
4.2.4.2.2 Approche unifiée pour le *VRPTW*, le *MDVRPTW* et le *PVRPTW* 49
4.2.4.2.3 Recherche réactive avec recherches tabous 49
4.2.4.3 Mémoire adaptative de Rochat et Taillard 50
4.2.4.4 Coopération entre recherche avec tabous et algorithmes évolutifs 51
4.2.4.4.1 Phase I 51
4.2.4.4.2 Phase II 52
4.2.4.4.3 Parallélisation de recherches avec tabous et d'algorithmes évolutifs 52
4.2.4.5 Programmation par contraintes 53
4.3 PRINCIPALES EXTENSIONS AU VRPTW 54
4.4 CHOIX DES ALGORITHMES 54
Chapitre 5 Coopération entre algorithmes avec recherches tabous et algorithmes évolutifs 55
5.1 MOTIVATIONS 55
5.2 RAPPELS THEORIQUES DE PARALLELISME 56
5.2.1 Conditions de Bernstein 56
5.2.2 Complexité parallèle 56
5.2.3 Finesse du grain 57
5.2.4 Type de parallélisation 58
5.2.5 Méthode de parallélisation 58
5.2.5.1 Parallélisation applicative 58
5.2.5.2 Parallélisation intégrée 59
5.2.5.3 Parallélisation réflexive 59

5.3 MEMOIRE CENTRALE ET METAHEURISTIQUES 59
5.3.1.1 Diagramme de collaboration encapsulée 60
5.3.2 Collaboration entre algorithmes *61*
5.3.3 Séquence d'exécution *63*
5.4 METHODES DE RESOLUTIONS : ALGORITHMES AVEC RECHERCHES TABOUS ET ALGORITHMES EVOLUTIFS 65
5.5 ARCHITECTURE LOGICIELLE 65
5.5.1 Mémoire centrale *65*
5.5.1.1 Populations de solutions de la mémoire centrale 66
5.5.1.2 Fonction de coût du classement dans la *mémoire centrale* 66
5.5.1.3 Communications asynchrones 67
5.5.2 Description de l'algorithme de construction vorace C3D *68*
5.5.3 Capsule constructive *69*
5.5.4 Capsule évolutive *70*
5.5.4.1 Opérateur de croisement et de réparation 70
5.5.4.2 Opérateur de sélection 70
5.5.4.3 Opérateur de mutation 71
5.5.5 Capsule Taburoute *72*
5.5.6 Composantes distinctes : Mémoire et capsules *73*
5.6 IMPLANTATIONS 74
5.6.1 Environnement coopératif *74*
5.6.2 Technologies *75*
5.7 DEVELOPPEMENT STRUCTURE 75
5.8 OBSERVATIONS SUR UML 76
Chapitre 6 Résultats, contributions et conclusions 77
6.1 EXPERIMENTATION 77
6.2 PRESENTATION DES RESULTATS 77
6.3 RESULTATS NUMERIQUES 78
6.4 PARAMETRES 79
6.4.1 Taburoute I *79*
6.4.2 Taburoute II *79*
6.4.3 Algorithme évolutif I *80*
6.4.4 Algorithme évolutif II *80*

6.5 TEMPS DE CALCULS REQUIS 80
6.6 TABLEAU DES RESULTATS 81
6.6.1 Nouvelle solution *85*
6.6.2 Interprétation *86*
6.7 TRAVAUX FUTURS 86
6.7.1 Recherche dispersée *87*
6.7.2 Combinaison avec d'autres capsules *87*
6.7.3 Balancement de charge de la mémoire centrale *88*
6.7.4 Migration des paramètres ou changement de processus *88*
6.7.5 Proximité des solutions dans l'espace *88*
6.8 CONTRIBUTIONS ET CONCLUSIONS 89
6.9 BIBLIOGRAPHIE 91
Chapitre 7 Annexe 98

Liste des tableaux

Tableau 1 Modificateur de visibilité 8

Tableau 2 Visibilité par défaut 8

Tableau 3 Résultats moyens des méthodes indépendantes par classe de problèmes de 100 villes et 1 dépôt de Solomon 82

Tableau 4 Résultats séquentiels (CR, CLM, RGP), parallèles (RT,TB,HG) et indépendants (LCKIND) moyens par classe de problèmes de 100 villes et 1 dépôt de Solomon 83

Tableau 5 Résultats préliminaires parallèles moyens par classe de problèmes de 1000 villes et 1 dépôt de Solomon 84

Tableau 6 Résultats par problèmes de classe R1 de 100 villes et 1 dépôt de Solomon 98

Tableau 7 Résultats par problèmes de classe C1 de 100 villes et 1 dépôt de Solomon 99

Tableau 8 Résultats par problèmes de classe RC1 de 100 villes et 1 dépôt de Solomon 100

Tableau 9 Résultats par problèmes de classe R2 de 100 villes et 1 dépôt de Solomon 101

Tableau 10 Résultats par problèmes de classe C2 de 100 villes et 1 dépôt de Solomon 102

Tableau 11 Résultats par problèmes de classe RC2 de 100 villes et 1 dépôt de Solomon 103

Liste des figures

Figure 1 Relation client/serveur 7
Figure 2 Diagramme de collaboration (niveau d'instance : objets, liens et stimuli) (UML v1.3) 9
Figure 3 Diagramme de collaboration (niveau de spécification : rôle de classification, d'association et messages) (UML v1.3) 10
Figure 4 Diagramme de séquence (UML v1.3) 10
Figure 5 Diagramme d'état (UML v1.3) 11
Figure 6 Diagramme d'activité (UML v1.3) 12
Figure 7 Diagramme de composante (UML v1.3) 13
Figure 8 Diagramme de déploiement (UML v1.3) 14
Figure 9 Coupe en espace-temps d'un parcours entre le nœud i et j 21
Figure 10 Visualisation 2D du VRPTW (problème C101) 22
Figure 11 Visualisation 3D de deux clients 23
Figure 12 Insertion GENI de type I 32
Figure 13 Insertion GENI de type II 32
Figure 14 Diagramme de collaboration encapsulée 60
Figure 15 Diagramme de collaboration (niveau de spécification) 61
Figure 16 Diagramme de collaboration (niveau d'instance) 62
Figure 17 Diagramme de séquence 64
Figure 18 Diagramme d'état de l'entité mémoire centrale 67
Figure 19 Diagramme d'état de l'entité capsule constructeur 69
Figure 20 Diagramme d'état de l'entité capsule évolutive 71
Figure 21 Diagramme d'état de l'entité capsule tabou 72
Figure 22 Diagramme de composantes 73
Figure 23 Diagramme de déploiement 74

Liste des équations

$$Min\sum_{v\in V}\left[\sum_{j\in C}x^{v}_{0,j}+\sum_{i\in N}\sum_{j\in N}c_{i,j}x^{v}_{i,j}\right] \quad (3\text{-}1)$$ 18

$$\sum_{v\in V}\sum_{j\in N}x^{v}_{i,j}=1 \qquad \forall i\in C \quad (3\text{-}2)$$ 18

$$\sum_{j\in N}x^{v}_{i,j}-\sum_{j\in N}x^{v}_{j,i}=0 \qquad \forall i\in C, v\in V \quad (3\text{-}3)$$ 18

$$\sum_{j\in N}x^{v}_{0,j}=1 \qquad \forall v\in V \quad (3\text{-}4)$$ 18

$$\sum_{j\in N}x^{v}_{j,n+1}=1 \qquad \forall v\in V \quad (3\text{-}5)$$ 18

$$x^{v}_{i,j}=1\Rightarrow \quad b_i+s_i+t_{i,j}\le b_j \qquad \forall i,j\in C, v\in V \quad (3\text{-}6)$$ 18

$$x^{v}_{0,j}=1\Rightarrow \quad b^{v}_{0}+t_{0,j}\le b_j \qquad \forall j\in C, v\in V \quad (3\text{-}7)$$ 18

$$x^{v}_{i,n+1}=1\Rightarrow \quad b_i+s_i+t_{i,n+1}\le b^{v}_{n+1} \qquad \forall i\in C, v\in V \quad (3\text{-}8)$$ 18

$$e_i\le b_i\le l_i \qquad \forall i\in C \quad (3\text{-}9)$$ 18

$$e_0\le b^{v}_{0}\le l_0 \qquad \forall v\in V \quad (3\text{-}10)$$ 18

$$e_{n+1}\le b^{v}_{n+1}\le l_{n+1} \qquad \forall v\in V \quad (3\text{-}11)$$ 18

$$x^{v}_{i,j}=1\Rightarrow \quad y_i-d_j=y_j \qquad \forall i,j\in N, v\in V \quad (3\text{-}12)$$ 18

$$y_0=Q, 0\le y_i \qquad \forall i\in C \quad (3\text{-}13)$$ 18

$$x^{v}_{i,j}\in\{0,1\} \qquad \forall i,j\in N, v\in V \quad (3\text{-}14)$$ 18

$$\sum_{i\in X}\sum_{j\in X}x^{v}_{i,j}\le |X|-1 \qquad \forall i,j\in N, v\in V, \forall X\subseteq N \quad (3\text{-}15)$$ 18

$$coût(X)=\sum_{v\in V}\sum_{i\in N}\sum_{j\in N}c_{i,j}x^{v}_{i,j} \quad (3\text{-}16)$$ 19

$$no\_véhicules(X)=\sum_{v\in V}\sum_{j\in C}x^{v}_{0,j} \quad (3\text{-}17)$$ 19

$$temps\_horaire(X)=\sum_{v\in V}(b^{v}_{n_v+1}-b^{v}_{0}) \quad (3\text{-}18)$$ 19

$$C3d_{i,j}=\begin{cases}\infty \text{ si } b_j>l_j\\ \alpha(c_{(x_i,y_i,b_i+s_i),(x_j,y_j,b_j)})+\beta(l_j-a_j)+\chi w_j\end{cases} \quad (3\text{-}19)$$ 20

$$Min\sum_{v\in V}\left[\sum_{j\in C}x^{v}_{0,j}+\sum_{i\in N}\sum_{j\in N}C3d_{i,j}*x^{v}_{i,j}\right] \quad (3\text{-}20) \qquad 21$$

Liste des sigles, abréviations et définitions

Algorithme évolutif : Algorithme génétique

Applet : Programme Java en bytecode s'exécutant avec un fureteur

Mémoire centrale: Espace mémoire permettant d'emmagasiner des solutions qui peuvent être reçues ou transmises à différents processus.

Bytecode : Code que la JVM peut interpréter

Capsule : Agent, processus autonome effectuant certains processus et possédant son propre espace-temps-mémoire

JVM : Java Virtual Machine

ORB : Object Request Broker

ROOM : Real-Time Object-Oriented Modeling Language

TSP : Problème du voyageur de commerce (Travelling Salesman Problem)

UML : Unified Modelling Language

VRP : Problème de construction de tournées de véhicules (Vehicule Routing Problem)

VRPTW : Problème de tournées de véhicules avec fenêtres de temps (Vehicule Routing Problem with Time Windows)

Dédicace

Aux deux êtres qui ont donné un sens à ma vie : à ma mère, à mon père.

Remerciements

Les plus sincères remerciements sont pour mon directeur et mon codirecteur de maîtrise. Mon directeur, Dr Teodor Gabriel Crainic, a su me donner la conviction de mes intérêts et a toujours été un modèle de réussite.
Mon codirecteur, Dr Rudolf Keller, a permis une collaboration inédite entre le génie logiciel et la recherche opérationnelle. Il fut très ouvert d'esprit et a su transcender sa passion.

Ce mémoire à vu le jour grâce au support et l'appui financier de la «bourse du mérite académique du Centre de Recherche sur les Transports» ainsi que de la bourse CTRF/GRTC-Bombardier.

Un remerciement tout spécial à mon stagiaire, Paul-Hubert Sirois, qui a su explorer la technologie Java3D; une pensée pour les administrateurs du réseau du CRT (Daniel, Guy et Pierre). pour quelques codes sources fournis par François Guertin, pour le support et le soutien de ma famille qui m'a permis de persévérer ainsi que les discussions avec mon collègue et ami Louis-Martin Rousseau qui ont permis une approche critique et constructive.

J'ai une pensée spéciale pour ma fiancée qui s'est montrée compréhensive, patiente et parfois même intéressée!

Dr Gendreau, Dr Potvin, Dr Laporte, ont été d'une influence non négligeable dans ce domaine et leur présence au CRT fut très appréciée, particulièrement le cours gradué de Dr Potvin sur les métaheuristiques. Bien sûr, d'autres sommités, Dr Solomon, Dr Lin, Dr Kernighan, Dr Taillard, Dr Badeau, Dr Desrosiers, Dr Rousseau, pour n'en nommer que quelques-uns, ont su guider cette recherche par leurs écrits et leurs conférences.

En terminant, mentionnons le support des compagnies DRA system et ObjectSpace pour les librairies utilisées.

Avant-propos

Au sein de notre société, les récents changements nous ont amenés de l'ère industrielle à l'ère de l'information en moins de cinquante ans. L'organisation de cette information et de son utilisation globale nous conduit à ce que certains appèlent l'ère de l'optimisation. L'ère de l'optimisation apporte une dimension quantitative à cette information et permet de mesurer l'impact des processus de planification.

Cette sensibilisation à la nécessité, d'une part de l'optimisation et d'autre part d'implantations formalisées répondant aux exigences de systèmes spécifiques, donne naissance à une combinaison prometteuse.

Dans une optique de recherche, il a été proposé d'évaluer l'impact d'une modélisation UML supportant un cadre de résolution coopératif distribué pour un problème bien particulier de recherche opérationnelle : le problème de tournées de véhicules avec fenêtres de temps.

Chapitre 1 Introduction

Le transport est essentiel au bon fonctionnement de notre économie de marché. Il permet l'approvisionnement des industries et la distribution des biens ainsi que le déplacement d'individus. Le *VRPTW* (problème de construction de tournées de véhicules avec fenêtres de temps) modélise bien un problème de transport très répandu qui est la livraison (ou la cueillette) de produits auprès de divers clients avec un nombre minimal de véhicules tout en minimisant la distance totale parcourue, en respectant les contraintes de temps des clients et de capacités des véhicules. Les transporteurs utilisant des algorithmes leur fournissant des solutions optimales ou s'en approchant (comportant une distance totale minimale pour un nombre de véhicule minimal) obtiennent des réductions de l'ordre de 10-15% par rapport aux solutions confectionnés manuellement selon Rousseau (1999). En plus de minimiser l'impact sur l'environnement, ces réductions permettent des économies d'échelle lorsqu'on sait que les transporteurs routiers canadiens de marchandises ont effectué plus de 14 milliards de dollars canadiens de recettes en 1997 selon Statistiques Canada (2000) et plus de 650 milliards aux États-Unis en 1983 soit approximativement 21% du PNB selon le rapport Kearney (1984) et 22.5% des coûts de productions des produits manufacturés.

La quête d'optimalité pour ces problèmes (ou de solutions s'en approchant), provient d'une part des enjeux économiques, comme nous l'avons remarqué mais d'autre part d'enjeux scientifiques. En prenant un petit problème de 100 clients où on doit trouver l'ordre de visite de ses clients ainsi que l'affectation des clients à des véhicules, on remarque qu'une énumération exhaustive des combinaisons de parcours dépasse les 100!, énumération dont le temps est prohibitif pour n'importe quel ordinateur contemporain. Pour surmonter cette difficulté, de nombreuses méthodes de recherche opérationnelle permettent une exploration plus efficace de l'espace des solutions. Par sa formulation, le *VRPTW* est dans la classe des problèmes NP-DUR; il est donc peu probable qu'un algorithme polynomial en temps existe pour ce problème (et par le fait même pour tous les autre problèmes de cette classe). Ainsi, les méthodes exactes de recherche

opérationnelle se limitent à la nature combinatoire de ces problèmes, rendant la résolution des problèmes réels difficiles par les temps de résolution prohibitifs.

L'apparition de métaheuristiques telles que les recherches avec tabous de Gendreau et al.(1994), de Garcia et al. (1993) et Glover (1975) et les algorithmes génétiques de Potvin et al. (1996), Benyahia et al. (1995) a produit des résultats fort prometteurs sur les instances de littérature de Solomon (1983). Les métaheuristiques permettent une exploration de l'espace de solution basée sur des notions abstraites de la distribution des solutions dans l'espace, de l'évolution prévue de la recherche ainsi que de la nature du problème. Généralement, ces méthodes permettent de fournir des solutions de qualité acceptable souvent très près de l'optimal avec une fraction du temps requis par les méthodes exactes. Le désavantage des métaheuristiques relève de leur sensibilité aux différents problèmes, nécessitant ainsi une fine calibration des paramètres d'exécution et souvent des changements dynamiques de ces paramètres pour s'adapter à chaque problème selon Benyahia et al. (1993).

Afin de contourner ce manque de robustesse, nous considérons une architecture logicielle coopérative initialement proposée par Crainic et al.(1997) permettant une combinaison parallèle de métaheuristiques telles que recherches avec tabous de Gendreau et al. (1994) et algorithme évolutif de Potvin et al. (1996). Nous croyons que la combinaison d'algorithmes évolutifs et de recherches avec tabous peut être profitable au sens où les divers algorithmes évolutifs peuvent servir de pont entre diverses recherches tabous et ainsi combiner leurs efforts dans une architecture coopérative tel que décrit dans Crainic et al. (1997). Les recherches avec tabous explorent le voisinage de solutions alors que les algorithmes évolutifs tentent d'extraire les propriétés avantageuses de paires de solutions. Nous avons initialement retenu une méthode de recherche avec tabous de Gendreau et al. (1994) et des algorithmes génétiques de Potvin et al. (1996) qui, séparément, fournissaient de bons résultats.

Les instances de littératures étant très différentes les unes des autres par leurs caractéristiques de distributions spatiales et de restrictions sur les fenêtres de temps, il est

souvent nécessaire de créer plusieurs ensembles de paramètres. L'approche retenue, où nous avons différentes instances du même algorithme avec des paramètres différents, et un échange de solution via une mémoire centrale, permet une plus grande robustesse aux différentes instances de problèmes selon Toulouse et al. (1998) et une diversification des solutions, renforçant ainsi les mécanismes prévenant la convergence prématurée des solutions vers un espace restreint d'exploration. Les expériences dans ce type de parallélisation à gros grain se sont donc avérées plus efficaces en terme de résultats numériques lorsqu'il y avait présence de multi-départs/multi-stratégies selon Crainic et al. (1993) et Crainic (1997).

Au sein de notre recherche, nous remarquons que la conception UML (Unified Modelling Language) permet une représentation de plusieurs aspects des modèles de programmation coopérative selon Oldevik et al. (1998) et qu'elle nous fournit une modélisation adéquate de notre architecture. En planifiant l'architecture de communication sous les divers aspects proposés par UML, il est possible de prévoir et de concevoir à priori l'ensemble des communications inhérentes à la parallélisation. La syntaxe de représentation proposée par UML a su modéliser les divers aspects de la parallélisation, à savoir, les modèles de classes, de rôle, les diagrammes d'instance, d'état et d'activité. Nous proposons ainsi une architecture logicielle permettant à des méthodes de résolution indépendantes d'interagir sans synchronisation par des échanges de solutions via une mémoire centrale qui constitue une population de solutions communes. Il est ainsi relativement facile d'ajouter d'autres méthodes de résolutions ou d'utiliser des paramètres différents. Les résolutions regroupent donc un ensemble de paramètres qui sont effectifs pour l'ensemble des jeux de données. Il n'y a donc pas d'ajustement à effectuer au niveau des paramètres pour la résolution de nouvelles instances de problèmes. Ainsi, nous regardons la modélisation UML au chapitre 2, la formulation du *VRPTW* et sa représentation en trois dimensions au chapitre 3, effectuons une brève revue de littérature sur le *VRPTW* et définissons les métaheuristiques utilisées au chapitre 4. Finalement au chapitre 5, nous modélisons notre architecture logicielle à l'aide de *UML* en détaillant la méthode de modélisation, de l'architecture retenue, des avantages d'une telle architecture et de sa généralisation possible à d'autres problèmes utilisant les

métaheuristiques. Au dernier chapitre, nous présentons les résultats sur les problèmes classiques de littérature en spécifiant les paramètres utilisés et en fournissant les travaux futurs pertinents.

Chapitre 2 Revue de la littérature sur la représentation et la modélisation d'architectures réparties par UML

Tel que mentionné par OMG (Object Management Group www.omg.org), la valeur stratégique de plus en plus importante des logiciels pour nombre d'industries, poussent vers le développement des techniques d'automatisation de production, d'amélioration de la qualité logicielle et de diminution des coûts de déploiement. Ces techniques incluent les technologies de composantes, la programmation visuelle, les patrons et les cadres de travail. Au sein de ces techniques, il y a émergence d'une notion de gestion de complexité des systèmes et d'un besoin de résolution de problèmes logiciels architecturaux récurrents reliés aux systèmes d'envergure parmi lesquels on peut distinguer la distribution, la concurrence, la réplication, la sécurité, le balancement de charge ainsi que la tolérance aux pannes.

La récente poussée du *World Wide Web* a accentué ces besoins architecturaux et c'est ainsi que la norme émergente UML (Unified Modeling Language : voir spécifications détaillées www.omg.org) gagne en popularité auprès des communautés, tant universitaire qu'industrielle (Digital, HP, i-Logix, IntelliCorp, IBM, ICON, MCI, Microsoft, Oracle, Rational Software, TI et Unisys) www.omg.org (2000).

2.1 Définition succincte de UML

La modélisation assure un développement uniforme et prévisible, éléments essentiels à mesure que croît la complexité d'un système. Ainsi, un langage de modélisation adéquat doit contenir les éléments du modèle, les concepts fondamentaux et la sémantique de modélisation, la notation, la représentation graphique des éléments du modèle, les directives générales et l'usage de mots clef dans des contextes définis. Dans ce contexte, UML est une synthèse en maturation de plusieurs notations graphiques développées en génie logiciel pour des analyses orientées objet et des méthodes de conception qui répond très bien à ces critères. UML est un langage de modélisation permettant la spécification,

la construction, la visualisation et la documentation de logiciels, de modèles d'affaire ainsi que de systèmes non-logiciels.

Développé par Booch, Rumbaugh et Jacobson, UML unifie les concepts provenant de OMT de Booch et al. (1995), les *use case* de Jacobson et al. (1992), le CRC (Class-Responsibility-Collaboration) de la communauté Smalltalk, les *Statecharts* de Harel et al. (1987) et plusieurs autres, rendant ainsi UML un standard de l'industrie approuvé par le comité de normalisation OMG (Object Management Group).

UML à été conçu dans le but de fournir un modèle expressif visuel, une possibilité d'extension et de spécialisation des concepts prédéfinis, une indépendance face aux langages de programmation et au processus de développement, une base formelle pour la compréhension des langages de modélisation ainsi qu'un support au développement des concepts de haut-niveau tels la collaboration, les cadres de travail, les patrons et les composantes.

2.2 UML distribué temps réel

Les définitions d'UML (v1.3 à ce stade) offrent les primitives des modèles distribués et temps réel. Dans cette ligne de pensée, Rational Software Corp. et ObjectTime Ltd. ont produit une version temps réel de UML et une suite d'outils développés au sein d'une application de modélisation UML orienté vers des modèles temps réel : Rose RealTime 98i (www.omg.org).

UML temps réel (www.omg.org) est une combinaison d'UML (v1.1), de modèle de rôle et de ROOM (Real-Time Object-Oriented Modeling language). Cette modélisation du développement de logiciels temps réel permet de mieux saisir les impacts de conception qui réfèrent souvent à la structure, aux cadres de travail et aux patrons de conception inhérents à l'architecture coopérative selon Kahkipuro (1999). Les modèles inhérents à UML temps réel permettent d'explorer les différentes perspectives des systèmes complexes.

UML devrait faciliter le cycle de développement logiciel d'une architecture coopérative et permettre la réutilisation du cadre d'application pour d'autres architectures utilisant les mêmes concepts de résolution.

2.3 Revue des modèles et diagrammes de base de UML

Nous examinons les méthodes de modélisation permettant de représenter une architecture logicielle coopérative soit les modèles de classe, de rôle, d'instance, diagramme d'état, d'activité, de composantes et de déploiement. Pour chacun de ces modèles, nous présentons la définition, la fonction et un exemple simple tel que fournis par UML (www.omg.org). Au chapitre six, nous reprendrons ces mêmes méthodes de représentation au sein de notre architecture coopérative répartie.

2.3.1 Modèle de classe (exemple d'un cas général)

Niveau de représentation du système : haut

Représentation : diagramme de classe

Définition : Graphe des éléments classifiés reliés par leurs différentes relations statiques. Ce diagramme peut contenir des *interfaces*, des *package,* des *relations* et des *instances* tel des *objets* et des *liens*. On peut parler ici d'un diagramme de structure statique.

Fonction : En définissant les relations universelles entre les ensembles de classe, on effectue une représentation des constructions possibles à partir de ces dernières. La représentation se fait sur les relations et la décomposition des classes et non sur les relations spécifiques de communications.

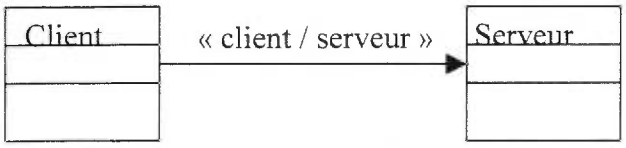

Figure 1 Relation client/serveur

Explication : La relation de clientèle existe entre une classe nommée client et une classe nommée serveur (fig. 1). Cette relation statique est dénotée par le stéréotype « client/serveur » qui peut être détecté à la compilation tel que décrit par Mandel (1999).

Types : Deux types de classes sont supportés dans les stéréotypes UML. Ce sont les classes « actives » et « synchrones ».

Visibilité : Il est possible de spécifier des changements aux visibilités traditionnelles des classes, des méthodes et/ou des attributs (voir tableau I et II). Les types de visibilités tels qu'utilisés par Java sont : public, privé, protégé et par défaut (tableau I). Si aucun modificateur n'est utilisé (identifié sous le nom de la classe ou comme préfixe de méthode ou d'attribut), la valeur de visibilité par défaut est attribuée selon qu'il s'agit d'une classe, d'une méthode ou d'un attribut
(tableau II).

Modificateur	Symbole
Public	+
Privé	-
Protégé	#
Défaut	

Tableau 1 Modificateur de visibilité

Modificateur	Classe	Méthode	Attribut
Final	*	*	*
Natif		*	
Synchronisé	*	*	
Passager			*
Volatile			*
Abstrait	*	*	
Statique	*	*	*

Tableau 2 Visibilité par défaut

2.3.2 Modèle de rôle (exemple d'un cas général)

Niveau de représentation du système : intermédiaire

Représentation : diagramme de collaboration

Définition : Une collaboration définit un ensemble de participants et de relations ayant un ensemble de buts communs. Le comportement est implanté par un ensemble d'objets qui échangent un stimulus dans une interaction globale conduisant à un but particulier. Les participants définissent le rôle des objets en interaction. Ainsi une collaboration spécifie un ensemble de rôles classifiés et de rôles d'associations. Les liaisons entre les objets et leurs rôles de classification définissent leur classification tandis que les liens entre les objets se conforment aux rôles d'association de la collaboration.

Fonction : Le diagramme de collaboration montre l'interaction entourant les rôles et leurs liens réciproques. Contrairement à un diagramme de séquence, un diagramme de collaboration présente les relations entre les objets jouant différents rôles. Par contre, il ne présente pas le temps comme une dimension séparée, il faut donc numéroter la séquence d'interaction entre les différents processus (fig. 2,3).

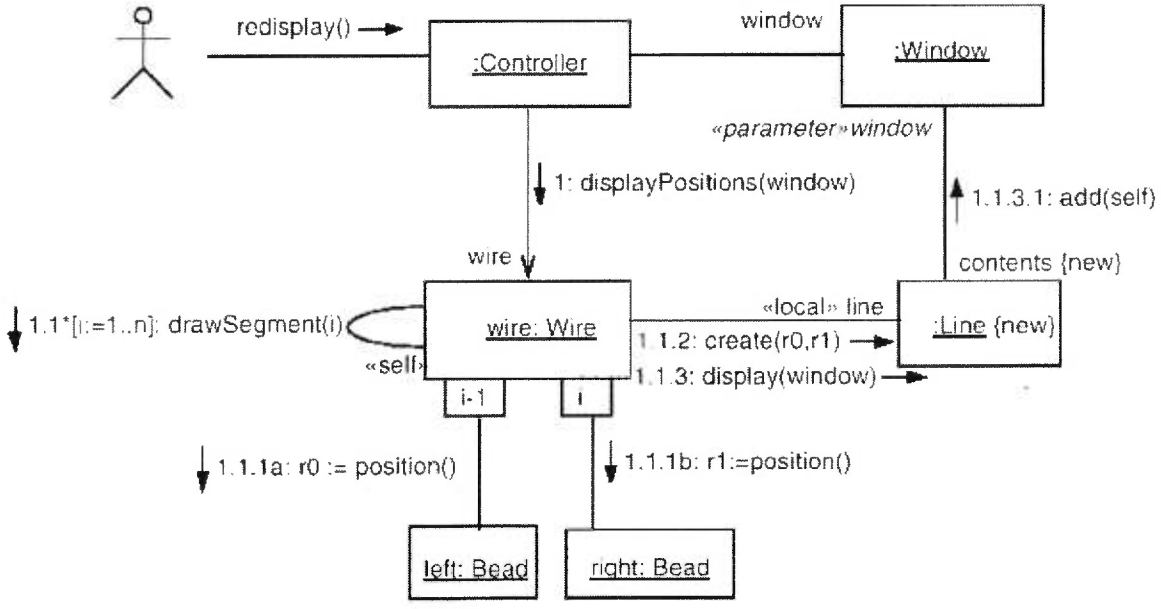

Figure 2 Diagramme de collaboration (niveau d'instance : objets, liens et stimuli) (UML v1.3)

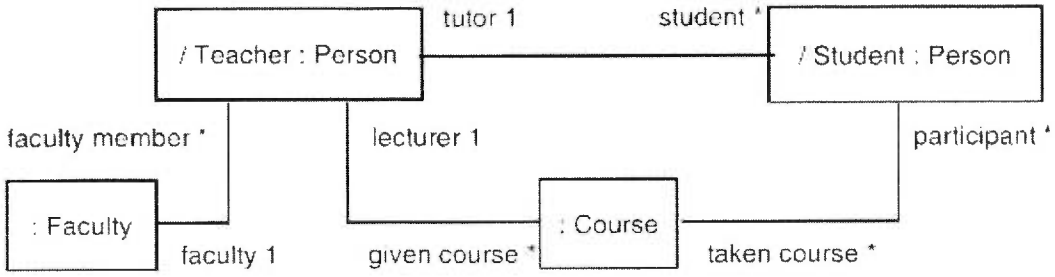

Figure 3 Diagramme de collaboration (niveau de spécification : rôle de classification, d'association et messages) (UML v1.3)

2.3.3 Modèles d'instance (exemple d'un cas général)

Niveau de représentation du système : bas

Représentation : diagrammes d'interactions (collaboration ou séquence (fig. 4))

Définition du diagramme de séquence : Le diagramme d'instance représente une interaction qui est un ensemble de messages entre les rôles de classification au sein d'une collaboration

Fonction du diagramme de séquence: représenter l'envoi de message dans le temps (généralement de haut en bas) au sein d'objets concurrents (dénotés par des boîtes avec d'épaisses bordures).

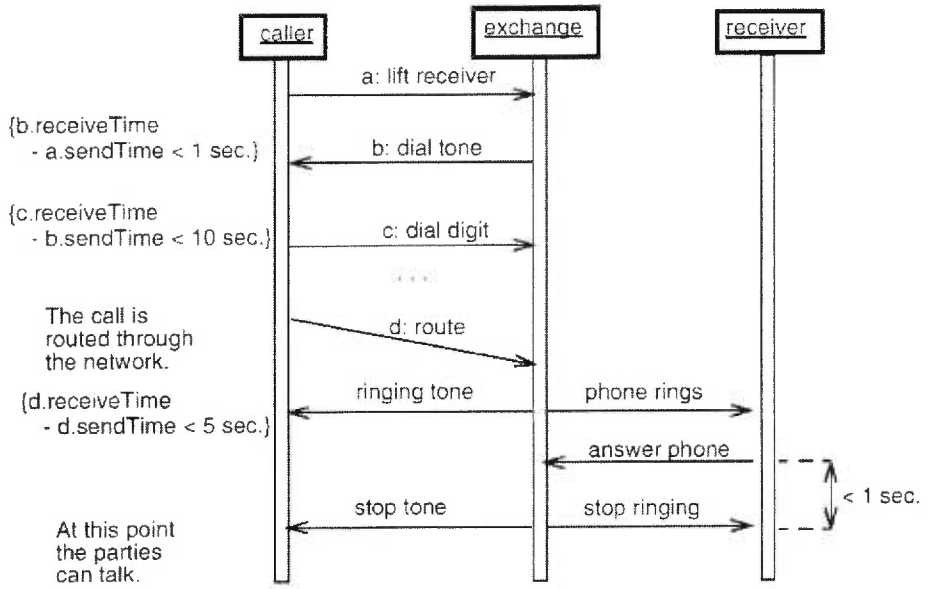

Figure 4 Diagramme de séquence (UML v1.3)

2.3.4 Diagramme d'état (exemple d'un cas général)

Défi**nition :** Un diagramme d'état (fig.. 5) représenté avec ses états, ses arcs et ses transitions permet de voir les différentes position d'un système et ces changements possibles.

Fonction : Un diagramme d'état représente le comportement dynamique d'entités répondant à la réception d'événements asynchrones.

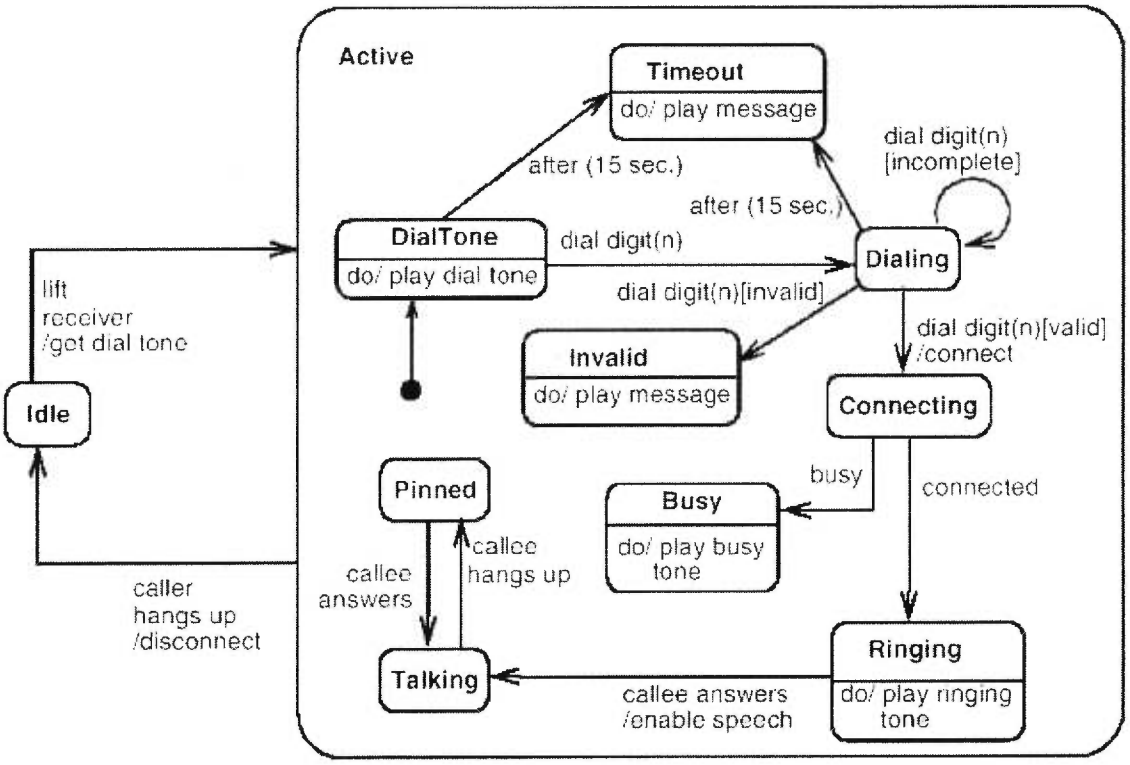

Figure 5 Diagramme d'état (UML v1.3)

2.3.5 Diagramme d'activité

Définition : Un diagramme d'activité (fig. 6) est une variation d'une machine à état dans lequel les états représentent l'exécution des actions et où les transitions sont effectuées lorsque les activités sont complétées.

Fonction : Il permet de regarder les flots engendrés par les processus internes dans les activités synchrones.

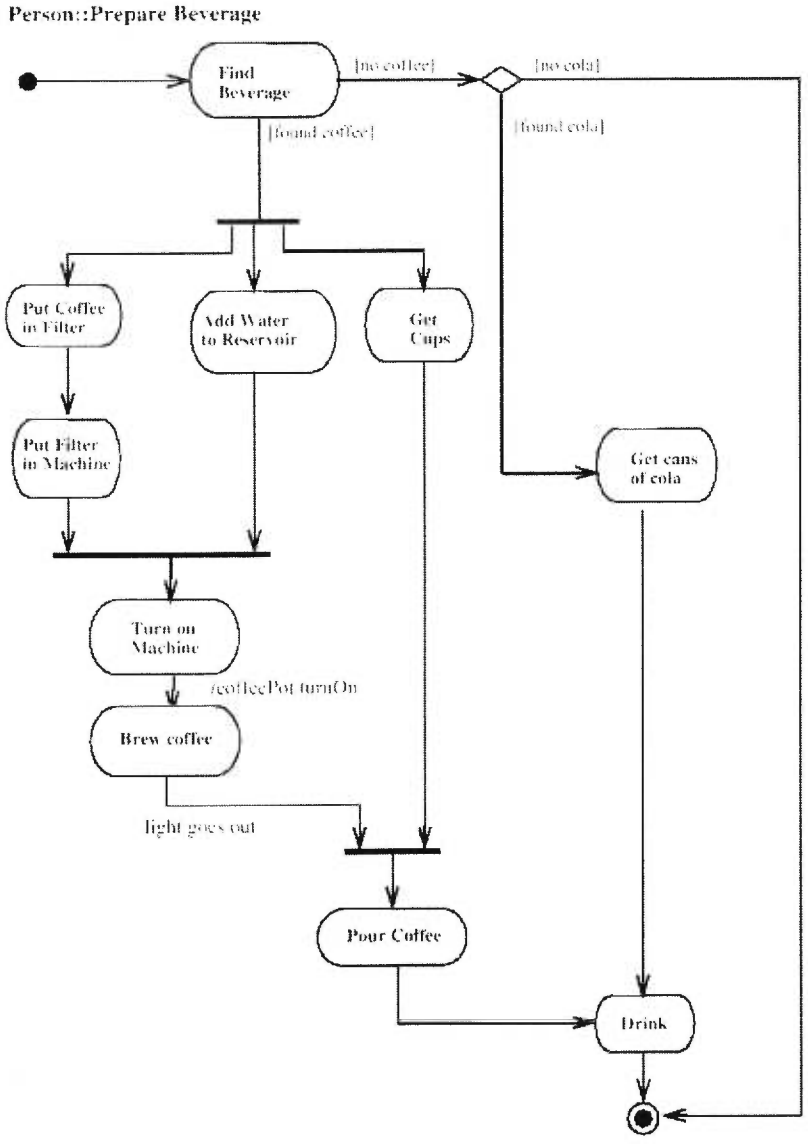

Figure 6 Diagramme d'activité (UML v1.3)

2.3.6 Diagramme de composantes

Définition : Un graphe de composantes (fig. 7) reliées les unes aux autres par des dépendances de relations. Les dépendances statiques, telles que des dépendances à la compilation entre programmes, sont indiquées en pointillés.

Fonction : Le diagramme de composantes nous indique les différentes dépendances entre les composantes du logiciel.

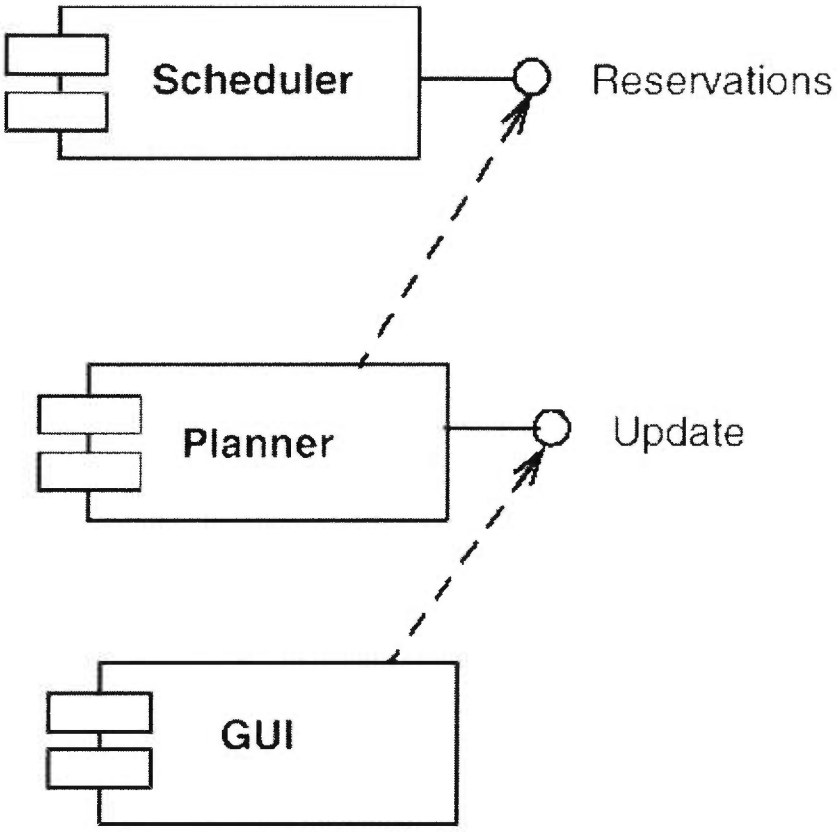

Figure 7 Diagramme de composante (UML v1.3)

2.3.7 Diagramme de déploiement

Définition : Le diagramme de déploiement (fig. 8) est un graphe dont les nœuds sont connectés par les communications d'association. Ces nœuds peuvent contenir des interfaces qui à leur tour peuvent contenir des objets. Il peut représenter la migration avec l'indication «devient» sur la relation de dépendance (n'apparaît pas sur ce diagramme).

Fonction : Le diagramme de déploiement montre la configuration à l'exécution des processeurs ainsi que des composantes logicielles, processus ou objets qui y sont exécutés. Notons que les composantes qui n'existent pas comme entités à l'exécution n'apparaissent pas sur ce diagramme mais plutôt sur les diagrammes de composantes.

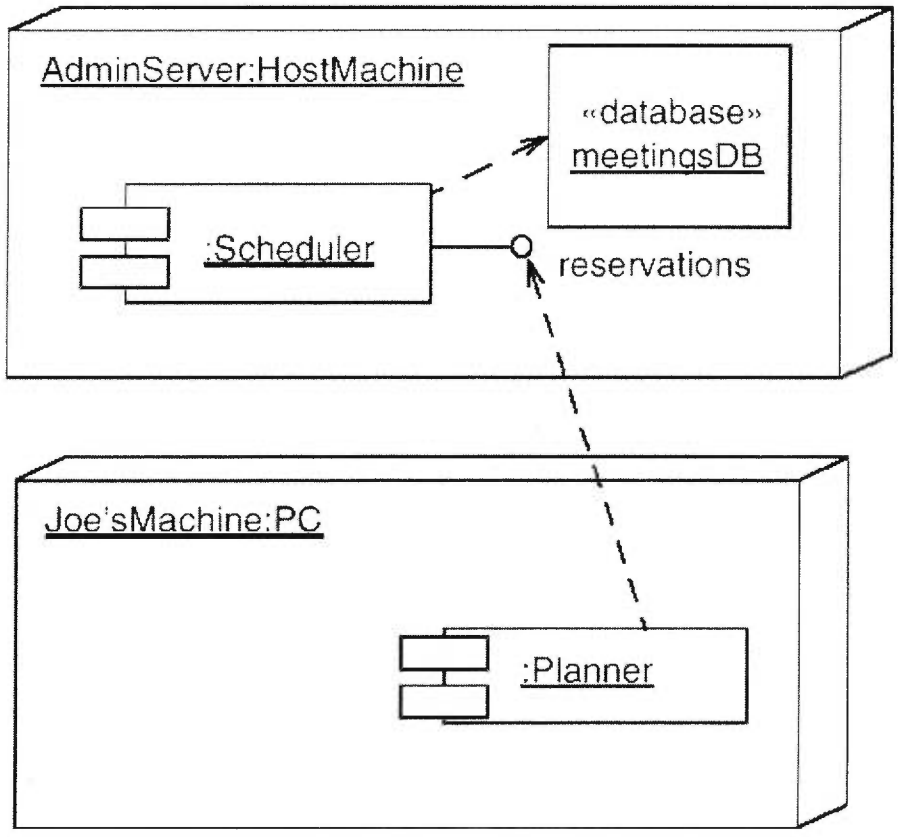

Figure 8 Diagramme de déploiement (UML v1.3)

2.4 Constatations

En utilisant les divers modèles de représentation fournis par UML, à savoir les modèles de classe, de rôle, d'instance, les diagrammes d'état, d'activité, de composantes et de déploiement, il est possible de décrire les divers aspects d'une architecture répartie. Ainsi, nous pouvons exprimer les différentes classes et leur type de relations, les rôles qu'ils occupent, les transitions possibles entre leurs divers états, l'ordre relatif des événements possibles, la dépendance entre les composantes du logiciel ainsi que la distribution des instances au niveau physique.

Les exemples simples présentés dans cette courte introduction à UML nous permettent de comprendre la représentation et les diverses fonctions du modèle de notre architecture répartie. Ainsi, nous verrons que ces représentations cadrent bien avec notre architecture logicielle coopérative et qu'elle en modélise les divers aspects pour la rendre simple, prévisible et facilement généralisable à d'autres architectures réparties du même type.

Chapitre 3 Formulation du VRPTW

3.1 Formulation

Rappelons que le problème de *VRPTW* est de trouver un ordonnancement de visites de clients et d'affecter chaque client à une route de visite d'un seul véhicule de façon à satisfaire les contraintes de capacités des véhicules, des horaires et des quantités de produit demandé par chaque client. L'objectif dans ce problème est de trouver l'ensemble des parcours qui minimisent la distance totale parcourue pour un nombre minimal de véhicules partant d'un dépôt et y retournant avant l'heure de fermeture.

Nous formulons le VRPTW selon un modèle de recherche opérationnelle de Solomon et al. (1988) dans la forme utilisée par Antes et al. (1995). Notons que le problème traité est de type livraison, mais pourrait aisément être converti en problème de type cueillette en inversant la demande. Un graphe $G := (N, A)$ représente notre problème où N représente les positions des clients et du dépôt et A *représente les arcs entre* i *et* $j \in N$. Plus spécifiquement, nous avons donc un ensemble $C = \{1, \ldots, n_c\}$ de clients qui doivent obtenir une livraison de marchandise du dépôt. L'ensemble des positions de ces clients ou nœuds est définie comme l'ensemble $N := C \cup \{0, n_c+1\}$ où 0 et n_c+1 représentent le dépôt. Une demande positive de produit d_i, un temps de service s_i et une fenêtre de temps définie comme étant l'horaire où le service est possible entre le temps e_i et le temps l_i, sont associés à chaque client i où $i = 1, \ldots, n_c$.

Une flotte de véhicules $V = \{1, \ldots, n_v\}$ est disponible au dépôt et chaque véhicule possède la même capacité (dans notre problème) Q *où* $Q \geq \max_i d_i$. Pour tous clients i et j, $i,j \in N$, nous connaissons le coût $c_{i,j}$ de voyage direct entre i et j et le temps de parcours $t_{i,j}$. Pour trouver l'ordre de visite des villes, nous définissons les variables de décisions comme suit :

$$x_{ij}^{v} = \begin{cases} 1 \text{ si le véhicule } v \in V \text{ voyage directement entre i et j} \\ 0 \text{ autrement} \end{cases}$$

Autres variables à déterminer

a_i = temps d'arrivée chez le client $i \in C$.

b_i = temps auquel le service débute chez le client $i \in C$.

b_0^v = temps auquel le véhicule v quitte le dépôt.

$b_{n_c+1}^v$ = temps auquel le véhicule v retourne au dépôt.

w_i = temps d'attente à $i \in C$.

s_i = temps de service à $i \in C$.

Constantes connues

e_i = borne inférieure sur la fenêtre de temps du client à $i \in C$.

l_i = borne supérieure sur la fenêtre de temps du client à $i \in C$.

c_{ij} = coût du déplacement de i à j $i, j \in C$.

L'attente est permise lorsqu'un véhicule arrive trop tôt chez le client j, c'est-à-dire avant e_j. Le temps auquel le service débute chez le client $j \; \varepsilon \; C$ se définit comme étant $b_j = \max\{e_j, b_i + s_i + t_{i,j}\}$ et le temps d'attente chez le client j comme étant $w_j = b_j - (b_i + s_i + t_{i,j})$.

En définissant y_i comme étant la charge maximale résiduelle du véhicule après avoir desservi le client $i \; \varepsilon \; C$ après $b_i + s_i$, il nous est possible d'écrire le modèle de VRPTW selon la formulation de Solomon et Desrosiers (1988).

3.2 Formulation

$$Min \sum_{v \in V} \left[\sum_{j \in C} x^{v}_{0,j} + \sum_{i \in N} \sum_{j \in N} c_{i,j} x^{v}_{i,j} \right] \quad \textbf{(3-1)}$$

sujet à:

$$\sum_{v \in V} \sum_{j \in N} x^{v}_{i,j} = 1 \qquad \forall i \in C \quad \textbf{(3-2)}$$

$$\sum_{j \in N} x^{v}_{i,j} - \sum_{j \in N} x^{v}_{j,i} = 0 \qquad \forall i \in C, v \in V \quad \textbf{(3-3)}$$

$$\sum_{j \in N} x^{v}_{0,j} = 1 \qquad \forall v \in V \quad \textbf{(3-4)}$$

$$\sum_{j \in N} x^{v}_{i,n+1} = 1 \qquad \forall v \in V \quad \textbf{(3-5)}$$

$$x^{v}_{i,j} = 1 \Rightarrow \quad b_i + s_i + t_{i,j} \le b_j \qquad \forall i, j \in C, v \in V \quad \textbf{(3-6)}$$

$$x^{v}_{0,j} = 1 \Rightarrow \quad b^{v}_{0} + t_{0,j} \le b_j \qquad \forall j \in C, v \in V \quad \textbf{(3-7)}$$

$$x^{v}_{i,n+1} = 1 \Rightarrow \quad b_i + s_i + t_{i,n+1} \le b^{v}_{n+1} \qquad \forall i \in C, v \in V \quad \textbf{(3-8)}$$

$$e_i \le b_i \le l_i \qquad \forall i \in C \quad \textbf{(3-9)}$$

$$e_0 \le b^{v}_{0} \le l_0 \qquad \forall v \in V \quad \textbf{(3-10)}$$

$$e_{n+1} \le b^{v}_{n+1} \le l_{n+1} \qquad \forall v \in V \quad \textbf{(3-11)}$$

$$x^{v}_{i,j} = 1 \Rightarrow \quad y_i - d_j = y_j \qquad \forall i, j \in N, v \in V \quad \textbf{(3-12)}$$

$$y_o = Q, 0 \le y_i \qquad \forall i \in C \quad \textbf{(3-13)}$$

$$x^{v}_{i,j} \in \{0,1\} \qquad \forall i, j \in N, v \in V \quad \textbf{(3-14)}$$

$$\sum_{i \in X} \sum_{j \in X} x^{v}_{i,j} \le |X| - 1 \qquad \forall i, j \in N, v \in V, \forall X \subseteq N \quad \textbf{(3-15)}$$

La fonction de coût euclidien des parcours des villes visitées

$$coût(X) = \sum_{v \in V} \sum_{i \in N} \sum_{j \in N} c_{i,j} x_{i,j}^{v} \quad \textbf{(3-16)}$$

Le nombre de véhicules utilisés

$$no\_véhicules(X) = \sum_{v \in V} \sum_{j \in C} x_{0,j}^{v} \quad \textbf{(3-17)}$$

Le temps d'utilisation réel des véhicules

$$temps\_horaire(X) = \sum_{v \in V} (b_{n_c+1}^{v} - b_{0}^{v}) \quad \textbf{(3-18)}$$

3.2.1 Détail des équations

En (3-1), nous avons la fonction objective qui représente la somme des coûts de parcours et du nombre de véhicules utilisés pour les n_v routes.

Les contraintes (3-2) assurent que chaque client est visité une et une seule fois.

Les contraintes (3-3) assurent que chaque véhicule arrive et part de chaque client visité.

Les contraintes (3-4) assurent que chaque route commençant au dépôt n'existe qu'une fois.

Les contraintes (3-5) assurent le retour au dépôt pour chaque tournée.

Les contraintes (3-6,3-11) définissent les fonctions de temps et (3.12-3.14) les contraintes de capacité et d'intégralité.

Les mesures (3-16,3-17, 3-18) permettent respectivement de, quantifier la solution selon la distance totale, d'identifier le nombre de véhicules utilisés ainsi que le temps total d'utilisation des véhicules.

3.2.2 Modification de la métrique de distance

Dans un TSP ou un VRP, la définition de distance entre deux clients qui nous vient naturellement à l'esprit est la distance que requiert le parcours direct entre ces deux clients. Dans les algorithmes utilisés pour le VRPTW, cette même métrique de distance est généralement utilisée.

Nous voulons utiliser une métrique de distance qui favorise non seulement les faibles distances planaires, mais aussi les faibles temps d'attentes w_j (pour éviter les pertes de temps), un faible écart entre a_j et l_j représentant une arrivée très tardive chez le client j à partir de i de telle sorte qu'un autre client pourrait difficilement être inséré entre i et j assurant ainsi un bon choix de l'arc (i,j).

Dans cette optique, nous proposons une utilisation graphique de la métrique de distance tridimensionnelle dynamique de Solomon (1987) où chaque client *i* est représenté en trois dimensions dans $(x,y,(b_i:b_i+s_i))$:

$$C3d_{i,j} = \begin{cases} \infty \text{ si } b_j > l_j \\ \alpha(c_{(i,j)}) + \beta(l_j - a_j) + \chi w_j \end{cases} \quad \textbf{(3-19)}$$

Cette fonction est dynamique au sens ou b_i (le début du service au client i) varie lors de la résolution. On initialise b_i à e_i.

Il nous est donc possible de modifier notre fonction objective en conservant les contraintes actuelles:

$$Min\sum_{v\in V}\left[\sum_{j\in C} x^{v}_{0,j} + \sum_{i\in N}\sum_{j\in N} C3d_{i,j} * x^{v}_{i,j}\right] \qquad \textbf{(3-20)}$$

3.2.2.1 Implication de la métrique

La matrice des distances n'est plus symétrique puisqu'on introduit les fenêtres de temps (i.e. il peut être possible d'aller de i vers j mais non l'inverse à cause des contraintes de temps)

Cette nouvelle métrique est une généralisation de l'ancienne métrique dans les cas réalisables lorsque $\beta = 0 \wedge \chi = 0$.

Dans les cas réalisables et où il n'y a pas de temps d'attente, la distance $C3d_{ij}$ est identique à c_{ij}. On pourrait aussi penser à ajouter une somme pondéré de t_{ij} à la métrique de coût, ce qui serait utile dans les cas où 1 unité de distance n'est pas parcourue en une unité de temps. Dans les problèmes de Solomon, une unité de distance est parcourue en une unité de temps (fig. 9), il n'est donc pas nécessaire de l'inclure dans la métrique pour ces problèmes mais présenterait un intérêt pour des problèmes réels.

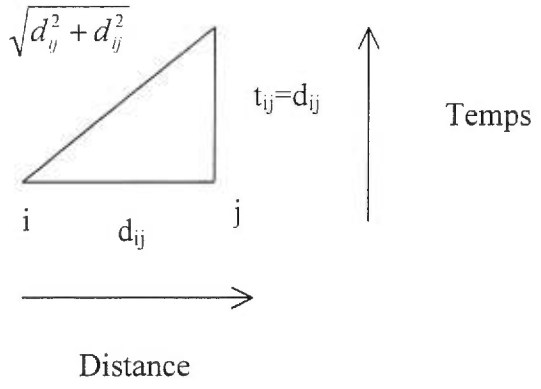

Figure 9 Coupe en espace-temps d'un parcours entre le nœud i et j

3.2.3 Visualisation 2D du VRPTW

La disponibilité sur l'Internet (www.iro.umontreal.ca/~lebouthi/vrptw) d'un applet Java permet de visualiser les problèmes et leurs solutions en 2D et 3D. La visualisation 2D (figure 10) permet d'obtenir une vue à vol d'oiseau du problème. La visualisation 3D (figure 11), pour sa part, permet un regard plus détaillé sur la notion de fenêtres de temps. Nous croyons que ce type de visualisation est pédagogique dans le sens où il permet de visualiser clairement les résultats de différentes méthodes de résolution. Mentionnons que les derniers fureteurs de Netscape™ et de Microsoft™ ne supportaient pas la dernière version de Java pour le Java3D, il est donc nécessaire de télécharger la mise-à-jour proposée automatiquement qui nécessite OpenGl.

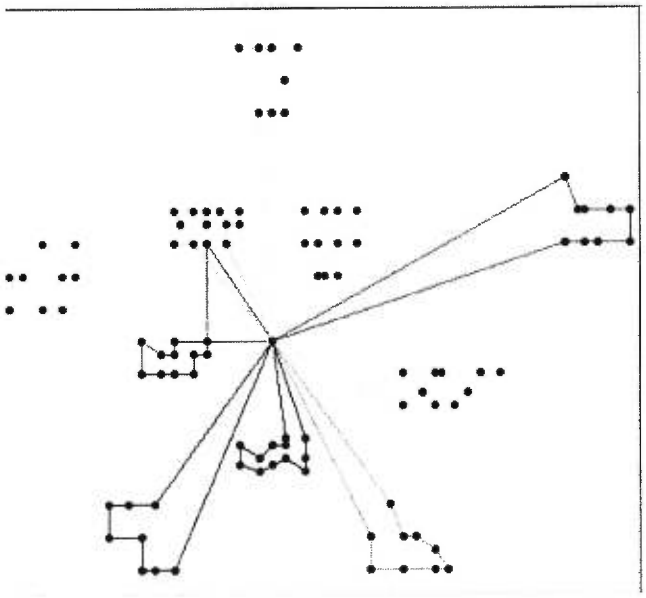

Figure 10 Visualisation 2D du VRPTW (problème C101)

Nous pouvons observer la représentation des tournées, les méthodes choisies ainsi que le résultat. Il est possible de déplacer des clients et de changer leurs contraintes pour en observer l'effet. Pour l'instant l'interface permet de voir graphiquement les tournées en 2D et en 3D (voir ci-bas) de solutions connues.

3.2.4 Visualisation 3D de deux clients

On remarque qu'il doit y avoir référence constante aux valeurs des fenêtres de temps pour justifier les choix algorithmiques lors de la visualisation d'une tournée de véhicule en deux dimensions. En effet, on remarque souvent des croisements et des détours qui sont en fait justifiés par les contraintes de temps. En représentant les tournées de véhicules en trois dimensions où la troisième dimension est le temps, il est possible de comprendre l'apparition de croisement dans les parcours de véhicules.

Grâce à la technologie Java3D, il est possible de représenter le problème en trois dimensions dans notre applet. Dans la partie inférieure du navigateur internet, nous avons un processus qui affiche en trois dimensions le problème. Les fenêtres de temps sont représentées par des cylindres sur l'axe des z placés aux coordonnées x,y des points correspondant. On peut distinguer une ligne de parcours à 45° ainsi qu'un petit arrêt qui représente respectivement le trajet parcouru en espace-temps ainsi que le temps de service (la ligne reprends un peu plus haut à la verticale, voir figure page précédente).

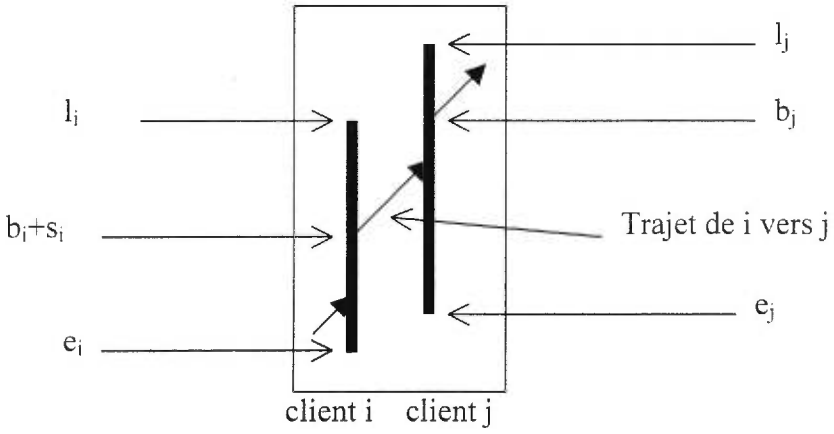

Figure 11 Visualisation 3D de deux clients

Ici chaque fenêtre de temps est représentée par un cylindre centré sur la position (x,y) et orienté sur l'axe du temps. On distingue clairement l'information graphique nécessaire à l'interprétation de ce trajet de i vers j. Dans la représentation 3D, il est possible d'effectuer des rotations et des déplacements sur l'axe des x,y et z par un glissement de la

souris ainsi que de faire des zoom avec la touche CTRL et un déplacement vertical de la souris.

3.3 Complexité

Le VRPTW est un problème NP-dur tel que décrit dans Beame et al. (1995), c'est-à-dire qu'il est peu probable qu'il existe un algorithme déterministe pouvant résoudre ce problème en temps polynomial. Trouver un algorithme polynomial pour le VRPTW permettrait de résoudre aussi l'ensemble des problèmes de cette classe et ainsi démontrer que P = NP.

On peut montrer qu'il y a la réduction VRPTW-> VRP-> TSP. Si $s_i=0,e_i=0,l_i=\infty \forall i$ et $Q=\infty$, le VRPTW se réduit à un VRP. Si v=1, le VRP se réduit à un TSP. Puisque le TSP est NP-dur selon Garey et al. (1979), le VRPTW l'est aussi.

Bien que la complexité du VRP et du VRPTW soit pour tous deux NP-dur, le VRPTW est plus difficile à résoudre empiriquement puisque les contraintes temporelles réduisent l'espace des solutions réalisables, les rendant plus difficiles à trouver dans certains cas. Ainsi, les approches approximatives en opposition aux méthodes exactes sont beaucoup plus utilisées pour résoudre ce problème. Les heuristiques et les métaheuristiques s'avèrent donc de très bons candidats comme algorithmes de résolution.

Chapitre 4 Revue de la littérature VRPTW

En regardant les différents algorithmes qui ont étés utilisés pour la résolution du VRPTW, nous remarquons une certaine évolution. Des heuristiques séquentielles aux metaheuristiques parallèles, il y a une panoplie de variantes intéressantes qui ont exploré certaines avenues de résolutions et que nous avons considéré afin de faire un choix sur les méthodes que nous utilisons. Regardons tout d'abord les méthodes de résolution du VRP pour ensuite se concentrer sur les généralisations pour le VRPTW.

4.1 VRP

Le problème de tournées de véhicules avec fenêtre de temps (*VRPTW*) est une généralisation du problème de tournées de véhicules sans fenêtre de temps (*VRP*). De nombreuses méthodes se sont attaquées au problème du VRP mais leurs adaptations au *VRPTW* ont connu plus ou moins de succès. Parmi les diverses méthodes utilisées pour la résolution du *VRP*, on peut distinguer les méthodes exactes, les méthodes hybrides, les heuristiques et les métaheuristiques.

Avec des méthodes exactes, Desrochers, Desrosiers et Solomon (1992) ont résolu à l'optimalité 21 des 56 problèmes de la série C1 de Solomon par génération de colonnes, Fisher, Jörnsten et Madsen (1991) en ont résolu 12 avec une relaxation d'un k-arbre et Kohl et Madesn (1995), Halse (1992) en ont trouvé 20 par séparation variable basée sur une relaxation Lagrangienne. Enfin, Kohl et Madsen (1995) ont résolu l'ensemble de 27 problèmes de la série C1 des problèmes de Solomon. Même s'il s'agit d'excellent résultats, il est important de mentionner qu'ils ont été obtenus sur la série C1 qui contient des clients par groupe et qui nécessite peu de branchement. Ainsi, la résolution par méthode exacte permet difficilement la résolution de plus gros problèmes ou simplement de problèmes nécessitant plus de branchement. C'est pour cette raison que les heuristiques et métaheuristiques sont souvent les méthodes de prédilection pour ces

problèmes. Nous concentrons ainsi notre revue sur ces méthodes, principalement la recherche avec tabous qui constitue l'approche la plus prometteuse.

4.1.1 Heuristiques

Nous concentrons notre revue sur les diverses catégories d'heuristiques de résolutions telles les méthodes de construction, la méthode des gains, d'insertion, les méthodes hybrides, méthode de groupe en premier et route en second, de route en premier et groupe en second, de l'algorithme de balayage, de l'algorithme de construction en pétale, de diverses heuristiques d'améliorations ainsi que des heuristiques hybrides. Notre revue sur le VRP s'inspire de l'état de l'art de Laporte (1999) qui est une source très pertinente.

4.1.1.1 Heuristiques de construction

Deux techniques principales sont utilisées pour la construction des solutions, la fusion de routes existantes par un critère de gain et l'assignation d'arcs aux routes par l'utilisation d'un coût d'insertion.

4.1.1.2 Algorithmes de gain

4.1.1.2.1 Clarke et Wright

L'algorithme des gains de Clarke et Wright (1964) est sans aucun doute, une des méthodes les plus connues pour le VRP. Dans une version parallèle de cet algorithme, où le nombre de véhicules est une variable de décision, les meilleures fusions possibles sont calculées en partant du haut de la liste. L'algorithme nécessite le calcul du gain possible entre (i,j) et de la fusion de route comme suit :

Pour calculer le gain $S_{ij}=c_{i0}+c_{0j}-c_{ij}$ pour i,j=1,…,n et i≠j, c_{ij} étant le côut de l'arc (i,j). Créer n routes (0,i,0) pour i=1,…n. Trier les gains en ordre décroissant.

Pour étendre les routes par fusion, considérer tour à tour chaque route (0,i,…,j,0) pour déterminer le premier gain s_{ki} ou s_{jl} qui peut être utilisé pour former une route réalisable, par la fusion de la route courante et d'une autre se terminant par (k,0) ou commençant par (0,l).

Ainsi l'algorithme fonctionne comme suit :
Pour un gain S_{ij} donné, trouver s'il existe deux routes, une partant de (0,j) et l'autre se terminant à (i,0) qui puissent être fusionnés pour former une route réalisable. Dans ce cas, combiner ces deux routes en effaçant (0,j) et (i,0) et introduisant (i,j).

4.1.1.3 Méthode d'insertion séquentielle

On remarque que les algorithmes d'insertion sont constitués de deux phases, la première constitue la sélection du prochain nœud à insérer et la deuxième la méthode d'insertion. L'heuristique d'insertion du plus proche voisin choisi séquentiellement et de façon répétitive l'arc le plus court qui n'a pas encore été sélectionné pour étendre le cycle courant. Plusieurs variantes incluent des critères de sélections plus subtils tels le plus grand angle formé entre les arcs. Pour sa part, l'insertion de Mole et Jameson (1976) construit une route à la fois en insérant un arc à la fois, tandis que la construction parallèle de Christofides et al. (1979) construit alternativement en séquentiel et en parallèle les différentes routes.

4.1.1.4 Algorithme groupe en premier, route en second

Probablement les plus connus, les algorithmes de type « groupe en premier, route en second » se basent sur une approche géométrique permettant de former des groupes de clients pour ensuite trouver des routes à l'intérieur de ces groupes. Une autre méthode proposée par Fisher et Jaikumar (1981) a produit d'excellents résultats en résolvant un problème d'affectation générale (GAP) par l'affectation à coût minimal des clients à deux points fictifs formant ainsi des groupes.

4.1.1.5 Algorithme route en premier, groupe en second

Golden et al. (1982) et Bodin et Berman (1979) utilisèrent ce genre d'algorithmes qui effectuent d'abord la construction de cycles ou de très grandes routes (généralement non-réalisable). Ces grands parcours sont, par la suite, divisés en plus petits parcours réalisables formant ainsi des solutions au *VRPTW*.

4.1.1.5.1 Algorithme de balayage

Adaptée aux instances euclidiennes du VRP, cette méthode popularisée par Gillet et Miller (1974) effectue un balayage circulaire autour du dépôt, pour former des groupes de nœuds rapprochés ayant des angles similaires. Ces groupes sont ensuite résolus comme différents *TSP* (Travelling Salesman Problem), chacun étant une tournée à un seul véhicule. Certains échanges peuvent ensuite êtres effectués entre les nœuds près des frontières. Notons que dans une perspective de VRPTW et dans une représentation tridimensionnelle, il serait intéressant de regarder les groupes ayant des rayons tridimensionnels similaires.

4.1.1.6 Algorithme en pétale

Cet algorithme de balayage génère plusieurs routes, appelées pétales, pour ensuite faire une sélection en résolvant un problème de partitionnement. Notons que si les routes correspondent à des secteurs continus des arcs, alors le problème peut être résolu en temps polynomial selon Ryan et al. (1993).

4.1.2 Heuristiques d'amélioration

L'application de descentes locales permet de trouver de meilleures solutions dans un voisinage immédiat si le minimum local n'a pas été atteint. Elle consiste à faire des permutations entre arcs à l'intérieur de la même tournée ou entre tournées. Les descentes locales les plus connus sont dans la famille des λ-opt (2-opt, 3-opt, n-opt).

Dans l'amélioration de tournée λ-opt (λ=2,3,4,…,n) proposées par Lin et al. (1965), il s'agit d'enlever λ arcs pour remettre les chaînes associées dans la meilleure combinaison possible. Une variation intéressante est le α-λ-opt, où seulement les α plus courts gains sont explorés. Vérifier qu'une solution est λ-optimale s'effectue en temps $O(n^{\lambda})$.

Or (1976) proposa une méthode qui consiste à déplacer des suites consécutives de 1 à 3 arcs. Vérifier qu'une solution est or-optimale s'effectue en temps $O(n^2)$. Renaud, Boctor et Laporte (1996) ont développé une version du 4-opt* qui tente de réassigner une sous-chaîne d'au plus ω sommets et une autre de deux sommets. Vérifier qu'une solution est 4-opt*-Optimale s'effectue en temps $O(\omega n^2)$.

Kinderwater et Savelsbergh (1997), Thompson et Psaraftis (1993) ont produit des méthodes d'échanges inter-routes cycliques ou non, que se soit avec échange de chaîne, croisement de chaîne ou un déplacement de chaîne.

Held et Karp fournissent une méthode permettant de calculer une borne inférieure à la valeur d'une solution optimale pour un problème donné, borne ne pouvant être inférieure à $(2/3)C^{opt}$ lorsque l'inégalité du triangle est vérifiée tel que décrit dans Held et al. (1970). Notons que l'application typique d'un λ-opt à une solution rends la solution à 5% de la borne inférieure de Held-Karp dans le cas d'une 2-opt, à 3% lors de 3-opt et à 2% dans le cas de Or-Opt.

4.1.3 Méthodes hybrides

4.1.3.1 CCAO

La combinaison des méthodes de construction et d'amélioration permet de bénéficier de l'avantage des deux méthodes. CCAO, une hybridation comprenant les méthodes **C**onvex hull, **C**heapest insertion, largest **A**ngle et **O**r-opt construit une solution contenant les nœuds de la couverture convexe dans un premier temps de façon à minimiser les coûts des détours et dans un deuxième temps insère les autres nœuds de façon à maximiser l'angle du détour créé. Une post-optimisation de type Or-opt est effectuée en fin de

procédure. On note que les nœuds dans une tournée optimale possède le même ordonnancement que dans une couverture convexe selon Flood (1956).

4.1.3.2 GENIUS

GENI (GENeralized Insertion), une hybridation d'algorithme de construction et d'insertion introduite par Gendreau, Hertz et Laporte (1992) et une post-optimisation US (Unstringing, Stringing) produit une excellente heuristique hybride nommée GENIUS. Regardons la procédure de construction et d'insertion GENI et ensuite la post-optimisation US.

4.1.3.2.1 GENI

La construction se fait à partir de trois nœuds choisis aléatoirement à partir desquels sont inséré d'autres sommets pour former un cycle Hamiltonien. L'insertion d'un nœud x se fait seulement entre deux nœuds de son voisinage $V_p(x)$ défini comme étant les p nœuds déjà insérés les plus proches de x. Le paramètre p oscillant généralement entre 3 et 20 selon la taille du problème permet de contrôler la profondeur de la recherche et ainsi de limiter la complexité de l'algorithme. Supposons une insertion du nœud x entre les nœuds $i,j \in V_p(x)$ et que $k \in V_p(i+1)$ soit sur le chemin de j à i et que $l \in V_p(j+1)$ soit sur le chemin de i à j, l'insertion peut se produire de deux façons pour chaque orientation possible d'un cycle (figure 12, 13.):

Insertion de type I

Soient $k \neq i$ et $k \neq j$. L'insertion de x dans un cycle entraîne l'élimination des arêtes $(i,i+1)$, $(j,j+1)$ et $(k,k+1)$ qui sont remplacées par (i,x), (x,j), $(i+1,k)$ et $(j+1,k+1)$

Insertion de type II

Soient $k \neq j$ et $k \neq j+1$, $l \neq$ i *et* $l \neq$ i+1. L'insertion de x dans un cycle entraîne l'élimination des arêtes $(i,i+1)$, $(l-1,l)$, $(j,j+1)$ et $(k-1,k)$ qui sont remplacées par (i,x), (x,j), $(l,j+1)$, $(k-1,l-1)$, et $(i+1,k)$

On remarque que GENI permet l'insertion entre deux nœuds non-consécutifs pour ensuite réoptimiser le chemin entre ces deux nœuds. Ainsi une insertion standard suivi d'un 3-opt est équivalent à une insertion de type I.

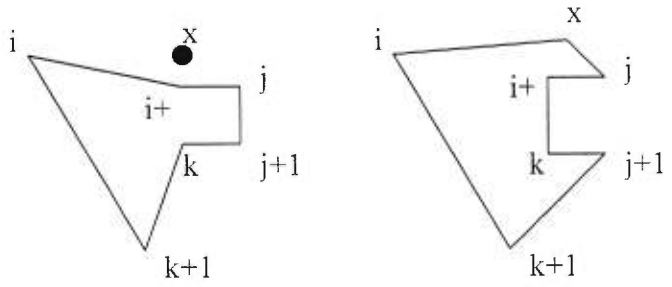

Figure 12 Insertion GENI de type I

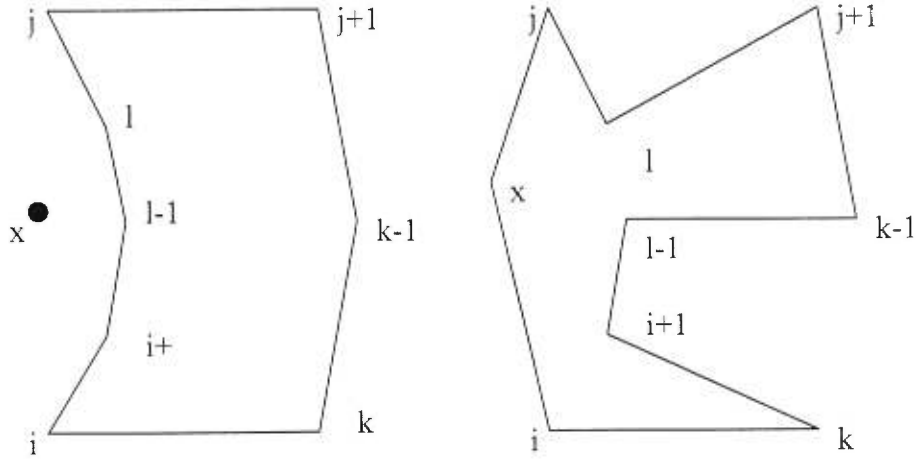

Figure 13 Insertion GENI de type II

4.1.3.2.2 US

La procédure de post-optimisation US (Unstringing-Stringing) consiste à considérer le retrait de chaque nœud pour tenter leur réinsertion dans le but de produire une tournée moins coûteuse. Notons que le retrait (unstringing) est symétrique aux insertions (stringing) et peut être effectué de deux façons :

4.1.3.2.2.1 Retrait de type I

Soient j ε V_p(i+1) et k ε V_p(i-1) un nœud sur le chemin de i+1 à j-1. Le retrait de i entraîne le retrait de (i-1,i), (i,i+1), (k,k+1) et (j,j+1) qui sont remplacées par (i-1,k), (i+1,j), (k+1,j+1).

4.1.3.2.2.2 Retrait de type II

Soient j ε V_p(i+1) et k ε V_p(i-1) un nœud sur le chemin de j+1 à i-2 et que l ε V_p(k+1) sur le chemin de j à k-1. Le retrait de i entraîne le retrait de (i-1,i),(i,i+1),(j-1,j), (l,l+1) et (k,k+1) qui sont remplacées par (i-1,k), (l+1,j-1), (i+1,j) et (l, k+1).

L'algorithme GENIUS applique les procédures d'insertion du nœud retiré *i* au meilleur cycle *k* en considérant à chaque fois les deux types d'insertions et d'orientations possibles de *k*. Rappelons qu'il est possible de réduire la complexité en utilisant la notion de p-voisinage qui considère seulement les p plus proches noeuds. Ainsi, pour un problème de taille *n*, la complexité de GENI est $O(np^4+n^2)$ puisque l'étape d'insertion et de retrait de complexité $O(p^4+n^2)$ s'effectue $n - 3$ fois.

Dans une étude de différents algorithmes de recherche locale, Johnson et McGeoch (1997), ont effectué différentes comparaisons dans des environnements similaires et confirment que GENI surpasse toutes les heuristiques de construction en produisant d'excellents résultats qui ne dépassent la borne inférieure de Held-Karp que de 9.1%, alors que l'algorithme de Christofides était à 9,8% pour les problèmes testés. Lorsque comparé aux heuristiques d'optimisation locale, il surpasse 2-opt sur les problèmes de 100 nœuds pour p ≥5 mais n'obtient pas d'aussi bons résultats que 3-opt ou 2-opt sur des problèmes de plus grande taille. Cependant, GENIUS prend toute sa force lorsqu'il est guidé par une méta-heuristique comme TABUROUTE que nous regarderons dans la prochaine section.

4.1.4 Métaheuristiques

4.1.4.1 Recherches avec tabou (TS)

Proposé par Glover (1989, 1990), les méthodes de recherches avec tabou tentent d'éviter le piège des minimums locaux en permettant une dégradation de la solution durant un certain nombre d'itérations. Une recherche avec tabous part d'une solution initiale x_1 et visite à chaque itération t le meilleur voisin x_{t+1} jusqu'à ce qu'un critère d'arrêt soit rencontré. On remarque que si $f(x)$ est le coût de la solution x à un minimum local, $f(x_{t+1})$ n'est pas inférieur que $f(x_t)$, mais qu'éventuellement il est possible d'obtenir une solution de meilleur coût si le minimum local à $f(x)$ n'est pas le minimum global. Pour éviter de boucler sur les solutions déjà explorées, les solutions récemment examinées seront dites tabou pour empêcher un retour arrière après ces dites itérations. Pour économiser l'espace mémoire, les solutions seront plutôt représentées par leurs attributs plutôt que par les tournées qu'elles représentent.

Au cours des dix dernières années, les recherches tabous ont étés appliquées au VRP par de nombreux auteurs dont les dernières implantations fournissent les meilleurs résultats de la littérature. Dans cette section, nous regarderons le travail de différents auteurs parmis lesquels on distingue les travaux de Gendreau, Hertz et Laporte (1994), de Taillard (1993), de Xu et Kelly (1996), de Rego et Roucairol (1996) et finalement de Taillard (1995) qui introduit un concept de mémoire adaptative.

4.1.4.1.1 Taburoute

L'algorithme Taburoute de Gendreau, Hertz et Laporte (1994) exploite des nouvelles particularités. GENI, un algorithme d'insertion pour le TSP, est utilisé pour explorer le voisinage défini comme étant toutes les solutions pouvant êtres atteintes en enlevant un arc et en l'insérant dans une autre route contenant les p plus proches voisins. En utilisant

des pénalités pour les solutions irréalisables en temps ou en capacité, il est possible de sortir des minima locaux.

Notons que Taburoute optimise les routes dans lesquelles un arc a été inséré avec US, une postoptimisation de décomposition-composition développée par Gendreau, Hertz et Laporte (1992).

Dans son parcours, Taburoute n'utilise pas de liste tabou mais plutôt des étiquettes tabous dont la durée varie entre 5 et 10 itérations.
Il utilise la diversification en pénalisant les arcs qui ont été déplacés fréquemment.
À l'initialisation, les solutions générées sont évaluées et seulement les plus prometteuses sont retenues.

4.1.4.2 Algorithme de Taillard

L'implémentation du tabou de Taillard (1993) utilise plutôt un voisinage de type λ-échanges utilisés par Osman (1993) et procède à des insertions simples plutôt qu'à des insertions GENI, ce qui accélère le processus. Les routes dans lesquelles il y a eu des insertions sont réoptimisées en utilisant l'algorithme de Volgenant et Jonker (1983).

La décomposition du problème s'effectue par secteur en partant du dépôt, et par région concentrique. Cette décomposition s'applique très bien aux problèmes planaires lorsque les nœuds sont distribués de façon homogène, et s'inscrit très bien dans le cadre d'une parallélisation.

4.1.4.3 Algorithme de Xu et Kelly

L'algorithme de Xu et Kelly (1996) utilise une structure de voisinage un peu plus complexe en effectuant des échanges d'arcs entre deux routes. Il permet d'une part, un repositionnement global des arcs dans d'autres routes, et d'autre part, une amélioration des routes locales.

4.1.4.4 Algorithme de Rego et Roucairol

L'utilisation de chaînes d'éjections de Rego et al. (1996) consiste au déplacement d'un arc à la position occupée par un autre arc et du déplacement de ce dernier à la position occupée par un troisième arc, et ainsi de suite dans une réaction en chaîne à ι niveau. Une production de bonnes solutions est obtenue mais sans comparaison aux meilleurs algorithmes TS.

4.1.4.5 Recherche granulaire de Toth et Vigo

L'observation de Toth et Vigo (1998) que les plus longs arcs ont une chance inférieure d'être présents dans les meilleures solutions, permet d'effectuer une recherche avec tabou avec une granularité variable en fonction de la longueur des arcs explorés. D'excellents résultats ont été obtenus en des temps records. Mentionnons que l'utilisation de méthodes simples, rapides, avec perte de qualité minimale et robustesse des paramètres, est une avenue très intéressante pour les problèmes plus gros du domaine commercial.

4.1.4.6 Algorithmes évolutifs

L'idée de base des algorithmes évolutifs est de simuler l'évolution au sein d'une population de solutions échangeant certaines informations pour produire de nouvelles solutions possiblement mieux adaptées.

Introduit par Holland (1975), le concept d'algorithme évolutif nécessite : un encodage qui permet de représenter une solution, une méthode de sélection et de croisement qui permet de combiner les solutions entre elles et finalement une méthode de mutation permettant d'intégrer la diversité à la population. Ce concept d'algorithme évolutif fut appliqué aussi au VRPTW. Nous regarderons les particularités des algorithmes évolutifs dans la section VRPTW.

4.2 VRPTW

L'ajout des fenêtres de temps rend le problème dynamique, au sens où une modification à un endroit de la tournée peut se répercuter sur les temps d'arrivée de tous les arcs subséquents, pouvant rendre la tournée irréalisable. Contrairement à la contrainte de capacité, la réalisabilité d'une tournée en temps dépend de l'ordonnancement des arcs. Regardons un peu les généralisations des concepts introduits dans la section VRP ainsi que les heuristiques et métaheuristiques qui ont été adaptées au VRPTW.

4.2.1 Construction de routes

La construction de route, une à la fois ou en parallèle revient à déterminer quel est le prochain client à insérer et dans quelle route il doit l'être. Les méthodes de construction sont des généralisations des méthodes sans fenêtre de temps sur lesquelles on effectue deux principales utilisations des fenêtres de temps. Premièrement, on peut utiliser les fenêtres de temps comme élément contraignant, c'est-à-dire que les choix des arcs à ajouter sont influencés seulement par les distances dans la mesure on l'ajout de l'arc ne rend pas la solution irréalisable. Deuxièmement, il est possible de pondérer les choix des arcs avec l'attente, la longueur des fenêtres de temps ainsi que le moment d'arrivée dans la fenêtre de temps.

Par une méthode de gains, Solomon (1987) permet d'insérer les clients dans la meilleure position possible en regard à une somme pondérée de la distance et du temps additionnel requis. Cette méthode d'insertion qui donne de très bons résultats, nommé I2 est une généralisation de l'approche de Mole et Jameson (1976) et est un excellent exemple d'utilisation des propriétés des fenêtres de temps.

4.2.1.1 Génération par villes (DENN, Double-Ended Nearest Neighbour)

Dans cette méthode, on commence par sélectionner une ville à partir de laquelle on générera le reste de la tournée. De cette ville on trouve celle qui est la plus proche et on l'ajoute à la suite. On ajoute successivement la ville qui est la plus proche de l'une des deux extrémités de la sous-tournée, jusqu'à ce que ce qu'il ne soit plus possible de rajouter ou d'insérer une ville dans cette tournée. On continue ensuite à construire les autres tournées de cette façon jusqu'à ce que chaque tournée soit complète. Cette heuristique peut nous donner autant de solutions initiales que le problème a de ville. Cet algorithme est légèrement supérieur au NN (Nearest Neighbour) où la construction s'effectue seulement dans un sens. La complexité en temps est de $O(n^2)$.

4.2.1.2 Génération par arc (MF, Multiple Fragment)

On sélectionne en premier lieu les arcs les plus courts parmi une possibilité de n^2 arcs réalisables qu'on ajoute à la tournée courante s'il préserve sa faisabilité, sinon on ajoute à une autre des tournées. Tout au long du processus, il est important de s'assurer qu'il n'y a pas introduction de cycles. Cette heuristique permet de générer une solution de bonne qualité en terme de distance totale mais peut contenir beaucoup de tournée si les fenêtres de temps sont très contraintes. Elle est dans $O(n^2)$ puisqu'il faut regarder tous les arcs.

4.2.1.3 Génération par arc hybride (SAH, Shortest Arcs Hybridation)

Une variante de la génération par arc où l'on ne choisit pas nécessairement les n meilleurs arcs en introduisant une possibilité de refuser de prendre un arc pour prendre un autre choix dans l'ordre. Il est possible de spécifier une borne sur le nombre de refus de façon à obtenir des solutions qui contiennent quand même de bons arcs. Typiquement les solutions optimales ne contiennent pas tous les n meilleurs arcs mais contiennent souvent les arcs qui sont dans les 20% plus courts, ce qui peut réduire considérablement l'espace de recherche.

4.2.1.4 Génération aléatoire

Une des meilleures façons d'obtenir de la diversité est la génération aléatoire de solution. Afin de diminuer les solutions irréalisables, on peut choisir successivement les villes qui possèdent le moins de villes voisines réalisables, c'est-à-dire qui peuvent êtres visitées en respectant les contraintes de temps et de capacités. On choisit au hasard un successeur parmi ces villes en biaisant vers les plus courts chemins. On ajoute ensuite à la tournée appropriée et on met à jour la liste des successeurs. Cette méthode permet de générer plusieurs solutions de piètre qualité mais d'une très grande diversité.

4.2.2 Amélioration de routes

Le voisinage d'une solution peut être défini par l'ensemble des solutions qui peuvent êtres atteint par une manipulation simple de cette solution initiale. Une des façons d'explorer le voisinage d'une solution donnée est de faire des échanges de positions de λ arcs inter-routes ou intra-routes jusqu'à ce que la λ-optimalité soit atteinte dans chaque tournée, c'est-à-dire que toutes les possibilité de λ-échanges ont été considérés et que la meilleure a été sélectionnée. C'est Solomon, Baker et Schaffer (1988) qui ont pour la première fois adapté ces procédures de λ-opt aux fenêtres de temps en déplaçant jusqu'à trois clients consécutifs entre deux clients consécutifs apparaissant plus tard dans une tournée. On note que les procédures de 2-opt et 3-opt sont moins performantes que Or-opt sur les VRPTW puisqu'il y a une notion de direction implicite qui est créee par les fenêtres de temps, il est donc difficile de changer l'ordre de la tournée tout en préservant les fenêtres de temps.

4.2.3 Combinaison d'heuristiques

Une construction vorace initiale peut-être suivie d'une phase d'amélioration ou bien, tel que proposé par Russell (1995) ou Cordone et Calvo (1997), une amélioration peut-être inclus dans la phase de construction. Cette dernière méthode ayant l'avantage de ne pas

faire l'optimisation à la fin de la construction et donc de pouvoir plus facilement explorer le voisinage local à différentes étapes de la construction.

4.2.4 Métaheuristiques

Comme pour le VRP, les métaheuristiques se situent aussi au cœur des méthodes prometteuses pour le VRPTW. On peut distinguer : les algorithmes évolutifs, le recuit simulé, les recherches avec tabous. Pour leur part, les algorithmes évolutifs tentent d'améliorer la solution à chaque déplacement dans le voisinage, tandis que le recuit simulé et les recherches avec tabous permettent le parcours d'un voisinage de qualité inférieure lorsqu'un minimum local est atteint. Les métaheuristiques produisent des solutions d'excellente qualité, mais se heurtent souvent à deux problèmes : elles nécessitent des temps d'exécution assez longs, et la fonction permettant le passage d'une solution réalisable à une autre solution réalisable du voisinage n'est pas toujours triviale. En fait, toute opération sur l'ordre de parcours affecte le moment d'arrivée chez le client et cause facilement des retards. Notons que ce passage peut-être relativement aisé pour les problèmes classiques simples selon Golden et al. (1997) mais avec l'ajout de contraintes additionnelles de problèmes réels, il devient un défi important. En transformant les contraintes dures (c.-à-d. fenêtres de temps) en contraintes molles avec pénalités, la réalisabilité est facilement conservée d'un voisinage à l'autre puisqu'on permet le retard dans Taillard (1993).

On observe souvent des combinaisons de métaheuristiques avec d'autres méthodes par exemple un algorithme évolutif combiné avec une heuristique vorace simple fut développé par Potvin et Bengio (1996). Cet algorithme évolutif utilisait différentes formes de voisinages tels les croisements, les recombinaisons et les mutations pour améliorer une population générée par une heuristique d'insertion.

4.2.4.1 Algorithmes évolutifs

L'algorithme évolutif ou génétique, introduit par Holland (1975) se base sur la théorie de l'évolution de Darwin. Ainsi, dans un bassin de population donné, les plus forts individus auront plus de chance de se reproduire et ainsi de transmettre leurs gênes pour produire de meilleurs descendants, augmentant la qualité globale de la population. Dans cette ligne de pensée, divers algorithmes évolutifs ont été développés par Blanton et Wainwright (1993), Thangiah et Petrovic (1998) et par Potvin et Bengio (1996). On constate alors que la résolution, par un algorithme génétique, du problème de VRPTW nécessite une définition de l'encodage qui représentera les propriétés de nos solutions, un algorithme de croisement qui extrait les meilleures propriétés de deux solutions pour idéalement produire une meilleure descendance et un algorithme de mutation qui fera évoluer certains gênes. Regardons un peu le mécanisme général d'encodage, de génération de solutions initiales, de sélection, de mutation et de croisement.

4.2.4.1.1 Encodage

Le problème de tournées de véhicules avec fenêtres de temps est en fait un problème d'ordonnancement. Une représentation classique sous forme de chaîne de bits s'avère donc mal adaptée, les solutions étant mieux représentées par des permutations des éléments à visiter. La représentation traditionnelle est la représentation par chemin ou par adjacence.

La représentation par chemin consiste à placer les villes les unes à la suite des autres, dans l'ordre de visite. Cette représentation encode principalement l'information reliée à l'ordre relatif des villes. La représentation par adjacence, où une ville i est présente à la position j de l'encodage s'il existe un arc de i à j, conserve l'information relative aux arcs utilisés dans la tournée. Potvin et Bengio (1996) favorisent la représentation par adjacence qui permet de construire plus facilement des parcelles de tournée. On note que le sens de la tournée est important à cause des fenêtres de temps.

4.2.4.1.2 Solutions initiales

La qualité de la population initiale est très importante pour la génération de meilleures solutions. La diversité de la population est très importante pour éviter la convergence prématurée vers des minimums locaux. Cette diversité de population se définit initialement et se maintient en combinant diverses stratégies de croisement et de mutation. Les méthodes de générations de solutions initiales sont généralement la génération par villes, par arc et par arc hybride telles que définies dans la section sur les constructions de routes.

4.2.4.1.3 Opérateur de sélection

Autant la qualité des solutions et l'encodage peut-être important, la sélection permet pour sa part de combiner les bonnes solutions. Il faut donc doser le choix de solutions de bonne qualité tout en conservant la diversité pour prévenir les convergences prématurées de la population vers un minimum local. On note que l'introduction de sélection aléatoire biaisée vers les plus adaptés est une notion très importante dans cette recherche de diversité.

4.2.4.1.4 Opérateur de croisement

Les opérateurs les plus connus de la littérature sont OX (Order Crossover) et ER (Edge Recombination). Il est possible de les utiliser dans des proportions différentes a l'intérieur d'une même population dans le but d'introduire de la diversité. Les opérateurs de croisement tentent de combiner les informations des parents au bénéfice de la descendance. Malheureusement, le croisement implique la division des informations de chaque parent ce qui engendre souvent des pertes de schémas généraux. Les schémas se définissent comme un ensemble de caractéristiques propres à une solution qui peuvent être une suite relative de nœuds, une présence de certains nœuds ou encore une absence de certains nœuds. Goldberg (1989) remarque que plus les schémas sont longs, plus il y a

haute probabilité de destruction des schémas lors du croisement. Il devient donc très important de choisir des opérateurs de croisement qui tentent de préserver ces schémas. Regardons deux opérateurs de croisement OX et ER qui tentent de préserver certains schémas initiaux.

4.2.4.1.4.1 OX, Order crossover

L'opérateur de croisement Order Crossover tente de préserver l'ordre relatif des villes dans le deuxième parent. Il nécessite une représentation sous forme de chemin et non d'adjacence. Il consiste à copier une sous-chaîne du premier parent dans l'enfant et ensuite de copier le reste du deuxième parent dans l'enfant. Une sous-chaîne de longueur et de position aléatoire est donc choisie du premier parent pour être copié dans l'enfant. Les positions restantes sont remplies dans l'ordre par les nœuds du deuxième parent qui ne sont pas encore présents dans l'enfant et qui ne violent pas les contraintes de capacité et de fenêtres de temps.

4.2.4.1.4.2 ER, Edge recombination crossover

En tentant de préserver le plus d'arcs possible des deux parents de façon à minimiser l'introduction d'arcs additionnels, l'opérateur de croisement ER semble offrir de meilleurs solutions selon Potvin et al. (1996). Dans une première étape, il faut construire une table de liste d'adjacence qui indique à quelles villes la ville i est adjacente dans les deux parents pour chacune des villes.

Un arc est actif pour la ville i et la ville j qui lui est adjacente si cet arc n'est pas encore présent dans l'enfant. La construction de l'enfant s'effectue avec des chemins qui conduisent aux villes avec le plus petit nombre d'arcs actifs de façon à éviter les impasses (les villes n'ayant plus d'arcs actifs). Les choix de la ville initiale, de reprise lors d'impasse ainsi que de bris d'égalité peuvent êtres changés et influencent les résultats. On note que l'arc final n'est pas représentatif des parents puisqu'on a aucun contrôle sur son choix.

4.2.4.1.5 Opérateur de réparation

Le croisement engendre souvent une dégénérescence des schémas mais aussi peut rendre les solutions irréalisables. Dans le cas du VRPTW, il est possible que certaines contraintes de temps ou de capacité ne soient plus respectées après un croisement. Il est alors nécessaire de réparer les solutions non-réalisables. Une méthode simple de réparation consiste à parcourir dans l'ordre les tournées pour enlever les arcs violant les contraintes. Les nœuds disjoints peuvent êtres réinsérés dans le graphe selon diverses procédures d'insertions. Il faut lors de la réparation conserver au maximum les propriétés de la solution initiale tout rendant cette solution réalisable.

4.2.4.1.6 Opérateur de mutation

Dans le but d'introduire de la diversité et d'explorer de nouvelles régions de l'espace de solution, il est important d'introduire une mutation dans la population. Cette mutation peut s'effectuer avant ou après le croisement. Avant ou après le croisement, la mutation s'effectue sur les parents ou après le croisement sur chaque chromosome résultant de l'enfant. Chaque mutation s'effectue avec une probabilité α typiquement de l'ordre de 1% à 5%, pourcentage qui peut varier au cours du processus selon Potvin et al. (1996).

4.2.4.1.7 Descentes locales

On remarque qu'il est important de faire des descentes locales sur l'ensemble de la population afin d'éviter des arcs croisés dans les enfants. Les descentes locales les plus connus sont du type λ-opt avec des profondeurs de recherche variées (voir section précédente). On remarque que les descentes locales améliore la qualité globale de la population et qu'il est aussi préférable de combiner des solutions sur lesquelles le même type de descente locale ont étés effectuées pour profiter pleinement des algorithmes de croisement qui sont souvent sensibles à ce genre de variations.

4.2.4.2 Recherche avec tabous

La recherche avec tabous est une métaheuristique qui a été présentée expressément avec une mémoire ou plutôt un ensemble de mémoire. Proposé par Golver (1986), la recherche avec tabous modifie localement une solution de manière itérative et garde en mémoire certaines caractéristiques de ces modifications pour prévenir un retour sur ces solutions déjà visitées, d'où le nom de liste de tabous. Cette métaheuristique s'applique à de nombreux problèmes d'optimisation combinatoire dont le *VRPTW*.

Dans une première phase, il est question de construire une solution initiale. Cette solution initiale sera modifiée par des modifications locales. Ces modifications n'améliorent pas nécessairement la solution mais tentent de diriger la recherche vers un espace de solution plus prometteur. Dans cette optique, Glover (1989,1990) proposa plusieurs stratégies permettant de diriger et d'améliorer la recherche : aspiration d'une solution interdite si elle est la meilleure connue, liste de tabous à court terme, l'usage d'une mémoire à long terme où l'on force l'utilisation d'une modification jamais utilisée durant un grand nombre d'itérations, la pénalisation des modifications proportionnelles à leur fréquence d'utilisation afin de favoriser l'exploration des espaces de solutions non encore explorés. L'utilisation de la diversification de la recherche dans les espaces de solutions non explorés et de l'intensification de recherche dans les espaces de solution prometteurs prennent de plus en plus d'importance dans les bonnes implantations de cette méthode.

On peut schématiser la recherche avec tabous comme suit :
Générer une solution initiale s_0, poser $s^* = s_0$, initialiser les mémoires tabous et iter à 0.
Répéter tant que le critère d'arrêt n'est pas satisfait :
Choisir s_{k+1} dans le voisinage de la solution s_k en vérifiant que cette solution n'est pas tabou
Si $s_{k+1} \leq s^* \Rightarrow s^* = s_{k+1}$, iter++
Mettre à jour les mémoires tabous

Les variantes de ce schéma général sont très nombreuses, on peut par exemple effectuer des recherches locales très simples ou très complexes, avoir diverses représentations pour les mémoires tabous, des phases de diversifications et d'intensification ainsi que des mémoires tabous de longueur variable.

Rochat et Taillard (1995) et Taillard et al. (1997) utilisèrent un parcours avec tabous ainsi que des méthodes d'améliorations par échange d'arcs.

Garcia, Potvin et Rousseau (1996) ont implémenté une recherche parallèle synchrone avec tabous (PTABU). Leurs recherches considèrent plusieurs voisinages et appliquent les échanges 2-opt et Or-opt en partant d'une solution générée avec l'insertion I2 de Solomon. Après un certain nombre d'itérations sans amélioration, un algorithme de réduction de routes est appliqué en tentant d'enlever les clients des routes en contenant peu pour les réinsérer dans d'autres via les échanges Or-opt. On note que la faisabilité est conservée durant les processus de recherche. Potvin, Kervahut, Garcia et Rousseau (1996), pour leur part, utilisèrent une meilleure recherche séquentielle avec tabous basée sur des échanges de types 2-opt et 3-opt successifs, ce qui améliora PTABU.

4.2.4.2.1 Recherche avec tabous pour le VRPSTW (Vehicule Routing Problem with Soft Time Windows)

Taillard et al (1997) ont introduit une nouvelle heuristique d'échange (CROSS) au sein d'une recherche avec tabous permettant la résolution des problèmes de tournées de véhicules avec fenêtres de temps molles, problème où les retards aux clients sont permis avec pénalité. Cette heuristique d'échange explore les échanges possibles de deux segments de longueur d'au plus L de routes différentes ou d'une même route tout en préservant l'ordre de visite, ce qui est très pertinent dans le cas de présence de fenêtre de temps. Les évaluations de ces échanges sont effectuées en temps constant par une approximation des retards occasionnés par ces mouvements.

Les solutions initiales sont produites par une heuristique d'insertion aléatoire qui minimise le coût d'insertion I1 tel que définit par Solomon (1987). Chacune de ces solutions est divisée en rayon d'espace disjoint qui sera affecté à un processus de recherche avec tabous différents. Les nouvelles solutions trouvés par les recherches sont placées dans une mémoire adaptative sous formes de routes. Elles peuvent être ensuite combinées par un opérateur de croisement pour produire de nouvelles solutions qui seront transmises aux recherches avec tabous. Une heuristique basée sur GENIUS permet une post-optimisation des solutions améliorant légèrement 10 solutions et produisant 3 meilleures solutions connues.

Dans une version parallèle, seize recherches avec tabous communiquent avec une mémoire adaptative dans une architecture maître-esclave pour la résolution de sous-problèmes (ensemble de routes). Ces routes sont obtenues par la décomposition des solutions initiales produites par une version de l'heuristique I2 de Solomon (1987). Ces routes sont ensuite recombinées pour produire de nouvelles solutions qui serviront aux recherches avec tabous qui à leur tour produiront de nouvelles solutions qui seront décomposées et remises dans la mémoire adaptative. Dans leur implantation, on retrouve ainsi 16 recherches avec tabous qui utilisent des paramètres différents et des solutions de

départs différentes, un répartiteur qui fourni des solutions aux recherches qui en font la demande, des processus de décomposition et d'initialisation ainsi qu'une mémoire adaptative et son gestionnaire. Cette approche ne produit aucune dégradation de la qualité de solution par rapport à la version séquentielle pour un même travail. On remarque que le nombre de processeurs semble peu affecter la qualité de la solution, ce qui est très encourageant. Par contre, la décomposition des solutions en sous-problèmes composés de tournées nécessite, pour une amélioration de la rapidité de la résolution, des problèmes où il y a beaucoup de tournées ce qui n'est pas toujours le cas. On note aussi certains temps d'attente non négligeables de la part des recherches avec tabous du fait que les solutions doivent être fréquemment reconstruites à partir de sous-problèmes.

4.2.4.2.2 Approche unifiée pour le *VRPTW*, le *MDVRPTW* et le *PVRPTW*

Cordeau, Laporte et Mercier (2000) ont introduit un algorithme de résolution avec recherches Tabous pour le PVRPTW (Planning Vehicule Problem with Time Windows) et le MDVRPTW (multiple depot Vehicle Routing Problem with Time Windows) qui sont des généralisations du VRPTW. Les solutions sont évaluées par une somme pondérée dynamiquement des temps de parcours totaux et des excès de capacité, de durée et de fenêtres de temps. Les solutions sont caractérisées par un ensemble d'attributs qui indique pour chaque client i, quel véhicule k effectue la visite. Ainsi lorsqu'un client i est retiré d'une route, il est possible d'interdire sa réinsertion dans la même route pour θ itérations par l'affectation de l'attribut tabou à (i, k). Cette recherche avec tabous qui permet l'exploration de solutions irréalisables effectue une diversification en pénalisant les solutions par un facteur proportionnel aux fréquences d'apparition de chacun de leurs clients dans la liste des tabous. Combinée à une adaptation de GENIUS, cette méthode obtient d'excellents résultats sur les problèmes de Solomon.

4.2.4.2.3 Recherche réactive avec recherches tabous

Chiang et Russell (1997) ont développé une recherche avec tabous combinant un algorithme de construction parallèle. Ils se sont inspirés des listes avec tabous réactifs de

Battiti et Tecchiolli (1994) qui fait varier dynamiquement la longueur de la liste de tabous. On constate que plus la liste est longue, plus on empêche les cycles dans la recherche, par contre lorsque la liste est trop longue, la recherche devient trop contrainte. Il devient alors intéressant de faire varier la longueur de la liste en fonction des résultats de la recherche : si aucune solution réalisable n'est trouvée durant une itération, la liste de tabous est réduite de 1 élément et dans le cas où une solution serait trouvée, cette solution est rajoutée à la liste. Si une même solution est insérée plus de *REP* fois dans la liste, elle est placée dans la liste des solutions souvent trouvées. Lorsque l'ensemble des solutions souvent trouvées est plus grand que *chaos* alors la liste des tabous augmente de *zincrease* éléments.

4.2.4.3 Mémoire adaptative de Rochat et Taillard

Le concept de mémoire adaptative unifie plusieurs concept de métaheuristiques à mémoire. Comme le mentionne l'article de Taillard et al (1998), les algorithmes génétiques, les recherches par dispersion et les recherches avec tabous utilisent tous des formes de mémoires qui possèdent les particularités suivantes : Premièrement, elles mémorisent des solutions ou des caractéristiques des solutions visitées durant le processus de recherche; deuxièmement, elles font usage d'une procédure créant une nouvelle solution à partir des informations mémorisées et troisièmement, elles sont dotées d'une recherche locale, que ce soit une méthode gloutonne, une recherche avec tabous élémentaire ou un recuit simulé.

Dans une version adaptée au *VRPTW*, la mémoire adaptative de Rochat et Taillard (1995) consiste en un bassin de routes triées selon la valeur de leur solution d'origine respective. Ainsi, ces routes peuvent être combinées différemment pour produire des nouvelles solutions complètes. Il est possible d'avoir, en parallèle, différents processus de recherche avec tabous qui extraient de la mémoire adaptative des routes de véhicules sélectionnées selon la valeur de leur solution d'origine respective. Ces routes sont combinées en solutions qui sont ensuite améliorées par les recherches avec tabous pour être retournées à la mémoire. Les solutions incomplètes produites par la combinaison de

routes sont complétées par une heuristique de construction. On arrive ainsi à combiner différemment les routes en solutions complètes, utilisant ainsi les effets des divers processus de recherche avec tabous. Une attention particulière doit être portée pour qu'un client ne se retrouve pas deux fois dans la même solution. À l'époque, cette méthode a permit d'augmenter la qualité de 14 solutions des VRPTW de littérature. Elle est encore considérée comme une des meilleures méthodes.

4.2.4.4 Coopération entre recherche avec tabous et algorithmes évolutifs

Gehring et Homberger (2000) proposèrent une approche parallèle en deux phases combinant une recherche avec tabous et un algorithme évolutif. La première phase permet de minimiser le nombre de véhicules tandis que la deuxième phase tente de réduire la distance totale. On peut résumer cette métaheuristique hybride comme suit

4.2.4.4.1 Phase I

Calculer une solution actuelle (*sact*) en utilisant une version modifiée de Clarke et Wright (1964);

Initialiser la meilleure solution: *sbest* := *sact*;

Tant que le temps limite TL1 pour la première phase n'est pas atteint faire

Tant que λ-voisins réalisables de *sact* non pas été générés faire

Sélectionner aléatoirement un des divers opérateurs de mouvement;

Générer un voisin réalisable *sneigh* de *sact* en utilisant l'opérateur de déplacement 1 fois ou plusieurs fois si le critère de stabilisation n'est pas rencontré;

Générer une solution *sneigh'* avec éventuellement un nombre réduit de véhicules en appliquant un opérateur Or-opt modifié à *sneigh;*

Évaluer *sneigh'* et l'insérer dans l'ensemble des voisins;

Sélectionner le meilleur des λ-voisins réalisables générés : *sbestneigh;*

Mettre à jour la solution actuelle *: sact :=sbestneigh;*

Si *sact* est meilleur que *sbest* alors mettre à jour la solution : *sbest* :=*sact*;

Initialiser la liste de tabous *Tlist*

4.2.4.4.2 Phase II

Tant que la limite de temps TL2 pour la deuxième phase n'est pas excédée faire

Sélectionner aléatoirement un opérateur de déplacement;

Générer $c \cdot n$ voisins de *sact* avec l'opérateur de mouvement choisi

Évaluer chaque voisin réalisable et l'insérer dans l'ensemble des voisins actuels

Sélectionner le meilleur des voisins générés qui n'est pas dans la liste des solutions tabous et qui respecte le critère d'aspiration : *sbestneigh*;

Mettre à jour la solution : *sact* := *sbestneigh*;

Mettre à jour la liste des solutions tabous *Tlist*;

Si *sact* est meilleur que *sbest*, alors mettre à jour la meilleure solution : *sbest* := *sact*;

Dans la première phase l'opérateur de mouvement est choisi parmi un ensemble d'opérateurs qui inclut l'opérateur Or-opt, 1-opt de Osman (1993) et du 2-opt* de Potvin et Rousseau (1995). L'opérateur est répété de 3 à 10 fois ou jusqu'à ce que le critère de stabilisation soit atteint, c'est-à-dire que lorsque plus d'un nombre prédéfini voisin de la solution actuelle *sact* ont été généré successivement sans améliorer la meilleure solution courante *sbest*

Dans la seconde phase, la recherche avec tabous est appliquée. La liste des tabous contient les ensembles d'arcs entre le dépôt et les clients qui ont été éliminés durant l'opérateur de mouvement menant à une meilleure solution.

La sélection du meilleur voisin se fait à partir des $c \cdot n$ meilleurs voisins de la solution courante, c étant une constante > 0 et n le nombre de nœuds. Cette sélection s'effectue en prenant la solution qui pour le plus petit nombre de routes (premier critère), possède la plus petite distance totale et dont ses arcs ne sont pas dans la liste des tabous.

4.2.4.4.3 Parallélisation de recherches avec tabous et d'algorithmes évolutifs

Un modèle maître esclave a été choisi permettant le contrôle de différents processus contenant la méthode à deux phases décrite plus haut. Chacun des processus est initialisé avec une valeur racine différente pour la génération des nombres aléatoires entraînant des points de départs différents dans l'espace de solutions. Les différents processus communiquent leurs solutions dans une pile de solutions actuelles *sact* et extraient des solutions d'une autre *sglobbest* qui contient le liste des meilleures solutions globales dans l'ordre de leur apparition. La migration d'une solution s'effectue lorsque la solution de la pile *sact* est meilleure que *sglobbest*. De plus dans les phases de diversification, si la solution actuelle est de moins bonne qualité que la meilleure solution, la recherche repart de la meilleure solution. Cette implantation utilise une version faible de coopération définie par Enslow (1978) puisqu'elle possède les quatre caractéristiques suivantes : exécution concurrente de processus autonomes, coopération de processus par échange de solutions, processus de communication hiérarchique par l'utilisation d'un modèle maître-esclave et l'utilisation de mémoire partagée pour les communications.

4.2.4.5 Programmation par contraintes

La programmation par contraintes (PLC) permet d'ajouter des contraintes additionnelles à un modèle existant sans modifier la structure globale de la méthode en effectuant une séparation entre le modèle et la recherche. La programmation par contraintes coupe les branches d'explorations irréalisables ou à coût supérieur par propagation des contraintes inhérentes aux problèmes. La programmation par contraintes fonctionne bien lorsque les contraintes sont très restrictives mais requiert généralement des temps de calculs prohibitifs. Lorsque combinées à des méthodes de recherche à voisinage variable, la PLC peut faire une exploration complète des régions locales prometteuses. Rousseau, Pesant et Gendreau (2000) ont proposé une méthode hybride combinant des chaînes d'éjections simplifiées, une recherche à voisinage variable et des post-optimisations GENIUS. De très bons résultats et 12 nouveaux résultats ont étés obtenus.

4.3 Principales extensions au VRPTW

Il y a de nombreuses extensions possibles au VRPTW pour refléter les conditions réelles. Pour n'en nommer que quelques-unes : VRPBTW (backhaul : où la cueillette et la livraison se fait au sein d'un même problème), VRPBMTW (backhaul, multiple time-windows), flotte hétérogène, multicommodité, multiple dépôt, conditions initiales, fenêtres de temps « molles », coûts dépendant de la charge, du temps et des commodités, contraintes des chauffeurs pour n'en mentionner que quelques-unes. Nous avons choisi de nous concentrer sur le problème de base du VRPTW qui modélise les fenêtres de temps et les contraintes de capacités des véhicules. Le VRPTW peut être certainement étendu aux diverses facettes de la réalité mais il est important de pouvoir comparer les résultats avec la littérature avant de faire ces généralisations. Dans cette optique nous avons choisi le VRPTW qui est d'une magnitude supérieure au VRP en terme de difficulté de résolution et sur lequel plusieurs travaux de recherches ont déjà été publiés.

4.4 Choix des algorithmes

Nous remarquons que les meilleurs résultats de littérature ne sont pas produits par un seul algorithme mais par un sous-ensemble d'algorithmes. Il n'y a donc pas d'algorithme universel et il semble difficile a priori de savoir quel type d'algorithme fonctionnera bien sur une instance particulière du problème. La résolution coopérative est certainement une solution dans la mesure où elle combine différents algorithmes prometteurs. Nous avons ainsi choisi de combiner quelques algorithmes prometteurs parmi les recherches tabous, les algorithmes évolutifs ainsi que les heuristiques de construction et d'optimisations locales. Les choix se sont portés sur une adaptation avec fenêtre de temps de Taburoute utilisant GENIUS, des algorithmes génétiques combinant OX et ER ainsi que les optimisations locales de types λ-opt. Il sera question de comparer nos résultats avec les méthodes utilisant les recherches avec tabous, les méthodes utilisant du parallélisme ou des algorithmes évolutifs.

Chapitre 5 Coopération entre algorithmes avec recherches tabous et algorithmes évolutifs

5.1 Motivations

L'utilisation du parallélisme nous permet d'être en avance de plusieurs année sur la puissance de calcul d'une seule machine. Crainic et al., (1997) ont démontré que la parallélisation coopérative permet d'obtenir des solutions de qualité équivalentes aux algorithmes séquentiels pour des efforts de calcul similaires mais dans des temps réels de calculs réduits. Il permet aussi une plus grande robustesse par rapport à la qualité des solutions produites lorsque les différents processus utilisent des méthodes de résolutions différentes, des paramètres différents et que les solutions de départ sont différentes.

Dans une architecture coopérative, l'espace des solutions est exploré simultanément par plusieurs algorithmes qui doivent éviter les recoupements et qui communiquent leurs solutions par une mémoire centrale. Cette forme de parallélisme où il y a coopération via une mémoire centrale possède certains avantages : elle permet de combiner diverses stratégies indépendantes et d'en observer leurs effets combinés sans interactions directes, il n'est donc pas nécessaire d'interrompre les processus pour effectuer les communications puisqu'il n'y a pas d'intercommunication. Avant de regarder la modélisation par UML de notre architecture coopérative, nous ferons quelques rappels généraux sur les conditions nécessaires à la paralélisation, sur la complexité ainsi que les types de parallélisation.

5.2 Rappels théoriques de parallélisme

5.2.1 Conditions de Bernstein

Les conditions de Bernstein (1966) stipulent que nous avons des processus P et Q en parallèle ou en ordre séquentiel dépendant seulement s'il n'existe :
Aucun chevauchement entre les entrées de P et les sorties de Q et vice-versa
Aucun chevauchement entre les sorties de P, les sorties de Q et les entrées d 'autres tâches.

Dans notre architecture, il n'y a que des algorithmes dont les interactions s'effectuent par la mémoire centrale ce qui est beaucoup plus facile à gérer que des inter-communications. Ainsi, la dépendance n'est pas considérée au sens où l'ordre d'exécution de nos processus n'est pas nécessaire. Il n'y a donc pas de synchronisation qui doit s'effectuer puisque nous n'avons pas de dépendances entre les processus. Ce mode de communication asynchrone facilite grandement l'implantation et n'interrompt pas ou ne fait pas attendre les processus lors de communications.

5.2.2 Complexité parallèle

Au même titre que nous avons la NP-Complétude de Cook (1972) qui représente la classe des problèmes non-déterministes polynomiaux, nous avons la notion de NC-Complétude pour la complexité parallèle introduite par Beame et al. (1995). Nous rappelons que la Classe P (déterministe) est l 'ensemble de tous les langages L qui sont décidables en temps séquentiel $n^{O(1)}$ et la classe NC est l 'ensemble de tous les langages L qui sont décidables en temps parallèle $(\log n)^{O(1)}$ et processeur $n^{O(1)}$. Il y a donc une analogie entre le temps séquentiel polynomial et l'espace polylogarithmique qui peut être représenté par une machine de Turing temps $n^{O(1)}$ simulé en espace $(\log n)^{O(1)}$

Beame et al. (1995) fournissent certaines pistes nous portant à croire que P ≠ NC du fait que certains problèmes héritent du séquentiel faisable en temps et processeurs $n^{O(1)}$) et que d'autres problèmes sont hautement parallélisables en temps $(\log n)^{O(1)}$ et processeurs $n^{O(1)}$. Ainsi, la complexité s'applique aussi au parallélisme et il y a avantage réel à utiliser le parallélisme lorsque les problèmes sont hautement parallélisables.

5.2.3 Finesse du grain

La finesse du parallélisme peut se diviser soit en gros grain lorsqu'il y a répartition des tâches en plusieurs algorithmes travaillant sur les mêmes données (parallélisation de données), soit en grain fin lorsque les algorithmes sont décomposés (parallélisation fonctionnelle). Le gros grain nécessite beaucoup moins de communication et se gère plus facilement. Dans la parallélisation des méthodes de résolutions pour des problèmes combinatoires, il y a généralement deux méthodes utilisées. La première méthode à grain fin consiste à diviser la partie d'exploration et d'évaluation du voisinage pour la recherche avec tabous ce qui nécessite beaucoup de communication et un très fort synchronisme. De nombreux exemples de parallélisation à grain fin ont été proposés pour différents problèmes par Crainic et al., Garcia et al. (1995) ainsi que par Taillard (1990, 1991, 1994). La deuxième méthode à gros grain utilise plutôt diverses recherches avec tabous au lieu d'une seule. Ces différentes recherches tabous utiliseront différents paramètres et diverses solutions de départ pour tenter d'explorer, par des chemins différents, l'espace des solutions. Crainic et al. (1993) proposèrent une méthode de résolution pour la conception de réseaux utilisant divers processus ayant des paramètres différents et points de départs différents. Taillard (1993) a pour sa part, proposé une méthode de résolution pour le *VRP* et par la suite pour le *VRPTW* permettant de décomposer le problème initial en sous-problème, résolvant chacun des sous-problèmes sur des processus différents comportant des paramètres et des points de départs aussi différents.

5.2.4 Type de parallélisation

Il existe plusieurs type de parallélisme : distribuée, parallèle, répartie et coopératif. Les algorithmes distribués ou concurrent affectent les tâches à divers processeurs. Il est ainsi possible d'utiliser les stations inactives (la nuit par exemple) pour combiner leur puissance de calcul. Les algorithmes parallèles effectuent une division des tâches sur différents clients qui sont contrôlés par un serveur. Les algorithmes répartis possèdent les mêmes méthodes d'exécution et n'importe quel processeur peut initier une opération, on peut ainsi avoir une tolérance aux défaillances. Enfin, les algorithmes coopératifs tels que définis par Enslow (1978) possèdent les quatre caractéristiques suivantes : exécution concurrente de processus autonomes (gros grains), coopération de processus par échange de solutions, processus de communication hiérarchique par l'utilisation d'un modèle serveur-client et l'utilisation de mémoire partagée pour les communications.

Les algorithmes coopératifs sont un raffinement des algorithmes parallèles en fournissant une indépendance aux différents processus. Dans notre cas, nous avons choisi la coopération pour sa simplicité de parallélisation ainsi que la possibilité d'ajouter différents algorithmes de résolution séquentiels pour bénéficier des effets individuels et combinés de ceux-ci.

5.2.5 Méthode de parallélisation

Nous retrouvons trois types de méthodologies de parallélisation soit la parallélisation applicative, intégrée et réflexive tels que défini par UML (2000).

5.2.5.1 Parallélisation applicative

Utilisation du concept orienté objet et structuré sous forme de librairie qui permet l'accès aux primitives de bas niveau du parallélisme. Il est nécessaire de spécifier manuellement le contrôle des événements.

5.2.5.2 Parallélisation intégrée

Utilisation d'un langage de haut-niveau sous forme de bibliothèques de contrôle fusionnant les objets et les activités, les messages et les invocations à distance ainsi que les transmissions de message et les transactions.

5.2.5.3 Parallélisation réflexive

Avec une parallélisation réflexive, le comportement des processus s'adapte dynamiquement au contexte d 'exécution. Par exemple, un processus pourra changer de stratégie si le travail produit n'est pas aussi bon que celui produit par les autres. La coopération est tout à fait compatible avec une paralléllisation réflexive puisque l'interaction entre les processus s'effectue dynamiquement via une mémoire centrale et que le contenu de cette mémoire peut influencer les décisions prises par les processus. Notre architecture représente aussi une parallélisation réflexive.

5.3 Mémoire centrale et métaheuristiques

Nous avons retenu le concept général de *mémoire centrale* de Crainic (1997) qui agit comme une population de solutions permettant le partage des solutions complètes entre les algorithmes choisis: algorithmes de constructions et d'optimisations locales, algorithmes évolutifs et recherches tabous. Nous effectuons ainsi une parallélisation coopérative. Cette coopération entre différents algorithmes ne s'effectue pas directement mais via une mémoire centrale où chacun des algorithmes peut envoyer ou recevoir des solutions. UML nous permet de représenter notre architecture sous les divers aspects nécessaires à la compréhension des mécanismes inhérents au développement définis par Jézéquel et al. (1998). Ainsi, nous présentons notre architecture logicielle à l'aide des concepts de UML introduits au chapitre 2, à savoir les diagrammes de collaboration, de séquences, d'état, de composantes et de déploiement.

5.3.1.1 Diagramme de collaboration encapsulée

L'ajout à UML des stéréotypes de capsule, port et connecteur permettent de modéliser les systèmes temps réel.

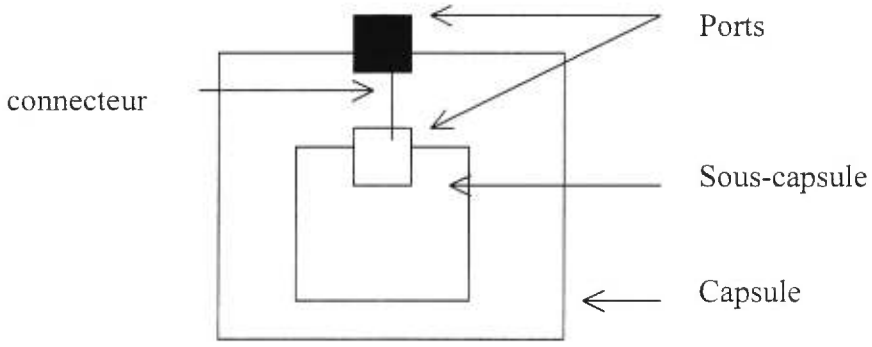

Figure 14 Diagramme de collaboration encapsulée

Ports : Les ports représentent la partie physique et fournissent un lien vers l'extérieur par un protocole donné.
Capsules : Entité indépendante interagissant avec l'entourage via ses ports
Connecteurs : Identifie les liens de communications entres capsules.

Nous utilisons le concept de capsules qui sont des composantes (objets) autonomes, interactives et encapsulées par une interface de communication qui permet la gestion de messages asynchrones. Chaque capsule possède sa localisation conceptuelle en espace-temps, son adresse de courrier et son comportement propre. Cette notion de capsule est cohérente au parallélisme et est représentée dans notre architecture logicielle par les méthodes de résolution indépendantes que nous utilisons. Ces méthodes indépendantes de résolution prennent la forme de capsules lorsque nous leur ajoutons les protocoles de communication avec la mémoire centrale contenant la population de solutions.

5.3.2 Collaboration entre algorithmes

On peut distinguer dans la partie du haut des diagrammes de collaborations (figures 15 et 16) la mémoire centrale et dans la partie du bas les différents algorithmes impliqués. Ces algorithmes de résolution produisent simultanément des solutions qui sont transmises à la mémoire centrale. La mémoire centrale dispose d'un mécanisme de classement qui lui permet de fournir adéquatement à ces algorithmes des solutions de départ ou des solutions nécessaires à la poursuite de leur exécution. Il est ainsi possible qu'une solution soit améliorée par un des algorithmes, retournée à la mémoire centrale et par la suite utilisée par un autre algorithme et ainsi de suite.

Ce mécanisme de transfert du résultat d'une exploration permet une coopération entre les divers algorithmes indépendants.

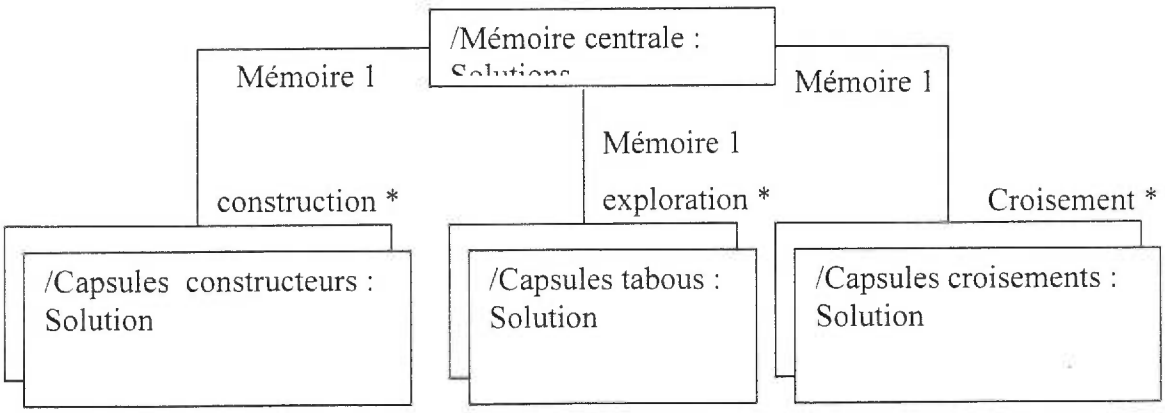

Figure 15 Diagramme de collaboration (niveau de spécification)

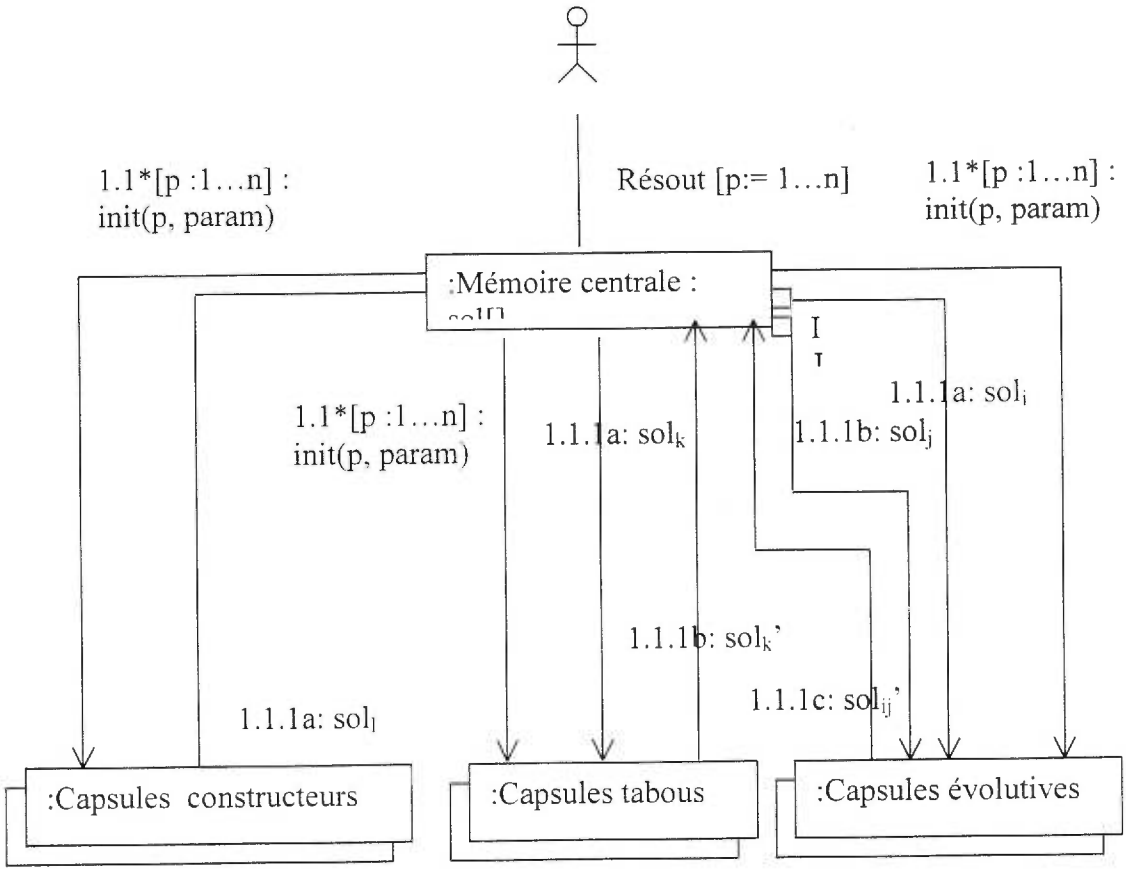

Figure 16 Diagramme de collaboration (niveau d'instance)

p : problème de n villes, coordonnées des villes, demandes et contraintes de temps;

sol : solution de n villes, ordre du parcours des villes;

param : paramètres d'initialisation

Les numéros des phases indiquent l'ordre relatif d'exécution des tâches. Les phases portant les mêmes numéros peuvent s'exécuter simultanément.

La phase 1.1 est la phase d'initialisation pour chaque processus.

Les phases 1.1.1 effectuent le transfert de problèmes et de solution. Notons que les capsule évolutives reçoivent leurs deux solutions indépendamment ce qui pourrait permettre une provenance de sources différentes (i.e., un autre mémoire centrale).

On constate sur le diagramme de collaboration (figure 11) que la communication est bidirectionnelle entre la mémoire centrale pour l'envoi de paramètres d'initialisation, de

solutions de départ ou de solutions produites. L'usager initie la mémoire centrale qui lui retournera les résultats finaux.

On observe clairement une collaboration indépendante des différents types de capsules (figure 15).

5.3.3 Séquence d'exécution

En regardant le diagramme de séquence (figure 17), nous pouvons observer le déroulement des différentes phases de l'algorithme coopératif. Dans l'ordre, les solutions entières générées par les algorithmes constructifs sont fournies à la *mémoire centrale* une fois les optimisations locales complétées, ce qui permet l'initialisation de la population initiale. À partir de ce même bassin de population de solution, les algorithmes évolutifs effectuent des croisements pour générer de nouvelles solutions et les *capsules* de recherches avec tabous qui effectuent des optimisations basées sur Taburoute de Gendreau et al. (1997). Ce partage de solutions est bidirectionnel entre la *mémoire centrale* et les *capsules* dans le sens où les capsules utilisent des solutions de la mémoire centrale et lui retournent des solutions lorsqu'il y a amélioration. Ainsi, une solution peut parcourir les différentes capsules en retournant à la mémoire centrale à chaque amélioration. On utilise alors les effets combinés des *capsules* impliquées. On observe l'envoi des messages au sein des objets concurrents : les différentes capsules et la *mémoire centrale*. Cette séquence est répétée jusqu'à ce qu'une certaine limite de temps soit atteinte.

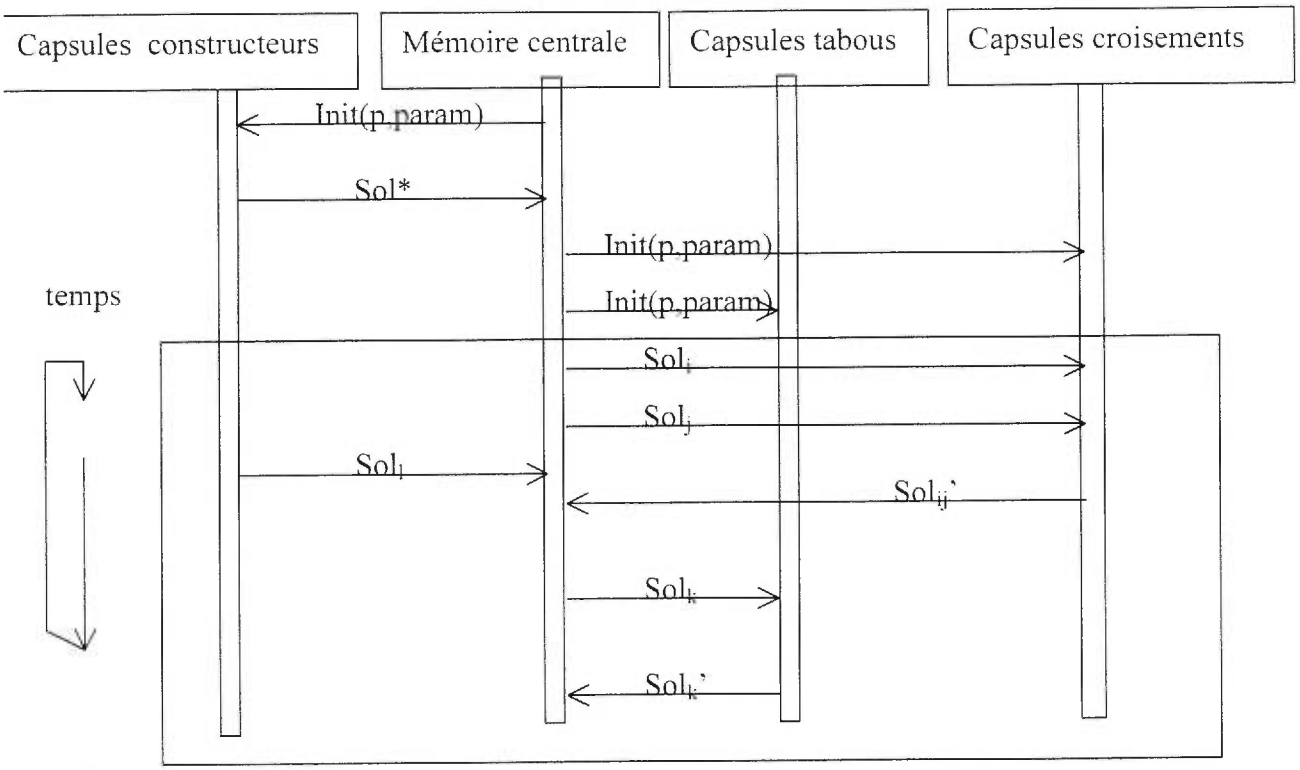

Figure 17 Diagramme de séquence

5.4 Méthodes de résolutions : algorithmes avec recherches tabous et algorithmes évolutifs

Nous détaillons ici les méthodes que nous avons combinées à savoir, les algorithmes avec recherches tabous et les algorithmes évolutifs pour leurs efficacités respectives et les espaces de solutions différents qu'ils explorent. Le succès d'un algorithme évolutif dépend de la diversité de la population et de la qualité de celle-ci. Ainsi, il est possible d'utiliser les recherches tabous pour alimenter la population en solution de bonne qualité et en diversité, prévenant ainsi une convergence prématurée de la population vers un minimum local. Ces méthodes indépendantes que sont les algorithmes évolutifs et les recherches tabous pourront ainsi combiner leurs efforts via une *mémoire centrale* qui fait office de population de solutions.

5.5 Architecture logicielle

Notre architecture coopérative est composée de capsules qui coopèrent via une mémoire centrale. Regardons les détails de la mémoire centrale ainsi que des capsules impliquées à savoir des algorithmes de construction, deux algorithmes évolutifs (OX et ER) et un Taburoute avec deux ensemble de paramètres et des solutions de départs différentes.

5.5.1 Mémoire centrale

La mémoire centrale constitue une population de solutions entières et réalisables qui sera alimentée par les différentes capsules et qui permet l'échange de solutions entre celles-ci Les solutions y sont classées selon leur coût selon un mécanisme de classement basé sur une somme pondérée du temps total de parcours, de la distance, de l'attente, du nombre de véhicules et du temps restant chez les clients avant la fin de la fenêtre de temps. Nous regardons ce mécanisme de classement et la population un peu plus en détails.

5.5.1.1 Populations de solutions de la mémoire centrale

Les solutions provenant des capsules sont communiquées à la mémoire centrale qui forme une population de solutions. Cette même population sert aussi de population pour las *capsules évolutives* et permet de fournir des solutions de départs et de diversifications aux *capsules* Taburoutes. La population est divisée en deux parties : une partie de solutions en entraînement qui n'ont pas subi encore de post-optimisations et une partie de population composée de solutions adultes, qui ont subi des post-optimisations. Un mécanisme s'assure que les solutions en entraînement subiront les post-optimisations et pourront être utilisées par les capsules comme solutions adultes.

5.5.1.2 Fonction de coût du classement dans la *mémoire centrale*

Notons que seulement les solutions réalisables sont retenues et que les solutions ne sont représentées qu'une seule fois dans le mémoire centrale. La fonction de coût choisie pour la population est une somme pondérée du temps total de parcours, de la distance, de l'attente, du nombre de véhicules et du temps restant chez les clients avant la fin de la fenêtre de temps:

$C(p) = W_1$*Temps_total + W_2*Distance_totale + W_3*Attente_totale + W4*Nombre_véchicules + W_5*Temps_restant_clients

Équation 1 Fonction de coût d'évaluation pour l'algorithme évolutif

Plus cette valeur est petite pour une solution, meilleure la solution se trouve en terme de qualité globale.

Nous définissons :

Temps_total : La différence de temps entre le départ du premier véhicule et l'arrivé du dernier véhicule

Distance_totale : La distance totale parcourue par tous les véhicules

Attente_totale : Le temps total d'attente par tous les chauffeurs

Nombre_véhicules : Le nombre total de véhicules requis

Temps_restant_clients : $\sum_{i} li - ai$

En tenant compte de la notion de contrainte disponible (attente et capacité), il est possible de faire ainsi des choix plus éclairés sur la combinaison de solutions en sachant qu'on combine deux solutions très contraintes ou non. Cette fonction de coût, qui est une version de C3D pour une solution (voir chapitre 3), est utilisée par les capsules évolutives lors de la sélection et par les capsules Taburoutes lors de la phase de diversification.

5.5.1.3 Communications asynchrones

Les communications entre la *mémoire centrale* et les capsules sont asynchrones et bidirectionnelles et il n'y a pas de communication entre capsules. Dans cette configuration, nous nous retrouvons avec des mémoires locales (pour chaque capsule) et une mémoire centrale qui récupère les solutions de chaque capsule. Chaque solution récupérée est optimisée avec λ-opt (2-opt, 3-opt) et Or-opt avant d'être classé selon la fonction de coût (voir figure 18).

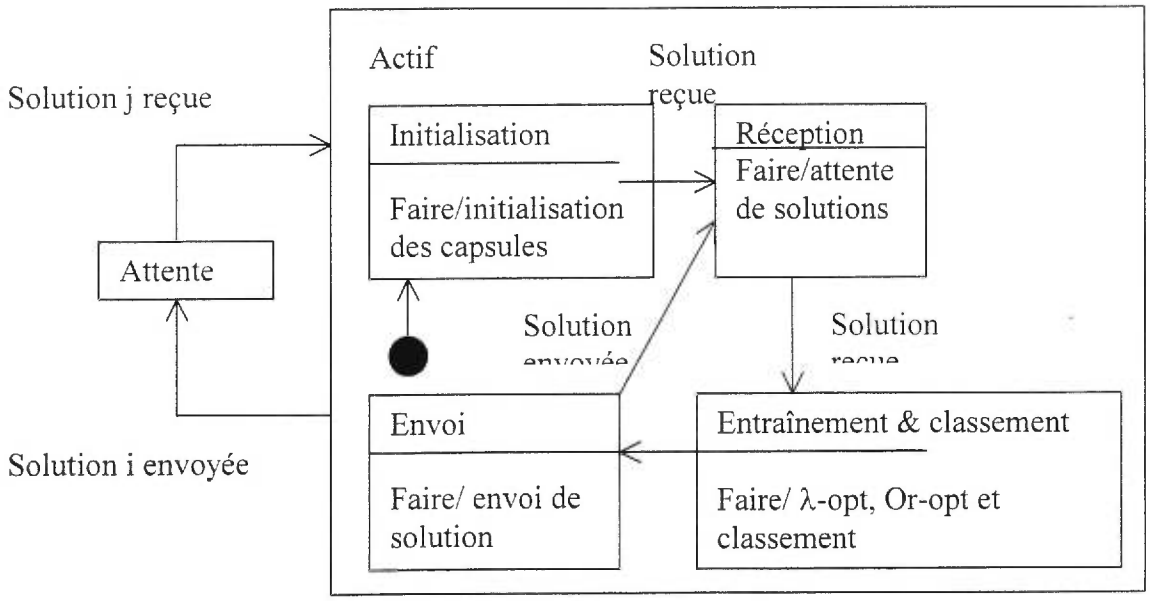

Figure 18 Diagramme d'état de l'entité mémoire centrale

5.5.2 Description de l'algorithme de construction vorace C3D

Une des solutions produites au sein de notre capsule constructive est produite à l'aide d'un algorithme qui favorise la sélection d'arcs moins contraints dont les nœuds ont une moyenne de successeurs fortement contraints. On cherche ainsi à relier des nœuds qui sont difficiles à rejoindre par d'autres nœuds du graphe.

1. Calculer la matrice des distances $C3D_{i,j}$ (voir chapitre 3)
2. Calculer les médianes des colonnes et des rangées de la C3D (plus ces médianes sont élevées, moins les arcs reliés à un certain nœud sont contraints.)

La somme des médianes des rangées et des colonnes nous donne un indice sur le niveau de contrainte associé à un problème donné. On introduit ainsi une métrique de contraintes associées à chaque problème.

3. En partant du nœud le plus contraint non sélectionné, trouver un arc réalisable et libre (où aucun nœud n'est associé à deux arcs) qui possède la plus petite C3D mais dont la médiane d'un de ses nœuds est la plus élevée.
4. Si un arc est trouvé, l'ajouter à la tournée courante et retourner à 2; S'il n'y a pas d'arc réalisable libre, insérer dans une autre tournée ou créer une autre tournée et retourner à 3.

5.5.3 Capsule constructive

Nos capsules évolutives et Taburoute utilisent le bassin de population de la *mémoire centrale* qui est initialisée par la capsule constructrice. La capsule constructrice comprend les méthodes de constructions suivantes (voir chapitre 4) :

DENN (Double-Ended Nearest Neigbor: n solutions)

SAH (Shortest Arcs Hybridation : n solutions)

MF (Multiple Fragment: 1 solution)

C3D (notre algorithme de construction vorace : 1 solution)

Un total de 2n + 2 solutions rendues réalisables sont envoyées à la mémoire centrale.

Par la suite des générations aléatoires de solution permettront d'alimenter la population en solutions diversifiées. Chaque solution aléatoire est construite en prenant aléatoirement un nœud biaisé vers celui qui possède le moins de successeurs, un successeur étant défini comme étant un nœud pouvant être atteint en respectant les contraintes de temps et de capacité. Une suite de nœuds est choisie jusqu'à ce qu'un tour soit complété, un autre tour est ensuite formé et ainsi de suite jusqu'à ce que tous les nœuds soient choisis.

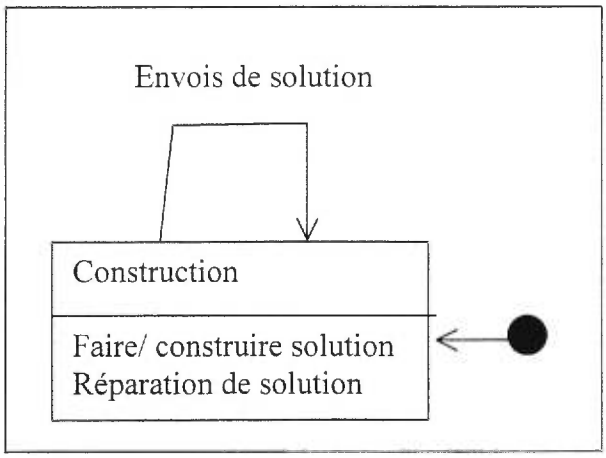

Figure 19 Diagramme d'état de l'entité capsule constructeur

5.5.4 Capsule évolutive

En plus d'une population initiale générée par la capsule constructive, nos capsules constructives nécessitent des opérateurs de croisement, un opérateur de sélection et de mutation.

5.5.4.1 Opérateur de croisement et de réparation

Dans le but d'obtenir une certaine diversité, nous avons diverses capsules génétiques travaillant sur le même *mémoire centrale* : une capsule ER et une capsule OX. Par contre, les opérateurs de croisement traditionnels (voir chapitre 4) de type ER (Edge Recombinaison), OX (Order CrossOver), PMX, cycle crossover où position crossover ne fonctionnent pas très bien pour le VRPTW selon Blanton et al. (1993) puisqu'ils sont basés sur une précédence locale et qu'ils dépendent uniquement du contenu des chromosomes impliqués. Il est donc nécessaire qu'un mécanisme de réparation tente de reconstruire un ordre de parcours réalisable. Après le croisement OX ou ER, les solutions sont réparées en examinant route par route les nœuds pour enlever ceux qui causent des irréalisabilités. Ces nœuds sont ensuite soit insérés dans d'autres tournées lorsque possible ou dans une nouvelle tournée.

5.5.4.2 Opérateur de sélection

Nous effectuons une sélection en fonction du rang de la solution qui est classé selon la fonction de coût de la mémoire centrale.

Soit r_i le rang d'une solution classée selon la fonction de coût de la mémoire centrale, la valeur attribuée à ce rang devient $v_i = 2r_i/n(n+1)$

On peut appliquer une sélection proportionnelle en choisissant la solution i avec un nombre aléatoire x entre 0 et 1 tel que $\sum_{k=0}^{i} v_k \leq x$

5.5.4.3 Opérateur de mutation

Avant le croisement, nous introduisons une probabilité α (1/n %) pour chaque nœud qu'il soit déplacé à un autre endroit dans une des tournées. Ainsi chaque copie de parents mutés est utilisée seulement pour le croisement. Il y a ainsi introduction de diversité dans la population. Il n'y a pas de réparation effectuée à ces copies.

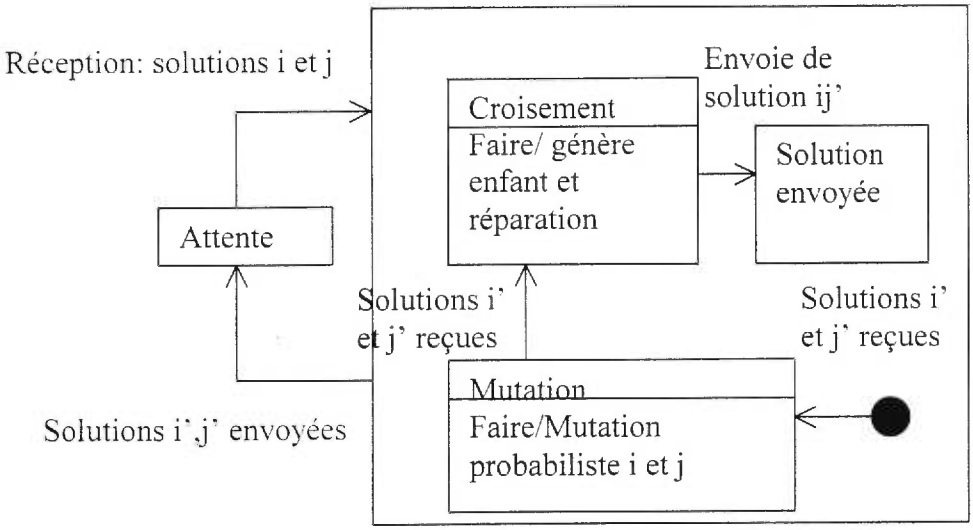

Figure 20 Diagramme d'état de l'entité capsule évolutive

5.5.5 Capsule Taburoute

Le Taburoute utilisé est une version du Taburoute VRP de Gendreau et al. (1994) adapté au VRPTW. L'adaptation fut assez directe, permettant seulement les insertions GENI lorsque les fenêtres de temps sont respectées. La nouvelle capsule Taburoute est assez simple, elle reçoit une solution de la mémoire centrale sur laquelle elle effectuera une exploration avec tabous qu'elle renverra à la mémoire centrale lors d'amélioration. Avant la phase de diversification, la solution améliorante sera renvoyée à la mémoire centrale. La mémoire centrale lui retournera une solution qui pourra constituer un nouveau point de départ si elle est meilleure que la meilleure solution de cette recherche avec tabous (figure 21).

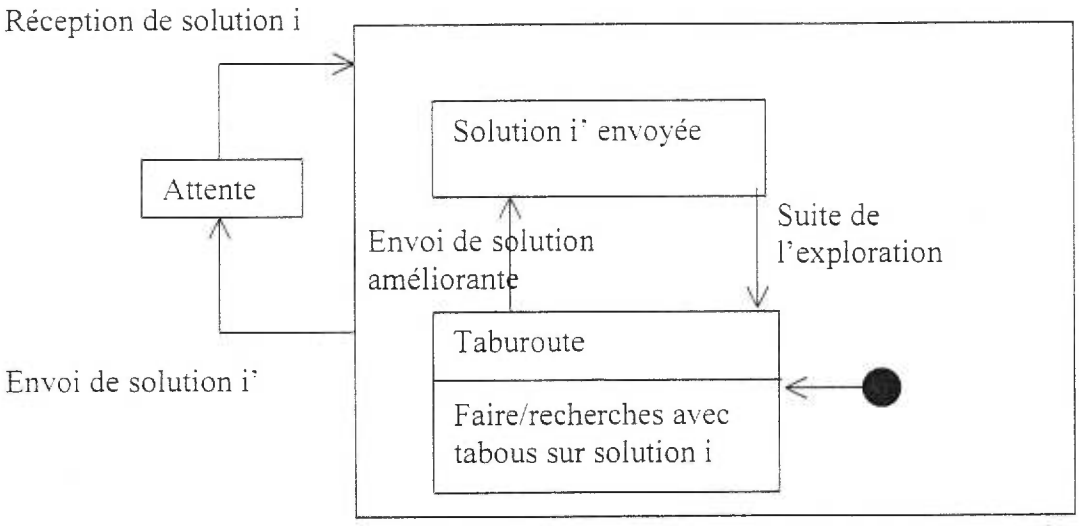

Figure 21 Diagramme d'état de l'entité capsule tabou

5.5.6 Composantes distinctes : Mémoire et capsules

On remarque que la solution constitue l'unique dépendance de relation. Cette relation est créée dynamiquement à l'exécution entre les capsules et est présente à la compilation avec le classeur de la *mémoire centrale* (Fig. 22). Notre architecture coopérative se décompose en deux parties distinctes, les capsules et la mémoire centrale. En utilisant des méthodes de résolution indépendantes et en les combinant via l'utilisation d'une méthode centrale, nous avons simplifié grandement la gestion des communications et du parallélisme ce que nous constatons dans notre modélisation UML (Unified Modelling Langage).

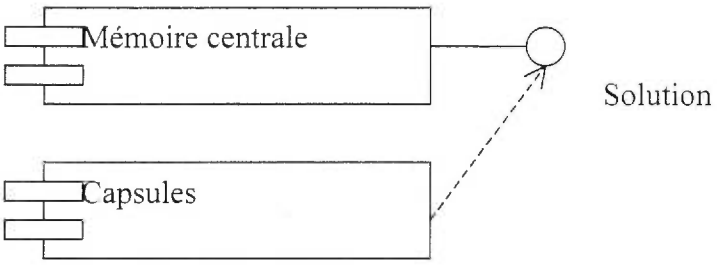

Figure 22 Diagramme de composantes

5.6 Implantations

5.6.1 Environnement coopératif

Notre architecture logicielle supporte une couche physique MIMD (multiple-instruction, multiple-data). Nous avons ainsi différentes capsules qui effectuent simultanément des tâches différentes sur divers espaces de solutions. Dans notre implantation, ces processus peuvent être indifféremment sur un système à plusieurs processeurs ou sur des stations de travail indépendantes ou une combinaison des deux. Ce choix offre une grande flexibilité d'exécution. Dans notre implantation, la mémoire centrale est gérée par une machine avec la capsule constructive et les quatre capsules génératrices de solutions (2 recherches avec tabous et 2 algorithmes évolutifs) se situent sur 4 machines différentes.

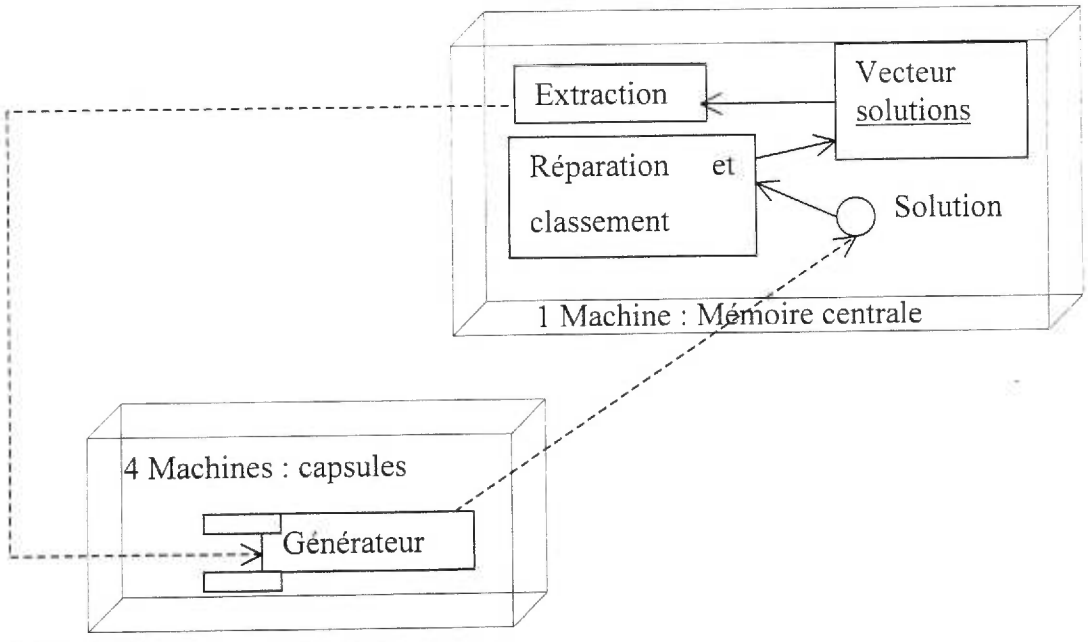

Figure 23 Diagramme de déploiement

5.6.2 Technologies

L'implantation fut effectuée en Java sous Interdev avec les bibliothèques de ObjectSpace™ (www.objectspace.com) et de DRA Systems™ (www.opsresearch.com).

Le langage Java est souvent critiqué pour ses piètres performances, spécialement pour la génération de processus selon Schader et Korthaus (1998). Notre architecture étant à gros grain, les processus possèdent une durée de vie assez longue. Cette lenteur de génération de processus est négligeable sur l'ensemble des opérations. Nous profitons pleinement de l'architecture Java (ramasse-miettes distribué, migrations de processus, accès transparents aux processus distants).

Dans notre architecture, la représentation logicielle est indépendante du contexte matériel d'exécution en terme de plate-forme et de distributions des processus. En d'autres termes un changement de système d'exploitation ou de types de machines parallèles utilisés (ordinateur parallèle ou parc d'ordinateur) n'influence que les paramètres de configurations de l'architecture.

5.7 Développement structuré

L'architecture utilisée fut clairement divisée par la modélisation UML permettant ainsi un développement modulaire pour chacune des capsules et de la *mémoire centrale*.

Le temps de développement distribué se résuma à définir les ports et les connecteurs tels que définis dans Bergner et al., (1997) sur nos diverses méthodes de résolution. Les divers schémas simplifiés dans ce document permirent une transition un à un dans le modèle de l'ORB d'ObjectSpace™.

La portabilité de Java et de ObjectSpace™ (entièrement écrit en Java) permet l'exécution sur un parc de système hybrides Windows™ NT et Solaris™, de systèmes à un processeur ou de systèmes multiprocesseurs à mémoire partagée.

5.8 Observations sur UML

UML nous a donc permis de regarder les aspects statiques de notre modèle et les aspects dynamiques. Nous avons identifié les communications et dans quelles circonstances elles apparaissent. L'architecture proposée est très simple du fait qu'elle ne requiert pas de synchronisme puisqu'il n'y a pas de dépendance entre les différentes parties de la résolution. Elle devrait permettre l'ajout d'autres types de capsules en définissant les ports et les protocoles de communication pour chacune d'elles.

Chapitre 6 Résultats, contributions et conclusions

6.1 Expérimentation

L'objectif de notre expérimentation était, d'une part de montrer la facilité de réalisation d'une telle architecture par une modélisation appropriée, et d'autre part de mesurer la stabilité et la qualité des solutions produites. Non seulement les résultats obtenus sont comparables aux résultats mentionnés dans la littérature et la combinaison de méthodes de résolutions s'avère utile, mais notre architecture s'avère tout à fait adaptée aux plus gros problèmes. Nous présentons et analysons ainsi les résultats dans la perspective de combinaison de méthodes séquentielles simples.

6.2 Présentation des résultats

Avant toute présentation ou analyse de résultats, il est important de regarder quelques nuances par rapport aux comparaisons possibles entre différents algorithmes. Tel que relevé par Barr et al. (1995), la comparaison de l'exécution des différents algorithmes de VRP est régie par différents éléments. Puisque chaque algorithme est régi par un ensemble de paramètres, il est important d'utiliser le même ensemble de paramètres pour tous les tests. Les paramètres doivent être assez généraux pour pouvoir être appliqués à un autre ensemble de tests sans être re-calibrés. Par conséquent, les résultats doivent être fournis pour une configuration de paramètres. Les meilleurs résultats avec des paramètres différents peuvent accompagner ces résultats. La représentation de certaines valeurs doit être constante en précision, en particulier les distances ne doivent pas être tronquées. Un changement de précision des distances représenterait un tout autre problème. Nous porterons une attention particulière à ces comparaisons lors de la présentation de nos résultats.

6.3 Résultats numériques

Nous fournissons les résultats numériques de notre résolution coopérative avec algorithmes évolutifs et recherches avec tabous. Les résultats et tous les calculs on étés fait avec des nombres réels à double précision. Les problèmes utilisés sont les 56 problèmes Euclidiens à 100 clients et un dépôt créés par Solomon (1987). Les clients sont distribués dans un carré de $[0,100]^2$ unités. Six ensembles de problèmes tests sont définis, C1, C2, R1, R2, RC1 et RC2. Les clients des problèmes de la classe C sont regroupés en régions alors que ceux de la classe R sont distribués uniformément. Les problèmes de la classe RC combinent ces deux propriétés. Les fenêtres de temps sont réduites au dépôt pour les problèmes de type 1 permettant d'avoir moins de clients par route et plus large pour les problèmes de type 2. Les temps de service sont de 10 unités de temps pour les problèmes de classe R et RC et 90 unités pour ceux de classe C. Nous présentons aussi des résultats préliminaires sur une extension des problèmes de Solomon, soit 60 problèmes à 1000 clients et un dépôt qui respectent la même structure que les précédents.

On retrouve l'identificateur du problème dans la première colonne, le nombre de véhicules et la distance totale pour RT : Rochat et Taillard (1995), CR : Chiang et Russell (1997), TB : Taillard et al. (1997), HG : Homberger et Gehring (200) CLM: Cordeau et al. (2000), Rousseau et al. (2000). Nous indiquons dans la dernière colonne une étoile lorsqu'il y a égalité avec le meilleur résultat et deux étoiles lorsque notre résultat est meilleur pour notre résolution CKL : Le Bouthillier et al. (2000).

Pour la résolution des 56 problèmes de Solomon, nous avons utilisé 5 stations de travail PC 5550Mhz, 128Mo utilisant Windows NT4 sp6 et la librairie d'ObjectsSpace. Les temps de calcul des résultats sont limités à 3600 secondes au total par problème sur un cluster de 5 PIII 500Mhz 128Mo. Il y a donc 12 minutes de temps réel qui est alloué à chaque processus.

6.4 Paramètres

Notre mémoire centrale possède une taille fixe de 2n+2 solutions, les solutions de moindre qualité selon notre fonction de classement sont éliminées lorsque de meilleures solutions arrivent. Toutes les solutions de la mémoire centrale subissent des optimisations locales 2-opt, 3-opt et Or-opt avant d'être classées. Rappelons la fonction de classement :

$C(p) = W_1$*Temps_total + W_2*Distance_totale + W_3*Attente_totale + W4*Nombre_véchicules + W_5*Temps_restant_clients

Paramètres utilisés pour la mémoire centrale, la sélection pour les capsules évolutives ainsi que lors de la diversification de la recherche avec tabou :

W1= 0,2, W2=0,2, W3=0,1, W4=0,4, W5=0,1

6.4.1 Taburoute I

p-voisinage : 15% des nœuds

Nombre de solutions initiales : n/15

Durée des étiquettes tabous : [5,10] itérations

Facteur de pénalité des arcs avec déplacements fréquents : 1

Racine pour génération de nombre aléatoire : a

Solution initiale : A

6.4.2 Taburoute II

p-voisinage : 20% des nœuds

Nombre de solutions initiales : n/20

Durée des étiquettes tabous : [10,15] itérations

Facteur de pénalité des arcs avec déplacements fréquents : 0.5

Racine pour génération de nombre aléatoire : b

Solution initiale : B

6.4.3 Algorithme évolutif I

Croisement : OX

Mutations avant le croisement: 1% sur chaque copie de parent

Fonction de sélection : biaisé vers le meilleur classement de la mémoire centrale

6.4.4 Algorithme évolutif II

Croisement : ER

Mutations avant le croisement: 1% sur chaque copie de parent

Fonction de sélection : biaisé vers le meilleur classement de la mémoire centrale

6.5 Temps de calculs requis

L'imposition d'une limite de temps de 720 secondes sur chacun des 5 processeurs pour l'exécution de notre méthode avec les problèmes de 100 villes permet une comparaison équitable avec les parties séquentielles. Les différentes méthodes utilisent entre 1 minute et plusieurs heures de calculs sur différentes machines avec différents langages et différentes méthodes parallèles ou séquentielles de résolutions. Les comparaisons de temps sont ainsi difficiles à effectuer mais on peut obtenir une idée sur les ordres de grandeur. Par exemple, RPG : Rousseau et al. (2000) utilisent 11000 secondes sur un UltraSparc, CR : Chiang et Russel (1997) environ entre 2000 secondes et 10000 secondes selon le problème sur un Pentium 166 Mhz, HG: Homberger et Gehring (2000) 300 secondes sur un ensemble de 5 pentiums 200Mhz, TB : Taillard et al. (1997) environ entre 2000 et 13000 secondes sur un IBM 6000 RISC, Rochat et Taillard (1995) environ entre 2000 et 13000 secondes sur un Silicon Graphics Indigo 100Mhz.

Nos résultats sont trouvés souvent avant la fin de cette période de temps pour les problèmes des classes C1 et C2. Le travail total fourni par notre méthode peut être de 5*750 secondes ou 4*750 secondes s'il est considéré que la mémoire centrale effectue peu de travail, ce qui est dans une même ordre de grandeur que les autres méthodes.

6.6 Tableau des résultats

Nous présentons les résultats par moyenne sur chaque classe de problème (R1, C1 et RC1, R2, C2, RC2). Le nombre de véhicules moyen utilisés par classe de problèmes ainsi que la distance totale moyenne figurent par type de problème. Le pourcentage d'écart pour le nombre de véhicule se fait par rapport à HG : Homberger et al. (2000) qui obtient à notre connaissance les meilleurs résultats en moyenne. La colonne LCKIND représente la moyenne des meilleurs résultats fournis par les méthodes indépendantes utilisées par notre algorithme coopératif.

Sur une moyenne des 56 problèmes, nous obtenons un surplus de véhicule de 0,8 % et de meilleures distances de 0.2 % comparativement à HG qui obtient les meilleurs résultats en moyenne sur la distance pour le plus petit nombre de véhicules utilisés. Rochat et Taillard obtiennent les meilleurs résultats en moyenne sur la distance totale mais utilisent plus de véhicules. On constate que les classes C1 et C2 sont très faciles à résoudre puisque la majorité des méthodes arrivent aux même résultats. En regardant les meilleurs résultats qui sont en caractère gras, nous constatons qu'il n'y pas une seule méthode qui produit tous les meilleurs résultats. Les résultats détaillés se trouvent en annexe.

No	TB1	TB2	ER	OX
R1	12,17 1211,10	12,17 1215,45	12,25 1211,38	12,49 1218,79
C1	**10,00** **828,38**	**10,00** **828,38**	**10,00** **828,38**	**10,00** **828,38**
RC1	11,50 1388,24	11,67 1384,65	11,88 1394,28	11,74 1380,29
R2	**2,94** **955,18**	2,96 958,95	2,94 966,78	3,02 954,32
C2	**3,00** **589,86**	**3,00** **589,86**	**3,00** **589,86**	**3,00** **589,86**
RC2	3,25 1145,29	3,25 1147,00	3,38 1118,63	3,31 1144,13
AVG	7,14 1019,67	7,18 1020,71	7,24 1018,22	7,26 1019,29
(%)	0,00285 -0,00135	0,00737 -0,000335	0,0167 **-0,0028**	0,0193 -0,0017

Tableau 3 Résultats moyens des méthodes indépendantes par classe de problèmes de 100 villes et 1 dépôt de Solomon

No	CR (1997)	CLM (2000)	RGP (2000)	RT (1995)	TB (1997)	HG (1999)	LCKIND (2000)	LCK (2000)
R1	12,17 1203,03	12,08 1210,14	12,08 1210,21	12,25 1208,50	12,17 1209,27	11,92 1220,97	12,17 1211,10	12,17 1209,27
C1	**10,00** **828,38**	**10,00** **828,38**	**10,00** **828,38**	**10,00** **828,38**	**10,00** **828,38**	**10,00** 828,94	**10,00** **828,38**	**10,00** **828,38**
RC1	11,88 1397,44	11,50 1389,78	11,63 1382,78	11,88 1377,39	11,50 1389,22	11,50 1388,24	**11,50** 1388,24	11,50 1389,22
R2	2,73 985,68	2,73 969,57	3,00 941,08	2,91 961,72	2,82 980,27	2,73 968,55	2,94 966,78	2,82 965,91
C2	**3,00** 591,42	**3,00** **589,86**	**3,00** **589,86**	3,00 **589,86**	**3,00** **589,86**	**3,00** **589,86**	**3,00** **589,86**	**3,00** **589,86**
RC2	**3,25** 1231,57	**3,25** 1134,52	3,38 **1105,22**	3,38 1119,59	3,38 1117,44	**3,25** 1140,42	**3,25** 1145,29	**3,25** 1143,70
AVG	7,17 1039,59	7,09 1020,37	7,18 **1009,59**	7,23 1014,24	7,14 1019,07	**7,07** 1022,83	7,14 1021,61	7,12 1021,06
(%)	0,015 0,016	0,004 -0,002	0,016 **-0,013**	0,024 -0,008	0,011 -0,004	**0,000** 0,000	0,011 -0,001	0,008 -0,002

Tableau 4 Résultats séquentiels (CR, CLM, RGP), parallèles (RT,TB,HG) et indépendants (LCKIND) moyens par classe de problèmes de 100 villes et 1 dépôt de Solomon

Sur deux problèmes de la série RC1, la recherche avec tabu a donné de meilleurs résultats lorsque utilisée seule mais en moyenne, il n'y a pas de dégradation de solution en combinant les méthodes indépendantes mais plutôt une légère amélioration. En considérant que le temps de calcul alloué aux méthodes séquentielles est de 3600 secondes, que le temps alloué à chaque méthode lorsqu'utilisée dans l'algorithme coopératif est de 720 secondes par processeur (soit 720*5=3600 secondes au total), on obtient un rapport linéaire de temps d'exécution parallèle sur séquentiel. Il faut par contre être conscient que les méthodes séquentielles, une fois combinées, ne constituent plus le même algorithme et que la comparaison ne peut s'effectuer qu'à titre indicatif. Mentionnons aussi qu'une légère amélioration (0,2%) des résultats obtenus est toutefois très appréciable pour ce genre de problèmes.

No	HG (1999)	LCK (2000)
R1	**91,90** 57186	92,20 **57034,82**
C1	**96,00** 43273	**96,00** **43246,22**
RC1	**90,00** **50668**	**90,00** 50679,36
R2	**19,00** **31930**	**19,00** 31946,06
C2	**30,20** **17570**	**30,20** 17570,28
RC2	**19,00** **27012**	**19,00** 27084,94
AVG	**57,68** 37939,83	57,73 **37926,95**
%	0,0000 0,0000	0,00087 **-0,00034**

Tableau 5 Résultats préliminaires parallèles moyens par classe de problèmes de 1000 villes et 1 dépôt de Solomon

Nous utilisons les mêmes temps de calculs pour les problèmes à 1000 villes que Homberger et Gehring (2000), soit 3000 secondes par processeur et obtenons des résultats sensiblement équivalents (tableau 7). En comparant nos solutions et celles de Homberger et al. des problèmes de 100 villes par rapport aux problèmes de 1000 villes, on constate qu'il y réduction de l'écart moyen, ce qui peut indiquer une bonne robustesse de notre architecture face aux plus gros problèmes.

6.6.1 Nouvelle solution

Une meilleure solution a été trouvée pour le problème rc204

Distance : 798.46

3 véhicules

0 81 96 54 41 39 42 44 43 40 36 35 37 38 72 71 93 67 84 85 63 33 32 30 28 26 27 29 31 34 50 95 56 64 66

0 69 98 82 10 11 15 16 17 47 14 12 53 60 78 73 79 7 8 46 45 5 3 1 4 6 2 88 55 100 70 61 68

0 80 91 92 94 62 51 89 76 18 23 21 48 19 49 20 57 99 52 87 9 13 86 74 59 97 75 58 77 25 24 22 83 65 90

6.6.2 Interprétation

Dans l'ensemble des résultats comparables à la littérature sont obtenus avec notre méthode coopérative. Il est très important de noter que les méthodes séquentielles impliquées pourraient être remplacées par de meilleures méthodes et ainsi améliorer la qualité des résultats. Pour un effort de calcul similaire, mais des temps de calcul réduit, des résultats de meilleures qualités sont obtenus et combinent la qualité des méthodes séquentielles indépendantes.

6.7 Travaux futurs

La liste des processus suivie par chaque solution n'est pas répertoriée. Il serait intéressant de savoir quel est le parcours des meilleures solutions pour être en mesure d'apprécier l'influence des divers algorithmes. Nous pouvons quand même pour le moment déduire d'après les données brutes, qu'il n'est pas nécessaire de faire un calibrage des paramètres et que les paramètres différents pour les capsules de même type (i.e. les deux capsules évolutives ER, OX et les deux capsules Taburoute) suffisent à diversifier la recherche.

Le critère d'ordonnancement devrait être raffiné en ajoutant des étiquettes tabous sur les solutions trop fréquemment utilisées dans la mémoire centrale. Le critère d'arrêt basé sur une limite de temps est présentement trop restrictif, il devrait plutôt se baser sur l'évolution de la mémoire centrale et du comportement des processus ou sur une approximation de la couverture de l'exploration de l'espace des solutions.

Des statistiques sur les fréquences d'arcs utilisés, d'entrées et sorties de solutions des processus permettraient de faire un suivi des processus pour éventuellement remplacer certains processus inefficaces par d'autre processus ou effectuer un changement dynamique de paramètres.

Nous avons déjà des résultats préliminaires sur les problèmes à 1000 clients et prévoyons effectuer d'autres expériences sur les problèmes de Solomon étendus de 200 à 800 clients pour le comportement de notre architecture dans divers environnements.

6.7.1 Recherche dispersée

La performance d'un algorithme évolutif repose principalement sur la méthode de croisement et le choix des parents.

Un algorithme évolutif tente d'extraire les bons chromosomes de deux parents pour générer un enfant possédant de bons chromosomes. Il serait intéressant d'effectuer des classifications de chromosomes pour identifier les mauvaises combinaisons et les bonnes. L'utilisation de la recherche dispersée (scatter search) introduite par Glover (1997) pour trouver des solutions qui correspondent à des combinaisons convexes de solutions d'élite devrait offrir de meilleurs résultats que les simples croisements de deux parents.

6.7.2 Combinaison avec d'autres capsules

Notre architecture permet d'intégrer facilement d'autres métaheuristiques pour la résolution du VRPTW. Il suffit d'ajouter un port et un connecteur à chacun d'eux et idéalement d'avoir un code Java. Cette combinaison permettrait d'obtenir une diversification de la population. Nous prévoyons ainsi augmenter le nombre de capsules en utilisant plus de processus et des algorithmes de recherches différents pour vérifier la tolérance de notre méthodologie aux communications intensives.

6.7.3 Balancement de charge de la mémoire centrale

Même avec une architecture à gros grain, la demande de solution à la *mémoire centrale* pourrait éventuellement saturer les communications. Il serait relativement aisé d'effectuer une redirection vers une copie de la *mémoire centrale* pour certaines requêtes. Il pourrait même s'y trouver des *mémoires centrales* différents avec différentes méthodes de classement et de sélection, formant ainsi des hiérarchies de mémoire.

6.7.4 Migration des paramètres ou changement de processus

Avec l'ajout de mémoire persistante, il est possible que certaines mémoires deviennent mauvaises ou convergent trop vite. Une migration d'autres mémoires vers ces dernières permettrait de redémarrer le processus.

Il est possible que certains types de processus de résolution soit mal orientés pour un type de problème. La *mémoire centrale* serait en mesure d'identifier ces processus et de les remplacer par des processus plus performants.

6.7.5 Proximité des solutions dans l'espace

Il serait intéressant pour les algorithmes évolutifs d'inclure dans la fonction de coût une notion de proximité dans l'espace des différentes solutions dans le but de combiner les solutions ayant certaines tournées «similaires», renforçant ainsi le mécanisme d'adaptabilité. Notre métrique nous permettrait de combiner des solutions ayant le même niveau de contrainte ou des niveau de contraintes différentes.

6.8 Contributions et conclusions

Rappelons que le problème de tournées de véhicule avec fenêtre de temps (*VRPTW*) est un problème fréquemment rencontré dans nombres d'industries, spécialement en transport. Une réduction des coûts associés à ce transport et des temps de calculs peut justifier l'utilisation de résolutions coopératives.

La modélisation UML de notre architecture coopérative permet une validation des différents aspects du modèle représentés par le diagramme de collaboration, de séquence, d'état, d'activité, de composante et de déploiement qui permettent de simuler le comportement d'un système parallèle et d'en prévoir les interactions. Nous avons ainsi développé un concept générique d'une architecture coopérative appliqué à un cas particulier, la résolution du *VRPTW*. Nous croyons que ce concept de mémoire centrale peut s'appliquer à plusieurs problèmes d'optimisations combinatoires dans une combinaison de métaheuristiques.

L'architecture coopérative développée simplifie la parallélisation de métaheuristiques dans le sens où il faut simplement combiner ces algorithmes indépendants par des échanges de solutions avec une mémoire centrale pour obtenir un résultat globalement supérieur aux résultats individuels des méthodes indépendantes. Il est aussi possible de rajouter des capsules contenant d'autres algorithmes de résolutions dans le but de combiner les effets de ces différentes recherches. Le choix des capsules actuelles à savoir les recherches avec tabous et les algorithmes évolutifs semble être une excellente combinaison de solutions et de recherche au sein de voisinage de solutions.

Nous avons adapté la métrique de calcul de distance de Solomon permettant entre autres une visualisation en trois dimensions du problème, fournissant ainsi plus d'informations graphiques, permettant le classement de solutions dans la mémoire centrale, offrant une nouvelle possibilité de génération de solutions initiales et permettant la catégorisation des problèmes par niveau de contraintes associés.

Au niveau des performances, nous avons eu par l'utilisation d'une parallélisation gros grain, des performances très acceptables sous Java en bénéficiant de tous les avantages intrinsèques à ce paradigme.

Mentionnons également des solutions comparables aux meilleures valeurs connues ainsi qu'une meilleure solution trouvée.

6.9 Bibliographie

Antes, J. , Derigs, U., *A new parallel tour construction algorithm for the vehicle routing problem with time windows*, tech. rep., Lehrstuhl fur Wirtschaftsinformatik und Operations Research, Universitet zu Koeln, 1995.

Badeau, P., Gendreau, M., Guertin, F. Potvin, J.Y., Taillard, E, *A Parallel Tabu Search Heuristic for the Vehicle Routing Problem with Time Windows*, Transportation Research 5C, 109-122, 1997.

Barr, R.S. Golden, B.L. Kelly, J.P. Resende, M.G.C., Stewart Jr. W.R., *Designing and reporting on computational experiments with heuristic methods*, Journal of Heuristics 1, 9-32., 1995.

Battiti, R., Tecchhiolli, G., *The reactive Tabu Search,* ORSA Journal on Computing 6:2, 126-140, 1994.

Beame, P., Cook, S., Edmonds, J., Impagliazzo, R., Pitassi, T., *The Relative Complexity of NP Search Problems,* Proceedings of the Twenty-Seventh Annual ACM Symposium on the Theory of Computing, pp. 303-314, 1995.

Benyahia, I., Potvin, J.Y., Dube, D., *Parameter Optimization for a Vehicle Routing Heuristic Using Genetic Algorithms,* CRT Pub. no 938, 1993.

Bergner, K. , Rausch, A., Sihling, M., *Using {UML} for Modeling a Distributed {Java} Application*, Technische Universitat Munchen, Institut fur Informatik, 1997.

Bernstein, A. J., "*Analysis of programs for parallel processing*," IEEE Transactions on Computers, vol. EC-15,5, pp. 757-762, 1966.

Blanton, J.L. , Wainwright, R.L., *Multiple vehicle routing with time and capacity constraints using genetic algorithms*. In Proceedings of the 5th International Conference on Genetic Algorithms, pages 452-459, San Mateo, CA, 1993.

Booch, G., Rumbaugh, J., *Unified Method for Object-Oriented Development*, Rational Software Corporation, 1995.

Bodin, L., Berman, L., *Routing and scheduling of school buses by computer,* Transportation Science, 13(2), 113-129, 1979.

Chiang, W.C., Russell, R.A., A reactive tabu search metaheuristic for the vehicle routing problem with time windows. INFORMS Journal on Computing, 9 : 417-430, 1997.

Chiang, W.C., Russell, R.A., *Hybrid heuristics for the vehicle routing problem with time windows*, Working paper, Department of Quantitative Methods, University of Tulsa, Tulsa, OK, 1993.

Christofides, N. Mingozzi, A., Toth, P., *The vehicule Routing Problem,* In N. Christofides, Mingozzi, A., Toth, P, Sandi, C. (eds), Combinatorial Optimization. Wiley, Chichster, 1979.

Clarke, G., Wright, J.W. Sheduling of vehicules from a central depot to a number of delivery ponts, Operations Research 12, 568-581, 1964.

Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M.M., Soumis, F., *The VRP with Time Windows*, GERAD G-99-13, 1999.

Cook, Stephan A., *A Hierarchy for Nondeterministic Time Complexity,* Conference Record, Fourth Annual ACM Symposium on Theory of Computing, pp. 187-192, 1-3, 1972.

Cordeau, J.-F., Laporte, G., Gendreau, M., A tabu search heuristic for periodic and multi-depot vehicle routing problems, Networks, 30 :105-119, 1997

Cordone, R. , Calvo, R. Wolfer, *A heuristic for vehicle routing problem with time windows*. Internal report 97.012, Politechnico di Milano, Dipartimento di Electronica e Informazione, Milan, Italie, 1997.

Crainic,T.G.; Toulouse,M.; Gendreau,M., *Towards a Taxonomy of Parallel Tabu Search Algorithms*, INFORMS Journal on Computing, 9(1):61--72, 1997.

Crainic, T.G , Toulouse, M, *Parallel Metaheuristics,* Fleet Management and Logistics, 205-251, 1998.

Crainic, T.G., *Méthodes Parallèles de Recherche avec Tabous,* Calculateurs Parallèles, Réseaux et Systèmes répartis, 171-193, 1998.

Crainic, T.G., Toulouse, M., Gendreau, M., *Towards a Taxonomy of Parallel Tabu Search Algorithms,* INFORMS Journal on Computing, 9:1, 61-72, 1997.

Crainic,T.G., Gendreau, M., *Towards an Evolutionary Method-Cooperating Multi-Thread Parallel Tabu Search Hybrid.* Meta-Heuristics 98: Theory & Applications, 331-344, 1998.

Desrochers, M., Desrosiers, J., Solomon, M., A new Optimization Algorithm for the Vehicle Routing Problem with Time Windows, Operations Research, 1992, 40, 342-354

Enslow, H.P., What is a "Distributed" Data Processing System?, Computer 11, 13-21, 1978.

Fisher, M., Jörnsten, K.O., Madsen, O.B.G., *Vehicle Routing with Time Windows*, Research report 4C/1991, IMSOR, the Tech. University of Denmark, 1991.

Fisher, M.L., Jaikumar, R., A generalized assignment heuristic for the vehicle routing, Networks 11, 109-124, 1981.

Flood, M.M.. *The traveling salsman problem.* Operations Resarch, 4 :61-75, 1956.

Garcia, B. L., Potvin, J.-Y., Rousseau, J.-M., A parallel implementation of the tabu search for vehicle routing problems with time window constraints, Computers & Operations Research, 21, 1994.

Garey M., Johnson, D., *Computers and Intractibility, a Guide to the Theory of NP-Completeness*, W.H. Freeman and Co., 1979.

Gendreau, M. Hertz, A., Laporte, G., *A tabu search heuristic for the vehicle routing problem, Manag. Sci.*, vol. 40, pp. 1276-1290, 1994.

Gendreau, M. Hertz, A., Laporte, G.,, New insertion and postoptimization procedures for the traveling salesman, Operations Res., vol. 40, pp. 1086-1094, 1992.

Gillet, B.E., Miller, L.R., *A heuristic algorithm for the vehicule dispatch problem*, Operations Research 22, 240-349, 1974.

Glover, F., Heuristics for Interger Programming Using Surrogate Constraints, Decision Sciences 8, pp. 156-166, 1977.

Glover, F., Tabu search Part I, *ORSA Journal on Computing*, 1 :190-209, 1989.

Glover, F., Tabu search Part II, *ORSA Journal on Computing*, 2 :4-32, 1990.

Goldberg, D., Genetic Algorithms in Search, Optimization, and Machine Learning, Addison-Wesley, 1989.

Golden, B., Assad, A., Levy, L., Gheysens, F., The fleet size and mix vehicle routing problem, Management Science & Statistics, Working Paper No 82-020, University of Maryland at College Park, 1982.

Golden, B.L., Magnanti, T.L., Nguyen, H.Q., *Implementing vehicle routing algorithms*. Networks, 7:113-148, 1977.

Harel, D. , Pnueli, A. , Schmidt, J. P. , Sherman, R. , *On the Formal Semantics of State Charts, Proc. 2nd Symp. on Logic in Computer Science (LICS 87)*, pp. 54-64, IEEE Computer Society Press, 1987.

Holland, J., *Adaptation in Natural and Artificial Systems*, Ann Arbor: University of Michigan Press, 1975.

Homberger, J., Gehring, H., Two evolutionary metaheuristics for the vehicle routing problem with time windows, INFOR, 37:297-318, 1999.

Jacobson, I., Christerson, M., Jonsson, M., Overgaard, P., *OO Software Engineering, A Use Case Driven Approach*, Addison-Wesley, 1992.

Jézéquel, J.-M. , Le Guennec, A., Pennaneach, F., *Validating Distributed Software Modeled with UML,* The Unified Modeling Language, UML'98 - Beyond the Notation. First International Workshop, Mulhouse France, June 1998,, LNCS, Vol. 1618, pp. 331-340, Springer, 1998.

Johnson, D.S., McGeoch, L.A., *The travelling Salesman problem : a case study,* Local Search in Combinatorial Optimisation, John Wiley and Sons Ltd., 215-310, 1997.

Kahkipuro, P., *UML Based Performance Modeling Framework for Object-Oriented Distributed Systems*, UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings, LNCS, Vol. 1723, Springer, 1999.

Kearney, A. T., Inc., *Measuring and Improving Productivity in Physical Distribution*, A report prepared for the National Council of Physical Distribution Management, NCPDM, Oak Brook, IL., 1984.

Keller, R.K., Badidi, E., Kropf, Peter G., Van Dongen, V., *Dynamic Server Selection in Distributed Object Computing Systems*, Herwig Unger, editor, Proceedings of the Workshop on Distributed Computing on the WEB, pages 39-47, Rostock, Germany, German Computer Society (GI), Rostock University Press, 1998.

Keller, R.K., Khriss, I., Elkoutbi, M., *Automating the Synthesis of UML Statechart Diagrams from Multiple Collaboration Diagrams*, Proceedings of the International Workshop on the Unified Modelling Language UML'98: Beyond the Notation, pages 115-126bis, Mulhouse, France, June 1998.

Kinderwater, G.A.P., Savelsbergh, M.W.P., *Vehicle Routing: Handling Edge Exchanges*. In E.H.L. Aarts, J.K. Lenstra (eds), *Local Search in Combinatorial Optimization*. Wiley, Chichester, 1997.

Kohl, N, Madsen, O, An optimization algorithm for the vehicule routing problem with time windows bases on lagrangian relaxation, Operation Research, vol. 45, No. 3, 1997.

Laporte, G., Gendreau, M., Potvin, J.-Y., Semet, F., *Classical and Modern Heuristics for the Vehicle Routing Problem*, SIAM Monography on Discrete Mathematics and Applications, 2000.

Lin, S., Kernighan, B., *Computer solutions of the traveling salsman problem*, Bell System Technical Journal 44, 2245-2269, 1965.

Mole, R.H., Jameson, S.R., A sequantial route-building algorithm emplying a generalized saving criterion, Operational Research 27, 503-51, 1976.

Oldevik, J., Berre, Arne J., *{UML}-based Methodology for Distributed Systems*, Proceedings of The Second International Enterprise Distributed Object Computing Workshop, IEEE, 1998.

Or, I., *Traveling Salesman-Type Combinatorial Problems and Their Relation to the Logistcs of Blood Banking*, Ph.D. Thesis, Dept. of Industrial Engineering and Management Sciences, Northwestern University, 1976.

Osman, I.H., Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem, Annals of Operations Research 41, 421-451, 1993.

Potvin, J.-Y., Bengio S., The vehicle routing problem with time windows - part 2: Genetic search, ORSA J. Comp., vol. 8, pp. 165-172, 1996.

Rego, C., Roucairol, C., *A Parallel Tabu Search Algorithm using Ejection Chains for the Vehicle Routing Problem*. In I.H. Osman and J.P. Kelly (eds), MetaHeuristics: Theory and Applications. Kluwer, Boston, 1996.

Renaud, J. Boctor, F.F., Laporte, G, *A fast composite heuristic for the symmetric traveling salesman problem*, INFORMS Journal on Computing 8, 134-143., 1996.

Rochat, Y., Taillard, E.D. Probabilistic diversification and intensification in local search for vehicle routing, Journal of Heuristics 1, 147-167., 1995.

Russell, R.A., Hybrid heuristics for the vehicle routing problem with time windows, Transportation Science, 29:156-166, 1995.

Ryan, D.M. Hjorring, C., Glover, F., *Extensions of the petal method forvehicule routing*, Journal of the Operational Research Society 44, 289-296, 1993.

Schader, M., Korthaus, A., *Modeling Java Threads in UML*, The Unified Modeling Language -- Technical Aspects and Applications, pp. 122-143, Physica-Verlag, Heidelberg, 1998.

Solomon, M. , Desrosiers, J., Time window constrained routing and scheduling problems, Transp. Sci., vol. 22, pp. 1-13, 1988.

Solomon, M. , Vehicle Routing and Scheduling with Time Window Constraints: Models and Algorithms, Department of Decision Sciences, University of Pennsylvania, 1983.

Solomon, M., Algorithms for the vehicle problem and scheduling problem with time window constraints, Operations Research 35, 254-265, 1987.

Solomon, M., Baker, E.,, Schaffer J., Vehicle routing and scheduling problems with time window constraints: Efficient implementations of solution improvement procedures. In B.L. Golden and A.A. Assad, editors, Vehicle routing: Methods and studies, pages 85-106. North-Holland, Amsterdam, 1988.

Statistiques Canada, Bullettin de service – Tranports teresste et maritme, no 50-002-XIB, Statistiques Canada, 2000.

Taillard, E. , Badeau, P., Gendreau, M, Guertin, F., J.-Y. Potvin, *A tabu search heuristic for the vehicle routing problem with soft time windows*. Transportation Science, 31:170-186, 1997.

Taillard, E. Parallel iterative search methods for vehicle routing problems, Networks, vol. 23, pp. 661-673, 1993.

Taillard, E., Gambardella, L. M., Gendreau, M., Potvin, J.-Y., *Programmation à mémoire adaptative*, Technical report IDSIA-79-97, IDSIA, Lugano, 1997. Published in: Calculateurs Parallèles, Réseaux et Systèmes répartis 10, 117-140, 1998.

Thangiah, S. R., Petrovic, P., *Introduction to genetic heuristics and vehicle routing problems with complex constraints*. In Advances in computational and stochastic

optimization, logic programming, and heuristic search, Oper. Res. / Comput. Sci. Interfaces Ser. 9, pages 253-286. Kluwer, Boston, 1998.

Thompson, P.M., Psaraftis, H.N., Cyclic transfer algorithms for the multi vehicle routing and scheduling problems, Operations Research 41, 935-946, 1993.

Toth, P., Vigo, D., *Granular Tabu Search. Working paper*, DEIS, University of Bologna., 1998.

Toulouse, M., Crainic, T.G., Sansò, B., Thulasiraman, K., Global Optimization Properties of Parallel Cooperative Search Algorithms: A Simulation Study, Parallel Computing, 26:1,91-112, 2000.

Volgenant, A., Jonker, R *The symmetric traveling salesman problem and edge exchange in minimal-trees*, European Journal of Operational Research 12, 394-403, 1983.

Watson, J.-P., Ross, C. Eisele, V., Denton, J. , Bins, J., Guerra, C., Whitley, D., Howe, A., *The Traveling Salesrep Problem*, Edge Assembly Crossover, and 2-opt., 1998.

Xu, J., Kelly, J.P., A network flow-based tabu search heuristic for the vehicle routing problem, Transportation Science 30, 379-393, 1996.

Chapitre 7 Annexe

No	RT (1995)	CR (1997)	TB (1997)	HG (1999)	CLM (2000)	RGP (2000)	LCK (2000)
R101	19,00	19,00	19,00	19,00	19,00	19,00	19,00
	1650,80	1651,01	1650,79	1650,80	1650,80	1650,79	1650,79
R102	17,00	17,00	17,00	17,00	17,00	17,00	17,00
	1486,12	1489,13	1487,60	1486,12	1488,10	1486,11	1487,60
R103	14,00	13,00	13,00	13,00	13,00	13,00	13,00
	1213,62	1299,15	1294,24	1292,85	1299,78	1292,67	1294,24
R104	10,00	10,00	10,00	9,00	10,00	10,00	10,00
	982,01	985,98	982,72	1013,32	984,00	987,35	982,72
R105	14,00	14,00	14,00	14,00	14,00	14,00	14,00
	1377,11	1382,05	1377,11	1377,11	1377,11	1377,11	1377,11
R106	12,00	12,00	12,00	12,00	12,00	12,00	12,00
	1252,03	1255,34	1259,71	1260,12	1253,23	1252,61	1259,71
R107	10,00	10,00	10,00	10,00	10,00	10,00	10,00
	1159,86	1124,72	1126,69	1120,85	1113,69	1110,40	1126,69
R108	9,00	9,00	9,00	9,00	9,00	9,00	9,00
	980,95	971,03	968,59	970,05	964,38	969,02	968,59
R109	11,00	12,00	11,00	11,00	11,00	11,00	11,00
	1235,68	1013,16	1214,54	1194,73	1199,63	1212,95	1214,54
Rl10	11,00	10,00	11,00	10,00	10,00	10,00	11,00
	1080,36	1174,49	1080,36	1182,49	1125,04	1124,40	1080,36
R111	10,00	10,00	10,00	10,00	10,00	10,00	10,00
	1129,88	1119,48	1104,83	1099,46	1108,90	1096,72	1104,83
Rl12	10,00	10,00	10,00	9,00	10,00	10,00	10,00
	953,63	970,87	964,01	1003,73	957,04	962,36	964,01
Moyenne	12,25	12,17	12,17	11,92	12,08	12,08	12,17
	1208,50	1203,03	1209,27	1220,97	1210,14	1210,21	1209,27

Tableau 6 Résultats par problèmes de classe R1 de 100 villes et 1 dépôt de Solomon

No	RT (1995)	CR (1997)	TB (1997)	HG (1999)	CLM (2000)	RGP (2000)	LCK (2000)
C101	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94
C102	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94
C103	10,00 828,06	10,00 828,06	10,00 828,06	10,00 828,94	10,00 828,06	10,00 828,06	10,00 828,06
C104	10,00 824,78	10,00 824,78	10,00 824,78	10,00 828,94	10,00 824,78	10,00 824,78	10,00 824,78
C105	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94
C106	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94
C107	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94
C108	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94
C109	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94	10,00 828,94
Moyenne	10,00 828,38	10,00 828,38	10,00 828,38	10,00 828,94	10,00 828,38	10,00 828,38	10,00 828,38

Tableau 7 Résultats par problèmes de classe C1 de 100 villes et 1 dépôt de Solomon

No	RT (1995)	CR (1997)	TB (1997)	HG (1999)	CLM (2000)	RGP (2000)	LCK (2000)
RC101	15,00	15,00	14,00	14,00	14,00	14,00	14,00
	1623,58	1642,82	1696,94	1697,43	1708,80	1696,94	1696,94
RC102	13,00	13,00	12,00	12,00	12,00	12,00	12,00
	1477,54	1540,97	1554,75	1558,07	1558,07	1554,74	1554,75
RC103	11,00	11,00	11,00	11,00	11,00	11,00	11,00
	1262,02	1302,73	1264,27	1263,09	1262,98	1270,71	1264,27
RC104	10,00	10,00	10,00	10,00	10,00	10,00	10,00
	1135,83	1155,07	1135,83	1138,57	1135,48	1140,58	1135,83
RC105	13,00	13,00	13,00	13,00	13,00	13,00	13,00
	1733,56	1735,84	1643,38	1637,15	1644,43	1633,72	1643,38
RC106	12,00	12,00	11,00	11,00	11,00	12,00	11,00
	1384,92	1395,37	1448,26	1432,12	1427,13	1384,03	1448,26
RC107	11,00	11,00	11,00	11,00	11,00	11,00	11,00
	1230,95	1235,28	1230,54	1232,26	1231,53	1232,26	1230,54
RC108	10,00	10,00	10,00	10,00	10,00	10,00	10,00
	1170,70	1171,40	1139,82	1147,26	1149,79	1149,26	1139,82
	11,88	11,88	11,50	11,50	11,50	11,63	11,50
Moyenne	1377,39	1397,44	1389,22	1388,24	1389,78	1382,78	1389,22

Tableau 8 Résultats par problèmes de classe RC1 de 100 villes et 1 dépôt de Solomon

No	RT (1995)	CR (1997)	TB (1997)	HG (1999)	CLM (2000)	RGP (2000)	LCK (2000)
R201	4,00 1281,58	4,00 1272,70	4,00 1254,80	4,00 1252,37	4,00 1253,26	4,00 1252,37	4,00 1254,80
R202	4,00 1088,07	3,00 1264,44	3,00 1214,28	3,00 1198,45	3,00 1197,66	3,00 1191,70	3,00 1214,28
R203	3,00 948,74	3,00 950,08	3,00 951,59	3,00 942,64	3,00 945,55	3,00 954,58	3,00 942,64
R204	2,00 869,29	2,00 855,21	2,00 941,76	2,00 854,88	2,00 849,62	3,00 751,07	2,00 855,21
R205	3,00 1063,24	3,00 1035,60	3,00 1038,72	3,00 1013,47	3,00 1008,52	3,00 994,42	4,00 1038,72
R206	3,00 912,97	3,00 917,59	3,00 932,47	3,00 913,68	3,00 913,18	3,00 929,03	3,00 932,47
R207	3,00 814,78	2,00 914,39	3,00 837,20	2,00 971,34	2,00 948,23	3,00 856,36	2,00 837,20
R208	2,00 738,60	2,00 746,85	2,00 748,01	2,00 731,23	2,00 734,85	2,00 746,38	2,00 738,60
R209	3,00 944,64	3,00 935,67	3,00 959,47	3,00 910,55	3,00 916,47	3,00 909,86	3,00 923,96
R210	3,00 967,50	3,00 982,60	3,00 980,90	3,00 955,39	3,00 964,22	3,00 966,43	3,00 963,37
R211	2,00 949,50	2,00 967,32	2,00 923,80	2,00 910,09	2,00 933,75	3,00 799,66	2,00 923,80
Moyenne	2,91 961,72	2,73 985,68	2,82 980,27	2,73 968,55	2,73 969,57	3,00 941,08	2,82 965,91

Tableau 9 Résultats par problèmes de classe R2 de 100 villes et 1 dépôt de Solomon

No	RT (!995)	CR (1997)	TB (1997)	HG (1999)	CLM (2000)	RGP (2000)	LCK (2000)
C201	3,00 591,56	3,00 591,56	3,00 591,56	3,00 591,56	3,00 591,56	3,00 591,56	3,00 591,56
C202	3,00 591,56	3,00 591,56	3,00 591,56	3,00 591,56	3,00 591,56	3,00 591,56	3,00 591,56
C203	3,00 591,17	3,00 591,17	3,00 591,17	3,00 591,17	3,00 591,17	3,00 591,17	3,00 591,17
C204	3,00 590,60	3,00 603,05	3,00 590,60	3,00 590,60	3,00 590,60	3,00 590,60	3,00 590,60
C205	3,00 588,88	3,00 588,88	3,00 588,88	3,00 588,88	3,00 588,88	3,00 588,88	3,00 588,88
C206	3,00 588,49	3,00 588,49	3,00 588,49	3,00 588,49	3,00 588,49	3,00 588,49	3,00 588,49
C207	3,00 588,29	3,00 588,29	3,00 588,29	3,00 588,29	3,00 588,29	3,00 588,29	3,00 588,29
C208	3,00 588,32	3,00 588,32	3,00 588,32	3,00 588,32	3,00 588,32	3,00 588,32	3,00 588,32
Moyenne	3,00 589,86	3,00 591,42	3,00 589,86	3,00 589,86	3,00 589,86	3,00 589,86	3,00 589,86

Tableau 10 Résultats par problèmes de classe C2 de 100 villes et 1 dépôt de Solomon

No	RT (1995)	CR (1997)	TB (1997)	HG (1999)	CLM (2000)	RGP (2000)	LCK (2000)
RC201	4,00	4,00	4,00	4,00	4,00	4,00	4,00
	1438,89	1456,33	1413,79	1415,00	1406,94	1406,94	1413,79
RC202	4,00	3,00	4,00	3,00	3,00	4,00	3,00
	1165,57	1532,25	1164,25	1389,57	1407,52	1161,28	1407,52
RC203	3,00	3,00	3,00	3,00	3,00	3,00	3,00
	1079,57	1078,73	1112,55	1060,45	1073,39	1066,99	1112,55
RC204	3,00	3,00	3,00	3,00	3,00	3,00	3,00
	806,75	849,10	831,69	799,12	806,12	801,40	798,46
RC205	4,00	4,00	4,00	4,00	4,00	4,00	4,00
	1333,71	1762,12	1328,21	1302,42	1326,83	1324,51	1328,21
RC206	3,00	3,00	3,00	3,00	3,00	3,00	3,00
	1212,64	1168,12	1158,81	1196,12	1160,91	1153,93	1158,81
RC207	3,00	3,00	3,00	3,00	3,00	3,00	3,00
	1085,61	1133,23	1082,32	1112,60	1062,05	1096,99	1082,32
RC208	3,00	3,00	3,00	3,00	3,00	3,00	3,00
	833,97	872,68	847,90	848,10	832,36	829,69	847,90
	3,38	3,25	3,38	3,25	3,25	3,38	3,25
Moyenne	1119,59	1231,57	1117,44	1140,42	1134,52	1105,22	1143,70

Tableau 11 Résultats par problèmes de classe RC2 de 100 villes et 1 dépôt de Solomon