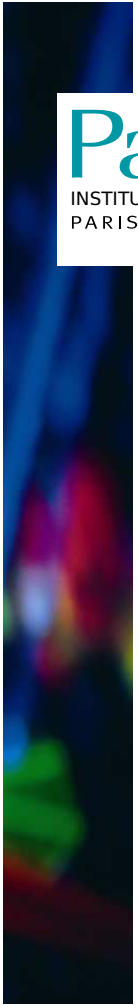
Des métaheuristiques pour le guidage d'un solveur de contraintes dédié à la planification automatisée de véhicules

François Lucas



ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

MINES ParisTech

École doctorale n°432 : Sciences des Métiers de l'Ingénieur

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

l'École nationale supérieure des mines de Paris

Spécialité « Informatique temps-réel, robotique et automatique »

présentée et soutenue publiquement par

François LUCAS

le 12 juillet 2012

Des métaheuristiques pour le guidage d'un solveur de contraintes dédié à la planification automatisée d'itinéraires de véhicules

~ ~ ~

Metaheuristics for the guidance of a constraint solver dedicated to automated vehicle path planning

Directeur de thèse : **Arnaud de La Fortelle**
Co-directeur de thèse : **Patrick Siarry**
Encadrement industriel : **Christophe Guettier**

Jury

Mme Christine Solnon, Professeur, LIRIS (CNRS UMR 5205), INSA Lyon	Rapporteur
M. Nacer M'Sirdi, Professeur, LSIS (CNRS UMR 7296), Polytech Marseille	Rapporteur
M. Arnaud de La Fortelle, Professeur, Centre de robotique, Mines ParisTech	Examinateur
M. Patrick Siarry, Professeur, LiSSi, EA 3956, Université Paris-Est Créteil	Examinateur
M. Christophe Guettier, Ingénieur de Recherche, Sagem Défense Sécurité	Examinateur
M. Jin-Kao Hao, Professeur, LERIA (UPRES EA 2645), Université d'Angers	Examinateur
M. Nicolas Durand, Professeur, MAIAA, École Nationale de l'Aviation Civile	Examinateur

MINES ParisTech
Centre de robotique (CAOR)
60 boulevard Saint Michel, 75272 Paris Cedex 06, France

T H È S E

Remerciements

Je tiens tout d'abord à remercier mes encadrants académiques, Arnaud de La Fortelle et Patrick Siarry, pour le rôle qu'ils ont accepté d'endosser durant ces trois ans. J'ai été touché par leur sympathie, leur culture, leur simplicité, et leurs mots d'encouragement à mon égard.

Je remercie Christophe Guettier, à l'origine de cette thèse, de m'avoir permis de vivre cette expérience. Je retiendrai en exemple son énergie débordante et sa motivation à attaquer de front les nouveaux problèmes.

Je remercie les responsables de la R&T de Sagem, ainsi que les responsables du programme Combat Terrestre, de m'avoir permis d'intégrer les équipes de Massy. Je remercie également mes responsables hiérarchiques successifs : Anne-Marie Milcent, Pascal Rouvière et Pascal Froment.

Je salue l'ensemble de mes collègues de Sagem, avec qui j'ai passé d'excellents moments : Jacques Yelloz, Géraud Allard, Nicolas Brogard, Thomas Dazenière, Jordane Grenier, Georges-Olivier Reymond, Geneviève Sella, Bastien Deloison, Pascal Gaden, Damien Dufresne, Julien Oudot, Didier Levavasseur, Frédéric Révéland, Hugues Berthaud, François Crollet. Une petite pensée également vont aux secrétaires et assistantes : Dominique Chenebault, Annick Nguyen Van Qui et Martine Fouchet, ainsi qu'un clin d'oeil amical au jeune retraité Bernard Françon.

Je souhaite tous mes voeux de réussite aux trois stagiaires avec qui j'ai été amené à travailler : Sutkhemaroath Thou, Christian Joubert et Matthieu Rakotojaona-Rainimangavélo, qui ont chacun fourni un travail de grande qualité.

Je fais un salut général à tous les doctorants du laboratoire CAOR de Mines-ParisTech, du laboratoire LiSSi de l'Université Paris-Est Créteil, et tous à ceux que j'ai pu côtoyer durant mes formations. Qu'ils m'excusent pour l'absence de remerciements nominatifs, je risquerais d'en oublier. Je leur souhaite à tous une carrière passionnante et épanouissante.

Je terminerai cette page en adressant bien entendu mes pensées à ma famille. Je remercie en particulier mes parents pour leur présence et leur soutien. À mon père pour m'avoir inculqué le goût des sciences, la curiosité et la rigueur. À ma mère pour sa valeur du travail, sa simplicité et sa générosité. J'embrasse mes soeurs, Nathalie et Anne, ainsi que leurs conjoints respectifs, Lionel et Yann. Des gros poutous pour mes neveu et nièce, Arthur et Yuna, récemment arrivés en ce monde. J'adresse toute mon affection à mon frère Jean-Philippe. Mes derniers remerciements vont bien entendu à ma compagne Katayoun, dont je suis grandement reconnaissant d'avoir supporté durant trois ans mon engagement dans cette thèse. Je lui réitère tout mon amour et mon attachement, en espérant que les vents nous porteront loin tous les deux.

Table des matières

Introduction

Menée en collaboration avec le laboratoire CAOR de Mines ParisTech, cette thèse CIFRE s'inscrit dans le cadre de travaux menés par Sagem sur la modernisation des systèmes d'information employés par les acteurs opérationnels de l'Armée de Terre. Ces systèmes proposent aujourd'hui un ensemble de *services*, parmi lesquels un service de planification d'itinéraires, dédié à la préparation des missions. Cet outil d'aide à la décision permet à chaque *chef de section*, après avoir défini les objectifs de chaque unité (ou groupe d'unités) dont il a le commandement, de déterminer pour chacune d'elles le meilleur itinéraire à emprunter. Cette étape est réalisée en amont des opérations. Une fois la mission démarrée cependant, s'il est nécessaire de modifier les plans en raison d'un aléa (route barrée, ennemi...), cet outil n'est pas employé car il ne dispose pas de la réactivité nécessaire à une prise de décision en urgence. Pour des problèmes difficiles, plusieurs minutes voire plusieurs dizaines de minutes peuvent en effet être requises pour obtenir une solution optimale.

Aujourd'hui, Sagem souhaite proposer un service de planification capable d'être utilisé en cours de mission. Plus spécifiquement, nous nous intéressons à la planification de véhicules, pour lesquels les exigences de réactivité sont cruciales. En effet, sauf cas particulier, les doctrines militaires interdisent à un véhicule de stationner de manière prolongée lors d'une opération, car il deviendrait une cible facile à atteindre. En cas d'imprévu, le planificateur doit par conséquent être capable de proposer un chemin alternatif dans un délai d'une dizaine de secondes. L'objectif de la thèse est donc de mettre au point une stratégie de résolution permettant au planificateur de satisfaire ces exigences.

L'outil existant a été conçu pour traiter des problèmes consistant à déterminer, pour un ensemble d'entités, un itinéraire allant d'un point courant à un point désiré en un temps minimum. Le caractère générique du solveur permet d'appliquer diverses contraintes sur les itinéraires des véhicules : positions ou routes rendues obligatoires ou interdites, coordinations (contraintes temporelles), contraintes sur la vitesse du véhicule, contraintes dites de *capacité* : énergie disponible, bande passante de communication requise, distances de sécurité aux autres entités... L'approche de résolution développée a été réalisée en Programmation par Contraintes [Fage 96], et a fait l'objet de travaux préalables [Guet 07]. Une stratégie de recherche, basée sur une méthode de *sonde* (*Probe Backtrack Search*) [Sakk 00], permet d'orienter une résolution par séparation-évaluation en triant les variables de décision selon leur distance à la solution d'un problème relaxé, résolu au préalable. Le problème relaxé considéré est un chemin de distance minimale depuis la position courante du véhicule à son objectif final, à l'aide de l'algorithme Dijkstra [Dijk 71]. Si cette stratégie se révèle efficace pour des instances de problèmes simples, ses performances chutent rapidement dès lors que les contraintes appliquées sont fortes. En effet, ces dernières rendent potentiellement le problème NP-difficile, et par conséquent complexe à résoudre.

Les travaux menés durant cette thèse se situent au croisement de trois grands domaines de recherche : l'Intelligence Artificielle, la Recherche Opérationnelle et la Programmation par

Contraintes (PPC). En raison du cadre applicatif de nos travaux, ceux-ci sont basés sur l'approche de résolution existante. Pour limiter le spectre de nos travaux, nous nous sommes restreints à un problème mono-véhicule. Néanmoins, pour une question de généricité, l'approche définie devra être applicable au cas multi-véhicules. Nous traitons également deux types de contraintes en particulier : les points de passage obligatoires, et les contraintes de capacité. Pour ces dernières, nous distinguerons deux cas : les contraintes additives (borne appliquée sur la somme d'un ensemble de variables) et les contraintes non additives (borne appliquée sur chaque variable). Ces problèmes de planification sont proches de plusieurs problèmes connus de la littérature, pour lesquels un grand nombre d'approches de résolution existent : algorithmes de plus courts chemins simples [Dijk 71, Hart 68, Harv 95] et contraints [Hass 92, Jutt 01], métaheuristiques et programmation linéaire pour les problèmes de logistique tels que le voyageur de commerce [Cern 85, Glov 91, Gref 85, Appl] ou les tournées de véhicules [Lapo 92, Gend 01], programmation par contraintes pour les problèmes de plus courts chemins avec contraintes de ressources [Mena 09, Dung 11, Benc 12] et chemins contraints avec points obligatoires [Ques 06]. Bien que ces méthodes soient très performantes pour résoudre les problèmes mentionnés, il n'existe pas, à notre connaissance, de travaux permettant conjointement des problèmes hybrides, tels que la détermination d'un chemin de coût minimum avec points de passage obligatoires et contraintes de capacité pouvant dépendre de la vitesse du véhicule.

Dans ce document, nous présentons une première approche originale consistant à employer une métaheuristique pour le guidage du solveur de contraintes existant. Le but de cette approche originale est d'hybrider plusieurs types de méthodes (métaheuristiques et programmation par contraintes), traditionnellement confrontées, afin de tirer parti des avantages de chacun. Les métaheuristiques ont la capacité de résoudre, en un temps très restreint, des problèmes complexes. En revanche, elles n'offrent aucune garantie d'optimalité quant à la solution retournée. La programmation par contraintes, pour sa part, offre une sémantique évoluée permettant d'exprimer très simplement un problème et propose des méthodes de résolution garantissant la complétude de l'approche. Ses performances sont en revanche très variables selon les problèmes traités. Ici, nous détaillons plus précisément un algorithme de colonies de fourmis (ou ACO pour *Ant Colony Optimization*) [Dori 96] défini pour traiter un problème relaxé, de complexité NP, du problème de planification. Ce problème consiste à déterminer un chemin de coût minimum, comme précédemment, mais prenant cette fois en compte les contraintes de points de passage (équivalent à un chemin hamiltonien). La solution retournée par la métaheuristique est alors employée pour trier les variables du solveur de contraintes. L'hybridation entre colonies de fourmis et programmation par contraintes a déjà été expérimentée [Soln 02, Khic 10], avec une approche différente. Dans ces travaux, les auteurs emploient ACO en tant que méthode d'exploration au sein d'une résolution de programmation par contraintes. L'apprentissage réalisé est alors utilisé dans un deuxième temps pour guider une recherche classique. Dans notre cas, ACO est également employé en amont, à la manière d'une étape de précalcul. Il est en revanche employé hors du cadre de la programmation par contraintes, et ne traite qu'un sous-problème complexe, permettant potentiellement d'accélérer la résolution partielle. Le guidage du solveur de programmation par contraintes est en outre réalisé grâce à la méthode de sonde, déjà mise en place. Cette méthode permet de guider efficacement la recherche du solveur de contraintes à l'aide d'une solution qui n'est pas nécessairement une solution au problème global (et donc robuste face à une différence dans la modélisation des deux problèmes, partiel et global). Dans ce document, nous analysons l'effet du guidage par métaheuristiques sur les performances du solveur de contraintes face au guidage initial, et l'appliquons sur un problème spécifique de gestion des communications radio (capacités non additives et non linéaires).

Nous présentons également une seconde approche, combinant aux métaheuristiques de l'approche précédente un mécanisme de relaxation lagrangienne emprunté d'un algorithme du domaine des télécommunications [Jutt 01]. Cette méthode de résolution approchée est mise en œuvre pour traiter efficacement la classe des problèmes de planification avec contraintes additives. Elle permet ainsi de garantir, avec un temps de calcul très faible, que la solution partielle utilisée pour le guidage satisfait les contraintes de capacité appliquées — et, par conséquent, mène à une solution globale réalisable. Nous analysons les résultats face à l'approche originelle, sur des instances de problèmes très contraints. Nous montrons en outre qu'il est possible de tirer parti du caractère dynamique des colonies de fourmis pour accélérer la résolution. Un problème spécifique de gestion d'énergie est traité en application. Enfin, nous apportons au cours des différents travaux plusieurs améliorations notables à la modélisation logique du problème en programmation par contraintes.

Nous montrons dans la suite du document que les deux approches permettent de satisfaire les exigences formulées pour une application de l'outil dans un cadre opérationnel. Le chapitre 1 décrit en détail le contexte des travaux, présente et formalise le problème. La modélisation du solveur de contraintes existante y est détaillée, ainsi que plusieurs points d'amélioration que nous avons pu apporter. Le chapitre 2 fait un état de l'art général des méthodes de résolution employées dans le cadre de problématiques liées à la planification d'itinéraires. Le chapitre 3 présente les approches de résolution partielles que nous avons mises au point, pour des problèmes génériques. Les performances, comparées à des approches de référence, sont étudiées et discutées. Dans le chapitre 4, nous appliquons ces approches à des problèmes spécifiques, et présentant un intérêt dans un cadre militaire. Nous y présentons également plusieurs améliorations portées au modèle de contraintes, et pouvant être appliquées dans le cas général. Enfin, dans le chapitre 5, nous présentons des travaux exploratoires sur d'autres modèles logiques de chemins, visant à répondre à des exigences particulières. Nous achevons ce document par une conclusion, et donnons quelques perspectives quant aux nouvelles problématiques pouvant être abordées sur la base de nos travaux.

Chapitre 1

Navigation de véhicules terrestres en environnement hostile

1.1 L'enjeu de la navigation dans le domaine militaire

1.1.1 L'émergence des armées modernes

Les mondes de l'informatique et des télécommunications ont connu une évolution spectaculaire ces vingt dernières années. Nés en grande partie grâce à l'investissement réalisé par les armées des grandes nations, qui y ont vu un intérêt stratégique évident, les principales innovations technologiques de ces domaines ont été réalisées par le secteur de la défense jusqu'à la fin des années soixante. Les industries civiles s'emparaient alors de ces technologies, une fois celles-ci portées dans le domaine public. Depuis, le mouvement s'est inversé et le secteur technologique public mue désormais bien plus rapidement que n'en est capable le monde militaire. Les raisons majeures de cette difficulté de la défense à suivre les évolutions du marché sont les exigences de fiabilité, de sécurité (cryptage des communications par exemple), de robustesse... bien plus strictes dans le domaine militaire que pour un utilisateur lambda. Néanmoins, les possibilités offertes par ces nouvelles technologies sont grandes, et le milieu militaire y voit une opportunité pour faire face aux enjeux des conflits modernes.

Bien que peu encline au changement en raison de son organisation hiérarchique complexe et des doctrines qui la régissent, l'armée française poursuit sa transformation. Comme dans l'ensemble des autres pays développés, celle-ci tente de s'adapter pour profiter des progrès récents dans le domaine des Nouvelles Technologies de l'Information et des Communications (NTIC), en intégrant le concept d'*opérations réseau-centrées* (en anglais *Network-Centric Warfare* ou NCW [Albe 01]). Ce concept, érigé aux Etats-Unis à l'issue de la première guerre du Golfe, bouleverse sensiblement la manière dont les opérations sont menées. Partant du constat que chaque entité engagée sur un théâtre d'opérations réalise des observations et dispose d'informations potentiellement utiles aux autres, il vise la mise en place d'un réseau de communication accessible à tous les acteurs et permettant à chacun de transmettre ses observations. La mutualisation et la synthèse de ces données permettent ainsi aux responsables opérationnels d'améliorer la connaissance globale de la *situation tactique*[1]. Les opérations militaires en coalition[2] peuvent

1. Terme militaire pour désigner les informations du terrain utiles à la tactique des opérations : position, équipements, cibles et objectifs des unités amies ; position et objectifs possibles des unités ennemies ; etc.

2. Ces opérations se résument encore aujourd'hui au simple partage de forces de frappe (en Afghanistan par exemple, les militaires français bénéficient parfois d'appuis aériens américains).

également tirer profit de ce concept : les échanges d'information peuvent être envisagés sur des réseaux inter-alliés pour augmenter les capacités de communication. Les intérêts d'un tel réseau sont multiples : mieux relayer les ordres, éviter les tirs fratricides, pister les ennemis, ou encore préciser les manœuvres en facilitant la communication lors d'opérations conjointes entre unités. Or, ces informations échangées n'apportent aucune plus-value si elles ne sont pas correctement gérées au niveau commandement par un outil de navigation efficient, afin de synthétiser les données et de déterminer les choix tactiques appropriés.

1.1.2 La navigation dans l'Armée de Terre

Le terme de navigation est vague, car il s'applique à plusieurs notions distinctes. Le site Wikipedia propose la formulation suivante :

> La navigation est la science et l'ensemble des techniques qui permettent de :
>
> 1. connaître la position (ses coordonnées) d'un mobile par rapport à un système de référence, ou par rapport à un point fixe déterminé ;
> 2. calculer ou mesurer la route à suivre pour rejoindre un autre point de coordonnées connues ;
> 3. calculer toute autre information relative au déplacement de ce mobile (distances et durées, vitesse de déplacement, heure estimée d'arrivée, etc.).

Cette définition donne une idée de l'enjeu tactique que représente la navigation pour le domaine militaire, et plus particulièrement l'Armée de Terre. La première définition s'applique par exemple aux drones, dont une des problématiques est de déterminer au mieux leur position dans l'espace ainsi que celles d'ennemis potentiels, afin de faire du suivi de cible au sol (utilisation combinée de centrales inertielles et de caméras gyrostabilisées). Le suivi de cible est également un enjeu pour l'artillerie, qui peut être amenée à devoir traiter des menaces mouvantes (emploi de calculateurs balistiques). Les définitions 2 et 3 renvoient notamment à la phase préparative d'une mission, durant laquelle les responsables opérationnels doivent assigner des objectifs à chaque entité engagée sur le terrain. À cet effet, ils doivent déterminer et vérifier au préalable la durée approximative de la mission, l'itinéraire à emprunter, les besoins spécifiques à la réussite de cette mission, etc. C'est à ce type de navigation que nous allons nous intéresser par la suite (c'est-à-dire la recherche d'un *plan de navigation* pour les unités). Nous détaillons ci-après les principaux outils de navigation employés lors de phases opérationnelles, ainsi que les systèmes qui en bénéficient.

1.1.2.1 Systèmes de commandement

Les opérations de combat terrestre sont régies par plusieurs niveaux de commandement. Ceux-ci sont susceptibles de varier selon les situations, mais il existe une structure « standard » de l'Armée de Terre . La figure 1.1 schématise cette structure allant du GTIA (Groupement Tactique Inter-Armes) jusqu'au soldat ou véhicule engagé sur le terrain. Au cours d'une opération, un réseau tactique est déployé et s'étend jusqu'au niveau section (jusqu'à 15 kilomètres d'étendue). Sous ce réseau tactique, il y a un réseau pour les combattants communiquant avec les véhicules et dont la portée est d'un kilomètre environ.

En termes de niveaux de commandement donc, ils sont au nombre de trois :

– le GTIA : relais du commandement inter-armées ou inter-alliés ;

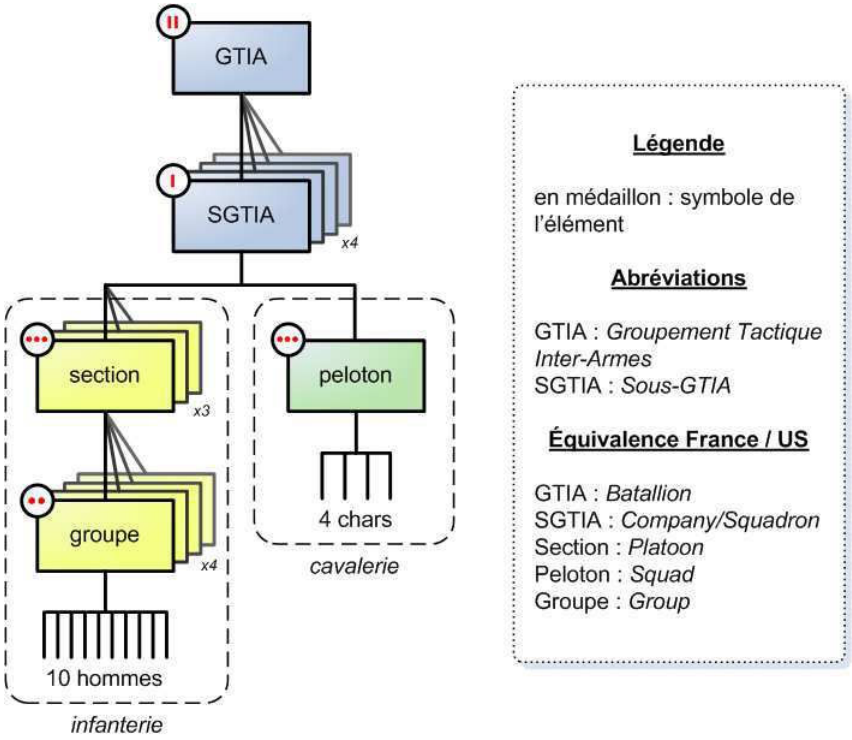


FIGURE 1.1 – Structure type d'un SGTIA.

– le SGTIA : relais du commandement inter-armes ;
– la section / le peloton.

Le commandement des opérations se fait par chaîne descendante. Les responsables opérationnels au niveau GTIA déterminent les actions à mener par les SGTIA (sous-GTIA) à l'aide d'un outil de préparation des ordres. Il en résulte des *OPORD* (*OPeration ORDers*), des ordres formatés (contenant la description des ordres, les zones interdites, les objectifs de mission, les zones à atteindre) transmis au niveau de commandement inférieur, les SGTIA. Chacun d'entre eux élabore alors, en fonction des consignes reçues, ses propres OPORD à destination des sections. Enfin, chaque section réitère la même opération en déterminant les ordres à fournir au personnel engagé sur le terrain. Ainsi, la navigation est décidée à tous les niveaux de commandement. Chaque émission d'OPORD implique des décisions concernant le déploiement des unités et leur progression sur le terrain. Les problèmes sont similaires, seule change la granularité des actions : la navigation issue d'un OPORD pour un char émis par le chef de peloton va prendre effet sur un terrain de $1000m \times 1000m$, tandis qu'un OPORD au niveau GTIA va pouvoir couvrir une zone de $20km \times 20km$ (ordres de grandeurs approximatifs).

La chaîne d'information est au contraire montante : les unités font des compte-rendus d'observation au commandement de la section, qui les relaie au SGTIA, ce dernier les transmettant lui-même au GTIA. Une synthèse des informations est réalisée à chaque niveau de commandement. Elle peut amener les responsables opérationnels à modifier au besoin les objectifs, si les données compromettent l'efficacité des opérations. Si tel est le cas, le niveau de commandement concerné émet un FRAGO (*FRAGmentary Order*), qui modifie tout ou partie des objectifs de la mission en cours, et implique de nouvelles décisions sur la navigation. C'est une opération souvent redoutée, car ces décisions doivent être prises rapidement, parfois de manière approximative. Les outils d'aide à la planification de mission font en effet souvent défaut dans ce cas.

1.1.2.2 Véhicules terrestres

Les véhicules qui composent le parc de l'Armée de Terre sont nombreux, et de divers types (les véhicules de logistique et de maintenance ne sont pas cités) :

- le VAB (Véhicule de l'Avant Blindé), véhicule léger conçu pour le transport de troupes sur les zones d'action ; il peut transporter dix personnes en plus de ses deux conducteurs ;
- le VBL (Véhicule Blindé Léger) conçu pour les missions de reconnaissance ou de liaison ; il embarque quatre passagers (conducteur compris) et a des capacités amphibie ;
- le PVP (Petit Véhicule Protégé) polyvalent ; embarquant cinq à dix personnes (conducteur compris) selon les modèles, il est prévu pour diverses missions : surveillance, patrouille / contrôle de zone, liaison, escorte, transport ;
- l'Aravis, un véhicule blindé à roues conçu pour des missions de patrouille, escorte de convoi, ouverture d'itinéraires... il est bien adapté aux missions de sécurisation et de maintien de la paix, notamment en zone urbaine [3] ;
- l'AMX-10 RC, un véhicule blindé à roues avec capacités amphibie prévu pour des missions de reconnaissance. Doté d'un canon 105mm, il dispose de munitions explosives ou antichar ; un véhicule plus léger (et non amphibie) qui équipe l'armée française est l'ERC-90 (Engin de Reconnaissance avec Canon 90mm) ;
- l'AMX-10 P et son remplaçant le VBCI (Véhicule Blindé de Combat d'Infanterie) conçus pour le débarquement des combattants au plus près de l'objectif. Ce dernier est composé d'un équipage de deux personnes et permet le transport de neuf soldats avec leur équipement. Parmi ses variantes est proposé le VPC (Véhicule de Poste de Commandement) ;
- l'AMX-56 ou Char Leclerc, le char d'assaut dernière génération (conçu pour le combat numérisé) ;
- le MPCV (*Mine Protected Clearance Vehicle*) Buffalo prévu pour les missions de déminage.

Une grande variété de véhicules peuvent être ainsi employés en fonction du type de mission. Tous les véhicules impliqués dans une mission militaire doivent suivre des consignes précises de navigation, et nécessitent l'élaboration par le commandement d'un itinéraire planifié à l'avance. Ils sont interdépendants des autres unités engagées, pour des questions de sécurité (consignes de distance et d'intervisibilité).

1.1.2.3 Drones aériens

Les avions sans pilote, aussi appelés drones ou UAV (*Unmanned Aerial Vehicle* en anglais, voir figure 1.2) sont apparus dans l'armée au début des années 1990. Ce sont des avions miniatures radioguidés initialement dédiés à des missions d'observation et de désignation de cibles. A ce titre, ils emportent à leur bord une caméra stabilisée permettant de scruter la zone survolée, ainsi qu'une charge utile [4] spécifique au type de mission pour lesquels ils sont conçus. Aujourd'hui, certains drones sont également prévus pour être équipés de missiles et entrent ainsi dans la catégorie des drones de combat (ou UCAV pour *Unmanned Combat Aerial Vehicle*). Les drones actuellement utilisés dans le monde militaire se classent en trois catégories principales, selon leur durée d'autonomie en vol et leur capacité de charge utile. Dans l'ordre croissant de ces capacités, il existe les drones tactiques (ou TUAV pour *Tactical UAV*), les drones à moyenne altitude (ou MALE pour *Medium Altitude Long Endurance*), et les drones à haute altitude (ou HALE pour *High Altitude Long Endurance*).

3. Source : www.armyrecognition.com

4. ensemble des équipements pouvant être embarqués dans un véhicule, de poids maximal et volume spécifiés par le constructeur.

FIGURE 1.2 – Exemple de drone tactique : le Sperwer de Sagem. *Source Sagem.*

FIGURE 1.3 – Station de contrôle d'un drone Predator (à gauche le pilote du drone, à droite l'opérateur en charge des équipements embarqués). *Source US Air Force.*

Les drones (hormis les micro-drones tactiques) nécessitent une logistique importante, pour le transport et la maintenance notamment. La gestion de leur navigation se fait par le biais d'une *station sol* (GCS pour *Ground Control Station* en anglais), souvent intégrée dans une remorque de camion (voir figure 1.3). La réception des données depuis le drone peut se faire de deux manières différentes : soit à l'aide d'une station de liaison de données (GDT pour *Ground Data Terminal*, voir figure 1.4) pour les drones tactiques, soit directement par liaison satellite.

Les consignes de navigation peuvent notamment prendre en considération les zones d'intérêt pour la surveillance du terrain, l'altitude de vol (compromis entre risque de détection et qualité d'observation) ainsi que l'énergie restante du drone. Cette dernière contrainte influe directement sur l'autonomie du drone, qui est la problématique majeure de ce type de véhicule. Elle limite en effet les capacités opérationnelles et engendre un surcoût logistique, lorsque l'autonomie est faible.

1.1.2.4 Robots du combattant

Les drones terrestres (ou UGV pour *Unmanned Ground Vehicle*) pour l'Armée de Terre sont à un stade bien plus précoce de leur développement, en termes de capacité d'action, que ne le

FIGURE 1.4 – Station de liaison de données pour drone Sperwer. *Source Sagem.*

FIGURE 1.5 – Le PackBot, plateforme robotique déclinée pour de multiples tâches : inspection, déminage, exploration d'environnements hostiles (centrales nucléaires...). *Source iRobot.*

sont les drones aériens. En effet, l'espace de bataille au sol est bien plus complexe, du point de vue de la navigation, que ne l'est l'espace aérien. Il faut notamment gérer le franchissement (par exemple une marche ou un fossé) ou le contournement d'obstacles potentiels. Il faut également traiter les problématiques d'estimation du positionnement du véhicule : à l'inverse des drones aériens où le relief de l'environnement survolé est globalement connu, ce qui peut leur permettre de recaler leur position, il n'existe en général pas de modèle numérique du terrain exploré par les robots terrestres. L'environnement est en outre beaucoup plus dynamique pour un UGV que pour un UAV : un autre véhicule, ou la présence d'un soldat peut contraindre un robot terrestre à modifier son itinéraire, tandis qu'un drone aérien n'a aucune chance de se trouver sur la route d'un autre véhicule, si les procédures de vol sont respectées. Enfin, un robot terrestre est sensible aux pertes de communication : il peut se retrouver privé de données de localisation (perte des signaux GPS aux abords ou à l'intérieur de bâtiments) ou tout simplement de liaison radio avec sa station de contrôle (fréquent lors de la perte de la vue directe).

Les armées modernes n'utilisent encore les robots terrestres que dans quelques missions précises comme le déminage (voir figure 1.5), et de manière non autonome. De nombreuses

FIGURE 1.6 – Reconstruction d'environnement en 3D avec les algorithmes CoreSLAM, du consortium CoreBots (Concours DGA Carotte 2011).

recherches ont cependant été menées depuis une dizaine d'années, aboutissant à de multiples prototypes définis pour des cas d'utilisation différents. Hormis les robots de combat, on leur attribue essentiellement deux rôles :

- les *véhicules fardiers* (aussi couramment appelés *robots mules*) sont dédiés au portage de lourdes charges, afin d'alléger les soldats dans leurs opérations ; ils sont généralement de grande dimension, et l'attention est portée sur leur capacité de franchissement d'obstacles ;
- les *robots d'exploration* urbains sont destinés à analyser la menace éventuelle dans une rue ou un bâtiment. Ils sont souvent de taille réduite, afin d'être facilement transportables et activés si nécessaire.

Dans le cas des robots d'exploration, l'autonomie des véhicules est souvent fournie par l'utilisation de techniques de SLAM (*Simultaneous Localization And Mapping*) [Durr 06] utilisant des nappes laser ou des caméras actives. Ces techniques permettent à la fois au robot de se localiser et de reconstituer l'environnement dans lequel il évolue sous forme d'une carte 2D, voire même 3D (voir figure 1.6).

L'exploration intérieure d'un bâtiment peut être vue comme un environnement dont on a une connaissance partielle à un temps donné. Il est possible de constituer un graphe de cet environnement, dont les zones non explorées constituent des points d'intérêt à traiter durant la mission. Le besoin de replanification s'avère ainsi nécessaire lorsque les informations sur l'environnement sont mises à jour. Le nouvel itinéraire peut avoir à satisfaire des contraintes de temps ou d'énergie, et doit être déterminé avec un faible temps de calcul, afin que le robot reste réactif lors de son exploration.

1.1.2.5 ORTAC, un outil de préparation de mission militaire

ORTAC (pour *Organisation du Réseau Tactique des Acteurs au Contact*) est un outil de planification tactique développé par Sagem. Il permet, lors de la phase de préparation de mission et au niveau d'une section, de définir les ordres de navigation à fournir aux unités engagées. Il prend en compte les données suivantes :

- la position de départ d'un ou plusieurs véhicules (appelé *point d'entrée* de la zone d'action), ainsi qu'une liste de points finaux potentiels pour chacun d'entre eux (appelés *points d'extraction* de la zone) ;
- des objectifs de mission ;
- des zones dangereuses à éviter ;

FIGURE 1.7 – Terminal d'information SITEL embarqué dans un véhicule. *Source Sagem.*

- des coordinations entre véhicules : synchronisations, fenêtres de temps ou disjonctions [5] ;
- des contraintes sur la sécurité, l'énergie ou encore la bande passante de communication des véhicules.

Cet outil est implémenté sur un outil de cartographie, sur lequel un militaire expérimenté élabore en premier lieu un graphe de progression tactique. Une fois les paramètres de la mission enregistrés, ORTAC est invoqué afin de déterminer la planification optimale des unités. Enfin, ces dernières acquièrent les informations de navigation à l'aide de l'équipement SITEL (Systèmes d'Information Terminaux Elémentaires) dont elles sont dotées (voir figure 1.7).

Cet outil peut en outre être utilisé à des fins de prédiction de mouvements ennemis. En prenant en compte la dernière position connue de leurs unités, et en estimant leurs objectifs potentiels, ORTAC peut déterminer ceux qui sont le plus susceptibles d'être visés et donc déterminer pour chacun leur itinéraire vers cet objectif.

Cet outil est aujourd'hui opérationnel et disponible sur les équipements SITEL. Il présente les avantages d'être très complet et de déterminer les itinéraires optimaux avec une granularité de temps de la minute. Il est également possible de replanifier un itinéraire en cours de mission. Cependant, l'évaluation du problème peut nécessiter plusieurs minutes de calcul et pose un problème en termes de réactivité sur le terrain. Il est donc à ce jour privilégié pour la préparation de mission.

5. Contrainte permettant d'éviter les tirs fratricides en interdisant, si un véhicule longe une zone dangereuse, que tout autre véhicule ne se trouve du côté opposé à cette zone.

1.1.3 Une capacité clé : l'adaptation à l'environnement

Dans les faits, une mission militaire se déroule rarement de la manière dont elle a été planifiée, car elle est soumise à de nombreuses contingences. L'environnement n'est en effet pas figé, et certains événements peuvent nuire au bon déroulement d'une mission. Pour une entité terrestre par exemple, de nombreux aléas peuvent survenir :

- présence d'ennemis sur le plan de navigation : personnes armées non détectées auparavant, embuscades ;
- détection d'un engin explosif (IED, ou *Improvised Explosive Device* en anglais) sur le bord d'une route ;
- aléas climatiques : arbre tombé en travers d'une route, pont détruit, relief trop abrupt... rendant un itinéraire non empruntable ;
- problèmes de communication radio ;
- ressources énergétiques insuffisantes ;
- mise à jour d'objectifs.

Si de tels événements surviennent en cours de mission, notamment pour un groupe de véhicules, il est urgent de déterminer un nouveau plan. En effet, les doctrines militaires interdisent (autant que possible) à un véhicule en mission de rester statique, car il devient alors très vulnérable. La capacité à replanifier une mission est ainsi une fonction décisionnelle majeure dans le déroulement des opérations militaires. Les systèmes de préparation de missions (utilisés pour la planification initiale) font cependant souvent défaut, en raison de leur manque de rapidité pour déterminer la solution optimale, car les problèmes traités sont complexes. Cette fonction est donc, encore aujourd'hui, réalisée manuellement par les responsables opérationnels. Cette réalité constitue une limite opérationnelle sérieuse, pouvant induire de nombreuses approximations.

La recherche de réactivité dans les outils de planification n'est pas propre au domaine militaire. Elle fait l'objet de travaux de recherche dans de nombreux domaines civils, comme par exemple :

- les transports (routage de véhicules), dont les plans de livraisons peuvent être bousculés par de multiples aléas au cours d'une journée : trafic, disponibilité d'un client, retards... ;
- la planification de production, dont le but est d'optimiser la gestion de ressources matérielles et humaines, afin d'améliorer la productivité (utilisation de Progiciels de Gestion Intégrés, ou *Enterprise Resource Planners* en anglais) ;
- la gestion de satellites d'observation sensibles aux conditions météorologiques terrestres (couverture nuageuse sur une zone à photographier).

Dans la suite de ce chapitre, nous allons nous attacher au problème « cœur » de la planification de véhicules militaires en mission, très proche des problèmes traités par l'outil de préparation de mission ORTAC. La modélisation présentée ci-dessous sera utilisée dans le reste du document pour tenter de résoudre efficacement le problème, afin de se rapprocher au mieux des exigences de réactivité qu'un système de replanification comporte.

1.2 Description du problème

1.2.1 Présentation informelle

Dans ce problème, nous traitons plus précisément de la planification de véhicules terrestres non robotisés. Dans une opération militaire réelle, les véhicules se déplacent par groupes de quatre ou cinq. Ce groupe est dirigé par un véhicule en particulier (dans lequel le chef de section se trouve). C'est à ce dernier que le plan de navigation est attribué, les autres ne faisant que le

suivre. Ainsi, nous supposerons qu'un véhicule dans notre problème correspond à un groupe de véhicules dans la réalité.

Le terrain de la zone d'opérations est représenté sous forme d'un graphe, parfois appelé graphe de *praticabilité*. Ce graphe est déterminé lors de la phase de préparation de mission par des militaires expérimentés. Les nœuds y représentent des points géographiques, par exemple un croisement de routes, et sont définis par des coordonnées de latitude et de longitude, ainsi qu'un nom explicite et un identifiant unique. Les arêtes, chacune définie par un nœud de départ et un nœud de destination, représentent les axes de progression supposés franchissables par les véhicules pour naviguer d'une position connue à une autre. Pour pouvoir attribuer des paramètres différents selon le sens où l'arête est emprunté, le graphe est défini comme directionnel. Par hypothèse cependant, si un arc (a, b) existe, un arc (b, a) existe nécessairement.

Un véhicule est caractérisé par une position initiale (ou position courante). Il lui est attribué une mission qui se compose d'une position finale (en général le point d'extraction de la zone) ainsi qu'une liste d'objectifs, qui peuvent être interprétés comme des positions que le véhicule doit obligatoirement visiter. Ces objectifs peuvent exprimer diverses opérations à effectuer : observations, appui feu, embarquement/débarquement de soldats... Toutes les positions mentionnées doivent être définies comme des nœuds du graphe. Dans la suite du document, nous nous référerons souvent à ces objectifs sous le nom générique de *points de passage obligatoires*.

Le problème consiste à déterminer pour ce véhicule le plan de navigation optimal, c'est-à-dire le chemin dans le graphe permettant de *minimiser le temps total de la mission*. Cet chemin devra relier la position courante à la position finale après avoir parcouru l'ensemble des objectifs. Pour déterminer ce plan, les données de distance des arcs sont calculées et associées au graphe. De plus, la vitesse maximale du véhicule est connue : on peut donc déterminer le temps nécessaire au véhicule pour parcourir un arc, en supposant que la vitesse de ce dernier est constante sur cette arc. Le temps total de la mission est calculé comme la somme des temps de parcours des arcs qui composent le chemin.

D'autres contraintes peuvent également être ajoutées : des nœuds ou arcs dont le passage est interdit (pour signifier un danger), ou encore diverses contraintes appelées de manière générique *contraintes de capacité*. Ces contraintes peuvent être additives (variables associées aux nœuds ou arcs du chemin dont la somme des valeurs ne doit pas dépasser une limite fixée), ou exclusives (variables associées aux nœuds ou arcs du chemin dont chaque valeur doit être supérieure ou inférieure à une limite fixée). Plusieurs exemples de contraintes de capacité seront considérés dans la suite du document, notamment une contrainte d'énergie (contrainte de type additive, voir section 4.1) et une contrainte de communication radio (contrainte de type additive, voir section 4.2).

1.2.2 Formalisation générique du problème

Soit un graphe de praticabilité $G = (V, E)$ où V et E désignent respectivement l'ensemble des nœuds et l'ensemble des arcs du graphe G. $v_s \in V$ et $v_g \in V \setminus \{v_s\}$ sont les positions de départ et d'arrivée du véhicule, respectivement. $V_m \subseteq V \setminus \{v_s, v_g\}$ représente l'ensemble des points de passage obligatoires à parcourir dans G. Soit une fonction de distance $d : E \to \mathbb{N}$ donnée en entrée du problème. Formellement, le problème d'optimisation décrit ci-avant consiste à trouver le chemin $p^\star$ tel que :

$$p^\star = \min_{p \in P(v_s, v_g, V_m)} \sum_{e \in p} t(e) \tag{1.1}$$

où $P(v_s, v_g, V_m)$ représente l'ensemble des chemins de v_s à v_g dans G passant par l'ensemble des nœuds de V_m. $e \in p$ est une écriture simplifiée désignant les arcs e du chemin p, qui est un sous-graphe de G. $t(e)$ est une variable de temps associée à e, définie comme :

$$\forall e \in p,\ t(e) = \frac{d(e)}{s(e)} \tag{1.2}$$

où $s(e) \in \{0, S_{max}\}$ est la vitesse du véhicule le long de l'arc e. La vitesse maximale S_{max} du véhicule est fournie en entrée du problème.

Une solution au problème de planification ci-dessus est un chemin p avec une valeur de temps discrète associée à chaque arc $e \in p$. Dans la formalisation décrite ici, les variables de temps $t(e)$ n'étant pas contraintes, les valeurs de temps optimales sont obtenues pour $s(e) = S_{max}$. Cependant, l'application d'un modèle de capacité pourra induire des contraintes sur ces variables, et rendre le problème plus difficile.

1.3 Complexité et problèmes proches de la littérature

1.3.1 Exemple et méthode empirique

Le problème décrit ci-avant paraît simple au premier abord. Il est pourtant complexe, et la *Méthode d'Élaboration d'une Décision Opérationnelle* (MEDO) enseignée en formation militaire n'est d'aucun secours pour l'officier en charge de l'établissement des ordres. Cette méthode est un processus exhaustif de réflexion logique préalable aux décisions, mais n'offre en aucun cas un moyen de résolution de problèmes d'optimisation. En observant l'exemple simple de la figure 1.8, pour lequel quatre solutions possibles (ou non, en fonction des contraintes) sont proposées, on s'aperçoit qu'il peut être difficile de juger de la qualité d'une solution en visionnant simplement un graphe ou une carte. Cette difficulté devient plus grande encore si l'on tient compte de contraintes additives qui vont influencer la vitesse du véhicule (par exemple une contrainte de consommation), et par conséquent faire que le chemin le plus court (en distance) ne sera pas forcément le plus rapide. Dans notre exemple, parmi les quatre exemples proposés, la solution (b) semble visuellement la meilleure en termes de distance. Cependant, peut-être que le nœud précédant G dans le chemin est situé en haut d'une colline, ce qui va augmenter la consommation du véhicule. Si en complément les réserves du véhicule sont très limitées, ce dernier devra ainsi réduire sa vitesse sur le reste de l'itinéraire afin de ne pas tomber en panne. Le temps de la mission sera peut-être au final plus important que s'il avait pris un autre itinéraire (par exemple le (c)). Nous n'avons représenté que quatre solutions dans cet exemple, mais il en existe bien entendu un grand nombre d'autres, ce nombre croissant rapidement avec la taille du problème. Pour un opérationnel chargé d'établir rapidement de nouveaux ordres en cours de mission, cet exercice représente donc une difficulté majeure.

Ci-dessous, nous montrons que le problème étudié peut être rapproché de deux problèmes de la littérature connus pour leur complexité. Ces problèmes sont NP-difficiles (voir section 2.1). En d'autres termes, les temps de résolution croissent potentiellement de manière exponentielle avec la taille du problème.

1.3.2 Couvrir un ensemble de nœuds dans un graphe

Les problèmes de couverture de nœuds dans un graphe sont des « classiques » en théorie des graphes, qui n'ont pas tous la même complexité.

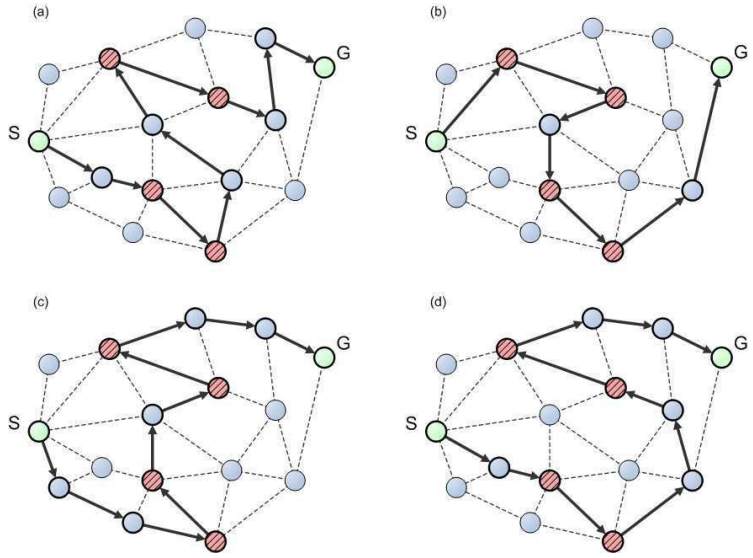


FIGURE 1.8 – Exemples de solutions pour un problème de planification avec quatre objectifs. Les nœuds hachurés sont les objectifs ; les nœuds S et G sont les points de départ et d'arrivée respectivement.

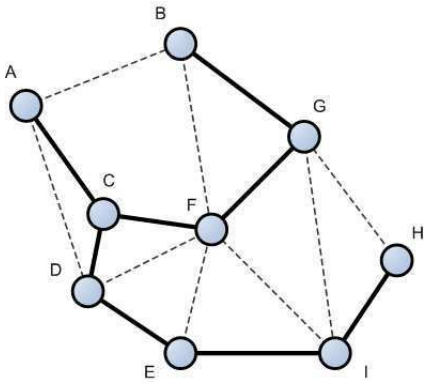


FIGURE 1.9 – Exemple d'arbre couvrant de longueur minimale.

1.3.2.1 L'arbre couvrant de poids minimal

Ce problème fait partie des problèmes enseignés très tôt aux étudiants qui sont initiés à la théorie des graphes. Considérant un graphe $G = (V, E)$ où V et E représentent respectivement l'ensemble des nœuds et l'ensemble des arêtes de G, il consiste à trouver un arbre couvrant de V dans G (c'est-à-dire un graphe partiel connexe $G' = (V, E')$ de G pour lequel le nombre de sommets $|V| = |E'| + 1$, ce qui exclut tout cycle) et dont le poids est minimal (le poids de l'arbre étant égal à la somme des poids des arêtes qu'il possède). La figure 1.9 illustre par un exemple la solution à un problème d'arbre couvrant minimal.

Ce problème se résout en temps polynômial, et possède une complexité en pratique de $\Theta(|E| \log |V|)$. Cependant, il ne s'applique pas à la définition du problème de nos travaux car, dans notre cas, il ne s'agit pas de couvrir l'ensemble des nœuds du graphe, mais un sous-ensemble d'entre eux seulement.

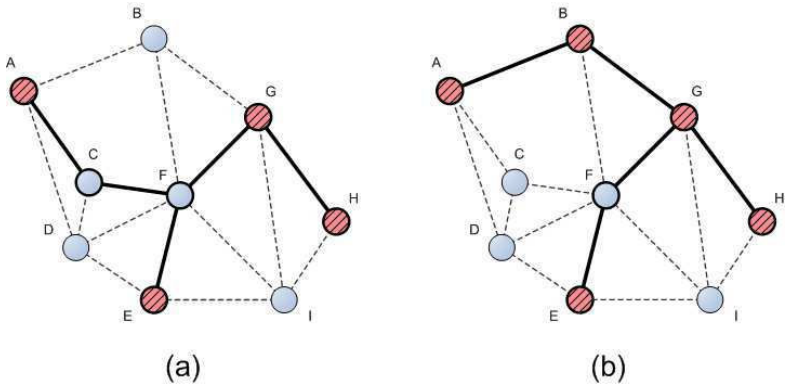


FIGURE 1.10 – Exemples d'arbre de Steiner de longueur minimale. Les nœuds hachurés représentent les nœuds V_m à couvrir obligatoirement ; les autres nœuds $\{V \setminus V_m\}$ désignent ceux pouvant être empruntés de manière optionnelle. (a) Problème avec quatre nœuds obligatoires. (b) Problème avec cinq nœuds obligatoires.

1.3.2.2 L'arbre de Steiner

Le problème de l'arbre de Steiner minimal dans un graphe (en référence à Jakob Steiner, mathématicien suisse du XIX^{e} siècle) est proche de la définition du problème de l'arbre couvrant de poids minimal. Il s'agit de trouver, pour un graphe pondéré non orienté $G = (V, E)$ et un ensemble de nœuds $V_m \subseteq V$, un arbre de poids minimal couvrant de V_m dans G (c'est-à-dire un sous-graphe partiel connexe $G' = (V', E')$ de G où $V_m \subseteq V' \subseteq V$ et $|V'| = |E'| + 1$). La figure 1.10 donne deux exemples d'arbre de Steiner minimal.

Ce problème est plus complexe que le précédent. En effet, cette définition amène à évaluer un bien plus grand nombre de solutions, car il faut considérer les arbres pour toutes les combinaisons de V' possibles, et non plus pour un ensemble V de taille fixe. Il a été établi que sa complexité est NP-difficile, et il figure de ce fait dans la liste du *Compendium NP* [Gare 79, Cres 05] sous le nom de *Minimum Steiner Tree*. Celui-ci est à bien distinguer du problème de couverture minimale de nœuds appelé *Minimum Vertex Cover*, un problème de partitionnement qui consiste à trouver le sous-ensemble V' de V de cardinalité minimale tel que, pour toute arête $e = (u, v) \in E$, au moins u ou v appartient à V'.

En faisant abstraction des contraintes de capacité, on peut donc ramener le problème présenté à la section 1.2 en une recherche d'arbre de Steiner, où G est le graphe de la zone d'opérations et V_m contient la liste des objectifs assignés au véhicule, ainsi que le point initial et le point final. A partir de l'arbre issu de la résolution du problème, il est possible de reconstruire un chemin partant du point initial au point final. Dans l'exemple de la figure 1.10.a, en prenant le nœud A comme point initial et H comme point final, nous pouvons reconstruire le chemin suivant : $[A, C, F, E, F, G, H]$. Dans cet exemple, le nœud F a trois voisins. En développant le chemin depuis A, il faut donc veiller à visiter E avant d'emprunter le chemin allant de E à H. Il est utile de noter que si un nœud de l'arbre possède plus de trois voisins (ou si le point initial ou final a au moins trois voisins), plusieurs combinaisons dans l'ordre de visite des nœuds sont possibles. On obtient alors plusieurs chemins (de taille équivalente) en solution de notre problème. A noter qu'il peut être nécessaire de faire quelques modifications dans la définition du problème de l'arbre de Steiner si l'on ne souhaite pas que le véhicule puisse repasser par le point initial ou final. Dans l'exemple de la figure 1.10.b, si A est le point initial et G le point final, aucun chemin ne peut être trouvé. Il faut donc adapter la définition initiale du problème,

établie par le Compendium NP de la manière suivante :

- INSTANCE : un graphe $G = (V, E)$, une métrique donnée par des poids $s : E \to \mathbb{N}$ associés aux arêtes et un ensemble $S \subseteq V$ de nœuds requis ;
- SOLUTION : un arbre de Steiner, c'est-à-dire un sous-arbre $G' = (V', E')$ de G avec $V' \supseteq S$;
- MESURE : la somme des poids dans le sous-arbre.

en ajoutant la contrainte suivante : $|E'_s| = |E'_g| = 1$ avec $E_s = \{e = (u, v) \in E' \mid u = v_s \vee v = v_s\}$ et $E_g = \{e = (u, v) \in E' \mid u = v_g \vee v = v_g\}$, où v_s est la position initiale du véhicule et v_g est la position finale.

1.3.2.3 Le problème du voyageur de commerce (TSP)

Ce problème, que nous désignerons par l'acronyme TSP (pour *Traveling Salesman Problem* en anglais) dans la suite du document, est peut-être le plus connu parmi ceux de la liste du Compendium NP. C'est un cas d'étude très utilisé dans l'enseignement de la notion de complexité en théorie des graphes, car il possède un algorithme de complexité factorielle très simple à expliciter.

Le but est de trouver, pour un ensemble C de « villes », un *tour* de C de longueur minimale, c'est-à-dire un circuit passant une et une seule fois par chaque ville de C (aussi appelé circuit hamiltonien). La longueur du tour est définie comme la somme des distances entre deux villes consécutives, la matrice des distances étant fournie en entrée du problème.

Comme dans le cas précédent, en ne considérant pas les contraintes de capacité, il est possible de rapprocher le problème décrit en 1.2 d'un problème de voyageur de commerce. En effet, il s'agit de trouver non pas un circuit, mais un chemin hamiltonien de longueur minimale, dont les extrémités sont le point initial et le point final. Voici la définition du problème du voyageur de commerce, donnée dans le Compendium NP, répertorié sous le nom de *Minimum Traveling Salesperson* [Cres 05] :

- INSTANCE : un ensemble C de m villes, les distances $d(c_i, c_j) \in \mathbb{N}$ pour chaque couple de villes $c_i, c_j \in C$;
- SOLUTION : un tour de C, c'est-à-dire une permutation $\pi : [1..m] \to [1..m]$;
- MESURE : la longueur du tour, c'est-à-dire $d(\{c_{\pi(m)}, c_{\pi(1)}\}) + \sum_{i=1}^{m} d(\{c_{\pi(i)}, c_{\pi(i+1)}\})$.

Il est possible d'adapter cette définition à notre problème en la modifiant légèrement, de la manière suivante :

- INSTANCE : un ensemble C de m villes, une ville de départ c_s, une ville de destination c_g, les distances $d(c_i, c_j) \in \mathbb{N}$ pour chaque couple de villes $c_i, c_j \in C \cup c_s, c_g$;
- SOLUTION : un chemin de c_s à c_g couvrant C, c'est-à-dire une permutation $\pi : [c_s, 1..m-1] \to [2..m, c_g]$;
- MESURE : la longueur du chemin, c'est-à-dire $d(\{c_s, c_{\pi(1)}\}) + \sum_{i=1}^{m-1} d(\{c_{\pi(i)}, c_{\pi(i+1)}\}) + d(\{c_{\pi(m)}, c_g\})$.

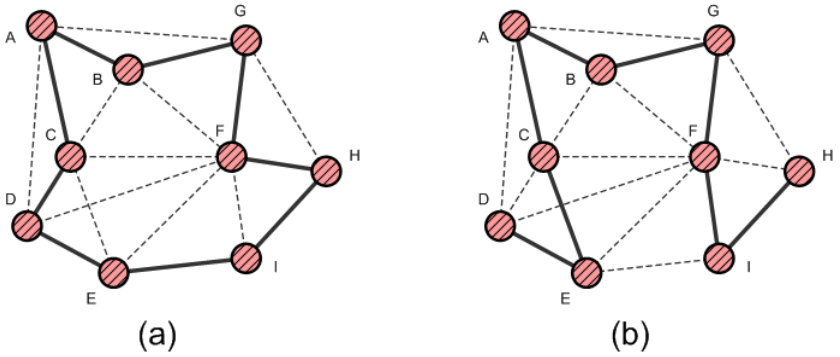


FIGURE 1.11 – Illustration d'une solution au problème de Voyageur de Commerce. Il est important de noter que dans cette illustration, à la différence de celle du problème de l'arbre de Steiner, tous les points sont obligatoires. Les autres nœuds du graphe n'apparaissent pas mais sont implicitement présents (les arêtes représentant les plus courts chemins entre points obligatoires). (a) Résolution du problème classique. (b) Résolution du problème de chemin hamiltonien (et non d'un circuit) avec D comme point de départ et H comme point d'arrivée.

où C, c_s et c_g sont respectivement la liste des objectifs, la position initiale et la position finale du véhicule. Il est à noter que, dans le problème de planification traité, les distances $d(c_i, c_j)$ ne sont pas connues initialement. Il est donc nécessaire de calculer, pour chaque couple de points objectifs, la distance du plus court chemin dans le graphe. D'après la nouvelle définition ci-dessus, la structure du problème reste fortement similaire et le caractère NP-difficile du problème est inchangé. La figure 1.11 illustre par un exemple la solution de chacun des problèmes définis ci-dessus. L'exemple donné ne fait pas apparaître un graphe complet (comme la définition le suggère) car il est possible d'exclure certains liens si l'on sait que les plus courts chemins passeront par d'autres points obligatoires.

1.3.3 Minimiser un plus court chemin sous contraintes

Trouver un plus court chemin dans un graphe en assurant le respect de contraintes additives fait également partie des problématiques combinatoires les plus connues de la littérature, notamment pour leur application directe dans la gestion de réseaux de télécommunication (à côté par exemple des problèmes de chemins disjoints). Ce problème porte en français le nom de plus court chemin avec/sous contraintes, ou encore de routage avec qualité de service. Les noms sont nombreux en anglais : *shortest weight-constrained path*, *constrained shortest path*, *restricted shortest path*, *QoS Routing* (*QoS : Quality of Service*)... en raison des terminologies distinctes ayant été adoptées par les communautés scientifiques travaillant sur des domaines d'application différents. Ce problème est NP-difficile ; la définition donnée dans le *Compendium NP* est la suivante :

- INSTANCE : un graphe $G = (V, E)$, une fonction de longueur $l : E \to \mathbb{N}$, une fonction de poids $w : E \to \mathbb{N}$, des nœuds spécifiés $s, t \in V$ et un entier W ;
- SOLUTION : un chemin dans G de poids total d'au plus W, c'est-à-dire une séquence de nœuds distincts $[s = v_1, v_2, ..., v_m = t]$ telle que, pour tout $1 \leq i \leq m-1$, $(v_i, v_{i+1}) \in E$ et $\sum_{i=1}^{m-1} w(v_i, v_{i+1}) \leq W$;

– MESURE : la longueur du chemin, c'est-à-dire $\sum_{i=1}^{m-1} l(v_i, v_{i+1})$.

Dans le problème décrit en 1.2, en faisant abstraction des points objectifs, nous retrouvons aisément la définition du problème ci-dessus où s et t sont respectivement le point initial et final du véhicule et où l et w sont respectivement les fonctions de distance et de capacité associées aux arcs du graphe.

Nous n'avons mentionné ci-avant que les contraintes additives. Il est bon de préciser que, pour les contraintes multiplicatives, il suffit de transposer les valeurs de ces contraintes sous une forme logarithmique pour se ramener à un problème de contraintes additives. Quant aux contraintes exclusives, que nous avions mentionnées dans la section 1.2, il suffit d'exclure du graphe les arcs dont les valeurs ne sont pas conformes aux bornes spécifiées pour la recherche d'un plus court chemin. Ces dernières ne sont donc pas un facteur de complexité supplémentaire pour le problème, et peuvent même simplifier ce dernier en excluant l'évaluation de certaines solutions.

1.4 Modélisation du problème

1.4.1 Hypothèses de départ

1. En comparaison avec l'outil ORTAC présenté en section 1.1.2.5, nous ne traitons que le cas mono-véhicule. Le cas multi-véhicules pourra être traité en appliquant parallèlement, pour chaque véhicule, l'approche de résolution mise en place pour ce problème. Seule subsistera la gestion des synchronisations éventuelles entre véhicules, à ajouter en aval des résolutions. Bien entendu, ces dernières accroissent la complexité du problème.
2. Dans la modélisation de l'outil ORTAC, sur laquelle nous nous baserons, un véhicule n'est pas autorisé à passer plus d'une fois par un nœud ou une arête le long d'un itinéraire. Cette interdiction traduit une contrainte opérationnelle qui est qu'un véhicule en mission doit éviter d'emprunter plusieurs fois une route le même jour, afin de minimiser les risques d'embuscade.
3. Le graphe modélisant l'environnement est supposé connexe. De plus, il est orienté afin de pouvoir appliquer des contraintes de capacité de valeur différente selon le sens de la route empruntée. Néanmoins, si un arc pour aller du nœud u au nœud v existe, alors il existe nécessairement un arc antagoniste allant de v à u.
4. Comme mentionné plus tôt, la vitesse du véhicule est supposée constante le long d'un arc.

1.4.2 Modélisation du problème et de la fonction de coût

Le modèle présenté dans ci-après permet, en Programmation par Contraintes et en Programmation Linéaire en Nombres Entiers, de définir le problème défini en section 1.2. Une grande partie de ce modèle était existant, et constitue la base du solveur de contraintes de l'outil de planification ORTAC que nous avons repris. Nous détaillons les quelques évolutions que nous y avons apportées.

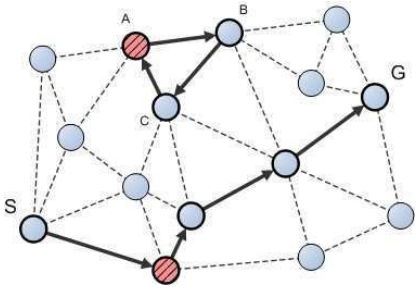


FIGURE 1.12 – Illustration d'une solution satisfaisant les équations de flot 1.3 à 1.6 mais ne constituant pas un chemin.

1.4.3 Modélisation d'un chemin

Un chemin p est modélisé à l'aide d'un modèle de flots, basé sur les équations de Kirchhoff. Pour cela, on associe à chaque arc e de G une variable $\phi(e) \in \{0, 1\}$ dont la valeur vaudra 1 si l'arc appartient à p, et 0 sinon. En définissant les ensembles $\omega^+(v)$ et $\omega^-(v)$ comme l'ensemble des arcs entrants et sortants du nœud v respectivement, on établit que :

$$\sum_{e \in \omega^+(v_s)} \phi(e) = \sum_{e \in \omega^-(v_g)} \phi(e) = 0 \tag{1.3}$$

$$\sum_{e \in \omega^-(v_s)} \phi(e) = \sum_{e \in \omega^+(v_g)} \phi(e) = 1 \tag{1.4}$$

$$\forall v \in V \setminus \{v_s, v_g\}, \sum_{e \in \omega^+(v)} \phi(e) = \sum_{e \in \omega^-(v)} \phi(e) \leq 1 \tag{1.5}$$

$$\forall v \in V_m, \sum_{e \in \omega^+(v)} \phi(e) = \sum_{e \in \omega^-(v)} \phi(e) > 0 \tag{1.6}$$

Selon les équations 1.3 et 1.4, v_s est défini comme une *source* de flot de valeur 1, et v_g est défini comme un *puit* de flot de valeur 1. Pour tous les autres nœuds $v \in G$, par le biais de l'équation 1.5, on vérifie que la somme des flots entrant dans v équivaut à la somme des flots quittant v, et est unique. En outre, l'équation 1.6 contraint ce modèle de flot à passer par chacun des points $v_m \in V_m$ en fixant la valeur des flots entrants et sortants de v_m à 1.

Il est à noter que ces trois équations ne forment pas des conditions suffisantes pour que la solution soit un chemin. La figure 1.12 illustre un contre-exemple pour lequel chacune des contraintes précédemment établies est respectée, mais où le résultat n'est pas un chemin valide. En effet, les contraintes ci-dessus ne garantissent pas l'absence de cycles (même entre deux nœuds) hors d'un chemin. Pour exclure toute présence de cycle, le modèle utilise la *propagation* des valeurs de temps le long des arcs du chemin, que nous présentons ci-après (équations 1.11 et 1.12).

Pour améliorer les performances de propagation des contraintes de flot définies ci-avant, nous y avons adjoint la contrainte supplémentaire :

$$\forall e = (u, v) \in E, e' = (v, u) \in E,\ \phi(e) = 1 \Rightarrow \phi(e') = 0 \tag{1.7}$$

Étant donné que le flot est unitaire, et que l'on suppose qu'un véhicule n'est pas autorisé à repasser par un même nœud, deux arcs antagonistes ne peuvent pas simultanément avoir la valeur 1. Cette équation simple nous a permis, sans effort, de diviser au minimum par quatre les temps de calcul sur l'ensemble des essais réalisés.

1.4.4 Expression de la fonction de coût

La distance $d(e)$ d'un arc $e = (u, v)$ est définie par une fonction euclidienne appliquée entre les deux nœuds u et v, et dont l'unité est le mètre. Le temps $t(e)$ que met le véhicule pour traverser un arc est défini avec l'unité de la seconde.

Le problème d'optimisation consiste à déterminer le chemin $p^\star$, à l'aide des variables de décision $\phi(e)$ du modèle de flots, tel que le temps total $T^\star$ soit minimum. Ainsi, on définit :

$$T^* = min \sum_{e \in E} \phi(e).t(e) \tag{1.8}$$

sous les contraintes

$$\forall e \in E, t(e) \geq 0 \tag{1.9}$$

$$\forall e \in E, \begin{cases} \phi(e) = 1 \Rightarrow t(e) \geq \frac{d(e)}{S_{max}} \\ \phi(e) = 0 \Rightarrow t(e) = 0 \end{cases} \tag{1.10}$$

Les variables $t(e)$ constituent également les variables de décision du problème. Dans la modélisation de l'équation 1.10, nous avons supprimé les variables de temps $s(e)$ initialement insérées dans le modèle logique. En effet, cela induisait des variables de décision supplémentaires alors qu'il suffit d'appliquer une contrainte sur le temps pour contraindre la vitesse du véhicule.

Pour résoudre le problème de la cohérence d'un chemin, et assurer l'absence de tout cycle dans la solution, on associe à chaque nœud un temps correspondant au cumul des valeurs de temps des arcs du chemin depuis v_s. On propage ainsi les variables de temps le long des nœuds du chemin, à l'aide d'une fonction $t : V \to \mathbb{N}$, de la manière suivante :

$$t(v_s) = 0 \tag{1.11}$$

$$\forall v \in V \setminus \{v_s\},\ t(v) = \sum_{e=(u,v) \in \omega^+(v)} \phi(e).(t(u) + t(e)) \geq 0 \tag{1.12}$$

L'équation (1.11) définit un temps $t = 0$ au nœud initial. L'équation (1.12) propage les valeurs de temps en tenant compte des valeurs de flot des arcs entrants (excepté pour v_s). Ces équations permettent dans un premier temps d'appliquer des fenêtres de temps à chaque nœud (nous ne traiterons pas ce cas). Elles permettent en outre de supprimer les cycles en induisant des contraintes de précédence le long du chemin.

Preuve d'absence de cycle : d'après l'équation (1.5), pour tout nœud $v \in p$ (à l'exception de v_s), seule une arête entrante sera de flot égal à 1. De plus, d'après l'équation (1.10), le temps associé à cette arête est strictement positif (on ne considère pas d'arcs de distance nulle dans notre problème). Les temps propagés le long du chemin sont donc strictement positifs et croissants. Dans l'exemple de la figure 1.12, en appliquant l'équation (1.12), on aboutit à :

$$\begin{aligned} t(A) &= t(C) + t((C, A)) \\ t(B) &= t(A) + t((A, B)) \\ t(C) &= t(B) + t((B, C)) \end{aligned}$$

Or par définition :

$$\begin{aligned} t((C, A)) &> 0 \\ t((A, B)) &> 0 \\ t((B, C)) &> 0 \end{aligned}$$

et par conséquent, t étant défini sur $\mathbb{N}$:

$$t(A) > t(C)$$
$$t(B) > t(A)$$
$$t(C) > t(B)$$

On aboutit donc à un système irréalisable. L'application des contraintes de temps de l'équation (1.12) implique un ordre total entre les variables de temps, permettant d'exclure les solutions indésirables telles que celle illustrée par la figure 1.12, quelle que soit la longueur du cycle.

1.4.5 Contraintes de capacité

Nous présentons ici un modèle générique de contraintes de capacité. $w \in \mathbb{N}$ définissant la valeur de capacité associée à un arc e, et $W \in \mathbb{N}$, avec $W > 0$, est la valeur maximale autorisée pour la contrainte de capacité, on exprime une contrainte de capacité additive de la manière suivante :

$$\sum_{e \in E} w(e).\phi(e) \oplus W \tag{1.13}$$

avec $\oplus = \{\leq, <, \geq, >\}$. Ainsi, la somme des valeurs de capacité le long du chemin doit respecter une borne inférieure ou supérieure. Pour que le modèle soit cohérent, on s'assure que $w(e) = 0$ si $\phi e = 0$. Dans le cas de capacités non additives, on vérifiera que chaque arc du chemin respecte les contraintes imposées :

$$\forall e \in E, \phi(e) = 1 \rightarrow \ w(e) \oplus W \tag{1.14}$$

$w(e)$ n'est pas ici exprimé car il dépend du choix de la métrique de capacité définie, qui devra être explicitée pour chaque problème spécifique. Celle-ci pourra être fonction de diverses données du problème (vitesse du véhicule, type de terrain, distance à un objet spécifié...). Une capacité additive pourra servir, par exemple, à modéliser une quantité maximale de carburant disponible pour un véhicule dans le problème présenté en section 4.1). $w(e)$ représentera alors l'énergie consommée le long de l'arc e. Une capacité non additive pourra être employée dans le cas où chaque arc du chemin doit satisfaire des exigences particulières, comme un bande passante minimale dans le problème de communication radio traité en section 4.2. Dans ce cas, $w(e)$ modélisera la bande passante disponible le long d'un arc.

Pour le cas des capacités additives, de la même manière qu'avec les équations 1.11 et 1.12, il est possible de propager les valeurs le long du chemin afin de connaître la capacité « consommée » à chaque nœud, depuis le point initial de l'itinéraire.

1.5 Conclusion

Dans ce chapitre, nous avons présenté le contexte industriel de nos travaux. Nous avons défini de manière générale le problème de planification rencontré, exprimé ce dernier formellement, et nous avons fixé les limites de nos travaux (hypothèses de travail). Nous avons exprimé la NP-complexité du sujet traité en faisant un parallèle avec plusieurs problèmes célèbres de la littérature. Enfin, nous avons enfin présenté la modélisation employée dans les travaux existants et proposé quelques modifications afin d'améliorer les performances de la résolution. Dans le chapitre qui suit, nous établissons un état de l'art des méthodes de résolution pouvant être employées dans le domaine général de la planification d'itinéraires.

Chapitre 2

Etat de l'art des méthodes de résolution pour la planification d'itinéraires

Ce chapitre propose un état de l'art dans le domaine de la planification d'itinéraires. Celle-ci ne doit pas être confondue avec la planification dite de mission (en robotique et dans le domaine spatial essentiellement) où l'objectif est de trouver un *plan de mission*, c'est-à-dire un ordre partiel d'action optimal d'une série de tâches, potentiellement concurrentes, afin de satisfaire au mieux des objectifs fixés.

La planification d'itinéraires est présente dans des domaines variés tels que :

- la logistique : par exemple, les routes devant emprunter une flotte de camions de livraison en tenant compte de contraintes spécifiques [Pill 11] ;
- les télécommunications : par exemple, les algorithmes au sein des protocoles de réseaux sans fil [Jacq 01] pour transmettre des données d'un nœud à un autre rapidement, sans pour autant saturer le réseau ;
- les jeux vidéo [Buli 10] : par exemple, les déplacements d'un joueur dans un monde virtuel calculés sur une grille, en respectant les contraintes de fluidité du jeu.

Face à ces problématiques, diverses méthodes de résolution existent. Nous présentons, dans la suite du chapitre, celles qui sont le plus en lien avec le problème traité dans la thèse. Nous présentons tout d'abord, en section 2.2, les différents algorithmes de résolution de problèmes de plus court chemin (sans puis avec contraintes). Dans un second temps, nous présentons les approches exactes de Programmation Linéaire en Nombres Entiers (section 2.3) et de Programmation Logique avec Contraintes (section 2.4). Enfin, nous présentons en section 2.5 les différentes approches métaheuristiques appliquées aux problèmes combinatoires, et leur application à la résolution du problème de voyageur de commerce. Auparavant, nous faisons un bref rappel sur les classes de complexité permettant de qualifier la difficulté des problèmes rencontrés.

2.1 Classes de complexité

La *théorie de la complexité* est le domaine de recherche qui classifie les *problèmes de décision* en fonction de leur difficulté intrinsèque (c'est-à-dire de la quantité des ressources, en temps et en mémoire, nécessaire pour résoudre ce problème). Un problème de décision, associé à des paramètres d'entrée, est un problème dont la réponse est *oui* ou *non*. S'il est impossible de

résoudre ce problème, sans restriction de temps ou de mémoire, celui-ci est dit *indécidable*. Sinon, il entre dans au moins une des classes décrites ci-après (nous ne présentons ici que les classes relatives à la complexité en temps).

La classe P est la classe des problèmes de décision pouvant être résolus en temps polynômial par rapport à la taille des paramètres (autrement dit en temps $O(n^k)$ avec n la taille de la donnée et k un entier) par une machine déterministe (au sens de la *machine de Turing* [Turi 95]). La classe NP est la classe des problèmes de décision pouvant être résolus en temps polynômial par une machine non déterministe (et $P \subseteq NP$). Également, les classes EXPTIME et NEXPTIME sont les classes des problèmes de décision pouvant être résolus en temps exponentiel par une machine déterministe et non déterministe respectivement.

Par définition, pour une classe de complexité C, un problème de décision est dit *C-difficile* si ce problème est au moins aussi difficile que tous les problèmes dans C. De plus, un problème C-difficile appartenant à C est dit *C-complet*. Pour prouver qu'un problème Π est C difficile, on montre généralement qu'un problème C-complet connu peut se *réduire* à Π, c'est-à-dire être transformé en un problème de la forme de Π (on peut également montrer que tout problème de C se réduit à Π). Pour les problèmes de la classe NP par exemple, on peut utiliser la liste des 21 problèmes NP-complets de Karp [Karp 72] (il faut se restreindre aux réductions en temps polynômial pour NP).

Tout problème d'optimisation peut être transformé en un problème de décision équivalent. Par exemple, le problème de décision qui se rapporte au problème du voyageur de commerce est celui consistant à savoir s'il existe un chemin hamiltonien parcourant un ensemble de villes tel que sa distance soit inférieure à une valeur d. Ce dernier étant NP-complet, on qualifie par extension le voyageur de commerce de problème NP-difficile, bien qu'il s'agisse d'un abus de langage.

2.2 Algorithmes de plus court chemin

Nous présentons ici les algorithmes de résolution de problèmes de chemin de coût minimum. La dernière section présente les méthodes pour les problèmes avec capacité. Cependant, ces approches ne sont pas capables de résoudre un problème avec points de passage.

Le problème consistant à trouver un plus court chemin d'un point A à un point B dans un graphe ou un quadrillage, potentiellement dynamique ou seulement partiellement connu, a fait l'objet de nombreuses recherches. Il en résulte bon nombre d'algorithmes, dont nous présentons ici les plus connus. Ce problème est de classe P (voir section précédente). Lorsque la dimension d'un problème de planification est faible (comme c'est le cas pour les itinéraires de véhicules, en comparaison par exemple de la planification de mouvement d'un bras articulé avec de multiples degrés de liberté), les approches déterministes restent les plus efficaces pour sa résolution [Ferg 05]. La dernière partie de cette section présente les approches de résolution pour le problème de plus court chemin sous contraintes, qui est NP-difficile.

2.2.1 Algorithmes de base

Pour résoudre ce problème, une première approche consiste à employer des méthodes systématiques. Aussi appelées recherches *non informée* (le tableau 2.1 récapitule les principales méthodes et caractéristiques), elles parcourent sans stratégie l'ensemble de l'espace d'état afin de trouver une solution. Cet espace d'état est modélisé par un arbre dans lequel un nœud représente une position courante (le sommet d'un graphe ou la case d'un quadrillage), et les nœuds

Type de recherche		C(Tps)	C(Mem)	CPL	OPT
- en largeur	[Norv 95]	$O(bd)$	$O(b^d)$	OUI	OUI
- par coût uniforme		$O(bd)$	$O(b^d)$	OUI	OUI
- en profondeur		$O(bm)$	$O(bm)$	OUI*	OUI*
- en profondeur limitée		$O(bl)$	$O(bl)$	NON	NON
- itérative en profondeur		$O(bm)$	$O(bm)$	OUI	OUI
- bidirectionnelle		$O(b^{\frac{d}{2}})$	$O(b^{\frac{d}{2}})$	OUI	OUI
- aléatoire [Lang 92]			$O(1)$	NON	NON
- par incohérence [Harv 95, Korf 96, Wals 97]		**	**	OUI	OUI

(*) si le graphe ne comporte pas de cycles ; (**) dépend de l'approche choisie ; b : facteur de branchement (nombre moyen de fils par nœud) ; d : profondeur de la solution ; m : profondeur maximale de l'arbre ; l : profondeur limite définie.

TABLE 2.1 – Liste récapitulative des principales méthodes de parcours dans un arbre. **C(Tps)** : complexité pire cas en temps ; **C(Mem)** : complexité pire cas en mémoire ; **CPL** : complétude (évaluation de toutes les solutions valides) ; **OPT** : optimalité (garantie de trouver l'optimalité).

fils sont les positions accessibles depuis cette position courante. Les méthodes de recherche non informée parcourent cet arbre afin de trouver une solution, et réalisent un *retour arrière* (aussi appelé *retour sur trace*, ou *backtrack* en anglais) lorsqu'elles sont bloquées, afin de modifier un choix précédemment réalisé.

Parmi les méthodes mentionnées, l'approche la plus répandue pour les recherches de plus court chemin est l'algorithme de Dijkstra [Dijk 71] (recherche par coût uniforme). Cet algorithme, de complexité $O(p + n.\log(n))$ pour un graphe de p arcs et n sommets, trouve un plus court chemin en empruntant en priorité les arêtes les plus courtes. Si cet algorithme est très efficace pour calculer les plus courts chemins depuis un sommet vers tous les autres sommets du graphe (calcul d'une matrice de distances par exemple), il est en revanche peu adapté pour déterminer un plus court chemin entre deux points dès lors que la taille du graphe est grande au regard de celle du plus court chemin. Une grande partie de l'espace d'état va être explorée inutilement, car la recherche est menée dans toutes les directions possibles (et non pas orientée vers l'objectif). La figure 2.1 illustre le déroulement d'une telle recherche.

Par conséquent, les grands espaces d'état rendent nécessaire la mise en œuvre de stratégies de recherches. Celles-ci consistent à utiliser les informations du problème (recherche *heuristique*) pour orienter les recherches vers l'objectif qui est notre point d'arrivée. Un algorithme efficace va trouver la solution optimale en restreignant au mieux l'espace de recherche.

2.2.2 Les algorithmes avec heuristique

Souvent, en planification d'itinéraires, les seules informations disponibles sont les coordonnées du point de départ et du point d'arrivée (et éventuellement la matrice des distances). Dès lors, l'idée est de les utiliser pour orienter la recherche. Les premiers algorithmes qui ont vu le jour sont des algorithmes de type *best-first search* (BFS). Ces algorithmes vont explorer l'espace d'état en sélectionnant en priorité les nœuds situés le plus près de l'objectif, selon une heuristique estimant la distance à ce dernier. Ils sont très efficaces pour des environnements « dégagés », en l'absence de pièges comme des impasses. Si la recherche se trouve bloquée cependant, aucune règle ne va pouvoir guider les mécanismes de retour arrière, pour explorer les autres solutions locales.

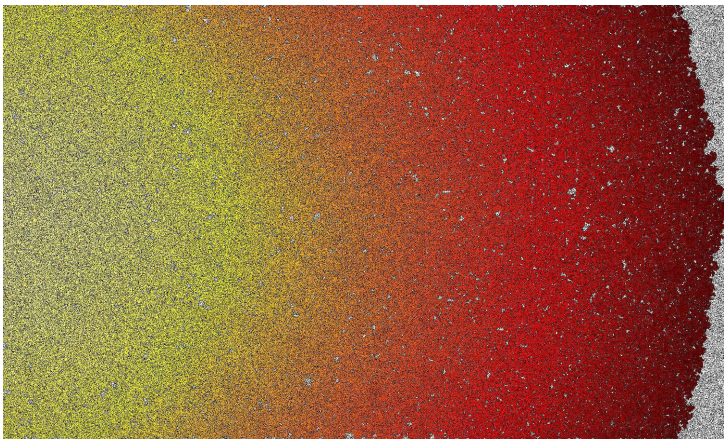


FIGURE 2.1 – Illustration de l'expansion d'une recherche par coût uniforme sur un grillage 4-connecté. Les cases en noir sont des cases interdites, les cases en blanc sont les cases non explorées par l'algorithme. Les cases en jaune sont celles qui ont été explorées tôt dans la recherche, tandis que les cases en rouge ont été visitées tardivement. *Source : [Thay].*

2.2.3 L'algorithme A*

Pour contrer cet inconvénient, l'algorithme A* a vu le jour. Inventé par Hart, Nilsson et Raphael [Hart 68] en 1968, A* peut être considéré comme une combinaison de l'algorithme de Dijkstra et d'un algorithme *best-first*. Il établit pour chaque nœud une fonction de coût, qui est la somme du coût réel du chemin depuis le départ et de l'estimation du coût pour atteindre l'objectif.

Formellement, l'algorithme A* met en place deux listes : une *liste ouverte* et une *liste fermée*. La première liste contient les nœuds situés à la frontière de la recherche, et candidats à l'expansion de celle-ci. La seconde contient les nœuds qui ont déjà été sélectionnés (cette liste n'est pas nécessaire dans le cas particulier d'un arbre).

Initialement, les deux listes sont vides. L'algorithme entame la recherche au point de départ et commence par placer les nœuds voisins dans la liste ouverte. Les candidats sont sélectionnés dans un ordre de type *best-first*, c'est-à-dire ceux dont la valeur de la fonction suivante est la plus faible en premier :

$$f(n) = g(n) + h(n)$$

$g(n)$ est le coût du plus court chemin depuis l'origine de la recherche jusqu'au nœud n, et $h(n)$ est l'estimation heuristique de $h^*(n)$ qui est le plus court chemin à l'objectif. Une fois le nœud sélectionné, il est retiré de la liste ouverte et placé dans la liste fermée. On met alors la liste ouverte à jour en inscrivant les nouveaux nœuds candidats : si un nœud était déjà inscrit, on compare les valeurs et on retient la plus faible. Une fois la recherche terminée, le chemin est reconstruit avec la liste fermée depuis l'objectif jusqu'au point de départ.

L'algorithme A* est optimal [Dech 83]. Si l'heuristique utilisée ne surévalue jamais le coût de la distance, il est également *admissible*, c'est-à-dire que la première solution trouvée est la solution optimale. La meilleure heuristique est une heuristique qui se rapproche le plus de la valeur réelle du chemin, sans toutefois être supérieure : $h(n) \leq h^*(n)$ (où $h^*(n)$ représente le coût réel minimal pour aller de n à l'objectif). En planification d'itinéraires, on prend souvent la distance euclidienne entre le point évalué et l'objectif.

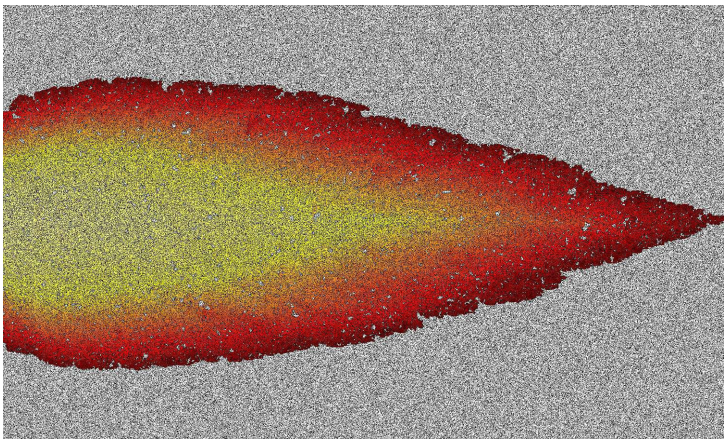


FIGURE 2.2 – Illustration de l'expansion d'une recherche A* sur un grillage 4-connecté. Les cases en noir sont des cases interdites, les cases en blanc sont les cases non explorées par l'algorithme. Les cases en jaune sont celles qui ont été explorées tôt dans la recherche, tandis que les cases en rouge ont été visitées tardivement. *Source : [Thay].*

Cet algorithme est le plus répandu pour les problèmes de recherche de chemin entre un sommet origine et un sommet destination. Il permet de réduire fortement le nombre d'évaluations menées lors d'une recherche (voir figure 2.2, en comparaison d'une recherche par coût uniforme présentée en figure 2.1). Une implémentation utilisant une structure de tas binaire pour la liste ouverte assure de très bonnes performances à l'approche. Un autre algorithme appelé AO* est très proche, en terme de fonctionnement, de l'algorithme A*. Il n'utilise cependant pas de liste fermée mais reconstruit un graphe ET/OU à partir des variables explorées.

2.2.3.1 Algorithmes A* à caractère *anytime*

A* a la capacité d'être admissible, ce qui est un avantage. Cependant, dans le cas où plusieurs chemins vont être intéressants, il va tous les considérer parallèlement jusqu'à ce que l'évaluation détermine le meilleur. Pour des cas complexes, l'algorithme mettra peut-être trop de temps pour trouver la solution optimale, tandis qu'une bonne solution sous-optimale suffirait.

La solution à ce problème est donnée par l'algorithme WA* (pour *Weighted A**) qui apparaît dès 1970 [Pohl 70a] [Pohl 70b] : afin d'augmenter la *directivité* de l'algorithme, on va surpondérer l'heuristique dans la fonction d'évaluation des nœuds :

$$f(n) = (1 - w).g(n) + w.h(n)$$

où w est une valeur réelle entre 0 et 1. Si $h(n)$ ne surestime pas la distance réelle, WA* reste admissible tant que $w \leq 0,5$. Mais l'idée est justement de paramétrer w supérieur à $0,5$, afin que les nœuds les plus proches de l'objectif soient favorisés. Ainsi, si deux chemins sont envisagés pour mener à l'objectif, le comportement WA* sera d'emprunter celui qui semble le plus prometteur puis d'aller jusqu'à l'objectif, tandis que A* évaluera le second si c'est nécessaire. Une illustration du comportement de l'algorithme est présentée en figure 2.3, pour différentes valeurs de w.

Le second intérêt de cette variante est son caractère *anytime* (capacité d'un algorithme à mettre à disposition une solution à tout moment, en assurant la convergence vers l'optimalité), très enviable pour une exécution en temps incertain. En effet, comme le souligne [Hans 07] avec Anytime A*, il n'y a aucune raison d'arrêter la recherche une fois la première solution trouvée

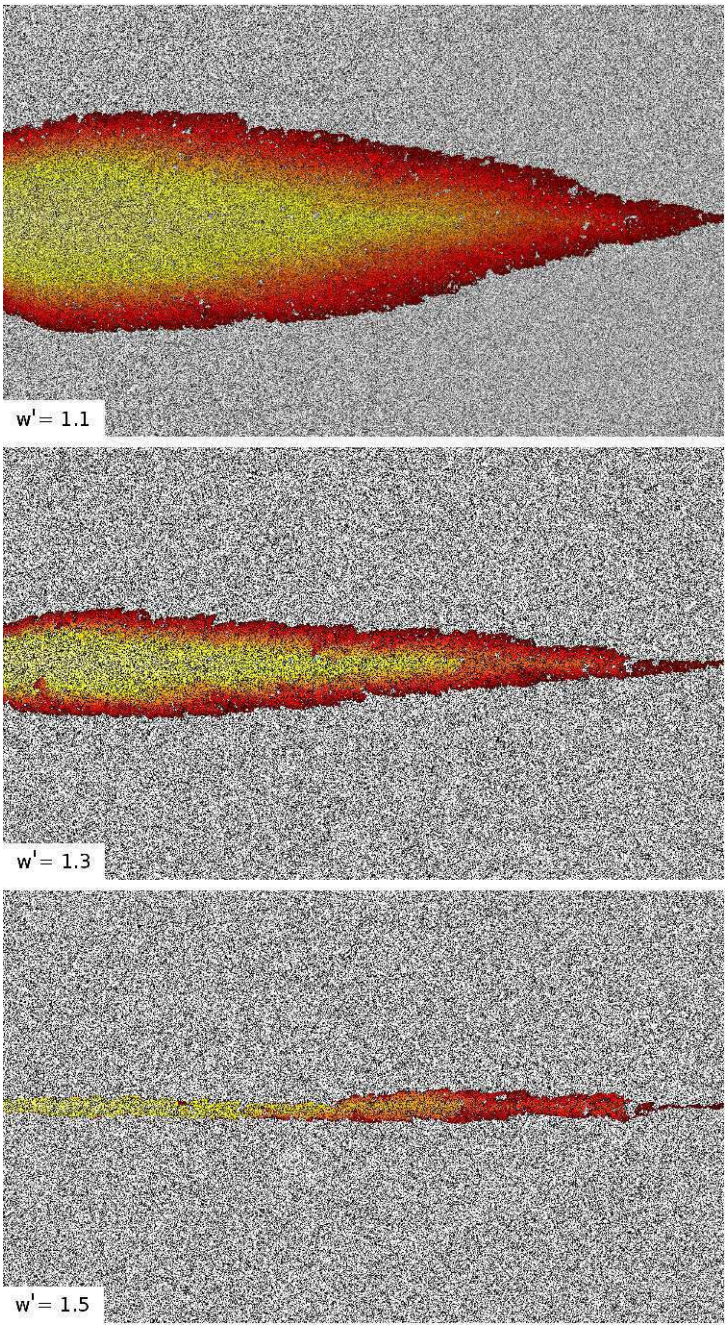


FIGURE 2.3 – Illustration de l'expansion d'une recherche WA* pour trois valeurs $w' = \frac{w}{1-w} = \{1,1; 1,3; 1,5\}$ sur un grillage 4-connecté. Les cases en noir sont des cases interdites, les cases en blanc sont les cases non explorées par l'algorithme. Les cases en jaune sont celles qui ont été explorées tôt dans la recherche, tandis que les cases en rouge ont été visitées tardivement. *Source : [Thay].*

bien que l'algorithme ne soit pas admissible. Le coût de cette solution permet en outre de borner la recherche afin de ne pas explorer tout l'espace. En pratique, Anytime A* évalue plus de nœuds que A*. En revanche, on obtient une première solution plus rapidement. De plus, la convergence est assurée car si la solution optimale n'a pas encore été trouvée, elle est nécessairement située dans l'espace borné par la meilleure solution courante et sera donc évaluée.

2.2.3.2 Algorithmes A* avec restriction d'espace mémoire

Les algorithmes type A* peuvent se révéler gourmands en mémoire. C'est pourquoi plusieurs algorithmes ont été développés pour en limiter la complexité. Les travaux ont été initiés dès 1984/1985 [Berl 84, Korf 85] par des travaux sur les recherches en profondeur croissante (*iterative-deepening* en anglais). Les chercheurs présentent notamment IDA* (*Iterative-Deepening* A*) et ses améliorations [Rein 94], qui effectuent une série de recherches dont la valeur maximale de la fonction de coût est bornée (basé sur une recherche en profondeur limitée). L'intérêt de cette implémentation est qu'en effectuant une série de recherche en profondeur, la complexité en mémoire est linéaire avec la profondeur de la recherche (on ne mémorise pas le coût et le nœud parent pour chaque nœud visité). L'approche reste optimale, et il est démontré que le nombre de nœuds évalués tend asymptotiquement vers le nombre de nœuds évalués par A*.

D'autres travaux ont porté sur la mise en place de conditions de sauvegarde des points visités pour réduire l'espace mémoire au fil de la recherche. L'idée globale développée avec MA* (*Memory-Bounded* A*) [Chak 89] est de réduire l'allocation mémoire utilisée par la recherche dans des zones peu intéressantes en supprimant les nœuds de la liste ouverte qui ont les valeurs $f(n)$ les plus fortes (on limite la taille de la liste à un facteur donné). L'inconvénient majeur est que cette méthode risque de supprimer une trop grande partie de l'information établie lors de la recherche, et par conséquent difficile à implémenter correctement [Korf 93]. Dans SMA* (*Simplified* MA*) [Russ 92], une amélioration est effectuée pour conserver une partie des résultats : si une branche de l'arbre de recherche a été abandonnée, plutôt que de supprimer simplement les nœuds de la liste, on va d'abord remonter la valeur minimale des nœuds candidats à l'expansion au niveau du nœud parent, pour savoir quelle est au mieux la valeur que l'on pourra obtenir en reprenant la recherche dans cette direction. Les expérimentations menées sur le problème du *15-puzzle* ont montré que SMA* surpassait MA*, IDA* et A* (respectivement, du meilleur au moins bon).

D'autres améliorations ont également été faites par la suite comme l'algorithme WSMA* (*Weighted* SMA*) [Kain 94] qui est une version bidirectionnelle de SMA*.

2.2.3.3 Algorithmes A* pour environnements dynamiques ou incertains

L'origine des travaux portant sur les plus courts chemins en environnement dynamique provient en partie des jeux vidéo. Dans les *role-playing games* notamment, un personnage se déplace dans un environnement 3D dynamique. Pour mouvoir ce personnage, le joueur effectue un clic à un endroit de la carte et le personnage doit s'y rendre en empruntant le chemin le plus court tout en évitant les obstacles et les autres personnages. Le problème réside dans le temps de calcul de ce chemin : il faut qu'il soit quasiment imperceptible pour un joueur afin d'assurer la fluidité de l'exécution et ne pas être agaçant à la longue. La recherche de chemin en robotique mobile a également fait l'objet de nombreux travaux. Les recherches de chemin sont souvent réalisées sur un quadrillage représentant la surface (deux dimensions donc) de l'environnement. Lorsqu'un obstacle se trouve sur une case, cette dernière est « bloquée » et ne peut donc être empruntée.

L'idée majeure pour améliorer les méthodes provient d'une constatation : certaines recherches peuvent n'être que peu différentes des précédentes. En conséquence, pour accélérer le calcul d'un nouveau chemin, on va réutiliser les résultats acquis lors des précédents déplacements.

Dans un premier temps, les chercheurs ont tenté d'adapter directement les algorithmes de recherche optimaux [Nils 80, Jarv 85] pour les porter dans des architectures dynamiques. Les expérimentations sont alors effectuées sur des robots autonomes [Zeli 92] qui vont tenter de rejoindre un point objectif : si une incohérence est détectée, le robot va replanifier son itinéraire en effectuant une nouvelle recherche. Ces méthodes de force brute sont optimales, mais globalement inefficaces, particulièrement pour les environnements de grande taille pour lequels la recherche peut prendre beaucoup de temps. D'autres algorithmes ont donc été élaborés pour améliorer leurs performances en termes de besoins temps réel, à savoir rapidité d'exécution et/ou espace mémoire. Il en résulte deux grandes familles, développées parallèlement dans la seconde moitié des années 80, que nous présentons maintenant.

2.2.3.3.a Les algorithmes à heuristique temps-réel

La première famille d'algorithmes regroupe les recherches dites à heuristique temps-réel. L'idée de départ est qu'une recherche type A* doit attendre la fin de l'exécution pour délivrer la solution optimale, ce qui peut s'avérer dispendieux pour des environnements de grande taille. Or pourquoi attendre la fin de l'exécution pour qu'un robot commence à se déplacer ? Pour accélérer l'exécution, les algorithmes à heuristique temps-réel font le choix de ne considérer qu'un environnement de taille finie, limitée par un horizon ou une profondeur de recherche. Les chercheurs proposent différents algorithmes [Lume 86, Pirz 90] mais le premier faisant vraiment référence est RTA* (pour *Real-Time* A*) [Korf 87, Korf 88], amélioré par la suite : LRTA* (*Learning* RTA*) [Korf 90] avec méthode d'apprentissage, RTAA* (*Real-Time Adaptive* A*) [Koen 06] puis GAA* (*Generalized Adaptive* A*) [Sun 08] implémentant une mise à jour des heuristiques... Ces méthodes ont été largement dérivées et ont démontré leur efficacité sur d'autres types de problèmes [Koen 98], comme le *24-puzzle* ou encore les problèmes de planification de tâches...

Contrairement aux algorithmes à heuristique incrémentale décrits ci-après, le chemin ainsi défini peut se révéler sous-optimal, car l'espace de recherche ne va pas être considéré de manière globale. En revanche, pour des applications temps réel exigeant une forte réactivité ou engageant de nombreux agents (par exemple dans le cas du déplacement d'une « armée » dans un jeu vidéo), ces méthodes présentent une grande efficacité.

2.2.3.3.b Les algorithmes à heuristique incrémentale

Pour faire face à la surcharge de calcul ou d'utilisation mémoire, de nombreux algorithmes ont vu le jour, dont la performance est atteinte au détriment de l'optimalité de la solution. L'algorithme D* (pour *Dynamic* A*) [Sten 94] va s'imposer en 1994 comme la première grande référence face aux problèmes dynamiques. Il sera amélioré pour être dérivé notamment en *Focussed* D* [Sten 95] puis en D* *Lite* [Koen 02a], le dernier permettant une simplification de l'exécution.

Les algorithmes (*Focussed*) D* et D* *Lite* sont applicables à plusieurs cas de figure. On peut les mettre en œuvre dans les cas où l'environnement est totalement inconnu (détection locale à l'aide de capteurs), connu à priori (on dispose de la carte de la zone, mais on doit faire face à l'incertitude : obstacles mobiles, accès condamnés...) ou partiellement connu (zones d'ombre). Ils ont la capacité de réutiliser les données des recherches précédemment réalisées en conservant en mémoire la *liste ouverte* élaborée par une recherche type A*. Plus particulièrement, D* Lite

s'appuie sur l'algorithme LPA* (*Lifelong Planning* A*) [Koen 02b]. LPA* effectue une première recherche à la manière d'A* et attend une mise à jour des informations. Si au cours du déplacement une incohérence (différence entre la fonction de coût attendue lors du déplacement le long de l'itinéraire et celle effectivement détectée) survient, alors un mécanisme de propagation permet d'actualiser les valeurs de coût des nœuds candidats pour la recherche. Récemment, un autre algorithme incrémental plus efficace que LPA* a été mis au point : *Fringe-Saving* A* [Koen 07]. Cet algorithme a fait l'objet, tout comme LPA*, d'une dérivation pour être applicable aux environnements dynamiques. Issu de D* *Lite*, *Dynamic Fringe-Saving* A* [Koen 09] a ainsi été créé. Basé sur un algorithme incrémental plus efficace, il surpasse logiquement son prédécesseur : le gain annoncé est un facteur 2,5 en temps d'exécution.

2.2.4 Algorithmes pour le problème de plus court chemin sous contraintes

Le problème de plus court chemin sous contraintes consiste, comme précédemment, à trouver un chemin pour aller d'un sommet A à un sommet B dans un graphe, mais en prenant cette fois en compte une seconde métrique (autre que le coût) associée aux arêtes du graphe. La résolution doit garantir que la somme des valeurs de cette métrique le long du chemin est inférieure à une valeur plafond (voir la formalisation de ce problème en section 1.3.3). Il est à noter qu'un problème sous contraintes multiplicatives peut être ramené à un problème sous contraintes additives, en travaillant avec la valeur logarithmique des pondérations. Quant au problème de contraintes min/max, il est résolu avec un simple algorithme de plus court chemin une fois les arêtes invalides supprimées.

Pour gérer l'introduction d'une contrainte additive, des approches basées sur des heuristiques intégrées à un algorithme de plus court chemin de type Dijkstra ou Bellman-Ford [Kort 00] (alternative à Dijkstra, moins performante mais capable de traiter des problèmes avec pondérations négatives, dès lors qu'aucun cycle négatif n'est détecté) ont été expérimentées. Les travaux exploitant cette approche [Ravi 02] montrent cependant des performances assez variables en comparaison des méthodes de résolution spécifiques présentées ci-après. Étant démontré par [Wang 96] que le problème est NP-complet (même dans le cas de graphes acycliques), plusieurs approches d'*approximation* ont été développées. Elles sont de deux types : les approches *géométriques* et les approches *algébriques*, présentées ci-dessous. En troisième lieu, nous présentons les approches du problème généralisé à k métriques de contraintes.

2.2.4.1 Approches géométriques

Les approches géométriques de résolution de plus courts chemins sous contraintes sont basées sur une relaxation lagrangienne, comme l'algorithme LARAC [Jutt 01]. LARAC, dont le pseudo-code est donné par l'algorithme 1, s'appuie sur l'approche duale du problème.

c est la fonction de *coût* à minimiser, d est la fonction de *délai* dont la somme des valeurs le long d'un chemin p ne doit pas excéder Δ_{delay}, et $Dijkstra(G, s, t, c)$ est l'algorithme [Dijk 71] qui retourne un chemin de coût c minimal entre les sommets s et t. Au début de l'algorithme LARAC, on calcule un plus court chemin p_c selon la métrique c afin de vérifier si le chemin de coût minimal respecte les contraintes ($d(p_c) \leq \Delta_{delay}$). Si tel est le cas on retourne simplement p_c. Sinon, on calcule un plus court chemin p_d selon la métrique d afin de s'assurer qu'il existe au moins un chemin possible pour aller de s à t en respectant les contraintes. Si ce n'est pas le cas il est inutile de poursuivre la recherche. Sinon, on débute la phase itérative de la recherche. Dans cette phase, on va déclarer une nouvelles fonction de coût $c_\lambda = c + \lambda d$ où le paramètre λd joue le rôle d'une pénalité pour les arêtes dont la valeur de délai est importante. De plus, les

Algorithme 1 Algorithme LARAC [Jutt 01]

Requiert: graphe G, sommet de départ s, sommet d'arrivée t, fonction de coût c, fonction de délai d, borne max de délai Δ_{delay}
Renvoie: un chemin p de coût $c(p)$ minimum tel que $d(p) \leq \Delta_{delay}$

$p_c \leftarrow Dijkstra(G, s, t, c)$
si $d(p_c) \leq \Delta_{delay}$ **alors**
  retourner p_c
fin si
$p_d \leftarrow Dijkstra(G, s, t, d)$
si $d(p_d) > \Delta_{delay}$ **alors**
  retourner "Pas de solution"
fin si
boucle
  $\lambda \leftarrow \frac{c(p_c)-c(p_d)}{d(p_d)-d(p_c)}$
  $r \leftarrow Dijkstra(G, s, t, c_\lambda)$
  si $c_\lambda(r) = c_\lambda(p_c)$ **alors**
    retourner p_d
  sinon si $d(r) \leq \Delta_{delay}$ **alors**
    $p_d \leftarrow r$
  sinon
    $p_c \leftarrow r$
  fin si
fin boucle

chemins p_c et p_d jouent le rôle de bornes inférieure et supérieure à la solution optimale $p^\star$ (voir figure 2.4) : on sait ainsi que nécessairement $c(p_c) < c(p^\star)$ et que $c(p_d) \geq c(p^\star)$. Dans la boucle de l'algorithme LARAC, on va itérativement modifier l'une ou l'autre des bornes en calculant successivement un plus court chemin selon la métrique c_λ, celle-ci étant mise à jour entre deux itérations (recalcul de λ).

Cette méthode a depuis été reprise et généralisée [Xiao 05], pouvant traiter des problèmes incluant plus de deux paramètres. De plus, la mise à jour de λ est réalisée à l'aide d'une recherche binaire. Le pseudo-code de l'approche est donné par l'algorithme 2.

Ici, les bornes supérieure et inférieure de la solution optimale sont définies par λ_{start} et λ_{end}, et la mise à jour de λ est faite de manière dichotomique. τ est un paramètre de réglage défini au préalable, en fonction de la distance désirée de la solution finale à la solution optimale du problème. Les algorithmes LARAC et LARAC-BIN ont démontré leur grande efficacité dans la résolution de problèmes pratiques (voir leurs articles relatifs). Une autre approche plus récente, basée sur une résolution par simplexe (voir section 2.3.1) du problème primal, montre également des performances intéressantes [Xiao 06].

2.2.4.2 Approches algébriques

Les approches algébriques de résolution de plus courts chemins sous contraintes sont basées sur le calcul de schémas d'approximation. Les schémas d'approximation sont un type d'algorithmes d'approximation, c'est-à-dire produisant, pour un problème de minimisation, une solution de coût au maximum $(1 + \epsilon)OPT$ (où OPT est le coût de la solution optimale). Par définition, un tel algorithme est polynômial avec la taille de l'instance mais dépend de $\frac{1}{\epsilon}$:

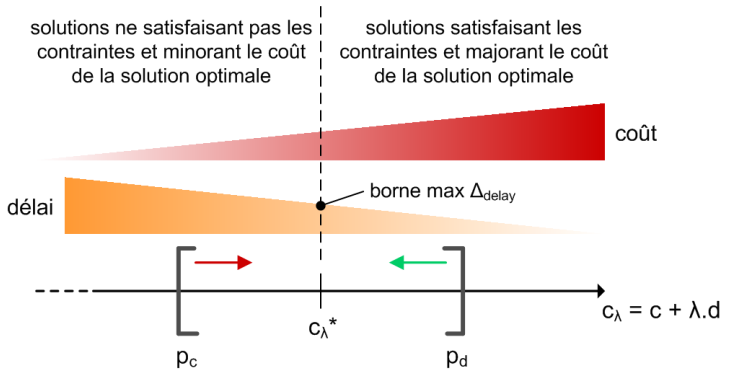


FIGURE 2.4 – Illustration de l'approche du LARAC. Les solutions p_c et p_d jouent le rôle de bornes dans la recherche de la solution optimale, de coût c^*_λ. Il est à noter que les fonctions de coût et de délai ne varient pas nécessairement de manière linéaire, et que la valeur de délai de la solution optimale n'est pas nécessairement Δ_{delay} (elle peut être inférieure).

Algorithme 2 Algorithme LARAC-BIN [Xiao 05]

Requiert: graphe G, sommet de départ s, sommet d'arrivée t, fonction de coût c, fonction de délai d, borne max de délai Δ_{delay}, paramètre d'approximation τ
Renvoie: un chemin p de coût $c(p)$ minimum tel que $d(p) \leq \Delta_{delay}$

$p_c \leftarrow Dijkstra(G, s, t, c)$
si $d(p_c) \leq \Delta_{delay}$ **alors**
  retourner p_c
fin si
$p_d \leftarrow Dijkstra(G, s, t, d)$
si $d(p_d) > \Delta_{delay}$ **alors**
  retourner "Pas de solution"
fin si
si $d(p_d) = \Delta_{delay}$ **alors**
  retourner p_d
fin si
$\lambda_{start} \leftarrow 0, \lambda_{end} \leftarrow \frac{c(p_d)-c(p_c)}{\Delta_{delay}-d(p_d)}$
tant que $(\lambda_{end} - \lambda_{start})(\Delta_{delay} - d(p_d)) > \tau$ **faire**
  $\lambda \leftarrow (\lambda_{start} + \lambda_{end})/2$
  $r \leftarrow Dijkstra(G, s, t, c_\lambda)$
  si $d(r) = \Delta_{delay}$ **alors**
    retourner r
  sinon si $d(r) < \Delta_{delay}$ **alors**
    $\lambda_{end} \leftarrow \lambda$
  sinon
    $\lambda_{start} \leftarrow \lambda$
  fin si
fin tant que
retourner $r \leftarrow Dijkstra(s, t, c_\lambda)$

si sa complexité est potentiellement exponentielle en $\frac{1}{\epsilon}$, alors on parle de *schéma d'approximation polynômial* (ou PTAS pour *Fully-Polynomial Time Approximation Scheme*. Si elle est polynômiale, alors on parle de *schéma d'approximation totalement polynômial* (ou FPTAS pour *Fully-Polynomial Time Approximation Scheme*). Les premiers travaux aboutissant à un FPTAS pour les problèmes de plus courts chemins contraints sont fournis par [Hass 92], et valides dans le cas de graphes acycliques. Cette approche est reprise et améliorée plus tard par [Lore 99], qui présente un FPTAS simple applicable à tout type de graphe. Enfin, une autre amélioration pour les graphes acycliques est proposée par [Ergu 02].

2.2.4.3 Approches pour le problème généralisé

Dans ce problème (qui porte généralement le même nom que le problème précédent, dans lequel une seule contrainte est considérée), il s'agit de considérer le cas général de k contraintes additives (et k valeurs de ressources potentiellement différentes). Plusieurs travaux ont été menés afin de résoudre ce problème. Tout d'abord, un algorithme de programmation dynamique a été présenté par [Blok 95] dans le cas particulier du problème avec deux paramètres. Pour résoudre un problème dans le cas général, les approches de programmation dynamique sont employées très tôt [Joks 66]. Cependant, leur performance se révèle assez mauvaise en pratique dès lors qu'on augmente la taille du problème. Plusieurs améliorations ont été proposées par la suite, utilisant des méthodes d'étiquetage des nœuds (*Node Labeling* en anglais, ou encore *Label Setting*) indépendamment développées [Anej 83, Desr 83, Desr 88, Jaum 96]. Les travaux les plus récents [Dumi 03] présentent de très bonnes performances, même sur grandes instances. Parallèlement, des travaux ont été menés sur d'autres approches, comme une méthode basée sur une relaxation lagrangienne [Hand 80, Carl 03], une méthode linéaire par séparation-évaluation [Beas 89] (voir description de la méthode de séparation-évaluation en section 2.3.4). En programmation par contraintes (voir section 2.4), des méthodes de filtrage spécifiques [Sell 03] ont été développées, ainsi que des contraintes globales pour k=1 [Rgin 99] et dans le cas général [Mena 09] pour des problèmes définis sous forme d'automates finis.

2.3 La Programmation Linéaire en Nombres Entiers (PLNE)

Nous présentons ici l'approche générique de PLNE. Cette approche permet de traiter un ensemble de problèmes dont la fonction de coût est linéaire, de même que les contraintes. Très efficace, cette approche implique néanmoins un effort dans la modélisation précise du problème. De plus, il n'est pas possible d'exprimer de contraintes non linéaires, par exemple pour les capacités.

Certains éléments de cette section ont été tirés de [Dema 03] et [Hama 09].

2.3.1 La Programmation Linéaire (PL)

Un problème de programmation linéaire (PL) est un problème d'optimisation de la forme :

$$\min cx \tag{2.1}$$

sous les contraintes :

$$Ax \leq b \tag{2.2}$$

avec $x \in \mathbb{R}^n_+$, $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{n \times m}$, $b \in \mathbb{R}^m$ où n est le nombre de variables et m est le nombre d'inégalités linéaires posées en contraintes du problème. Le problème décrit est un problème

de minimisation, mais il est possible de le ramener à un problème de maximisation, en posant $\max -cx$. Une solution à ce problème est une instanciation de toutes les variables x qui satisfasse les m contraintes $Ax \leq b$. L'ensemble de ces solutions, appelées *réalisables*, est noté $S = \{x \in \mathbb{R}_+^n \mid Ax \leq b\}$. Les solutions *optimales* sont les solutions $S^\star \subseteq S$ telles que $\forall x \in S, \forall x^\star \in S^\star, cx \geq cx^\star$.

Plusieurs méthodes existent pour résoudre un programme linéaire, la plus célèbre étant la méthode du *simplexe* [Dant 51]. Elle repose sur une interprétation géométrique du problème. L'ensemble des solutions S, s'il est non nul, est un polyèdre convexe (c'est un polytope si les contraintes font de S un ensemble fini) dont la solution optimale se trouve en un sommet (ou éventuellement sur une face du polyèdre). L'algorithme du simplexe se déplace le long des arêtes du polyèdre, partant d'une solution réalisable (sommet de départ), de manière à améliorer successivement la fonction de coût. La convexité de S garantit l'optimalité globale de la solution finale trouvée. D'autres approches telles que la *méthode de l'ellipsoïde* de Khachiyan [Khac 79] puis la *méthode de projection* de Karmarkar [Karm 84] sont pour leur part basées sur des méthodes de *point intérieur* : contrairement au simplexe qui se déplace à la surface de S, celles-ci partent d'une région intérieure de S et améliorent itérativement l'approximation de la solution optimale. Bien que de complexité polynômiale, contrairement au simplexe [Klee 69], ces méthodes restent souvent moins efficaces en pratique.

L'étude du programme linéaire *dual* du problème dit primal 2.1, donné par

$$\{\max ub \mid uA \leq c, u \in \mathbb{R}_+^m\} \tag{2.3}$$

peut en outre permettre de déterminer l'absence de solution au problème si le dual n'est pas borné. Dans le cas contraire, toute solution réalisable du problème dual est une borne inférieure du problème d'optimisation initial. Ces solutions peuvent être utiles pour la définition de coupes (voir section suivante).

2.3.2 Présentation générale de la PLNE

Un grand nombre de problèmes, notamment de Recherche Opérationnelle, peuvent se traduire sous forme de programmes linéaires. Néanmoins, la programmation linéaire exige que toutes les variables soient définies en tant que réels. Certains problèmes ont un ensemble de variables à valeurs discrètes, et se définissent comme suit :

$$\{\min cx \mid Ax \leq b, x \in \mathbb{N}^n\} \tag{2.4}$$

avec généralement A,b et c à valeurs dans $\mathbb{Z}$. Cette classe de problèmes s'appelle la Programmation Linéaire en Nombres Entiers (PLNE, ou encore ILP pour *Integer Linear Programming* en anglais). Au premier abord, ces problèmes semblent très similaires à ceux vus précédemment. En réalité, ils sont nettement plus difficiles à résoudre : les contraintes dites d'*intégrité*, vérifiant que les variables sont bien entières, font des instances de PLNE des problèmes NP-difficiles [Gare 79]. Dans ces problèmes, il n'est plus possible d'appliquer les méthodes de résolution de la PL. Prenons un exemple simple. Pour un chantier de rénovation, un entrepreneur a besoin de 250 kg de plâtre. Un revendeur propose des sacs de 50 kg à 80 euros pièce, et des sacs de 100 kg à 120 euros pièce. Le problème, pour cet entrepreneur, est donc de choisir la quantité de produits de chaque type qui minimise le coût d'achat. Le problème s'écrit donc comme suit :

$$\{\min 80.x_1 + 120.x_2 \mid 50.x_1 + 120.x_2 \geq 250\} \tag{2.5}$$

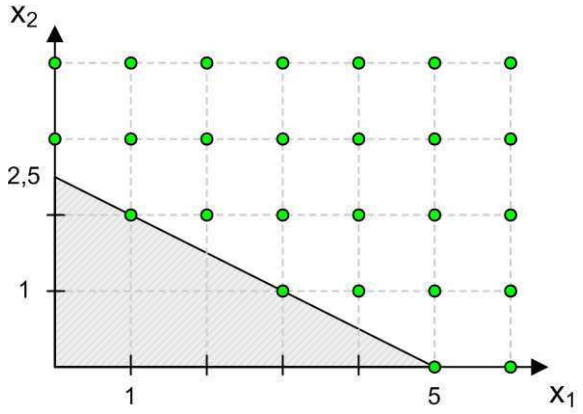


FIGURE 2.5 – Illustration de l'espace des solutions pour le problème de la formule (2.5).

Ce problème est illustré sur la figure 2.5. En utilisant une méthode de la PL, on aboutirait à une solution simple : acheter $x_2 = 2,5$ sacs à 100 kg et $x_1 = 0$ sacs de 50 kg, ce qui fait bien 250 kg de plâtre pour $2,5 \times 120 = 300$ euros. Sur la figure, il s'agit de l'intersection de la droite d'équation $x_2 = 2,5 - 0,5.x_1$ avec l'axe des ordonnées, qui est l'un des sommets du polyèdre de l'ensemble des solutions (ce dernier étant non fermé car on peut dans le pire des cas acheter autant de produits que l'on veut). Or il n'est évidemment pas possible d'acheter la moitié d'un sac, et les contraintes d'intégrité réduisent l'ensemble des solutions du problème aux points indiqués sur la figure. Ces contraintes, si l'on reprend la solution optimale du PL en arrondissant à la valeur supérieure pour satisfaire l'inéquation, impliquent donc d'acheter $x_2 = 3$ sacs de 100 kg pour un coût de $3 \times 120 = 360$ euros. Or la solution optimale à ce problème est bien entendu d'acheter $x_2 = 2$ sacs de 100 kg et $x_1 = 1$ sac de 50 kg pour un coût de $2 \times 120 + 1 \times 80 = 320$ euros. Il ne suffit donc pas d'arrondir la valeur trouvée dans l'approche de PL pour trouver la solution optimale du problème de PLNE.

Pour résoudre un problème de PLNE, il y a deux grands types d'approches communément employées, présentées maintenant. La première ajoute des contraintes afin de réduire l'espace du polyèdre des solutions (sans jamais exclure de solution entière) et garantit que la première solution trouvée est la solution optimale. La seconde sépare le polyèdre initial en plusieurs polyèdres plus petits, à l'aide de contraintes d'intégrité, et applique un algorithme de séparation-évualuation. Cette dernière approche est donc *anytime*.

2.3.3 Les coupes de Gomory

La résolution par *coupes de Gomory* [Gomo 63], également appelée méthode des *plans sécants* (*cutting planes* en anglais) se base sur la forme standard du problème (forme également employée pour la méthode du simplexe), qui introduit des *variables d'écart* afin de transformer les contraintes d'inégalité en contraintes d'égalité. Soit le problème de PLNE suivant :

$$min\ 3x_1 - 4x_2 + x_3 \tag{2.6}$$

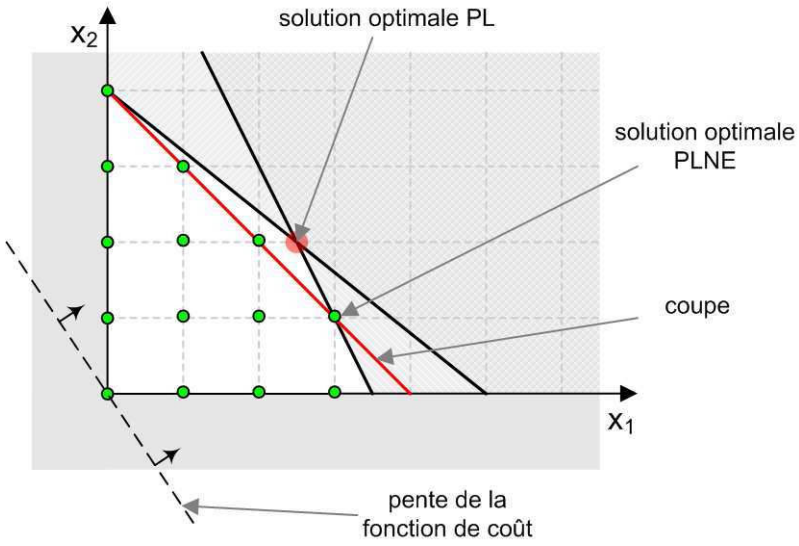


FIGURE 2.6 – Illustration d'un exemple de coupe en PLNE.

sous les contraintes :

$$\begin{aligned} 2x_2 - x_3 &\geq 2 \\ x_1 + x_2 &\leq 5 \\ 2x_1 - x_2 + 4x_3 &= 6 \\ x_1, x_2, x_3 &\geq 0 \end{aligned} \tag{2.7}$$

La forme standard de ce problème est la suivante :

$$min\ 3x_1 - 4x_2 + x_3 \tag{2.8}$$

sous les contraintes :

$$\begin{aligned} 2x_2 - x_3 - x_4 &= 2 \\ x_1 + x_2 + x_5 &= 5 \\ 2x_1 - x_2 + 4x_3 &= 6 \\ x_1, x_2, x_3, x_4, x_5 &\geq 0 \end{aligned} \tag{2.9}$$

Le problème sous forme standard fait apparaître les variables x_4 et x_5 qui viennent compenser les valeurs des autres variables, permettant l'écriture d'une égalité. Dans la méthode de coupes de Gomory, ces variables d'écart sont utilisées pour faire respecter les contraintes d'intégrité en introduisant de nouvelles contraintes au sein du problème. A chaque itération, le problème est résolu en PL par la méthode du simplexe et une nouvelle *coupe* (voir figure 2.6) est réalisée tant que la solution au problème optimal n'est pas entière. La résolution en PL utilisée pour l'évaluation de sous-problèmes est une *relaxation*, puisque l'on ne tient pas compte des contraintes d'intégrité des variables de décision. Dans cette approche, l'ensemble des solutions est maintenu dans un unique polyèdre et les coupes réalisées n'excluent jamais de solution entière potentielle.

2.3.4 La méthode par Séparation-Évaluation

La méthode par *Séparation-Évaluation* (*Branch & Bound* en anglais), tout comme la méthode de coupes de Gomory, fait itérativement appel à une résolution du problème en PL et introduit

de nouvelles contraintes dans ce dernier. Cependant, l'approche est sensiblement différente. On calcule tout d'abord la solution au problème initial en PL, à l'aide par exemple d'une méthode du simplexe. Si la solution optimale n'est pas entière, par exemple si une variable $x_1 = 3/2$, alors on va séparer le problème en deux problèmes distincts (étape de *séparation*) : on évalue d'une part le problème initial avec une nouvelle contrainte $x_1 \leq 1$, et d'autre part ce même problème initial avec la contrainte $x_1 \geq 2$. La figure 2.7 illustre la démarche. C'est une approche de type *diviser pour régner*, dans laquelle on découpe en sous-problèmes tant qu'une solution entière n'est pas trouvée ou que le sous-problème traité est réalisable. Ce processus d'évaluation itératif peut être représenté sous forme d'un graphe binaire (voir figure 2.8) où chaque nœud représente un problème et les éventuels nœuds fils désignent les sous-problèmes après étape de séparation. Les feuilles représentent les solutions réalisables (dans ce cas le coût est calculé) ou les problèmes qui n'ont pas de solution. Les valeurs de coût des solutions trouvées servent ensuite de borne pour les évaluations suivantes : si un sous-problème n'est pas susceptible de posséder de meilleure solution (on calcule pour cela des bornes inférieures pour le coût, en relachant les contraintes d'intégrité par exemple), il n'est pas évalué. On dit alors que l'arbre (de recherche) est *élagué*.

En termes de *stratégie de recherche* — ordre dans lequel les nœuds de l'arbre sont évalués — les nombreuses applications de cette approche ont démontré qu'une évaluation des sous-problèmes en *profondeur d'abord* donne généralement de meilleurs résultats, car elle permet une économie à la fois en mémoire et en calcul (face à un parcours en largeur). Le choix de la variable non entière pour séparer le problème, ainsi que le choix de la priorité dans l'évaluation des sous-problèmes, peuvent être décidés par des heuristiques propres au problème afin d'améliorer le guidage de la résolution.

Des approches de résolution combinant les méthodes de coupes et de séparation-évaluation ont depuis vu le jour. Ainsi est né l'algorithme *Branch & Cut* (que l'on pourrait traduire par *séparation-coupe*, mais le terme anglais est conservé dans la littérature francophone) dont [Mitc 00] fait un état de l'art. La méthode consiste à traiter un problème de PLNE en résolvant tout d'abord une relaxation du problème en PL (de manière identique aux deux approches). Puis, si la solution trouvée n'est pas entière, on réalise des coupes (une ou plusieurs, en évaluant l'évolution des solutions trouvées). Si, à l'issue de cette étape, la nouvelle solution n'est toujours pas satisfaisante, alors on partitionne le problème en sous-problèmes par la méthode de séparation-évaluation. Cette méthode permet de tirer bénéfice de l'efficacité de la méthode de coupes sur certains problèmes, tout en s'affranchissant de ses problèmes de lenteur de convergence. Elle tire également bénéfice du paradigme *diviser pour régner* de la méthode de séparation-évaluation, tout en ayant l'avantage de réduire le nombre de sous-problèmes à considérer. D'autres méthodes, comme le *Branch & Price* [Barn 98], et plus récemment le *Branch, Cut & Price* [Lada 01], utilisant des mécanismes de *génération de colonnes*, se révèlent également efficaces pour les problèmes de PLNE de très grande taille.

Pour le cas particulier du problème de voyageur de commerce, le solveur le plus efficace à ce jour a été développé au sein de l'Institut de Technologie de Géorgie, et se dénomme *Concorde* [Appl]. Il est capable de résoudre de manière optimale des problèmes allant jusqu'à 85900 villes (en utilisant un réseau d'ordinateurs effectuant des calculs en parallèle).

2.4 La programmation logique avec contraintes (PLC)

La PLC, plus communément appelée Programmation par Contraintes, est un outil de modélisation et de résolution étendue de la programmation logique. Sa sémantique permet une ex-

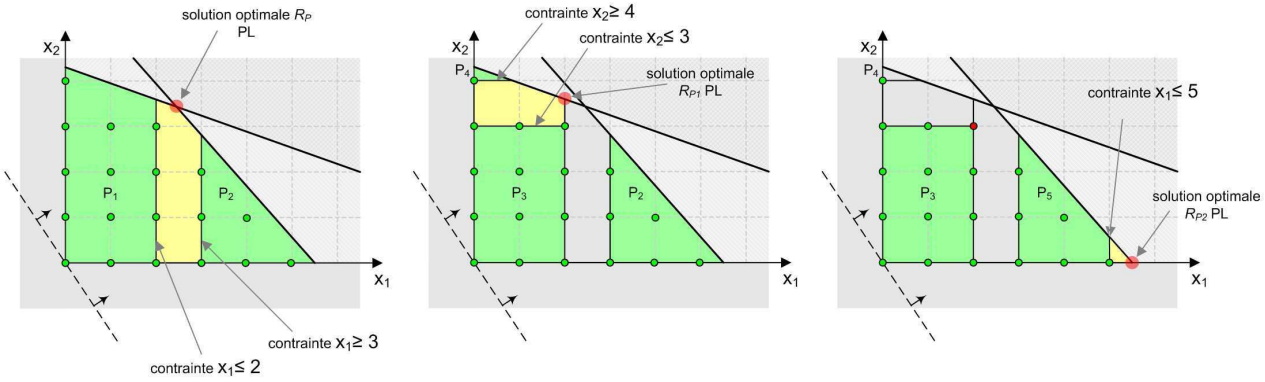


FIGURE 2.7 – Illustration des premières étapes de résolution d'un problème de PLNE par séparation-évaluation.

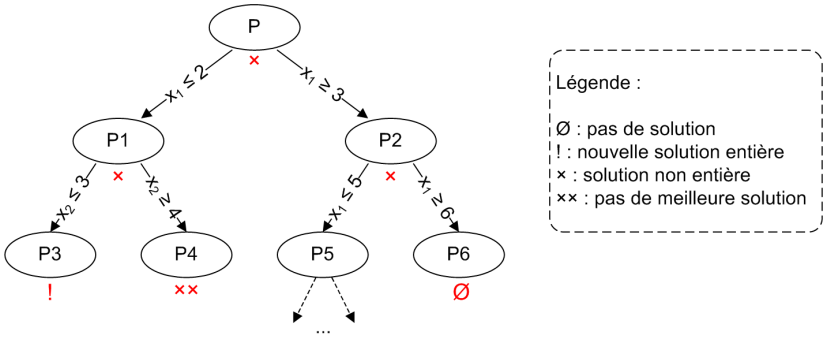


FIGURE 2.8 – Arbre de recherche de la résolution par séparation-évaluation du problème de la figure 2.7.

pressivité beaucoup plus aisée des problèmes, et permet en outre d'exprimer des contraintes non linéaires. Elle est notamment employée en recherche opérationnelle, où sa généricité permet de traiter un grand nombre de problèmes. La PLC permet ainsi de modéliser aisément le problème traité dans cette thèse, de manière globale.

2.4.1 Historique et présentation générale

Les travaux de R. Kowalski [Kowa 74] et d'A. Colmerauer [Colm 70] sur la programmation logique constituent les prémices du domaine de programmation logique par contraintes (PLC). La programmation logique [Lloy 87] est un paradigme de programmation déclarative (et non impérative comme l'ensemble des langages de programmation communément répandus), qui distingue l'aspect logique (expression du problème) de l'aspect contrôle (déduction-inférence). Résoudre un problème nécessite de spécifier un ensemble de prédicats et faits logiques, en utilisant une sémantique déclarative basée sur les clauses de Horn (sous-ensemble de la logique des prédicats). Puis, de manière transparente pour l'utilisateur, un solveur logique combine l'emploi d'un arbre de dérivation logique sur un but donné, avec les principes de substitution et d'unification des variables associées. Ce principe de résolution est appelé SLD (règle de Sélection, stratégie Linéaire, programme Défini) [Robi 65] et offre une méthode de traitement générique pour la démonstration automatique ou la résolution de problèmes.

Les premiers langages implémentant le paradigme de programmation logique apparaissent dès de début des années 1970. En 1972 notamment, A. Colmerauer et P. Roussel mettent au point *Prolog* [Colm 96], qui deviendra l'un des principaux langages utilisés. L'application de ces derniers reste néanmoins limitée, car la modélisation des données est restreinte à une expression sous forme de termes, et les calculs ne peuvent être dirigés que par les buts (et non par les données). La PLC, dont les premiers travaux aboutissent en 1986 [Jaff 87], généralise la programmation logique en exploitant les techniques algébriques de résolution de contraintes. Elle enrichit la sémantique déclarative par l'expression de contraintes et étend la sémantique opérationnelle à des structures mathématiques diversifiées :

- **les booléens** : la structure logique admise est l'algèbre de Boole, permettant, à partir de variables binaires, d'exprimer un système de contraintes en utilisant les opérateurs *et* ($\wedge$), *ou* ($\vee$), *non* ($\neg$), *implication* ($\rightarrow$ ou $\Rightarrow$), *équivalence* ($\leftrightarrow$ ou $\Leftrightarrow$) ;
- **les rationnels** ($\mathbb{Q}$) **et les réels** ($\mathbb{R}$) : la PLC est ici étendue aux systèmes de contraintes linéaires du type $A.x \leq b$, à variables rationnelles ou réelles. Les méthodes de résolution de contraintes employées sont directement empruntées au domaine de programmation linéaire (voir section 2.3.1), notamment le simplexe [Dant 51] ;
- **les entiers relatifs** ($\mathbb{Z}$) : la PLC considère ici les systèmes de contraintes linéaires dans les *domaines finis* (c'est-à-dire dont les variables sont définis dans un sous-ensemble fini de $\mathbb{Z}$), pour lesquels des mécanismes spécifiques ont été mis en place (présentés en section 2.4.2).

La PLC est ainsi capable de modéliser un spectre plus large de problèmes, tout en disposant des mécanismes de résolution efficaces (dès lors que le problème reste de nature linéaire). Elle a fait l'objet d'une formalisation rigoureuse [Fage 96] ; parallèlement, le langage Prolog s'est enrichi pour aboutir à Prolog IV [Colm 90], qui a même fait l'objet d'une standardisation (ISO/IEC 13211-1). Permettant l'expression de systèmes de contraintes, on parle alors de Programmation par Contraintes (PPC). Par la suite, de nombreux langages et outils ont vu le jour :

- langages et compilateurs basés sur Prolog : SICStus [SICS], CHIP [Dinc 88], ECLiPSe [Wals 97], GNU-Prolog [GNU], YAProlog [YAPr] ;
- *framework* d'unification de langages Prolog : Gecode [Geco] ;

– librairies de PPC pour langages impératifs : IBM ILOG CPLEX CP Optimizer [IBM], Choco [Team 10], JaCoP [JaCo] ;
– langages spécialisés : Oz [Smol 95], Claire [Clai].

La PPC constitue aujourd'hui une alternative crédible aux algorithmes spécialisés de Recherche Opérationnelle pour de nombreux problèmes, comme par exemple les problèmes d'ordonnancement [Case 94].

2.4.2 PLC dans les domaines finis

La PLC dans les domaines finis est un domaine de recherche s'attachant à résoudre les problèmes de *satisfaction de contraintes* (CSP pour *Constraint Satisfaction Problems* en anglais). Leur formalisation est apparue pour la première fois dans la littérature au sein des travaux de Montanari [Mont 74] sur les réseaux de contraintes. Ces problèmes sont exprimés sous forme d'un triplet $P = \langle X, D, C \rangle$ où X est l'ensemble des variables du problème, D est l'ensemble des domaines de valeurs des variables de X, et C est l'ensemble des contraintes. Une variable est dite *instanciée* si une valeur unique de son domaine lui est attribuée. Une solution est une *instanciation* (on parle aussi d'*affectation*) de toutes les variables de X qui satisfasse l'ensemble des contraintes C. Une contrainte $c \in C$ est une relation impliquant k variables (k est appelé *arité* de c). Cette relation peut être de plusieurs types :
– logique : implication ($\Leftarrow, \Rightarrow$), équivalence ($\Leftrightarrow$) ;
– arithmétique : $=, \neq, <, >, \leq, \geq$;
– relations spécifiques appelées *contraintes globales* : par exemple la contrainte `AllDifferent` imposant que toutes les valeurs d'une instanciation soient différentes.

Il est démontré, par le théorème de Cook [Cook 71], que les CSP sont NP-complets. Pour le prouver, on montre qu'un CSP peut être exprimé sous forme d'un CSP binaire (c'est-à-dire un CSP dans lequel chaque contrainte implique au plus deux variables) en introduisant de nouvelles variables. Or le problème SAT ou *problème de satisfiabilité*, un problème NP-complet de référence en théorie de la complexité, est polynômialement réductible en un CSP binaire.

Une catégorie particulière de CSP est la classe des CSOP (pour *Constraint Satisfaction Optimization Problems*), qui consiste à optimiser une fonction de coût tout en satisfaisant un ensemble de contraintes. C'est donc plus particulièrement cette catégorie qui nous intéressera par la suite.

2.4.3 Résolution d'un problème de satisfaction de contraintes

La première façon de résoudre un CSP est de fixer successivement les valeurs de chaque variable et de vérifier que l'affectation respecte les contraintes. Cette méthode systématique est appelée *générer et tester*, elle est très simple mais très inefficace en pratique en raison de l'explosion combinatoire du nombre des affectations possibles avec la taille du problème. Il est à noter que la taille du problème est généralement définie comme le nombre de valeurs du plus grand domaine de D élevé à la puissance du nombre de variables de X (soit $|D|^{|X|}$).

Une recherche de solution peut être représentée sous forme arborescente. Chaque niveau de l'arbre correspond à l'instanciation d'une variable (la hauteur de l'arbre est donc égale au nombre de variables de décision du problème). Un nœud au niveau i représente donc une instanciation partielle des i premières variables. Si cette instanciation ne viole pas les contraintes du problème, alors ce nœud possède des nœuds fils dont le nombre correspond au nombre de valeurs possibles de la variable correspondant au niveau inférieur. Un nœud fils représente soit une instanciation complète qui valide les contraintes — c'est alors une solution du problème —

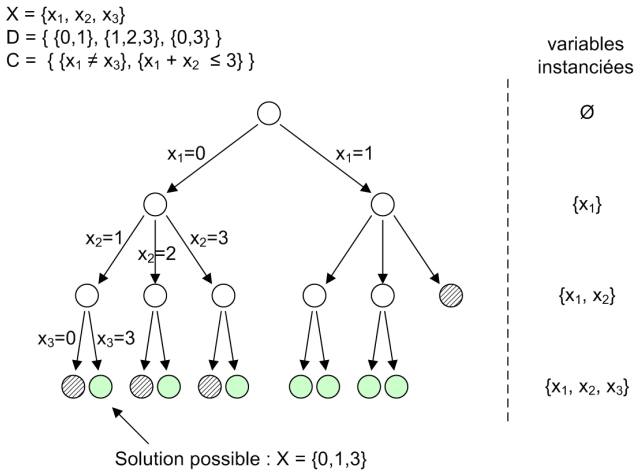


FIGURE 2.9 – Exemple d'arbre de recherche. Les feuilles hachurées violent au moins une contrainte, tandis que les feuilles pleines représentent une solution réalisable.

soit une instanciation qui viole au moins une contrainte. La représentation exhaustive de toutes les branches d'un arbre de recherche décrit ainsi l'ensemble des états possibles du problème. La figure 2.9 illustre un exemple simple.

Lors de la recherche d'une solution, on parcourt l'arbre en *profondeur d'abord*. On instancie itérativement les variables jusqu'à aboutir à une feuille. Soit l'instanciation viole des contraintes, auquel cas on continue la recherche, soit elle est valide et une nouvelle solution est trouvée. Dans le cadre d'un CSOP, la recherche continue également car cette solution ne garantit pas d'être la meilleure trouvée. Pour ce faire, on réalise un retour arrière : on remonte au nœud supérieur dans l'arbre, correspondant à l'étape précédant l'instanciation de la dernière variable considérée. On va alors tenter d'appliquer une autre valeur à cette variable si d'autres nœuds fils existent. Lorsque toutes les valeurs pour cette variable ont été tentées, on remonte récursivement au niveau supérieur.

Dans le cas d'un CSOP, un grand nombre de solutions valides peuvent être évaluées. Contrairement à une recherche de solution dans un CSP, qui s'arrête à la première solution réalisable, il est nécessaire de continuer d'évaluer les solutions réalisables afin de prouver que celle trouvée optimise bien la fonction de coût. Il faut donc bien distinguer dans cette évaluation le temps nécessaire pour trouver la solution optimale (qui peut être très court) et le temps pour prouver qu'il n'existe pas de solution meilleure (qui peut être beaucoup plus long). Pour réduire le nombre d'évaluations, il est possible d'utiliser une méthode de recherche par *séparation-évaluation* déjà mentionnée dans la section 2.3.4. L'idée est de comparer à chaque nœud, si une solution au problème a déjà été trouvée, la borne minimale (dans le cas d'un problème de minimisation) de l'instanciation partielle avec le coût de la meilleure solution trouvée. Si cette borne est supérieure, alors on sait — sans avoir à instancier les variables non encore affectées — qu'aucune meilleure solution ne pourra être trouvée avec l'instanciation partielle en cours. L'arbre sous-jacent au nœud actuel peut donc être élagué (ce qui économise des efforts de calcul). La figure 2.10 schématise cette méthode. Le calcul de la borne est rendu possible à l'aide d'un mécanisme dit de *propagation de contraintes*.

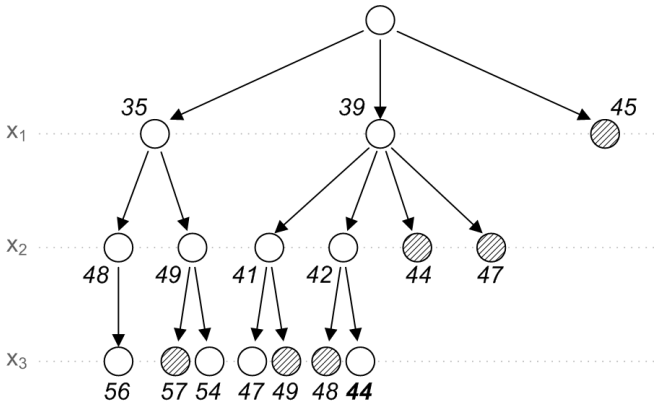


FIGURE 2.10 – Exemple d'arbre de recherche par séparation-évaluation (les valeurs de gauche sont sélectionnées en priorité). Les solutions non réalisables ne sont ici pas représentées. A chaque affectation d'une variable, la valeur de la borne minimale du coût de l'instanciation est calculée. Lorsque toutes les variables sont affectées et qu'une solution réalisable est trouvée, sa valeur sert de borne maximale pour les futures évaluations. Lors des évaluations ultérieures, si la borne minimale calculée est supérieure à la valeur de la meilleure solution trouvée jusqu'alors, le sous-arbre est élagué (nœuds hachurés). Dans cet exemple (construit arbitrairement), la valeur en gras est la solution optimale.

2.4.4 La propagation de contraintes

La propagation de contraintes [Mont 74, Mack 77] est une méthode de filtrage permettant, en analysant la structure du problème sans pour autant résoudre ce dernier, de supprimer les valeurs des variables dont on sait qu'elles n'aboutiront pas à une solution réalisable. Autrement dit, en analysant les contraintes entre les différentes variables, ces mécanismes sont capables de réduire le domaine des valeurs de ces variables et ainsi la taille de l'espace de recherche. Prenons un exemple simple : soit trois variables $X = \{x_1, x_2, x_3\}$ et leurs domaines de valeurs respectifs $D = \{\{1, 2, 3, 4\}, \{0, 1, 2\}, \{3, 4, 5, 6\}\}$. Soit les contraintes suivantes : $x_1 - x_3 = x_2$ et $x_2 = 0$. On peut donc aisément conclure que $x_1 = x_3$: par conséquent, le domaine des valeurs possibles de X devient $D = \{\{3, 4\}, \{0\}, \{3, 4\}\}$ car toute autre valeur ne satisferait pas les contraintes énoncées précédemment. Ainsi, le nombre de combinaisons — qui est aussi le nombre de solutions potentielles à évaluer — se voit fortement réduit. La propagation des contraintes ne garantit cependant pas que toutes les combinaisons restantes sont possibles. Ainsi, si l'on ajoute la contrainte $x_1 \neq x_3$, D demeure inchangé, tandis que les instanciations $X = \{3, 0, 3\}$ et $X = \{4, 0, 4\}$ sont invalides.

Les algorithmes de propagation de contraintes sont employés préalablement à la résolution, mais également au cours de celle-ci. Rappelons que la résolution consiste à fixer successivement la valeur de chacune des variables (opération appelée *labeling* en anglais). Le fait de fixer une valeur unique à une variable revient à ajouter une nouvelle contrainte (d'égalité) au problème. Ainsi, cette contrainte peut être propagée afin de réduire le domaine des valeurs des variables impliquées par des contraintes communes. Dans l'exemple précédent, si l'on fixe $x_1 = 3$, alors on peut propager la contrainte $x_1 \neq x_3$, et conclure que la seule valeur possible pour x_3 est 4.

Plusieurs mécanismes de propagation de contraintes existent [Bess 06]. Les plus employés sont la vérification en avant (*Forward-Checking*, ou FC) et le maintien de la cohérence d'arc (*Maintaining Arc Consistency*, ou MAC). Il consiste, pour chaque contrainte $c \in C$, à vérifier que

pour toute valeur du domaine d'une variable impliquée dans c il existe au moins une valeur dans le domaine de chaque autre variable également impliquée dans c qui satisfasse c. Les algorithmes les plus connus sont AC-3 [Mack 77] et AC-4 [Mohr 86], mais ceux-ci ont été largement dérivés et améliorés (voir [Regi 04] pour une bibliographie exhaustive).

2.4.5 L'énumération

Outre la propagation de contraintes, l'autre étape importante dans la résolution d'un CSP est l'*énumération*. Nous l'avons vu précédemment, un espace d'état peut être représenté sous forme arborescente. Mais il est possible de construire plusieurs arbres différents, selon l'ordre de sélection des variables et des valeurs de celles-ci. Dans l'exemple de la figure 2.9, l'ordre de sélection des variables est x_1, x_2 puis x_3, et l'ordre de sélection des valeurs est un ordre croissant. Ces critères de choix sont souvent déterminants dans une résolution. Ils constituent un élément clé de la recherche, appelé *stratégie de branchement*.

Il existe plusieurs stratégies de branchement, qui s'avèrent plus ou moins efficaces selon les problèmes traités. Les stratégies les plus répandues sont de type *First Fail* : elles consistent à choisir en priorité les variables qui sont le plus susceptibles d'échouer dans la recherche d'une solution. La figure 2.11 montre l'intérêt de ces stratégies sur un exemple très simple : si l'on a connaissance que la variable x_1 a de grandes chances d'échouer, il est préférable de la positionner en tête de l'arbre (c'est-à-dire l'instancier en priorité). En effet, si elle viole les contraintes, elle permet ainsi d'élaguer le sous-arbre et par conséquent un grand nombre d'évaluations intermédiaires. Connaître les risques d'échec de l'instanciation d'une variable est souvent impossible. Plusieurs stratégies de ce type, basées sur la connaissance de la structure du problème, sont donc souvent utilisées :

- choix en priorité de la variable de plus petit domaine. C'est également l'exemple de la figure 2.11 : on met en tête de l'arbre les variables avec le nombre de valeurs le plus faible, si bien que les sous-arbres ont la taille la plus importante possible. Si une instanciation échoue, on élaguera donc une partie plus grande de l'espace d'état ;
- choix en priorité de la variable la plus contrainte. A la première itération — lorsqu'aucune variable n'est instanciée — on prend la variable de plus petit domaine. Ensuite, on choisit les variables les plus contraintes en priorité (ayant potentiellement plus de chances d'en violer au moins une) ; une autre stratégie, très proche, consiste à instancier en dernier les variables non liées entre elles (celles-ci ne s'opposant pas les unes aux autres) ;
- choix en priorité de la variable disposant de la borne minimale (resp. maximale) la plus basse (resp. haute) (si cette valeur est relative à une fonction de coût par exemple).

Dans le cas de problèmes peu contraints, une stratégie propre au problème traité (au regard d'une fonction à optimiser, comme un choix de *moindre regret*) reste souvent la meilleure alternative.

Quant à la stratégie de sélection des valeurs, elle consiste cette fois à sélectionner en priorité celle qui a le plus de chances d'aboutir à une solution réalisable (dans l'exemple de la figure 2.11, on choisirait donc en priorité la seconde valeur de x_3, si c'est possible). Là encore il existe des stratégies génériques :

- choix en priorité de la valeur avec le plus grand *support* (c'est-à-dire le plus grand nombre parmi les autres variables de valeurs consistantes avec la valeur choisie) ;
- choix en priorité de la valeur dont découle le plus grand produit des tailles de domaines des variables non encore instanciées ;
- choix en priorité de la valeur réduisant le moins les tailles de domaines des variables non encore instanciées.

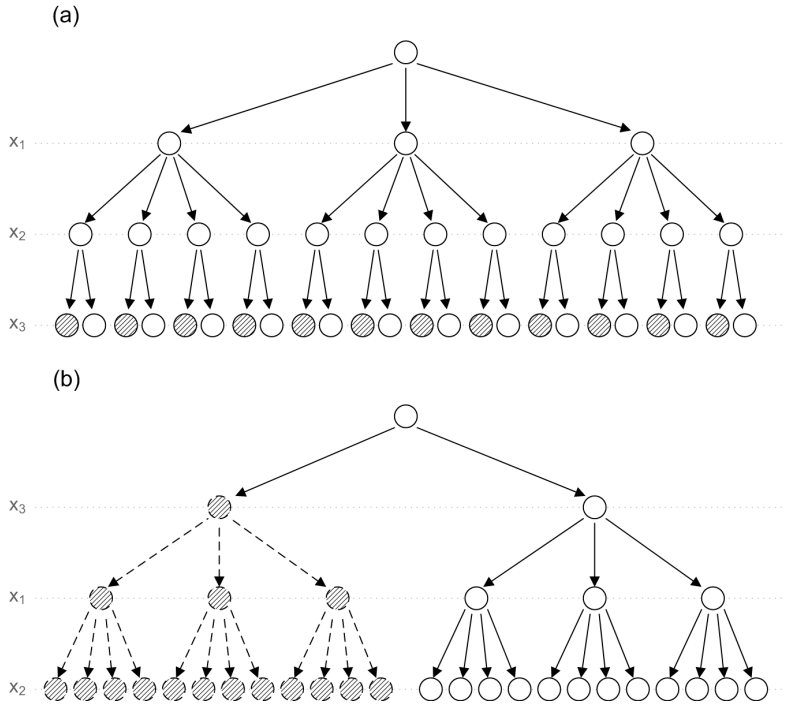


FIGURE 2.11 – Deux exemples d'arbre de recherche pour un même problème. On suppose ici que les variables x_1,x_2 et x_3 ont respectivement 3,4 et 2 valeurs possibles. Si l'on suppose également que le problème est irréalisable pour la 1^e valeur de la variable x_3, on s'aperçoit qu'une instanciation au plus tôt dans la recherche permet un élagage de la moitié de l'arbre de recherche (sous-arbre en pointillés) tandis qu'une instanciation tardive mène à un nombre beaucoup plus important d'évaluations. Nous appelons évaluation chaque tentative d'instanciation d'une variable, correspondant aux flèches pleines : 39 dans l'exemple (a), 17 dans l'exemple (b).

Cependant, les meilleures stratégies restent parfois propres au problème traité. La plupart du temps, on choisira en priorité la valeur minimisant une fonction de coût, ou approchant le plus une fonction d'une valeur désirée. Pour le problème des N *reines* par exemple (un problème classique de satisfaction de contraintes consistant à placer N reines sur un échiquier de taille $N \times N$ sans qu'aucune ne menace une autre), il est admis que la meilleure stratégie consiste à choisir en priorité les valeurs situées au centre du domaine.

2.4.6 Le solveur ORTAC

2.4.6.1 Présentation

Le solveur ORTAC est un outil développé au sein de Sagem. Il a été conçu pour résoudre des problèmes de planification d'itinéraires de véhicules tels que ceux présentés en section 1.4, à des fins de préparation de missions militaires au niveau tactique d'une section de combat. L'approche de résolution employée se veut générique, dans le but de pouvoir intégrer différents types de contraintes de mission. Aussi, le choix de son auteur s'est porté sur une approche en programmation par contraintes. Ce choix se justifie notamment par :

– le haut niveau d'expressivité de cette approche, permettant d'exprimer facilement plusieurs

variantes d'un problème ;
- le caractère *anytime* de la résolution (amélioration de la solution courante au cours du temps) ;
- l'efficacité des algorithmes de filtrage existants ;
- les possibilités offertes en termes de stratégies de recherche.

ORTAC est codé en langage Prolog et compilé à l'aide de *SICStus Prolog*. La modélisation du problème en contraintes reprend la formalisation décrite dans le chapitre 1 (voir section 1.4.2), dans le cas multi-véhicules [Guet 07].

Dans cet outil, il est possible de spécifier les caractéristiques et objectifs de mission de plusieurs unités. Il est également possible de coordonner les déplacements de ces unités à l'aide de différentes contraintes (synchronisations, disjonctions...)

2.4.6.2 Stratégie de résolution

La stratégie de résolution mise en place dans l'outil est inspirée des travaux de H. El Sakkout et M. Wallace [Sakk 00] sur le *Probe Backtrack Search*, plus particulièrement le *Static Probe Backtracking*. Initialement, cette approche a été conçue pour accélérer la résolution de problèmes d'ordonnancement dynamiques. Lors d'une perturbation des données d'entrée du problème, la précédente solution est utilisée pour guider la nouvelle résolution en redéfinissant notamment l'ordre d'instanciation des variables pour le mécanisme de retour sur trace. Dans le cas d'ORTAC, c'est la solution à un *problème relaxé* qui est employée. L'idée est de résoudre très rapidement un problème simplifié pour disposer d'une solution partielle « prometteuse » (voir section 3.1.3 pour plus de détails). Cette solution partielle est alors utilisée pour définir l'ordre d'instanciation des variables lors de la résolution du problème global. Pour les problèmes de planification traités, c'est un algorithme de coût minimum de type Dijkstra [Dijk 71] qui est employé, afin de résoudre un problème de plus court chemin entre la position courante et la position finale de chaque véhicule.

2.5 Métaheuristiques pour l'optimisation combinatoire

Les métaheuristiques sont des méthodes d'optimisation très générales. Tout comme la PLC, elles ont été conçues pour traiter un grand nombre de problèmes : leur généricité est ainsi très appréciée en Recherche Opérationnelle. Leurs qualités sont néanmoins différentes. Tandis que la PLC privilégie l'expressivité et l'optimalité sur des instances de taille moyenne, les métaheuristiques privilégient la rapidité de convergence sur des problèmes de grande taille. Il est possible de modéliser l'ensemble du problème de cette thèse pour être résolue par des métaheuristiques, mais au coût d'une adaptation peu évidente. De plus, l'absence d'optimalité est un handicap aux spécifications du système demandé. Enfin, intégrer le cas multi-véhicule par la suite sera tout autant coûteux en termes de développement.

2.5.1 Présentation

Les métaheuristiques sont des méthodes de recherche stochastiques génériques destinées à la résolution de problèmes d'optimisation difficiles. Inspirées de phénomènes naturels pour certaines — l'origine de chacune est précisée dans les sections suivantes — elles ont été mises en œuvre pour résoudre de manière efficace des problèmes dont l'étendue combinatoire rend inapplicable les approches complètes traditionnelles. Elles n'offrent pas en général de garanties d'optimalité, mais présentent la capacité de trouver rapidement, lorsqu'elles sont correctement implémentées, une très bonne solution sous-optimale en un temps réduit. Parfois décriées par les théoriciens

car dépendantes d'un paramétrage établi empiriquement, ces méthodes représentent pourtant une alternative très appréciée des ingénieurs voire de certains chercheurs (comme en biologie moléculaire, pour la détermination de structures spatiales de protéines), face à des problèmes très complexes.

Les métaheuristiques fonctionnent généralement de manière itérative, produisant à chaque cycle une ou plusieurs solutions. N'étant pas complètes, ces méthodes possèdent chacune un critère d'arrêt (généralement un nombre de cycles donné, ou un critère sur l'amélioration du coût de la solution). Le caractère stochastique des méthodes d'exploration propres aux métaheuristiques leurs permettent, lorsqu'un optimum local est atteint, de ne pas y rester bloqué (*a contrario* des méthodes de recherche locale).

Les sections suivantes présentent les métaheuristiques les plus usitées dans le monde de la recherche et de l'ingénierie. Nous nous restreignons aux approches d'optimisation combinatoire, et présentons leur application au problème du voyageur de commerce. Auparavant, nous décrivons rapidement les méthodes de recherche locale, qui précédèrent l'avènement des métaheuristiques, ainsi que les méthodes gloutonnes de construction pouvant être employées pour la génération de solutions (par exemple à l'initialisation d'une métaheuristique).

2.5.2 Méthodes de construction d'une solution

Ces méthodes sont des procédures de construction gloutonnes, c'est-à-dire qu'elles élaborent la solution en réalisant une série d'évaluations locales, et en retenant chaque fois la meilleure option possible (suite d'optima locaux). Elles s'appuient sur la connaissance du problème afin d'élaborer rapidement une solution de bonne qualité. Dans le cadre des recherches sur le TSP, plusieurs méthodes ont vu le jour. Elles sont de trois types : la construction par *plus proche voisin*, la construction par *insertion* et la construction par *réparation*. La première (aussi appelée NNH pour *Nearest-Neighbor Heuristic* en anglais), partant d'un sommet quelconque, construit un tour en ajoutant successivement à la solution partielle le sommet non visité le plus proche. Cette méthode simple construit généralement des solutions de qualité très moyenne : bien que celles-ci soient souvent communes en grande partie avec la solution optimale, quelques arcs de grande longueur sont le plus souvent introduits (voir exemple figure 2.12).

La méthode de construction par *insertion* construit tout d'abord un sous-tour élémentaire, puis introduit itérativement les nœuds encore libres dans la solution. A cet effet, elles peuvent utiliser plusieurs heuristiques :

- insertion par proximité : ajout du nœud dont la distance maximale à un nœud du tour est minimale ;
- insertion par éloignement : ajout du nœud dont la distance minimale à un nœud du tour est maximale ;
- insertion de moindre coût : ajout du nœud dont l'insertion incrémente le moins le coût du tour ;
- insertion aléatoire : ajout d'un nœud choisi aléatoirement.

Pour construire un sous-tour élémentaire, on élabore généralement l'enveloppe convexe de l'ensemble des nœuds à l'aide par exemple de l'algorithme de *parcours de Graham* [Corm 01] ou de *marche de Jarvis* (également appelée méthode du « papier cadeau ») [Jarv 73]. La figure 2.13 présente un exemple de construction par insertion de moindre coût (sur le même exemple que la figure 2.12).

La méthode de construction par *réparation* consiste à utiliser la solution d'un problème proche (non NP-difficile) et de réparer la solution afin qu'elle satisfasse le problème posé. Le meilleur exemple pour le problème de voyageur de commerce est l'algorithme de Christofides

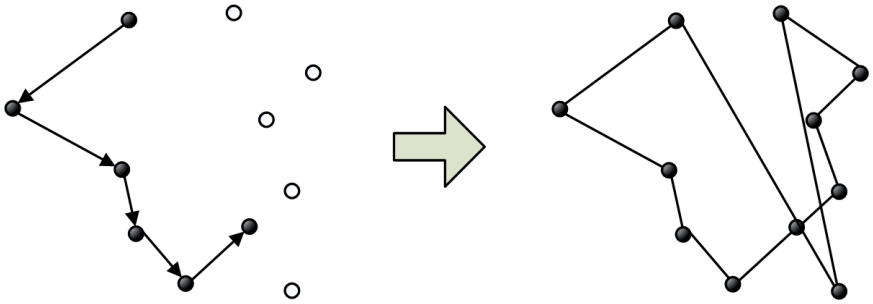


FIGURE 2.12 – Illustration simple de la méthode de construction par plus proche voisin. Le schéma de gauche représente la solution partielle, à l'issue de la 5^{e} itération (les flèches schématisent les étapes successives de construction de la solution depuis le sommet supérieur gauche). Le schéma de droite représente la solution une fois la construction achevée. On observe bien ici le biais de cette approche, qui est facilement amenée à isoler un sommet et contrainte de l'emprunter à la fin, lorsqu'aucun autre sommet libre ne subsiste. La méthode engendre des arcs de grande longueur pénalisant la qualité de la solution et pouvant introduire, comme ici, des chevauchements d'arêtes.

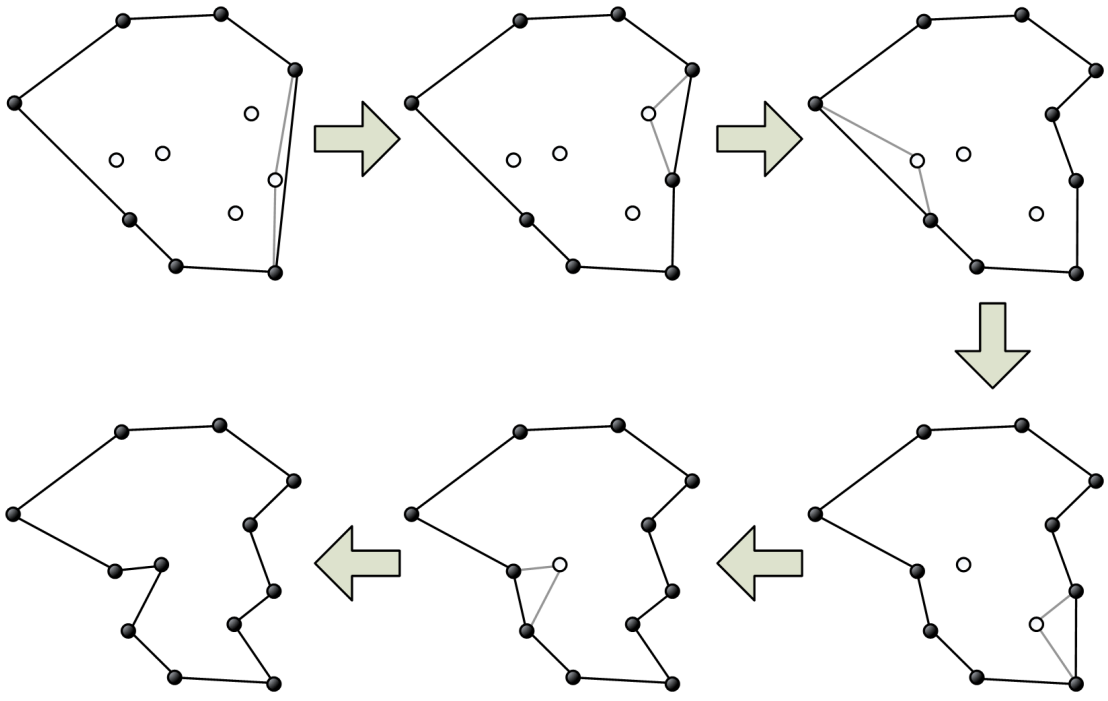


FIGURE 2.13 – Illustration d'une méthode de construction par insertion de moindre coût à partir de l'enveloppe convexe des sommets. Dans cet exemple, la méthode donne la meilleure solution, mais cette construction n'offre aucune garantie d'optimalité.

FIGURE 2.14 – Illustration de la méthode de construction de l'algorithme de Christofides, basée sur l'utilisation d'un arbre couvrant de poids minimum.

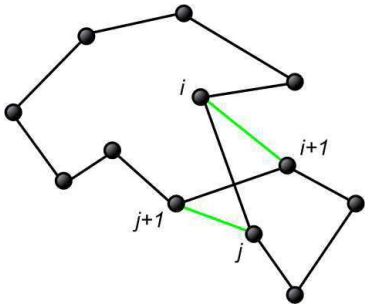


FIGURE 2.15 – Illustration d'une détection de chevauchement et amélioration de la solution par l'algorithme de 2-opt.

[Chri 76], qui part d'un arbre couvrant minimum pour élaborer une solution (voir figure 2.14). De plus, cet algorithme présente l'avantage d'être une 3/2-approximation (c'est-à-dire que la solution est nécessairement inférieure à $3/2.OPT$, où OPT est le coût de la solution optimale).

2.5.3 Méthodes de recherche locale

Les méthodes de recherche locale [Aart 97] sont des procédures d'amélioration (ou de raffinement) d'une solution. Les plus connues d'entre elles, initialement développées pour le TSP, sont les algorithmes *2-opt* [Croe 58] et *3-opt* [Bock 58], repris plus tard par S. Lin avec *k-opt* [Lin 65] puis formalisés pour devenir l'algorithme de *Lin-Kernighan* [Lin 73]. L'algorithme 2-opt est basé sur une constatation géométrique simple : si une solution à un TSP contient au moins un croisement (c'est-à-dire l'enjambement d'une arête par une autre), alors cette solution n'est pas optimale. Elle peut en outre être améliorée en détectant ces croisements et en substituant la paire d'arêtes incriminées par deux arêtes de taille totale plus courte, et non chevauchées. Pour deux sommets i et j quelconques, si les arêtes $(i, i+1)$ et $(j, j+1)$ se croisent, cela signifie, par inégalité triangulaire, que $d(i,j)+d(i+1,j+1) < d(i,i+1)+d(j,j+1)$. On peut donc substituer les arêtes $(i, i+1)$ et $(j, j+1)$ par les arêtes (i, j) et $(i+1, j+1)$ (voir exemple figure 2.15). Si la structure de données de la solution est un tableau, cela revient à inverser l'ordre des éléments situés entre les indices $i+1$ et j (compris). Les méthodes de type k-opt sont équivalentes dans la logique : elles n'évaluent un échange non pas entre 2 mais k arêtes dans la solution (comparant l'ensemble des combinaisons possibles).

Les méthodes de recherche locale aboutissent généralement à un optimum local. De plus, pour un même problème, de multiples optima locaux peuvent être atteints. Pour deux solutions initiales très proches, une amélioration par recherche locale peut aboutir à deux optima

locaux différents. C'est pourquoi ont été développées plus récemment des approches itératives de recherche locale (ou ILS en anglais, pour *Iterated Local Search*) [Lour 02]. Ces méthodes construisent une solution puis l'améliorent par recherche locale. Une fois un optimum local atteint, on perturbe la solution et on réitère une recherche locale. Certaines méthodes intègrent ce principe dans un processus stochastique, et sont applicables à divers types de problèmes. Elles sont donc à ce titre considérées comme des métaheuristiques : c'est le cas de l'algorithme LSMC [Mart 91] (présenté à la fin de la section 2.5.4.1.c).

2.5.4 Métaheuristiques à solution courante unique

2.5.4.1 La méthode du recuit simulé

2.5.4.1.a Les origines

Inspirée du domaine de la métallurgie, cette méthode a été mise au point au début des années 1980 dans les laboratoires d'IBM par S. Kirkpatrick *et al.* [Kirk 83], et indépendamment par V. Černý [Cern 85]. Elle se base sur les procédés industriels de solidification des métaux. Il est en effet établi qu'un métal à température élevée doit être refroidi lentement (en opposition à la méthode dite de la *trempe*) afin de laisser le temps aux atomes de s'agencer de manière optimale — ce qui se traduit par un état d'énergie minimale du matériau et donc une configuration stable le rendant plus résistant aux contraintes mécaniques. Ces propriétés thermodynamiques sont modélisées dès les années 1950 par N. Metropolis *et al.* [Metr 53] qui mettent au point un algorithme basé sur une chaîne de Markov utilisant une distribution de *Boltzmann* pour échantillonner l'espace de recherche. Cet algorithme a été repris plus tard par W.K. Hastings [Hast 70], et généralisé à d'autres distributions, aboutissant à l'*algorithme de Metropolis-Hastings*.

2.5.4.1.b Présentation de l'approche

L'algorithme de recuit simulé s'appuie sur cette modélisation. On construit tout d'abord une solution initiale dont la valeur de coût est l'*énergie* $E = E_0$. On fixe une valeur de *température* $T = T_0$ suffisamment « élevée ». On sélectionne ensuite une solution candidate dans le voisinage de cette solution initiale, dont on détermine l'énergie E_n. Si la variation $\Delta E = E_n - E$ est négative, on accepte la solution candidate comme solution courante. Si elle est positive ou nulle, elle est acceptée avec une probabilité $\exp^{-\frac{\Delta E}{T}}$. Cette loi appelée *règle de Métropolis* (issue des travaux de modélisation thermodynamique mentionnés ci-avant) emploie la variable de température : on constate que plus T est élevée, plus une solution médiocre (comparativement à la solution courante) aura de chances d'être néanmoins acceptée. C'est ce facteur qui permet à la méthode du recuit simulé de maintenir l'exploration de l'espace de recherche, même si un optimum local est atteint.

Une illustration de la méthode est présentée sur la figure 2.16. Pour la phase de recuit, deux approches sont fréquemment employées. La première consiste à abaisser la température par *paliers* : l'algorithme s'exécute en maintenant la température constante, jusqu'à atteindre un *équilibre thermodynamique*. Cet équilibre désigne le moment à partir duquel l'énergie du système ne décroît plus : il peut par exemple être défini par un nombre fixé de cycles durant lequel la température ne varie pas, ou par une valeur seuil d'amélioration du coût sur un nombre de cycles donné. Une fois cet équilibre atteint, on abaisse la température à un palier inférieur (ces paliers étant définis par une loi de décroissance) et on continue la recherche. L'algorithme s'arrête lorsque le critère d'arrêt de l'algorithme est atteint (on parle alors de *système figé*), par exemple une valeur seuil de la température. La seconde approche consiste non pas à abaisser la

température par paliers, mais entre chaque nouvelle solution, de manière continuelle. On peut alors choisir une règle de décroissance simple comme $T_{i+1} = \lambda.T_i$ avec $\lambda \in [0, 1[$ généralement très proche de 1 afin de ne pas converger trop vite et donc de ne pas stagner dans un optimum local.

2.5.4.1.c Application au TSP

La méthode du recuit simulé a été appliquée à un grand nombre de problèmes, notamment de Recherche Opérationnelle. Une solution est modélisée par un vecteur correspondant à l'ordre successif des villes à visiter (par exemple, $S = [2, 4, 5, 1, 3]$ pour modéliser le tour $2 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 3 \rightarrow 2$ d'un problème à cinq villes) appelé représentation en *chemin*. On peut construire la solution initiale de plusieurs manières, aléatoirement ou par recherche locale par exemple. Le voisinage d'une solution peut être défini de différentes manières. Il peut être constitué de l'ensemble des solutions issues d'une permutation d'un sous-ensemble de villes dans la solution initiale : en choisissant aléatoirement deux indices i et j, une solution voisine est construite en inversant l'ordre des villes entre les indices i et j. Par exemple, soit une solution $S = [2, 7, 4, 5, 1, 8, 3, 6]$ à un problème quelconque de 8 villes. Si $i = 2$ et $j = 6$ (l'indice initial étant 0), la solution devient alors $S' = [2, 7, \mathbf{3,8,1,5,4}, 6]$. Le coût ou énergie d'une solution est calculé à l'aide d'une matrice de distances. C'est ce voisinage qu'utilise Černý [Cern 85] ; en revanche, dans son implémentation, ce n'est pas la température qui est modifiée au cours de l'exécution (celle-ci étant fixée à $T = 0, 1$). On va en fait décroître la taille du voisinage en réduisant l'écart possible entre les indices i et j : ainsi, les permutations autorisées concerneront un nombre de plus en plus réduit de villes. Efficace jusqu'à des instances de quelques centaines de villes, ce type d'approche montre vite ses limites au-delà, en raison du grand nombre de combinaisons de solutions de qualité médiocre (solutions avec croisements). Plus récemment, en revanche, des approches hybridées avec des méthodes de recherche locale appelées *Iterative Local Search* ont vu le jour. Dans ces approches, on ne se contente pas de choisir une solution voisine de la solution courante : on va utiliser un algorithme d'amélioration locale, afin d'obtenir un minimum local. Les solutions voisines sont alors obtenues en perturbant cette solution. Par exemple, on va permuter deux points dans la solution, dans le but d'engendrer des croisements d'arêtes dans la solution. En exécutant une recherche locale de type 2-opt au cycle suivant, on a alors peu de chances de retomber sur la solution précédente (on obtiendra donc un nouvel optimum local). Les méthodes de type ILS impliquent davantage de charge de calcul à chaque cycle qu'un algorithme de recuit simulé simple, mais chaque solution évaluée est un minimum local (de la méthode de recherche locale utilisée). Un algorithme très efficace basé sur ce principe est le LSMC (pour *Large Step Markov Chain*) [Mart 91] : basé sur un algorithme de recuit simulé, il emploie une méthode de recherche locale type Lin-Kernighan et une fonction de perturbation spécifique dite de *double-pont* (voir l'article pour davantage de détails). Cet algorithme est capable de résoudre efficacement des problèmes de plusieurs milliers de villes.

2.5.4.2 La recherche tabou

2.5.4.2.a Présentation de l'approche

Le recherche tabou est une méthode d'optimisation mise au point par F. Glover en 1986 [Glov 86]. Basée sur une méthode de recherche locale (recherche d'une solution dans le voisinage améliorant la fonction de coût), sa philosophie fut d'introduire un mécanisme de mémoire des dernières solutions explorées, sous forme d'une liste appelée *liste tabou*. Les éléments de cette

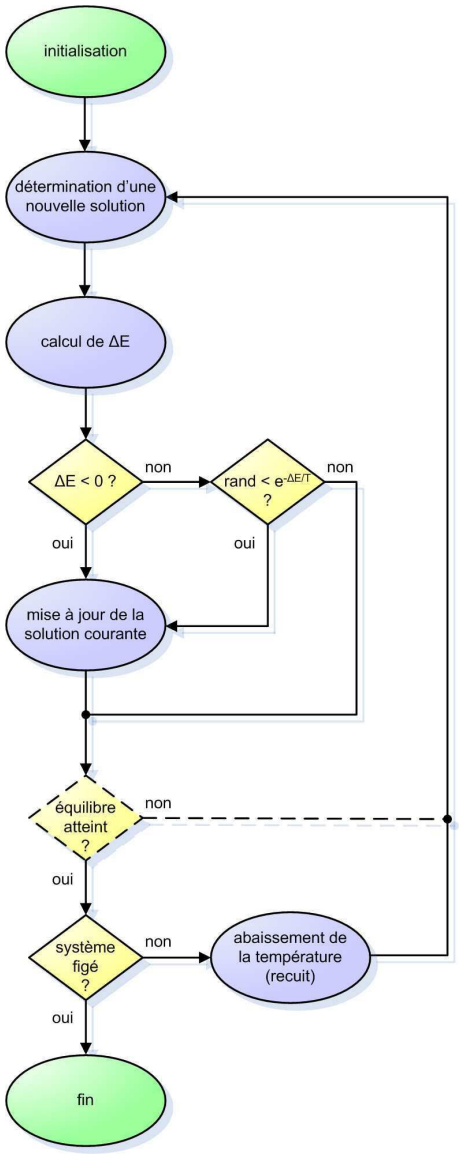


FIGURE 2.16 – Schéma d'approche du recuit simulé. Les étapes en pointillés sont propres à la méthode de décroissance par paliers de température.

liste sont alors exclus de la recherche dans le but d'interdire l'exploration d'une zone déjà visitée. Tabou se démarque ainsi de la méthode du recuit simulé, qui ne capitalise pas sur les exécutions préalables de la recherche.

Afin de limiter les opérations de comparaison de la liste tabou — qui rendraient rapidement l'algorithme inefficace — et pour des raisons évidentes d'espace mémoire, la taille de la liste est limitée : en pratique, elle est souvent faible (moins d'une dizaine d'éléments). Cette taille est un paramètre de configuration important de la métaheuristique. Organisée sous forme de FIFO, ou *file* — structure où le premier élément ajouté est le premier retiré — la liste tabou est donc une mémoire temporaire empêchant à court terme la recherche de faire des mouvements cycliques dans l'espace d'état. C'est ce même mécanisme qui permet à la méthode de s'extraire d'un optimum local, puisque la recherche ne peut ni stagner ni revenir en arrière. Le pseudo-code de l'approche est fourni par l'algorithme 3.

Algorithme 3 Métaheuristique Recherche Tabou

$S \leftarrow solutionInitiale()$
$ListeTabou \leftarrow \emptyset$
tant que Condition d'arrêt non atteinte **faire**
  $ListeCandidats \leftarrow \emptyset$
  pour chaque $S_{candidat} \in meilleursVoisins(S)$ **faire**
    si $\neg Tabou(S_{candidat}, ListeTabou)$ **alors**
      $ListeCandidats \leftarrow S_{candidat}$
    fin si
  fin pour
  $S_{candidat} \leftarrow meilleureSolution(ListeCandidats)$
  $miseAJourListeTabou(S_{candidat}, ListeTabou)$
fin tant que
retourner La meilleure solution trouvée

Une liste tabou peut en principe contenir des informations de toutes natures :
- des solutions sous forme complète, ce qui peut cependant poser des difficultés pour des instances de problèmes de grande taille (espace mémoire, rapidité des comparaisons) ;
- des *mouvements* : modifications apportées pour aboutir à une nouvelle solution (voir section 2.5.4.2.d pour un exemple). Très économe en complexité (mémoire et temporelle), cette implémentation introduit néanmoins un biais en interdisant potentiellement des solutions non empruntées ;
- des valeurs de coût des solutions : très simple, cette implémentation nécessite cependant que les solutions au problème aient des valeurs de coût très variables et que la probabilité de trouver deux solutions de même coût soit très faible ;
- etc.

En pratique, c'est souvent le choix d'une liste de mouvements qui est fait. Ces mouvements correspondent à la définition qui est donnée au voisinage des solutions.

2.5.4.2.b Définition du voisinage

L'ensemble des voisins potentiels d'une solution va dépendre de l'opérateur qui sera employé pour perturber une solution. Dans ce voisinage, la méthode tabou compare l'ensemble des solutions afin de sélectionner la meilleure d'entre elles. En pratique, la taille de ce voisinage est

limitée, en imposant des restrictions aux opérateurs de perturbation, afin que le nombre d'évaluations ne soit pas trop grand (des exemples sont donnés à la section 2.5.4.2.d). Ces restrictions peuvent priver l'algorithme du meilleur optimum local, mais peuvent néanmoins introduire une diversité dans la recherche.

Modéliser une solution interdite par le biais de mouvements induit des situations indésirables : on risque d'interdire une solution du voisinage alors qu'elle pourrait mener à un nouvel optimum. Pour éviter ces situations, il est possible d'ajouter des *critères* aux éléments d'une liste tabou afin d'autoriser l'algorithme, dans certains cas, à braver l'interdiction d'un mouvement. Ce mécanisme s'appelle l'*aspiration*, et doit être implémenté judicieusement, pour ne pas réintroduire de cycles dans la recherche (voir [Glov 97] pour plus de détails).

2.5.4.2.c Mécanismes avancés

La recherche tabou a été expérimentée sur un large panel de problèmes. Pour certains d'entre eux, des chercheurs ont constaté que la recherche restait cantonnée à une zone restreinte de l'espace des solutions. Des stratégies de *diversification* ont donc été développées, afin de permettre à l'algorithme de sortir de cette zone et forcer l'exploration d'un espace plus vaste. La plupart des auteurs ont aussi fait le constat que certaines zones prometteuses étaient trop rapidement parcourues, et que la recherche pouvait passer à côté de l'optimum pour s'en éloigner rapidement. Des stratégies d'*intensification* ont donc également vu le jour.

On peut caractériser une solution à l'aide d'*attributs*, qui sont des critères d'évaluation booléens d'une solution. Dans un problème de coloriage de graphe, par exemple, on peut définir un prédicat p prenant pour paramètre un sommet v et une couleur c. Dans ce cas, pour une solution S définie comme un vecteur de couleurs dont les indices sont les identifiants des sommets, $p_{x,c}(S) = VRAI$ si et seulement si $S(x) = c$ et $FAUX$ sinon. Lors d'une recherche tabou, il est possible d'analyser les attributs des solutions explorées afin d'en tirer avantage. Ainsi, s'il s'avère que les solutions explorées ont en commun certains attributs dont les valeurs ne varient pas, on va pouvoir réaliser une diversification, de manière à positionner la recherche dans un espace où ces attributs sont modifiés. Au contraire, si l'on s'aperçoit que les meilleures solutions rencontrées ont des attributs communs, on peut estimer que ceux-ci sont un facteur de réussite et ainsi mener une intensification en encourageant les solutions possédant ces mêmes attributs. Ces évaluations statistiques vont ainsi pouvoir guider la stratégie de diversification ou d'intensification de l'algorithme (voir par exemple les travaux de E. Taillard sur le problème d'affectation quadratique [Tail 91]).

2.5.4.2.d Application au TSP

Pour le problème du voyageur de commerce, on utilise souvent la représentation en chemin. Pour définir le voisinage d'une solution, on peut employer plusieurs types de permutations :

- l'*inversion* : principe de base de la recherche locale *2-opt* [Croe 58], on va inverser un sous-chemin de la solution, de taille définie ;
- la *transposition* : on échange l'ordre de deux villes dans la solution ;
- le *déplacement* : on retire une ville de la solution et on l'intercale à un autre endroit.

Si l'on décide d'employer une inversion, il suffit de mémoriser dans la liste tabou le mouvement opéré. Par exemple, si $S = [7, 9, 4, \mathbf{5,1,8,3}, 2, 6]$ et si le meilleur voisin est $S' = [7, 9, 4, \mathbf{3,8,1,5}, 2, 6]$ (inversion de l'ordre des villes entre 5 et 3), il n'est nécessaire de conserver que la trace de cette permutation, à savoir le couple $< 3, 5 >$.

Pour limiter le nombre de candidats dans le voisinage, plusieurs solutions existent. On peut par exemple limiter l'ampleur des permutations : si l'on fait le choix d'employer des transpositions, on va pouvoir se restreindre aux combinaisons dans laquelle une ville ne peut être permutée avec une autre que si elles sont séparées de moins de k villes l'une de l'autre. On peut également se restreindre à évaluer le voisinage sur un segment seulement de la solution (les bornes de ce segment changeant au cours du temps). Pour plus de détails sur ces implémentations, voir [Glov 91, Glov 97].

2.5.5 Métaheuristiques à base de population

2.5.5.1 Les algorithmes évolutionnaires

2.5.5.1.a Origines et familles d'algorithmes

Les algorithmes évolutionnaires (ou évolutionnistes) sont inspirés de la théorie de l'évolution des espèces vivantes initiée par C. Darwin [Darw 76] dans la seconde moitié du XIX^{e} siècle. C'est en réalité une grande famille d'algorithmes, car de nombreux chercheurs se sont indépendamment lancés dans l'adaptation des principes de la génétique à partir du milieu des années 50. Pour de plus amples détails, un historique et une classification très complète de ces travaux sont établis dans [Foge 06].

C'est à partir de 1965 et les travaux de I. Rechenberg [Rech 65, Rech 73] qu'apparaissent les premières méthodes d'optimisation stochastiques, basées sur des *stratégies d'évolution*. Conçues pour résoudre des problèmes d'optimisation continue, elle sont souvent considérées comme la première métaheuristique à avoir vu le jour [Beye 02]. L'idée de base de ces méthodes consiste à tout d'abord échantillonner une *population* d'*individus* (un ensemble de solutions), aléatoirement ou non. Dans le cas continu, une solution est définie comme un vecteur de valeurs réelles (qui sont les variables du problème). De ces individus *parents*, on ne va retenir que les meilleurs par une étape de *sélection*. Afin de générer une nouvelle *génération* d'individus (appelés *enfants*), ces individus subissent une *mutation* dans laquelle on va ajouter ou retrancher une valeur aléatoire (tirée selon une loi de distribution normale par exemple) à chaque variable qui les compose. Ce processus est répété itérativement jusqu'à un critère d'arrêt, les individus enfants devenant les parents de la génération suivante.

Parallèlement aux stratégies d'évolution, apparaissent les premiers travaux qui aboutiront à une autre branche d'algorithmes évolutionnaires : la *programmation génétique*, ou *programmation évolutionnaire* [Lang 02, Poli 08]. Elle s'applique au domaine du *Machine Learning* dans lequel on tente d'apprendre à un ordinateur à résoudre un problème de manière autonome, ou plus précisément à définir un programme répondant au mieux à une tâche demandée par l'utilisateur. Les premiers travaux en programmation génétique débutent dès la fin des années 1950 [Frie 58, Frie 59, Samu 63, Samu 67], mais le terme de *programmation évolutionnaire* apparaît au sein des travaux de L. Fogel *et al.* [Foge 66]. Il faut attendre la fin des années 80 pour qu'un paradigme « standard » de la programmation génétique soit établi [Koza 92], formalisant les mécanismes issus de travaux préliminaires : sélection, mutation, croisement (*crossover* en anglais) [Cram 85], paramétrage... Les premières applications sont cantonnées à des problèmes simples, car les algorithmes sont gourmands en temps de calcul. Il faut faire évoluer une population de solutions, qui sont des programmes dont les instructions peuvent être soit ajoutées, soit retirées, soit recombinées. En évoluant, ils deviennent en outre généralement plus volumineux et le calcul de la *fitness* — c'est-à-dire la qualité de la solution au regard du résultat attendu — monopolise alors davantage de temps CPU. La programmation génétique a donc dû attendre

l'avènement des ordinateurs modernes, rapides et capables de calculs parallèles (ce dernier critère étant très intéressant dans les métaheuristiques à base de population, lorsque ses individus évoluent indépendamment) pour donner des résultats probants.

La troisième branche d'algorithmes évolutionnaires est celle des *algorithmes génétiques*. Elle est appliquée aux problèmes d'optimisation combinatoire (bien que des travaux aient adapté cette méthode aux problèmes continus [Chel 00, Haup 04]). Il s'agit de l'analogie la plus fidèle à la génétique réelle, puisqu'une solution est le plus souvent définie sous forme d'un *chromosome* ou *génotype*, qui est le vecteur de valeurs correspondant aux variables de décision du problème. Le *phénotype* associé est donc le résultat de l'instanciation de ces variables, qui se traduit par la qualité de la solution au problème d'optimisation. Au cours des générations, c'est directement sur le génotype que vont être appliquées les opérations de croisement et de mutation afin de faire évoluer la population. La sélection faite à l'issue des évolutions se fait sur l'analyse de la qualité du phénotype. Historiquement, les algorithmes génétiques sont apparus en 1975 avec les travaux de J.H. Holland [Holl 75], et se sont diversifiés par différentes méthodes. Le premier ouvrage de référence, qui jette les bases de l'algorithme génétique « standard », est le livre de D.E. Goldberg [Gold 89]. La section suivante synthétise l'approche classique de ces algorithmes.

2.5.5.1.b Les algorithmes génétiques

La figure 2.17 illustre le fonctionnement de base d'un algorithme génétique.

Phase d'initialisation Dans un premier temps, on initialise l'algorithme en échantillonnant, de manière aléatoire ou non, une population de solutions (ou individus) de taille paramétrée. Ces individus sont ensuite évalués à l'aide d'une fonction de performance (ou *fitness* en anglais), pour déterminer leur qualité. A cet effet, on calcule généralement le coût des solutions. Cependant, si la taille de la population est grande ou si le nombre de générations est important, la phase d'évaluation peut se révéler coûteuse en temps de calcul (car répétée à chaque génération par la suite). Dans ce cas, le programmeur devra trouver une alternative en trouvant une fonction plus simple pour estimer la qualité d'un individu. A l'issue de cette évaluation, le processus d'évolution commence.

Reproduction Une boucle générationnelle commence par la phase de reproduction, en deux étapes. Dans un premier temps (phase de « duplication ») on sélectionne les meilleurs individus, qui sont recopiés d'autant plus souvent que leur performance est bonne. Puis ces solutions sont recombinées à l'aide d'un opérateur de croisement, en intervertissant par exemple une section commune de leur génotype (voir figure 2.18) afin de donner naissance à de nouveaux individus. Avec le croisement, on espère qu'au moins une solution fille tirera le meilleur parti des solutions parentes. Il est à noter qu'il est possible, à la différence de la génétique réelle, de croiser entre eux plus de deux individus.

Mutation Après le croisement, les solutions filles sont susceptibles de subir des mutations, c'est-à-dire des modifications libres de la solution. Cette phase a pour but de diversifier la recherche. Cependant, le taux d'individus subissant une mutation est généralement faible (par exemple 10%) afin de ne pas pénaliser l'apprentissage de l'algorithme.

Phase de remplacement Enfin, on va évaluer les individus de la nouvelle population et sélectionner les meilleurs individus dans l'ensemble de la population (parents et enfants) dans une

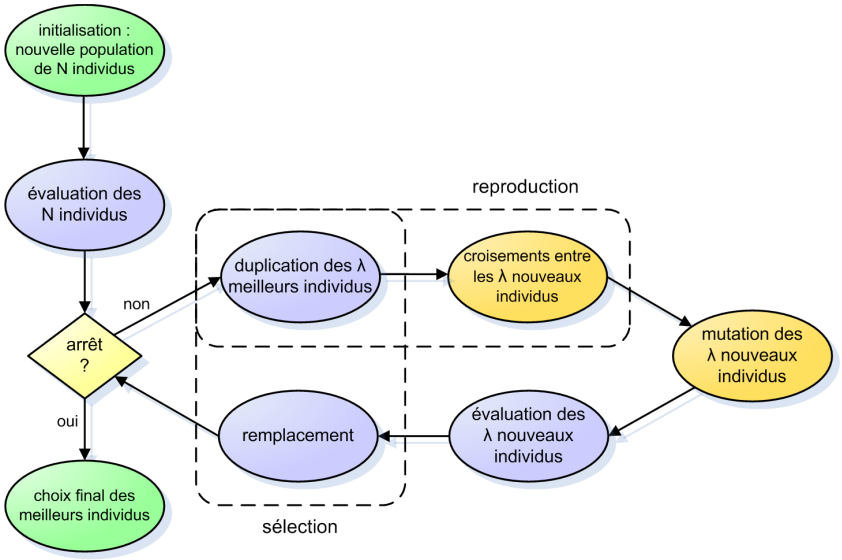


FIGURE 2.17 – Schéma d'approche de l'algorithme génétique. Les étapes de croisements et de mutations constituent la phase d'évolution des individus.

phase de *remplacement* afin d'en retirer les individus les moins performants. Sauf cas particulier, on veille à ce que la population des individus soit de taille constante au cours des générations (après remplacement). A l'issue d'une génération, si le critère d'arrêt n'est pas atteint (en général un nombre de générations défini comme paramètre de l'algorithme), le processus recommence.

Les algorithmes génétiques ont, comme les métaheuristiques précédemment présentées, fait l'objet d'hybridations avec d'autres méthodes d'optimisation, comme les méthodes de recherche locale. L'article [El M 06] dresse une liste assez complète de ces méthodes. Afin de traiter plus rapidement des problèmes de grande taille, ils ont également fait l'objet de travaux de parallélisation sur cartes graphiques, grâce aux technologies GP-GPU (*General-Purpose Processing on Graphics Processing Units*) permettant de programmer en C les processeurs graphiques. Une bibliographie complète de ces travaux est disponible en ligne [Hard 11].

2.5.5.1.c Application au TSP

Les algorithmes génétiques sont bien adaptés pour résoudre le problème du voyageur de commerce, et de nombreux travaux ont été menés sur le sujet. Le premier algorithme à avoir vu le jour est celui de R.M. Brady en 1985 [Brad 85], rapidement suivi par la communauté [Gold 85, Gref 85, Oliv 87]. Parallèlement, d'autres approches évolutionnaires ont également été appliquées au problème [Foge 88, Banz 90, Amba 91].

Pour ce problème, différentes modélisations ont été définies. La plus répandue utilise une représentation en chemin (où un tour est modélisé par un vecteur des villes qui seront successivement empruntées). Cependant, cette modélisation pose des problèmes quant à la mise en œuvre de l'opération de croisement. Prenons un exemple simple : si $S_1 = \{1, 5, 6, 3, 2, 4, 7\}$ et $S_2 = \{1, 4, 7, 3, 2, 6, 5\}$ sont croisées à l'aide d'un point de coupure après la troisième ville, alors les solutions filles seront $S'_1 = \{1, 5, 6, 3, 2, 6, 5\}$ et $S'_2 = \{1, 4, 7, 3, 2, 4, 7\}$. On remarque que ces solutions sont invalides, car elles passent deux fois par certaines villes, et en excluent d'autres. Différents opérateurs de croisement ont donc été imaginés. Une liste complète de ces derniers est présentée dans [Larr 99], qui dresse un comparatif expérimental. Cet article décrit également

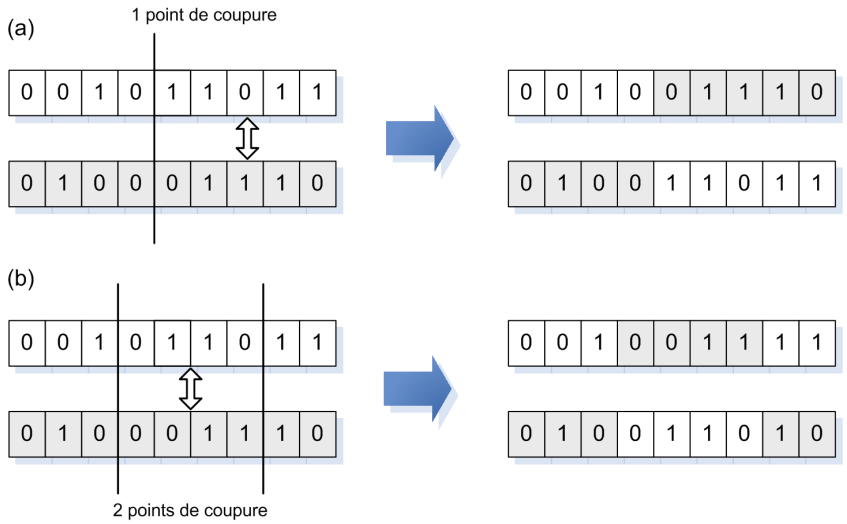


FIGURE 2.18 – Illustration d'une étape de croisement entre deux individus. (a) Croisement simple. (b) Croisement double.

différents opérateurs de mutations, et les autres représentations possibles (et leurs opérateurs associés), comme la représentation par *adjacence* [Gref 85], qui consiste en un vecteur dans lequel la position des villes (autrement dit leur index) détermine la ville précédente dans le tour. Par exemple, la solution $S = \{(3, 5, 7, 6, 4, 8, 2, 1\}$ désigne le tour $1 \rightarrow 3 \rightarrow 7 \rightarrow 2 \rightarrow 5 \rightarrow 4 \rightarrow 6 \rightarrow 8$. Partant de 1 (nécessairement), on vient lire la 1^{e} valeur de S qui est 3 : la ville suivant 1 dans le tour est donc 3. Puis on va lire la 3^{e} valeur de S qui est 7 : 7 est donc le successeur de 3 dans le tour. Puis on lit la 7^{e} valeur qui est 2... La modélisation par adjacence a une particularité : un même tour ne peut être représenté que de deux façons différentes (un par sens de parcours) car on part toujours de la ville 1. Dans une représentation en chemin, un même tour peut être représenté de $2n$ manières différentes (n étant le nombre de villes du problème).

Aujourd'hui, les algorithmes génétiques sont capables de résoudre des instances de TSP de plusieurs milliers de nœuds.

2.5.5.2 Les algorithmes de colonies de fourmis

Une partie des informations présentées ci-après sont tirées du livre [Dori 04] et du rapport technique [Stut 10].

2.5.5.2.a Origines de l'approche

Les colonies de fourmis (ou ACO pour *Ant Colony Optimization*) sont une métaheuristique appartenant aux méthodes d'*intelligence en essaim* : elles mettent en œuvre un mécanisme de mémoire collective permettant de guider la recherche en mutualisant sur les bonnes solutions déjà trouvées. Mise au point au début des années 1990 [Dori 92, Dori 96] avec *Ant System* et dérivée par la suite (voir sections suivantes), cette méthode est issue de l'observation des fourmis réelles qui communiquent par signaux chimiques avec les autres membres de la colonie, afin de signaler un itinéraire à suivre (voir figure 2.19). Elles utilisent pour cela des *phéromones*, une substance attractive naturellement sécrétée par les fourmis, qu'elles dispersent à leur passage, notamment lorsqu'une source de nourriture a été trouvée. Ce processus d'échange d'information

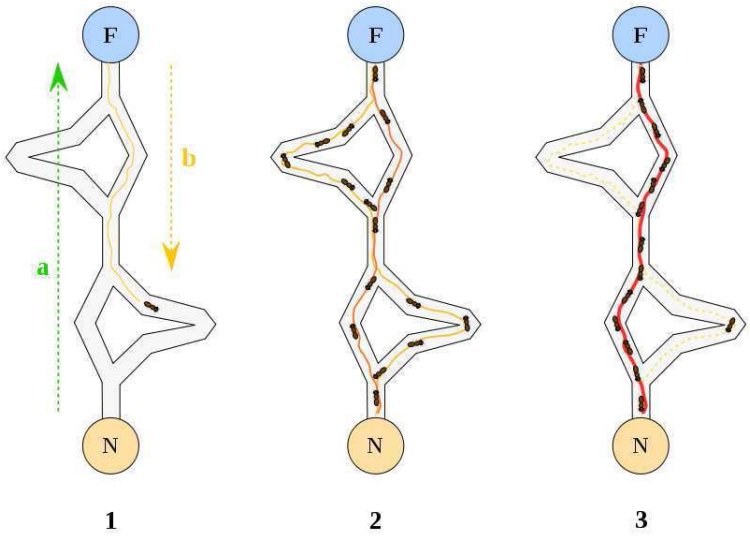


FIGURE 2.19 – Schéma de détermination d'un plus court chemin par les fourmis. 1) Une fourmi trouve une source de nourriture (F) puis revient au nid (N) en disséminant de la phéromone sur son passage. 2) D'autres fourmis rencontrant cette piste vont alors suivre cette phéromone et découvrir la source, déposant à leur tour le traceur chimique. Les portions de chemin les plus courtes vont permettre aux fourmis d'accéder plus rapidement à la source et d'en revenir : elles seront donc davantage marquées que les portions plus longues dans le même laps de temps. 3) Les fourmis sont incitées à prendre le chemin le plus court, et la phéromone sur les portions longues s'évapore, perdant leur attractivité. *Source : Wikipedia.*

utilisant l'environnement ambiant est appelé *stigmergie*. La sécrétion de phéromones permet, dans le cas des fourmis, d'aboutir à des itinéraires de distance quasi-optimale au cours du temps (entre le nid et la source de nourriture), bien que les capacités cognitives de ces insectes soient très limitées.

2.5.5.2.b Méthode générale

La structure générique des algorithmes ACO est décrite par l'algorithme 4. Les problèmes traités par les colonies de fourmis sont souvent définis par le biais d'un graphe $G = (V, E)$. Pour un TSP par exemple, V est l'ensemble des villes à parcourir et E l'ensemble des routes qui les relient. Ces routes sont caractérisées par leur distance, notée $d(i, j)$, pour tout couple de villes i et j.

Construction des solutions Pour construire une solution, une fourmi part d'un sommet de G choisi aléatoirement. Elle élit itérativement le prochain sommet à emprunter parmi ceux non encore présents dans la solution. Ce choix est influencé par deux heuristiques : une fonction de *guidage* notée $\eta : E \to \mathbb{R}$ et le modèle de phéromone qui correspond à une pondération (parfois appelé *attractivité*) sur les arêtes de G, et notée $\tau : E \to \mathbb{R}$. La fonction de guidage va aider la fourmi à construire une solution adaptée au problème, tandis que la phéromone va inciter à emprunter les arêtes des bonnes solutions trouvées dans les précédentes recherches de l'algorithme. En général, deux paramètres α et β permettent de pondérer les influences de τ et

Algorithme 4 Métaheuristique ACO

1: Définition des paramètres, initialisation des phéromones
2: **tant que** Condition d'arrêt non atteinte **faire**
3: Construire les solutions
4: Optionnel : appliquer une recherche locale
5: Mettre à jour la phéromone
6: **fin tant que**
7: **retourner** La meilleure solution trouvée

η respectivement.

Actions optionnelles Des actions spécifiques peuvent être menées pour améliorer les itinéraires des fourmis. On peut par exemple appliquer des méthodes de recherche locale comme 2-opt pour affiner les solutions (et supprimer les chevauchements d'arêtes). Ces méthodes doivent cependant être légères en charge de calcul, pour ne pas trop pénaliser les temps d'exécution de la métaheuristique.

Mise à jour du modèle de phéromone Cette opération est propre à chaque variante. Originellement, elle se décompose en deux étapes : l'*évaporation* des phéromones — on diminue d'un facteur ρ, le taux d'évaporation, la valeur de τ sur chaque arête — et le *renforcement* des itinéraires empruntés par les fourmis. L'évaporation permet d'instaurer un facteur temporel dans la mémoire des bonnes solutions (celles qui ne sont plus intéressantes sont progressivement « oubliées »). Le paramètre ρ et l'importance du renforcement sont décisifs sur la qualité de l'algorithme. Ils vont avoir une influence sur la vitesse de convergence de l'algorithme : une évaporation forte et/ou un renforcement trop important des bonnes solutions vont rapidement contraindre les fourmis à emprunter à nouveau les solutions déjà trouvées, pénalisant l'exploration.

2.5.5.2.c Application au TSP

Les fourmis réelles étant réputées pour leur capacité à trouver des plus courts chemins, les colonies de fourmis artificielles ont logiquement été appliquées en premier lieu sur le problème NP-complet du voyageur de commerce. Plusieurs variantes majeures de la métaheuristique pour ce problème sont présentées maintenant. Pour une description exhaustive des différentes méthodes, nous renvoyons le lecteur à [Dori 04, Monm 09].

Ant System **[Dori 96]** C'est le premier algorithme ACO a avoir vu le jour. Dans cette approche, une fourmi élit le prochain sommet à emprunter à l'aide de l'équation (2.10). Pour une fourmi k située au sommet v, la probabilité d'élire le sommet v' comme prochain point de passage est la suivante :

$$\forall v' \in N_k, (v,v') \in E, P_{v'}(k) = \frac{\tau_{v,v'}^{\alpha} \eta_{v,v'}^{\beta}}{\sum_{v'' \in N_k} \tau_{v,v''}^{\alpha} . \eta_{v,v''}^{\beta}} \tag{2.10}$$

E étant l'ensemble des arcs du graphe et N_k l'ensemble des sommets successeurs de v non encore empruntés. On pourra considérer comme successeurs l'ensemble des sommets du graphe G (le

graphe est alors complet), ou se restreindre à un ensemble de sommets localement voisins, afin de réduire la charge de calcul (on diminue drastiquement le nombre d'évaluations dans le cas de grandes instances de problèmes). $P_{(v,v')}$ est une probabilité, sa valeur est donc comprise dans l'ensemble $[0, 1]$. La fonction de guidage est définie comme :

$$\forall v' \in V, \eta_{(v,v')} = \frac{1}{d_{(v,v')}} \tag{2.11}$$

Elle incitera donc les fourmis à emprunter en priorité les villes à proximité de la ville courante. A l'issue de chaque cycle, lorsque toutes les fourmis ont construit une solution, le modèle de phéromone est mis à jour de la manière suivante :

$$\forall (v, v') \in E, \tau_{v,v'}(c+1) = (1-\rho).\tau_{v,v'}(c) + \sum_k \Delta\tau^k_{v,v'} \tag{2.12}$$

où le paramètre $\Delta\tau$, appelé facteur de *renforcement*, est défini comme suit :

$$\Delta\tau^k_{(v,v')}(c) = \begin{cases} \frac{Q}{L_k(c)} & \text{si } (v, v') \in S_k(c), \\ 0 & \text{sinon.} \end{cases} \tag{2.13}$$

où L_k est la longueur de la solution $S_k(c)$ de la fourmi k au cycle c et Q une constante. Plus une solution est de bonne qualité, plus la phéromone déposée (par unité de longueur) le sera en quantité importante.

Plusieurs implémentations alternatives de renforcement ont été développées, comme notamment :

- *Elitist Ant* [Whit 03] qui renforce davantage la meilleure solution locale ou globale trouvée afin d'orienter plus efficacement la recherche ;
- *Rank-Based Ant System* [Bull 99], qui s'appuie sur *Elitist Ant* et propose pour sa part de classer les solutions de chaque cycle par ordre de qualité, pour ne renforcer que les ω premières (ω étant une variable paramétrable) avec une pondération relative au rang de la solution ;
- *Best-Worst Ant System* [Cord 00], où seules les plus mauvaises solutions sont évaporées, tandis que les meilleures solutions locales sont renforcées.

MAX-MIN Ant System **[Stut 97, Stut 00]** Cet algorithme se base sur l'algorithme *Ant System* et modifie certains mécanismes. Tout d'abord, une seule solution est utilisée pour le renforcement des phéromones (la meilleure solution locale ou globale, selon l'implémentation). De plus, le taux de phéromone associé à chaque arête de G est borné par deux valeurs τ_{min} et τ_{max} (d'où le nom de la méthode). Chaque piste de phéromone est initialisée à la valeur maximum τ_{max}, et mise à jour de façon proportionnelle : lors d'un renforcement, les arêtes avec un taux de phéromone faible seront davantage renforcées que celles disposant déjà d'une attractivité forte. Les auteurs présentent aussi des variantes avec réinitialisation des pistes au cours de la recherche.

Les auteurs développent également une approche avec application d'une recherche locale. Capable de résoudre plus efficacement les problèmes de TSP que son prédécesseur, *MAX-MIN Ant System* est présenté comme une amélioration [Stut 10] par l'auteur même d'*Ant System*.

Ant Colony System [**Dori 97**] Cet algorithme dérive de *Ant System* et y introduit des mécanismes tirés d'autres approches, tels que *Ant-Q* [Gamb 95]. *Ant Colony System* utilise ainsi une « loi proportionnelle pseudo-aléatoire » dans la construction des solutions. Une fourmi k actuellement à la ville v choisira la prochaine ville v' comme suit :

$$v' = \begin{cases} \arg\max_{v'' \in N_k} [\tau_{v,v''}.\eta_{v,v''}^{\beta}] & \text{si } q \leq q_0, \\ J & \text{si } q > q_0. \end{cases} \tag{2.14}$$

où q est une variable aléatoire uniformément distribuée sur $[0,1]$, q_0 est une variable sur $[0,1]$ fixée (paramètre de l'algorithme) et J est une ville sélectionnée aléatoirement grâce à la formule (2.10) dans laquelle $\alpha = 1$. La variable q_0 permet à la recherche d'adopter deux comportements différents : la recherche tendra à une *diversification* si $q > q_0$ avec un comportement similaire à *Ant System*, où au contraire à une *intensification* si $q \leq q_0$ par une approche purement gloutonne de sélection de l'arête la plus attractive. Il est à noter que, dans cette implémentation des colonies de fourmis, le paramètre α est abandonné.

Ant Colony System met en œuvre deux mécanismes distincts de mise à jour des phéromones : une mise à jour *locale*, c'est-à-dire en cours de recherche, opérée par chaque fourmi de la manière suivante :

$$\tau_{i,j} = (1 - \epsilon).\tau_{i,j} + \epsilon.\tau_0 \tag{2.15}$$

où ϵ est appelé *facteur de décroissance*. Le paramètre τ_0 (valeur initiale du modèle de phéromone) est fixé avec une valeur $\tau_0 = \frac{1}{n.L_{NN}}$, où n est le nombre de villes du problèmes et L_{NN} la longueur d'un tour préalablement calculé par recherche locale de plus proche voisin (voir section 2.5.2). Ce mécanisme de mise à jour locale permet de décroître l'attraction des arêtes empruntées par une fourmi et de les rendre moins attractives pour les autres fourmis, d'où une augmentation de la diversité des solutions. La mise à jour *globale* du modèle de phéromone (c'est-à-dire celle effectuée en fin de cycle) est similaire à celle de l'approche *MAX-MIN Ant System* : on ne met à jour que la meilleure solution (locale ou globale).

Dans chaque approche de colonies de fourmis, le paramétrage est spécifique au type d'instance : répartition des villes — distribution homogène ou au contraire groupement en *clusters* —, nombre de villes, symétrie ou non dans les « distances » entre les villes... Un grand nombre de travaux traitent les problématiques de paramétrage de ces algorithmes : ils sont résumés dans [Stut 10].

Outre l'application au problème de TSP classique, plusieurs variantes d'algorithmes de colonies de fourmis ont été développées pour le problème de TSP dynamique. Dans ce problème, les distances entre les villes sont amenées à changer au cours du temps. Cette propriété a initialement été introduite pour modéliser le trafic routier, un embouteillage résultant en un allongement de la distance initiale (par distance, nous parlons ici en fait de coût, en l'occurrence le temps de parcours). Certaines versions de TSP dynamiques permettent également l'ajout ou la suppression de villes. Une première approche pour traiter ces problèmes la littérature est P-ACO (pour *Population-based ACO*) [Gunt 02], parfois également appelé *FIFO-Queue ACO* en raison de la structure de données employée pour maintenir une *population* de bonnes solutions précédentes, utilisée pour construire une heuristique de guidage. Elle peut être vue comme une alternative à l'élitisme pour les problèmes dynamiques. La particularité de P-ACO est l'attention portée à la rapidité d'exécution. Notamment, les mécanismes d'élection d'une nouvelle ville et de mise à

jour du modèle de phéromone sont modifiés afin de réduire leur complexité temporelle et d'offrir une meilleure réactivité de l'algorithme. Une seconde approche, AS-DTSP [Eyck 02], est une adaptation de l'algorithme *Ant System* (décrit plus haut) pour le cas du TSP dynamique. Dans cette implémentation, une borne basse est définie pour les taux de phéromone afin de se prémunir des cas où une arête ne serait plus empruntée. De plus, les auteurs proposent un mécanisme dit de *shaking* permettant, après une modification de l'environnement, de redistribuer en partie les quantités de phéromone afin de relancer l'exploration, tout en garantissant que l'ordre d'importance des valeurs entre les arêtes soit conservé (maintien de l'apprentissage).

2.5.5.3 L'optimisation par essaim particulaire

L'optimisation par essaim particulaire est une autre approche d'intelligence en essaim. Mise au point en 1995 [Kenn 95] par J. Kennedy et R.C. Eberhart, cette métaheuristique s'inspire notamment des études sur les mouvements coordonnés des bancs de poisson et des nuées d'oiseaux. Elle a fait l'objet de nombreuses études et constitue aujourd'hui une méthode éprouvée [Cler 05, Lazi 09], notamment pour les problèmes d'optimisation continue. Bien que des travaux d'adaptation au problème du TSP aient été réalisés avec succès (une liste exhaustive de ces approches est menée dans [Gold 08]), ce n'est pas la méthode la plus adaptée pour la résolution de TSP. Nous renvoyons donc le lecteur aux ouvrages mentionnés ci-avant, et ne nous étendrons pas davantage sur la description de l'approche.

Chapitre 3

Une méthode de sonde à base de métaheuristiques

3.1 Un guidage efficace pour une résolution rapide

3.1.1 Rappel des objectifs

Les travaux que nous présentons maintenant ont pour objectif d'améliorer les performances du solveur ORTAC en termes de temps de résolution. Le but est de permettre à cet outil, initialement développé pour la préparation de missions militaires, d'être embarqué sur les équipements de liaison des véhicules (vétronique) afin de permettre une replanification en cours de mission. Comme nous l'avions mentionné en section 1.1.3, les temps de résolution doivent être conformes à la doctrine militaire, qui interdit à un véhicule (sauf instruction) de stationner au cours de la mission. Dans les faits, il faut donc que les nouveaux itinéraires soient déterminés dans un délai inférieur à la dizaine de secondes pour satisfaire les critères d'exigence.

Le problème traité ici se retreint au cas mono-véhicule. Cependant, **en vue d'une intégration future des travaux à l'approche multi-véhicule**, la modélisation existante devra être conservée.

3.1.2 Bases techniques des travaux

Pour accélérer la résolution du solveur ORTAC existant (voir section 2.4.6), nous sommes repartis de l'idée mise en œuvre par son auteur. Les travaux réalisés [Guet 07] montrent que l'utilisation de la solution partielle d'une relaxation du problème permet un meilleur guidage de la résolution par séparation-évaluation et améliore substantiellement ses performances. Cependant, le problème relaxé considéré est un simple plus court chemin allant du point de départ au point d'arrivée. La solution partielle obtenue ne considère donc pas les points de passage obligatoires, ni les contraintes de capacité éventuelles. Le guidage réalisé se base ainsi sur une information permettant à la recherche d'évaluer en priorité les solutions dont les flots sont situés le long de l'axe constitué entre les points de départ et d'arrivée, sans garantie de faisabilité. Une instance de problème très contrainte (contraintes de capacité fortes, points de passage obligatoires très éloignés de la solution partielle) va retarder sensiblement la découverte d'une solution réalisable, et par extension augmenter le nombre de solutions évaluées par la méthode de séparation-évaluation en l'absence de borne de coût (voir pour plus de détails sur la méthode en section 2.4.3). Pour corriger ce problème, nous avons donc envisagé de considérer un problème relaxé plus complexe. En prenant en compte les contraintes en amont de la résolution globale, on va

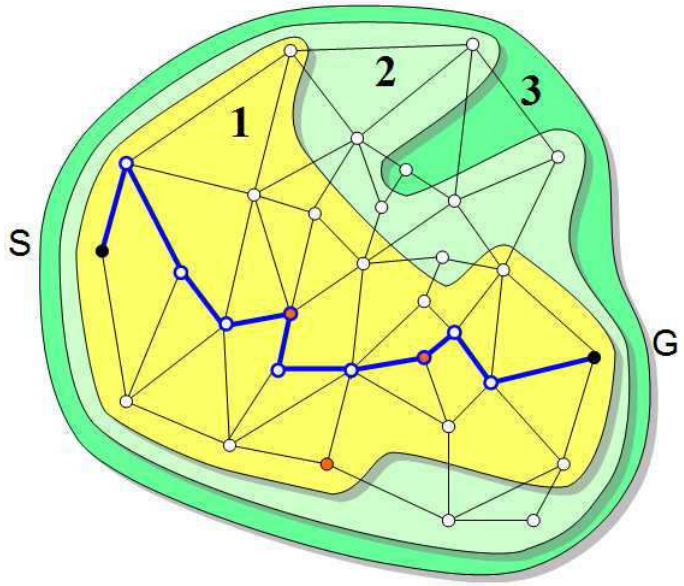


FIGURE 3.1 – Illustration de la méthode de sonde appliquée à un exemple simple. Les points S et G désignent les points de départ et d'arrivée respectivement. Les nœuds en orange sont les points obligatoires. L'itinéraire en gras (bleu) est la solution partielle utilisée : un chemin de plus courte distance entre S et G (notons qu'elle ne passe pas nécessairement par les points obligatoires). La métrique ici appliquée est une métrique de *bonds minimum* : les ensembles désignent les nœuds situés à 1, 2 et 3 bonds de la solution partielle.

pouvoir guider la résolution vers des solutions réalisables et permettre à la séparation-évaluation d'élaguer efficacement l'arbre de recherche.

3.1.3 Précisions sur la méthode de sonde

Comme nous l'avons mentionné précédemment, la méthode de sonde va utiliser la solution d'une relaxation du problème pour calculer une distance entre certaines variables de décision du problème et cette solution. Cette distance est utilisée pour trier ces variables, afin de l'orienter vers une zone prometteuse de l'espace d'état. Dans notre application, ces variables de décision sont les variables de flot ϕ associées aux arcs du graphe G. Trier ces variables va permettre à la résolution par séparation-évaluation, si le problème n'est pas trop contraint, de déterminer plus rapidement un bon itinéraire pour le problème. La figure 3.1 montre une illustration de cette méthode sur un exemple simple. Dans cet exemple, une métrique dite de *bonds minimum* est appliquée. Elle consiste à calculer, pour chaque nœud, le nombre minimal d'arcs à emprunter (donc de « bonds » à réaliser) dans le graphe pour atteindre la solution partielle. Ces valeurs, associées aux nœuds, sont déterminées en appliquant un algorithme de plus court chemin dans lequel chaque valeur de distance est fixée à 1. Les valeurs associées aux arcs sont quant à elles calculées comme étant la valeur maximale entre les valeurs des nœuds de départ et d'arrivée.

3.1.4 Description de la nouvelle approche

Nous présentons une nouvelle approche de résolution dans laquelle nous traitons un problème de plus court chemin avec *prise en compte des points de passage obligatoires*, formalisé en section 1.3.2.3. Cette approche est décrite schématiquement dans la figure 3.2, et détaillée ci-dessous. Nous reprenons pour cela le formalisme exposé en section 1.4.2, qui sera conservé pour la suite du document.

La détermination d'une solution de plus court chemin passant par l'ensemble des points de passage obligatoires est une version relaxée du problème initial puisqu'on ne prend pas en compte les contraintes de capacité éventuelles. De plus, cette solution partielle ne garantit pas qu'un véhicule ne passera qu'une fois à chaque sommet : l'unicité du flot à chaque sommet du graphe (imposé par le modèle de la formule 1.5 en section 1.4.3) n'est ainsi pas assuré.

Modélisation du problème relaxé Dans cette étape, on extrait le graphe G des données du problème et on construit le graphe complet $G' = (V', E')$ constitué des nœuds $V' = V_m \cup \{v_s, v_g\}$. Pour tout couple de nœuds $u, v \in V'^2$, on calcule le plus court chemin dans G. La distance $d_{u,v}$ de ce plus court chemin est alors associée à l'arête $e = (u, v) \in E'$. Les distances associées aux arcs de G' représentent donc les distances réelles dans G, et non une estimation obtenue par un calcul de distance euclidienne.

Calcul du chemin hamiltonien minimal Trouver le chemin hamiltonien de longueur minimale dans G', correspondant au problème dérivé du TSP et formalisé en section 1.3.2.3, est NP-complet. Aussi, nous présentons en section 3.2 un algorithme spécifique de résolution. Cette résolution s'appuie sur la matrice de distances calculée précédemment pour calculer un plus court chemin de longueur *réelle*.

Reconstruction de la solution La reconstruction est réalisée en recalculant les plus courts chemins entre chaque couple de points du chemin hamiltonien. On obtient donc un plus court chemin dans G, passant par l'ensemble des points obligatoires.

La modélisation du problème relaxé implique l'exécution de $|V'|$ recherches de plus court chemin de type Dijkstra. Dans le cas où V_m est grand, il est possible de se passer de ces recherches et de pondérer les arcs de G' avec une estimation de distance (par exemple une distance euclidienne). Cette méthode permet de n'effectuer que $|V'| - 1$ recherches. En revanche, la solution peut être fortement biaisée si les valeurs de distance des chemins réels sont très élevées au regard de leur estimation. Les graphes les plus importants se limitant à quelques centaines de nœuds, et nos travaux privilégiant l'optimalité, nous n'avons pas retenu cette alternative.

Lors de la reconstruction de la solution, on peut se passer de recalculer des chemins déjà déterminés lors de la modélisation, en sauvegardant l'ensemble des chemins. Cette alternative est cependant coûteuse en espace mémoire dès lors que les instances sont grandes.

3.1.5 Justification et originalité de l'approche

Le problème de trouver un chemin hamiltonien minimal, s'il constitue une relaxation du problème global, est cependant lui aussi NP-difficile. Aussi, traiter cette relaxation avec un algorithme déterministe peut aboutir à des temps de calculs significatifs. Or, nous souhaitons ici que la détermination d'une solution partielle soit réalisée très rapidement, car elle ne constitue qu'un premier pas dans la résolution du problème global. Pour faire face à ces exigences de rapidité, le choix d'une résolution partielle par métaheuristiques s'est imposé naturellement. Ces méthodes

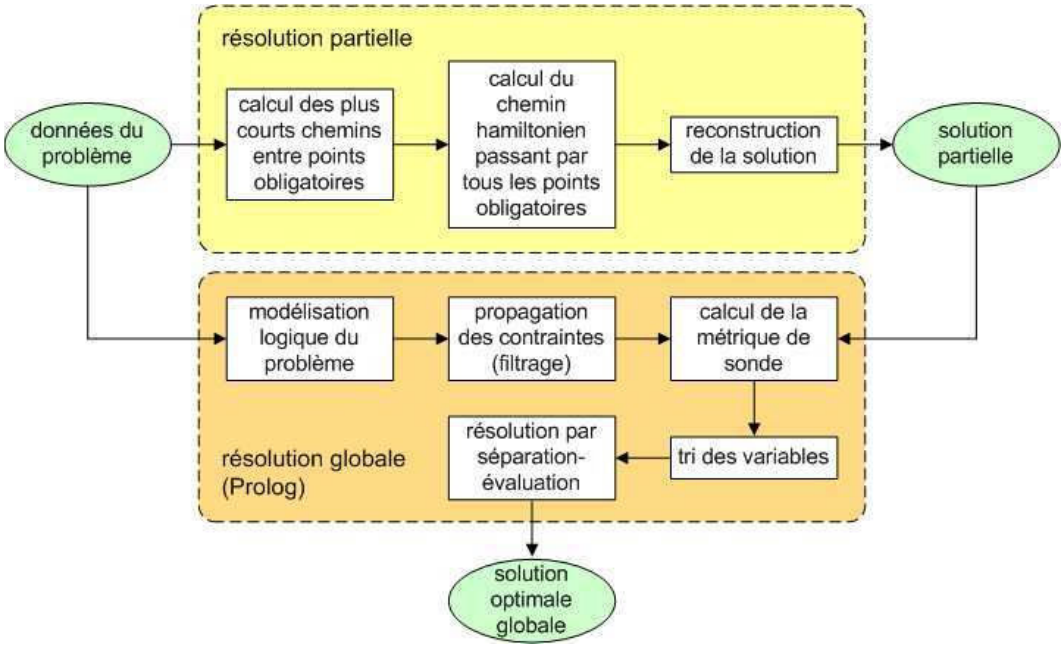


FIGURE 3.2 – Schéma descriptif de la nouvelle approche de résolution.

(voir section 2.5), de caractère *anytime*, ont la capacité de trouver très rapidement une bonne solution, sans toutefois disposer de garantie d'optimalité. Dans notre cas, cette optimalité n'est pas requise puisque la solution partielle ne sert que pour le guidage de la recherche. Si une solution partielle non optimale est utilisée, le solveur de programmation par contraintes restera à-même de trouver la solution optimale au problème global, et de fournir la preuve d'optimalité. Ceci étant, la meilleure solution partielle possible reste souhaitée puisqu'elle permettra (on l'espère) un guidage vers la zone la plus prometteuse de l'espace d'état.

La méthode présentée ci-après implémente une adaptation des métaheuristiques de colonies de fourmis, dont la solution est utilisée pour guider le solveur de programmation par contraintes. L'originalité de cette approche hybride est de pouvoir tirer parti des avantages de chaque méthode : la rapidité des métaheuristiques à trouver une bonne solution rapidement, et les garanties d'optimalité et de complétude de la programmation par contraintes. Bien entendu, le caractère *anytime* de l'approche est également conservé.

L'algorithme des colonies de fourmis a été choisi pour plusieurs raisons. Tout d'abord, il est bien adapté pour résoudre le problème du voyageur de commerce (voir section 2.5.5.2.c). Contrairement à d'autres métaheuristiques, cette méthode bénéficie d'un mécanisme d'apprentissage permettant de mieux explorer l'espace de recherche selon ses spécificités. Ensuite, elle manipule une structure de graphe qui traduit bien les problématiques de planification d'itinéraire, et permet d'imaginer des aménagements dans l'implémentation initiale afin de l'adapter à des problèmes plus avancés. Enfin, elle est également appropriée pour les problèmes dynamiques : on va pouvoir maintenir l'apprentissage et modifier les caractéristiques de l'environnement tout en maintenant l'exécution de l'algorithme. Cette propriété nous sera utile dans les travaux présentés en fin de ce chapitre.

3.2 Les colonies de fourmis pour la couverture des points obligatoires

3.2.1 Description de l'algorithme

Pour résoudre le problème de chemin hamiltonien minimal (dont les sommets de départ et d'arrivée sont précisés), il a fallu adapter l'algorithme des colonies de fourmis. Ce problème étant très proche du problème de voyageur de commerce, les modifications à apporter ne sont que légères. La seule différence notable réside dans la construction de la solution. Dans l'algorithme classique — et quelles que soient ses variantes — dédié à la résolution du problème de voyageur de commerce, une fourmi construit une solution de la manière suivante : partant d'un nœud choisi aléatoirement, on élit itérativement le prochain nœud du tour parmi l'ensemble des nœuds de V' non encore visités, et ce tant que cet ensemble n'est pas vide. Dans le cas du chemin hamiltonien minimal, il faut incorporer dans la recherche les contraintes apportées par les points de départ et d'arrivée. En effet, on ne peut appliquer directement la méthode de construction sus-mentionnée puisque les nœuds v_s et v_g doivent nécessairement se trouver aux extrémités du chemin. Nous avons donc développé plusieurs approches de construction présentées ci-dessous.

Construction *uni-directionnelle* Cette méthode débute la construction en initialisant la recherche au nœud v_s, puis élit itérativement le prochain nœud du chemin parmi l'ensemble des nœuds $V'\backslash\{v_s, v_g\}$ non encore visités, et ce tant que cet ensemble n'est pas vide. Une fois la procédure achevée, v_g est ajouté en queue du chemin. On respecte ainsi les exigences imposées par le problème.

Construction *bi-directionnelle* Cette méthode est similaire à la construction uni-directionnelle, mais elle va initialiser les recherches des fourmis en alternant entre v_s et v_g. On utilise pour cela une numérotation associée aux fourmis : si le numéro de la fourmi est impair, elle débute en v_s et effectue sa recherche dans $V'\backslash\{v_s, v_g\}$, puis v_g est ajouté en fin de séquence. Si la fourmi est de numéro pair, alors elle débute sa recherche du nœud v_g construit sa solution en élisant successivement les nœuds du chemin dans $V'\backslash\{v_s, v_g\}$, et achève son parcours par le point v_s.

Construction avec *initialisation aléatoire* Dans cette méthode, la construction d'une solution démarre d'un nœud choisi au hasard dans V', et non plus nécessairement des points v_s ou v_g. Cette construction est ainsi la plus fidèle aux algorithmes « classiques » de colonies de fourmis. La solution est ensuite construite itérativement en sélectionnant itérativement le prochain nœud dans l'ensemble des nœuds de V' non encore visités. Cependant, on risque d'élire le nœud v_s ou v_g, qui se retrouvera alors au centre de la séquence en construction. Pour corriger ce problème, une fois que le nœud v_s (repectivement v_g) est atteint, on ajoute v_g (respectivement v_s) immédiatement après. On continue alors la recherche jusqu'à ce que tous les nœuds de V' aient été visités. Une fois la recherche terminée, on sépare la séquence obtenue en deux sous-séquences, en introduisant un point de coupure entre les points v_s et v_g (qui se suivent nécessairement). Ces deux sous-séquences sont alors interverties, pour obtenir une solution dont les points extrêmes sont bien v_s et v_g (l'ordre du chemin peut être inversé, car partant de v_g : l'ordre de la liste peut alors être permuté, mais ce n'est pas strictement nécessaire). La méthode est résumée par la figure 3.3.

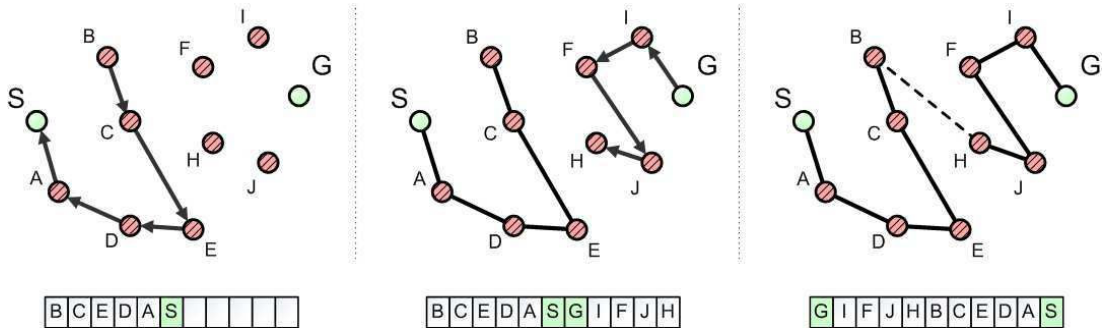


FIGURE 3.3 – Schéma descriptif de la construction avec initialisation aléatoire. S et G sont respectivement les points v_s et v_g. La recherche se fait dans un graphe non orienté, les flèches indiquent simplement la progression de la recherche. On démarre la recherche d'un point choisi aléatoirement, jusqu'à élire S ou G, ici S (schéma de gauche). On continue alors la recherche depuis G, jusqu'à avoir parcouru tous les nœuds. On intervertit alors les deux sous-séquences, ce qui correspond à connecter entre eux le nœud de départ de la recherche et le dernier visité.

3.2.2 Implémentation et paramétrage

Décider de l'algorithme de colonies de fourmis à implémenter n'est pas chose aisée, tant les variantes sont nombreuses. Aussi, nous avons fait le choix de repartir d'un algorithme simple, en nous basant sur *Ant System* [Dori 96], présenté en section 2.5.5.2.c. Puis, nous avons mis en place une méthode (dite de *racing* dans la littérature des métaheuristiques) qui compare les performances de l'approche avec plusieurs jeux de paramétrage différents. En évaluant le gain apporté par les mécanismes qui nous semblaient intéressants, nous avons pu, au fur et à mesure des essais, enrichir notre approche et aboutir à des colonies de fourmis véritablement efficaces pour notre problème.

Loi proportionnelle pseudo-aléatoire Dans notre implémentation, nous avons réintroduit le paramètre q_0 de l'approche *Ant Colony System* [Dori 96] qui permet, de temps à autre dans la recherche, une sélection du voisin disposant de la plus forte probabilité de sélection plutôt qu'un tirage probabiliste. Ce paramètre a permis d'apporter un léger gain en termes de convergence.

Recherche locale L'algorithme d'amélioration de solution *2-opt* (voir section 2.5.3) a été modifié afin qu'il s'applique à l'amélioration de chemins (et non de cycles). Nous l'avons inséré dans la métaheuristique, en l'appliquant à l'issue de chaque génération de solution. Cette modification est coûteuse en temps de calcul, mais apporte un gain substantiel dans la qualité des chemins. Elle supprime en effet tous les chevauchements d'arêtes, généralement nombreux au cours des premières générations, ce qui améliore sensiblement le coût et donc la qualité de la solution. Au final, le nombre de cycles nécessaires à la convergence des colonies de fourmis est fortement réduit.

Réduction du voisinage Nous avons implémenté une méthode de réduction de voisinage des nœuds. Cette méthode connue [Lawl 85, Rein 94] est utile dans le cas de grandes instances de problèmes (supérieures à la centaine de « villes »). Elle calcule initialement, pour chaque ville, les h plus proches villes voisines et les conserve dans une structure de données. Ce calcul

s'effectue avec une complexité en temps de $O(n^2.\log(n))$. Lors de la construction d'une solution, chaque fourmi ne va ainsi considérer que les villes les plus proches comme villes candidates. Si en revanche toutes ces villes ont déjà été parcourues, alors la fourmi évaluera l'ensemble des villes du graphe (fonctionnement classique). Cette implémentation permet un gain substantiel en temps de calcul, pour une valeur correcte de h. Cette dernière doit être suffisamment faible pour apporter un gain réel en termes de nombre d'évaluations, et ne pas nécessiter un espace mémoire trop important. Elle doit également être suffisamment importante pour éviter d'isoler trop souvent des nœuds (aboutissant à évaluer l'ensemble des nœuds candidats).

Renforcement global Dans notre implémentation, le renforcement global ne considère que la meilleure solution *locale*. Ce choix n'est pas anodin, et motivé dans la section 3.2.4. L'utilisation d'une heuristique d'amélioration locale après la construction d'une solution assure néanmoins que les solutions locales seront de relativement bonne qualité.

Renforcement local Nous avons implémenté le mécanisme de renforcement local proposé dans *Ant Colony System*. Nous ne sommes cependant pas parvenus à obtenir de jeu de paramètres permettant d'observer un gain des performances, malgré les préconisations de paramétrage des auteurs, et l'avons par conséquent abandonné.

Taux de phéromone minimal De manière similaire à *MAX-MIN Ant System*, nous avons borné la valeur minimale des taux de phéromone. Ce choix n'a au départ pas vocation à améliorer la recherche, mais permet simplement de se prémunir des valeurs nulles des variables τ dans le cas d'évaporations successives trop importantes. Ce cas est en effet induit par le mécanisme de réduction de voisinage, qui va pouvoir inhiber les arêtes appartenant aux solutions renforcées. Sans ce mécanisme, le renforcement global garantit qu'au moins deux arêtes de chaque nœud verront leur taux de phéromone incrémenté, et par conséquent que la probabilité d'élection de chaque nœud voisin ne risque pas d'aboutir à une division par zéro.

Paramétrage retenu Au final, nous avons retenu le paramétrage suivant, pour des instances allant jusqu'à 150 nœuds (nous n'avons pas évalué les paramètres au-delà) :

- nombre de cycles : $c = 2000$ (« temps » observé pour la convergence de l'algorithme avec les paramètres ci-après) ;
- nombre de fourmis : $m = 10$ (préconisation des travaux de M.Dorigo) ;
- nombre de voisins : $h = 15$ (préconisation des ouvrages traitant du sujet, conseillant une valeur de l'ordre de la dizaine de voisins) ;
- $\tau_0 = 0,001$ (valeur plus élevée que les préconisations des travaux de M. Dorigo) ;
- $\alpha = 1$ (α est ici fixé à 1, comme dans *Ant Colony System*) ;
- $\beta = 3$ (valeur plus faible que M. Dorigo, qui fixe $\beta = 5$) ;
- $\rho = 0,05$ (valeur plus faible que M. Dorigo, qui fixe $\rho = 0,1$) ;
- $q_0 = 0,9$ (emprunté à *Ant Colony System*) ;
- $Q = 1$ (préconisation des travaux de M. Dorigo).

3.2.3 Méthodologie d'expérimentation

Afin de comparer expérimentalement les trois mécanismes de construction proposés en section 3.2.1, nous avons choisi d'utiliser un *benchmark* de la littérature. Nous avons donc emprunté une instance de test de la célèbre librairie TSPLIB [Rein 08] dédiée aux problèmes de voyageur de

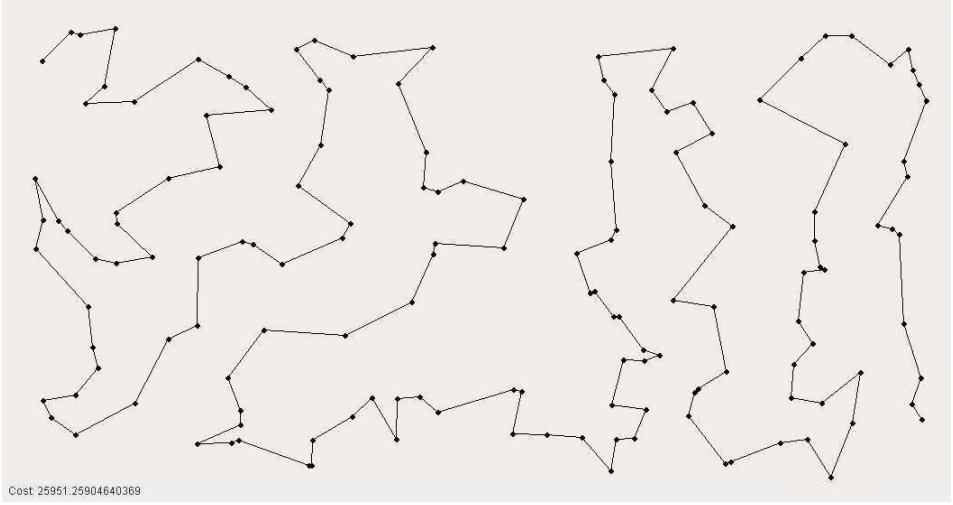


FIGURE 3.4 – Solution optimale au problème de plus court chemin hamiltonien pour l'instance *kroA150*, avec $v_s = 128$ et $v_g = 40$.

commerce : l'instance *kroA150*. Nous avons choisi cette instance car elle contient 150 villes, un nombre suffisamment élevé pour permettre de distinguer les performances des différentes approches, sans toutefois être trop élevé pour ne pas crouler sous les charges de calcul. Sauf pour des besoins expérimentaux éventuels, nos problèmes n'atteindront pas une telle quantité de points obligatoires (car il n'est pas envisageable d'exiger une solution optimale en quelques secondes). L'autre intérêt de cette instance est que la répartition des villes est relativement homogène, ce qui traduit mieux la réalité des scénarios que nous aurons à traiter.

On notera qu'à la différence d'un problème de voyageur de commerce, dont la solution optimale est unique pour chaque instance, il existe un grand nombre de solutions optimales pour un problème de chemin hamiltonien minimum, puisqu'il existe exactement $\frac{n.(n-1)}{2}$ (pour le problème considérant les distances symétriques) combinaisons selon les points de départ et d'arrivée choisis. De plus, bien que disposant d'une arête de moins, le coût d'une solution optimale peut être supérieur à celui de la solution optimale du voyageur de commerce (l'infériorité n'est garantie que si v_s et v_g sont voisins dans la solution optimale). Pour notre instance *kroA150*, nous avons choisi (arbitrairement) de prendre les nœuds $v_s = 128$ et $v_g = 40$ (en utilisant un indexage démarrant à 0), déterminés en calculant :

$$v_s = \{v \in V' |\ \forall w \in V' \backslash \{v\},\ d(v, v_0) < d(w, v_0)\} \text{ avec } v_0 = \left\{ \min_{v=\{x,y\} \in V'} \{x\}\ ,\ \min_{v=\{x,y\} \in V'} \{y\} \right\}$$

et

$$v_g = \{v \in V' |\ \forall w \in V' \backslash \{v\},\ d(v, v_f) < d(w, v_f)\} \text{ avec } v_f = \left\{ \max_{v=\{x,y\} \in V'} \{x\}\ ,\ \max_{v=\{x,y\} \in V'} \{y\} \right\}$$

Les nœuds v_0 et v_f correspondent aux nœuds situés aux extrémités supérieure gauche et inférieure droite de l'ensemble des villes respectivement. La solution optimale à ce problème, déterminée expérimentalement, est un chemin de distance $d = 25951,25$ (voir figure 3.4).

Pour l'expérimentation, nous avons mis en place une méthodologie générique de test, valable pour l'analyse de tout algorithme de type *anytime*, et dont l'architecture fonctionnelle est

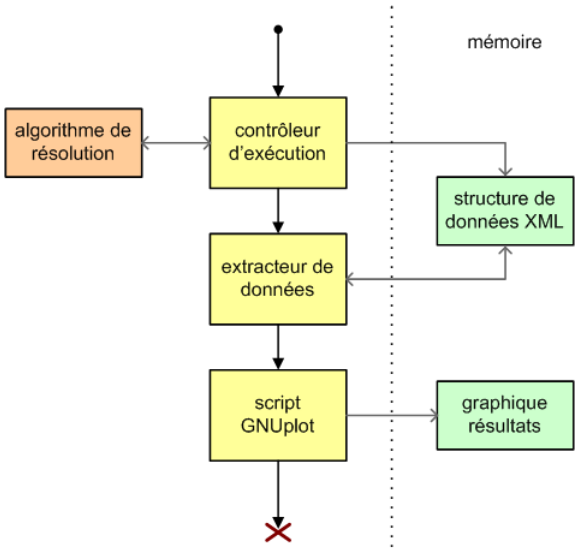


FIGURE 3.5 – Structure fonctionnelle du programme de test réalisé. Les résultats issus des simulations sont conservés en mémoire afin de pouvoir être comparés ou réinterprétés avec des critères différents.

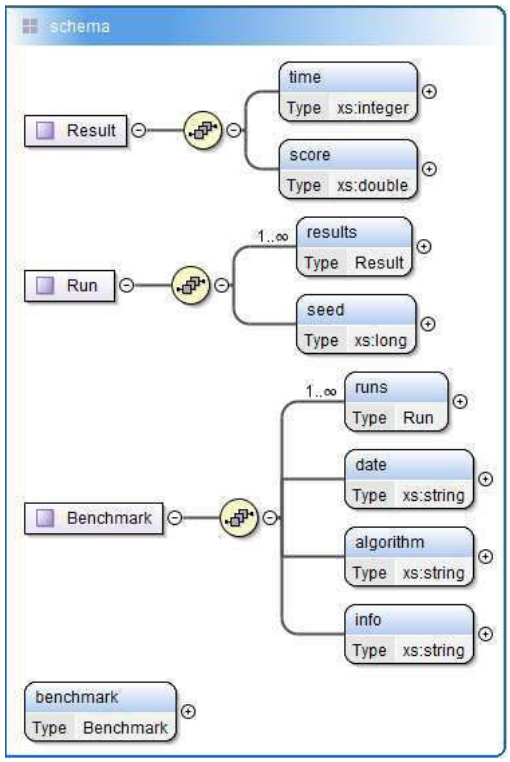


FIGURE 3.6 – Schéma des structures de données XML pour l'expérimentation (impression-écran du logiciel *<oXygen/> XML Editor*).

présentée en figure 3.5. Ce programme est constitué d'un contrôleur d'exécution, ici chargé de lancer successivement 150 résolutions de l'algorithme, pour chaque mécanisme de construction proposé. Pour conserver une trace de ces résolutions, nous avons défini une structure de données, présentée par la figure 3.6. Le type *Benchmark* correspond à la sauvegarde de l'ensemble des données de 150 exécutions, pour un même algorithme (c'est-à-dire l'algorithme des colonies de fourmis avec un mécanisme de construction donné). Cette structure contient une liste d'éléments de type *Run* correspondant à une exécution de l'algorithme. Un élément de type *Run* est utilisé au cours d'une exécution et permet de sauvegarder, chaque fois qu'une nouvelle solution est trouvée, la valeur de celle-ci (champ *score*) ainsi que la date correspondante (champ *time*). Ce champ *time* peut être employé pour mémoriser le délai (en secondes) d'obtention de la solution depuis le lancement de l'algorithme, ou bien le numéro de l'itération au cours d'une recherche. En pratique, on préfèrera les itérations car elles permettent de comparer les résultats générés depuis des stations de calcul différentes. Pour conserver la notion de temps de calcul, on pourra garder une trace du temps de calcul total de l'algorithme pour être capable de déduire le temps moyen par itération (temps par cycle dans le cas des colonies de fourmis). On conserve en outre en mémoire la graine (champ *seed*) utilisée pour déterminer les valeurs aléatoires afin d'être capable de rejouer *a posteriori* les scénarios de manière identique. Une fois les recherches achevées, le contrôleur passe la main à un extracteur de données qui va parcourir les données sauvegardées pour générer un graphique à l'aide d'une librairie *GNUplot*.

3.2.4 Résultats et discussion

3.2.4.1 Comparaison des méthodes de construction

Les résultats comparatifs sont donnés par la figure 3.7. On note que la construction unidirectionnelle est la moins performante des trois méthodes proposées. Ce résultat était prévisible, puisque cette approche souffre d'un défaut : celui d'évaluer tous ses chemins depuis la même origine, pouvant aboutir au cas présenté par la figure 3.8. Le nœud v_g est ajouté en fin de séquence quelle que soit la solution, et ne « profite » donc pas de l'heuristique de guidage. Seul le caractère purement aléatoire peut ainsi permettre à l'algorithme de trouver l'optimum global. Avec la construction bidirectionnelle, on peut contrecarrer ce défaut : en démarrant une recherche sur deux depuis v_g, la recherche permet d'explorer différemment l'espace de recherche en s'appuyant sur l'heuristique de guidage, aboutissant à une meilleure exploration. Le choix de renforcer à chaque cycle la meilleure solution *locale*, et non globale, prend ici tout son sens : elle force à réaliser un apprentissage diversifié (renforcer uniquement la meilleure solution globale peut amener à retomber dans les travers de la construction unidirectionnelle).

De manière moins prévisible, les résultats montrent également que les performances de la construction bidirectionnelle surpassent la construction avec initialisation aléatoire. On peut donc supposer que le guidage de la construction bidirectionnelle aboutit à un apprentissage davantage efficace pour les zones proches des points de départ et d'arrivée. Cette conclusion justifie l'emploi d'un renforcement global de la meilleure solution *locale*. Ainsi, on contraint l'algorithme à renforcer un chemin qui, alternativement, part une fois sur deux du point de départ, et une fois sur deux du point d'arrivée.

Pour se convaincre que ces résultats ne sont pas propres à l'instance choisie, nous avons réalisé la même expérimentation en générant une série d'instances distinctes (c'est-à-dire bénéficiant de points de départ et d'arrivée différents). Nous avons donc répété la méthodologie utilisée ci-avant, en modifiant légèrement la structure de données afin de conserver en mémoire les points de départ et d'arrivée de chaque instance (intégrés au type *Run*). Au final, nous avons donc

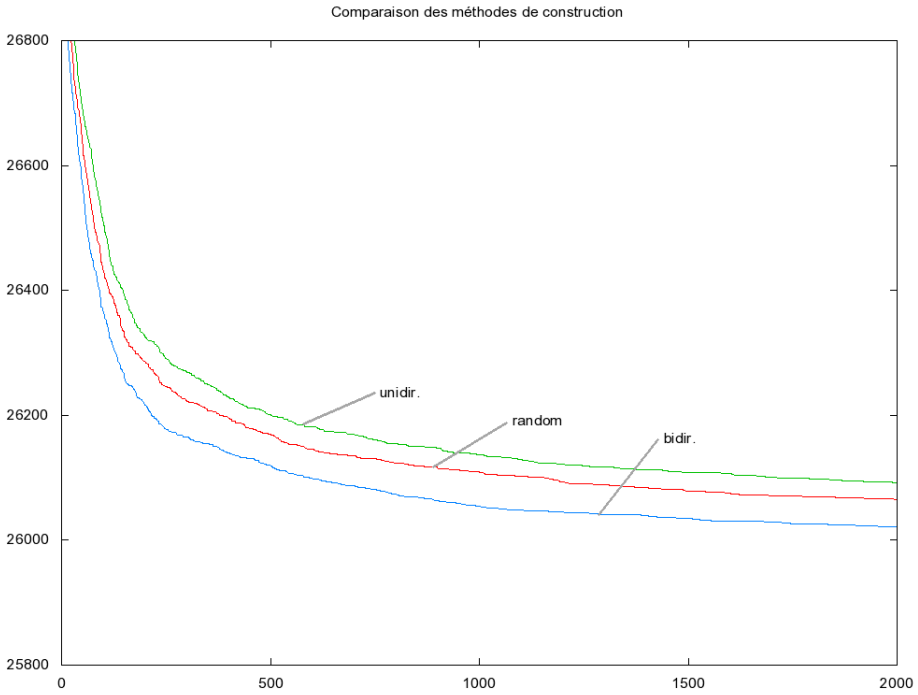


FIGURE 3.7 – Comparaison des performances de l'algorithme pour les trois mécanismes de construction présentés : construction unidirectionnelle (*unidir.*), construction bidirectionnelle (*bidir.*) et construction avec initialisation aléatoire (*random*). Le graphique présente les valeurs moyennes de coût (sur 150 exécutions, en unités de distance) des meilleures solutions trouvées en fonction du nombre de cycles de l'algorithme.

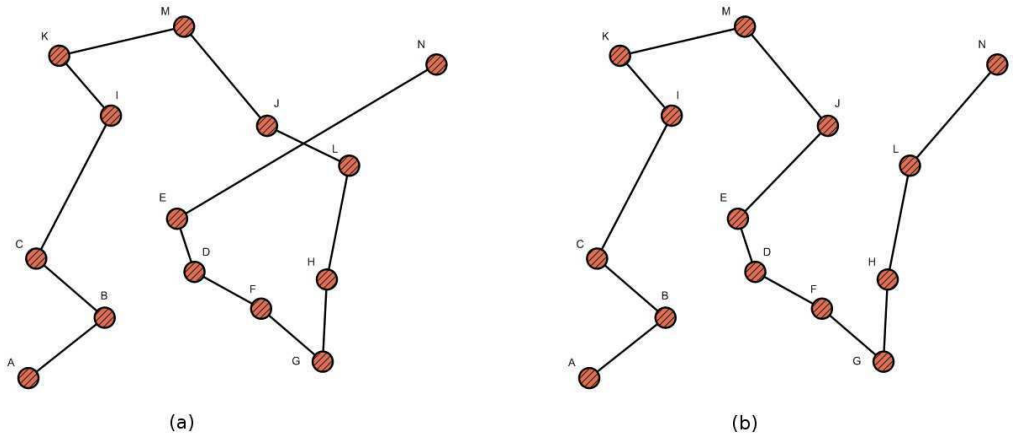


FIGURE 3.8 – Exemple type de solution optimale locale rencontrée par la méthode de construction unidirectionnelle.

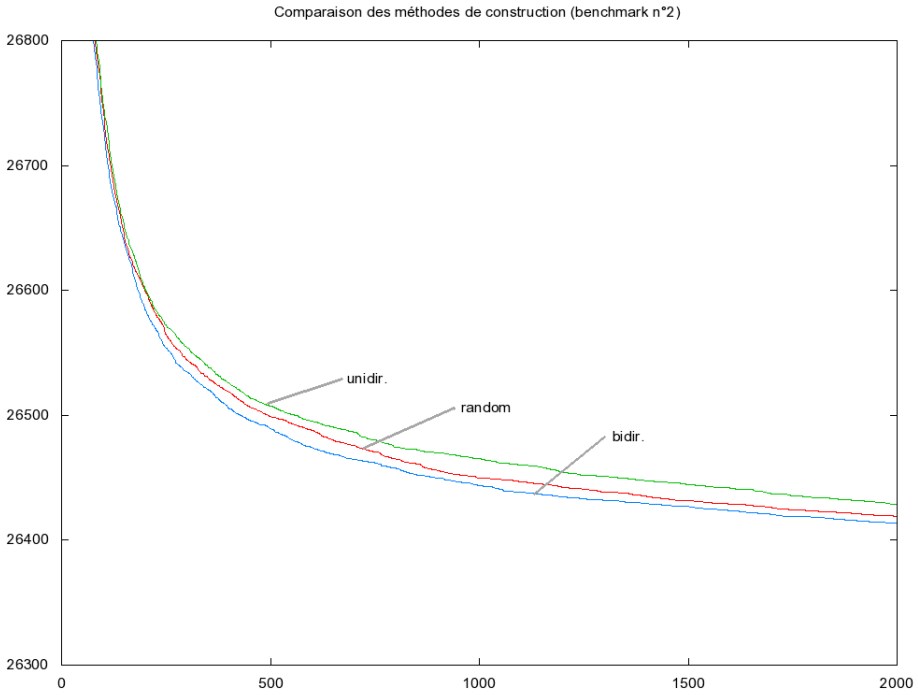


FIGURE 3.9 – Comparaison des performances de l'algorithme pour les trois mécanismes de construction présentés : construction unidirectionnelle (*unidir.*), construction bidirectionnelle (*bidir.*) et construction avec initialisation aléatoire (*random*). Le graphique présente les valeurs de coût (moyennées sur 500 instances distinctes, en unités de distance) des meilleures solutions trouvées en fonction du nombre de cycles de l'algorithme.

testé les approches sur 500 instances distinctes, générées aléatoirement, de manière unitaire. Les résultats sont donnés par la figure 3.9.

Les résultats sont donc cohérents avec ceux obtenus précédemment. Bien que l'écart soit moindre que précédemment, on observe bien une supériorité des performances moyennes de la méthode par construction bidirectionnelle face à la méthode à initialisation aléatoire. Dans le cas de problèmes de chemins hamiltoniens, il est donc préférable de ne pas suivre « à la lettre » la philosophie de l'approche de colonies de fourmis développée dans la littérature pour le problème de voyageur de commerce.

3.2.4.2 Comparaison de notre approche avec des approches concurrentes

Dans le graphique de la figure 3.9, nous pouvons également observer que l'algorithme n'a pas nécessairement totalement convergé à l'issue des 2000 cycles. Nous nous sommes alors demandés si notre approche de colonies de fourmis était davantage efficace que d'autres métaheuristiques. Afin de répondre à cette interrogation, nous avons souhaité développer une autre métaheuristique réputée efficace pour résoudre notre problème, et comparer les performances en termes de qualité de convergence sur une base de temps commune. Nous avons opté pour une méthode de type ILS (*Iterated Local Search*) intégrée à un algorithme de recuit simulé. Dans cet algorithme, nous élaborons une solution initiale en utilisant une heuristique de plus proche voisin (voir section 2.5.2). Puis nous démarrons la méthode de recuit simulé, dont nous avons paramétré la température initiale à la valeur $T_0 = 750$. Nous employons également une loi de décroissance

continuelle telle que $T_{i+1} = \lambda.T_i$ avec $\lambda = 0,999$. Pour la condition d'arrêt, la température pour lequel le système est supposé figé est fixée à $T_f = 0,001$. A chaque itération, l'algorithme va perturber la solution courante en effectuant deux opérations de permutation. Chaque permutation consiste à intervertir la position de deux villes dans la séquence, celles-ci étant déterminées aléatoirement. Afin de conserver la validité de la solution (chemin allant de v_s à v_g), les villes situées en tête et queue de la séquence ne peuvent être sélectionnées dans cette opération. A l'issue de ces permutations, un algorithme *2-opt* est utilisé pour déterminer une nouvelle solution correspondant à un optimum local. Si le coût de cette nouvelle solution est meilleur que la nouvelle solution courante, elle est acceptée et mémorisée. Sinon, la solution peut être néanmoins conservée si le tirage aléatoire réalisé satisfait l'équation de Boltzmann.

La méthodologie d'expérimentation de la section précédente a été appliquée à l'algorithme de recuit simulé. De manière similaire à l'approche de colonies de fourmis, 150 exécutions ont été réalisées afin d'obtenir les valeurs de coût moyennes par cycle. Il est à noter que l'algorithme de recuit simulé, paramétré selon les valeurs définies ci-avant, réalise 13521 cycles au total. Afin de comparer les deux approches sur une base de temps identique, les temps de cycles de chaque algorithme ont été relevés, *sur une même machine*. Pour des questions de précision (nos systèmes d'exploitation n'étant pas des systèmes temps-réel non préemptifs), les temps de calcul ont été réalisés sur la base de 1000 cycles pour les colonies de fourmis, et de 5000 cycles pour la méthode de recuit simulé. Les résultats obtenus sont, sur un ordinateur portable équipé d'un processeur *Intel Core 2 Duo P7350* (cadencé à $2GHz$) et de 3 Go de mémoire vive, respectivement de $66,832s$ et $27,607s$, soit environ $66,8ms$ par cycle pour les colonies de fourmis et $5,5ms$ par cycle pour le recuit simulé. Les courbes de performance, rapportées sur une abscisse temporelle identique, sont illustrées par la figure 3.10.

Les résultats de la comparaison montrent que l'algorithme de recuit simulé développé, malgré sa grande simplicité, montre d'étonnantes performances puisqu'il surpasse notre algorithme de colonies de fourmis. Ce résultat, décourageant au premier abord, a permis d'aboutir à une réflexion sur la manière d'améliorer les colonies de fourmis. Puisque les méthodes ILS ont prouvé leur efficacité, nous avons cherché une manière de les introduire dans notre approche. Nous l'avons finalement réalisé à l'aide d'une méthode simple, préexistante dans la littérature [Dori 04] : distinguer deux catégories de fourmis. La première catégorie (dite de « fourmis de construction »), similaire à celles employées jusqu'à maintenant, élabore des solutions en tirant bénéfice de l'apprentissage. La seconde catégorie (dite de « fourmis d'amélioration ») va utiliser la meilleure solution globale et la perturber (nous reprenons pour cela la fonction de perturbation mise en place dans la méthode de recuit simulé), avant d'appliquer une méthode *2-opt*. Cette approche vise ainsi à combiner l'efficacité des approches ILS avec la faculté d'apprentissage des colonies de fourmis, afin de ne pas rester bloqué dans un optimum local. Nous avons expérimenté cette approche sur le benchmark précédent et observé les résultats présentés en figure 3.11, que l'on compare aux approches précédentes.

L'approche de colonies de fourmis obtenue montre une nette supériorité en termes de performances moyennes. Elle tire parfaitement parti à la fois de l'approche ILS et de l'apprentissage, en ayant la capacité de converger rapidement vers la solution optimale. On observe que c'est la seule approche qui converge bien vers l'asymptote de coût 25951,25 (coût de la solution optimale) *sur l'ensemble des exécutions* avant le terme des 2000 cycles. S'agissant d'une moyenne des coûts, cela signifie que *toutes* les recherches trouvent bien la solution optimale. De plus, elle est capable d'offrir une solution 0,1%-optimale (c'est-à-dire de coût $c < (1 + 0,1\%).c^\star$ où $c^\star$ est le coût de la solution optimale) dans un temps moyen de 12 secondes environ, ce qui est tout à fait satisfaisant. Deux résultats sont en outre intéressants. En premier lieu, la répartition

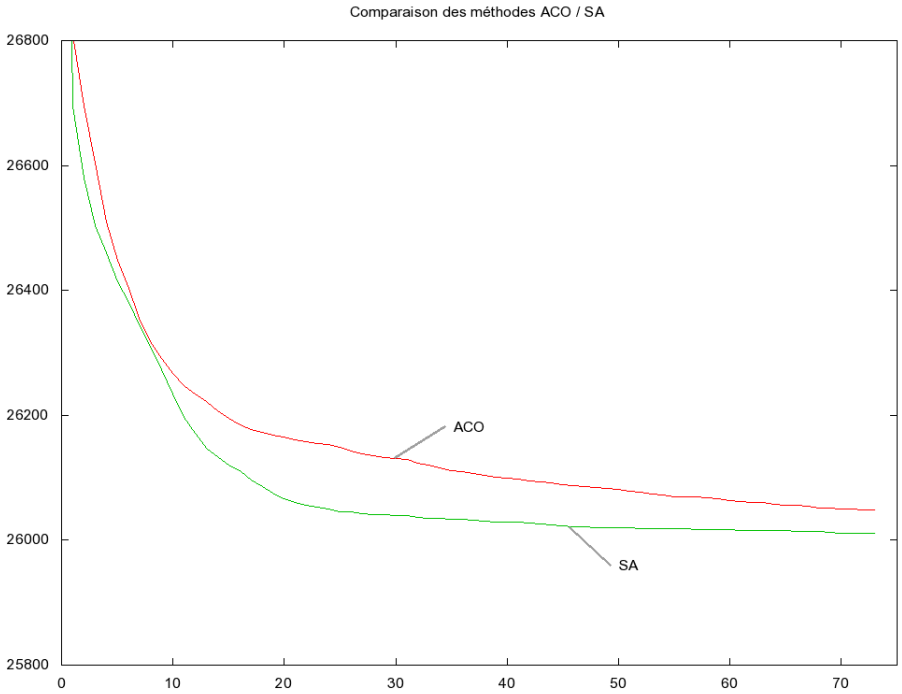


FIGURE 3.10 – Comparaison des performances des algorithmes de colonies de fourmis (ACO) et de recuit simulé (SA). Le graphique présente les valeurs moyennes de coût (sur 150 exécutions, en unités de distance) des meilleures solutions trouvées en fonction du temps (en secondes).

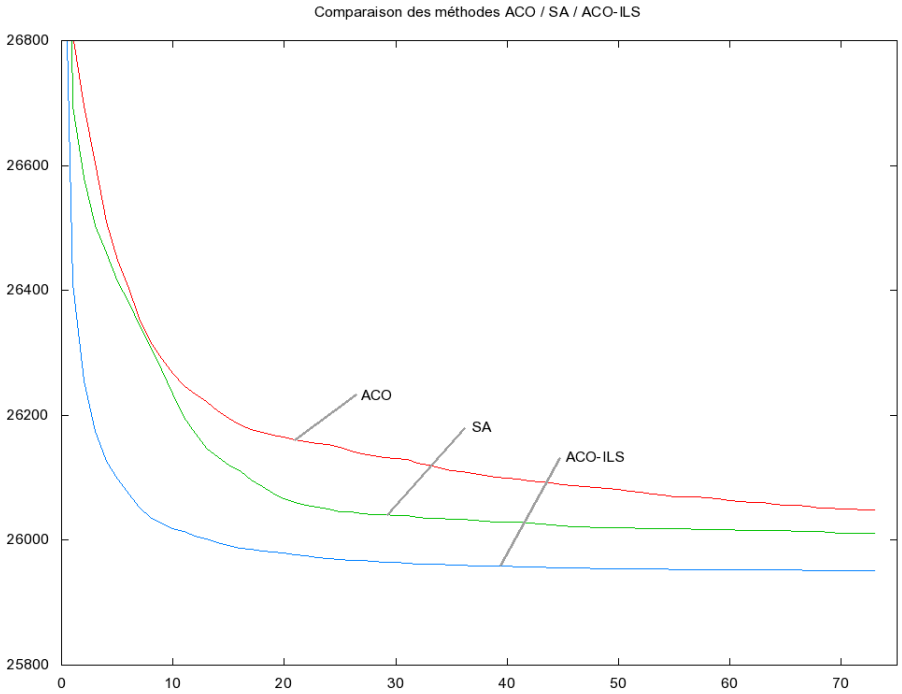


FIGURE 3.11 – Comparaison des performances des algorithmes de colonies de fourmis (ACO), de recuit simulé (SA), et de colonies de fourmis avec fourmis d'amélioration (ACO-ILS). Le graphique présente les valeurs moyennes de coût (sur 150 exécutions, en unités de distance) des meilleures solutions trouvées en fonction du temps (en secondes).

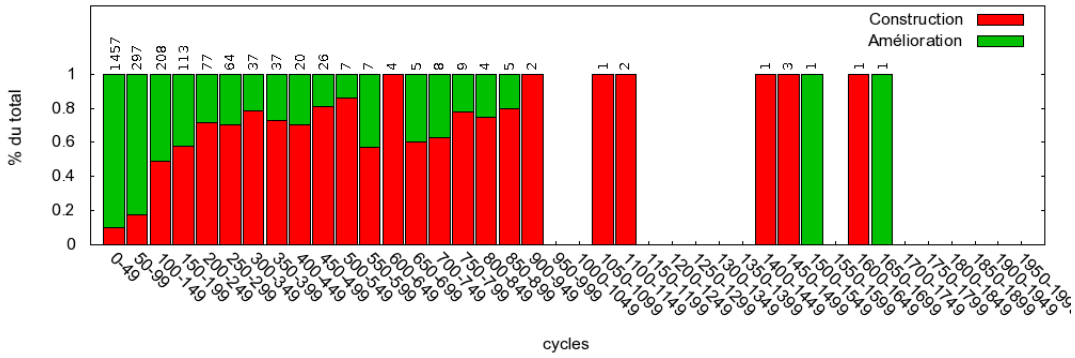


FIGURE 3.12 – Proportion moyenne (sur 150 exécutions) du type de fourmis participant à l'amélioration de la meilleure solution, pour chaque série de 50 itérations de l'algorithme. Les valeurs indiquées au-dessus de l'histogramme correspondent au nombre total d'améliorations réalisées au cours des 150 exécutions, pour chaque série de cycles. L'absence d'histogramme traduit une absence d'amélioration de la solution dans la tranche de cycles concerné.

des effectifs de chaque catégorie de fourmis participant à l'amélioration de la solution, donnée par la figure 3.12, montre que les fourmis d'amélioration ont un rôle prépondérant dans les premiers cycles de la recherche ; les fourmis de construction, bénéficiant de l'apprentissage, prennent ensuite l'ascendant en mutualisant sur l'apprentissage réalisé. En deuxième lieu, cette implémentation gomme les différences de performances entre les méthodes de construction à initialisation aléatoire et de construction bidirectionnelle. Les résultats observés à l'issue d'expérimentations complémentaires montrent que les courbes de performance sont confondues (nous n'avons par conséquent pas jugé utile de les présenter).

3.3 Intégration des colonies de fourmis dans la résolution globale

L'algorithme présenté en section précédente a été intégré dans l'approche de résolution globale, présenté en figure 3.2. Pour l'étage de résolution partielle, nous ne détaillerons pas les étapes de modélisation du problème (qui consiste à extraire les données du problème et à calculer les plus courtes distances pour générer le graphe G') et de reconstruction de la solution (qui concatène une série de plus courts chemins). Quant à la résolution globale, hormis la suppression de l'étape de calcul de plus court chemin, elle reste pour le moment inchangée.

Les sections suivantes détaillent la comparaison de l'approche initiale avec notre approche hybride.

3.3.1 Méthodologie d'analyse et d'expérimentation

Analyser les performances de résolution d'un solveur de programmation logique avec contraintes (PLC) n'est pas simple, car les mécanismes de propagation de contraintes sont réalisés de manière transparente pour l'utilisateur. Sous l'environnement de compilation *SICStus Prolog*, que nous utilisons pour nos travaux, trois paramètres sont cependant mis à disposition :

– un paramètre *time*, afin de connaître le temps processeur à un temps t dans la recherche ;

– un paramètre *backtrack*, afin de connaître le nombre de retours sur trace opérés par l'algorithme de recherche ;
– un paramètre *memory*, qui précise la quantité de mémoire utilisée pour la recherche.

Le paramètre de quantité mémoire consommée ne sera que peu regardé car, bien qu'embarquée, notre application est peu critique à cet égard. De plus, le compilateur *SICStus* n'est pas capable de fonctionner avec plus de 256Mo (limite interne d'adressage). Cette limite est cependant conforme à nos prérequis techniques : nous ne nous soucierons donc de ce critère qu'à but informatif. Nous évaluerons les différentes approches à l'aide des paramètres *time* et *backtrack*. Comme nous l'avions évoqué en section 3.2.3, la valeur de temps étant dépendante d'un grand nombre de critères — fréquence d'horloge et quantité de cache du processeur, système d'exploitation, préemptibilité du programme, quantité et temps d'accès à la mémoire vive... —, il n'est pas envisageable de comparer les valeurs de temps de calcul de différentes approches de PLC exécutées sur deux machines différentes. Or, dans le cas d'une résolution en programmation logique, il n'est pas possible de compabiliser un nombre de cycles comme précédemment. Nous disposons du nombre de retours sur trace, mais là encore, ces derniers ne sont pas nécessairement équivalents à un simple compteur d'appel à une fonction de programmation impérative. En effet, ils ne sont pas le reflet des temps de propagation pouvant être induits, ces derniers pouvant être très variables. En conclusion, pour une comparaison fiable, les comparaisons devront être réalisées sur une même station de calcul. De même, les valeurs de temps et de retour sur trace devront être analysées de manière découplée.

La méthodologie de test est identique à celle réalisée pour les comparatifs des algorithmes de colonies de fourmis (voir figure 3.5). Seule la structure de données, pour la sauvegarde des résultats, a été modifiée. Un aperçu de cette structure est donné par la figure 3.13. Plusieurs différences sont à noter. Dans cette structure, le type *SolvingData* rassemble les résultats d'une exécution. Ces résultats présentent les données suivantes :
– le temps (*optTime*) et le nombre de retours sur trace (*optBacktracks*) pour *trouver* la solution optimale ;
– le temps (*proofTime*) et le nombre de retours sur trace (*proofBacktracks*) pour achever l'exécution, c'est-à-dire *prouver* que la dernière solution est la solution optimale ;
– le coût de la solution optimale (*score*).

Le solveur de contraintes a un comportement déterministe : pour son analyse, une exécution suffit. Pour étudier les comportements du solveur pour chaque sonde, il est donc nécessaire de le confronter à plusieurs configurations différentes. Le type *Result* nous permet de stocker les résultats d'une exécution pour une configuration précise, comportant :
– le nœud de départ (*startNode*) ;
– le nœud d'arrivée (*endNode*) ;
– la liste éventuelle des nœuds obligatoires (nommés *inclusions*).

Le type *Results*, enfin, permet de conserver les résultats du solveur pour un ensemble de configurations, sur une instance donnée.

Le *benchmark* réalisé porte sur trois scénarios de test issus de cas réels, fournis par *Sagem* :
– scénario 1 : 23 nœuds, 76 arcs ;
– scénario 2 : 22 nœuds, 68 arcs ;
– scénario 3 : 22 nœuds, 74 arcs.

Pour chaque instance, nous avons mis en place 5 jeux de configurations, chaque jeu comportant 10 configurations. Chaque jeu de configuration correspond à un nombre donné de points obligatoires, de 1 à 5. L'annexe 5.3.4 détaille les instances ainsi que les configurations mises en place. Pour chacune des instances, nous avons comparé, selon le nombre de points obligatoires :

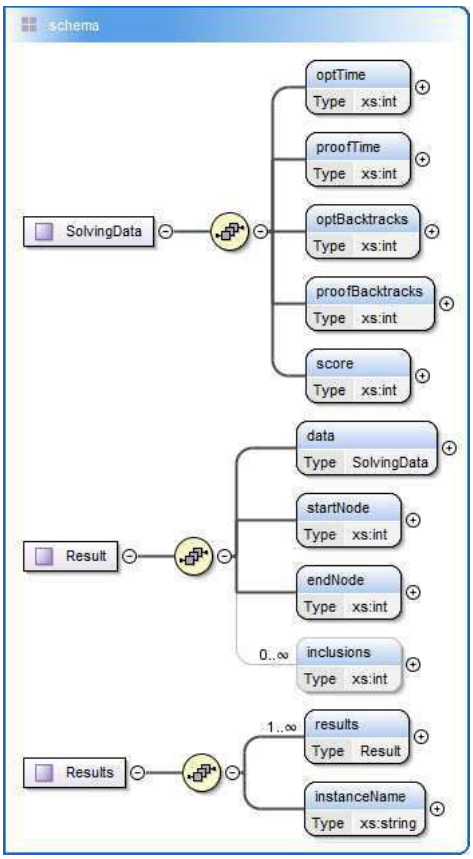


FIGURE 3.13 – Schéma des structures de données XML pour l'expérimentation du solveur de contraintes (impression-écran du logiciel *<oXygen/> XML Editor*).

– le temps de calcul moyen et maximum d'obtention de la solution optimale ;
– le temps de calcul moyen et maximum de preuve d'optimalité ;
– le nombre de retours sur trace moyen et maximum d'obtention de la solution optimale ;
– le nombre de retours sur trace moyen et maximum de preuve d'optimalité.

Trois approches ont été comparées :

– une approche *REF* (référence) sans guidage (pas de tri des variables) ;
– une approche *SP* (*shortest path*) de guidage avec plus court chemin ;
– une approche *ANTS* de guidage avec colonies de fourmis.

Les résultats sont fournis dans la section suivante.

La taille des instances ainsi que le faible nombre de points de passage ne justifient pas ici l'emploi d'un algorithme de résolution partielle telle qu'ACO, en raison du faible nombre de combinaisons possibles de séquences de points de passage. Ces tailles de graphe nous sont imposées en raison du modèle de chemin employé au sein du solveur de contraintes. Avec ce modèle, qui a l'avantage d'être générique, il devient en effet difficile d'obtenir une preuve d'optimalité en un temps raisonnable pour des graphes dont la taille dépasse une trentaine de nœuds. Des essais ont été bien réalisés sur de grandes instances (telles que celle de la section 3.4.4), permettant d'aboutir à une solution après quelques secondes (le temps de propagation des variables étant ici sensible), mais l'absence d'optimalité et les temps d'exploration de l'espace d'état ne justifie plus l'emploi d'une telle approche. Le choix de la modélisation du solveur de contraintes étant imposé, et hors du cadre de cette thèse, nous devons ici nous y restreindre. Cependant, le choix d'ACO reste cohérent à terme en perspective d'une mise à l'échelle avec une nouvelle modélisation.

Pour cette expérimentation, nous traitons un problème sans contrainte de capacité. Ainsi, nous pouvons avoir un meilleur aperçu de l'efficacité intrinsèque du guidage. Cependant, bien que dépourvu de contrainte, les variables sont présentes dans le modèle (initialisées dans un domaine $[0, 10000]$) et propagées le long du graphe. Les variables de temps sont exprimées en secondes, et la vitesse des véhicules est limitée à $30km/h$ (progression urbaine).

3.3.2 Résultats et discussion

Les expérimentations ont été menées sur un ordinateur portable *Samsung Q310*, doté d'un micro-processeur *Intel Core 2 Duo P7350* cadencé à 2GHz et de 3Go de mémoire vive. La figure 3.14 présente les résultats de temps de calcul pour chacune des approches, sur chacune des trois instances (horizontalement). Les graphiques de la colonne de gauche décrivent les temps d'obtention de la solution optimale, ceux de droite les temps de preuve d'optimalité. Les histogrammes présentent le temps moyen, en millisecondes et sur 10 exécutions, pour chaque jeu de configurations. Les barres d'erreurs sont ici utilisées pour représenter le temps de calcul maximum de la série associée. Structurés de manière analogue, les résultats de nombre de retours sur trace sont présentés par la figure 3.15.

Nous nous intéressons en premier lieu au temps de calcul d'obtention des solutions optimales. Tout d'abord, on note que la méthode sans guidage présente des performances très médiocres sur les instances 1 et 2 (jusqu'à 35 secondes dans le cas de l'instance 1, ce qui n'est pas représenté ici). Si elle est *a contrario* la meilleure sur la 3^{e} instance, la très grande variabilité en termes d'efficacité en font une méthode à proscrire dans le cadre applicatif de nos travaux. Concernant les méthodes avec guidage, on note que les performances moyennes des deux méthodes sont très bonnes sur les instances 1 et 2. On observe que l'approche *SP* présente néanmoins des temps de calcul maximum sensiblement plus élevés que l'approche *ANTS* sur plusieurs jeux de configurations. Sur l'instance 3, les performances moyennes sont légèrement en deça pour

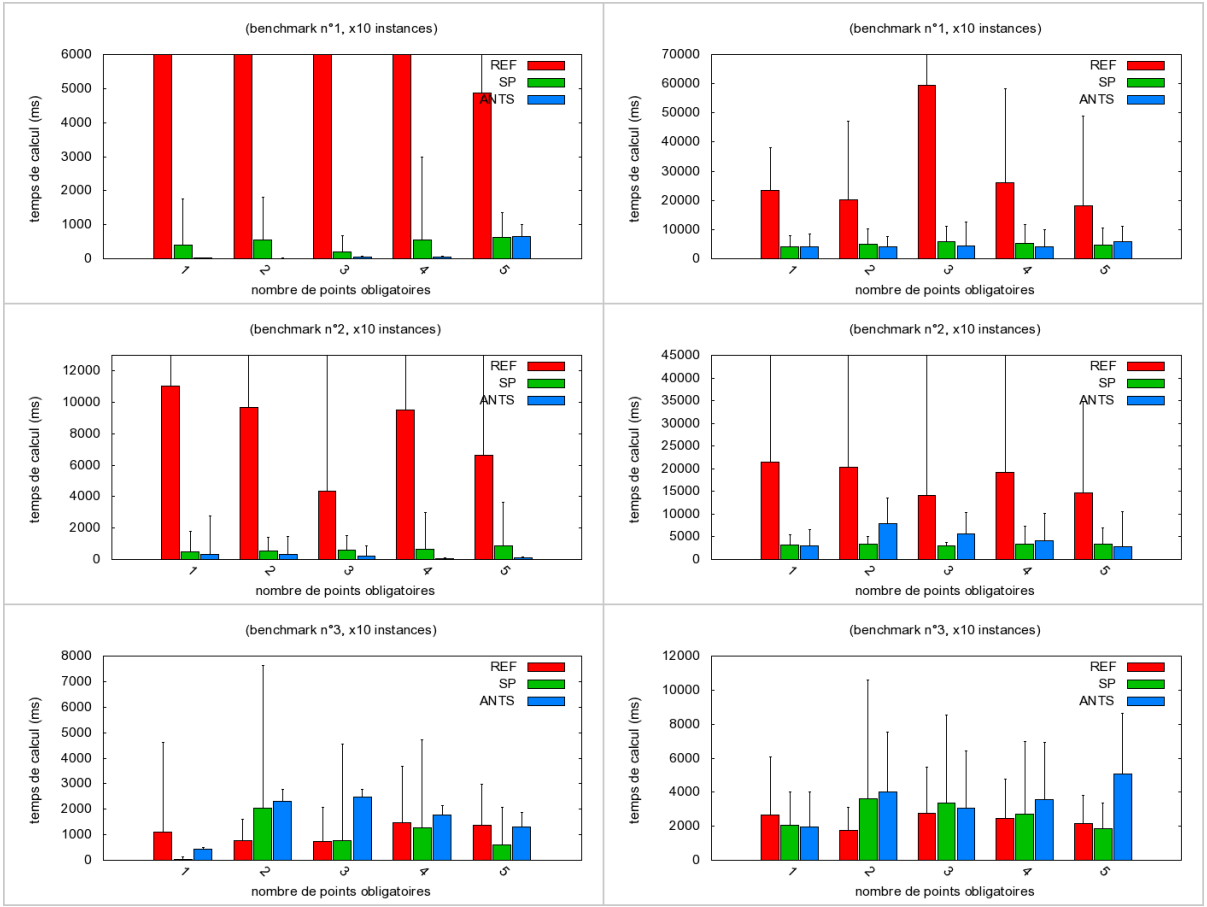


FIGURE 3.14 – Temps de calcul moyens (histogrammes) et maximum (barres d'erreur) des trois approches REF (sans guidage), SP (guidage par plus court chemin simple) et ANTS (guidage par colonies de fourmis) sur les *benchmarks* 1,2, et 3 (horizontalement). La colonne de gauche présente les temps d'obtention de la solution optimale ; la colonne de droite présente les temps d'obtention de la preuve d'optimalité. Les temps sont donnés en millisecondes.

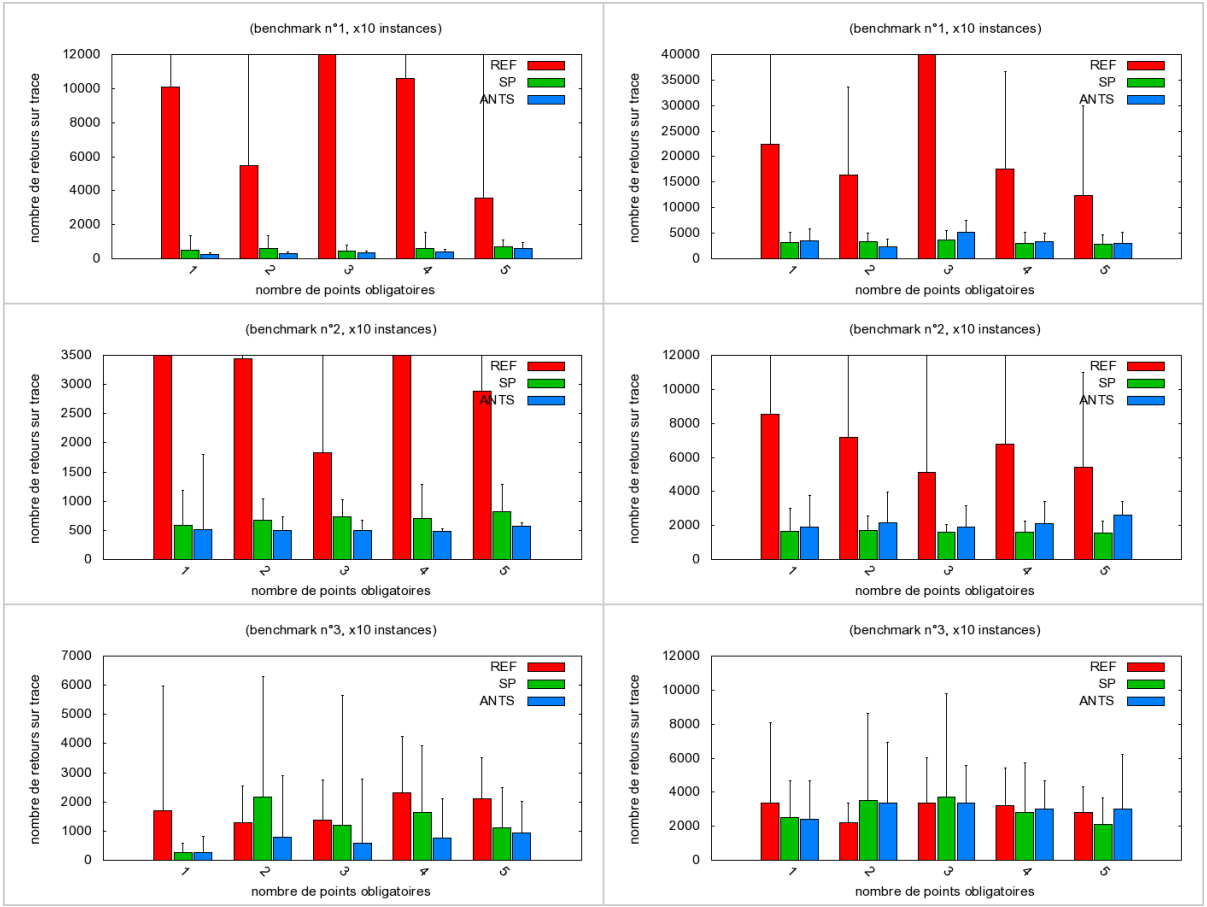


FIGURE 3.15 – Nombre de retours sur trace moyen (histogrammes) et maximum (barres d'erreur) des trois approches REF (sans guidage), SP (guidage par plus court chemin simple) et ANTS (guidage par colonies de fourmis) sur les *benchmarks* 1,2, et 3 (horizontalement). La colonne de gauche présente les retours sur trace pour l'obtention de la solution optimale ; la colonne de droite présente les retours sur trace pour l'obtention de la preuve d'optimalité.

l'approche *ANTS*. Cependant, encore une fois, les temps de calcul maximum sont bien mieux « maîtrisés ».

En analysant cette fois les temps de calcul d'obtention des preuves d'optimalité, on peut noter que les écarts de performances sont moins importants. Pour l'approche sans guidage, le même constat qu'auparavant peut être fait : une très bonne efficacité sur l'instance 3, mais des performances en deça pour les instances 1 et 2. Pour les méthodes avec guidage, on peut noter que les temps de calcul sont globalement similaires sur l'instance 1, et à l'avantage de l'approche *SP* sur l'instance 2. Cela va à l'encontre des résultats précédents. Ainsi, il est intéressant de noter que **si *ANTS* améliore le temps d'obtention des solutions optimales, elle n'améliore cependant pas le temps d'obtention des preuves d'optimalité**. C'est pourtant ce à quoi on était en droit de s'attendre : en trouvant la solution optimale plus rapidement, la résolution par séparation-évaluation est en théorie capable d'élaguer l'arbre de recherche plus efficacement.

Sachant que l'approche *ANTS* fournit une solution partielle qui est le chemin de distance minimum parcourant l'ensemble des points obligés, et que le problème global n'est ici pas contraint, pourquoi se fait-il que la solution optimale ne soit pas trouvée immédiatement dans tous les cas ? La raison est tout simplement que la résolution partielle n'interdit pas, à l'inverse du modèle logique du problème global, de repasser par un même nœud. Dans le cas où une résolution est guidée par une solution partielle repassant par un même nœud, le guidage devient incomplet et par conséquent faillible.

3.3.3 Conclusion

En conclusion, l'approche de colonies de fourmis n'améliore pas drastiquement les performances face à l'approche de plus court chemin, cette dernière s'avérant relativement efficace sur les instances utilisées. En revanche, elle permet d'obtenir des temps d'obtention de la solution optimale plus contenus. Cette approche est ainsi **plus stable face aux modifications structurelles du problème.** En effet, ces résultats nous montrent qu'une modification mineure du problème peut modifier totalement les performances du solveur. Le meilleur exemple est celui de la méthode sans guidage, extrêmement performante sur la 3^{e} instance, et très médiocre sur les deux autres. Pourtant, les graphes des trois instances ne présentent pas de différence importante (voir annexe). **La stabilité de l'approche de colonies de fourmis est cohérente avec le cadre applicatif des travaux, car elle renforce le caractère** *anytime* **de l'approche** (en offrant davantage de garanties sur la qualité de la solution en cas de besoin immédiat).

Si les résultats des essais réalisés montrent une bonne efficacité de l'approche *SP*, c'est parce qu'elle reste appliquée à des instances de petite taille. Sur des problèmes plus grands, la méthode est bien moins performante dès lors que les points de passage obligatoires sont situés « loin » de la solution partielle (qui est plus ou moins un axe v_s–v_g en général). C'est là que l'approche *ANTS* prend tout son sens, car elle va pouvoir réaliser un guidage efficace vers chacun des points obligatoires. Pour démontrer ces propos, nous avons élaboré une quatrième instance de test, de plus grande taille cette fois. Le graphe comporte ici 33 nœuds et 130 arcs. Les jeux de configurations et une illustration du graphe sont fournis en annexe 5.3.4. L'instance étant de taille plus importante, un délai d'expiration de la recherche (*timeout*) a été mis en place et fixé à 30 secondes. La figure 3.16 présente les résultats comparatifs des approches *SP* et *ANTS* en termes de temps d'obtention d'une solution (la meilleure dans le délai imparti, et donc pas nécessairement optimale).

Les résultats montrent ici un net avantage pour l'approche *ANTS*, dont les temps d'obtention moyens de solutions sont bien plus faibles. De plus, ces temps comportent un nombre plus important de solutions optimales que la méthodes *SP*. Ainsi, sur 50 configurations différentes,

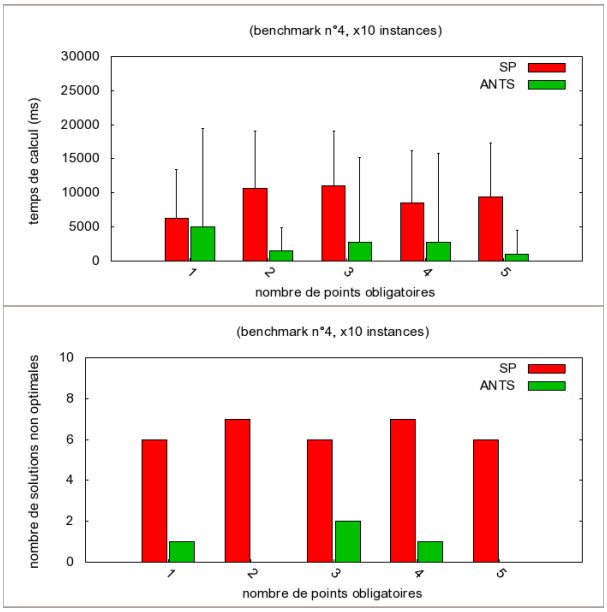


FIGURE 3.16 – HAUT : Temps de calcul moyens (histogrammes) et maximum (barres d'erreur) d'obtention de la meilleure solution (pas nécessairement optimale !) pour les approches SP (guidage par plus court chemin simple) et ANTS (guidage par colonies de fourmis) sur l'instance 4. Les temps sont donnés en millisecondes. BAS : Nombre de cas où la solution optimale n'a pas été trouvée après 30 secondes (délai d'expiration de la recherche), pour chaque jeu de configurations.

ANTS échoue à 4 reprises pour trouver la solution optimale dans le temps imparti, tandis que *SP* échoue à 32 reprises.

3.4 Aller plus loin dans le guidage

3.4.1 Limites du guidage actuel

Les résultats présentés en section précédente révèlent un gain de performance de la nouvelle approche sur les instances utilisées. Cependant, comme nous l'avions mentionné à la fin de la section 3.3.1, le problème traité n'inclut pas de contraintes sur les capacités. Les résultats ne sont donc valables que dans le cas de problèmes non ou peu contraints. Dans le cas où le problème est fortement contraint, les résultats sont en revanche beaucoup moins prévisibles. En effet, les capacités ne sont pas du tout considérées dans la résolution partielle. Or **l'efficacité d'un guidage peut être totalement anéantie par l'influence de ces capacités**. Afin de se prémunir contre cette éventualité, il faut donc se préoccuper des contraintes dès la résolution partielle.

Pour traiter des contraintes de capacité additives, il ne suffit pas d'adapter la méthode ACO. Nous avons initialement songé à modifier le mode d'apprentissage (c'est-à-dire les conditions de renforcement des taux de phéromone) d'ACO pour ne prendre en considération que les fourmis dont les solutions satisfont les contraintes. Mais cette approche est inexacte dans le sens ou les valeurs de capacité sont associées aux arcs du graphe G, et non à ceux du graphe G'. Les valeurs associées à ces derniers ne sont calculées qu'à partir des plus courts chemins entre points

obligatoires. Aussi, un plus court chemin entre deux points obligatoires A et B qui aboutisse à une non satisfaction des contraintes de capacité forcerait ACO à modifier l'ordre de parcours des points obligatoires, alors qu'il peut exister un chemin de A à B permettant de satisfaire ces contraintes (et aboutissant à une solution optimale).

Afin de résoudre ce problème, nous nous sommes tournés vers la littérature du domaine des télécommunications. Ce domaine traite en effet notamment de problèmes de gestion de la qualité de services dans les réseaux. Nous y avons emprunté un algorithme appelé LARAC (voir section 2.2.4.1), permettant de résoudre un problème DCLC (*Delay Constrained Least Cost*, ou de coût minimum avec contraintes sur les retards). Le coût équivaut ici dans notre cas au temps de la mission, et le retard est notre contrainte additive de capacité. Nous présentons dans la section qui suit comment nous y avons adjoint la gestion des contraintes de points de passage obligatoires.

3.4.2 Intégration du LARAC dans la méthode de résolution partielle

L'algorithme LARAC (pour *Lagrange Relaxation-based Aggregated Cost*) est un algorithme, simple de mise en œuvre, qui permet de résoudre les problèmes de plus court chemin comportant une contrainte additive. L'approche est décrite en section 2.2.4.1. Intrinsèquement, cette méthode ne permet pas de prendre en compte des points de passage obligatoires. Cependant, une petite adaptation permet de contourner cette limite.

Pour s'exécuter, l'algorithme LARAC nécessite la mise en place d'un algorithme de plus court chemin, typiquement celui de *Dijkstra* [Dijk 71]. Afin de traiter les contraintes, plusieurs appels successifs sont réalisés, en modifiant la fonction de coût associée aux arêtes du graphe. Or, dans notre problème avec points de passage obligatoire, notre algorithme de colonies de fourmis n'est rien d'autre qu'un algorithme de plus court chemin qui passe par l'ensemble des points obligatoires. Aussi, nous avons repris l'approche du LARAC pour y intégrer un appel non plus à Dijkstra, mais à ACO. Bien entendu, dans la pratique, la matrice M de distances utilisée dans ACO devra être recalculée avant chaque appel. Le pseudo-code de l'approche est fourni par l'algorithme 5.

De cette manière, on est capable de **déterminer simultanément la séquence optimale des points de passage obligatoires et l'ensemble des chemins** entre chaque couple de points qui satisfont la contrainte de capacité globale.

3.4.3 Dynamiser la recherche ACO

L'implémentation fournie par l'algorithme 5 est une adaptation simple, et efficace en termes de résolution. Néanmoins, dans la pratique, elle mène à réaliser consécutivement plusieurs exécutions indépendantes d'ACO. Dès lors, pourquoi ne pas réutiliser les informations des précédentes recherches pour tenter d'accélérer la résolution ? Nous l'avions mentionné dans la section 3.1.5, l'une des qualités d'ACO est son adaptation aux problèmes dynamiques. Or notre problème est bien dynamique, puisqu'il consiste à trouver récursivement la solution à un problème de structure équivalente, dans lequel seule la matrice de distances est modifiée.

Pour résoudre le problème de TSP dynamique, nous n'avons pas fait le choix de l'implémentation directe d'un algorithme de la littérature (voir fonction 2.5.5.2.c). L'algorithme P-ACO est une version très différente des algorithmes ACO courants, et nécessitait de gros efforts de modification de l'algorithme dont nous disposions. En outre, les mécanismes de mise à jour locale du modèle de phéromone d'*Ant System* n'ayant pas abouti à des résultats concluants, nous n'avons pas souhaité implémenter AS-DTSP tel quel. Nous sommes donc repartis de l'algorithme

Algorithme 5 Algorithme LARAC-ACO

Requiert: sommet de départ s, sommet d'arrivée t, fonction de coût c, fonction de délai d, ensemble de points obligatoires V_m, borne max de délai Δ_{delay}
Renvoie: un chemin p de coût $c(p)$ minimum tel que $d(p) \leq \Delta_{delay}$ et $V_m \subset p$

$M_c \leftarrow calcule_pcc(G, V_m, c)$
$p_c \leftarrow ACO(s, t, V_m, M_c)$
si $d(p_c) \leq \Delta_{delay}$ **alors**
  retourner p_c
fin si
$M_d \leftarrow calcule_pcc(G, V_m, d)$
$p_d \leftarrow ACO(s, t, V_m, M_d)$
si $d(p_d) > \Delta_{delay}$ **alors**
  retourner "Pas de solution"
fin si
boucle
  $\lambda \leftarrow \frac{c(p_c)-c(p_d)}{d(p_d)-d(p_c)}$
  $M_\lambda \leftarrow calcule_pcc(G, V_m, \lambda)$
  $r \leftarrow ACO(s, t, V_m, M_\lambda)$
  si $c_\lambda(r) = c_\lambda(p_c)$ **alors**
    retourner p_d
  sinon si $d(r) \leq \Delta_{delay}$ **alors**
    $p_d \leftarrow r$
  sinon
    $p_c \leftarrow r$
  fin si
fin boucle

ACO défini en section 3.2. Nous avons songé à deux types d'implémentation afin de réutiliser les résultats des recherches précédentes :

1. introduire une fonction d'*atténuation* des taux de phéromone, similaire au mécanisme de *shaking* de l'algorithme AS-DTSP (voir section 2.5.5.2.c). Cette fonction a pour rôle d'atténuer les *variations* entre les différents taux, en augmentant les taux de phéromones devenus faibles et en atténuant les taux de valeur élevée. L'objectif est de conserver l'apprentissage de la dernière recherche, puisqu'un écart persistera entre les différentes valeurs, mais néanmoins de maintenir une certaine diversification de la recherche en limitant cet écart ;
2. réinitialiser le modèle de phéromone et comparer les deux chemins p_c et p_d courants de l'algorithme LARAC pour définir le plus court (selon la nouvelle métrique de coût) en tant que meilleure solution trouvée. On relance donc totalement l'exploration, mais l'action des fourmis d'amélioration (opérant une recherche locale basée sur la meilleure solution trouvée) est renforcée par la mise à disposition d'une solution performante.

3.4.4 Cadre d'expérimentation

Pour notre expérimentation, nous avons repris la méthodologie décrite en section 3.2.3. La structure de données XML utilisée est la même. Cependant, nous avons modifié notre usage de cette structure. Précédemment, le type *Run* correspondait à une exécution des métaheuris-

tiques, dans lequel les valeurs des meilleures solutions étaient successivement inscrites. Le même problème était alors évalué 150 fois, aboutissant à une unique structure de données *Benchmark* contenant 150 éléments *Run*. Cette fois, nous allons utiliser le type *Benchmark* comme représentant une exécution de l'algorithme LARAC-ACO. ACO étant appelé plusieurs fois dans LARAC, chaque appel correspondra à un élément de type *Run*. A la différence de l'expérimentation précédente, chaque élément *Run* correspondra à un problème différent puisque les valeurs de distance du problème relaxé auront été modifiées entre deux appels à ACO. L'ordre des éléments *Run* dans la structure a donc cette fois une signification (celui du numéro de la recherche ACO). Pour faire une comparaison statistique, nous avons réalisé 150 itérations de recherche avec LARAC-ACO, chaque recherche aboutissant donc à un élément de type *Benchmark*.

Le *benchmark* utilisé pour cette expérimentation emploie un graphe issu d'un scénario réel, du niveau tactique d'un *Bataillon*. Sa taille est donc imposante : 479 nœuds et 2852 arcs. Le but, en employant une instance de cette dimension, est double : montrer l'efficacité de l'algorithme sur des problèmes de grande taille, et permettre d'avoir des chemins entre points obligatoires suffisamment variables (en fonction de la pénalité appliquée par la relaxation lagrangienne) pour contraindre la résolution du LARAC.

Pour définir les contraintes de capacité du graphe, nous avons réalisé une petite application permettant d'appliquer une image de fond à celui-ci. La valeur de contrainte appliquée à un arc est alors définie comme la moyenne des valeurs de luminance (valeur d'intensité lumineuse en vidéo) des pixels de l'image aux coordonnées des nœuds source et de destination de cet arc, à l'aide de la formule suivante :

$$Y = 0,3.R + 0,59.G + 0,11.B$$

où Y est la valeur de luminance (pour respecter la notation de la norme de chrominance YUV), et R,G,B sont respectivement les valeurs de rouge, vert et bleu de l'image (codée chacune sur 8 bits). Y, et par extension la valeur moyenne appliquée en contrainte aux arcs est donc également une valeur comprise entre 0 et 255. Le graphe ainsi que l'image sont donnés en annexe (voir section 5.3.4).

Nous avons élaboré 9 instances différentes utilisant le même graphe et les mêmes points de passage obligatoires. Ceux-ci sont au nombre de 75 (une valeur bien plus élevée que pour un scénario réaliste de niveau *Section*), et définis de telle sorte que leur répartition soit homogène sur le graphe, afin que de nombreux chemins soient envisageables. Les instances diffèrent donc uniquement dans la définition de leur point de départ et d'arrivée, ainsi que leur valeur de capacité maximale. Ces valeurs de capacité maximale ont été définies en fixant une valeur arbitrairement située dans une fourchette basse entre la valeur de capacité du plus court itinéraire non contraint (soit $d(p_c)$ dans la terminologie du LARAC) et celui du plus court chemin selon la métrique de contraintes (soit $d(p_d)$). Par exemple, pour la première instance où $d(p_c) = 20371$ et $d(p_d) = 17149$, nous avons fixé la contrainte maximale à $\Delta_{delay} = 18000$.

Afin de laisser le temps à l'algorithme pour converger vers une solution de bonne qualité, nous avons volontairement accru le nombre de cycles d'ACO en le fixant à 2500 cycles (soit davantage que lors de l'expérimentation de l'algorithme ACO sur un problème qui comptait deux fois plus de points de passage). Cette expérimentation permettra donc, dans le même temps, d'observer la valeur souhaitable du nombre de cycles pour un paramétrage efficace.

Trois approches sont comparées dans cette expérimentation :

– l'approche statique, de référence ;
– l'approche dynamique (1) utilisant la fonction d'atténuation suivante :

$$\forall e \in E', \tau_e = \tau_0 + \frac{\tau_e - \tau_0}{4}$$

– l'approche dynamique (2) réinitialisant le modèle de phéromone et définissant p_c ou p_d comme meilleure solution courante.

3.4.5 Résultats et discussion

Les résultats qui sont comparés sont ceux du temps de calcul nécessaire à chaque recherche ACO pour trouver la solution de référence (meilleure solution trouvée). L'optimalité est importante car chaque résultat d'ACO définit une nouvelle borne pour la recherche LARAC. Le nombre de points de passage définis ici est tout à fait crédible pour l'obtention d'une solution optimale par ACO. Afin que les résultats ne soient pas dépendants des performances de la machine de calcul utilisée, les temps sont définis en nombre de cycles de l'algorithme ACO.

Lors de nos expérimentations, nous avons tout d'abord utilisé l'approche mise au point en section 3.2 avec le même paramétrage. Cependant, lors de nos premiers essais, nous nous sommes aperçus que l'efficacité d'ACO pouvait être très variable, selon la recherche exécutée par LARAC. Cela était dû à la grande variabilité des variables de coût : nous avons observé des chemins optimaux allant d'une valeur de l'ordre de la dizaine de milliers à une valeur dépassant le million d'unités. C'est par conséquent l'étape de *renforcement* qui était la cause de la défaillance. En effet, le facteur de renforcement, défini comme l'inverse du coût de la meilleure solution trouvée (voir la formule 2.13 en section 2.5.5.2.c). Avec une fonction de distance dont les valeurs sont très grandes, le renforcement est ainsi quasi nul et l'apprentissage s'opère avec difficulté. Pour contrer cette difficulté, il est possible de normaliser la quantité de phéromone déposée. Dans notre cas, nous avons mis en place un paramétrage adaptatif de la valeur de la constante Q. Le renforcement se révélant très efficace dans le *benchmark* de la section 3.2, pour laquelle la valeur de coût de la solution optimale était d'environ 20000, nous avons fixé $Q = \frac{L_{NNH}}{20000}$ où L_{NNH} est la valeur de coût d'une solution obtenue par heuristique de plus proche voisin (exécutée à l'initialisation de l'algorithme). Ce paramétrage adaptatif simple permet ainsi de conserver une valeur significative de $\Delta\tau$ lorsque la métrique de distance est forte. Il a montré de bonnes performances sur l'ensemble des recherches menées ci-après. Notre approche devient ainsi **efficace malgré la grande variabilité possible des valeurs de distances**. Cette robustesse est renforcée par la présence des fourmis d'amélioration, qui permettait déjà à l'algorithme de progresser malgré la défaillance de l'apprentissage dans la première phase de la recherche (sans toutefois permettre, souvent, de trouver l'optimum rapidement). Par conséquent, nous avons décidé de conserver ce type de paramétrage, même dans l'approche ACO seule. Enfin, nous précisons qu'il n'est pas nécessaire de définir de constante pour le paramètre η dans la formule 2.10 car cela n'affecte pas la valeur finale de la probabilité (la constante étant présente également dans le dénominateur, elle annule celle du numérateur).

La figure 3.17 présente les résultats des trois approches envisagées. L'histogramme désigne le nombre moyen de cycles nécessaires à ACO pour obtenir la solution de référence, à chaque recherche lancée par LARAC. La barre d'erreur désigne l'écart-type de ce nombre de cycles. On note tout d'abord que l'algorithme LARAC n'a besoin que de sept itérations pour converger. Ce nombre de recherches est constant entre deux itérations car l'algorithme LARAC est déterministe, si l'algorithme de plus court chemin intégré l'est (c'est pour cela qu'on exige une solution optimale de la part d'ACO). De plus, les recherches 1 et 2 correspondent à la recherche de p_c et p_d, et ne peuvent pas être dynamiques : les performances des approches dynamiques ne sont donc pas supérieures à l'approche statique. A l'issue de ces deux recherches cependant, on note rapidement **un gain de performance des approches dynamiques** face à l'approche statique. Les deux approches dynamiques permettent de guider très rapidement la recherche pour des problèmes dont la solution est très proche de la solution précédente. On note en particulier

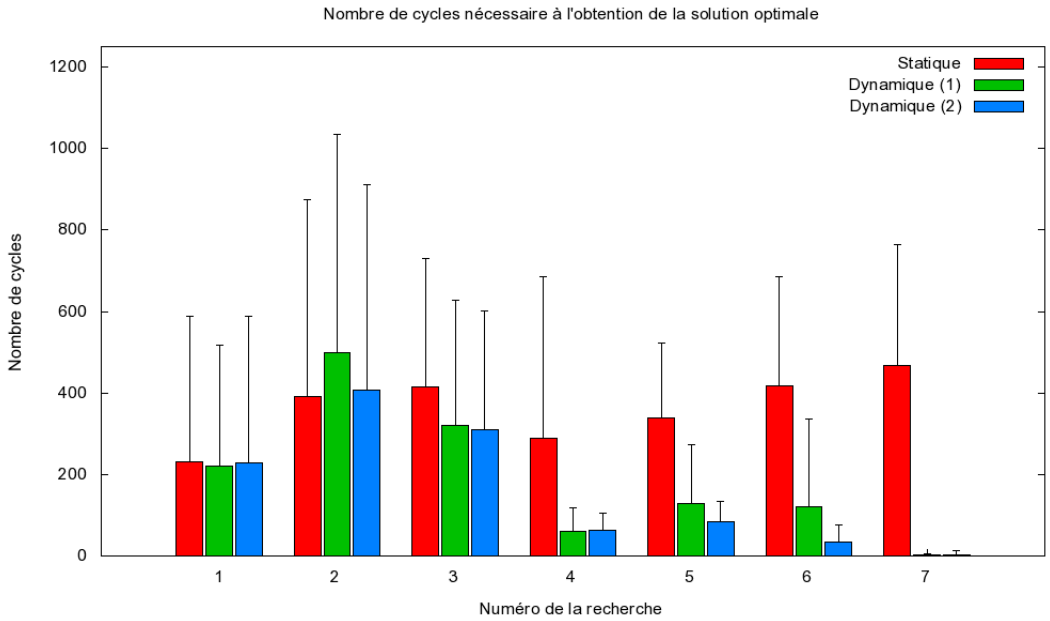


FIGURE 3.17 – Comparaison des approches de réinitialisation d'ACO pour l'algorithme LARAC-ACO : réinitialisation statique, dynamique (1) et dynamique (2). Le graphique illustre le nombre moyen de cycles, sur 150 exécutions, nécessaires à l'obtention de la solution de référence, pour chaque recherche ACO initiée par LARAC. Les barres d'erreurs représentent les valeurs d'écart-type.

que la dernière recherche a une solution totalement identique à la précédente, aboutissant à un délai d'obtention quasi nul de la solution de référence. Finalement, les deux approche dynamiques ont un comportement quasi-équivalent sur l'ensemble des recherches, avec néanmoins un léger avantage pour l'approche (2), qui utilise directement p_c ou p_d comme meilleure solution courante au début de la recherche, dans les 5^e et 6^e exécutions. Cela peut s'expliquer par le fait que la seconde approche, disposant dès le départ d'une solution courante proche de l'optimum, permet aux fourmis d'amélioration (celles pratiquant une recherche locale sur une perturbation de la meilleure solution trouvée) de faire progresser la recherche plus efficacement. Également, on observe lors de la 2^e recherche que l'approche dynamique (1) se montre légèrement moins performante que les deux autres approches. Cela s'explique par le fait que le modèle de phéromone a conservé en partie l'apprentissage réalisé précédemment, tandis que ce dernier ne profite pas à la recherche (la solution étant très différente). C'est donc un point faible manifeste de ce type d'approche dynamique. En définitive, c'est donc la seconde approche dynamique qui sera conservée pour la suite de nos travaux.

Enfin, les résultats nous montrent que pour des problèmes jusqu'à 75 nœuds, il est souhaitable de configurer le nombre de cycles à $C = 1000$ pour les deux premières recherches. Ensuite, il peut être judicieux de configurer un nombre de cycles *glissant* : si aucune amélioration n'est observée après $C = 300$ cycles, on stoppe la recherche.

3.4.6 Conclusion

L'approche LARAC-ACO est une combinaison simple et efficace permettant de résoudre un problème du chemin hamiltonien minimum tout en assurant la tenue d'une contrainte de

capacité additive. Avec une modification minimaliste, nous avons dérivé l'utilisation du LARAC initialement employé en télécommunications, pour l'appliquer à notre problème de planification. L'interprétation des recherches ACO comme un problème dynamique nous permet en outre de tirer parti de l'efficacité de l'approche de colonies de fourmis en réduisant la quantité de cycles nécessaires à l'obtention des solutions optimales. Bien que LARAC soit potentiellement sensible à l'échec d'une recherche de plus court chemin, l'efficacité de notre algorithme ACO (défini dans la section 3.2) nous permet de l'employer en remplacement d'une approche déterministe classique. De plus, le paramétrage adaptatif employé a montré son efficacité sur des problèmes très variables en termes de valeurs de coûts et de contraintes.

Au final, cette approche se révèle très efficace dans la résolution de problèmes très contraints. Il est intéressant de noter qu'**elle n'induit pas de surcoût de calcul si le problème n'est pas contraint**, car l'algorithme LARAC n'effectue alors qu'une seule recherche ACO. C'est pourquoi LARAC-ACO pourra être utilisé en tant que solveur partiel **dans tous les cas**, en remplacement de la seule approche ACO présentée en début de ce chapitre. Notons qu'avec le traitement des contraintes additives, LARAC-ACO résout en fait le problème global, et non une relaxation. Cette approche aboutie permet d'envisager une intégration dans le problème initial, qui est un problème multi-véhicules. Le problème mono-véhicule constitue donc une relaxation de ce problème initial, car il ne comprend pas les contraintes de coordination des véhicules.

L'intégration de l'approche LARAC-ACO dans le solveur ORTAC est évaluée dans les chapitres suivants, dans lesquels nous traitons différentes contraintes de mission spécifiques.

Chapitre 4

Application de l'approche à des problèmes de planification spécifiques

4.1 Navigation sous contraintes d'énergie

4.1.1 Présentation du problème

4.1.1.1 Motivation

Quand on aborde la planification de véhicules ou de drones (sans se restreindre au domaine militaire), l'une des problématiques les plus récurrentes est celle de la gestion de l'énergie. En effet, déterminer un itinéraire qui minimise un temps de trajet, sans contrainte particulière sur la vitesse, revient à minimiser la distance de ce trajet. Or cette planification peut amener à emprunter des itinéraires coûteux en énergie (par exemple une côte ou un terrain peu praticable). Considérer une limite de consommation au cours de la mission peut donc avoir une forte influence sur la planification optimale du véhicule : pour minimiser son temps de trajet en tenant une contrainte forte de consommation, ce dernier devra peut-être privilégier un itinéraire détourné lui permettant de progresser à pleine vitesse plutôt qu'un itinéraire court, dans lequel il devra réduire sa vitesse afin d'abaisser sa consommation.

Nous traitons ce problème dans la suite de cette section, en considérant un modèle de consommation présenté dans la section 4.1.1.3. Ce modèle est simpliste : un modèle de consommation réel est fortement lié aux spécificités du véhicule, et prend en compte un grand nombre de paramètres. Il permet néanmoins de contraindre fortement le problème et induit un enjeu en termes de stratégie de recherche. Nous définissons le modèle logique de consommation en section 4.1.2.1, puis nous montrons en section 4.1.2.3 comment résoudre ce problème avec l'approche de résolution définie précédemment. Le modèle d'énergie employé introduit une contrainte *additive*, permettant d'expérimenter l'approche avec relaxation lagrangienne présenté en section 3.4.

4.1.1.2 Formalisation du problème d'optimisation

Le problème de planification sous contraintes d'énergie a été formalisé sous le nom de *Energy-constrained Fastest Path with Waypoints* (EFP+W) [Luca 12b]. Il s'exprime de la manière suivante :

$$p^\star = \min_{p \in P'(v_s, v_g, V_m)} \sum_{e \in p} t(e) \tag{4.1}$$

où

$$P'(v_s, v_g, V_m) = \left\{ p \in P(v_s, v_g, V_m) \mid \sum_{e \in p} c(e) \leq R_0 \right\} \tag{4.2}$$

$P(v_s, v_g, V_m)$ est l'ensemble des chemins de v_s à v_g, passant par tous les points de V_m. $c(e)$ représente la variable de consommation d'un arc e, définie en section 4.1.1.3. R_0 est la ressource d'énergie disponible pour la mission. $t(e)$ désigne la variable de temps associée à e, définie comme suit :

$$\forall e \in E_p,\ t(e) = \frac{d(e)}{s(e)} \tag{4.3}$$

où $s(e)$ est la vitesse du véhicule le long de e. $s(e)$ est définie dans l'ensemble $\{0, S_{max}\}$, S_{max} étant une entrée du problème.

Une solution au problème EFP+W est un chemin p ainsi qu'une valeur de temps discrète associée à chaque arc de p.

4.1.1.3 Contraintes d'énergie

Le modèle de consommation est exprimé à l'aide d'un modèle de flux d'énergie (énergie consommée par unité de temps). Ce flux d'énergie est composé de deux paramètres :

- une valeur de flux constant f_0, correspondant au flux d'énergie consommé lorsque le véhicule est à l'arrêt (nul dans le cas d'une propulsion électrique) ;
- une valeur de flux instantané f_i correspondant au flux d'énergie consommé pour mouvoir le véhicule (croissant avec le carré de sa vitesse).

Pour chaque arc $e \in p$, le flux d'énergie $f(e)$ consommé est donné par (les conversions d'unités n'apparaissent pas dans les équations) :

$$f(e) = f_0 + f_i(e) \tag{4.4}$$

où

$$f_i(e) = m(e).\frac{s(e)^2}{Q} \tag{4.5}$$

Q est une constante utilitaire. $m(e)$ est un *facteur de consommation*, correspondant à une valeur de pourcentage, où 100% est une route « plate » (une route en montée et en descente auront un facteur de consommation supérieur et inférieur à 1 respectivement). Le flux $f(e)$ peut être intégré sur le temps pour déterminer l'énergie effective consommée $c(e)$ nécessaire au véhicule pour traverser e :

$$\forall e \in p, c(e) = f(e).t(e) \tag{4.6}$$

La figure 4.1 est une représentation graphique de la valeur de $c(e)$ (en L/100km) en fonction du flux f_0 et de la valeur de vitesse $s(e)$.

4.1.1.4 Paramètres

Les constantes utilisées pour le modèle de consommation ont été choisies de manière empirique. Le critère que nous avons cherché à satisfaire est l'obtention (pour un facteur de consommation $m = 100\%$) d'une consommation minimale à la vitesse de 20km/h, et une consommation

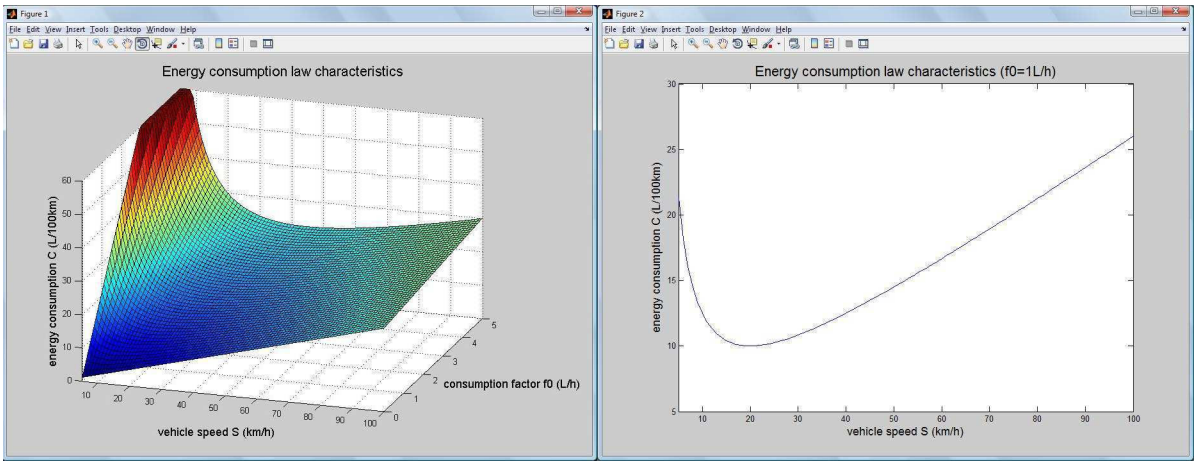


FIGURE 4.1 – Graphique de gauche : loi de consommation d'énergie pour une arc e avec $m(e) = 100$. Le graphique présente la consommation d'énergie effective $c(e)$ (en L/100km) consommée en fonction de la vitesse $s(e)$ du véhicule et de la consommation statique f_0. Pour une question de lisibilité, l'axe $s(e)$ débute à 5km/h (le flux croît infiniment quand la vitesse tend vers 0). Graphique de droite : loi de consommation d'énergie pour le cas particulier où $f_0 = 1$L/h.

de l'ordre de 15L/100km pour une vitesse moyenne de 60km/h (consommation proche d'un véhicule de transport de troupes). Les paramètres retenus sont les suivants :

- $S_{max} = 60$km/h ;
- $f_0 = 1$L/h ;
- $Q = 400$.

Les facteurs de consommation m ont été générés en combinant trois fonctions trigonométriques basiques. Pour un arc e :

$$f_1(e) = 100 + 70.(\sin(\frac{d(e)}{40}))$$

$$f_2(e) = 100 + 40.(\cos(\frac{d(e)}{25}))$$

$$f_3(e) = 100 + 60.(\sin(\frac{d(e)}{11}))$$

$m(e)$ est exprimé comme la valeur moyenne de f_1, f_2 et f_3 :

$$m(e) = \frac{f_1(e) + f_2(e) + f_3(e)}{3}$$

La figure 4.2 illustre la courbe de m pour des valeurs de distances dans le domaine $[0; 1200]$.

Les valeur $m(e)$ obtenues, dépendant uniquement de la distance, ne sont en aucun cas réalistes. Cette fonction a été mise en place dans le but de générer des valeurs de manière « aléatoire ». Il est à noter que le trois fonctions $f_1(e)$, $f_2(e)$ et $f_3(e)$ ont le même *offset*, afin d'être centrées sur 100 (coefficient dit neutre). Les amplitudes et les périodes ont été définies empiriquement afin d'obtenir des écarts de valeurs élevés pour une faible variation de distance.

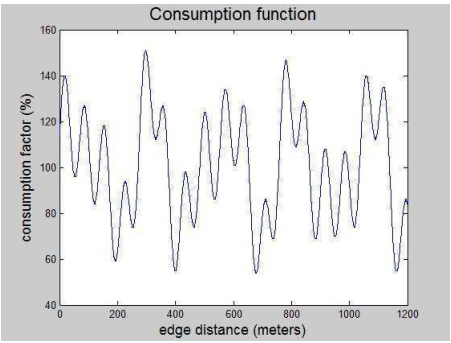


FIGURE 4.2 – Fonction de génération du facteur de consommation d'un arc, selon sa distance.

4.1.2 Résolution du problème

4.1.2.1 Modélisation en Programmation par Contraintes

Il est à noter que les équations de temps et de consommation dépendent toutes deux de la vitesse s du véhicule. Pour limiter le nombre de variables impliquées dans le modèle, plutôt que d'intégrer des variables de vitesse s, nous avons choisi d'exprimer les contraintes de consommation en modélisant directement c comme une fonction de t uniquement. Le fait de réduire le nombre de variables a deux intérêts : potentiellement réduire les temps de calcul (puisque chaque variable est sujette à une propagation de contraintes), et limiter l'effet des erreurs d'arrondis dues à la discrétisation, qui sont une problématique majeure pour ce type de problèmes. c est donc exprimé comme suit (les conversions d'unités n'apparaissent pas dans les équations) :

$$\forall e \in E, c(e) = \begin{cases} f_0.t(e) + \frac{m(e).d(e)^2}{Q.t(e)} \text{ si } \phi(e) = 1 \\ 0 \text{ sinon.} \end{cases} \tag{4.7}$$

et la contrainte additive de consommation le long du chemin est :

$$\sum_{e \in E} \phi(e).c(e) \leq R_0 \tag{4.8}$$

où R_0 est la ressource initiale d'énergie du véhicule pour la mission. Le problème d'optimisation toujours posé par :

$$\min \sum_{e \in E} \phi(e).t(e) \tag{4.9}$$

où $\phi \in \{0, 1\}$ et $t \in \mathbb{N}$ sont les variables de décision du problème.

4.1.2.2 Amélioration du modèle logique de chemins

Les variables de temps étant directement contraintes par les variables de consommation, leur propagation peut se révéler très coûteuse en temps de calcul en raison du caractère non linéaire des contraintes. Or, dans notre modèle logique initial, ces variables sont directement utilisées pour vérifier la cohérence du chemin (c'est-à-dire l'absence de boucles), à l'aide des équations 1.11 et 1.12.

Aussi, pour permettre une vérification rapide de la cohérence des chemins sans nécessiter la propagation des variables de temps, nous avons introduit un nouveau type de variables, nommées

id et associées aux nœuds du graphe, jouant le rôle de propagateur spécifique. Ces *identifiants*, tout comme les variables de temps précédemment, sont utilisés pour propager des contraintes de précédence le long du chemin :

$$id(v_s) = 1 \tag{4.10}$$

$$\forall v \in V \backslash \{v_s\}, id(v) = \sum_{e=(u,v)\in\omega^+(v)} \phi(e).(id(u)+1) \geq 0 \tag{4.11}$$

$$id(v_g) = 1 + \sum_{e\in E} \phi(e) \tag{4.12}$$

Chaque variable $id(v)$ est définie dans $\{0, |V|\}$, puisqu'un véhicule ne peut passer qu'une fois par chaque nœud du graphe (comme précisé en section 1.4.1). Ces variables peu contraintes, et de domaines de valeurs réduits, permettent d'assurer de manière très efficace la cohérence du chemin. A l'issue de ces travaux, ce modèle a d'ailleurs été appliqué de manière généralisée à l'ensemble des problèmes traités. Il est à noter que le modèle de propagation des temps peut être conservé, pour introduire des contraintes de fenêtre de temps sur les nœuds par exemple (ce cas ne sera pas traité).

4.1.2.3 Résolution

Le problème présenté ci-avant est caractérisé par des contraintes de capacité non linéaires qui sont fonction des variables de vitesse associées aux arcs, donc indirectement du temps. De plus, ces contraintes sont particularisées pour chaque arc, avec l'introduction des paramètres $m(e)$: il n'est donc pas possible de considérer l'application d'une vitesse unique à l'ensemble du chemin.

En raison de l'interdépendance entre la fonction de coût et la contrainte de consommation, la méthode LARAC-ACO présentée en section 3.4 n'est pas applicable directement. En effet, l'algorithme LARAC est conçu pour résoudre des problèmes dont les valeurs de coût et de capacité associées aux arcs sont fixées. Or, dans notre cas, ces valeurs ne sont pas fixées mais dépendent toutes deux de la vitesse du véhicule le long de l'arc considéré.

Il est cependant possible de contourner ce problème en extrayant un multi-graphe — c'est-à-dire un graphe pouvant posséder plusieurs arcs entre deux nœuds — à partir du graphe initial. Dans ce multi-graphe, on génère autant d'arcs que de valeurs discrètes de temps possibles (chaque arc correspondant à une vitesse fixée). Chaque arc dispose ainsi d'un couple $< t, c >$ de valeur fixe. De ce multi-graphe, on extrait alors un graphe simple préalablement à chaque appel à ACO dans l'algorithme LARAC. L'extraction est réalisée de telle sorte que ce graphe simple ne comporte que les arcs de coût réduit c_λ de valeur minimale. L'implémentation de l'algorithme est donné en section 4.1.2.5.

4.1.2.4 Guidage du solveur de contraintes

Ne guider le solveur de contraintes qu'au niveau de ses variables de flots peut aboutir à des performances très médiocres. En effet, l'étiquetage des variables de temps le long d'un chemin peut se révéler fastidieux en raison de la propagation des contraintes de consommation. C'est pourquoi, en complément du guidage des variables de flots, il peut être judicieux d'appliquer des contraintes jouant le rôle de bornes maximales sur les variables de temps. En effet, lors de la résolution partielle, la solution retournée comporte des valeurs de temps qui, si le modèle de flot est valide (c'est-à-dire respectant l'unicité des flots à chaque nœud), sont nécessairement réalisables. En revanche, si le modèle de flot échoue à trouver un chemin cohérent selon le modèle

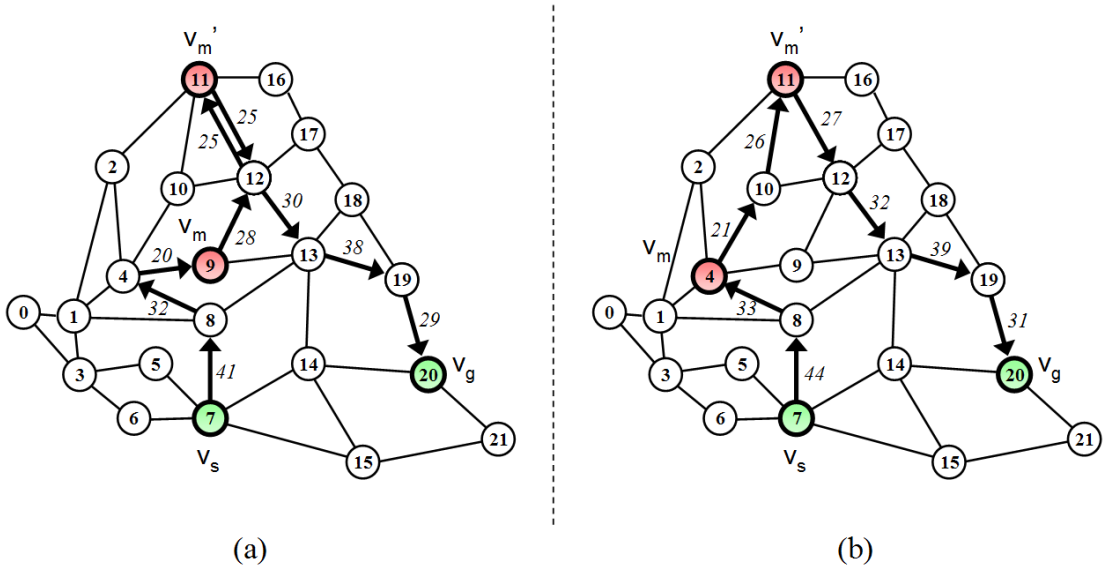


FIGURE 4.3 – Illustration des deux cas de figure possibles pour l'utilisation des variables de temps de la solution partielle. Les valeurs en italique sont une illustration possible des temps de traversée des arcs empruntés, en secondes. (a) La solution partielle n'est pas conforme au modèle de flots du solveur de contraintes (la solution passe deux fois au nœud 12). (b) La solution partielle est cohérente avec le modèle de flots du solveur de contraintes.

de contraintes, ces bornes ne peuvent être appliquées. La figure 4.3 illustre par un exemple les deux cas de figure possibles pouvant survenir dans notre implémentation. La solution partielle étant composée d'un ensemble de chemins unitaires juxtaposés, elle est susceptible de contenir plusieurs fois un nœud non obligatoire. C'est le cas de l'exemple (a), qui n'est par conséquent pas valide avec le modèle de contraintes. Aussi, les valeurs de temps obtenues ne peuvent être utilisées pour borner la recherche. Dans l'exemple (b) en revanche, le chemin est cohérent avec le modèle de contraintes. Le solveur partiel garantissant que les contraintes de capacité ne sont pas violées, ce chemin ainsi que les valeurs de temps associées constituent une solution réalisable du point de vue du problème global. On peut donc utiliser ces variables de temps pour borner (temporairement seulement, afin de garantir la complétude de l'approche) les variables de temps du problème global.

Pour ce faire, un test est ajouté dans l'implémentation du solveur de contraintes. Après l'étape d'étiquetage des flots, on détermine si l'instanciation courante est équivalente à celle de la solution partielle. Si ce n'est pas le cas, la résolution se poursuit comme précédemment. Si c'est le cas, des contraintes d'infériorité (non strictes) sont mises en place sur les variables de temps, pour les borner à la valeur obtenue dans la solution partielle. Par exemple, dans l'exemple de la figure 4.3.(b), on définit que $t(7,8) \leq 44$. Le caractère non strict de l'inéquation permet de garantir que la solution partielle sera trouvée. Une fois l'ensemble des valeurs de temps évaluées, les contraintes sur les temps sont relâchées afin de conserver le caractère complet de la recherche.

Cette méthode permet de guider très efficacement les variables de temps, dès lors que la solution partielle satisfait le modèle de flot. Les résultats présentés en section 4.1.3 attestent du gain de performance apporté par cette implémentation.

4.1.2.5 Implémentation du solveur partiel

La méthode de génération du multi-graphe est primordiale pour la qualité de la solution. En effet, le calcul des valeurs de consommation doit être parfaitement calqué sur le modèle exprimé en contraintes dans le modèle logique du problème. Si le modèle s'avérait légèrement différent (absence d'arrondis sur le calcul de certains paramètres), les valeurs de temps de la solution obtenue ne pourraient pas être appliquées en tant que bornes sur les variables de temps dans le solveur de contraintes.

Pour implémenter l'algorithme dérivé du LARAC présenté en section 4.1.2.3, il n'est pas possible d'utiliser la version initiale proposée par *Jüttner et al.* [Jutt 01]. En effet, celle-ci travaille avec une sauvegarde des chemins de coût et de délai minimums (p_c et p_d respectivement). Or, dans notre adaptation, il n'est pas concevable d'appliquer cette implémentation. En effet, entre deux recherches, le graphe simple servant à la recherche est *mis à jour*. En d'autres termes, en recalculant les valeurs de coût et de délai associés à p_c et p_d dans notre cas, les valeurs obtenues entre deux recherches varieraient, ce qui reviendrait à disposer de bornes fluctuantes ! Pour éviter ce désagrément, nous avons implémenté et testé une variante du LARAC appelé LARAC-BIN [Xiao 05]. Celle-ci utilise une recherche binaire pour mettre à jour la valeur de λ. Cependant, si cette méthode est très efficace pour obtenir une valeur approchée (il est possible de paramétrer cette valeur approchée), la convergence de l'algorithme peut être très lente. Lors de nos tests sur un problème simple, nous avons abouti à une soixantaine d'itérations de recherche (contre 4 à 8 recherches dans le cas d'une exécution classique de LARAC). Aussi, nous avons développé notre propre implémentation du LARAC. Celle-ci reste basée sur l'algorithme LARAC initial mais se contente de sauvegarder les valeurs de coût et de délai des chemins p_c et p_d, qui ne sont jamais recalculés. En outre, la condition d'arrêt de l'algorithme a elle-aussi été modifiée. Celui-ci s'arrête lorsque deux valeurs identiques de λ ont été obtenues consécutivement.

L'algorithme 6 présente le pseudo-code de l'algorithme LARAC mis en œuvre. Cet algorithme ne conserve en mémoire que les valeurs de coût et de délai des chemins de poids minimum (c_c et d_c pour la borne inférieure, c_d et d_d pour la borne supérieure), qui sont utilisées pour calculer λ. Ainsi, les valeurs ne sont pas susceptibles d'être modifiées de manière intempestive. En outre, le chemin p_d est mis à jour en prenant pour valeur r lorsque le chemin r trouvé tient les contraintes de délai, afin de sauvegarder la meilleure solution au problème. Ainsi, lorsque la condition d'arrêt ($\lambda = \lambda_{-1}$) est atteinte, p_d est renvoyé. Il n'est pas utile en revanche de conserver en mémoire les chemins p_c. Cet algorithme est en pratique équivalent à l'algorithme LARAC originel, et converge tout aussi rapidement.

4.1.3 Résultats

Cette section présente les résultats obtenus sur les instances de *benchmark* mises en place (voir annexe 5.3.4) de notre approche employant le solveur partiel LARAC-ACO-M présenté en section précédente. A titre de comparaison, en l'absence de méthode équivalente dans la littérature, nous avons implémenté une approche de résolution séquentielle que nous noterons ACO+SA. Dans cette approche, un algorithme ACO est utilisé pour obtenir un plus court chemin empruntant l'ensemble des points de passage. Une fois ce chemin obtenu, un algorithme de recuit simulé est employé pour déterminer les temps de traversée à appliquer à chaque arc du chemin pour minimiser le temps de trajet tout en satisfaisant les contraintes de capacité. Bien entendu, cette deuxième approche est sous-optimale car elle considère de manière disjointe les problématiques d'optimisation que sont les points de passage et la gestion des contraintes de consommation.

Algorithme 6 Algorithme LARAC-ACO-M

Requiert: sommet de départ s, sommet d'arrivée t, fonction de coût c, fonction de délai d, ensemble de points obligatoires V_m, borne max de délai Δ_{delay}
Renvoie: un chemin p de coût $c(p)$ minimum tel que $d(p) \leq \Delta_{delay}$ et $V_m \subset p$

$G_t \leftarrow extraction\_multi\_graphe(G)$
$G' \leftarrow extraction\_graphe(G_t, c)$
$p_c \leftarrow ACO(G', s, t, V_m)$
$d_c \leftarrow d(p_c)$
si $d_c \leq \Delta_{delay}$ **alors**
  retourner p_c
fin si
$G' \leftarrow extraction\_graphe(G_t, d)$
$p_d \leftarrow ACO(G', s, t, V_m)$
$d_d \leftarrow d(p_d)$
si $d_d > \Delta_{delay}$ **alors**
  retourner "Pas de solution"
fin si
$c_c \leftarrow c(p_c)$
$c_d \leftarrow c(p_d)$
$\lambda_{-1} \leftarrow infini$
boucle
  $\lambda \leftarrow \frac{c_c - c_d}{d_d - d_c}$
  si $\lambda = \lambda_{-1}$ **alors**
    retourner p_d
  fin si
  $G' \leftarrow extraction\_graphe(G_t, c_\lambda)$
  $r \leftarrow ACO(G', s, t, V_m, c_\lambda)$
  si $d(r) \leq \Delta_{delay}$ **alors**
    $p_d \leftarrow r$
    $c_d \leftarrow c(r)$
    $d_d \leftarrow d(r)$
  sinon
    $c_c \leftarrow c(r)$
    $d_c \leftarrow d(r)$
  fin si
  $\lambda_{-1} \leftarrow \lambda$
fin boucle

Les tableaux de résultats présentés ci-après ne comportent que les temps de calcul du solveur de contraintes afin de montrer les effets de la résolution partielle sur le guidage. Dans tous les cas, la solution partielle a pu être obtenue dans un délai inférieur à 500ms, ce qui est négligeable au regard des temps potentiellement obtenus pour le solveur de contraintes. L'ensemble des essais ont été réalisés sur un PC portable *Samsung Q310*, équipé d'un microprocesseur *Intel Core 2 Duo P7350* cadencé à 2GHz et de 3Go de mémoire vive.

Le tableau 4.1 présente les résultats de l'approche de résolution séquentielle ACO+SA. Avant toute interprétation, on peut en premier lieu observer la complexité du problème, introduite par

Approche de résolution partielle ACO+SA					
Benchmark	Instance	Ancien modèle		Nouveau modèle	
		T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)
Benchmark 1	mission_1_1	16 (189)	1810	0 (189)	234
	mission_1_2	2246 (155)	17331	31 (155)	171
	mission_1_3	**15 (416)**	–	**492167 (398)**	–
	mission_1_4	16 (224)	43836	16 (224)	390
	mission_1_5	15 (333)	104552	15 (333)	733
	mission_1_6	**409907 (445)**	–	**433511 (432)**	–
Benchmark 2	mission_2_1	5429 (372)	18767	62 (372)	483
	mission_2_2	0 (384)	18564	16 (384)	359
	mission_2_3	32 (401)	–	32 (401)	1779
	mission_2_4	**443496 (727)**	–	175283 (502)	177842
	mission_2_5	**392233 (763)**	–	**409285 (725)**	–
	mission_2_6	–	–	**215484 (1751)**	–
Benchmark 3	mission_3_1	16 (812)	1451	16 (812)	728
	mission_3_2	842 (665)	1138	390 (665)	718
	mission_3_3	–	–	–	–
	mission_3_4	**267 (1948)**	–	**31 (1948)**	–
	mission_3_5	–	–	–	–
	mission_3_6	**392311 (1651)**	–	**390361 (1651)**	–

TABLE 4.1 – Résultats de l'approche ACO+SA présentée sur les instances du *benchmark* proposé. Un comparatif est ici réalisé, entre l'ancien et le nouveau modèle logique. Les cases T_1 et T_2 présentent respectivement les temps pour trouver la solution optimale, et le temps nécessaire au solveur de contraintes pour obtenir la preuve d'optimalité. Les valeurs indiquées entre parenthèse dans les cases T_1 sont les valeurs des meilleures solutions trouvées. Les cases présentant une police en caractère gras indiquent les cas où la recherche a échoué à trouver la solution optimale. Un délai maximum pour la recherche a été fixé à 500 secondes.

les contraintes de consommation, sur des instances de taille pourtant modestes (moins de 25 nœuds). Deux implémentations sont ici évaluées : une version initiale du solveur de contraintes (ancien modèle) et la nouvelle version intégrant les variables spécifiques de cohérence de chemin. On note tout d'abord que le nouveau modèle permet, lorsqu'une solution est trouvée, d'accélérer grandement les temps de propagation pour l'évaluation des autres solutions : dans l'instance *mission_1_1* par exemple, la preuve d'optimalité est obtenue en un peu plus de 200ms contre presque 2s auparavant. Au total, sur les cinq instances *mission_1_2, 1_4, 1_5, 2_1* et *2_2*, le solveur met moins d'une seconde à terminer la recherche quand il lui fallait plusieurs dizaines voire une centaine de secondes pour le faire auparavant.

Le gain apporté par le propagateur est donc manifeste, dans les cas où une solution est trouvée. Malgré cela, l'approche séquentielle échoue dans huit cas (sur un total de quinze instances proposées)à trouver la solution optimale en 500 secondes, quel que soit le modèle. Dans deux à trois cas, aucune solution n'est d'ailleurs trouvée. Ce type de résolution montre clairement ses limites, car il ne considère pas de manière conjointe la tenue des contraintes de points de passage et celle de l'énergie du véhicule.

Le tableau 4.2 présente les résultats de l'approche LARAC-ACO-M sur les instances de *benchmark* mises en place. Pour la lisibilité, les résultats de l'approche ACO+SA ont été recopiés dans la partie gauche du tableau. On peut noter le gain apporté par LARAC-ACO-M sur le *benchmark* 1, puisque cette approche permet de résoudre l'ensemble des instances et d'apporter la preuve d'optimalité dans un délai inférieur à une seconde. En comparaison, ACO+SA échouait

Benchmark	Instance	ACO+SA		LARAC-ACO-M	
		T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)
Benchmark 1	mission_1_1	0 (189)	234	0 (189)	218
	mission_1_2	31 (155)	171	16 (155)	125
	mission_1_3	**492167 (398)**	–	15 (306)	546
	mission_1_4	16 (224)	390	0 (224)	374
	mission_1_5	15 (333)	733	0 (333)	717
	mission_1_6	**433511 (432)**	–	0 (363)	328
Benchmark 2	mission_2_1	62 (372)	483	47 (372)	437
	mission_2_2	16 (384)	359	16 (384)	359
	mission_2_3	32 (401)	1779	31 (401)	1685
	mission_2_4	175283 (502)	177842	0 (502)	1997
	mission_2_5	**409285 (725)**	–	**496738 (678)**	–
	mission_2_6	**215484 (1751)**	–	0 (787)	–
Benchmark 3	mission_3_1	16 (812)	728	47 (812)	749
	mission_3_2	390 (665)	718	0 (665)	561
	mission_3_3	–	–	2060 (2276)	–
	mission_3_4	**31 (1948)**	–	187 (1798)	–
	mission_3_5	–	–	**496973 (2653)**	–
	mission_3_6	**390361 (1651)**	–	**497300 (1568)**	–

TABLE 4.2 – Résultats comparatifs des deux approches de résolution partielle, ACO+SA et LARAC-ACO-M, avec le nouveau modèle de propagation, sur les instances du *benchmark* proposé. Les cases T_1 et T_2 présentent respectivement les temps pour trouver la solution optimale, et le temps nécessaire au solveur de contraintes pour obtenir la preuve d'optimalité. Les valeurs indiquées entre parenthèse dans les cases T_1 sont les valeurs des meilleures solutions trouvées. Les cases présentant une police en caractère gras indiquent les cas où la recherche a échoué à trouver la solution optimale. Un délai maximum pour la recherche a été fixé à 500 secondes.

à trouver la solution optimale en 500s pour deux des six instances. Pour le *benchmark* 2, LARAC-ACO est capable de trouver la solution optimale dans cinq cas sur six (contre trois sur six pour ACO+SA). Toutefois, il échoue à trouver la solution optimale sur l'instance *mission_2_5*. De même, il n'est pas capable de terminer la recherche dans les cas *2_5* et *2_6*. Pour le *benchmark* 3, LARAC-ACO-M trouve la solution optimale dans quatre cas sur six (contre deux sur six précédemment). Il échoue à trouver les solutions optimales de deux instances, et de terminer la recherche dans quatre cas sur six.

En conclusion, l'approche LARAC-ACO apporte un gain manifeste dans la résolution de problèmes complexes tel que des instances EFP+W, puisqu'elle permet au solveur de contraintes de trouver la solution optimale immédiatement. Cela est rendu possible par le fait que la solution apportée par le solveur partiel est une solution réalisable au problème global. Cependant, dans le cas de certaines instances (*2_6*, *3_3*, *3_4*), le grand nombre de solutions potentielles (dont le coût est proche de la solution optimale) rend nécessaire une évaluation exhaustive de l'espace de recherche, aboutissant à l'absence de preuve d'optimalité en 500s. En outre, **la résolution des instances *2_5*, *3_5* et *3_6* est pénalisée par la différence sensible entre les modèles de chemin du solveur partiel et du solveur global**. Dans notre cas, le solveur partiel autorise le fait de repasser par un même nœud puisque la solution partielle est construite à l'aide d'une juxtaposition de chemins élémentaires entre points obligatoires. Or le modèle de flots du solveur de contraintes n'autorise pas le passage multiple à un même nœud du graphe. Le guidage des flots aboutissant en premier lieu à une solution non réalisable, le solveur doit réaliser un ensemble de retours sur trace pour déterminer la solution optimale. L'étiquetage des flots ne correspondant pas à la solution partielle fournie, les contraintes sur les temps ne peuvent être appliquées, aboutissant à une évaluation exhaustive des combinaisons de valeurs de temps. Cette différence dans les modèles ne pose, dans le cas de problèmes « classiques » (c'est-à-dire sans contraintes non linéaires), pas de problème puisque la propagation des contraintes est généralement rapide. Or, dans le cas particulier du problème EFP+W, la propagation des variables de temps est fastidieuse car ces dernières sont impliquées dans les contraintes de consommation. Une idée pourrait consister à réparer la solution partielle pour éviter à un chemin de repasser par un même nœud, corrigeant ainsi les différences entre les modèles, mais la mise en place d'une telle méthode est loin d'être triviale.

4.1.4 Conclusion

Dans ce section, nous avons présenté une nouvelle approche de résolution, basée sur l'approche décrite en section 3.4, dédiée à la gestion spécifique de contraintes additives non linéaires. Nous avons appliqué cette approche à un problème de gestion d'énergie, avec des résultats très satisfaisants. La résolution partielle implémentée, permettant de résoudre très rapidement des instances du problème global, permet de guider très rapidement le solveur de contraintes vers une solution réalisable. En outre, le nouveau modèle logique de cohérence de chemin permet d'analyser efficacement la validité d'une instanciation des flots sans avoir à faire intervenir les variables de temps, dont les temps de propagation sont longs en raison de leur implication dans les contraintes de consommation.

Ainsi, cette approche nous permet d'envisager le traitement de problèmes non linéaires, pour lequels la programmation par contraintes montre ses limites. En revanche, elle reste handicapée par la légère différence du modèle de construction d'un chemin entre solveur partiel et solveur global, qui peut encore aboutir à des contre-performances et échouer à résoudre un problème en temps raisonnable.

4.2 Gestion des communications radio pour véhicules en mission

Les travaux présentés ci-après ont fait l'objet d'une première publication à la conférence *Military Communications* (MILCOM) [Luca 10]. Suite à l'amélioration de l'approche, et l'obtention de nouveaux résultats, un nouvel article [Luca 12a] a été soumis et accepté pour l'édition 2012 de la même conférence.

4.2.1 Présentation du problème

Le problème traité dans cette section est celui de la gestion de contraintes de communications lors de la détermination d'itinéraires de véhicules. La motivation de ces travaux est liée à la dépendance croissante des entités terrestres envers les réseaux tactiques. En effet, les armées modernes sont basées sur le concept d'*opérations réseau-centrées*, qui vise à élaborer un réseau tactique partagé par l'ensemble des entités engagées sur un théâtre d'opération. Ce réseau a notamment pour but de permettre :

- la mutualisation d'informations issues des différents observateurs du terrain (groupes de reconnaissance, drones, fantassins) pour aboutir à une amélioration de la situation tactique par les responsables opérationnels ;
- la mise à jour réactive des ordres de mission et le relai d'observations susceptibles de bénéficier aux entités confrontées à une menace.

Ce concept apporte donc un gain en termes de capacités opérationnelles (efficacité des actions, sécurité des acteurs). Il implique néanmoins des évolutions importantes dans la définition des systèmes communicants, qui nécessitent désormais le déploiement d'une infrastructure réseau essentielle à la tenue de la mission.

Afin de bénéficier au mieux des gains opérationnels de ces systèmes, il faut donc que les entités bénéficient d'une connexion en continu au réseau. Or, selon la configuration du terrain et les capacités de déploiement de l'infrastructure, cette connexion peut être faible (en termes de bande passante), voire nulle. C'est pourquoi il peut être judicieux de considérer une contrainte de bande passante minimale *dès la phase de planification* de ces entités, afin de garantir au mieux l'accès au réseau. Dans la suite de ces travaux, nous considérerons uniquement la problématique de planification des véhicules.

Nous avons donc mis au point une métrique correspondant à la bande passante disponible par un véhicule le long des arcs du graphe. Présenté ci-après, le modèle de communication radio est simpliste, car non réaliste d'un point de vue physique (de nombreux facteurs physiques entrent en jeu dans la qualité d'un lien de communication radio). Cependant, comme dans le cas du problème de gestion d'énergie de la section précédente, il permet d'induire une complexité de calcul du point de vue de la planification. A la différence de la métrique de consommation toutefois, la contrainte sur la bande passante n'est pas additive mais appliquée à chacun des arcs empruntés par le véhicule. Nous expérimentons donc par le biais de ce problème l'application de l'approche ACO de la section 3.1.

4.2.2 Modèle de liaisons radio

Nous présentons ci-après 3 types de radio, correspondant aux technologies disponibles par les forces terrestres :

- la radio dite « tactique » : c'est la radio classique dont dispose l'ensemble des forces. Elle offre une bonne capacité de réception dans l'ensemble des situations, hormis les zones très enclavées. Le débit offert est cependant faible : quelques dizaines de kilobits par seconde ;

– la radio de type *Wi-Fi* (normes 802.11) et *WiMAX* (normes 802.16) : elle offre un débit beaucoup plus important que la radio tactique, mais ce débit tend à décroître rapidement avec la vitesse de déplacement des véhicules.
– La radio de type satellitaire, qui offre un large débit pourvu que le véhicule soit statique ou se déplace à très faible vitesse. Au delà, la connexion est perdue (phénomène de décrochage).

Nous supposons que les véhicules ont à disposition ces 3 technologies.

Ainsi, la radio satellitaire peut être modélisée comme suit :

$$b_{sat}(e) = \begin{cases} B_{sat} \text{ si } s(e) \leq s_{sat} \\ 0 \text{ sinon.} \end{cases} \tag{4.13}$$

où $s(e) = \frac{d(e)}{t(e)}$ est la vitesse du véhicule le long de l'arc e, B_{sat} est la bande passante de la radio satellitaire et S_{sat} est la vitesse de décrochage (toute réception est compromise au-delà de $s(e) = s_{sat}$). L'équation peut s'écrire de manière équivalente par :

$$b_{sat}(e) = B_{sat}.\left\lfloor \min\left\{1, \frac{S_{sat}}{s(e)}\right\}\right\rfloor \tag{4.14}$$

où $\lfloor x \rfloor$ désigne l'arrondi à la valeur entière inférieure de x.

La radio tactique peut être modélisée par la fonction suivante :

$$b_{tac}(e) = C_1(e).B_{tac} \tag{4.15}$$

où $C_1(e)$ est une constante dans $[0, 1]$, exprimée en pourcentage, désignant la qualité de réception de la radio tactique le long de l'arc e, et B_{tac} est la bande passante offerte par la radio tactique. Ce modèle ne dépend pas de la vitesse du véhicule.

La radio de type *WiMax* peut être modélisée (naïvement) par :

$$b_{wi}(e) = \max\left\{B_{wi}, C_2(e).\frac{\alpha}{s(e)^2}\right\} \tag{4.16}$$

B_{wi} est la bande passante maximale disponible pour la radio *WiMAX*, $C_2(e)$ est la qualité de réception de cette radio le long de l'arc e et α est une constante utilitaire.

La figure 4.4 illustre la bande passante disponible pour chacune des radios le long d'un arc e, en fonction de la vitesse $s(e)$ du véhicule. Les paramètres choisis pour le modèle sont les suivants :

– $B_{tac} = 100$ko/s ;
– $B_{wi} = 1200$ko/s ;
– $B_{sat} = 1600$ko/s ;
– $S_{sat} = 20$km/h ;
– $\alpha = 320000$.

Finalement, la bande passante disponible le long d'un arc e, à la vitesse $s(e)$, est égale à :

$$b(e) = \max\left\{b_{tac}, b_{wi}, b_{sat}\right\} \tag{4.17}$$

$b(e)$ est donc une fonction définie par morceaux. La contrainte de bande passante imposée pour le problème est la suivante :

$$\forall e \in p,\ b(e) \geq B_{min} \tag{4.18}$$

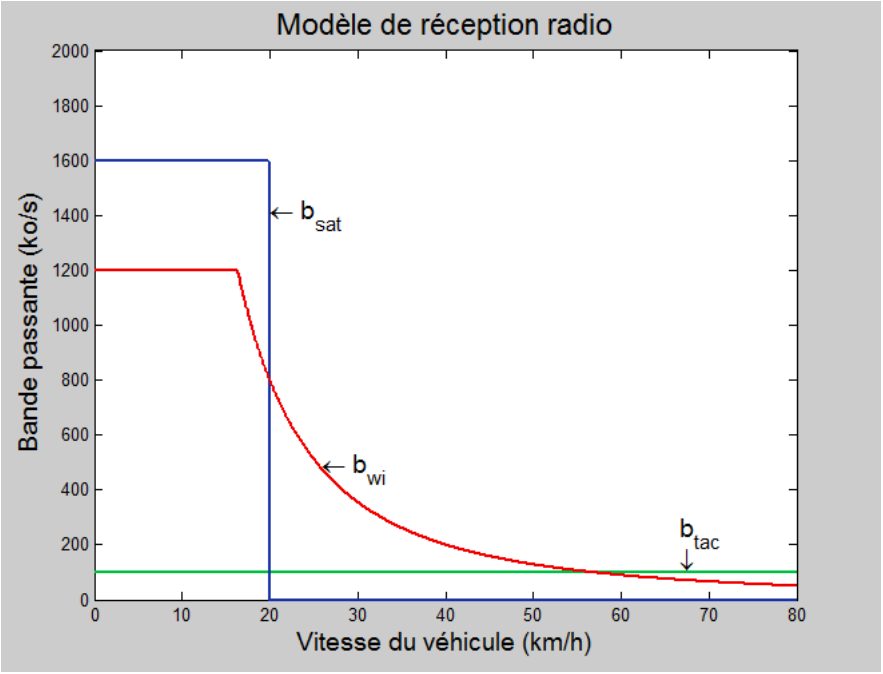


FIGURE 4.4 – Bande passante du modèle de radio présenté, en fonction de la vitesse $s(e)$ du véhicule. Les constantes $C_1(e)$ et $C_2(e)$ sont ici égales à 1 (qualité de réception maximale).

où B_{min} est un paramètre d'entrée du problème. La difficulté apportée par cette contrainte est due à la non-linéarité de b_{wi}, ainsi qu'au caractère spécifique des constantes C_1 et C_2 (potentiellement différentes pour chaque arc).

Ici, tout comme dans le problème de gestion d'énergie, le chemin le plus court n'est pas nécessairement le plus rapide car le véhicule est contraint de fortement ralentir pour maintenir la qualité de la liaison radio dans les zones atténuées. La stratégie de résolution doit donc prendre en compte ces contraintes afin d'assurer un guidage efficace du solveur de contraintes.

4.2.3 Résolution

4.2.4 Solveur partiel

Contrairement au problème de gestion d'énergie traité en section 4.1, il ne s'agit pas ici d'un problème de contraintes additives, mais simplement d'une borne maximale sur chacune des valeurs de bande passante associées aux arcs. Ce problème est ainsi plus simple à résoudre, puisque la vitesse de véhicule sur un arc ne dépend que des propriétés de cet arc (et non de ceux de l'ensemble du chemin).

Aussi, pour la résolution partielle, il est possible de réutiliser l'approche basée sur ACO afin de guider la résolution globale. Il n'est pas nécessaire de recourir à l'approche LARAC-ACO ; cependant, il faut modifier les valeurs de coût des arcs du graphe d'environnement afin que celles-ci prennent en compte les contraintes de bande passante. Plutôt que de considérer la distance des arcs, ce sont donc ici les temps de traversée qui sont évalués. Ceux-ci sont déterminés comme suit :

$$\forall e \in E, t(e) = \begin{cases} 0 \text{ si } B_{min} > B_{sat} \\ \frac{d(e)}{S_{sat}} \text{ si } B_{sat} \geq B_{min} > B_{wi} \\ \frac{d(e)}{s_{wi}(e)} \text{ si } B_{wi} \geq B_{min} > B_{tac} \\ \frac{d(e)}{S_{max}} \text{ si } B_{min} \leq S_{tac}. \end{cases} \tag{4.19}$$

où S_{max} est la vitesse maximale autorisée pour le véhicule. $s_{wi}(e)$ est la vitesse maximale à laquelle peut rouler ce véhicule sans compromettre la valeur de bande passante imposée, définie par l'équation 4.21. Sachant que

$$b_{wi}(e) = C_2(e).\frac{\alpha}{s(e)^2} \geq B_{min} \tag{4.20}$$

il vient

$$s_{wi}(e) \leq \sqrt{C_2(e).\frac{\alpha}{B_{min}}} \tag{4.21}$$

Ce modèle est propre au solveur partiel, car la valeur de t(e) est ici fixée. Il n'est valable que parce que le problème traité est une relaxation, et ne saurait être répété dans la résolution globale (dont les variables de temps ou de vitesse peuvent éventuellement être soumises à d'autres contraintes, pouvant rendre les égalités de l'équation 4.19 irréalisables). En revanche, nous pouvons tirer parti de ce modèle sous forme d'inégalités, ce qui est présenté maintenant.

4.2.5 Solveur global

Au niveau du solveur global, nous avons également apporté quelques modifications au modèle de bande passante initial. En effet, implémenter les équations 4.14, 4.15 et 4.16 sans autre contrainte peut mener à des temps de calculs élevés, même pour des problèmes de petite taille. En effet, chaque fois que la valeur d'une variable de temps est fixée, le solveur doit calculer la valeur des trois bandes passantes et déterminer la valeur maximale parmi ces trois valeurs. Or, il est possible de mettre en place des bornes sur la valeur de vitesse (et donc indirectement de temps) en utilisant les informations du modèle. Ces bornes nous permettent de réaliser une coupe sur les domaines de valeurs des variables de vitesse et ainsi d'accélérer fortement le temps d'obtention d'une solution. Les bornes sont introduites de manière similaire aux équations 4.19 et 4.21, à la différence qu'il s'agit cette fois de contraintes de supériorité (et non d'égalité stricte, afin d'être en mesure de spécifier des contraintes supplémentaires (condition de généricité de nos travaux) :

$$\forall e \in E, t(e) \geq \begin{cases} 0 \text{ si } B_{min} > B_{sat} \\ \frac{d(e)}{S_{sat}} \text{ si } B_{sat} \geq B_{min} > B_{wi} \\ \frac{d(e)}{s_{wi}(e)} \text{ si } B_{wi} \geq B_{min} > B_{tac} \\ \frac{d(e)}{S_{max}} \text{ si } B_{min} \leq S_{tac}. \end{cases} \tag{4.22}$$

Etant défini sur $[0, \infty[$, $t(e)$ vaut nécessairement 0 si $B_{min} > B_{sat}$.

4.2.6 Résultats

Nous ne présentons pas ici les résultats comparatifs des approches avec et sans contraintes de coupe sur les variables de temps (équation 4.19). La comparaison peut être faite en se référant à

nos travaux publiés à MILCOM'10 [Luca 10], dans lesquels ces coupes n'avaient pas encore été mises en œuvre (les temps de calcul y sont bien plus importants). Ici, seul le modèle avec coupes est employé. De plus, toutes les méthodes de résolution partielles utilisent le graphe intégrant la contrainte de bande passante dans les valeurs de coût (voir section 4.2.4).

Le tableau 4.3 présente les résultats de trois approches, que nous avons comparées :

1. une approche avec un solveur partiel utilisant un algorithme de plus court chemin entre v_s et v_g, noté ici « SP » ;

2. une approche avec un solveur partiel utilisant l'algorithme basé sur ACO (présenté en section 3.1), notée ici « ACO » ;

3. une approche identique à l'approche ACO, à laquelle nous avons adjoint des contraintes de supériorité sur les temps de la solution partielle dans le solveur global (de manière similaire aux travaux menés en section 4.1.2.4), et notée « ACO(2) » dans la suite de la section.

Ces trois approches ont été testées sur un benchmark composés de trois scénarios réels, dans lesquels une zone de l'environnement a été volontairement pénalisée (les valeurs $C2$ associées aux arcs de cette zone ont été atténués fortement) afin d'inciter la recherche à éviter cette zone. Pour chacun des scénarios, six séries (de 0 à 5 points de passage) de dix instances ont été générés. Le tableau résume, pour chaque série, les temps moyens T_1 d'obtention de la solution optimale et temps moyens T_2 d'obtention de la preuve d'optimalité. Les temps de résolution ont été limités à 500 secondes : le tableau indique entre parenthèses, dans la colonne T_2, le nombre d'instances n'ayant pas abouti à une preuve d'optimalité dans ce laps de temps.

Le détail des expérimentations est fourni en annexe 5.3.4. Les essais ont été réalisés sur un ordinateur portable *Samsung Q310* équipé d'un micro-processeur *Intel Core 2 Duo P7350* cadencé à 2.0GHz, et de 3Go de mémoire vive.

Le tableau montre que les performances de l'approche SP sont très bonnes sur les scénarios 1 et 2. Ainsi, le modèle de coupe mis en place pour la métrique de bande passante ainsi que le propagateur (déjà mis en place pour les contraintes d'énergie) permettent de résoudre en temps très restreint ce problème. Néanmoins, on observe sur le scénario 3 (pour 3, 4 et 5 points de passage) que l'approche est très peu efficace, et dépasse largement la dizaine de secondes. Pour la série avec 5 points de passage, on observe même que trois instances n'obtiennent pas la preuve d'optimalité dans le délai fixé de 500 secondes. L'approche ACO, *a contrario*, se révèle parfois un peu moins performante sur les scénarios 1 et 2. Par exemple, elle met jusqu'à 5 secondes de plus dans le scénario 1, pour $|V_m| = 3$, que l'approche SP. En revanche, les temps de calcul obtenus sur le scénario 3 sont beaucoup plus raisonnables, avec 11 secondes en moyenne dans le pire cas (contre 66 pour SP). De plus, cette moyenne intègre l'ensemble des dix instances (contre seulement 7 pour SP, qui échoue 3 fois). Les performances de l'approche ACO sont donc parfois légèrement en retrait par rapport à celles de SP, mais toujours beaucoup plus raisonnables dans les pires cas. Quant à l'approche ACO(2), qui utilise les valeurs de temps de la solution partielle pour guider le choix des valeurs du solveur de contraintes lorsque le modèle de flot est cohérent, elle permet un gain notable sur plusieurs instances « problématiques » pour ACO. Les deux exemples les plus probants sont les jeux d'instances 5 des scénarios 2 et 3, où les délais d'obtention de la preuve passent de $6,1$s à $1,1$s et de $11,5$s à $5,3$s respectivement. Cette approche peut donc tout à fait être utilisée dans un cadre embarqué, pour des tailles de problèmes équivalents.

Scénario	$\|V_m\|$	Résultats SP		Résultats ACO		Résultats ACO(2)	
		T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)
Scénario 1	0	142	344	143	346	17	304
	1	59	260	171	386	43	294
	2	443	737	185	483	129	539
	3	1065	2002	5051	7041	4227	6495
	4	794	1467	471	1107	152	717
	5	550	1302	2670	3876	123	1138
Scénario 2	0	240	519	1499	1810	9	268
	1	436	814	1118	1667	173	645
	2	433	812	1035	1449	190	578
	3	569	1330	1658	2875	71	912
	4	466	1221	372	1692	240	956
	5	546	1341	4600	6148	333	1112
Scénario 3	0	96	234	301	781	322	730
	1	1196	1441	337	925	123	823
	2	131	507	923	1873	836	1794
	3	28095	28861	2364	4433	1128	3365
	4	16010	17297	1675	4945	1889	5238
	5	46433	66748 **(3)**	8729	11587	2307	5300

TABLE 4.3 – Temps moyens du solveur de contraintes avec trois approches de résolution partielle : SP (*shortest path*), ACO et ACO(2). T_1 désigne le temps d'obtention de la solution optimale, et T_2 désigne le temps d'obtention de la preuve d'optimalité (fin de la recherche). Les valeurs entre parenthèses correspondent au nombre de recherches dont la preuve n'a pas été obtenue dans un délai de 500 secondes (non considérées dans le calcul de la valeur moyenne dans ce cas).

4.2.7 Discussion

Une première approche basée sur une application directe présentée en section 3.1 avait été présentée dans une communication à MILCOM'10 [Luca 10]. Or, les performances de l'approche présentée ci-avant sont bien meilleures que celles publiées à l'époque puisque deux ordres de grandeur ont été gagnés dans certains cas. Pourtant, l'unité de précision de la solution était la minute alors qu'il s'agit ici de la seconde ! Cela s'explique notamment par le nouveau modèle logique désormais employé. D'une part, le propagateur spécifique (voir section 4.1.2.2) permet de vérifier beaucoup plus rapidement la cohérence d'un chemin (absence de cycles externes au chemin) que ne le faisaient les variables de temps dans le modèle précédent. D'autre part, la mise en place de coupes sur les variables de temps (équation 4.22) permet d'écarter immédiatement les valeurs non réalisables au regard de la valeur B_{min} de bande passante imposée. De plus, la prise en compte des contraintes radio dans la résolution partielle permet d'améliorer le guidage et donc d'orienter la recherche du solveur de contraintes vers des solutions réalisables. Enfin, l'utilisation temporaire des valeurs de temps de la solution partielle en tant que borne supérieure sur les valeurs de temps du modèle logique permet de guider très rapidement le solveur de contraintes vers une solution de faible valeur de coût. Ces améliorations permettent, au vu des résultats obtenus, d'envisager une utilisation du solveur dans un cadre embarqué.

4.2.8 Conclusion

Nous avons proposé, dans cette section, une application de notre approche hybride pour un problème spécifique de gestion de bande passante de communication. Bien qu'en présence, de manière similaire au problème de gestion d'énergie traité plus tôt, de contraintes non linéaires

liées aux variables de temps du problème, il s'agit ici d'un problème de contraintes non additives mais propres à chaque arc du graphe. En réalité, ce problème est plus simple à résoudre qu'un problème de contraintes additives, pour laquelle la limite de capacité s'applique à la globalité de la solution. Ici, il est possible de déterminer pour chaque arc la valeur limite admissible pour la réalisabilité de la solution, sans avoir à recourir à une relaxation lagrangienne.

Néanmoins, les travaux publiés dans MILCOM'10 [Luca 10] montrent qu'en l'absence d'une approche spécifiquement adaptée à la gestion de contraintes de capacité, l'utilisation d'un solveur générique peut aboutir à des performances très moyennes. Cela est dû au caractère non linéaire des contraintes, rendant inefficaces les mécanismes de propagation de Programmation par Contraintes (basés sur une gestion de contraintes linéaires). Or, avec une approche plus fine employant à la fois des coupes (utilisant la connaissance du problème), en exploitant les valeurs de temps de la solution partielle, et en intégrant les contraintes dans la fonction de coût du solveur partiel, nous avons montré qu'il est tout à fait pertinent d'utiliser notre approche pour ce type de problème.

4.3 Exploration de zones d'intérêt en robotique mobile

La robotique mobile est un domaine de recherche en plein essor dans le milieu militaire, depuis quelques années, en raison des multiples capacités qu'elle offre. Dans les zones urbaines notamment, où le besoin d'observation est essentiel pour assurer la sécurité des opérations, les travaux sont nombreux. Dans cette section, nous proposons une application de notre outil de planification pour des problématiques d'exploration robotique, dans lesquelles il faut maximiser le nombre de zones à observer (appelés *points d'intérêt* par la suite) tout en satisfaisant une limite de temps à ne pas dépasser. Derrière cette application, nous démontrons qu'il est possible d'utiliser l'approche de résolution hybride proposée pour satisfaire une fonction de coût autre que celle du temps de mission.

4.3.1 Présentation du problème

Nous traitons en exemple d'application un problème dans lequel un robot dispose d'un graphe de l'environnement. Il a un ensemble de tâches à réaliser en un temps imparti (ces tâches n'étant pas nécessairement toutes réalisables en une seule mission). Il doit par conséquent déterminer le plan de navigation permettant de maximiser le nombre de tâches à réaliser. Une tâche est assimilée à une action en un point du graphe, et d'une durée fixée. Pour simplifier, nous considérerons que toutes les tâches ont une durée nulle : par conséquent, il suffit que ce robot passe à l'endroit indiqué pour avoir effectué cette tâche. Cette hypothèse n'est pas réaliste, mais cela ne change pas la structure du problème.

Ce problème est formellement défini comme suit. Étant donné un graphe $G = (V, E)$, un point de départ $v_s \in V$, un point d'arrivée $v_g \in V$, un ensemble de points d'intérêt $V_m \subseteq V$ et une échéance de temps de mission T_{max}, le problème consistant à maximiser la visite des points d'intérêt est donnée par :

$$\min_{p \in P''(v_s, v_g)} \sum_{e \in p} t(e) \tag{4.23}$$

avec

$$P''(v_s, v_g) = \left\{ \max_{p \in P'(v_s, v_g)} | \ p \cap V_m | \right\} \tag{4.24}$$

et

$$P'(v_s, v_g) = \left\{ p \in P(v_s, v_g) \mid \sum_{e \in p} t(e) \leq T_{max} \right\} \tag{4.25}$$

$P(v_s, v_g)$ est l'ensemble des chemins de v_s à v_g dans G. L'équation 4.25 définit que $P'(v_s, v_g)$ est l'ensemble des chemins de $P(v_s, v_g)$ dont le temps de mission total est inférieur à T_{max}. L'équation 4.24 définit $P''(v_s, v_g)$ comme le sous-ensemble de chemins de $P'(v_s, v_g)$ tel que le nombre de points d'intérêts visités (donc présents dans p) est maximum. Enfin, l'équation 4.23 spécifie que, s'il existe plusieurs chemins tel que le nombre de points de passage est maximum, la solution optimale est celle qui minimise le temps total de mission. Cette dernière équation s'impose car la maximisation du nombre de points d'intérêts est insuffisante pour assurer une solution de qualité (surtout si la condition imposée par T_{max} est faible).

Le problème proposé est donc un problème d'optimisation hiérarchique, consistant tout d'abord à maximiser le nombre de points d'intérêts à visiter *puis* à minimiser le temps de parcours total.

4.3.2 Adaptation de l'approche

4.3.2.1 Modèle logique

Pour modéliser en Programmation par Contraintes le problème présenté ci-avant, nous reprenons en grande partie le modèle existant :

$$\sum_{e \in \omega^+(v_s)} \phi(e) = \sum_{e \in \omega^-(v_g)} \phi(e) = 0 \tag{4.26}$$

$$\sum_{e \in \omega^-(v_s)} \phi(e) = \sum_{e \in \omega^+(v_g)} \phi(e) = 1 \tag{4.27}$$

$$\forall v \in V \setminus \{v_s, v_g\}, \sum_{e \in \omega^+(v)} \phi(e) = \sum_{e \in \omega^-(v)} \phi(e) \leq 1 \tag{4.28}$$

En revanche, les flots des points d'intérêt V_m ne peuvent plus être forcés à 1 (comme cela était fait dans l'équation 4.29) car cela mènerait à une solution irréalisable si tous les points ne peuvent être visités dans le temps imparti. Cette équation est donc remplacée par :

$$\Phi_m = \sum_{v_m \in V_m} \sum_{e \in \omega^+(v_m)} \phi(e) \tag{4.29}$$

où Φ_m est la somme des flots des points d'intérêt, qu'il va falloir maximiser.

4.3.2.2 Approche de résolution partielle

Pour la résolution partielle, nous sommes repartis de l'approche basée sur ACO (voir section 3.1) avec construction bidirectionnelle. Et pour adapter l'algorithme à ce nouveau problème, seule une modification très mineure est nécessaire.

Cette modification intervient au niveau de la construction d'une solution. Dans le problème initial, une fourmi construit une solution en sélectionnant itérativement un prochain point de passage, et ce tant que tous les points n'ont pas été visités. Il est désormais nécessaire, préalablement à l'évaluation de chaque nœud candidat, de s'assurer que l'ajout de ce nœud permette

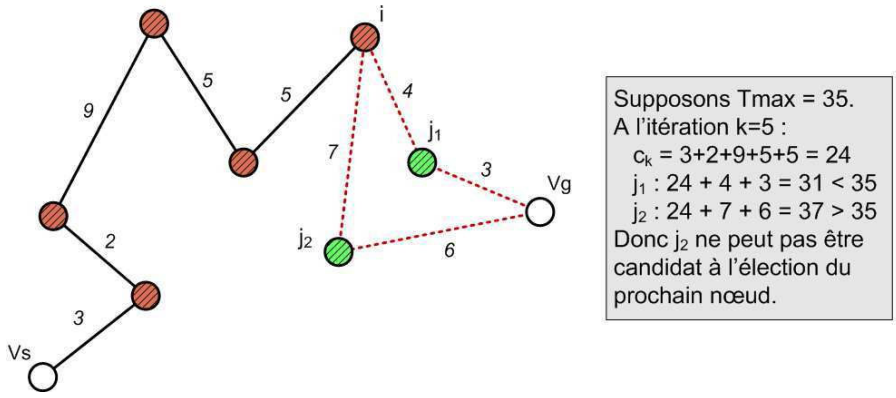


FIGURE 4.5 – Fonction de génération du facteur de consommation d'un arc, selon sa distance.

toujours d'atteindre la position finale désirée. Pour cela, nous disposons à tout moment du coût c_k à l'itération k de la solution en cours de construction. Nous disposons également de la matrice de coûts, permettant de connaître le coût pour atteindre le nœud candidat j depuis le nœud courant i ainsi que le coût pour atteindre le nœud final (par exemple v_g) depuis le nœud candidat j. On s'assure donc que tous les nœuds candidats j respectent la contrainte :

$$c_k + c(i,j) + c(j,v_g) \leq T_{max}$$

La figure 4.5 illustre l'intérêt de cette contrainte. La fonction de coût utilisée est bien entendu une fonction du temps (calculée pour $s = S_{max}$). Si aucun candidat restant ne respecte la contrainte, l'étape de construction s'achève en ajoutant le nœud v_g en queue de solution.

En ajoutant cette vérification (équivalente à une approche de *forward checking* dans la littérature, puisqu'on vérifie à chaque instanciation que celle-ci mène à une solution réalisable), on élimine ainsi les nœuds candidats qui violent la contrainte de temps imposée. De la même manière que le solveur global, l'évaluation des solutions pour le renforcement du modèle de phéromones est réalisée hiérarchiquement : on sélectionne en premier lieu les solutions qui maximisent les points d'intérêts visités, puis on élit celle dont le coût est minimal. La fonction utilisée pour le renforcement des solutions est conservée telle quelle, et ne fait pas intervenir le nombre de points de passage traversés.

Il est à noter que l'usage de fourmis d'amélioration a ici un intérêt limité. En effet, effectuer une perturbation à la meilleure solution courante, puis une recherche locale sur la solution obtenue, permet d'obtenir une solution de distance potentiellement plus faible mais ne permet pas d'obtenir une solution disposant de davantage de points d'intérêt. On pourrait donc imaginer mettre au point une heuristique d'insertion permettant, après recherche locale, de tenter d'introduire des points supplémentaire si le coût de la solution nous y autorise. Nous n'avons pas ici mené ces travaux, l'application du solveur au problème de robotique étant antérieure à l'introduction des fourmis d'amélioration dans notre algorithme ACO. Nous plaçons ces idées en perspectives futures de cette thèse.

4.3.2.3 Guidage avancé du solveur

Pour tirer pleinement parti des résultats de la résolution partielle, nous ne nous sommes pas contentés de trier les variables de flots à l'aide de la méthode de sonde présentée dans le

cadre de notre approche (voir section 3.1.3). Nous avons également mis en place des contraintes temporaires qui, si les flots de la solution évaluée dans le solveur de contraintes correspondent à la solution partielle, permettent d'appliquer une borne basse sur le nombre de points d'intérêts visitables dans le temps T_{max} imparti. Ainsi, on définit que :

$$\Phi_m \geq |p \cap V_m| \tag{4.30}$$

On définit également que

$$\forall e \in E, t(e) \leq t_p(e) \tag{4.31}$$

où $t_p(e)$ est le temps associé à l'arc e dans la solution partielle (0 si $\phi(e) = 0$). Il est à noter que si les valeurs de flots ne correspondent pas au chemin de la solution partielle, ces contraintes ne sont pas applicables. De plus, pour assurer la complétude de l'approche, ces contraintes sont relaxées à l'issue de l'évaluation de l'ensemble des solutions du sous-problème.

4.3.3 Expérimentation et résultats

L'expérimentation a été menée sur des instances issues du *benchmark* du problème de gestion d'énergie (voir section 4.1). Les contraintes de capacité ne sont pas ici prises en compte. En revanche, pour augmenter la difficulté des instances, nous avons tout d'abord résolu celles-ci sans prendre en compte de bornes sur le temps de la mission. Les solutions obtenues sont par conséquent les chemins qui minimisent le temps de parcours en passant par l'ensemble des points d'intérêt (puisque leur nombre est maximisé). Nous avons alors réutilisé les temps de réalisation de la mission de chacune des solutions pour les appliquer, après décrément de 1, en tant que bornes maximales des instances pour notre expérimentation. Par exemple, la solution à l'instance *1_6* sans contraintes est un chemin passant par les 5 points d'intérêt imposés et d'un temps total de 359 secondes. Nous avons alors fixé, pour notre expérimentation, un temps maximal $T_{max} = 358$s pour la réalisation de mission. De cette manière, la solution à ce problème ne peut passer par les 5 points d'intérêt spécifiés, et le solveur doit alors trouver le chemin qui optimise le temps en trouvant la meilleure combinaison de points d'intérêt. Les instances ne disposant pas de points d'intérêt ne sont pas ici présentées, car sans intérêt (puisqu'on ne peut les contraindre à un coût plus faible que le coût de la solution sans contrainte).

Les expérimentations ont été menées sur un ordinateur portable *Samsung Q310* équipé d'un microprocesseur *Intel Core 2 Duo P7350* et de 2Go de mémoire vive. À la différence des expérimentations précédentes, pour mieux refléter les performances de calcul des équipements de robotique mobile (aux capacités de calcul plus faibles qu'un ordinateur portable), **l'ordinateur a été configuré en mode « économie d'énergie »**, dans lequel le processeur ne tourne qu'à 40% de sa fréquence nominale, soit 800MHz.

Nous comparons dans le tableau 4.4 trois approches :

- une approche basée sur une résolution partielle de plus court chemin (SP pour *shortest path* hybridé avec l'ancienne implémentation du solveur de contraintes (sans propagateur spécifique, sans définition dynamique des bornes de variables) ;
- une approche basée sur une résolution partielle de plus court chemin (SP) avec la nouvelle implémentation du solveur de contraintes ;
- une approche basée sur une résolution partielle de colonies de fourmis (ACO) avec la nouvelle implémentation du solveur de contraintes.

Les performances de l'ancien solveur sont ici mentionnées à titre indicatif, afin de souligner également les gains en performances apportés par l'amélioration de l'implémentation du code Prolog.

Instance	T_{max}	SP ancien modèle		SP nouveau modèle		ACO nouveau modèle	
		T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)
1_3	288	9968	21871	1529	2761	15	749
1_4	215	7770	15320	592	1060	15	593
1_5	324	36505	46130	5585	6739	16	2247
1_6	358	2683	11404	359	1201	31	1076
2_3	327	2777	4742	406	671	16	515
2_4	407	1950	7753	125	452	16	296
2_5	509	15522	21762	1887	2371	296	998
2_6	649	10951	16583	639	764	16	219
3_3	1269	9017	15756	1436	2106	0	1497
3_4	1222	36725	49189	4695	6115	2355	3912
3_5	1540	45899	59751	9064	11092	15	3338
3_6	1469	11014	19407	2855	4275	2854	4212

TABLE 4.4 – Résultats des approches de résolution avec solveur partiel de plus court chemin (SP) et de colonies de fourmis (ACO), sur les instances tirées du *benchmark* de gestion d'énergie. Les cases T_1 et T_2 présentent respectivement les temps nécessaires au solveur de contraintes pour trouver la solution optimale, et pour obtenir la preuve d'optimalité (fin de la recherche).

En observant les résultats des trois approches, on peut tout d'abord noter les progrès réalisés dans la rapidité d'obtention de preuve d'optimalité avec la nouvelle approche. De plus, en comparant les approches SP et ACO, on note que la seconde est davantage performante sur l'ensemble des instances. ACO permet notamment dans certains cas de diviser sensiblement le temps de calcul et d'obtention de preuve d'optimalité, même sur des instances difficiles (*3_5* par exemple). Cet avantage est obtenu par l'adaptation faite des colonies de fourmis, qui sont à même de déterminer le chemin passant par les points d'intérêt qui minimisent le temps de trajet global. L'approche hybride implémentant ACO démontre ainsi un autre de ses avantages : sa capacité d'adaptation, en quelques lignes de code, à des problèmes d'optimisation de fonctions de coût différentes. Quelques instances restent problématiques pour l'approche ACO (instances *3_4* et *3_6*), toujours en raison de la différence entre le modèle de flot du solveur de contraintes et la construction du chemin retourné par ACO. Cependant, notre approche est capable de résoudre l'ensemble des instances dans un délai inférieur à 4,5 secondes, dans des conditions de calcul en embarqué, ce qui est satisfaisant pour une application robotique.

4.3.4 Conclusion

Les travaux menés sur cette problématique de robotique ont permis de mettre en avant les qualités de notre approche pour des problèmes de planification de structure différente des problèmes traités jusque là. Nous avons montré comment, avec quelques adaptations simples, notre stratégie de résolution permet de résoudre efficacement un problème dont la fonction de coût n'est plus uniquement le temps de parcours de véhicule mais consistant à maximiser l'exploration dans un délai imparti. Un autre point important est à souligner ici : il s'agit de la capacité d'adaptation des colonies de fourmis. Les mécanismes d'exploration et d'apprentissage qui les caractérisent permettent tout à fait de résoudre des problèmes d'optimisation hiérarchique tels que ceux rencontrés ici. Leur adaptation à ce problème n'est d'ailleurs en rien incompatible avec les précédentes adaptations réalisées, ce qui permet d'imaginer par exemple de traiter des problèmes de robotique avec gestion d'énergie.

4.4 Conclusion

Dans ce chapitre, nous avons mis en application l'approche de résolution proposée, hybridant le solveur de contraintes avec :

- l'algorithme ACO dans le cas de contraintes non additives ;
- l'algorithme LARAC-ACO dans le cas de contraintes additives.

Dans le premier cas, un problème de planification avec contraintes non linéaires de communications radio a été traité. Dans le second cas, un problème de planification avec contraintes non linéaires de consommation d'énergie a également fait l'objet d'expérimentations. Un troisième cas pratique, dont la fonction d'optimisation n'est plus le temps de mission, a enfin été abordé afin d'étudier l'applicabilité de notre approche pour des problèmes de structure différente. Dans chaque cas, l'approche hybride a permis de contenir les temps de calcul (d'optimalité et de preuve) à des délais raisonnables pour le contexte opérationnel grâce à l'utilisation de méthodes de guidage appropriées, et ce malgré le caractère non linéaire des contraintes, à l'inverse de la méthode de guidage existante.

Chapitre 5

Extensions du modèle de navigation

5.1 Modèle de flots multiples

Comme cela a été plusieurs fois évoqué, et notamment en conclusion de la section 4.1, les modèles de chemin entre solveur partiel et solveur global ne sont pas identiques. En effet, le premier construit une solution en juxtaposant plusieurs chemins unitaires, ce qui peut l'amener à repasser plusieurs fois par un même nœud. Or, le modèle logique mis en place dans le solveur de contraintes n'autorise pas cette éventualité. Cela peut ainsi avoir des conséquence sur l'efficacité du guidage de la solution partielle.

Dans un cadre de navigation purement militaire, le modèle logique est cohérent puisque la doctrine impose aux véhicules de ne pas emprunter à nouveau une portion d'itinéraire déjà empruntée. En revanche, pour une application robotique par exemple, il n'y a aucune raison d'interdire à un robot de retourner sur ses pas. Nous avons donc mis en œuvre un nouveau modèle de flot afin d'autoriser les « flots multiples » au niveau des nœuds. Nous nous restreignons ici aux problèmes dont les fonctions de coût associées aux arcs sont symétriques (c'est-à-dire que pour toute fonction $c : E \to \mathbb{N}$, et pour couple d'arcs $(e = (u, v), e' = (v, u)) \in E^2$, l'équation $c(e) = c(e')$ est vérifiée). Sous ces hypothèses, les flots associés aux arcs du graphe restent de capacité unitaire. En effet, si un véhicule emprunte deux fois un même arc, cela signifie que la solution est sous-optimale. Pour le démontrer, supposons qu'une telle solution existe. Soit un chemin passant deux fois par une arête $e = (u, v)$. On peut alors écrire la solution (modélisée par la séquence des nœuds à parcourir) , par :

$$p = sp_1 \to u \to v \to sp_2 \to u \to v \to sp_3$$

sp_1, sp_2 et sp_3 sont trois sous-chemins correspondant respectivement au sous-chemin menant du nœud initial à u, au sous-chemin réalisant la boucle de v à u, et au sous-chemin allant de v au nœud final. Or, les coûts étant symétriques, on a $c(s\bar{p}_2) = c(sp_2)$ où $s\bar{p}_2$ désigne la séquence inverse de sp_2. Ainsi, il existe nécessairement un chemin p' tel que

$$p' = sp_1 \to s\bar{p}_2 \to sp_3$$

avec $c(p') < c(p)$. Dans le cas de fonctions de coût non symétriques, ce modèle n'est en revanche pas nécessairement valable. Il n'est également pas valable si des fenêtres de temps sont appliquées aux objectifs.

5.1.1 Nouvelle modélisation

Le modèle de flot initial, présenté en section 1.4.2, est ici légèrement modifié pour devenir :

$$\sum_{e \in \omega^+(v_s)} \phi(e) = \sum_{e \in \omega^-(v_g)} \phi(e) = 0 \tag{5.1}$$

$$\sum_{e \in \omega^-(v_s)} \phi(e) = \sum_{e \in \omega^+(v_g)} \phi(e) = 1 \tag{5.2}$$

$$\forall v \in V \backslash \{v_s, v_g\}, \sum_{e \in \omega^+(v)} \phi(e) = \sum_{e \in \omega^-(v)} \phi(e) \tag{5.3}$$

$$\forall v \in V_m, \sum_{e \in \omega^+(v)} \phi(e) = \sum_{e \in \omega^-(v)} \phi(e) > 0 \tag{5.4}$$

Les équations 5.1, 5.2 et 5.4 restent inchangées. En revanche, les termes de l'équation 5.3 ne sont plus cette fois restreints à une valeur dans $\{0, 1\}$. L'inégalité a donc disparu, et la somme des flots à un nœud v peut en pratique prendre une valeur $\{0, n\}$ où $n = \min\{|\omega^+(v)|, |\omega^-(v)|, |V_m|\}$. Le modèle mis en place garde une restriction : les nœuds de départ et d'arrivée ne peuvent toujours être empruntés qu'une fois.

Ce modèle, comme précédemment, n'est pas suffisant pour assurer la cohérence d'un chemin car des flots cycliques peuvent être présents dans la solution. Cependant, il n'est plus ici possible d'associer une valeur telle qu'un temps ou un identifiant unique à chaque nœud afin de mettre en place un propagateur, puisque chaque nœud peut être emprunté plusieurs fois. Pour le calcul des temps par exemple, le calcul qui était fait avec l'équation 1.12 est rendu caduque en raison d'opérateur somme (auparavant, une seule valeur de la somme pouvait être non nulle, ce qui n'est plus le cas). Il faut donc cette fois associer les variables du propagateur aux arcs du chemin.

5.1.2 Propagateur pour flots multiples

Le propagateur présenté ci-dessous est un propagateur spécifique, à la manière de celui présenté en section 4.1.2.2. Pour l'anecdote, ce sont en réalité les travaux sur les flots multiples qui nous ont amenés à repenser le modèle logique du solveur initial. En effet, utiliser la propagation des temps dans le modèle multiflots a mené à de piètres performances, ce qui nous a poussé à trouver une alternative (présentée ici), que nous avons ensuite étendu (avec simplification) au modèle uniflot.

Pour vérifier la cohérence d'un chemin, nous avons mis en place des variables appelées identifiants et notées id, associées à chaque arc du graphe. Le domaine de valeurs de ces identifiants est $\{0, |E|\}$ puisqu'il est en théorie possible, dans le pire cas, d'emprunter chaque arc du graphe (en pratique, il faudrait que ce graphe admette un chemin eulérien de v_s à v_g et que $|\omega^-(v_s)| = |\omega^+(v_g)| = 1$ en raison de la restriction à un passage unique au niveau de v_s et v_g). La valeur de ces identifiants est définie telle que :

$$\forall e \in E, id(e) \begin{cases} = 0 \text{ si } \phi(e) = 0 \\ > 0 \text{ sinon.} \end{cases} \tag{5.5}$$

$$\forall e = (u, v) \in E \setminus \{\omega^-(v_s)\}, \phi(e) = 1 \Leftrightarrow \exists e' \in \omega^+(u),\ id(e) = id(e') + 1 \tag{5.6}$$

$$\sum_{e \in \omega^-(v_s)} id(e) = 1 \tag{5.7}$$

$$\sum_{e \in \omega^+(v_g)} id(e) = \sum_{e \in E} \phi(e) \tag{5.8}$$

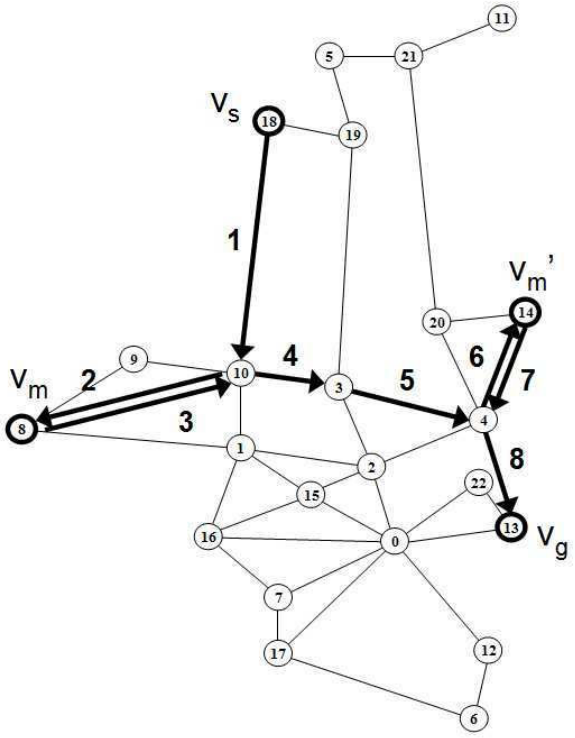


FIGURE 5.1 – Illustration d'une instanciation des identifiants d'arcs, pour le modèle multi-flots.

L'intuition de ce modèle est que si un arc $e = (u, v)$ est emprunté, alors son identifiant $id(e)$ est tel qu'il existe nécessairement un arc $e' = (w, u)$ précédant e et tel que $id(e) = id(e') + 1$. La figure 5.1 montre par un exemple le résultat d'une instanciation valide des identifiants. Formellement, l'équation 5.5 spécifie que l'identifiant d'un arc non emprunté est nul (et non nul dans le cas contraire). L'équation 5.6 spécifie que l'identifiant associé à un arc $e = (u, v) \in E$ correspond à l'incrément de l'identifiant d'un des arcs entrants de u si son flot est positif. L'équation 5.7 spécifie la valeur de l'identifiant de l'arc sortant (unique, d'après l'hypothèse que v_s n'est emprunté qu'une fois). L'équation 5.8 spécifie que la valeur de l'identifiant du dernier arc (égale au nombre d'arcs empruntés depuis v_s) doit être égale à la somme des flots dans G. Grâce à l'incrément des valeurs introduite dans l'équation 5.6, cette condition permet d'exclure toute instanciation qui présenterait au moins une boucle isolée du chemin (conditions de précédence violées). Indirectement, elle implique également que chaque identifiant ait une valeur unique.

Enfin, il est possible de réintroduire des temps de passage cumulés aux nœuds en les associant aux arcs, de manière similaire aux identifiants. Ces temps, dénotés t_f, sont calculés tel que :

$$\forall e \in E, t_f(e) = \begin{cases} t_f(e_p) + t(e) \text{ où } e_p = \{e' \in E \mid id(e) = id(e') + 1\} \text{ si } \phi(e) = 1 \\ 0 \text{ sinon.} \end{cases} \tag{5.9}$$

5.1.3 Résultats et discussion

Pour comparer les performances de notre solveur de contraintes avec modèle multiflots, nous avons repris le *benchmark* mis en place dans le cadre du problème de gestion de contraintes d'énergie (voir section 4.1). Les résultats de cette nouvelle approche sont directement comparés

Benchmark	Instance	Modèle uniflot		Modèle multiflots	
		T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)
Benchmark 1	mission_1_1	0 (189)	218	0 (189)	4009
	mission_1_2	16 (155)	125	0 (155)	983
	mission_1_3	15 (306)	546	0 (306)	166969
	mission_1_4	0 (224)	374	0 (224)	68110
	mission_1_5	0 (333)	717	0 (333)	392233
	mission_1_6	0 (363)	328	0 (355)	155751
Benchmark 2	mission_2_1	47 (372)	437	0 (372)	10249
	mission_2_2	16 (384)	359	0 (384)	1451
	mission_2_3	31 (401)	1685	0 (401)	9017
	mission_2_4	0 (502)	1997	0 (502)	29374
	mission_2_5	**496738 (678)**	–	0 (539)	23931
	mission_2_6	0 (787)	–	0 (787)	–
Benchmark 3	mission_3_1	47 (812)	749	0 (812)	6239
	mission_3_2	0 (665)	561	0 (665)	3338
	mission_3_3	2060 (2276)	–	0 (2276)	–
	mission_3_4	187 (1798)	–	0 (1798)	–
	mission_3_5	**496973 (2653)**	–	0 (1658)	–
	mission_3_6	**497300 (1568)**	–	0 (1441)	161664

TABLE 5.1 – Résultats comparatifs du solveur de contraintes avec modèle uniflot et multiflots. L'approche de résolution partielle utilisée est LARAC-ACO-M. Les cases T_1 et T_2 présentent respectivement les temps nécessaires au solveur de contraintes pour trouver la solution optimale et pour obtenir la preuve d'optimalité (fin de la recherche). Les valeurs indiquées entre parenthèse dans les cases T_1 sont les valeurs des meilleures solutions trouvées. Les cases présentant une police en caractères gras indique les cas où la recherche a échoué à trouver la solution optimale. Le délai maximum pour la recherche a été fixé à 500 secondes.

aux performances obtenues précédemment, avec modèle uniflot. Le tableau 5.1 résume les temps de calcul obtenus pour chaque approche sur les instances de problèmes proposées.

Les résultats obtenus montrent bien, maintenant que les modèles de chemin sont identiques entre solveur partiel et solveur global, que le guidage fonctionne parfaitement puisque l'approche multiflots trouve toutes les solutions optimales en une fraction de temps seulement. Les instances *2_5*, *3_5* et *3_6* qui posaient préalablement problème sont désormais résolues sans difficulté. On observe également sur l'instance *1_6* que le coût de la solution optimale est amélioré. En revanche, autoriser le passage multiple d'un véhicule à un nœud démultiplie le nombre des solutions possibles. Ces solutions ne sont plus écartées par le modèle logique et doivent donc être évaluées. Il en résulte des temps de calcul importants, comme cela peut être observé dans le tableau. Le cas le plus flagrant est celui de la mission *1_5*, dont la preuve d'optimalité est obtenue après quasiment 400 secondes contre moins d'une seconde auparavant.

Aussi, bien que le couplage avec la résolution partielle soit optimal, cette modélisation montre rapidement ses limites. Les performances obtenues sont très satisfaisantes pour un cadre d'utilisation *anytime*, dans lequel une solution peut être nécessaire à tout moment. En revanche, dans un cadre d'utilisation où la garantie d'optimalité est requise dans un temps réduit, le modèle multiflots présenté n'est pas celui à privilégier.

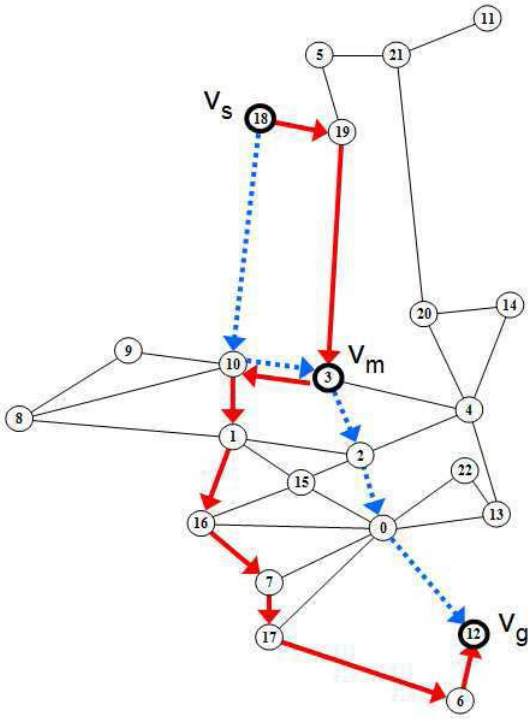


FIGURE 5.2 – Illustration de la solution au problème de planification robuste de l'instance $V_m = \{3\}$ du scénario 1 de l'expérimentation sur les contraintes de communications radio, pour le couple $(v_s, v_g) = (18, 12)$ (contraintes de capacité non considérées).

5.2 Planification robuste

5.2.1 Présentation

Nous présentons ici des travaux portant sur la mise en œuvre d'un solveur de planification « robuste », c'est-à-dire capable de définir non plus un chemin mais une paire de chemins disjoints. Cette application permet, dans le cas où un aléa intervient entre la définition des itinéraires et le début de la mission, de disposer à tout moment d'une solution alternative. La figure 5.2 illustre sur un exemple la solution à ce type de problème. Les possibilités d'une planification *proactive* sont donc ici étudiées. Dans les travaux qui suivent, nous avons limité la disjonction aux arcs (un même nœud peut donc être emprunté par les deux chemins) pour permettre l'application des contraintes de points de passage obligatoires. Deux fonctions d'optimisation sont analysées ci-après :

- *pire temps* : minimisation de la date maximale d'arrivée au nœud v_g, entre les deux chemins ;
- *temps total* : minimisation du temps cumulé le long des deux chemins.

Nous étudions ci-après plusieurs modélisations possibles pour ce type de problématique, ainsi que l'impact sur les performances de la résolution.

5.2.2 Modélisation avec système de flots unique

Nous étudions ici l'applicabilité d'un modèle basé sur un système de flots unique, c'est-à-dire un unique ensemble de variables de flots associé aux arêtes. Pour ce faire, nous avons adapté le modèle de flot de la section 1.4.2. Partant de ces équations, et en conservant la définition des flots $\phi(e) \in \{0, 1\}$ pour tout arc e de G afin d'assurer la disjonction en arcs, le modèle de flots devient :

$$\sum_{e \in \omega^+(v_s)} \phi(e) = \sum_{e \in \omega^-(v_g)} \phi(e) = 0 \tag{5.10}$$

$$\sum_{e \in \omega^-(v_s)} \phi(e) = \sum_{e \in \omega^+(v_g)} \phi(e) = 2 \tag{5.11}$$

$$\forall v \in V \backslash \{v_s, v_g\}, \sum_{e \in \omega^+(v)} \phi(e) = \sum_{e \in \omega^-(v)} \phi(e) \tag{5.12}$$

$$\forall v \in V_m, \sum_{e \in \omega^+(v)} \phi(e) = \sum_{e \in \omega^-(v)} \phi(e) = 2 \tag{5.13}$$

Les flots sortants de v_s et entrants de v_g sont désormais au nombre de deux, tout comme les flots entrant et sortant des points obligatoires. Il est à noter que pour ce problème, la disjonction appliquée aux arêtes complémentaires par l'équation 1.7 du modèle initial est abandonnée. En effet, elle pourrait aboutir à écarter les solutions dans lesquelles les chemins emprunteraient une même route, mais dans un sens contraire (comme dans l'exemple 5.2). Or ces solutions sont valides du point de vue opérationnel, car un seul chemin sera effectivement emprunté. La cohérence des chemins est assurée par la propagation de variables le long de ceux-ci, et dépend du problème d'optimisation à traiter.

5.2.2.1 Modélisation de l'optimisation *pire temps*

Pour modéliser ce problème, nous utilisons la propagation des valeurs de temps aux nœuds. Ceux-ci sont définis, pour chaque nœud v de G, comme le temps cumulé maximal des sous-chemins entrants :

$$t(v_s) = 0 \tag{5.14}$$

$$\forall v \in V \setminus \{v_s\},\ t(v) = \max_{e=(u,v) \in \omega^+(v)} \phi(e).(t(u) + t(e)) \tag{5.15}$$

La propagation des valeurs de temps assure en outre la cohérence du chemin, en imposant un ordre de précédence sur les arcs empruntés. Dans le modèle de flot initial, le calcul de $t(v)$ était réalisé en calculant une fonction somme, ce qui ne posait pas de problème compte tenu de l'unicité du flot entrant. Il est remplacé dans l'équation 5.15 par une fonction max, mais il est à noter que cette expression est valide pour le problème initial (seul le flot positif aboutit à une valeur non nulle, qui est retournée par max). La fonction de coût se définit donc simplement comme :

$$\min\ t(v_g) \tag{5.16}$$

5.2.2.2 Modélisation de l'optimisation *temps total*

Pour modéliser ce problème, il n'est pas nécessaire de propager de variables de temps le long des nœuds. Ces variables sont remplacées par un propagateur spécifique, composées de variables $id \in \left\{0, \frac{|E|}{2}\right\}$, associées aux arcs de G, assurant la cohérence du chemin :

$$\forall e \in E, id(e) \begin{cases} = 0 \text{ si } \phi(e) = 0 \\ > 0 \text{ sinon.} \end{cases} \tag{5.17}$$

$$\forall e = (u, v) \in E \setminus \{\omega^-(v_s)\}, \phi(e) = 1 \Leftrightarrow \exists e' \in \omega^+(u),\ id(e) = id(e') + 1 \tag{5.18}$$

$$\exists e \in \omega^-(v_s), id(e) = 1 \tag{5.19}$$

$$\exists e \in \omega^-(v_s), \exists e' \in \omega^+(v_g),\ id(e) = id(e') + 1 \tag{5.20}$$

$$\exists e \in \omega^+(v_g),\ id(e) = \sum_{e \in E} \phi(e) \tag{5.21}$$

Les équations 5.17 et 5.18 sont directement reprises du propagateur mis en place pour la modélisation multi-flots de la section précédente. En complément, ce propagateur définit donc un premier chemin dont les identifiants vont de 1 (équation 5.19) à une certaine valeur i (en v_g), et un second chemin allant de $i + 1$ (équation 5.20) à $\sum\ \phi(e)$.

La fonction de coût est alors définie comme la somme des temps de tous les arcs :

$$\min \sum_{e \in E} t(e) \tag{5.22}$$

où les temps associés aux arcs de flot nul sont fixés à 0.

5.2.3 Modélisation avec système de flots double

La modélisation avec système de flots double consiste tout simplement à dupliquer le modèle de flots mis en œuvre en section 1.4.2. Ce modèle est conservé dans son intégralité, et n'est pas modifié. Ainsi, dans la pratique, ce problème revient à traiter un problème multi-véhicules dont les arêtes des chemins de deux véhicules doivent être distincts. Aussi, il faut intégrer la contrainte suivante :

$$\forall ein E, \begin{cases} \phi_1(e) = 1 \Rightarrow \phi_2(e) = 0 \\ \phi_2(e) = 1 \Rightarrow \phi_1(e) = 0 \end{cases} \tag{5.23}$$

$\phi_1(e)$ désigne la variable de flot associée à e pour le premier chemin, et $\phi_2(e)$ celle associée à e pour le second chemin. Cette contrainte permet d'éviter qu'une même arête soit présente dans les deux chemins. Elle n'interdit cependant pas que deux arêtes complémentaires existent dans la solution. En effet, puisqu'un seul des deux chemins sera effectivement emprunté, une telle solution reste valide opérationnellement (comme dans l'exemple 5.2).

Cette modélisation présente *a priori* un inconvénient, qui est de doubler le nombre de variables (flots, temps, capacités...). Elle présente cependant l'avantage de pouvoir exprimer très simplement les deux fonctions d'optimisation (ce qui n'était pas le cas précédemment). Ainsi, on exprime le problème d'optimisation *pire temps* par

$$\min\ \max\{T_1, T_2\} \tag{5.24}$$

avec

$$T_1 = \sum_{e \in E} t_1(e) \text{ et } T_2 = \sum_{e \in E} t_2(e) \tag{5.25}$$

$t_1(e)$ (resp. $t_2(e)$ est la variable de temps associée au chemin 1 (resp. 2). Le problème d'optimisation *temps total* s'exprime quant à lui par

$$\min \sum_{e \in E} t(e) + \sum_{e \in E} t_2(e) \tag{5.26}$$

Pour l'expérimentation qui suit, nous avons réintroduit un propagateur de type *identifiants* décrit en section 4.1.2.2 afin d'améliorer les performances du solveur.

5.2.4 Méthode de guidage

Pour guider la résolution des problèmes présentés ci-avant, nous avons emprunté un algorithme connu du domaine des télécommunications : l'algorithme Suurballe-Tarjan [Suur 84]. Cet algorithme permet de déterminer, dans un graphe orienté pondéré positivement, le couple de chemins d'arcs disjoints dont le coût total est minimum (équivalent donc à l'optimisation de type *temps total*). Cet algorithme peut être utilisé pour les deux types d'optimisation présentés ci-avant car les solutions aux deux problèmes sont souvent identiques en pratique (pour les instances considérées).

L'algorithme 7 présente le principe de fonctionnement de la méthode, et la figure 5.3 illustre son exécution sur un exemple simple. Dans un premier temps, on utilise un algorithme de type Dijkstra pour calculer la valeur $c(v_s, v)$ de plus court chemin depuis v_s à chaque nœud $v \in V$, en utilisant un coût $c(e)$ associé aux arcs $e \in E$ fourni en entrée du problème. Puis on détermine le plus court chemin P de v_s à v_g dans G (étape 2 de la figure). Dans un troisième temps, on met à jour les valeurs de coût $c(e)$ en utilisant les valeurs de plus court chemin précédemment calculées : $c'(e) = c(e) - c(v_s, v) + c(v_s, u)$. Une fois les coûts mis à jour, on supprime de G les arcs de P afin d'éviter qu'ils soient à nouveau empruntés, et on définit un coût nul aux arcs complémentaires (c'est-à-dire de sens opposé) de P (étape 3). On détermine alors un nouveau chemin P' de v_s à v_g (étape 4). Enfin, si P et P' possèdent des arcs complémentaires, on supprime ceux-ci de la solution et on intervertit dans chaque solution les deux sous-chemins menant à v_g (étapes 5 et 6).

Algorithme 7 Algorithme Suurballe-Tarjan

Requiert: graphe $G = (V, E)$, sommet de départ v_s, sommet d'arrivée v_g, fonction de coût c
Renvoie: la paire de chemins de v_s à v_g dont le coût total est minimum

1: Calculer, $\forall v \in V \setminus \{v_s\}$, la valeur $c(v_s, v)$ de plus court chemin depuis v_s
2: Déterminer P le plus court chemin de v_s à v_g
3: Mettre à jour le coût des arcs : $\forall e = (u, v) \in E, c'(e) = c(e) - c(v_s, v) + c(v_s, u)$
4: Supprimer de G les arcs de P, placer à 0 les arcs complémentaires de P
5: Déterminer P' le plus court chemin de v_s à v_g dans le graphe modifié
6: Si P et P' contiennent des arcs complémentaires, supprimer ces arcs et intervertir les sous-chemins situés après
7: **retourner** P et P'

Nous avons intégré cet algorithme dans notre solveur partiel. Après avoir exécuté l'algorithme de colonies de fourmis afin de définir l'ordre dans lesquels emprunter les points de passage

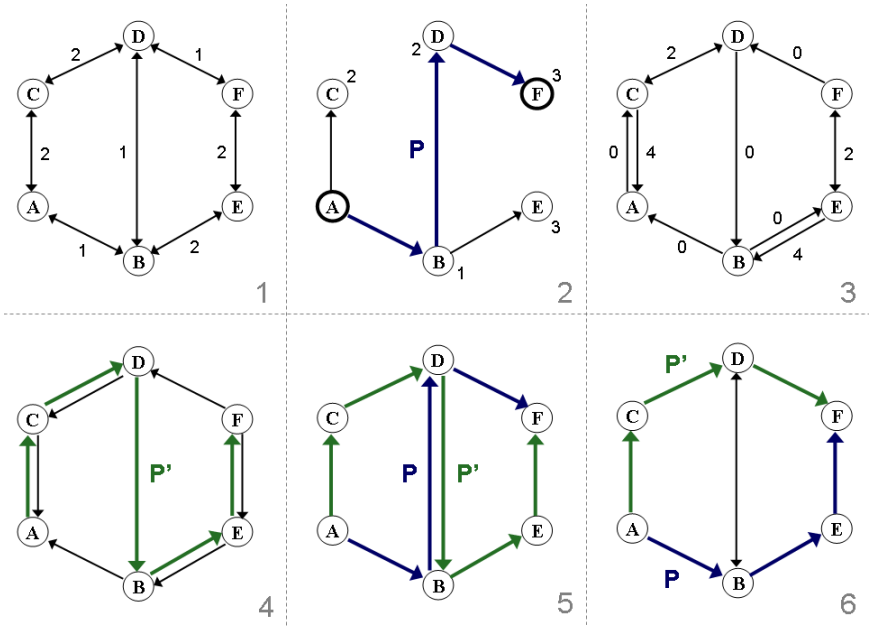


FIGURE 5.3 – Illustration des étapes de calcul de l'algorithme Suurballe-Tarjan (exemple tiré de la page *Wikipedia* anglophone dédiée à l'algorithme Suurballe). Dans cet exemple, le sommet A est le sommet de départ et F est le sommet d'arrivée.

obligatoires, nous appelons récursivement, pour chaque couple de nœuds obligatoires successifs de la séquence, l'algorithme Suurballe-Tarjan. Les deux chemins retournés sont alors ajoutés au couple de chemins précédent afin de former la solution partielle. Cette solution partielle permet ainsi de déterminer un couple de chemins passant tous deux par l'ensemble des points obligatoires. Elle n'est pas nécessairement optimale puisque les points obligatoires sont parcourus dans le même ordre. En pratique cependant, elle correspond souvent à la solution au problème global. La distance à la solution partielle est ensuite évaluée en considérant les deux chemins obtenus.

5.2.4.1 Résultats et discussion

Les deux modélisations décrites ci-avant, avec la méthode de guidage de la section 5.2.4, ont été expérimentées sur des instances de l'expérimentation mise en œuvre pour le problème de gestion de communications radio. Ces instances proposent différents nombres de points de passage obligatoires, de 0 à 5. Les contraintes de capacité ne sont pas ici prises en compte. Dans cette expérimentation, nous nous sommes limités à un petit nombre d'instances, assez révélatrices des qualités et inconvénients de nos travaux. Nous n'avons ainsi évalué nos approches que pour un couple de points (v_s, v_g) sélectionné au hasard (le couple $(18, 12)$ a été retenu), sur le scénario 1. Les résultats sont présentés dans le tableau 5.2, dans lequel la modélisation 1 réfère au système de flots simple et la modélisation 2 au système de flots double.

En analysant dans un premier temps la modélisation 1, on remarque l'écart de performances entre les deux fonctions d'optimisation, même pour des instances de problèmes « simples ». Cet écart est dû à la propagation des variables de temps dans la première approche, qui influe de manière très conséquente sur le temps de calcul (notamment le temps de preuve). La seconde

$\|V_m\|$	Modélisation 1				Modélisation 2			
	pire temps		*temps total*		*pire temps*		*temps total*	
	T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)
0	0 (195)	310144	0 (345)	201			62 (345)	1715
1	62572 (215)	281777	94 (387)	764	484 (215)	1420	188 (387)	2590
2	24934 (233)	344808	8424 (423)	10140	31 (233)	546	62 (423)	2292
3	–	–	–	–	*	5891	*	2103
4	–	–	81221 (499)	–	578 (237)	843	406 (465)	2371
5	–	–	–	–	936 (286)	1404	545 (561)	3462

TABLE 5.2 – Résultats des deux modélisations, pour chaque fonction d'optimisation, sur le scénario 1 de l'expérimentation de gestion de communications radio, pour le couple $(v_s, v_g) = (18, 12)$. T_1 est le temps nécessaire au solveur global pour trouver la solution optimale, T_2 est le temps nécessaire pour prouver l'optimalité de la solution (fin de la recherche). Un délai d'exécution maximum pour la recherche a été fixé à 500 secondes. Les cases mentionnant – indiquent qu'aucune solution n'a été trouvée (ou prouvée) dans le délai imparti. Les caractères * désignent l'absence de solution : dans ce cas, les temps T_2 indiquent le temps de preuve d'absence de solution.

approche apparaît plus efficace : dès lors qu'une solution est trouvée, elle est capable d'apporter la preuve d'optimalité très rapidement. En revanche, en l'absence de solution, l'exploration s'avère fastidieuse (on remarque que pour $|V_m| = 3$, le problème n'a pas de solution mais la résolution est incapable de le prouver dans un délai de 500 secondes). De manière globale, la modélisation s'avère décevante. De plus, nous l'avons observé sur d'autres instances, la modélisation proposée est en réalité insuffisante pour assurer la cohérence d'une solution. En effet, il est possible d'observer des solutions telles que celle illustrée par la figure 5.4. Dans cet exemple, on observe que l'un des chemins franchit un point de passage obligatoire, puis fait une boucle pour traverser à nouveau ce dernier ! La boucle assure en effet que le flot au niveau du point obligatoire est égal à 2. Le second chemin est donc « dispensé » de passer par ce point... ce qui peut sensiblement améliorer la qualité de la solution dans le cas d'une optimisation *temps total*. La présence possible de ces boucles explique ainsi les lenteurs de la résolution en l'absence de solution préalable. Ce modèle étant complexe à mettre en œuvre (chaque fonction d'optimisation nécessitant une adaptation), et au vu des performances de l'autre modélisation — que nous décrivons ci-dessous — nous l'avons abandonné plutôt que de chercher une solution afin d'assurer la cohérence des chemins (ceci est réalisable à l'aide de contraintes impliquant les identifiants des nœuds obligatoires et du nœud de départ). Point positif tout de même : en l'absence de points de passage (la cohérence est alors assurée), cette modélisation s'avère extrêmement efficace (et bien meilleure que la modélisation 2 dans ce cas précis).

Concernant la modélisation 2, les résultats montrent une bien meilleure performance. Les temps de calcul sont plus importants que la modélisation 1 dans le cas de l'instance où $V_m = \emptyset$. En revanche, pour les autres instances, ceux-ci restent en-deça de quatre secondes, ce qui est conforme à nos attentes. Ainsi, bien que l'ensemble des variables de décision soit dupliqué, les contraintes mises en place sont bien plus simples à vérifier par le solveur, quelle que soit la fonction d'optimisation choisie. Même pour l'instance $|V_m| = 3$, ce dernier est capable de détecter l'absence de solution dans un délai très contenu (alors que 500 secondes n'étaient pas suffisantes auparavant). Ces conclusions sont donc à la décharge de notre volonté initiale de mettre en œuvre un modèle unique de flots, pour lequel la cohérence des chemins est complexe à garantir.

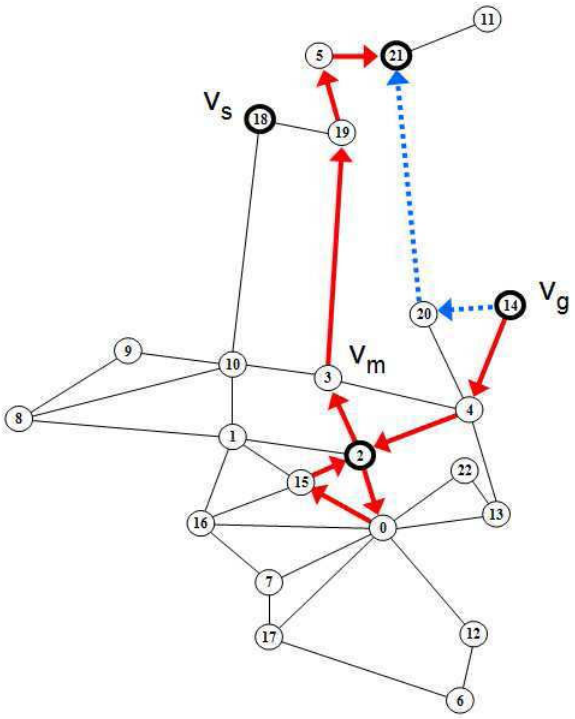


FIGURE 5.4 – Illustration d'une solution au problème de planification robuste qui est cohérente vis-à-vis de la modélisation 1, mais non valide.

Pour conclure, la modélisation 2 est de loin la plus simple à mettre en place, et la plus performante dans le cas général. Elle permet tout à fait d'envisager l'application de contraintes de capacité telles que celles étudiées plus tôt dans le document. On peut alors imaginer résoudre des problèmes consistant à optimiser le pire temps d'arrivée d'un véhicule en considérant les deux chemins disjoints, sous une contrainte limitante de capacité qui soit identique pour les deux chemins. Cela permet de déterminer le pire temps permettant au véhicule d'assurer sa mission, quelle que soit l'alternative choisie, avec les ressources dont il dispose.

5.3 Vers une méthode de sonde dynamique ?

Nous avons vu dans les applications de notre approche aux problèmes spécifiques du chapitre 4, qu'un mauvais guidage peut induire des temps de calcul très importants, notamment lorsque les modèles de contraintes engendrent des temps de propagation élevés. Les améliorations apportées, telles que la mise en œuvre de propagateurs spécifiques et de solveurs partiels intégrant les contraintes de capacité, rendent désormais pertinentes les approches de résolution employées. Cependant, dans le cas « générique » où l'approche de résolution partielle n'intègre pas l'ensemble des contraintes de mission, nous avons envisagé la mise en œuvre d'une méthode de sonde dynamique. Les solutions apportées sur cette problématique consistent à reconfigurer dynamiquement l'arbre de recherche du solveur global si l'exploration en cours n'aboutit à aucune solution.

Les travaux présentés ci-après n'ont pas abouti, pour deux raisons plus amplement décrites en section 5.3.4 : la pertinence applicative, et les limites techniques du compilateur. Cependant, l'implémentation algorithmique proposée ci-dessous est fonctionnelle et atteint le but fixé.

5.3.1 Objectifs

La méthode présentée dans le reste de cette section est dédiée aux problèmes pour lesquels la méthode de résolution présentée aux chapitres 3 et 4 n'est pas efficace (problèmes de taille supérieure à 50 nœuds, contraintes de mission non prises en charge dans la résolution partielle). En effet, celle-ci se base sur une unique solution partielle pour trier les variables de flots du modèle de contraintes. Si cette solution partielle ne mène à aucune solution globale (instanciations non réalisables), le solveur de contraintes peut être amené à réaliser un grand nombre de retours sur trace. L'idée de la méthode présentée ci-après part du constat qu'il peut être judicieux d'utiliser une nouvelle solution partielle, afin de trier à nouveau les variables de flot et ainsi de reconfigurer, en cours d'exécution, l'arbre de recherche. Ce mécanisme a pour objectif de réorienter la recherche vers une autre zone prometteuse de l'espace de recherche.

5.3.2 Description et implémentation

Pour mettre en œuvre cette approche de résolution, deux implémentations sont possibles :
- implémentation séquentielle : générer simultanément plusieurs solutions partielles candidates au tri des variables de flot, en amont de la résolution globale ;
- implémentation parallèle : générer une première solution partielle, afin de configurer la recherche globale. Le solveur partiel (métaheuristique) continue alors son exécution et met à disposition les nouvelles solutions trouvées pour le solveur de contraintes, en parallèle de la recherche.

La première implémentation est très simple à mettre en œuvre, puisqu'il est trivial de mémoriser non pas une mais n meilleures solutions partielles. Cela ne nécessite qu'une modification minime

d'ACO pour être mis en œuvre. La seconde implémentation est davantage complexe puisqu'elle implique à ACO de mettre à disposition ses solutions au solveur global. Or il n'est pas possible d'invoquer un code Prolog en cours d'exécution. Pour contourner ce problème, nous avons donc mis en œuvre un processus de communication appelé LINDA [Carr 89], implémenté dans SICStus Prolog.

Synchronisation Le processus LINDA, dédié à la programmation par contraintes, permet de mettre en œuvre un serveur de communication, accessible par plusieurs clients. Le serveur joue le rôle de *tableau* (*blackboard* en anglais), permettant aux clients de déclarer des faits logiques. Ces faits sont alors partagés avec l'ensemble des autres clients. De cette manière, nous pouvons faire communiquer l'algorithme ACO et le solveur de contraintes. Chaque fois que le solveur partiel dispose d'une solution, il la partage avec le solveur de contraintes sous forme d'un fait. Le solveur de contraintes peut alors récupérer cette solution lorsqu'il le souhaite (processus asynchrone).

Modifications apportées à ACO Une fois que l'algorithme ACO a mis une solution à disposition, la recherche doit interdire cette dernière afin de forcer la diversification. Pour cela, on utilise une *liste tabou* et un mécanisme de *forward checking* (similaire à celui mis en œuvre pour le problème de robotique en section 4.3.2.2) afin de s'assurer que la solution construite par une fourmi sera différente. En l'absence de vérification, si l'algorithme a fortement convergé, plusieurs fourmis pourraient être amenées à ne proposer que les solutions déjà connues.

Évolutions du solveur de contraintes Après avoir établi le modèle de contraintes, le solveur global récupère une solution sur le serveur LINDA (ou patiente le temps qu'une solution soit partagée). Il trie alors les variables et démarre la recherche. On établit alors une condition d'arrêt de la recherche : si aucune solution n'a été trouvée après un temps T_{max} ou après un nombre B_{max} de retours sur traces (à définir), l'exploration est stoppée. Le solveur se connecte alors sur le serveur LINDA pour disposer d'une nouvelle solution. Les variables sont triées une seconde fois et la recherche est relancée. Ce mécanisme peut être réitéré plusieurs fois, jusqu'à achever la recherche ou en se limitant aux solutions partielles de bonne qualité. Tout en offrant la possibilité de reconfigurer l'espace de recherche, on s'assure que la complétude de l'approche soit garantie.

5.3.3 Ne pas réévaluer l'espace déjà visité

Sous Prolog, il n'est pas à notre connaissance possible d'interrompre une fonction d'étiquetage et de modifier l'ordre des variables sans que cette fonction ne réévalue l'ensemble des valeurs de ces variables. Autrement dit, lorsqu'on interrompt une recherche, qu'on réordonne les variables et qu'on relance la recherche, l'ensemble des solutions est à nouveau évalué. Cela assure la complétude de l'approche, mais aboutit à réitérer les instanciations faites lors la première recherche. Pour éviter ce désagrément, nous avons recherché une solution algorithmique dans la littérature.

Le problème posé ici revient à séparer un problème initial en deux sous-problèmes, selon deux espaces distincts (plus exactement deux ensembles de domaines de valeurs) : celui déjà traité, et celui restant à traiter. Ceci peut être réalisé à l'aide d'un algorithme proposé par *Freuder & Hubbe* [Freu 95], permettant l'extraction de sous-problèmes de satisfaction de contraintes. Cet algorithme a initialement été conçu pour simplifier des problèmes dont on connaît des sous-problèmes insolubles. L'exemple explicatif donné par les auteurs est une instance simple du problème de coloration de graphe, illustré par la figure 5.5. Etant donné trois sommets

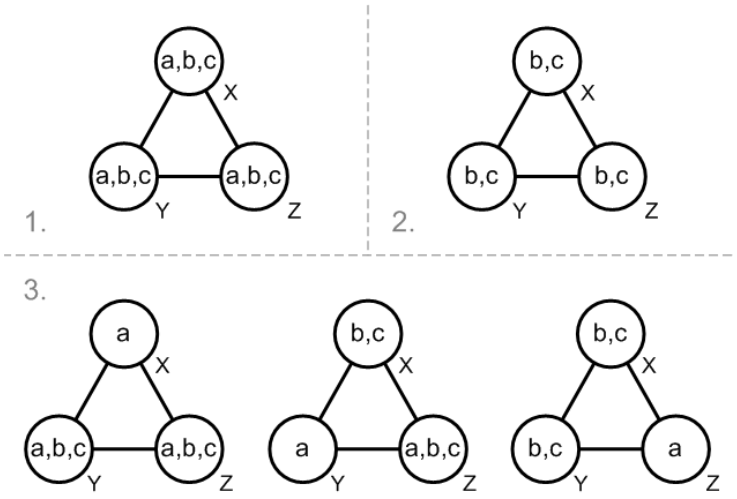


FIGURE 5.5 – Illustration de l'extraction de sous-problèmes de satisfaction de contraintes. Exemple tiré de [Freu 95].

X, Y et Z, pouvant chacun prendre trois couleurs possibles notées a, b ou c (figure 5.5.1), le problème est de trouver une instanciation telle que chaque sommet ait une couleur différente des sommets connexes. Bien que trivial dans ce cas, puisqu'il n'existe que 27 instanciations possibles, ce problème est NP-complet et rapidement difficile à résoudre dès lors que la dimension du problème augmente. Aussi, les auteurs s'appuient sur un constat simple : la même instance de problème, ne proposant que deux couleurs possibles (par exemple b et c, figure 5.5.2) ne possède aucune instanciation (sur les 8 combinaisons possibles) qui soit réalisable. On peut alors utiliser cette information pour extraire du problème initial le sous-problème complémentaire au sous-problème insoluble (qui, en conséquence, ne possèdera plus que 19 instanciations possibles soit une réduction de 35% de leur nombre initial). Dans la majorité des cas, comme ici, le sous-problème restant se compose de plusieurs instances distinctes (figure 5.5.3).

Le pseudo-code de la méthode de décomposition est donné par l'algorithme 8. La figure 5.5 illustre les différentes étapes de l'exécution sur l'exemple de la figure 5.5.

Algorithme 8 Algorithme Freuder-Hubbe

Requiert: : Problème P, Sous-Problème P_s

1: Déclarer une liste vide de sous-problèmes LP complémentaires de P_s
2: **tant que** le problème P ne correspond pas à P_s **faire**
3: Sélectionner une variable V dans P dont le domaine n'est pas équivalent à celui dans P_s
4: Diviser P en deux sous-problèmes : 1/ un problème P_1 dont le domaine de V correspond à celui dans P_s, et un problème P_2 dont le domaine de V est complémentaire à celui dans P_s
5: Déclarer P_1 en tant que problème courant ($P \leftarrow P_1$)
6: Ajouter P_2 à la liste LP
7: **fin tant que**
8: **retourner** LP

Dans notre cas, nous avons souhaité employer cet algorithme afin de séparer le sous-problème déjà traité du sous-problème restant, lors de l'arrêt de l'algorithme (condition d'absence d'amé-

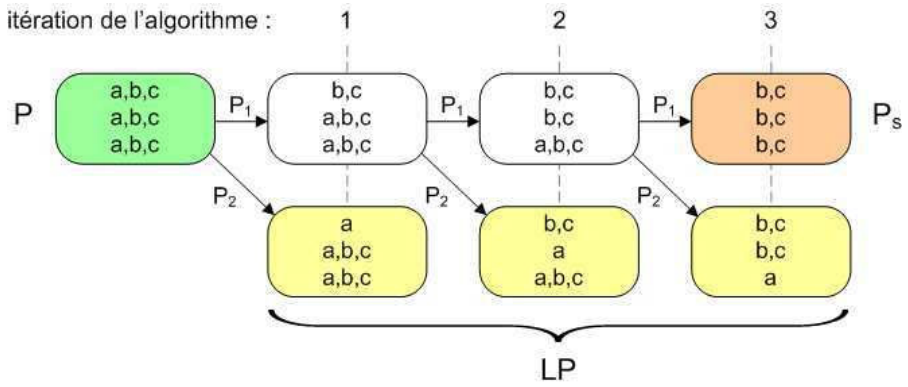


FIGURE 5.6 – Illustration de l'exécution de l'algorithme 8 pour le problème de la figure 5.5.

lioration atteinte). Or, pour connaître le sous-espace des variables de flot déjà exploré, il serait coûteux d'utiliser un mécanisme permettant de conserver la trace de cet espace. Cela consisterait en effet à mettre à jour à chaque instanciation la borne de la variable instanciée. Or, en disposant uniquement de la connaissance du problème initial, et de la dernière instanciation réalisée, l'algorithme de Freuder-Hubbe nous permet de dissocier les deux sous-problèmes. Il suffit en effet de déclarer dans l'algorithme 8 la dernière instanciation en tant que sous-problème P_s(dit « insoluble » dans l'exemple des auteurs). La liste de sous-problèmes LP renvoyée contient alors les deux sous-problèmes, qui sont nécessairement distincts. La figure 5.7 montre sur un exemple simple qu'un problème initial (quatre variables dans $\{0, 1\}$ à instancier) peut être décomposé en un ensemble de sous-problèmes (ou sous-espaces). Il est alors aisé de déterminer quels sont les sous-problèmes qui ont été traités, et ceux qui ne l'ont pas été. On parcourt pour cela, dans l'ordre d'instanciation, les variables d'un sous-problème. Si le domaine de la variable lue est identique à celui de la dernière instanciation, on passe à la variable suivante. Si le domaine est inférieur, alors le problème a été traité. Sinon, il n'a pas été traité.

Cette utilisation détournée de l'algorithme de Freuder-Hubbe permet ainsi de déterminer simplement le sous-problème restant d'une exploration interrompue, sans avoir à se préoccuper de la mémorisation de l'espace visité. Une fois les sous-problèmes restants déterminés, leurs variables sont triées selon une nouvelle solution partielle. Chaque sous-problème doit ensuite être traité séparément. La recherche se termine lorsque tous les sous-problèmes ont été évalués.

5.3.4 Discussion

L'implémentation de la méthode décrite ci-avant a posé plusieurs problèmes. Tout d'abord, la mise en œuvre du mécanisme de séparation en sous-problèmes mentionné ci-avant est en pratique assez complexe à implémenter. En effet, si la recherche est stoppée une première puis une seconde fois, le sous-problème évalué est alors lui aussi redivisé, aboutissant à gérer une liste potentiellement grande de sous-problèmes. De plus, chaque fois qu'une nouvelle solution partielle est employée, il est alors nécessaire de trier les variables de l'ensemble des sous-problèmes mémorisés. Enfin, une fois les variables triées, il faut déterminer (pour être efficace) le sous-problème contenant les instanciations les plus prometteuses vis-à-vis du guidage réalisé. De plus, des problèmes techniques liées à l'utilisation de la librairie LINDA sont survenus (les défaillances ayant été communiquées à SICStus, mais non résolues).

Nous avons réalisé quelques essais sur des instances de grande taille (avec précalcul de trois

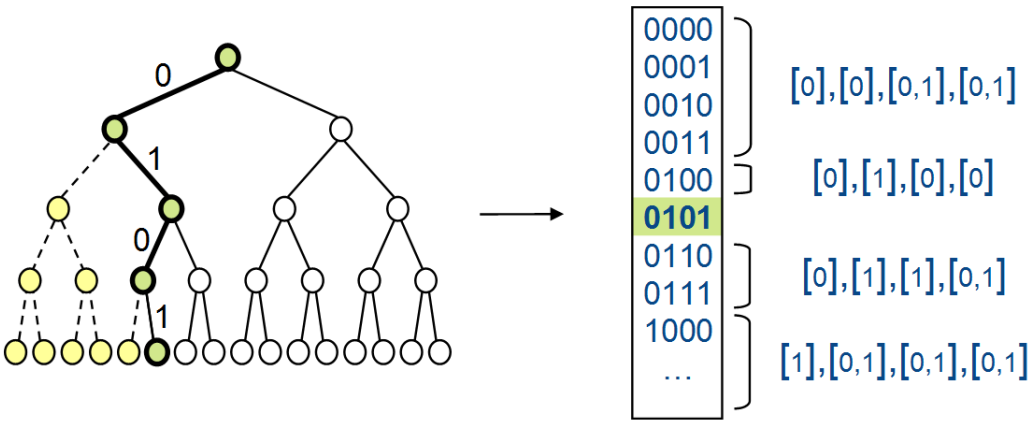


FIGURE 5.7 – Illustration de l'utilisation possible de l'algorithme de Freuder-Hubbe pour la séparation de deux sous-problèmes à partir d'une instanciation unique (en gras). L'espace visité (branches de l'arbre en pointillés) correspond aux deux domaines de valeurs du haut. L'espace restant (branches pleines) correspond aux deux domaines de valeurs du bas. Il est à noter que l'instanciation courante pourra être agrégé à l'un ou l'autre des espaces, selon que la solution a été évaluée ou non.

solutions partielles, pour pallier à la synchronisation) pour valider la faisabilité de notre approche. Cependant, de manière parallèle, une réflexion quant à la cohérence de nos travaux s'est alors imposée : pour de grands problèmes, dont les preuves d'optimalité ne peuvent être obtenues (hormis les problèmes simples), est-il justifié de privilégier une approche de Programmation par Contraintes lorsqu'une méthode d'optimisation approchée serait capable de résoudre le problème rapidement ? Pour des problèmes de taille plus raisonnables, n'est-il pas plus judicieux de mettre en œuvre un algorithme de résolution partielle intégrant l'ensemble des contraintes ?

Pour ces raisons, nous avons stoppé les recherches et nous sommes concentrés à améliorer notre approche avec sonde statique. Pour l'anecdote, c'est suite à ces travaux que nous avons eu l'idée d'implémenter l'approche avec relaxation lagrangienne présentée en section 3.4.

Conclusion

Nous avons présenté dans ce document une approche de résolution originale pour la résolution de problèmes de planification sous contraintes d'itinéraires de véhicules. L'hybridation d'un algorithme de colonies de fourmis avec un solveur de contraintes, par le biais d'une méthode de sonde, a montré expérimentalement un gain en termes de stabilité des temps de calcul. Si elle s'avère parfois légèrement moins performante qu'une approche basique pour les problèmes simples, elle permet de cantonner la résolution dans des délais respectables pour des problèmes très contraints, tout en garantissant l'optimalité de l'approche.

Pour le cas particulier des problèmes avec contraintes de capacité additives, nous avons mis au point une méthode implémentant une relaxation lagrangienne. Cette méthode approchée permet de résoudre, dans un délai très court, le problème consistant simultanément à trouver un plus court chemin passant par l'ensemble des contraintes *et* satisfaisant les contraintes de capacité. Elle présente ainsi l'avantage de garantir que la solution partielle ne viole pas les contraintes de capacité imposées, et permet donc d'orienter la recherche globale vers une zone réalisable de l'espace d'état.

Parallèlement à ces travaux, nous avons apporté plusieurs modifications à la modélisation logique du problème. Nous y avons adjoint plusieurs mécanismes permettant de réduire sensiblement les temps de calcul, notamment ceux induits par la propagation des contraintes. Les évolutions notables ont consisté à définir dynamiquement les bornes de variables du modèle (travaux non détaillés), à réaliser un propagateur spécifique pour la gestion de la cohérence du chemin, et à mettre en place des coupes — en utilisant les variables de temps de la solution partielle — ne pénalisant pas la complétude de l'approche. Dans le cadre d'une étude sur l'applicabilité de notre approche pour un problème de planification robuste, nous avons observé que l'utilisation de deux modèles de chemins, malgré la duplication de l'ensemble des variables, est beaucoup plus simple et efficace qu'un unique modèle aux règles de cohérence complexes.

Nos travaux ont été appliqués sur des problèmes de planification spécifiques : un problème de gestion d'énergie du véhicule (capacité additive), et un problème de gestion des communications radio (capacité non additive). Les expérimentations réalisées ont montré que nos approches étaient à même de résoudre ces problèmes dans les délais imposés, bien qu'il s'agisse dans les deux cas de contraintes non linéaires (pour lesquelles les temps de propagation peuvent se révéler très importants).

Lors des expérimentations, nous avons tout de même buté sur un obstacle majeur. En effet, les modèles de chemin du solveur de contraintes et de l'algorithme de résolution partielle ne sont pas exactement similaires : ce dernier autorise en effet le véhicule à repasser par un même nœud du graphe. Or le modèle de flot défini en programmation par contraintes exclut cette possibilité, pouvant aboutir, dans de rares cas, à un mauvais guidage. Si les conséquences sont peu perceptibles pour des problèmes simples, elles peuvent en revanche s'avérer catastrophiques pour des problèmes complexes (cf. problème de gestion d'énergie). Malgré les travaux exploratoires

menés sur la définition d'un nouveau modèle de chemin, en cohérence avec la résolution partielle, nous ne sommes pas parvenus à définir une implémentation satisfaisant les exigences. Il reste par conséquent quelques rares cas pour lesquels notre stratégie de recherche n'a pas les performances escomptées.

Perspectives futures

En termes de perspectives, l'intégration de cette approche de résolution pour le cas des problèmes multi-véhicules devra être menée et analysée. L'approche partielle actuelle pourra être employée pour guider l'instanciation des flots des chemins de chaque véhicule. Cela équivaut à résoudre, à partir d'un problème de planification de n véhicules, n problèmes mono-véhicules dans lesquels les contraintes de coordinations sont relaxées. Pour éventuellement traiter ces dernières, l'intégration de pénalités temporelles (et donc de coût) dans l'algorithme de colonies de fourmis pourrait en outre permettre la gestion de fenêtres de temps imposées à des nœuds ou arcs.

Au niveau du solveur de contraintes, l'exploration d'autres modèles logiques que celui des flots pour la définition des chemins ,est nécessaire pour assurer la mise à l'échelle de l'approche et la rendre cohérente avec l'emploi des métaheuristiques. Parmi les pistes à explorer, l'emploi de contraintes globales telles que `global-cardinality` ou `multi-cost-regular` pourront être envisagées. L'enjeu sera d'être en mesure de limiter la perte de généricité du modèle (expression de contraintes de fenêtres de temps aux nœuds, contraintes de capacité liées à la vitesse le long d'un arc...), et de s'assurer que la méthode de guidage puisse être appliquée.

Il reste à explorer la gestion de contraintes d'arcs obligatoires, et sa prise en charge dans la résolution partielle. Leur prise en compte au sein de l'algorithme de colonies de fourmis pourrait par exemple être obtenue en déclarant les nœuds relatifs à ces arcs comme points obligatoires, et en forçant la solution à emprunter ces arcs lors de la construction des solutions par les fourmis.

Pour le problème présenté dans l'application robotique, il ne semble pas exister d'équivalent dans la littérature parmi les nombreuses variantes de TSP. Soumettre ce problème à la communauté pourrait donc être envisagé. L'implémentation d'ACO présentée dans ce document pourrait être améliorée par la définition d'un mécanisme spécifique d'insertion après l'emploi de la recherche locale *2-opt* afin potentiellement d'améliorer la qualité, en termes de nombre de points visités, de la solution construite.

Pour le problème de planification robuste (ou de planification multi-véhicules avec des véhicules aux propriétés identiques), il serait intéressant d'analyser les possibilités de suppression des cas de symétries dans la solution.

Enfin, pour des problèmes de grande taille (et sortant du cadre des exigences de nos travaux), la méthode de sonde dynamique pourrait être reprise et consolidée par l'analyse de conditions d'arrêt plus évoluées. Alternativement, l'étude d'une sonde statique combinant la solution d'un problème relaxé et la solution d'une précédente recherche, pourrait donner de bons résultats pour les environnements très changeants.

Annexes

Évaluation du solveur de contraintes

L'expérimentation de la section 3.3.1 regroupe trois scénarios. Pour chaque scénario, six séries de dix instances ont été mises en place. Une instance correspond à un couple (v_s, v_g) donné : le tableau 5.3 récapitule les valeurs de v_s et v_g pour chaque instance. Un jeu d'instances correspond à l'ensemble des instances appliquées à un ensemble de points obligatoires donnés, et récapitulés dans le tableau 5.4.

Scénario 1 : 23 nœuds, 76 arcs (voir figure 5.8)

- Liste des nœuds : (0, 2008, 2804), (1, 1444, 2440), (2, 1936, 2516), (3, 1804, 2212), (4, 2336, 2348), (5, 1760, 828), (6, 2296, 3476), (7, 1576, 3008), (8, 628, 2364), (9, 1044, 2100), (10, 1444, 2160), (11, 2396, 688), (12, 2356, 3212), (13, 2432, 2756), (14, 2484, 1912), (15, 1708, 2624), (16, 1332, 2780), (17, 1576, 3236), (18, 1536, 1076), (19, 1844, 1140), (20, 2160, 1940), (21, 2064, 840), (22, 2320, 2560) ;
- Liste des arcs : (0, 2), (0, 7), (0, 12), (0, 13), (0, 15), (0, 16), (0, 17), (0, 22), (1, 2), (1, 8), (1, 10), (1, 15), (1, 16), (2, 0), (2, 1), (2, 3), (2, 4), (2, 15), (3, 2), (3, 4), (3, 10), (3, 19), (4, 2), (4, 3), (4, 13), (4, 14), (4, 20), (5, 19), (5, 21), (6, 12), (6, 17), (7, 0), (7, 16), (7, 17), (8, 1), (8, 9), (8, 10), (9, 8), (9, 10), (10, 1), (10, 3), (10, 8), (10, 9), (10, 18), (11, 21), (12, 0), (12, 6), (13, 0), (13, 4), (13, 22), (14, 4), (14, 20), (15, 0), (15, 1), (15, 2), (15, 16), (16, 0), (16, 1), (16, 7), (16, 15), (17, 0), (17, 6), (17, 7), (18, 10), (18, 19), (19, 3), (19, 5), (19, 18), (20, 4), (20, 14), (20, 21), (21, 5), (21, 11), (21, 20), (22, 0), (22, 13).

Scénario 2 : 22 nœuds, 68 arcs (voir figure 5.10)

- Liste des nœuds : (0, 4680, 700), (1, 3660, 60), (2, 3110, 780), (3, 3100, 1310), (4, 1810, 1590), (5, 1970, 1130), (6, 1540, 1170), (7, 1420, 1520), (8, 930, 780), (9, 800, 1430), (10, 3540, 2300), (11, 3950, 4980), (12, 2430, 1670), (13, 2100, 1940), (14, 2590, 2130), (15, 4630, 1830), (16, 2360, 1260), (17, 3050, 2500), (18, 3380, 4980), (19, 3170, 3550), (20, 2310, 850), (21, 4230, 3480) ;
- Liste des arcs : (0, 1), (1, 0), (1, 2), (2, 1), (2, 3), (2, 15), (2, 20), (3, 2), (3, 10), (3, 15), (3, 16), (4, 5), (4, 7), (4, 12), (4, 13), (5, 4), (5, 6), (5, 16), (5, 20), (6, 5), (6, 7), (6, 8), (6, 9), (7, 4), (7, 6), (7, 9), (8, 6), (8, 9), (9, 6), (9, 7), (9, 8), (10, 3), (10, 15), (10, 17), (10, 21), (11, 18), (11, 21), (12, 4), (12, 13), (12, 14), (12, 16), (13, 4), (13, 12), (13, 14), (14, 12), (14, 13), (14, 17), (15, 2), (15, 3), (15, 10), (15, 21), (16, 3), (16, 5), (16, 12), (16, 20), (17, 10), (17, 14), (17, 19), (18, 11), (18, 19), (19, 17), (19, 18), (20, 2), (20, 5), (20, 16), (21, 10), (21, 11), (21, 15).

Scénario 3 : 22 nœuds, 74 arcs (voir figure 5.10)

– Liste des nœuds : (0, 572, 7436), (1, 1892, 7304), (2, 2706, 4268), (3, 1936, 8646), (4, 2904, 6534), (5, 3828, 8382), (6, 3410, 9614), (7, 4906, 9680), (8, 4906, 7348), (9, 5126, 6358), (10, 4224, 4598), (11, 4620, 2244), (12, 6116, 4356), (13, 7326, 6138), (14, 7150, 8404), (15, 8426, 10516), (16, 6424, 2266), (17, 7040, 3432), (18, 8514, 4862), (19, 9570, 6622), (20, 10186, 8624), (21, 11506, 10142) ;
– Liste des arcs : (0, 1), (1, 0), (1, 2), (1, 3), (1, 4), (1, 8), (2, 1), (2, 4), (2, 11), (3, 1), (3, 5), (3, 6), (4, 1), (4, 2), (4, 8), (4, 9), (4, 10), (5, 3), (5, 7), (6, 3), (6, 7), (7, 5), (7, 6), (7, 8), (7, 14), (7, 15), (8, 1), (8, 4), (8, 7), (8, 13), (9, 4), (9, 12), (9, 13), (10, 4), (10, 11), (10, 12), (11, 2), (11, 10), (11, 12), (11, 16), (12, 9), (12, 10), (12, 11), (12, 13), (12, 17), (13, 8), (13, 9), (13, 12), (13, 14), (13, 18), (13, 19), (14, 7), (14, 13), (14, 15), (14, 20), (15, 7), (15, 14), (15, 21), (16, 11), (16, 17), (17, 12), (17, 16), (17, 18), (18, 13), (18, 17), (18, 19), (19, 13), (19, 18), (19, 20), (20, 14), (20, 19), (20, 21), (21, 15), (21, 20).

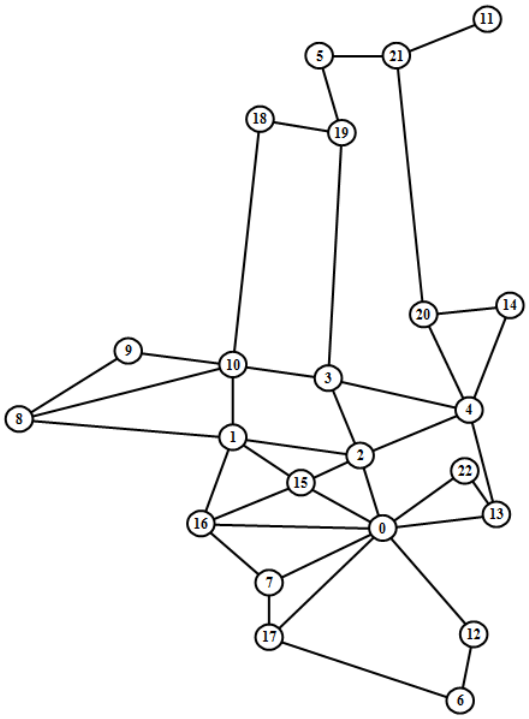


FIGURE 5.8 – Graphe du scénario 1.

Un quatrième scénario, mentionné en section 3.3.3, a également été mis au point. Ce scénario, établi sur la base du graphe du scénario 3, compte 33 nœuds et 130 arcs (voir figure 5.11) :
– Liste des nœuds : (0, 383, 6365), (1, 1700, 6360), (2, 2595, 9410), (3, 1795, 5149), (4, 2635, 7003), (5, 3480, 5189), (6, 2985, 3938), (7, 4649, 3839), (8, 4794, 6047), (9, 4883, 7328), (10, 4066, 8943), (11, 4462, 1392), (12, 5895, 9138), (13, 6903, 7500), (14, 6875, 5218), (15, 7940, 3021), (16, 6228, 11422), (17, 6826, 10085), (18, 7909, 8626), (19, 9122, 6813), (20, 9422, 4965), (21, 11118, 3384), (22, 11314, 5837), (23, 1330, 9511), (24, 2794, 11056), (25, 0824, 11162), (26, 9370, 9245), (27, 7959, 10816), (28, 8199, 1285), (29, 9530, 3841), (30, 5591, 2724), (31, 2339, 2614), (32, 6199, 991) ;
– Liste des arcs : (0, 1), (1, 0), (1, 3), (3, 1), (1, 4), (4, 1), (4, 2), (2, 4), (2, 1), (1, 2), (3, 5), (5, 3), (6, 3), (3, 6), (5, 7), (7, 5), (7, 6), (6, 7), (8, 7), (7, 8), (8, 4), (4, 8), (1, 8), (8, 1), (4, 9), (9, 4), (10, 4), (4, 10), (10, 11), (11, 10), (2, 11), (11, 2), (11, 12), (12, 11), (10, 12),

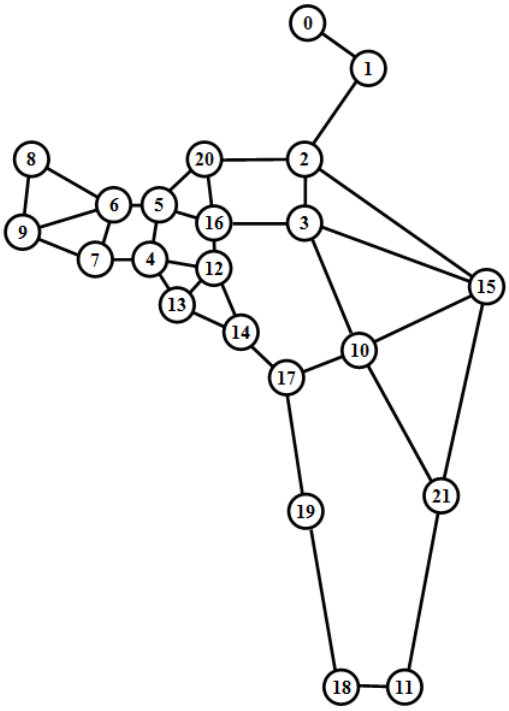


FIGURE 5.9 – Graphe du scénario 2.

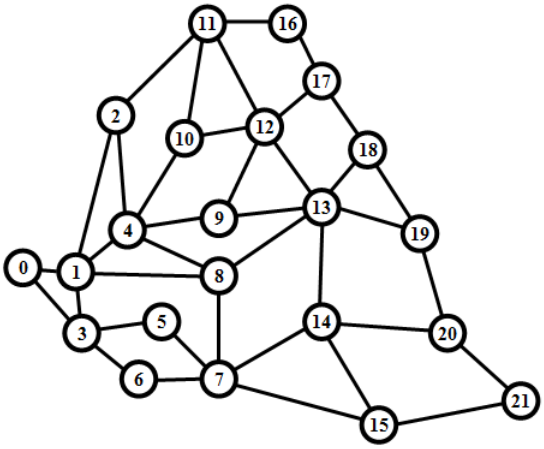


FIGURE 5.10 – Graphe du scénario 3.

(12, 10), (9, 12), (12, 9), (12, 13), (13, 12), (9, 13), (13, 9), (8, 13), (13, 8), (7, 14), (14, 7), (13, 14), (14, 13), (14, 15), (15, 14), (7, 15), (15, 7), (11, 16), (16, 11), (16, 17), (17, 16), (12, 17), (17, 12), (17, 18), (18, 17), (13, 18), (18, 13), (13, 19), (19, 13), (18, 19), (19, 18), (14, 20), (20, 14), (19, 20), (20, 19), (20, 21), (21, 20), (15, 21), (21, 15), (22, 21), (21, 22), (20, 22), (22, 20), (19, 22), (22, 19), (0, 3), (3, 0), (23, 2), (2, 23), (0, 23), (23, 0), (1, 23), (23, 1), (23, 24), (24, 23), (24, 11), (11, 24), (25, 24), (24, 25), (23, 25), (25, 23), (26, 19), (19, 26), (17, 26), (26, 17), (16, 27), (27, 16), (27, 26), (26, 27), (26, 22), (22, 26), (28, 21), (21, 28), (29, 21), (21, 29), (20, 29), (29, 20), (30, 7), (7, 30), (30, 28), (28, 30), (30, 6),

Instance	Scénario 1		Scénario 2		Scénario 3	
	v_s	v_g	v_s	v_g	v_s	v_g
1	11	17	1	18	21	2
2	18	12	20	11	15	16
3	8	14	21	9	7	11
4	17	14	19	8	0	19
5	6	9	1	11	3	18
6	0	18	13	1	2	15
7	11	7	20	21	11	21
8	5	6	9	11	20	2
9	9	12	9	1	17	3
10	8	11	4	21	3	21

TABLE 5.3 – Définition des couples (v_s, v_g) pour les trois scénarios.

Instance	Scénario 1	Scénario 2	Scénario 3
0	{ }	{ }	{ }
1	{ 3 }	{ 10 }	{ 9 }
2	{ 2,16 }	{ 12,10 }	{ 5,12 }
3	{ 21,16,10 }	{ 10,6,2 }	{ 10,4,14 }
4	{ 16,2,3,10 }	{ 12,15,17,3 }	{ 14,4,12,5 }
5	{ 16,3,10,4,15 }	{ 3,6,15,12,17 }	{ 13,5,10,14,4 }

TABLE 5.4 – Définition des points obligatoires V_m pour les trois scénarios.

(6, 30), (6, 31), (31, 6), (31, 30), (30, 31), (31, 3), (3, 31), (28, 32), (32, 28), (32, 30), (30, 32), (31, 32), (32, 31).

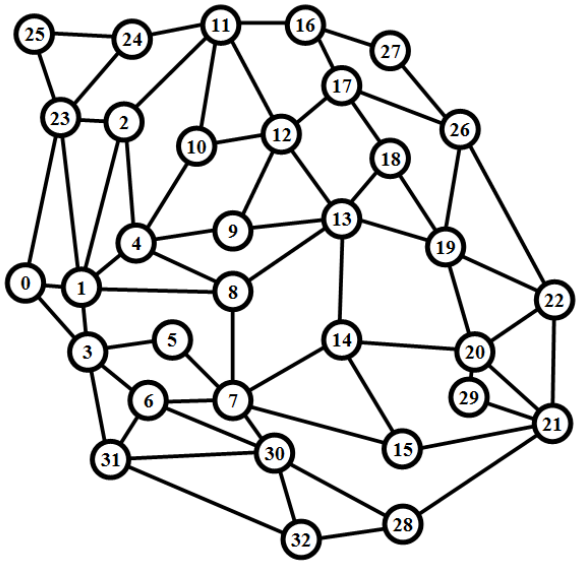


FIGURE 5.11 – Graphe du scénario 4.

Évaluation de l'approche LARAC-ACO

L'approche LARAC-ACO a été évaluée sur une grande instance. Cette dernière est issue d'un scénario Sagem d'inter-opérabilité entre une bataillon français et américain, mis au point pour une expérimentation entre les armées des deux nations. Sa taille est donc très importante (pour un graphe de planification militaire du moins) : 479 noeuds et 2852 arcs. Nous n'avons pas ici jugé utile de détailler l'ensemble de ces noeuds et arcs (car les données sont très volumineuses !), mais montrons un aperçu du graphe dans son ensemble avec la figure 5.12.

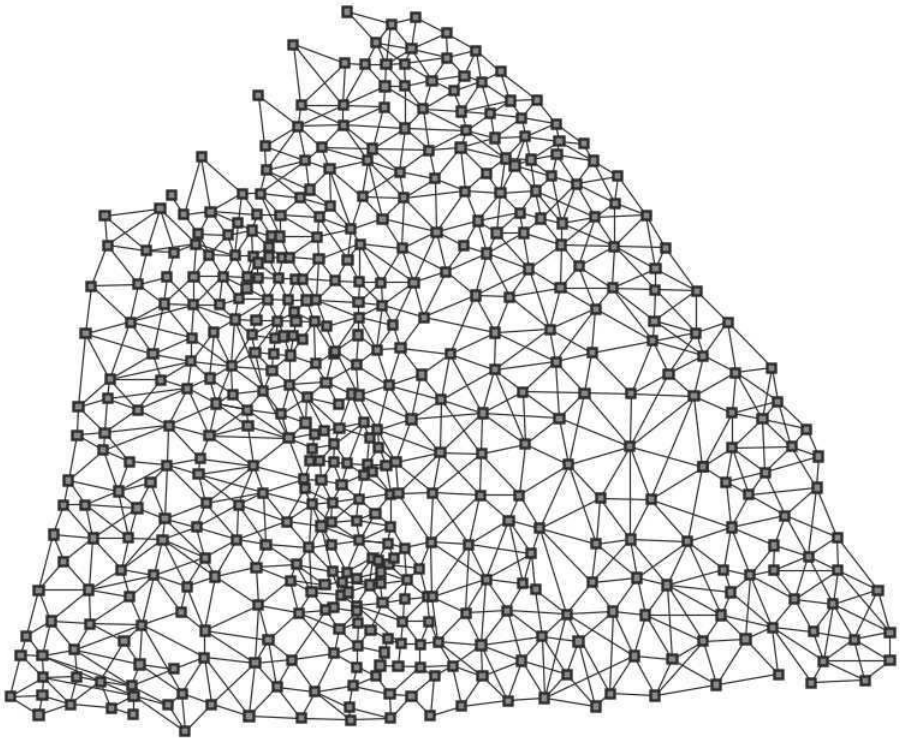


FIGURE 5.12 – Graphe du scénario utilisé pour l'expérimentation de l'approche LARAC-ACO.

Pour générer les valeurs de capacité liées aux arcs, comme mentionné dans la section 3.4.4, nous avons mis au point un programme permettant de définir cette valeur à partir d'une image appliquée en arrière plan du graphe. L'image que nous avons utilisé dans le cadre de cette expérience est celle de la figure 5.13. Cette image n'est autre qu'une texture de type treillis militaire, à laquelle nous avons appliqué un flou gaussien à l'aide d'un outil de retouche d'image. Après avoir réduit le graphe au fomat de l'image, on calcule pour chaque arc la valeur moyenne des valeurs de luminances des pixels de l'image situés aux coordonnées des noeuds relatifs à l'arc.

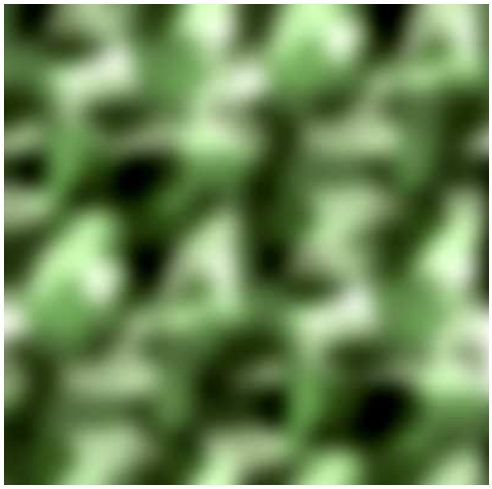


FIGURE 5.13 – Image utilisée pour la génération de valeurs de capacité dans l'expérimentation de l'approche LARAC-ACO.

Détails de l'expérimentation pour le problème de gestion d'énergie

Le détail des instances mises en place pour les trois *benchmarks* est décrit par les tableaux 5.5, 5.6 et 5.7. Ces instances, disponibles en format XML, peuvent être téléchargées librement sur la page Internet dédiée au *benchmark* [Luca].

Benchmark 1 : 23 vertices, 76 edges				
Mission ID	v_s	v_g	V_m	R_0
mission_1_1	7	11	$\emptyset$	475
mission_1_2	12	18	$\emptyset$	400
mission_1_3	13	5	$\{3, 8, 16\}$	700
mission_1_4	6	14	$\{7, 10, 15\}$	450
mission_1_5	11	8	$\{0, 4, 16, 19, 22\}$	800
mission_1_6	21	17	$\{2, 8, 10, 13, 18\}$	950

TABLE 5.5 – Description des instances du *Benchmark 1*.

Benchmark 2 : 22 vertices, 74 edges				
Mission ID	v_s	v_g	V_m	R_0
mission_2_1	1	18	$\emptyset$	750
mission_2_2	11	8	$\emptyset$	820
mission_2_3	9	21	$\{3, 4, 20\}$	600
mission_2_4	15	8	$\{3, 20, 21\}$	705
mission_2_5	11	9	$\{3, 7, 10, 14, 20\}$	1025
mission_2_6	15	1	$\{3, 5, 11, 13, 17\}$	1250

TABLE 5.6 – Description des instances du *Benchmark 2*.

Benchmark 3 : 22 vertices, 68 edges				
Mission ID	v_s	v_g	V_m	R_0
mission_3_1	21	2	$\emptyset$	1975
mission_3_2	6	16	$\emptyset$	2000
mission_3_3	0	21	$\{4, 14, 16\}$	2200
mission_3_4	15	18	$\{2, 5, 12\}$	2725
mission_3_5	20	16	$\{1, 2, 10, 14, 19\}$	3600
mission_3_6	7	19	$\{4, 6, 11, 14, 17\}$	3850

TABLE 5.7 – Description des instances du *Benchmark 3*.

Détails de l'expérimentation pour le problème de gestion des communications radio

Les trois scénarios utilisés pour cette expérimentation correspondent aux trois scénarios mis en place pour l'évaluation du solveur de contraintes (voir section 5.3.4). Nous y avons appliqué un modèle de qualité de réception radio. Nous avons considéré que la réception de la radio tactique était maximale partout sur le terrain ($C_1 = 1$ le long de tous les arcs, pour chaque scénario). Pour la qualité de réception de la radio *Wi-fi*, plusieurs arcs de chaque graphe de scénario ont été volontairement atténués afin d'induire une difficulté dans les problèmes à traiter. Voici, pour chaque scénario, les arcs (exprimés par leur noeud de départ et d'arrivée) dont les valeurs C_2 ont été atténuées :

- Scénario 1 : (0, 2), (0, 13), (0, 22), (1, 2), (2, 0), (2, 1), (2, 3), (2, 4), (2, 15), (3, 2), (3, 4), (4, 2), (4, 3), (4, 13), (4, 14), (4, 20), (13, 0), (13, 4), (13, 22), (14, 4), (15, 2), (20, 4), (22, 0), (22, 13) ;
- Scénario 2 : (3, 10), (3, 16), (5, 16), (10, 3), (10, 17), (12, 16), (13, 14), (14, 13), (14, 17), (16, 3), (16, 5), (16, 12), (17, 10), (17, 14), (17, 19), (19, 17) ;
- Scénario 3 : (1, 2), (1, 4), (1, 8), (2, 1), (2, 4), (2, 11), (4, 1), (4, 2), (4, 8), (4, 9), (4, 10), (8, 1), (8, 4), (8, 13), (9, 4), (9, 12), (9, 13), (10, 4), (10, 11), (10, 12), (11, 2), (11, 10), (11, 12), (12, 9), (12, 10), (12, 11), (12, 13), (13, 8), (13, 9), (13, 12).

Pour ces arcs, la valeur de qualité a été fixée à $C_2 = 0,1$ ($C_2 = 1$ pour les autres arcs du graphe). Les figures 5.14, 5.15 et 5.16 illustrent sur les graphes les arcs ayant été atténués (traits fins).

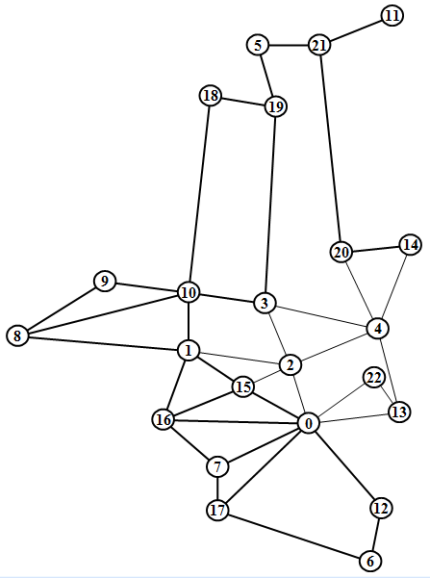


FIGURE 5.14 – Graphe du scénario 1 pour le problème de gestion des communications radio.

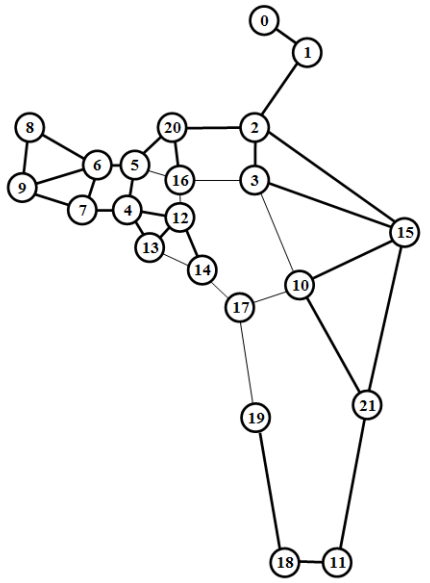


FIGURE 5.15 – Graphe du scénario 2 pour le problème de gestion des communications radio.

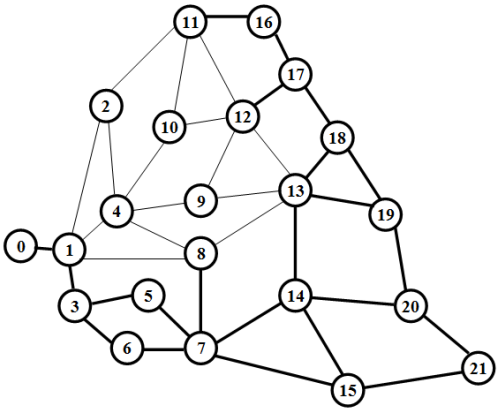


FIGURE 5.16 – Graphe du scénario 3 pour le problème de gestion des communications radio.

Références bibliographiques

[Aart 97] E. Aarts and K. Lenstra. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., 1997.

[Albe 01] D. Alberts, J. Garstka, R. Hayes, and D. Signori. *Understanding Information Age Warfare*. CCRP Publication Series, August 2001.

[Amba 91] B. Ambati, J. Ambati, and M. Mokhtar. "Heuristic Combinatorial Optimization by Simulated Darwinian Evolution : A Polynomial Time Algorithm for the Traveling Salesman Problem". *Biological Cybernetics*, Vol. 65, pp. 31–35, 1991.

[Anej 83] Y. Aneja, V. Aggarwal, and K. Nair. "Shortest Chain Subject to Side Conditions". *Networks*, Vol. 13, pp. 295–302, 1983.

[Appl] D. Applegate, R. Bixby, V. Chvátal, and W. Cook. `http://www.tsp.gatech.edu/concorde/index.html`.

[Banz 90] W. Banzhaf. "The "Molecular" Traveling Salesman". *Biological Cybernetics*, Vol. 64, pp. 7–14, 1990.

[Barn 98] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. "Branch-and-price : Column Generation for Solving Huge Integer Programs". *Operations Research*, Vol. 46, pp. 316–329, 1998.

[Beas 89] J. Beasley and N. Christofides. "An Algorithm for the Resource Constrained Shortest Path Problem". *Networks*, Vol. 19, pp. 379–394, 1989.

[Benc 12] P. Benchimol, W.-J. van Hoeve, J.-C. Régin, L.-M. Rousseau, and M. Rueher. "Improved Filtering for Weighted Circuit Constraints". *Constraints*, Vol. 17, No. 3, pp. 205–233, 2012.

[Berl 84] H. Berliner and G. Goetsch. "A Quantitative Study of Search Methods and the Effect of Constraint Satisfaction". *Technical Report CMU-CS-84-147*, 1984.

[Bess 06] C. Bessière. "Constraint Propagation". *Technical Report LIRMM 06020, CNRS / University of Montpellier*, March 2006.

[Beye 02] H.-G. Beyer and H.-P. Schwefel. "Evolution Strategies : A Comprehensive Introduction". *Natural Computing*, Vol. 1, No. 1, pp. 3–52, 2002.

[Blok 95] D. Blokh and G. Gutin. "An Approximation Algorithm for Combinatorial Optimization Problems with Two Parameters". *Australasian Journal of Combinatorics*, No. 14, pp. 157–164, 1995.

[Bock 58] F. Bock. "An Algorithm for Solving " Traveling-Salesman " and Related Network Optimization Problems". In : *Proceedings of the Fourteenth National Meeting of the Operations Research Society of America*, St. Louis, MO, USA, October 1958.

[Brad 85] R. Brady. "Optimization Strategies Gleaned from Biological Evolution". *Nature*, Vol. 317, pp. 804–806, 1985.

[Buli 10] V. Bulitko, Y. Björnsson, N. Sturtevant, and R. Lawrence. "Real-time Heuristic Search for Pathfinding in Video Games". In : P. González-Calero and M. Gómez-Martín, Eds., *Artificial Intelligence for Computer Games*, pp. 1–30, Springer, 2010.

[Bull 99] B. Bullnheimer, R. Hartl, and C. Strauss. "A New Rank-based Version of the Ant System : a Computational Study". *Central European Journal for Operations Research and Economics*, Vol. 7, No. 1, pp. 25–38, 1999.

[Carl 03] W. Carlyle and R. Wood. "Lagrangian Relaxation and Enumeration for Solving Constrained Shortest-Path Problems". In : *Proceedings of the Thirty-Eighth Annual Operational Research Society of New Zealand (ORSNZ) Conference*, University of Waikato, Hamilton, New Zealand, November 2003.

[Carr 89] N. Carriero and D. Gelernter. "LINDA in Context". *Communications of ACM*, Vol. 32, No. 4, pp. 444–458, April 1989.

[Case 94] Y. Caseau and F. Laburthe. "Improved CLP Scheduling with Task Intervals". In : *Proceedings of the Eleventh International Conference on Logic Programming (ICLP)*, pp. 369–383, Santa Margherita Ligure, Italia, June 1994.

[Cern 85] V. Černý. "Thermodynamical Approach to the Traveling Salesman Problem : an Efficient Simulation Algorithm". *Journal of Optimization Theory and Applications*, Vol. 45, pp. 41–51, 1985.

[Chak 89] P. Chakrabarti, S. Ghose, A. Acharya, and S. de Sarkar. "Heuristic Search in Restricted Memory". *Artificial Intelligence*, Vol. 47, pp. 197–221, 1989.

[Chel 00] R. Chelouah and P. Siarry. "A Continuous Genetic Algorithm Designed for the Global Optimization of Multimodal Functions". *Journal of Heuristics*, Vol. 6, No. 2, 2000.

[Chri 76] N. Christofides. "Worst-Case Analysis of a New Heuristic for the Travelling Salesman Problem". Tech. Rep. Report 388, Graduate School of Industrial Administration, Carnegie Mellon University, 1976.

[Clai] "Claire Language". `http://www.claire-language.com/`.

[Cler 05] M. Clerc. *L'optimisation par essaims particulaires.* Éditions Hermès-Lavoisier, 2005.

[Colm 70] A. Colmerauer. "Les systèmes-Q ou un formalisme pour analyser et synthétiser des phrases sur ordinateur". *Rapport interne*, Vol. 43, 1970.

[Colm 90] A. Colmerauer. "An introduction to Prolog III". Vol. 33, pp. 69–90, 1990.

[Colm 96] A. Colmerauer and P. Roussel. "The Birth of Prolog". In : T. Bergin and R. Gibson, Eds., *History of Programmation Languages II*, pp. 331–367, 1996.

[Cook 71] S. Cook. "The Complexity of Theorem Proving Procedures". In : *Proceedings of the Third Annual ACM Symposium on Theory of Computing (STOC)*, pp. 151–158, Palo Alto, CA, USA, October 1971.

[Cord 00] O. Cordon, I. D. Viana, F. Herrera, and L. Moreno. "A New ACO Model Integrating Evolutionaly Computation Concepts : the Best-Worst Ant System". In : *Proceedings of the Second International Workshop on Ant Algorithms, IEEE Transaction on Evolutionary Computation*, pp. 22–29, Brussels, Belgium, September 2000.

[Corm 01] T. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* MIT Press et McGraw-Hill, 2001.

[Cram 85] N. Cramer. “A representation for the Adaptive Generation of Simple Sequential Programs”. In : J. J. G. (Eds.), Ed., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pp. 183–187, Lawrence Erlbaum Associates, Carnegie Mellon University, Pittsburgh, PA, USA, July 1985.

[Cres 05] P. Crescenzi and V. Kann. “A Compendium of NP Optimization Problems”. 2005. `http://www.csc.kth.se/~viggo/wwwcompendium`.

[Croe 58] G. Croes. “A Method for Solving Traveling Salesman Problems”. *Operations Research*, Vol. 6, pp. 791–812, 1958.

[Dant 51] G. Dantzig. *Activity Analysis of Production and Allocation*. John Wiley & Sons, Inc., New-York, 1951.

[Darw 76] C. Darwin. *L'Origine des espèces (6^e édition)*. Reinwald, Paris, 1876. traduction d'Edmond Barbier.

[Dech 83] R. Dechter and J. Pearl. “The Optimality of A* Revisited”. In : *Proceedings of the National Conference on Artificial Intelligence*, pp. 95–99, Washington DC, USA, August 1983.

[Dema 03] S. Demassey. *Méthodes Hybrides de Programmation par Contraintes et Programmation Linéaire pour le Problème d'Ordonnancement de Projet à Contraintes de Ressources*. PhD thesis, Université d'Avignon et des Pays de Vaucluse, Avignon, France, 2003.

[Desr 83] J. Desrosiers, P. Pelletier, and F. Soumis. “Plus court chemin avec contraintes d'horaires”. *RAIRO*, Vol. 17, pp. 357–377, 1983.

[Desr 88] M. Desrochers and F. Soumis. “A Generalized Permanent Labeling Algorithm for the Shortest Path Problem with Time Windows”. *INFOR*, Vol. 26, pp. 191–212, 1988.

[Dijk 71] E. Dijkstra. “A Short Introduction to the Art of Programming”. In : *The Shortest Spanning Subtree of a Graph*, pp. 64–69, T.H. Eindhoven, 1971.

[Dinc 88] M. Dincbas, P. V. Hentenryck, H. Simonis, A. Aggoun, T. Graf, and F. Berthier. “The Constraint Logic Programming Language CHIP”. In : *Proceedings of the International Conference on Fifth Generation Computer Systems (FGCS)*, pp. 693–702, Tokyo, Japan, December 1988.

[Dori 04] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, USA, 2004.

[Dori 92] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Dipartimento di Elettronica, Politecnico di Milano, Milan, Italy, 1992. (in Italian).

[Dori 96] M. Dorigo, V. Maniezzo, and A. Colorni. “Ant system : Optimization by a Colony of Cooperating Agents”. *IEEE Transactions on Systems, Man and Cybernetics, Part B*, Vol. 26, No. 1, pp. 29–41, 1996.

[Dori 97] M. Dorigo and L. Gambardella. “Ant Colony System : a Cooperative Learning Approach to the Travelling Salesman Problem”. *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, pp. 53–66, 1997.

[Dumi 03] I. Dumitrescu and N. Boland. “Improved Preprocessing, Labeling and Scaling Algorithms for the Weight-Constrained Shortest Path Problem”. *Networks*, Vol. 42, pp. 135–153, 2003.

[Dung 11] P. Q. Dung. *LS(Graph) : A Constraint-based Local Search Framework for Constrainted Optimum Tree and Path Problems on Graphs*. PhD thesis, Université Catholique de Louvain, ICTEAM, Louvain-la-Neuve, Belgium, 2011.

[Durr 06] H. Durrant-Whyte and T. Bailey. *Simultaneous Localization and Mapping (SLAM) : Part I The Essential Algorithms*. Vol. 13, 2006.

[El M 06] T. El-Mihoub, A. Hopgood, L. Nolle, and A. Battersby. "Hybrid Genetic Algorithms : A Review". *Engineering Letters*, Vol. 13, No. 2, 2006.

[Ergu 02] F. Ergün, R. Sinha, and L. Zhang. "An improved FPTAS for Restricted Shortest Path". *Information Processing Letters*, Vol. 83, pp. 287–291, 2002.

[Eyck 02] C. Eyckelhof and M. Snoek. "Ant Systems for a Dynamic TSP : Ants Caught in a Traffic Jam". In : M. Dorigo, G. D. Caro, and M. Sampels, Eds., *Proceedings of the Third International Workshop on Ant Algorithms (ANTS)*, pp. 88–99, Springer-Verlag, 2002.

[Fage 96] F. Fages. *Programmation Logique par Contraintes. Cours de l'École Polytechnique*, Ellipses, Paris, 1996.

[Ferg 05] D. Ferguson, M. Likhachev, and A. Stentz. "A Guide to Heuristic-based Path Planning". In : *Proceedings of the Workshop on Planning under Uncertainty for Autonomous Systems at International Conference on Automated Planning and Scheduling (ICAPS)*, Monterey, CA, USA, June 2005.

[Foge 06] D. Fogel. "Foundations of evolutionary computation". In : *Proceedings of the SPIE*, pp. 01.1–01.13, Kissimmee, FL, USA, April 2006.

[Foge 66] L. Fogel, A. Owens, and M. Walsh. *Artificial Intelligence through Simulated Evolution*. John Wiley & Sons, Inc., 1966.

[Foge 88] D. Fogel. "An Evolutionary Approach to the Traveling Salesman Problem". *Biological Cybernetics*, Vol. 60, p. 139–144, 1988.

[Freu 95] E. Freuder and P. Hubbe. "Extracting Constraint Satisfaction Subproblems". In : *Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 548–557, Morgan Kaufmann, Montreal, Quebec, Canada, August 1995.

[Frie 58] R. Friedberg. "A Learning Machine : Part I". *IBM Journal of Research & Development*, Vol. 2, pp. 2–13, 1958.

[Frie 59] R. Friedberg, B. Dunham, and J. North. "A Learning Machine : Part II". *IBM Journal of Research & Development*, Vol. 3, pp. 282–287, 1959.

[Gamb 95] L. Gambardella and M. Dorigo. "Ant-Q : A Reinforcement Learning Approach to the Traveling Salesman Problem". In : *International Conference on Machine Learning*, pp. 252–260, Tahoe City, CA, USA, July 1995.

[Gare 79] M. Garey and D. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, New-York, NY, USA, 1979.

[Geco] "Gecode : a Generic Constraint Development Environment". `http://www.gecode.org`.

[Gend 01] M. Gendreau, G. Laporte, and J.-Y. Potvin. "Metaheuristics for the Capacitated VRP". In : P. Toth and D. Vigo, Eds., *The Vehicle Routing Problem*, pp. 129–154, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[Glov 86] F. Glover. "Future Paths for Integer Programming and Links to Artificial Intelligence". *Computers and Operations Research*, Vol. 13, No. 5, pp. 533–549, 1986.

[Glov 91] F. Glover. “Multilevel Tabu Search and Embedded Search Neighborhoods for the Traveling Salesman Problem”. *Manuscript, School of Business*, 1991.

[Glov 97] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, 1997.

[GNU] “GNU-Prolog”. `http://www.gprolog.org/`.

[Gold 08] E. Goldbarg, M. Goldbarg, and G. de Souza. *Particle Swarm Optimization Algorithm for the Traveling Salesman Problem*. book edited by F.Greco, Universidade Federal do Rio Grande do Norte, Brazil, 2008.

[Gold 85] D. Goldberg and R. L. Jr. “Alleles, Loci and the TSP”. In : J. J. G. (Ed.), Ed., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, pp. 154–159, Lawrence Erlbaum Associates, Carnegie Mellon University, Pittsburgh, PA, USA, July 1985.

[Gold 89] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Kluwer Academic Publishers, Boston, MA, USA, 1989.

[Gomo 63] R. Gomory. “An Algorithm for Integer Solutions to Linear Programs”. *Recent Advances in Mathematical Programming*, 1963. (initially published in Princeton IBM Math. Report, Nov. 58).

[Gref 85] J. Grefenstette, R. Gopal, B. Rosmaita, and D. V. Gucht. “Genetic Algorithms for the TSP”. In : J. J. G. (Ed.), Ed., *Proceedings of the First International Conference on Genetic Algorithms and Their Applications*, p. 160–165, Lawrence Erlbaum Associates, Carnegie Mellon University, Pittsburgh, PA, USA, July 1985.

[Guet 07] C. Guettier. “Solving Planning and Scheduling Problems in Network-based Operations”. In : *Proceedings of the Thirteenth International Conference on Principles and Practice of Constraint Programming (CP)*, Providence, RI, USA, September 2007.

[Gunt 02] M. Guntsch and M. Middendorf. “Applying Population Based ACO to Dynamic Optimization Problems”. In : M. Dorigo, G. D. Caro, and M. Sampels, Eds., *Proceedings of the Third International Workshop on Ant Algorithms (ANTS)*, pp. 111–122, Springer-Verlag, 2002.

[Hama 09] Y. Hamam and H. Talbot. “Programmation Linéaire et Optimisation Combinatoire”. 2009. Cours de cursus ingénieur ESIEE, Noisy le Grand, France.

[Hand 80] G. Handler and I. Zang. “A Dual Algorithm for the Constrained Shortest Path Problem”. *Networks*, Vol. 10, pp. 293–310, 1980.

[Hans 07] E. Hansen and R. Zhou. “Anytime Heuristic Search”. *Journal of Artificial Intelligence Research*, Vol. 28, pp. 267–297, 2007.

[Hard 11] S. Harding. “Genetic Programming On General Purpose Graphics Processing Units”. 2011. `http://www.gpgpgpu.com/`.

[Hart 68] P. Hart, N. Nilsson, and B. Raphael. “A Formal Basis for the Heuristic Determination of Minimum Cost Paths”. *Transactions on Systems, Science and Cybernetics, IEEE*, Vol. 4, No. 2, pp. 100–107, 1968.

[Harv 95] W. Harvey and M. Ginsberg. “Limited Discrepancy Search”. In : *Proceedings of the Fourteenth International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 607–613, Montréal, Québec, Canada, August 1995.

[Hass 92] R. Hassin. “Approximation Schemes for the Restricted Shortest Path Problem”. *Mathematics of Operations Research*, Vol. 17, No. 2, pp. 36–42, 1992.

[Hast 70] W. Hastings. "Monte Carlo Sampling Methods Using Markov Chains and Their Applications". *Biometrika*, Vol. 57, No. 1, pp. 97–109, 1970.

[Haup 04] R. Haupt and S. Haupt. *Practical Genetic Algorithms, Second Edition*. John Wiley & Sons, Inc., 2004.

[Holl 75] J. Holland. *Adaptation In Natural And Artificial Systems*. University of Michigan Press, 1975.

[IBM] "IBM ILOG CPLEX CP Optimizer". `http://www-01.ibm.com/software/integration/optimization/cplex-cp-optimizer`.

[JaCo] "JaCoP : a Java Constraint Programming Library". `http://jacop.cs.lth.se`.

[Jacq 01] P. Jacquet, P. Mühlethaler, T. Clausen, A. Laouiti, A. Qayyum, and L. Viennot. "Optimized Link State Routing Protocol for Ad Hoc Networks". In : *Proceedings of the IEEE International Multi-Topic Conference (INMIC)*, pp. 62–68, Lahore, Pakistan, December 2001.

[Jaff 87] J. Jaffar and J.-L. Lassez. "Constraint Logic Programming". In : *Proceedings of the Fourteenth Annual Symposium on Principles of Programming Languages*, pp. 111–119, ACM Press, Munich, Germany, January 1987.

[Jarv 73] R. Jarvis. "On the identification of the convex hull of a finite set of points in the plane". *Information Processing Letters*, Vol. 2, p. 18–21, 1973.

[Jarv 85] R. Jarvis. "Collision-Free Trajectory Planning Using the Distance Transforms". *Mechanical Engineering Transactions of the Institution of Engineers*, Vol. 10, No. 3, pp. 187–191, 1985.

[Jaum 96] B. Jaumard, F. Semet, and T. Vovor. "A Two-Phase Resource Constrained Shortest Path Algorithm for Acyclic Graphs". *Les Cahiers du GERAD*, Vol. G-96-48, 1996.

[Joks 66] H. Joksch. "The Shortest Path Route Problem with Constraints". *Journal of Mathematical Analysis and Application*, Vol. 14, pp. 191–197, 1966.

[Jutt 01] A. Jüttner, B. Szviatovszki, I. Mécs, and Z. Rajkó. "Lagrange Relaxation Based Method for the QoS Routing Problem". In : *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, pp. 859–868, Anchorage, AK, USA, April 2001.

[Kain 94] H. Kaindl and A. Khorsand. "Memory-Bounded Bidirectional Search". In : *Proceedings of the Twelfth National Conference on Artificial Intelligence*, pp. 1359–1364, Seattle, WA, USA, August 1994.

[Karm 84] N. Karmarkar. "A New Polynomial-time Algorithm for Linear Programming". *Combinatorica*, Vol. 4, No. 4, pp. 373–395, 1984.

[Karp 72] R. Karp. "Reductibility among Combinatorial Problems". *Complexity of Computer Computations*, pp. 85–103, 1972.

[Kenn 95] J. Kennedy and R. Eberhart. "Particle Swarm Optimization". In : *Proceedings of IEEE International Conference on Neural Networks*, p. 1942–1948, University of Western Australia, Perth, Australia, November 1995.

[Khac 79] L. Khachiyan. "A Polynomial Algorithm in Linear Programming". *Soviet Mathematics Doklady*, Vol. 20, pp. 191–194, 1979.

[Khic 10] M. Khichane. *Optimisation sous contraintes par Intelligence Collective Auto-adaptative*. PhD thesis, Université Claude Bernard, Octobre 2010.

[Kirk 83] S. Kirkpatrick, C. Gelatt, and M. Vecchi. “Optimization by Simulated Annealing”. *Science*, Vol. 220, No. 4598, pp. 671–680, may 1983.

[Klee 69] V. Klee and G. Minty. “How Good is the Simplex Algorithm ?”. In : *Proceedings of the Third Symposium on Inequalities*, pp. 159–175, University of California, Los Angeles, CA, USA, September 1969.

[Koen 02a] S. Koenig. “D* Lite”. In : *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 476–483, Edmonton, Alberta, Canada, July 2002.

[Koen 02b] S. Koenig and M. Likhachev. “Incremental A*”. *Advances in Neural Information Processing Systems*, Vol. 14, 2002.

[Koen 06] S. Koenig and M. Likhachev. “Real-Time Adaptive A*”. In : *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Hakodate, Japan, May 2006.

[Koen 07] S. Koenig and X. Sun. “The Fringe-Saving A* Search Algorithm - A Feasible Study”. In : *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 2391–2397, Hyderabad, India, January 2007.

[Koen 09] S. Koenig, X. Sun, and W. Yeoh. “Dynamic Fringe-Saving A*”. In : *Proceedings of the Eighth International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 891–898, Budapest, Hungary, May 2009.

[Koen 98] S. Koenig. “Real-Time Heuristic Search : Research Issues”. In : *Proceedings of the AIPS’98 Workshop on Planning as Combinatorial Search : Propositional, Graph-Based, and Disjunctive Planning Methods*, pp. 75–79, Carnegie Mellon University, Pittsburgh, PA, USA, June 1998.

[Korf 85] R. Korf. “Depth-First Iterative-Deepening : An Optimal Admissible Tree Search”. *Artificial Intelligence*, Vol. 27, pp. 97–109, 1985.

[Korf 87] R. Korf. “Real-time Heuristic Search : First Results”. In : *Proceedings of the Sixth National Conference on Artificial Intelligence*, pp. 133–138, Seattle, WA, USA, July 1987.

[Korf 88] R. Korf. “Real-time Heuristic Search : New Results”. In : *Proceedings of the Seventh National Conference on Artificial Intelligence*, pp. 139–144, Saint Paul, MN, USA, August 1988.

[Korf 90] R. Korf. “Real-time Heuristic Search”. *Artificial Intelligence*, Vol. 42, No. 2-3, pp. 189–211, 1990.

[Korf 93] R. Korf. “Linear-Space Best-First Search”. *Artificial Intelligence*, Vol. 62, No. 1, pp. 41–78, 1993.

[Korf 96] R. Korf. “Improved Limited Discrepancy Search”. In : *Proceedings of the Thirteenth National Conference on Artificial Intelligence (AAAI)*, pp. 286–291, Portland, OR, USA, August 1996.

[Kort 00] B. Korte and J. Vygen. *Combinatorial Optimization*. Springer, Berlin, Germany, 2000.

[Kowa 74] R. Kowalski. “Predicat Logic as Programming Language”. *Information Processing*, No. 74, pp. 569–574, 1974.

[Koza 92] J. Koza. *Genetic Programming : On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

[Lada 01] L. Ladányi, T. Ralphs, and L. Trotter. “Branch, Cut and Price : Sequential and Parallel”. *Computational Combinatorial Optimization*, Vol. 2241/2001, pp. 223–260, 2001.

[Lang 02] W. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

[Lang 92] P. Langley. “Systematic and Non-Systematic Search Strategies”. In : *Proceedings of the First International Conference on Artificial Intelligence Planning Systems (AIPS)*, pp. 145–152, College Park, MD, USA, June 1992.

[Lapo 92] G. Laporte. “The Vehicle Routing Problem : An Overview of Exact and Approximate Algorithms”. *European Journal of Operational Research*, Vol. 59, No. 3, pp. 345–358, June 1992.

[Larr 99] P. Larrañaga, C. Kuijpers, R. Murga, I. Inza, and S. Dizdarevic. “Genetic Algorithms for the Travelling Salesman Problem : A Review of Representations and Operators”. *Artificial Intelligence*, Vol. 13, pp. 129–170, 1999.

[Lawl 85] E. Lawler, J. Lenstra, A. Rinnooy-Kan, and D. Shmoys. *The Traveling Salesman Problem*. John Wiley & Sons, Chichester, UK, 1985.

[Lazi 09] A. Lazinica. *Particle Swarm Optimization*. Éditions In-Tech, 2009.

[Lin 65] S. Lin. “Computer Solutions of the Traveling Salesman Problem”. *Bell System Technical Journal*, Vol. 44, pp. 2245–2269, 1965.

[Lin 73] S. Lin and B. Kernighan. “An Effective Heuristic Algorithm for the Traveling-Salesman Problem”. *Operations Research*, Vol. 21, pp. 498–516, 1973.

[Lloy 87] J. Lloyd. *Foundations of Logic Programming*. Springer-Verlag, second Ed., September 1987.

[Lore 99] D. Lorenz and D. Raz. “A Simple Efficient Approximation Scheme for the Restricted Shortest Path Problem”. *Operations Research Letters*, Vol. 28, pp. 213–219, 1999.

[Lour 02] H. Lourenço, O. Martin, and T. Stützle. “Iterated local search”. In : F. Golver and G. Kochenberger, Eds., *Handbook of Metaheuristics*, pp. 321–353, Kluwer Academic Publisher, Norwell, MA, USA, 2002.

[Luca] F. Lucas and C. Joubert. `http://www-roc.inria.fr/imara/dw/caor/benchmarks/missionplanning`.

[Luca 10] F. Lucas and C. Guettier. “Automatic Vehicle Navigation with Bandwidth Constraints”. In : *Proceedings of the Military Communications Conference (MILCOM)*, San Jose, CA, USA, November 2010.

[Luca 12a] F. Lucas and C. Guettier. “Hybrid Solving Technique for Vehicle Planning with Communication Constraints”. November 2012. à paraître.

[Luca 12b] F. Lucas, C. Joubert, C. Guettier, P. Siarry, and A. de La Fortelle. “A New Benchmark for the Energy-Constrained Fastest Path with Waypoints Problem”. 2012. à paraître.

[Lume 86] V. Lumelsky and A. Stepanov. “Dynamic Path Planning for a Mobile Automaton with Limited Information on the Environment”. *IEEE Transactions on Automatic Control AC-31(11)*, 1986.

[Mack 77] A. Mackworth. “Consistency in Networks of Relations”. *Artificial Intelligence*, Vol. 8, pp. 99–118, 1977.

[Mart 91] O. Martin, S. Otto, and E. Felten. "Large-Step Markov Chains for the Traveling Salesman Problem". *Complex Systems*, Vol. 5, No. 3, pp. 299–326, June 1991.

[Mena 09] J. Menana and S. Demassey. "Sequencing and Counting with the textttmulticost-regular Constraint". In : *Proceedings of the 6th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR'09)*, pp. 178–192, Pittsburgh, PA, USA, 2009.

[Metr 53] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. "Equations of State Calculations by Fast Computing Machines". *Journal of Chemical Physics*, Vol. 21, No. 6, pp. 1087–1092, 1953.

[Mitc 00] J. Mitchell. *Branch-and-Cut Algorithms for Combinatorial Optimization Problems*. Oxford University Press, 2000.

[Mohr 86] R. Mohr and T. Henderson. "Arc and Path Consistency Revisited". *Artificial Intelligence*, Vol. 28, pp. 225–233, 1986.

[Monm 09] N. Monmarché, F. Guinand, and P. Siarry. *Fourmis Artificielles, nouvelles directions pour une intelligence collective*. Éditions Hermès-Lavoisier, 2009.

[Mont 74] U. Montanari. "Networks of constraints : Fundamental properties and applications to picture processing". *Information Science*, Vol. 7, No. 2, pp. 95–132, 1974.

[Nils 80] N. Nilsson. *Principles of Artificial Intelligence*. Tioga Publishing Company, 1980.

[Norv 95] P. Norvig and S. Russel. *Artificial Intelligence : A Modern Approach*. Editions Prentice Hall, 1995.

[Oliv 87] I. Oliver, D. Smith, and J. Holland. "A Study of Permutation Crossover Operators on the TSP". In : J. J. G. (Ed.), Ed., *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications*, pp. 224–230, Lawrence Erlbaum Associates, Massachussets Institute of Technology, Cambridge, MA, USA, July 1987.

[Pill 11] V. Pillac, M. Gendreau, C. Guéret, and A. Medaglia. "A review of dynamic vehicle routing problems". Tech. Rep. CIRRELT-2011-62, Centre interuniversitaire de recherche sur les réseaux d'enterprise, la logistique et le transport (CIRRELT), Montreal, Canada, 2011.

[Pirz 90] A. Pirzadeh and W. Snyder. "A Unified Solution to Coverage and Search in Explored and Unexplored Terrains Using Indirect Control". In : *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 2113–2119, Cincinnati, OH , USA, May 1990.

[Pohl 70a] I. Pohl. "First Results on the Effect of Error in Heuristic Search". *Machine Intelligence*, Vol. 5, pp. 219–236, 1970.

[Pohl 70b] I. Pohl. "Heuristic Search Viewed as Path Finding in a Graph". *Artificial Intelligence*, Vol. 1, No. 3, pp. 193–204, 1970.

[Poli 08] R. Poli, W. Langdon, and N. McPhee. *A Field Guide to Genetic Programming (with contributions by J.R. Koza)*. Lulu, 2008. Freely available for download at `http://www.gp-field-guide.org.uk`.

[Ques 06] L. Quesada, P. V. Roy, Y. Deville, and R. Collet. "Using Dominators for Solving Constrained Path Problems". In : *Proceedings of the 8th International Symposium on Practical Aspects of Declarative Languages*, pp. 73–87, Springer-Verlag, Charleston, SC, USA, January 2006.

[Ravi 02] R. Ravindran, K. Thulasiraman, A. Das, K. Huang, G. Luo, and G. Xue. "Quality of Services Routing : Heuristics and Approximation Schemes with a Comparative Evaluation". In : *Proceedings of IEEE Symposium on Circuits and Systems (ISCAS)*, pp. 775–778, Scottsdale, AZ, USA, May 2002.

[Rech 65] I. Rechenberg. *Cybernetic Solution Path of an Experimental Problem*. Royal Aircraft Establishment Library Translation, 1965.

[Rech 73] I. Rechenberg. *Evolutionsstrategie : Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Frommann-Holzboog Verlag, Stuttgart, Germany, 1973.

[Regi 04] J.-C. Régin. "CAC : Un algorithme d'arc-consistance configurable, générique et adaptatif". In : *Actes de la conférence JNPC'04*, pp. 315–329, Angers, France, Juin 2004.

[Rein 08] G. Reinelt. "TSPLIB Benchmark Library". 2008. `http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/`.

[Rein 94] A. Reinefeld and T. Marsland. "Enhanced Iterative-Deepening Search". *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 16, No. 7, pp. 701–710, 1994.

[Rgin 99] J.-C. Régin. "Arc Consistency for Global Cardinality Constraints with Costs". In : J. Jaffar, Ed., *Principles and Practice of Constraint Programming (CP'99)*, p. 390–404, Springer -Verlag, 1999.

[Robi 65] J. Robinson. "A Machine-oriented Logic Based on the Resolution Principle". *Journal of the Association for Computing Machinery (JACM)*, Vol. 12, No. 1, pp. 23–41, January 1965.

[Russ 92] S. Russell. "Efficient Memory-Bounded Search Methods". In : *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI)*, pp. 1–5, Vienna, Austria, August 1992.

[Sakk 00] H. E. Sakkout and M. Wallace. "Probe Backtrack Search for Minimal Perturbation in Dynamic Scheduling". *Constraints*, Vol. 5, No. 4, pp. 359–388, 2000.

[Samu 63] A. Samuel. *Some Studies in Machine Learning Using the Game of Checkers*. E.A. Feigenbaum and J. Feldman (Eds.), McGraw-Hill, New York, 1963.

[Samu 67] A. Samuel. "Some Studies in Machine Learning Using the Game of Checkers II - Recent Progress". *IBM Journal of Research & Development*, Vol. 11, No. 6, pp. 601–617, 1967.

[Sell 03] M. Sellmann, T. Gellermann, and R. Wright. "Cost-based Filtering for Shorter Path Constraints". In : *Proceedings of the Ninth International Conference on Principles and Practice of Constraint Programming (CP)*, pp. 694–708, Kinsale, Ireland, September 2003.

[SICS] "SICStus Prolog". `http://www.sics.se/isl/sicstuswww/site/index.html`.

[Smol 95] G. Smolka. "The Oz Programming Model". *Computer Science Today*, Vol. 1000, pp. 324–344, 1995.

[Soln 02] C. Solnon. "Ants can solve Constraint Satisfaction Problems". *IEEE Transactions on Evolutionary Computation*, Vol. 6, No. 4, pp. 347–357, August 2002.

[Sten 94] A. Stentz. “Optimal and Efficient Path Planning for Partially Known Environments”. In : *Proceedings of the International Conference on Robotics and Automation (ICRA)*, pp. 3310–3317, San Diego, CA, USA, May 1994.

[Sten 95] A. Stentz. “The Focussed D* Algorithm for Real-Time Replanning”. In : *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1652–1659, Montréal, Québec, Canada, August 1995.

[Stut 00] T. Stützle and H. Hoos. “MAX-MIN Ant System”. *Future Generation Computer Systems*, Vol. 16, No. 8, pp. 889–914, 2000.

[Stut 10] T. Stützle, M. López-Ibáñez, P. Pellegrini, M. Maur, M. M. de Oca, M. Birattari, and M. Dorigo. “Parameter Adaptation in Ant Colony Optimization”. *Iridia Technical Report Series, Report n°TR/IRIDIA/2010-002*, Janvier 2010.

[Stut 97] T. Stützle and H. Hoos. “Improvements on Ant System : Introducing MAX-MIN Ant System”. In : *Proceedings of the Third International Conference of Artificial Neural Networks and Genetic Algorithms*, Springer-Verlag, Norwich, UK, April 1997.

[Sun 08] X. Sun, S. Koenig, and W. Yeoh. “Generalized Adaptive A*”. In : *Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, pp. 469–476, Estoril, Portugal, 2008.

[Suur 84] J. Suurballe and R. Tarjan. “A Quick Method for Finding Shortest Pairs of Disjoint Paths”. *Networks*, Vol. 14, pp. 325–336, 1984.

[Tail 91] É. Taillard. “Robust Taboo Search for the Quadratic Assignment Problem”. *Parallel Computing*, Vol. 17, pp. 443–455, 1991.

[Team 10] C. Team. “Choco : an Open Source Java Constraint Programming Library”. Tech. Rep. 10-02-INFO, Ecole des Mines de Nantes, France, 2010. `http://www.em,.fr/z-info/choco-solver/pdf/choco-presentation.pdf`.

[Thay] J. Thayer. “Search Visualizations”. `http://www.cs.unh.edu/~jtd7/stills/`.

[Turi 95] A. Turing and J.-Y. Girard. *La machine de Turing*. Éditions du Seuil, 1995.

[Wals 97] T. Walsh. “Depth-Bounded Discrepancy Search”. In : *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI)*, pp. 1388–1393, Nagoya, Japan, August 1997.

[Wang 96] Z. Wang and J. Crowcroft. “Quality-of-Service Routing for Supporting Multimedia Applications”. *IEEE Journal on Selected Areas in Communications*, Vol. 14, pp. 1228–1234, September 1996.

[Whit 03] T. White, S. Kaegi, and T. Oda. “Revisiting Elitism in Ant Colony Optimization”. In : *Proceedings of Genetic and Evolutionary Computation Conference (GECCO)*, pp. 122–133, Chicago, IL, USA, July 2003. LNCS 2723.

[Xiao 05] Y. Xiao, K. Thulasiraman, G. Xue, and A. Jüttner. “The Constrained Shortest Path Problem : Algorithmic Approaches and an Algebraic Study with Generalization”. *AKCE International Journal of Graphs and Combinatorics*, Vol. 2, No. 2, pp. 63–86, 2005.

[Xiao 06] Y. Xiao, K. Thulasiraman, and G. Xue. “QoS Routing in Communication Networks : Approximation Algorithms Based on the Primal Simplex Method of Linear Programming”. *IEEE Transactions on Computer*, Vol. 55, No. 7, pp. 815–829, 2006.

[YAPr] “YAProlog : Yet Another Prolog”. `http://dcc.fr.up.pt/~vsc/Yap`.

[Zeli 92] A. Zelinsky. "A Mobile Robot Exploration Algorithm". *IEEE Transactions on Robotics and Automation*, Vol. 8, No. 6, pp. 707–717, 1992.

Publications

Hybridisation of Constraint Solving with an Ant Colony Algorithm for Vehicle On-Line Path Planning

Christophe Guettier[1], **Francois Lucas**[1] **and Patrick Siarry**[2]
[1]SAGEM, 27, Rue Leblanc, 75012 Paris, France
{christophe.guettier, francois.lucas}@sagem.com
[2]Univ. of Paris XII Val-de-Marne, Sciences and Technology Faculty
61, av du G^{al} de Gaulle, 94010 Creteil, France
patrick.siarry@univ-paris12.fr

Abstract

This paper presents a hybrid solving method for vehicle path planning problems. As part of the vehicle system architecture (vetronic), planning is dynamic and has to be activated on-line which require response times to be compatible with mission execution. The proposed approach combines a complete method based on constraint solving techniques with an Ant Colony (ACO) metaheuristic. ACO is used to solve a relaxed problem as a pre-processing step. The hybridisation then relies on a probing technique that order variables according to a metric built on a distance information to the best solution found by ACO, allowing to guide a Branch&Bound search method. Various forms of strategies are compared and evaluated on real world scenarios. Preliminary results exhibit response times close to vehicle control requirements, on realistic problem instances.

Introduction

Mainly in space and defense domains, mission planning has always been a major challenge for the planning community, in terms of problem formulation, modelling, search techniques and evaluation. In critical mission systems for military vehicles, planning has been so far considered separated from navigation, in particular at the tactical level. However, modern operations take place in urban environments which involve versatile threats and require high tactical mobility.

For manned vehicle applications, the goal is to provide driver decision support functionalities, such as advising the best route to follow under specific mission constraints. To face environment uncertainty (e.g. obstacles, hostile threats), on-line planning algorithms must have execution times close to that of human reflexes. For unmanned vehicles (UAV, UGV, etc.), navigation plans must be updated whenever the mission objective changes, the environment evolves significantly or the expected amount of resources is not satisfactory. In addition, and according to the vehicle type and its on-board system (also called vetronic in the following), on-line planning must be compatible with mission tempo.

Much research has been carried out on mission planning. Generic planning formalisms (Long and Fox 2000)(Ghallab et al. 1998) have highlighted problem complexity, which can be tackled through domain-independent search methods and heuristics. These approaches may not match on-line planning system requirements, embedded in vehicles. Specific techniques can fulfil on-line requirements such as in (Meuleau and al 2008), but may not encompass the spectrum of operational constraints.

This paper focuses on Constrained Vehicle Planning (CVP) problems. A hybrid path planning method is presented to address dynamic mission management for both manned or unmanned vehicles. To match both on-line response time and multiple requirements, we combine a complete solving based on Constraint Programming (CP) and Branch&Bound strategy with an Ant Colony Optimisation (ACO) algorithm. ACO is a stochastic metaheuristic that adapt easily to path planning and is efficient for discrete and dynamic state space problems (Di Caro & al 2005). We use it in a pre-processing step to build a probe that guides a Branch&Bound solving. CP provides generic expressiveness and efficient solving techniques for global optimisation and constraint satisfaction. Also, the problem instances we consider are relatively small, since environment horizon under consideration is limited.

To our knowledge, our approach is new. Other work has been led on ACO-CP hybridisation in the literature (Solnon et al. 2008) in which a CP solver uses an ACO algorithm as heuristic and backtracking method. Nevertheless, it is an incomplete method. In our case, the ACO solution is only used to order variables but the solving is still complete.

Experimentations underline interesting performances on representative problems (mission, urban, or open environment). Memory consumption, computation load and execution time are compatible with the shape of problem instances as well as on-line requirements.

Vehicle Navigation

Constrained Path Planning

The CVP problem consists in finding a path from the current vehicle location to an objective waypoint. Intermediate mandatory waypoints can be imposed to fulfil secondary objectives. This problem can be transformed into a TSP, known as NP-complete : the core difficulty is then to find an optimal sequence of mandatory waypoints to visit. According to user experience, two kinds of plan optimisation are interest-

ing: minimising mission duration and maximizing mission safety. However, various other operational metrics can be imposed as hard constraints, and represented as distances, such as energy or capabilities. The optimisation criterion is minimizing the time to destination. In addition, only vehicle energy will be considered as a secondary metric.

Example

As an illustrative example, let us consider the following situation inspired from a real case (fig. 1). During mission execution, the planner has only partial awareness of its environment. The knowledge horizon is given by ground observability, provided by vehicle team, sensors or external observations. Yellow-filled circles and arrows represent waypoints and feasible paths between waypoints respectively. Transparent circles and dotted arrows represent waypoints and transitions that are situated beyond the observability horizon. One or more waypoints, as the bold waypoint on the figure, may be imposed along the vehicle route.

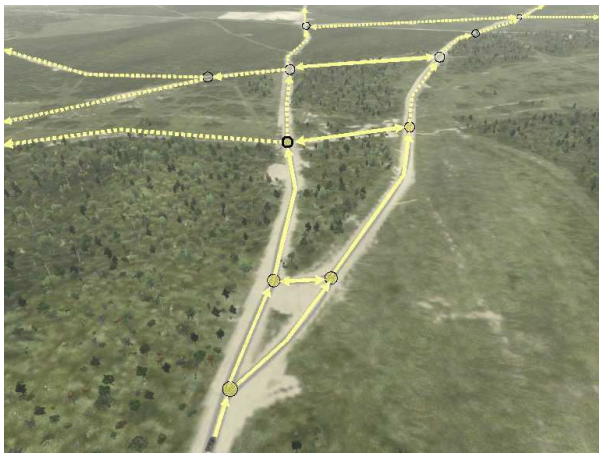


Figure 1: Example of alternative paths for a ground vehicle horizon, and corresponding to the line of sight of a cooperating UAV.

Environment horizon and response time

In our approach, on-line planning is solved over a limited horizon, from the current vehicle position. It corresponds to the terrain on which the vehicle vetronic has enough detailed information to characterise its trafficability. Therefore, the number of mandatory waypoints is relatively small. They correspond to short-term objectives or narrow manoeuvres executed by the vehicle. The following requirements drive response time of on-line planning:

- Computation time must be consistent with mission tempo, so that secondary objectives are reached.
- The generated plan must comply with vehicle control envelope. The control envelope on immediate feasible trajectories decreases when plan generation response time increases.
- In the case of manned vehicles, response time has to match pilot anticipation skills.
- In the case of unmanned vehicles, other processing (situation awareness, generation of flight control commands) must be done.

Note that if a plan cannot be quickly solved, the vehicle may stop if this is possible. This is an ultimate solution that is not satisfactory from an operational point of view. Finally, due to vetronic processing resources, computation load, memory usage and response time must be reduced as much as possible.

Hybrid solving approach

CP advantages combine a high level of expressiveness and powerful constraint solving techniques. It matches composite problems like CVP which requires formulation of different related models (Van Hentenryck et al. 1995). Following this approach, a flow $\{0, 1\}$ model of planning problems is proposed, which supports multiple distance metrics. This graph-based model of the terrain is very useful to represent tactical mobility (positions, progression axes, objectives), and vehicle abilities. CP also provides primitives such as Arc Consistency (AC) for constraint propagation, Branch and Bound (B&B) for optimisation and tree search (backtracking).

The ant colony search is a stochastic approach that combines heuristic search and learning in multiple cycles. It uses the same problem representation, but with a relaxed formulation. Only a single metric is considered and secondary objectives are modelled with a vertex parameter. Within a given cycle, a set of ants is deployed and finds some paths over the graph. A probabilistic law, depending on experience, is associated to a given ant in order to decide over alternative edges. Good quality solutions incrementally update the experience over several cycles. For a given cycle, the previous experience guides the search by attracting ants toward good path solutions. This technique allows the search to explore the state space despite any other implemented heuristic.

The hybridisation approach uses a static probing technique. The goal is to guide a complete strategy with the stochastic algorithm. The prober encapsulates the ACO search, which returns a probing solution to the relaxed problem. Instead of dynamic probing with tentative values such as in(El Sakkout & Wallace 2000), this search strategy uses a static prober which orders problem variables to explore according to the relaxed solution properties. Then, the solving follows the CP search strategy, combining B&B and AC.

Constrained Vehicle Planning Formulation

The terrain is represented as an undirected graph structure (see Fig. 1), where edges define progression axes and vertices tactical positions (or locations). Vertices also represent primary and secondary objectives. Other constraints can impose vertices or edges to be excluded or included in the vehicle plan. Lastly, operational metrics (protection, vulnerability, capacity) can also be associated to edges. The input specification can be expressed using terrain structure, initial

conditions, mission objectives and vehicle capabilites. The following elements are known off-line and characterise this input specification:

- Initial conditions: The starting location and resources initially available.
- Objectives: Some of the locations can correspond to secondary or primary objectives.

Basic constraints

The space of possible plans is represented as a directed graph $G(X, U)$ where the set of edges U represents possible progression axes and the set of vertices X possible position (or navigation) locations[1]. A vehicle starts from vertex $start$ and must reach its objective at vertex end. A path is defined by the set of positive flows. A set of variables $\varphi_u \in \{0, 1\}$ models a possible path from $start$ to end, where the edge u belongs to the path if and only if decision variable φ_u is instantiated to 1.

From an initial position to a final one, path consistency is asserted by the following constraints, where $\omega^+(x) \subset U$ and $\omega^-(x) \subset U$ are outgoing and incoming edges from vertex x, respectively.

$$\sum_{u \in \omega^+(start)} \varphi_u = 1, \quad \sum_{u \in \omega^-(end)} \varphi_u = 1, \quad (1)$$

$$\forall x \in X \setminus \{start, end\}, \sum_{u \in \omega^+(x)} \varphi_u = \sum_{u \in \omega^-(x)} \varphi_u \leq 1 \quad (2)$$

Nodes $start$ and end respectively represent current position and primary objective for the vehicle. Equation (2) ensures path connectivity and unicity while equation (1) imposes limit conditions for the extremities of the path. This constraint gives a linear chain alternating positions and mobility actions (along progression axes) along the graph.

Capability metrics

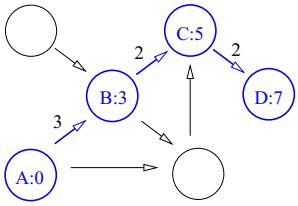
Assuming a given date D_x associated with a position (e.g. vertex) x, we formulate path length formulation (3) often considered in Operation Research (OR) (Gondran and Minoux 1995). Variable D_x expresses the time at which the vehicle reaches position x (see example in figure 2). Assuming that constants $d_{(x',x)}$ represent the time taken to perform a movement from location x' to x, we have:

$$\forall x \in X, D_x = \sum_{(x',x) \in \omega^-(x)} \varphi_{(x',x)}(d_{(x',x)} + D_{x'}) \quad (3)$$

Constants $d_{(x,x')}$ are critical decision variables in the problem and make constraints (3) non linear by terms $D_{x'}$. Finally, the mission schedule can be represented as $\Delta = \{(x, D_x) | \, x \in X, \; D_x > 0\}$.

An equivalent constraint-based formulation is also used for other mission metrics (Fig. 2), such as energy or capacity.

[1] In the remaining of the paper, a vertex is denoted by x, while an edge can be denoted either by u or by (x, x').



Graph in fig. is a spatial representation of possible moves (edges) and positions (nodes). Moves, that correspond to the set of positive values $\Phi = \{(A, B), (B, C), (C, D)\}$, are represented with blue arrows. Assuming a timing metric (edge values are speeds). Other operational metrics, such as protection, vulnerabiliy, available energy and security are similarly formulated in different experiments.

Figure 2: Illustrating a path with pass-by dates over a graph of locations and progression axes

Hybrid Search

The solving strategies focus on mission duration optimisation, that is minimising the time to destination. This date corresponds to one of the variable set $\{D_{end}\}$.The position end is the primary objective of the vehicle. Decision variables are path variables $\{\varphi_x\}$, timing variables $\{D_u\}$.

Reference algorithm

The basic algorithm is a "generate and test" approach that is described only for complexity analysis purpose. Between any couple of secondary objectives, a shortest path is precomputed. The algorithm then builds a quotient graph where the set of nodes includes all secondary objectives in addition to start and end ones. Quotient graph edges result from the precomputed shortest path, valued with the distance. Here the reference algorithm can only be applied for the relaxed problem (without energy or capability metrics). Algorithm complexity is $O(n!)$.

CP search strategy

All problem formulations and search strategies have been implemented in the $CLP(FD)$[2] SICStus prolog library. Basic search strategy makes use of B&B *minimise* predicate and $CLP(FD)$ constraint AC propagation algorithm. The B&B iterates over an arbitrary order of variables labeling. When a variable choice is done, AC propagates domain variables until a fixed point is reached.

Shortest path hybridisation

Designing a strategy consists in finding the right variables ordering and value filtering. The idea is to use the prober to statically order problem variables, as a preprocessing. Instead of using this solution as an initial tentative value, the prober estimates a distance between any problem variable and the probing solution. The search strategy then defines both variable and value ordering according to the resulting distance set. The technique does not suppress any choices points, such that the solving remains complete.

[2] Constraint Logic Programming, using Finite Domain as algebraic interpretation.)

Ant path hybridisation

Instead of considering a blind shortest path as prober, the proposed algorithm implements an ACO search that uses a similar model of the environment. As introduced before, the method deploys a set of ants over search cycles (the number of ants and the number of cycles being currently defined statically) and learns. At the end of each search, ants provide a set of paths, ordered according to their quality. Best ones (on the model of *Rank-based Ant System* (Bullnheimer & al 1999)) are selected to update experience by reinforcing weights associated to their edges. The experience enables search guidance towards interesting areas (where good solutions were found). Within an ant search, the choice of the next waypoint is given by the following probabilistic equation. For an ant k currently at vertex x, the probability for choosing a vertex x' as its next waypoint is given by:

$$\forall (x,x') \in X^2, x' \in \omega^+(x), P_{(x,x')}(k) = \frac{\tau_{x,x'}^{\alpha} \eta_{x'}^{\beta}}{\sum_{l \in \omega^+(x)} \tau_{x,x'}^{\alpha} \eta_{x'}^{\beta}} \quad (4)$$

where $P_{(x,x')}$ is a probability and thus belongs to $[0,1]$. The $\eta_{x'}$ parameter represents the search heuristic (which is simply the inverse of the minimal distance to go from vertex j to the next objective in this implementation). This parameter tends to choose the closest vertices to the objective. The $\tau_{(x,x')}$ parameter represents the edge weight to go from vertex x to vertex x'. This represents the experience acquired during previous search cycles, which tends to choose edges that belong to known good solutions. Parameters α and β are used for calibration and balance the importance between τ and η parameters.

The hybridisation schema of the ACO algorithm is similar to the shortest path one.

Discussion

There is no universal rule to parameterize the ACO algorithm. It depends on the problem, essentially in terms of graph size and connectivity. The choice of α and β can be decisive if there is a high risk for the search to follow a dead-end path. In this case, by privileging the η term (thus by setting β higher than α), the search will have more chance to fail (or to find bad solutions) if it takes a path towards the objective but which does not lead to it. An analogy can be made with A^* algorithm, whose worst case is a labyrinth in which the only way to reach the objective is opposite to the location's direction. In the other cases, it can be judicious to take more consideration to η that can fastly lead to good solutions. The N parameter depends on the size of the problem. The larger the problem is, the more important the ant population (represented by N) should be, as the number of possible solutions becomes high. The C parameter (number of search cycles, analogous to generations in genetic algorithms) more particularly depends on learning mechanisms. As we know, the quality of the search is a compromise between state space exploration and convergence speed. According to the chosen strategy, C should be low if a fast convergence is wished (the reinforcing edge parameters should then evolve fastly), or high in the contrary (and reinforcing should be mild to maintain alternative paths).

In the current version of our ACO algorithm, parameter values were empirically fixed as follows: $\alpha = 0.6$; $\beta = 1$; $\eta = 0.6$; $C = 6$; $N = 6$; $\rho = 0.08$ (ρ is the pheromone evaporation factor, used to decrease edges attractiveness over time).

Preliminary Results

To evaluate the efficiency of the hybrid algorithm, we compared the performance results with two other search methods : a simple constraint solving algorithm based on arbitrary variable ordering, and a hybrid constraint solving - shortest path algorithm.

Experiments have been run on a dual core CPU, at 2.53GHz with 2Gbytes of memory.

Problem instances To illustrate the approach, experiments on three benchmarks are presented. They are representative of vehicle planning for modern peace keeping missions, both in urban and open environments. The following table gives an idea of problem complexity.

Problems	Bench1	Bench2	Bench3
Environment	urban	urban	open
Vertices	23	22	22
Edges	76	74	68
Variables	723	654	702
Constraints	1944	1750	1886

For each benchmark, fifty distinct problem instances are generated, organised on five series. For a given serie, ten instances with starting and ending nodes are choosen on the graph diameter. For each serie, a fixed set waypoints is imposed. Over the five series the difficulty increases from one imposed waypoint to five ones (see example fig. 3).

Results Response times are given in figure 4 (the y-axis) for the three benchmarks, over the five series (x-axis). The measurement considers the search strategy response time, including the whole probing preprocessing.

The y-axis shows response time range for a given serie. The number of imposed waypoints does not affect response time for hybrid search strategies, compared to the reference algorithm complexity. In contrast, only benchmark 1 is a problem for the basic CP search, when increasing difficulty. Except on benchmark 3, the basic strategy delivers poor results, and even exceeds the ten seconds limit twice on the first benchmark. This underlines the efficiency of variable ordering approach. In twelve experiments over fifteen, ACO algorithm dominates the shortest path guided strategy. However, both hybridisation approaches have response times of a similar order of magnitude.

The total number of backtracks over the 10 runs is presented in figure 5 (y-axis) for each serie (x-axis). The number of backtracks is measured during the B&B search execution. In general, the number of backtracks reflects response times. Also, computation time is not spent on optimisation iterations but on backtracking and variables labeling. For both hybrid strategies, variable ordering guides the search

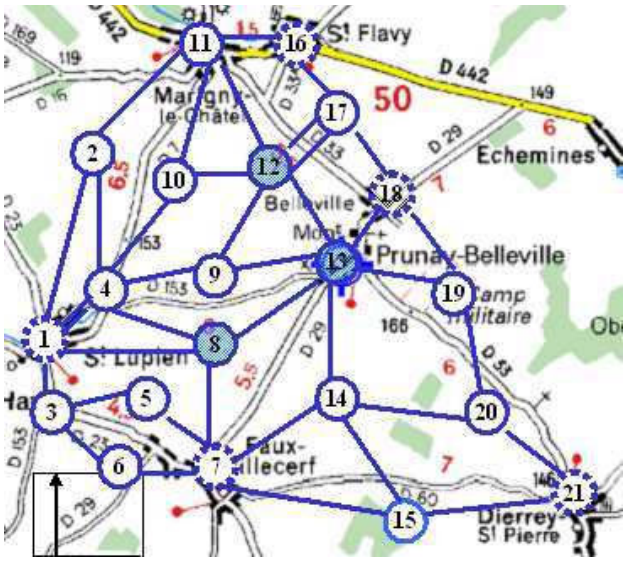


Figure 3: A serie of 3 imposed waypoints (colored circles) on an open environment benchmark. Dashed circles represent possible starting or ending nodes.

significantly so that it reduces the amount of backtracks. In fact, hybrid algorithms find the optimal solutions in few B&B iterations, with minimum backtracks. When comparing these two, ACO search gives better results (a lower number of backtracks), as metaheuristic outputs provide better strategy guidance. On series 2, 3 and 5 of benchmark 3, both hybrid strategies counterperform, compared to the pure CP search strategies. This suggests that the order (formulated by an operational expert, which generally follows a temporal order) is not so arbitrary! This is the case even for the hybrid ACO strategy, in spite of a lower number of backtracks. In these three series, computation time is spent on B&B optimisation for the hybrid ACO strategy while computation is lost in backtracking for the shortest path hybrid one.

The following table compares overhead computation for both shortest path and ACO algorithms. The latter provides acceptable execution time, and is worth the computation savings during the CP search.

Search cycle number and ants population size have been found acceptable for the problem. When these parameters are badly chosen, the metaheuristic may not find a solution. A naive approach would be to increase parameters as much as possible, but it can seriously impact solving time. In fact, these parameters choices represent a compromise between solving time and chances to find a (good) solution.

Overheads	Bench1	Bench2	Bench3
Shortest Path (ms)	$[0, 30]$	$[0, 31]$	$[0, 16]$
Ant Colony (ms)	$[31, 110]$	$[31, 94]$	$[47, 109]$

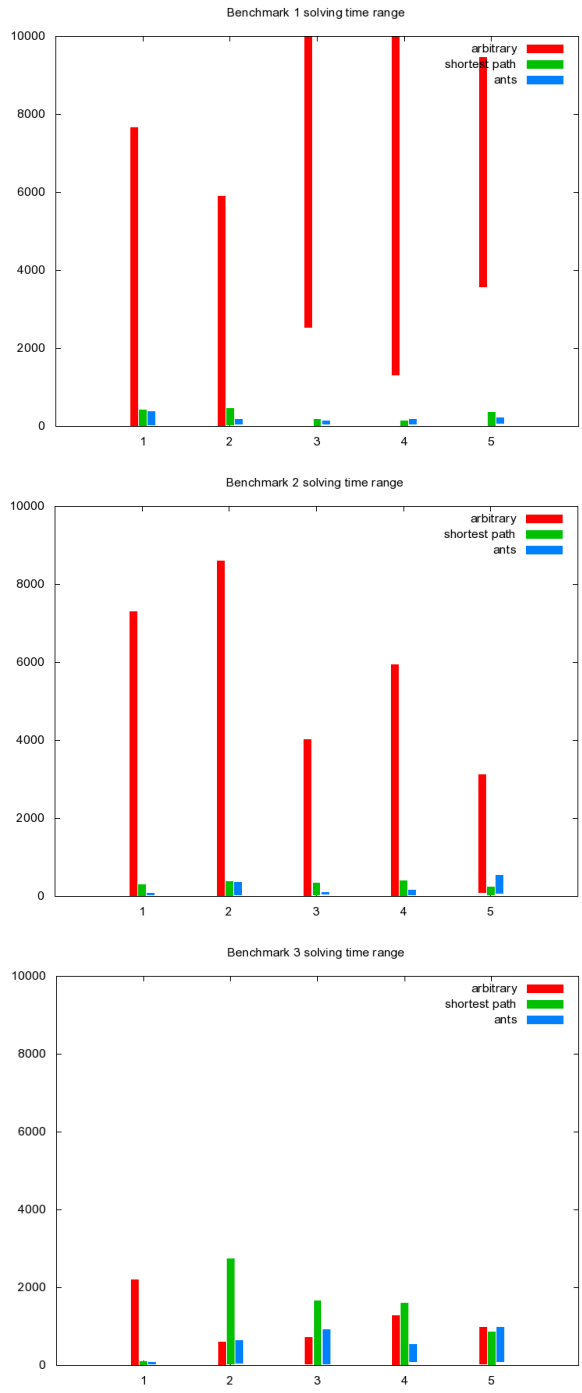


Figure 4: Minimum and maximum response times to reach optimal solution for the three benchmarks, with series of one to five number of imposed waypoints.

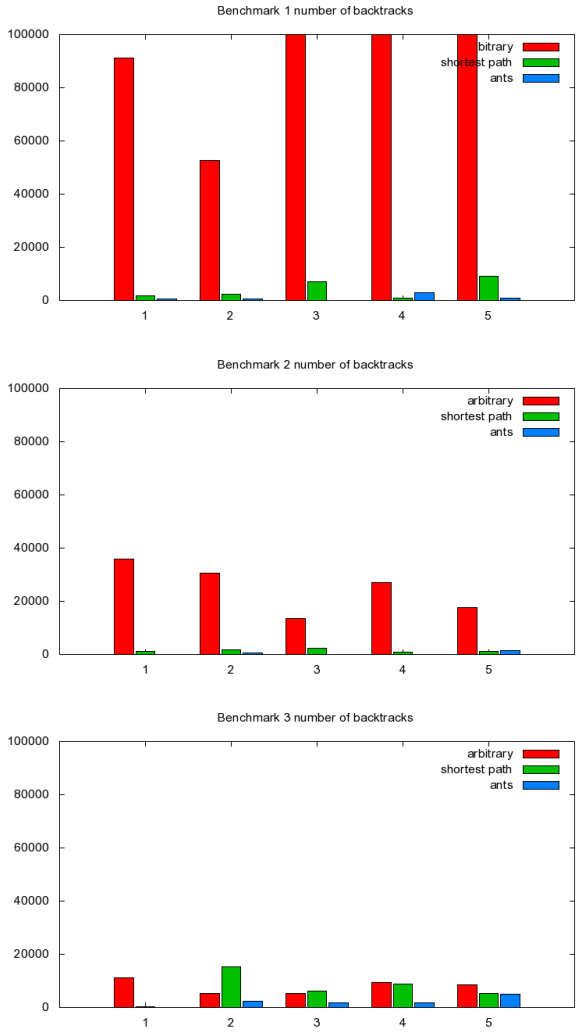


Figure 5: Total number of backtracks needed to find the optimal solution for the three benchmarks. It is summed over series of one to five number of imposed waypoints.

State of the art

Different techniques have been shown to be useful to tackle on-line planning problems, including theoretical formalisms, generic or specific heuristics, constraint solving, and local search techniques.

- Generic planners: Generic planning techniques can be applied to solve such problems. In (Long and Fox 2000), transportation problem classes are proposed, for which preprocessing and dedicated heuristics can be introduced to specialise generic search algorithms.
- Domain specific planners: Much planning research has been done for both military or civilian purposes, relying on specific planning frameworks such as Hierarchical Task Network (HTN)(Goldman et al. 2000).
- Planning with constraint solving: This is the case in IxTeT (Laborie and Ghallab 1995) and HSTS. In Reactive Model-based Programming Language (RMPL) (Kim & al 2001), an evolution of CC languages, the same paradigm is used to dynamically constrain planning representations of one or more remote agents.
- In many respects, CVP can be tackled with operation research algorithms, based on flow models. Basically, the CVP problems can be relaxed as a Traveling Salesman Problem, for which numerous algorithms have been proposed (see (Gondran and Minoux 1995) for example). Local searches have also been widely used for TSP problems. In particular, new metaheuristics such as ACO algorithms can be efficiently applied (Aarst and Lenstra 1997) to TSP.
- Much work addresses dynamic on-line planning, including computer games and exploration vehicles (like NASA's *Mars Rover*). Two efficient kinds of algorithms are particularly widespread : real-time heuristic search methods (like LRTA* (Korf 1990) or RTAA* (Koenig and Likhachev 2006)) and incremental heuristic search methods (like D* Lite (Koenig and Likhachev 2002). The first one only consider a local environment subset to solve, which is updated over time, to limit the problem size. The second approach consider the whole environment but reuses data from previous searches to gain execution time and avoid dead-ends.

Most of constraint programming frameworks are useful to design hybrid search techniques, by integrating OR and Linear Programming algorithms (Ajili and Wallace 2003). However, only a few ones such as (Knight et al. 01), have explored on-line planning requirements.

Conclusion

In this paper, we proposed a new hybrid ACO - CP algorithm to tackle vehicle path planning, whose response times must fit on-line requirements. The CP approach allows higher constraint expressiveness and global solving abilities, while the ACO algorithm is used as a probe to order search variables. The experimentations led on realistic examples clearly showed the interest of variables ordering for search guidance, resulting in a significant reduction of

response time. They also revealed better performance of the ACO algorithm over the shortest path probing, both in terms of solving time and total number of backtracks. However, calibration of ACO algorithms must be tuned according to problem size and structure. An interesting way to improve our approach would be to dynamically parameterise the ACO metaheuristic to fit problem and vetronic requirements. In this preliminary approach, starting location and resource availability are off-line parameters of our solver. A transition to dynamic aspect can be introduced by adding a sliding environment horizon, updating the state space as well as the vehicle state (current position, objectives and remaining resources). Here, the interest of ACO algorithms is to use edge reinforcement of previous searches in new computations. A comparison of this dynamic ACO with other well-known heuristic methods like LRTA* or D* Lite will also be investigated.

References

P. Albert, M. Khichane, C. Solnon : Integration of ACO in a Constraint Programming Language. In 6th International Conference on Ant Colony Optimization and Swarm Intelligence (ANTS), Bruxelles. pp. 84-95. LNCS 5217. Springer.

Aarst,E. and Lenstra,J.K. 1997. Local Search in Combinatorial Optimisation, McGraw Hill, Chichester, UK, 1997.

Ajili,F.; Wallace,M. 2003. Constraint and Integer Programming: Toward a Unified Methodology. In Chapter 6: Hybrid Problem Solving in ECLiPSe. Kluwer Academic Publishers, 2003.

Knight R.; Rabideau G.; Chien S.; Engelhardt B. and Sherwood R. Casper: Space Exploration through Continuous Planning. IEEE Intelligent Systems, vol 16, P.70, 2001.

El Sakkout,H. and Wallace M. Probe Backtrack Search for Minimal Perturbations in Dynamic Scheduling. Constraints Journal, Vol. 5, 2000.

Ghallab,M.; Howe,A.; Knowblock,C.; Mac Dermott,D.; Ram,A.; Veloso,M.; Weld,D.; and Wilkins,D. 1998. PDDL- The Planning Domain Definition Language. Technical Report, CVC TR-98-003/DCS TR-1165.

Gondran,M. and Minoux,M.1995. *Graphes et Algorithmes*, ed. Eyrolles, Paris.

Goldman,R.P.; Haigh,K.Z.; Musliner,D.J.; and Pelican,M. 2000. MACBeth: A Multi-Agent Constraint-Based Planner. Proceedings of the AAAI Workshop on Constraints and AI Planning. Menlo Park, Calif.: AAAI Press.

Laborie,P. and Ghallab,M. 1995. Planning with sharable resource constraints. Proceedings of IJCAI'95. Menlo Park, Calif.: International Joint Conference on Artificial Intelligence, Inc.

Long, D. and Fox, M. 2000. Automatic Synthesis and use of Generic Types in Planning. Proceedings of AAAI 2000. Menlo Park, Calif.: AAAI Press.

Meuleau,N.; Plaunt,C. and Smith D. 2008. Emergency Landing Planning for Damaged Aircraft. In SPA Workshop, International Conference on Automated Planning and Scheduling.

Van Hentenryck,P.; Saraswat,V and Deville,Y. 1995. Design, Implementation and Evaluation of the Constraint Language CC(FD), In Constraint Programming: Basics and Trends, A. Podelski Ed., Springer-Verlag.

Kim,P.; Williams, B and Abramson, M. 2001. Executing Reactive, Model-based Programs through Graph-based Temporal Planning. Proceedings of the International Joint Conference on Artificial Intelligence, Seattle, WA.

Bullnheimer B., Hartl R., Strauss C., A new rank-based version of the ant system : a computational study , Central European Journal for Operations Research and Economics, vol. 7, n1, p. 2538, 1999.

Di Caro G. A., Ducatelle F., Gambardella L., AntHocNet : an adaptive Nature inspired algorithm for routing in mobile ad hoc networks , European Transactions on Telecommunications (ETT), vol. 16, n5, 2005.

R. Korf : Real-Time Heuristic Search, Artificial Intelligence 42 (2-3), pages 189-211, 1990.

S. Koenig, M. Likhachev : Real-Time Adaptive A*, in Proceedings of the AAMAS, pages 281-288, 2006.

S. Koenig, M. Likhachev : D* Lite, in Proceedings of the AAAI, pages 476-483, 2002.

Information Sciences and Computer Engineering, Vol. 1, No. 2, (2010) 75–85

PHARUS

International Journal of
Information Sciences and Computer Engineering

journal homepage: http://www.ijisce.org

Constrained Navigation with Mandatory Waypoints in Uncertain Environment

François Lucas[a], Christophe Guettier[a], Patrick Siarry[b,*], Anne-Marie Milcent[a], Arnaud de La Fortelle[c]

[a]*Sagem Defense and Security, 27 rue Leblanc, 75012 Paris, France*
[b]*Paris XII University, LiSSi Laboratory, 61 avenue du Général de Gaulle, 94010 Créteil, France*
[c]*Mines ParisTech, CAOR Robotics Laboratory, Mathématiques et Systèmes, 60 boulevard Saint Michel, 75006 Paris, France*

Abstract– **This paper presents a hybrid solving method for vehicle path planning problems. As part of the vehicle system architecture (vetronic), planning is dynamic and has to be activated on-line, which requires response times to be compatible with mission execution. The proposed approach combines constraint solving techniques with an Ant Colony Optimization (ACO). The hybridization relies on a static probing technique which builds up a search strategy using a distance information between problem variables and a heuristic solution. Various forms of this approach are compared and evaluated on real world scenarios. Preliminary results exhibit response times close to vehicle control requirements, on realistic problem instances.**

Keyword: Path planning, ant colony optimization, probing, TSP, constraint programming, search.

1. Introduction

Path planning has been a major challenge for decades. It comes up in various fields such as *mobile robot mission planning* where the itinerary to a goal must be minimized, *video games* where character trajectory determination must fit with reactivity demands or *logistics* where complex resource management problems can directly affect company profits.

This paper focuses on the problem of constrained navigation with mandatory waypoints in uncertain environment. We are considering the case of a military vehicle on mission, which is given a final goal and a list of intermediate objectives to reach, whose sequence is not defined. Several constraints have to be taken into account: overall mission time, itinerary length, energy consumption, coordination with other vehicles, etc. Military mission planning has been so far considered separately from navigation issues and defined at mission preparation time. However, with modern on-board hardware and communication architecture (called *vetronic*), mission uncertainty can be managed on-line. In particular, it is possible to provide planning alternatives when contingencies occur.

The goal is to provide driver decision support by advising the best route to follow under specified mission constraints. System efficiency criteria are *solution optimality* and *reactivity*: it must bring the best solution in execution times close to that of human reflexes. This application can be extended to unmanned vehicles, where navigation plans must be updated whenever mission objectives change, the environment evolves significantly or the expected amount of resources is not sufficient. In this paper, we present a new hybrid approach mixing a complete constraint solving with a metaheuristic-based guiding method. Experiments made on representative problems (urban or open environment) show interesting performances.

The problem is described formally in section 2. Section 3 summarizes recent state of the art in the different research fields. The proposed approach is detailed in section 4 and tested in section 5, where results are analyzed.

2. Problem Formulation and Modeling

2.1. The problem

The problem consists in finding a path from the current vehicle location to an objective waypoint. Intermediate mandatory waypoints can be imposed to fulfill secondary objectives. According to the user experience, two kinds of plan optimization are interesting: minimizing overall duration and maximizing mission safety. However, various other operational metrics can be imposed as hard constraints. In the following, we will consider only the vehicle energy as an additional constraint.

This problem is halfway between several well-known instances of the literature:

- *Optimal path finding*, for which the problem is to find a fast, efficient and economic method to connect two points in a graph;
- *Travelling Salesman Problem* (TSP) which is the problem of finding the less expensive hamiltonian cycle over a series of nodes in a compact graph;
- *Vehicle Routing Problem* (VRP) where costs (distance, time) of multi-vehicle routing with capacity constraints must be optimized;

*Corresponding author:
Email address: siarry@univ-paris12.fr, Ph: +33 145171567

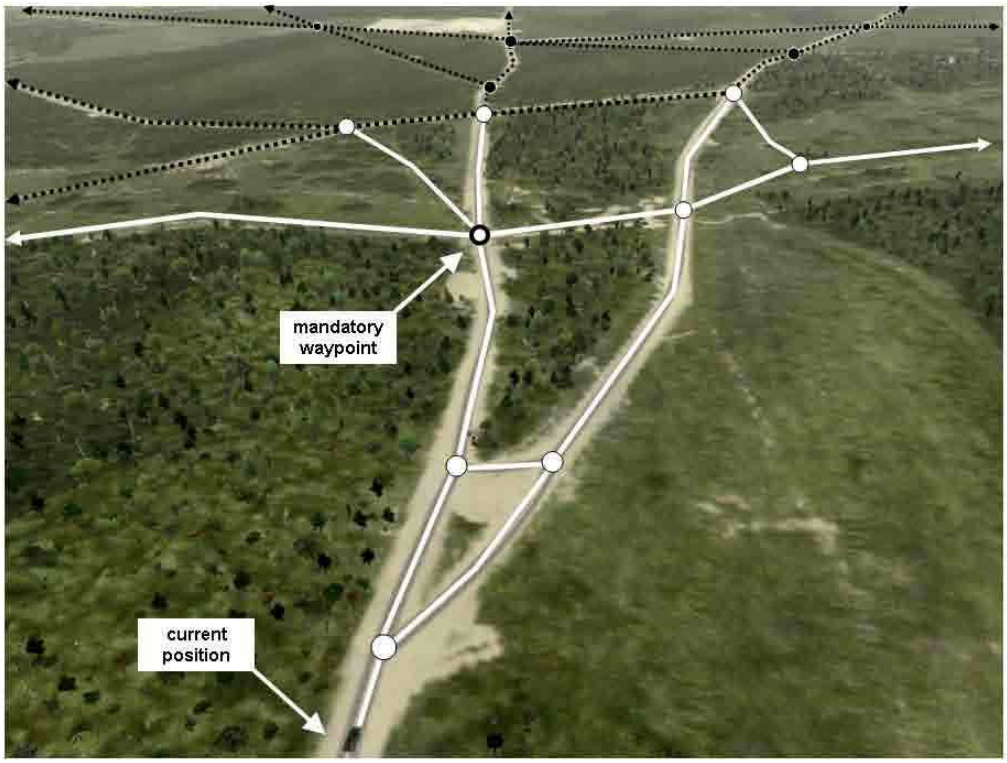


Fig. 1. Example of alternative paths for a ground vehicle horizon, and corresponding to the line of sight of a cooperating *Unmanned Aerial Vehicle* (UAV).

- *Constraint Satisfaction Problems* where the goal is to find correct assignments to problem variables satisfying a set of constraints.

When relaxing operational constraints, our vehicle navigation problem complexity is NP-hard in worst cases as it can be reduced in polynomial time to a TSP. The core difficulty is to find the optimal sequence of mandatory waypoints to visit with near real-time performances, despite a realistic problem size.

2.2. Example

As an illustrative example, let us consider the following situation inspired from a real case (figure 1). During mission execution, the planner has only a partial awareness of its environment. The knowledge horizon is determined by direct ground observation, but additional knowledge can also be provided by vehicle team, sensor networks or external surveillance systems. White dots and lines represent waypoints and feasible paths between waypoints respectively. Black dots and dashed lines represent waypoints and uncertain paths that are situated beyond the observability horizon. Arrows show lines that continue beyond the limits of the picture. Lastly, one or more waypoints may be imposed along the vehicle route.

2.3. Basic constraints

In our problem, we need to represent the ground topography of the vehicle area of interest. A graph model is used to represent the space of possible paths, where vertices are geographical points (for example crossroads) and edges are progression axes (for example a road or a meadow) that can be taken by vehicles. They are determined before the beginning of the mission using roadmaps, digital terrain models and UAV or satellite observations.

Formally, the graph G is defined as a couple $G = (V, E)$ where V is the set of vertices and E is the set of edges. V_m is a subset of V that represents the set of mandatory waypoints. A valid path starts from vertex v_{start} and reaches vertex v_{end} after having passed through all mandatory waypoints. A set Φ of variables $\varphi_e \in \{0, 1\}$ is defined, where each variable is associated to an edge e in order to model a possible path from v_{start} to v_{end}. An edge e does not belong to a feasible path when $\varphi_e = 0$. This is formulated as the following constraint, where $\omega(v)$ represents the set of incoming and outgoing edges of vertex v (incoming for ω^-, outgoing for ω^+). The graph is undirected, so that $\omega^+(v) = \omega^-(v)$ for each $v \in V$:

$$\sum_{e \in \omega^+(v_{start})} \varphi_e = 1, \quad \sum_{e \in \omega^-(v_{end})} \varphi_e = 1, \tag{1}$$

$$\forall v \in V \setminus \{v_{start}, v_{end}\}, \sum_{e \in \omega^+(v)} \varphi_e = \sum_{e \in \omega^-(v)} \varphi_e \leq 1 \tag{2}$$

Nodes v_{start} and v_{end} represent current position and primary objective for the vehicle respectively. Equation (2) ensures path connectivity and unicity while equation (1) imposes limit conditions on path start and end. These constraints give a linear chain alternating positions and mobility actions along the graph.

2.4. Capability metrics

Assuming a given date D_v associated with a position v, we are using a path length formulation (3) often considered in Operational Research (OR) [1]. Variable D_v expresses the time at which the vehicle reaches position v (see example in figure 2).

Assuming that constants $d_{(v,v')}$ represent the time taken to perform a movement from location v to v', we have:

$$\forall v \in V, \quad D_{v'} = \sum_{(v,v') \in \omega^-(v')} \varphi_{(v,v')}(D_v + d_{(v,v')}) \tag{3}$$

Constants $d_{(v,v')}$ are critical decision variables in the problem and make constraints (3) non linear. Finally, the mission schedule can be represented as $\Delta = \{(v, D_v) | \ v \in V, \ D_v > 0\}$. An equivalent constraint-based formulation is also used for other mission metrics (figure 2), such as energy or capacity.

2.5. The challenges

In traditional architecture design, the notion of *fast reactivity* is itemized as a system requirement. However, as described above, our problem is TSP-like and consequently of NP complexity. To keep tractability, the number of mandatory waypoints must remain realistic: on mission, rarely more than a ten of waypoints are imposed. In addition, in our approach, on-line planning is solved over a limited horizon from the current vehicle position. It corresponds to the terrain on which the vehicle vetronic has enough detailed information to characterise its trafficability. Therefore, the number of mandatory waypoints is relatively small. They correspond to short-term objectives or narrow manoeuvres executed by the vehicle.

Note that if a plan cannot be quickly solved, the vehicle may stop if this is possible. It is an ultimate solution that is not satisfactory from an operational point of view. Finally, due to vetronic processing resources, computation load, memory usage and response time must be reduced as much as possible.

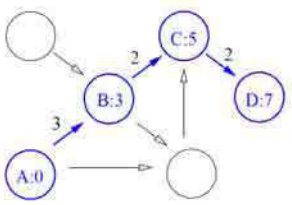
In the following, we present how a new hybrid approach, that mixes a complete method with a metaheuristic pre-processing mechanism, delivers compliant performances over real-case scenarios.

3. State of the Art

3.1. Mission Planners

3.1.1. Generic planners

Much research has been carried out on generic planning problems, often motivated by mission preparation in defense area. Domain-independent planners [2] and formalisms such as PDDL 2.0 [3] have emerged to tackle these complex problems. The related search methods can be complemented by pre-processing or dedicated heuristics to fit specific domain problems. However, these approaches may not match on-line requirements of reactive embedded systems.



Graph in 2.4 is a spatial representation of possible moves (edges) and positions (nodes). Moves, that correspond to the set of positive values $\Phi = \{(A, B), (B, C), (C, D)\}$, are represented with bold arrows. We are assuming an edge distance metric for optimization. Other operational constraints, such as protection, vulnerabiliy, available energy and security are similarly formulated in different experiments.

Fig. 2. Illustrating a path with pass-by dates over a graph of locations and progression axes.

3.1.2. Domain-dependent planners

Much research has been done for both military or civilian purposes, relying on specific planning frameworks such as Hierarchical Task Network (HTN)[4]. Some specific planning techniques fulfill on-line requirements, such as in [5], but may not encompass the spectrum of operational constraints.

3.1.3. CP planners

Constraint solving in planning has been integrated into various frameworks: this is the case of Ix-TeT [6] and HSTS [7]. In Reactive Model-based Programming Language (RMPL) [8], an evolution of Concurrent Constraint (CC) languages, the same paradigm is used to dynamically constrain planning representations of one or more remote agents.

3.2. Operational Research

Our problem can be considered using OR algorithms, based on flow models. As explained in section 2.1, it can be relaxed as a TSP, for which numerous algorithms have been proposed. Firstly, it can be tackled with a deterministic approach (see [1] for example). Local search methods have also been widely used for TSP problems [9]. To solve larger TSP problem instances, where deterministic approaches may take exponential time to solve, new metaheuristics have been developed. The most renown algorithms are Simulated Annealing [10], Genetic Algorithms [11] and Ant Colony Optimization (ACO) [12], for which various versions exist.

Most of CP frameworks are useful to design hybrid search techniques, by integrating OR and Linear Programming algorithms [13]. However, only a few ones such as [14], have explored on-line planning requirements.

3.3. Optimal path planning

A large number of heuristic-based search methods have been developed, mainly on the basis of the well-known A* [15]. Research has been done on *memory size limitation* [16], often on the principle of iterative-deepening searches [17]. Memory saving conditions are another family [18] that allow deleting the less interesting evaluated states. Others are dealing with the problem of *anytime solution availability* [19] to face situations with uncertain execution time. Finally, much work has been done to tackle *dynamic and uncertain environments*. Two families have emerged : incremental heuristic searches and real-time heuristic searches. Their properties are significantly different. Incremental searches consider the whole environment, and are optimal: to be efficient, they reuse information from previous searches. If a contingency occurs, propagation methods allow to update information and to prevent from reconsidering the entire problem. The most optimized incremental search algorithm is currently Dynamic FSA* [20]. On the other hand, real-time heuristic searches use a different approach by considering only a local portion of the environment. They are consequently suboptimal but very efficient on highly dynamic problem instances. Most recent works are hierarchical methods [21].

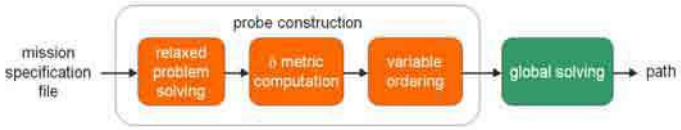


Fig. 3. Diagram of the complete solver using probing techniques.

4. Proposed Method

4.1. Global Solving

The global solving techniques use a Constraint Programming (CP) framework which combines a high level of expressiveness and powerful constraint solving techniques. The $\{0, 1\}$ flow problem (see formulas (1) and (2)) is expressed with variables and arithmetical constraints. The formulation can support multiple distance metrics over paths, even non-linear ones (formula (3)). This approach is very interesting to represent tactical mobility (positions, progression axes, objectives, visibility...), and vehicle abilities (fuel, trafficability...). Problem formulation and global solving method have been implemented with the $CLP(FD)$ domain of SICStus Prolog library. It uses state-of-the-art in discrete constrained optimization techniques:

- Arc Consistency-5 (AC-5) [22] for constraint propagation, that is managed by $CLP(FD)$ predicates. When a variable domain get reduced, AC propagates domain variables until a fixed point is reached.
- Variable filtering with correct values, using specific labelling predicates to instantiate problem domain variables. AC being incomplete, value filtering guarantees the search completeness.
- Tree search with standard backtracking when variable instantiation fails.
- Branch and Bound (B&B) for cost optimization, using *minimize* predicate.

The global solving techniques under consideration guarantee search completeness, solution optimality and proof of optimality. Designing a good solving method consists in finding the right variables ordering and values filtering, using domain or generic heuristic, and in general implemented with some specific labelling predicates.

Other possibilities exist to reinforce global solving. Arc consistency can integrate domain-specific consistency rules, while global optimization can be improved in many ways (generating bounds, branch and cut, branch and price, iterative deepening...). These techniques are not in the scope of the probing hybridization described in this paper.

4.2. The Probing Method

4.2.1. Overview

The goal of hybridizing global solving with stochastic approaches is to save the number of backtracks and to quickly focus the search towards good solutions. It consists in designing the tree search according to problem structure, revealed by the probe.

The idea is to use the prober to statically order problem variables, as a pre-processing. Instead of dynamic probing with tentative values such as in [23], this search strategy uses a static prober which orders problem variables to explore according to the relaxed solution properties. Then, the solving follows a standard CP search strategy, combining variable filtering, AC-5 and B&B.

As shown in figure 3, the probing technique proceeds in three steps (the three blocks on the left). The first one is to establish the solution to the relaxed problem. As a reference, we can for example compute the shortest path between starting and ending vertices, abstracting away mandatory waypoints. The next step is to establish a minimal distance $\delta(v)$ between any problem variable and the solution to the relaxed problem. This step can be formally described as follows. Let $V_s \subset V$ be the set of vertices that belong to the relaxed solution. The distance is given by the following evaluation:

$$\forall v \in V, \delta(v) = \min_{v' \in V_s}(min\_distance(v, v')) \quad (4)$$

where the distance is the number of vertices between v and v'. The last step uses the resulting partial order to sort problem variables in ascending order. At the global solving level the relaxed solution is useless, but problem variables are explored following this order.

4.2.2. Interests

Two interesting probe properties can be highlighted:

- *probe complexity*: since computation of minimum distance is polynomial between a vertex and any node is polynomial thanks to Dijsktra or Bellman-Ford algorithms, the resulting probe construction complexity is still polynomial in worst cases. The complexity of quicksort can in practice be neglected (see below for further details).
- *probe completeness*: since the probe does not remove any value from variable domains and the set of problem variables remains unchanged, the probe still guarantees global solving completeness.

Complexity analysis. Let γ be the cardinality of V_s and n the one of V. The complexity of probe construction is:

- worst case performance: $O(n^2)$;
- average case performance: $O(\gamma.n.\log(n))$.

Sketch of the proof. The probing method first determines the minimal distance between all vertices $v' \in V'$ where $V' = V \setminus V_s$ and any vertex $v_s \in V_s$. A Dijkstra algorithm run over a vertex v_s allows to compute the distance to any point of V' with $O(n.\log(n))$ worst case complexity where n is the number of nodes in V. This has to be run over each vertex of V_s and a comparison with previous computed values must be done for every vertex v', to keep the lowest one. Thus, the resulting complexity is $O(\gamma.n.\log(n))$. Variables must finally be sorted with a quicksort-like algorithm. The worst case complexity of this sort is $O(n^2)$ but is generally computed in $O(n.\log(n))$

(average case performance). Hence, the worst case complexity of the probing method is $O(n^2)$, but in practice behaves in $\max\{O(\gamma.n.\log(n)), O(n.\log(n))\} = O(\gamma.n.\log(n))$.

4.2.3. Pseudocode

Algorithm 1 synthesizes probe construction mechanisms. Firstly, a vector L_d of size n (n being the number of nodes in V) is created and initialized with infinite values. At the end of the execution, it will contain a value associated to each vertex, corresponding to the minimal distance between this vertex and the solution to the relaxed problem. To do so, a Dijkstra algorithm is run over each node of the solution. During a run, distances are evaluated and replaced in L_d if lower than the existing value (in the pseudocode, comparisons are made at the end of a run for easier explanation). Once minimal distances are all computed, they are used to rank the set of vertices V in ascending order (to be used by the complete solver).

Algorithm 1 Probe construction

```
1: Initialize a vector L_d of distances (with infinite values)
2: Get P the best solution of the relaxed problem
3: for each node v_i of P do
4:    L'_d ⇐ Run Dijkstra algorithm from v_i
5:    L_d = min(L_d, L'_d) (value by value)
6: end for
7: Sort V using L_d order
8: return the newly-ordered V list
```

4.3. A Stochastic Relaxed Problem Solver

Instead of considering a blind shortest path to solve the relaxed problem, the proposed algorithm implements an Ant Colony Optimization (ACO) search [12] that uses a similar model of the environment.

4.3.1. The Ant Colony algorithm

ACO belongs to the family of swarm intelligence metaheuristics. It has been initially developed to solve TSP instances, and is more generally well defined for discrete and possibly dynamic problems. It spreads a population of ants through the state space and iteratively reuses collective memory to improve the search. Similarly to the notion of generations in Genetic Algorithms, ACO deploys a series of search cycles. During a cycle, each ant builds a solution thanks to a probability law using both a guiding heuristic and the collective memory information. The latter is defined as an edge weight that varies over time and represents the *pheromone rate*. In nature, ants disseminate this chemical substance to remind the path. The shorter the path, the sooner the path will be taken by other ants and consequently the higher the pheromone rate will be. In the ACO algorithm, the pheromone model is updated at the end of a cycle : solutions are compared and best ones are used to improve collective memory by reinforcing related edges.

Formally, an ant builds a path through the state space by electing iteratively the next vertex to take to reach the goal. To do so, it uses the following formula. For an ant k currently at vertex v, the probability for choosing a reachable vertex v' as its next waypoint is given by formula (5).

$$\forall (v,v') \in E, v' \in \omega^+(v), P_{v'}(k) = \frac{\tau^{\alpha}_{v,v'} \eta^{\beta}_{v'}}{\sum_{v'' \in \omega^+(v)} \tau^{\alpha}_{v,v''} \eta^{\beta}_{v''}} \quad (5)$$

$P_{(v,v')}$ is a probability and thus belongs to $[0, 1]$. The $\eta_{v'}$ parameter is the guiding heuristic, which is described below. The $\tau_{(v,v')}$ parameter is the pheromone edge weight to go from vertex v to vertex v'. It represents the experience acquired during previous search cycles and tends to choose edges that belong to known good solutions. α and β are calibration variables that balance the importance given to τ and η parameters. It has an impact on algorithm convergence, as a search with a strong β value will be very orientated but may not allow a correct space exploration and thus an unexpected better solution discovery. The pheromone model update is made using the following formula:

$$\forall (v,v') \in E, \tau_{v,v'}(c+1) = \rho.\tau_{v,v'}(c) + \Delta\tau_{v,v'} \quad (6)$$

where ρ is the conservation factor ($1 - \rho$ corresponds to the pheromone *evaporation* factor, in analogy with real ants). It allows decreasing pheromone weight over time to remove attraction from bad paths. $\Delta\tau_{v,v'}$ equals to:

$$\Delta\tau_{v,v'} = \begin{cases} \frac{1}{L_{LB}} & \text{if } (v,v') \text{ belongs to the local best solution} \\ 0 & \text{otherwise.} \end{cases} \quad (7)$$

where L_{LB} is the length of the local best solution. Some improvements of the initial ACO algorithm can be made using [24, 25] but they are not discussed in this paper.

4.3.2. Adaptation to the problem

In the original TSP implementation of ACO, initial ant positions are randomly defined. The guiding heuristic $\eta_{v'}$ returns the distance inverse from the current point to v': the closer this point, the higher its probability to be selected.

In our implementation, ACO is not only used to solve the TSP-like problem (which is the sequence of mandatory points). It is also used to find the shortest path between each mandatory point. Consequently, the original guiding heuristic definition is not satisfactory as the shortest path between two points may be a long but *straightforward* edge instead of several small but derivating ones. That's why a new definition has been given to $\eta_{v'}$: it is henceforth the distance inverse to go from the candidate vertex v' to the goal. Thus, the algorithm will tend to choose the vertex that is the closest to the goal. In addition, all searches are starting either from the current point or from the final objective (the path is not a loop anymore).

The guiding heuristic orientates the search towards the final goal, but mandatory waypoints may not be aligned with the start-goal axis. Thus the probability that a path contains all the mandatory waypoints is very low, and the search will have a poor success rate. To counter this problem, intermediate goals are iteratively elected during an ant search. Currently at point v_m, the election probability of a v'_m point is made using formula (8).

$$\forall v'_m \in V_m, P_{v'_m} = \frac{\eta_{v_m,v'_m} + D_{v'_m}}{\sum_{v''_m \in V_m} \eta_{v_m,v''_m} + D_{v''_m}} \quad (8)$$

η_{v_m,v'_m} is the distance inverse to go from v_m to v'_m and tends to choose the closest point as the next intermediate goal. $D_{v'_m}$ is the distance proportional function to go from v'_m to the goal and tends to avoid from keeping isolated points along the search.

4.3.3. Pseudocode of the algorithm

Algorithm 2 ACO algorithm

```
1: Initialize pheromone model
2: Set global best path P_GB to null
3: for c ⇐ 1 to C do
4:    for each ant of the population do
5:       Run single ant search
6:    end for
7:    Get local best solution P_LB over all searches
8:    if P_LB ≤ P_GB then
9:       P_GB ⇐ P_LB
10:   end if
11:   Update pheromone model
12: end for
13: return the best solution P_GB
```

Algorithm 3 Run single ant search

```
1: Initialise path P ⇐ {v_start}
2: Define the current position v_current ⇐ v_start
3: Define M the set of mandatory waypoints
4: while M ≠ ∅ do
5:    Elect an intermediate goal v_goal from M
6:    Remove v_goal from M
7:    Reach objective v_goal
8: end while
9: Reach objective v_end
10: return P
```

Algorithm 4 Reach objective v_{obj}

```
1: while v_current ≠ v_obj do
2:    Elect next waypoint v_next
3:    Verify v_next validity
4:    Add v_next to P
5:    v_current ⇐ v_next
6: end while
7: return
```

Algorithm 5 Elect next waypoint

```
1: Get the set V' of v_current neighbors
2: Compute probability for each v' ∈ V'
3: Elect the next waypoint v_next
4: return The next waypoint v_next
```

Algorithm 2 is the main instance, and returns the best path found. The input parameters are the graph G, the list of mandatory waypoints M, and the starting and ending nodes. At first, the pheromone model P is created and initialized with constant values (that is, every edge is given a same weight). Then a loop is used to execute the C search cycles. In this loop, a vector of length N is created (N being the number of ants), corresponding to the list of potential solutions built by ants. This vector is filled iteratively using a second loop that each time runs a single search. Once all searches are achieved, the best local solution P_{LB} (i.e. the shortest one) is found, then compared to the current best solution P_{GB} and saved if better. And before a new series of searches is done, P is updated using the most interesting solutions locally found. The pheromone model update procedure is not detailed here but follows from formula (6) rules.

Algorithm 3 builds a single path and returns it (or may fail) as a candidate solution. A path is a list of vertices that can be reached two-two (it only contains the starting point at initialization). To ensure passing through all intermediate objectives, one is picked from M using formula 8 and a single search (on line 7) is launched to reach it. As long as the M list is not empty, a new intermediate goal is elected and the path is completed. Finally, a search is conducted to reach the final goal.

Algorithm 4 is a search that builds a path from the current point to a specified objective. To do so, the "ant" repeatedly chooses the next waypoint to pass through (on line 2). As the path may have taken a vertex that belongs to the M list, a check is proceeded. Consequently, if there are x mandatory waypoints in M, at most $x + 1$ single searches will be performed. Note that a search may fail if the ending node is expanded before all mandatory waypoints have been reached.

Algorithm 5 details operations done to choose a next waypoint. Firstly, the list of neighbors (accessible nodes) is built. Then, respective probabilities to be elected are computed using formula (5).

In the current implementation, C and N are constants and defined statically in a configuration file.

5. Results

This section focuses on the comparison of our hybrid approach with two other methods. The first one is only based on the complete solver (presented in section 4.1) without probing. We call it the *reference* algorithm in the following. The second method is the complete solver coupled with the probing mechanism that uses a basic shortest path algorithm (described in section 4.2). We simply call it the *shortest path* algorithm in the following.

5.1. Benchmarks

To make our comparison, the three methods are tested over three distinct benchmarks that correspond to real-world scenarios. They are representative of vehicle planning for modern peace-keeping missions, both in urban and open environments. The figure 5.1 gives an idea of problem complexity.

For each benchmark, five series of executions are done: between two series, a new mandatory waypoint is added (there are consequently 1 to 5 waypoints per series). For each series, ten distinct runs are led (distinct means that the couple $(start, end)$ changes).

5.2. ACO calibration

5.2.1. Discussion

There is no universal rule to parameterize the ACO algorithm. It depends on the problem, essentially in terms of graph size and

	Bench1	Bench2	Bench3
Environment	urban	urban	open
Vertices	23	22	22
Edges	76	74	68
Variables	723	654	702
Constraints	1944	1750	1886

Above: table of benchmark characteristics (the number of edges is considered over the directed graph).
On the right: graph of benchmark 2. Various mandatory waypoints (gray nodes) can be imposed. Black circles represent possible starting or ending nodes.

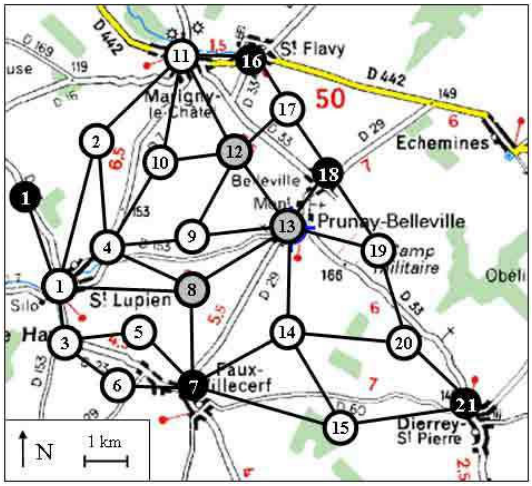


Fig. 4. Benchmark overview

connexity. The choice of α and β can be decisive if there is a high risk for the search to follow a dead-end path. In this case, by privileging the η term (thus by setting β higher than α), our research will have more chance to fail (or to find bad solutions) if it takes a path in direction to the objective but which does not lead to this one. An analogy can be made with A-star algorithm, whose worse case is a labyrinth in which the only way to reach the objective is opposite to the location's direction. In the other cases, it can be judicious to take more consideration to η that can fastly lead to good solutions. The N parameter depends on the size of the problem. The larger the problem, the more the ant population (represented by N) should be important, as the number of possible solutions becomes high. The C parameter (number of search cycles) more particularly depends on the learning mechanisms. As we know, the quality of the research is a compromise between state space exploration and convergence speed. According to the chosen strategy, C should be low if a fast convergence is wished (the reinforcing edge parameters should then evolve fastly), or high in the contrary (and reinforcing should be mild to maintain alternative paths).

5.2.2. Parameter values

For this test, we calibrated the ACO algorithm with the following parameters:

- $C = 10$;
- $N = 10$;
- $\alpha = 1$;
- $\beta = 5$;
- $\tau_{init} = 0.5$;
- $\rho = 0.9$.

These values were set empirically after comparing performances on several tries.

5.3. Comparison criteria

The performance of a method is evaluated depending on several criteria:

- the time to find the optimal solution;
- the time to prove the optimality;
- the number of backtracks done by over the branch & bound method;
- the memory space required.

Experiments have been run on a dual core CPU working at 2.53 GHz with 2 GB of memory. The results are presented below.

5.4. Results

5.4.1. Execution time

During the runs, execution time was limited to ten seconds. It corresponds to the upper limit of satisfiability: over this bound, it is considered as unacceptable. In the following, a "X" time value indicates a computation overrun.

The table in figure 5 brings the results in terms of execution time and backtracking. In the columns, the three algorithms are compared : REF is the reference algorithm, $SPATH$ is the shortest-path-based algorithm, and $ANTS$ is the ACO-based algorithm. For each method, the table presents:

- (a): the time needed to find the optimal solution respectively;
- (b): the time needed to prove optimality;
- (c): the number of backtracks needed to find the optimal solution.

For each, three values are given:

- MIN: the minimal value over the ten runs;
- MAX: the maximal value over the ten runs;
- $TOTAL$: the total value over the ten runs (i.e. the sum of the value for each run).

If a series of runs contains at least one value that exceeds the ten seconds limit (i.e. is marked with a "X"), then the $TOTAL$ value must not be considered as exact. It should be greater in real case because a run that failed finding a solution within the bound is saved as an execution time of ten seconds (it would normally be more). A series of runs is represented as a row. Far left is the number of the benchmark. Each benchmark is evaluated using

			REF			SPATH			ANTS		
			(a)	(b)	(c)	(a)	(b)	(c)	(a)	(b)	(c)
BENCH1	1MW	MIN	16	2984	2	0	297	0	31	140	0
		MAX	7656	x	23594	422	1750	719	375	3360	659
		TOTAL	28172	59716	91319	1139	8828	1716	938	11640	659
	2MW	MIN	0	1453	0	32	328	20	62	250	0
		MAX	5906	x	24320	453	2156	919	188	1734	180
		TOTAL	16045	48016	52761	1500	10843	2478	1048	7719	454
	3MW	MIN	2547	7109	9363	0	359	1	63	532	0
		MAX	x	x	30860	187	2313	6271	141	2969	72
		TOTAL	71704	93687	208666	702	11719	6963	863	18641	82
	4MW	MIN	1312	1890	3518	0	282	1	63	329	0
		MAX	x	x	20204	141	1703	195	407	1781	2103
		TOTAL	46875	61485	107614	845	8577	964	1312	12781	2841
	5MW	MIN	3578	5813	9411	16	594	1	78	703	0
		MAX	9454	x	24691	359	1672	6969	219	2078	225
		TOTAL	56003	84313	153560	1436	11343	9135	1404	14421	939
	ALL	MIN	0	1453	0	0	282	0	31	140	0
		MAX	x	x	30860	453	2313	6969	407	3360	2103
		TOTAL	218799	347217	613920	5622	51310	21256	5565	65202	4975
BENCH2	1MW	MIN	0	735	0	0	125	0	31	156	0
		MAX	7297	9797	17107	297	625	527	79	641	0
		TOTAL	13859	27064	36000	685	3484	1081	580	4157	0
	2MW	MIN	0	750	0	16	156	1	32	156	0
		MAX	8609	x	19429	375	796	745	359	1218	531
		TOTAL	11796	23872	30504	1138	3858	1835	1034	6078	705
	3MW	MIN	0	734	0	47	141	16	62	219	0
		MAX	4016	6063	8337	343	687	624	94	1063	46
		TOTAL	5406	17812	13385	1465	3891	2394	797	5486	48
	4MW	MIN	16	922	32	16	156	3	46	328	0
		MAX	5937	7453	13397	391	891	671	156	1172	54
		TOTAL	12279	26090	26936	736	4095	844	905	7030	101
	5MW	MIN	94	735	158	31	188	3	78	391	0
		MAX	3125	4312	6695	234	703	339	547	1906	786
		TOTAL	8313	19298	17677	985	4014	1077	1750	9047	1351
	ALL	MIN	0	734	0	0	125	0	31	156	0
		MAX	8609	x	19429	391	891	745	547	1906	786
		TOTAL	51653	114136	124502	5009	19342	7231	5066	31798	2205
BENCH3	1MW	MIN	15	453	0	15	344	0	47	406	0
		MAX	2204	2735	5070	94	1750	133	79	1750	12
		TOTAL	4907	11766	11067	250	8874	170	658	8767	14
	2MW	MIN	16	203	20	46	125	18	62	110	0
		MAX	594	984	1104	2734	3782	5403	640	3282	1127
		TOTAL	2612	6266	5209	7515	13173	15340	1935	16205	2266
	3MW	MIN	47	454	16	15	250	11	47	328	0
		MAX	719	1984	1649	1656	3329	3839	922	2797	1650
		TOTAL	2691	10533	5298	3015	12940	6178	1719	14470	1766
	4MW	MIN	31	375	41	15	218	4	93	547	2
		MAX	1281	1703	2694	1609	2390	3108	547	2469	797
		TOTAL	4719	8265	9383	4484	9483	8767	1806	14548	1749
	5MW	MIN	31	281	46	15	234	4	94	547	1
		MAX	984	1297	2092	860	1359	1535	984	2031	1822
		TOTAL	4265	7263	8628	2874	7109	5258	3125	13000	4945
	ALL	MIN	15	203	0	15	125	0	47	110	0
		MAX	2204	2735	5070	2734	3782	5403	984	3282	1822
		TOTAL	19194	54093	39585	18138	51579	35713	9243	66990	10740

Fig. 5. Benchmark results over a total of 450 runs.

	REF	SPATH	ANTS
BENCH1	284	162	**76**
BENCH2	215	167	**131**
BENCH3	236	320	**234**

Fig. 6. Maximum memory space required for a search (in MB).

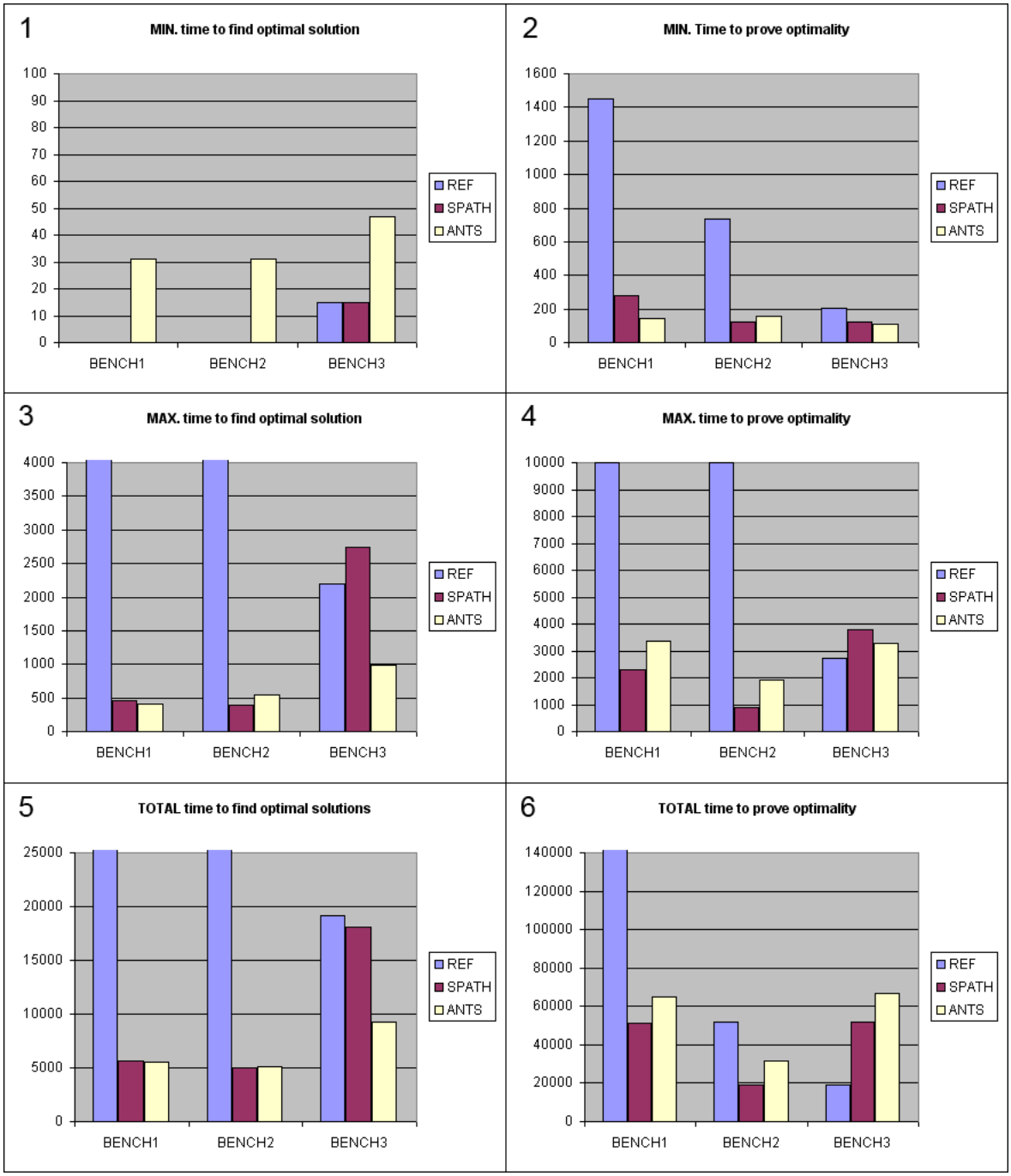


Fig. 7. Graphs illustrating main results from data table.

a different number of mandatory waypoints (MW), from one to five. The ALL rows are a synthesis of the five series over a same scenario (MIN, MAX and $TOTAL$ are consequently evaluated over the fifty runs).

Lastly, execution time is expressed in milliseconds. It is important to notice that these results were obtained using Prolog, that uses operating system function calls. Consequently, time values have a precision (or ”delta”) range of twenty to thirty millisec-

onds.

5.4.2. Memory consumption

Table 6 summarizes the maximum space required over all runs of each algorithm, for the three benchmarks. As memory is freed between two searches, it is unnecessary to present the memory amount needed for each run. Units are in megabytes, and values are rounded to the nearest greater integer (ceil). Bold values are lowest values over the three algorithms.

5.5. Analysis

We do not focus on the number of backtracks, as the results are strongly correlated with solving time. Here we analyze the results in terms of execution time, whose values are identified by, (*a*) and (*b*) in the table of figure 5. To have a clearer view of these results, the figure 7 shows a series of graphs illustrating data from *ALL* rows of the table in figure 5.

Graph 1 shows the minimal execution time needed to find the optimal solution. As we can see, REF and $SPATH$ may find optimal solutions very fastly, whereas $ANTS$ takes more time. It corresponds to the *overhead*, that is the execution time needed to run ACO searches. Graphs 3 and 5 show the maximal and total execution times needed to find the optimal solution. The latter allows to get an average execution time information (by dividing the total value by fifty, which is the number of runs considered). The first finding is that probing methods are very efficient on benchmarks 1 and 2, whereas the reference method (without probing) is not. The latter even exceeds several times the ten seconds execution time limit, that means it did not find the best solution within this bound. The second finding is that $SPATH$ and $ANTS$ methods are almost equally effective over the benchmarks 1 and 2. However, one can see that $SPATH$ is very inefficient on benchmark 3. It even has lower efficiency than the reference method. It is due to the fact that mandatory waypoint locations were quite distant from the start-goal axis. $ANTS$ is approximately two times faster than $SPATH$ on this benchmark. As the overhead of $ANTS$ is around 70 ms (3500 ms over 50 runs), the investment is about 5% of total $SPATH$ computation time and the gain is about 50%. As a first conclusion, over these three benchmarks, the ACO-based method is proven to be more efficient as it finds the best solution faster than the two other methods. It is more stable as it adapts to various problems without excessive performance variations.

Graphs 2, 4 and 6 respectively show minimal, maximal and total execution time needed to find the best solution and to prove its optimality. Surprisingly, results are quite different from previously. REF algorithm is still very inefficient over benchmark 1. Over benchmark 2, REF is not so inefficient as the total (and consequently the average) execution time remains correct. In fact, the problem is due to the failure of a search in the series with 2 mandatory waypoints (see in table 5). The reference algorithm even reveals to be very efficient in benchmark 3 and outperforms both $SPATH$ and $ANTS$! In addition, $SPATH$ is more efficient than $ANTS$: in terms of total computation time, it spends 39% less time over benchmark 2, and 21% less time over benchmarks 1 and 3. This has to be tempered by the fact that $ANTS$ needs more precomputation time, which leads results of the same order. But the conclusion is that $ANTS$ does not allow to prove the solution faster.

In terms of memory consumption (see table of figure 6), $ANTS$ algorithm reveals to have the best performances over the three approaches, for each benchmark.

6. Conclusion

We presented a hybrid approach that mixes an exact solving method guided by a metaheuristic. The latter uses a stochastic ACO algorithm to solve a relaxed version of the problem to order the variables. This pre-processing step allows the backtracking method of the complete solver to select the most promising variables first. We focused our attention on the adaptation of ACO to this problem and compared it to a deterministic implementation on small problem instances.

Through three realistic series of benchmarks, we showed that the ACO-based approach is as fast as using a brute-force shortest path on easy problems, despite the computation overhead. On most complex cases (when mandatory waypoints are distant from the start-end axis), we recorded a gain of 50% on average computation time, for 5% of extra pre-processing time. Our approach is consequently more stable and could easily solve bigger problem instances while it would rapidly become intractable for the deterministic approach. The ACO-based algorithm also revealed better memory consumption performances, which is very interesting in terms of on-line applications. Additionally, it could have interesting properties in terms of replanning capabilities. However, we also found that the method does not allow to prove optimality faster than the deterministic method.

References

[1] M. Gondran and M. Minoux, "Graphes et algorithmes," 1995.

[2] M. Fox and D. Long, "Automatic synthesis and use of generic types in planning," *Proceedings of the Artificial Intelligence Planning System*, pp. 196–205, 2000.

[3] M. Fox and D. Long, "Pddl2.1: An extension to pddl for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, 2003.

[4] D. M. R. Goldman, K. Haigh and M. Pelican, "Macbeth: A multi-agent constraint-based planner," *Proceedings of the 21st Digital Avionics Systems Conference*.

[5] D. S. N. Meuleau, C. Plaunt and T. Smith, "Emergency landing planning for damaged aircraft," *Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference*, pp. 3247–3259, 2009.

[6] P. Laborie and M. Ghallab, "Planning with sharable resource constraints," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1995.

[7] N. Muscettola, *Hsts: Integrating planning and scheduling*. Robotics Institute, Carnegie Mellon University, 1993.

[8] P. K. M. Abramson and B. Williams, "Executing reactive, model-based programs through graph-based temporal planning," *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 2001.

[9] E. Aarts and J. Lenstra, *Local Search in Combinatorial Optimization*. 1997.

[10] V. Cerny, "A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, pp. 41–51, 1985.

[11] D. Goldberg, "Genetic algorithms in search, optimization and machine learning," 1989.
[12] M. Dorigo and L. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, pp. 53–66, 1997.
[13] F. Ajili and M. Wallace, *Constraint and integer programming: Toward a unified methodology*. McGraw Hill, Kluwer Academic Publishers, 2003.
[14] R. K. G. R. S. Chien, B. Engelhardt and R. Sherwood, "Casper: Space exploration through continuous planning," *Journal of IEEE Intelligent Systems*, vol. 16, 2001.
[15] N. N. P. Hart and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems, Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.
[16] A. Reinefeld and T. Marsland, "Enhanced iterative-deepening search," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 16, no. 7, pp. 701–710, 1994.
[17] R. Korf, "Depth-first iterative-deepening: An optimal admissible tree search," *Artificial Intelligence*, vol. 27, pp. 97–109, 1985.
[18] S. Russel, "Efficient memory-bounded search methods," *Proceedings of the Tenth European Conference on Artificial Intelligence (ECAI)*, pp. 1–5, 1992.
[19] E. Hansen and R. Zhou, "Anytime heuristic search," *Journal of Artificial Intelligence Research*, vol. 28, pp. 267–297, 2007.
[20] X. S. S. Koenig and W. Yeoh, "Dynamic fringe-saving a*," *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 2, pp. 891–898, 2009.
[21] M. M. A. Botea and J. Schaeffer, "Near optimal hierarchical pathfinding," *Journal of Game Development*, vol. 1, no. 1, 2004.
[22] Y. Deville and P. V. Hentenryck, "An efficient arc consistency algorithm for a class of csp problems," in *Proceedings of the 12th International Joint Conference on Artificial intelligence (IJCAI)*, vol. 1, pp. 325–330, 1991.
[23] H. E. Sakkout and M. Wallace, "Probe backtrack search for minimal perturbations in dynamic scheduling," *Constraints Journal*, vol. 5, no. 4, pp. 359–388, 2000.
[24] T. White, S. Kaegi, and T. Oda, "Revisiting elitism in ant colony optimization," in *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO)*, vol. 2723/2003, p. 199, Springer Berlin / Heidelberg, 2003.
[25] B. Bullnheimer, R. Hartl, and C. Strauß, "A new rank based version of the ant system - a computational study," *Central European Journal for Operations Research and Economics*, vol. 7, no. 1, pp. 25–38, 1997.

François Lucas was born in France in 1985. He received his Master's degree from ESIEE Paris in 2008, with a major in Real-Time Embedded Systems His internships led him to adapt a ground robot into fully-automated driven system (*Sagem Defense & Security*, 2008) and to design an environment simulator aiming at validating various CBTC equipments separately (*Thales Rail Signaling Solution*, 2009). Since December 2008 he is a PhD student in Mines ParisTech. He is working in collaboration with Sagem on efficient solving strategies for constrained path planning to be embedded in military troop transport vehicles. His main research interests are new hybridization techniques of metaheuristics and complete constraint solvers.

Christophe Guettier had his PhD in 1997 from "Ecole des Mines de Paris" in the area of combinatorial optimisation applied to the engineering of large-scale parallel systems. Then, he joined the SME "Axlog Ingénierie" where he developed some researches on autonomous systems for the European Space Agency (spacecraft constellation, deep space probes) and for Dassault Aviation (unmanned combat aerial vehicle). In 2001, he has been recruited by Xerox PARC to work on the DARPA NEST (Networking Embedded System Technology) project led by Air Force Research Lab and Boeing. In 2002, he joined Imperial College London where he developed a research axis on solving constrained routing problems for ad hoc networks. He joined SAGEM DS in 2004 where he leads several new product developments.

Patrick Siarry was born in France in 1952. He received the PhD degree from the University Paris 6, in 1986 and the Doctorate of Sciences (Habilitation) from the University Paris 11, in 1994. He was first involved in the development of analog and digital models of nuclear power plants at Electricité de France (E.D.F.). Since 1995 he is a full professor in automatics and informatics. His main research interests are the applications of new stochastic global optimization heuristics to various engineering fields. He is also interested in the fitting of process models to experimental data, the learning of fuzzy rule bases, and of neural networks.

Anne-Marie Milcent is an Engineer graduated from Ecole Centrale Paris, and has also a Master of Science from Standford University in Management Science and Engineering. She started as a leading-edge R&D engineer in Sagem Defense Security, working on helicopter and airborne observation equipments. During two years, she has been project leader in haptic man - machine interfaces, opening the way of new generation interfaces for soldier systems. Then, she became in charge of R&D group in future combat system programs where she has coordinated several robotic and soldier systems architecture projects. She recently joined the medical industry where she is now a R&D leader in the GUERBET group.

Arnaud de La Fortelle is civil servant at the French Transport Ministry and is both director of the Mines robotics lab (CAOR) and of the Joint Research Unit LaRA (La Route Automatisée - the automated road, an INRIA-Mines ParisTech Consortium). He managed for LaRA several French and European projects (Puvame, Prevent/Intersafe, REACT, COM2REACT...) and is currently coordinator of the European project GeoNet and of the French project AROS. He has a Ph.D. in Applied Mathematics and engineer degrees for the French Ecole Polytechnique and Ecole des Ponts et Chaussées.

Automatic Vehicle Navigation with Bandwidth Constraints

François Lucas and Christophe Guettier
Sagem Defense & Security
27, Rue Leblanc, 75012 Paris, France
Email: {francois.lucas,christophe.guettier}@sagem.com

***Abstract*—This paper introduces a new approach to manage network access in tactical environment for specific vehicles. These vehicles must ensure continuity of communications on the move with the command chain and with other units, like dismounted soldiers. Due to tactical mobility, path loss and versatile threats, vehicle navigation must be managed online according to communication connectivity, mission and operational constraints. The paper presents an on-line planning technique that optimizes navigation plans according to available communications. The navigation problem is modelled using constraint formulations to express both connectivity and operational requirements. Navigation plans are solved and optimized according to a combination of a metaheuristic and constraint solving techniques. Various forms of optimization strategies are compared and evaluated on realistic scenarios.**

I. Introduction

In modern Network Centric Warfare (NCW), land tactical units will require support from vehicles to deal with important tactical data flows, images and video. This is particularly the case for command and control, reconnaissance and support vehicles. These vehicles will need to exchange data whatever the tactical mobility and the environment constrained with mission objectives, threats and obstacles.

This paper addresses constrained navigation with objective waypoints and bandwidth requirement in a dynamic environment. We consider a class of vehicles able to exchange data on the move thanks to plural network access. It is given a final goal and a list of non-sequenced intermediate objectives to reach. In addition to terrain constraints, bandwidth requirements have to be met along the vehicle navigation plan. In general, network bandwidth allocation and navigation plans are considered separately and always defined at mission preparation time. Opposed to this static view, we consider dynamic handover of vehicle communications, coupled with on-line navigation planning. This is particularly interesting in order to provide planning alternatives when contingencies occur or communications are perturbated.

The goal is to provide to the user (vehicle, company commander, platoon leader ...) a decision support functionality by solving a Bandwidth-constrained Vehicle navigation Planning problem (BVP). The planner advises on-line the best route to follow under specified mission constraints, and according to available bandwidth on the different network accesses. System efficiency criteria are *solution optimality* and *reactivity*: it must give the best solution in execution times close to those of mission needs. This approach can also be applied to unmanned vehicles, where navigation plans must be updated whenever the mission changes, the environment evolves, or the network access becomes too limited. To solve the BVP problem, we propose a new hybrid approach mixing a complete constraint solving with a metaheuristic-based guiding method. Experiments made on representative problems (urban or open environment) show interesting performances.

The problem is described formally in section II. Section III summarizes recent state of the art in the different research fields. The proposed approach is detailed in section IV and results are analysed in section V. We discuss the approach in section VI before concluding in section VII.

II. Navigation and Bandwidth Management

A. Problem context

With new observation and communication capabilities, tactical units will have to deal with tactical data exchanges, but also images and video. Hence, vehicle systems will need to support more and more bandwidth demands. This is already the case with Joint Terminal Attack Controllers (JTAC) that receive video from Unmanned Aircraft Vehicles (UAV) or aircraft fighters. With additional tactical radios, the JTAC vehicle must manage several data links with differents systems. Other examples of vehicles are:

- Intelligence, Surveillance, Target Acquisition and Recconaissance (ISTAR) vehicles. Figure 1 shows such a vehicle experimented at Phoenix 2008 [1];
- control and command vehicles, able to perform tactical exchanges on the move.

These light vehicles operate and hide in dangerous environments, disrupting network connectivity. However, they can access different kinds of networks: fixed or deployable infrastructures as well as tactical mobile networks (see section II-F for further details). Handling communication perturbations may involve either vehicle movements or real-time communication handover on the different available networks.

B. Bandwidth-constrained Vehicle Planning

The Bandwidth-constrained Vehicle Planning (BVP) problem consists in finding a path from the current vehicle location to an objective waypoint. Intermediate mandatory waypoints can be imposed to fulfil secondary objectives. The goal is to

978-1-4244-8179-8/10/$26.00 ©2010 IEEE

Fig. 1. ISTAR vehicle and its operator, managing video feedback from a UAV and reporting environment activity in the battle management system.

find the lowest duration mission itinerary, while ensuring a minimum radio bandwidth.

As an illustrative example, let us consider the following situation inspired from a real case (fig. 2). From a current location S, a vehicle is given a goal G and a list of mandatory waypoints to reach (mission objectives, identified with gray nodes). The best path must pass through all the mandatory waypoints and reach the goal while minimizing mission duration and ensuring a minimum communication bandwidth. However, the two metrics can be competing : the bandwidth can decrease as a function of speed, depending on the radio model. Moreover, radio reception quality can vary according to the ground topography (in the picture: the finer the edges, the lower the quality).

As the timing and bandwidth metrics are not linearly dependent, the problem is known as NP-hard [2]. In addition, the problem of finding the best mandatory waypoint sequence (which is not defined) is also NP-hard, as it can be reduced into a Traveling Salesman Problem (TSP) instance. However, compared to TSP state of the art, this work focuses on small but composite problem instances where short response times are needed.

C. Required Input Data

In our problem, trafficability is modeled as a graph over the vehicle area of interest. Vertices are geographical points (for example crossroads) and edges are progression axes (for example a road or a meadow). Radio propagation is modeled as a weight over edges, representing path loss as a ratio over best possible reception (see subsection II-F for more details). A travel distance is associated to each graph edge, and can be given by terrain analysis.

These data are preprocessed at mission preparation time, but they can be updated during mission execution in order to solve the problem on-line when necessary. Therefore, solving duration is a critical requirement to embed the path planner, and to obtain near-real time behavior.

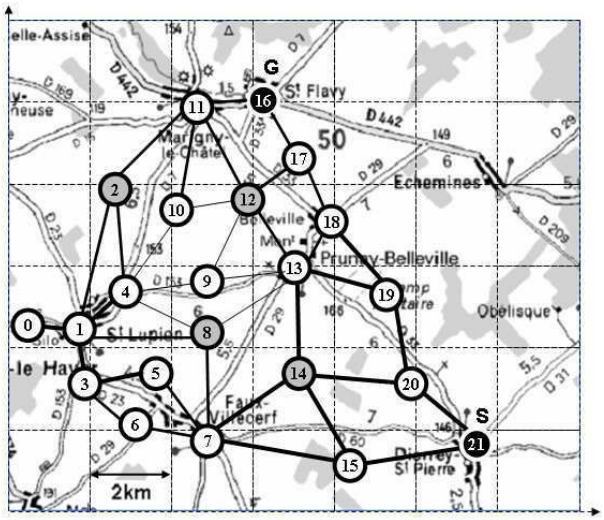


Fig. 2. Example of BVP problem with particularized bandwidth constraints on edges.

D. Basic constraint model

Formally, the trafficability graph G is defined as a couple $G = (V, E)$ where V is the set of vertices and E is the set of edges. V_m is a subset of V that represents the set of mandatory waypoints. A valid path starts from vertex v_{start} and reaches vertex v_{goal} after having passed through all mandatory waypoints. A set Φ of variables $\varphi_e \in \{0, 1\}$ is defined, where each variable is associated to an edge e in order to model a possible path from v_{start} to v_{goal}. An edge e does not belong to a feasible path when $\varphi_e = 0$. This is formulated as the following constraint, where $\omega(v)$ represents the set of incoming and outgoing edges of vertex v (incoming for ω^-, outgoing for ω^+). The graph is undirected, so that $\omega^+(v) = \omega^-(v)$ for each $v \in V$:

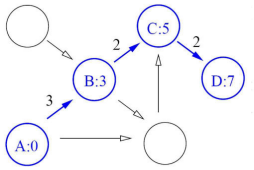
$$\sum_{e \in \omega^+(v_{start})} \varphi_e = 1, \quad \sum_{e \in \omega^-(v_{goal})} \varphi_e = 1, \quad (1)$$

$$\forall v \in V \setminus \{v_{start}, v_{goal}\}, \sum_{e \in \omega^+(v)} \varphi_e = \sum_{e \in \omega^-(v)} \varphi_e \leq 1 \quad (2)$$

Nodes v_{start} and v_{goal} represent current position and final objective for the vehicle respectively. Equation (2) ensures path connectivity and unicity while equation (1) imposes limit conditions on path start and end. These constraints give a linear chain alternating positions and mobility actions along the graph.

E. Timing metric

Assuming a given date D_v associated with a position v, we are using a path length formulation (3) often considered in Operational Research (OR) [3]. Variable D_v expresses the time at which the vehicle reaches position v (see example in figure 3). Assuming that constants $d_{(v,v')}$ represent the time taken to perform a movement from location v to v', we have:



Graph on the left is a spatial representation of possible moves (edges) and positions (nodes). Moves, that correspond to the set of positive values $\Phi = \{(A,B),(B,C),(C,D)\}$, are represented with bold arrows.

Fig. 3. Illustrating a path with pass-by dates over a graph of locations and progression axes.

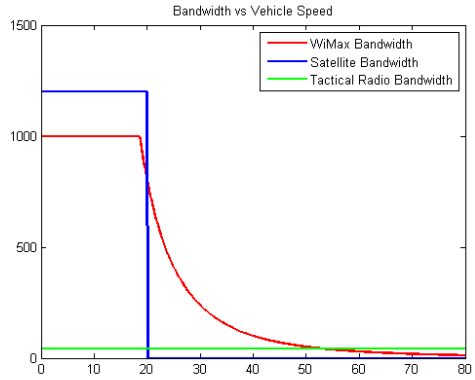


Fig. 4. Different types of radio communication system capabilities.

$$\forall v \in V, \quad D_{v'} = \sum_{(v,v') \in \omega^-(v')} \varphi_{(v,v')}(D_v + d_{(v,v')}) \qquad (3)$$

where terms $d_{(v,v')} = \frac{dist(v,v')}{speed(v,v')}$ and thus $D_{v'}$ are critical decision variables in the problem. This makes constraints (3) non linear. The mission schedule can be represented as $\Delta = \{(v, D_v) |\ v \in V,\ D_v > 0\}$.

As (3) is a cumulative function, the overall mission cost to minimize is simply $D = D_{v_{goal}}$.

F. Bandwidth metric

On mission, soldiers may have access to several radio communication systems. According to the equipment specification and use, bandwidth availability can differ (see the graphic on fig. 4):

- tactical mobile network systems that provide low but constantly available bandwidth, regardless of the speed (green curve);
- deployable infrastructure, such as *WiMax*-like standards that provide high but fastly decreasing bandwidth over speed (red curve);
- fixed infrastructure, such as satellite-based communication systems that provide high bandwidth with a very limited mobility (blue curve), due to tracking antenna.

In our model, bandwidth constraints are associated to edges. Considering that only one communication system is used at a time, the global bandwidth availability metric is a piecewise function. Assuming that $TacBw_e$,$WiMaxBw_e$ and $SatBw_e$ are tactical, *WiMax*-like and satellite available bandwidth for an edge e respectively, the global available bandwidth is defined as :

$$Bw_e = \varphi_e * \max\{TacBw_e, WiMaxBw_e, SatBw_e\} \qquad (4)$$

Bandwidths are defined as follows for tactical network, deployable and fixed infrastructure, respectively with equations (5, 6 and 7):

$$TacBw_e = C_{1e} * MaxTacBw \qquad (5)$$

$$WiMax_e = \max\{MaxWiMaxBw, C_{2e} * \frac{\alpha}{S_e^3}\} \qquad (6)$$

$$SatBw_e = MaxSatBw * \lfloor \frac{\min\{S_e, S_{sat}\}}{S_e} \rfloor \qquad (7)$$

where $C1$ and $C2$ are transmission quality rates defined in [0,1] (used to model radio path loss along edges), α is a constant parameter (see below), S_e is the speed value over edge e, S_{sat} is the maximum allowed speed for satellite communication, and $MaxTacBw$, $MaxWiMaxBw$ and $MaxSatBw$ are maximum bandwidth values for each type of communication. We suppose that satellite reception quality is constant over space.

We chose $\alpha = 6,400,000$ so that WiMax bandwidth is equal to $800KB/s$ at a speed of $20km/h$ and to $40KB/s$ at a speed of $55km/h$ approximately ($54.2km/h$ to be more precise). We also set $MaxTacBw = 40KB/s$, $MaxWiMaxBw = 1000KB/s$ and $MaxSatBw = 1200KB/s$ at a speed limit of $S_{sat} = 20km/h$. These values are those depicted in figure 4. Indeed this model is simple since available bandwidth depends on many other parameters (traffic demand and policy, communication distance, number of users, latency, etc.). Our model only captures the trade-off between vehicle speed and network bandwidth. The minimum available bandwidth is constrained for all edges from equation (4) with constraint (8):

$$\forall e, Bw_e > Bw_{min} \qquad (8)$$

III. State of the Art

Several types of mission planners have been developed for navigation purposes:

- Domain-independant planners [4] based on Planning Domain Description Language (PDDL) formalisms [5] can deal with complex actions but solving heuristics are not appropriate for complex constraint formulations.
- Constraint solving techniques have been widely investigated in planning areas: Ix-TeT [6], Heuristic Scheduling Testbed System (HSTS) [7], Reactive Model-based Programming Language (RMPL) [8]... Similarly, planning frameworks like Hierarchical Task Network (HTN)[9] or [10] have been developed to tackle specific operational domains. To our knowledge, none of these approaches address navigation and bandwidth management simultaneously.

The problem can also be relaxed as a Traveling Salesman Problem (TSP), when considering mandatory waypoints. Deterministic approaches [3], local search methods [11] and metaheuristics such as Simulated Annealing [12], Genetic Algorithms [13] or Ant Colony Optimization (ACO) algorithms [14] can be employed, but they have to be extended to deal with mixed integer or continuous programming techniques. In particular, ACO are known to be efficient on discrete and possibly dynamic problems.

Lastly, heuristic search methods based on the well-known A* [15] can also be envisaged. Two categories of algorithms have emerged in the two last decades: incremental heuristic searches[16] and real-time heuristic searches [17]. Again, these approaches necessitate extensions to deal with bandwidth management.

IV. Solving the BVP

To solve the problem rapidly enough for on-line navigation management, our approach hybridizes Constraint Programming (CP) techniques with metaheuristics. We introduce a new probe backtrack method as a hybridization mechanism.

A. Constraint Programming framework

The global solving techniques use a (CP) framework that combines a high level of expressiveness and powerful constraint solving techniques. The $\{0,1\}$ flow problem (see formulas (1) and (2)) is expressed with variables and arithmetic constraints. It uses state-of-the-art in discrete constraint optimization techniques, based on *SICStus Prolog* engine and its CLP(FD) library:

- Arc Consistency-5 (AC-5) [18] for constraint propagation, that is managed by $CLP(FD)$ predicates. When a variable domain get reduced, AC-5 propagates domain variables until a fixed point is reached.
- Variable filtering with correct values, using specific *labeling* predicates to instantiate problem domain variables. AC-5 being incomplete, value filtering guarantees the search completeness.
- Tree search with standard backtracking when variable instantion fails. It is associated to Branch and Bound (B&B) for cost optimization.

B. The Ant Colony metaheuristic

1) Algorithm Description: ACO belongs to the family of swarm intelligence metaheuristics. It spreads a population of ants through the state space (each one trying to build a solution) over a series of search cycles. The strategy used by ants is both based on a guiding heuristic and a collective memory model called *pheromone*, in analogy with real ants. The pheromone model is defined as an edge weight that is updated at the end of each cycle. If an edge belongs to a good known solution, the associated weight will be enforced to incitate future ants to take it.

Formally, an ant builds a path through the state space by iteratively electing the next vertex to take to reach the goal. To do so, it uses the following formula. For an ant k currently at vertex v, the probability of choosing a reachable vertex v' as the next waypoint is given by formula (9).

$$\forall (v,v') \in E, v' \in \omega^+(v), P_{v'}(k) = \frac{\tau_{v,v'}^{\alpha} \eta_{v'}^{\beta}}{\sum_{v'' \in \omega^+(v)} \tau_{v,v''}^{\alpha} \eta_{v''}^{\beta}} \quad (9)$$

$P_{(v,v')}$ is a probability and thus belongs to $[0,1]$. The $\eta_{v'}$ parameter is the guiding heuristic (described below). The $\tau_{(v,v')}$ parameter is the pheromone edge weight to go from vertex v to vertex v'. α and β are calibration variables that balance the importance given to τ and η parameters. It has an impact on algorithm convergence, as a search with a strong β value will be very orientated but may not allow a correct space exploration and thus an unexpected better solution discovery. The pheromone model update is made using the following formula:

$$\forall (v,v') \in E, \tau_{v,v'}(c+1) = \rho.\tau_{v,v'}(c) + \Delta\tau_{v,v'} \quad (10)$$

where ρ is the *conservation* factor. It allows decreasing pheromone weight over time to remove attraction from bad paths. $\Delta\tau_{v,v'}$ equals to:

$$\Delta\tau_{v,v'} = \begin{cases} \frac{1}{L_{LB}} & \text{if } (v,v') \in LB \\ 0 & \text{otherwise.} \end{cases} \quad (11)$$

where L_{LB} is the length of the local best solution.

2) BVP Problem Adaptation: In our solving strategy, we use ACO to address the problem of finding the best sequence of mandatory waypoints. To do so, a complete subgraph $G' = (V', E')$ is extracted from the initial G graph, where $V' = V_m \cup \{v_{start}, v_{goal}\}$ (V_m is the set of mandatory waypoints). Each distance value of E' edges is estimated using associated vertex coordinates. The difference with the classical TSP solving is that optimization is not made over a loop, and starting and ending nodes of ant paths are imposed.

Once a sequence is found, a simple A* algorithm is computed over G between each pair of vertices to build up a complete path. In the following, this *relaxed* solution (as it does not take into account bandwidth constraints) will be used to guide the global constraint solver.

C. The Probing Method

The goal of hybridizing global solving with a stochastic approach is to save the number of backtracks and to quickly focus the search towards most promising solutions. It consists in designing the tree search according to problem structure, revealed by the probe.

The idea is to use the ACO relaxed solution to order problem variables, as a preprocess. Instead of dynamic probing with tentative values such as in [19] (that consequently makes the algorithm non-complete), this search strategy uses a static probe which orders problem variables to explore according to the relaxed solution properties. Then, the solving follows

Mission Id	*Environment*	*Nodes*	*Edges*
1	urban	23	76
2	open	22	74
3	mixed	22	68

Fig. 5. Environment type and graph size

a standard CP search strategy, combining variable filtering, AC-5 and B&B.

Once the relaxed solution is available, the probe backtrack construction consists of two steps. Firstly, a minimal distance $\delta(v)$ between problem variables (that are φ_e, see section II-D) and the relaxed solution is computed. The second step uses these results to sort problem variables in ascending order. At the global solving level, the relaxed solution is useless, but problem variables will be explored following this order. To sort problem variables, we first compute a distance metric between each vertex of the graph and the relaxed solution. This method has polynomial complexity and still guarantees the global solver completeness (see [20] for more details). We experiment three different probing-based strategies :

1) *Basic probe*: distance metric is computed using the maximum value over the minimum number of bounds (that is to say the number of edges to take) between the partial solution path and edge extremities;
2) *Distance probe*: distance metric is computed using an average over the minimum euclidean path distance between the partial solution path and edge extremities;
3) *Bi-metric probe*: distance metric is computed as in *basic probe*, but further ordering is done between variables of same values according to bandwidth range maximum values (highest values are ranked in upper positions).

V. Results

For our experimentations, we used limited size instances to show the difficulty of solving BVP with two independent metrics (mission duration and bandwidth requirements). We consider three benchmarks inspired from realistic missions. Figure 5 depicts their characteristics which correspond to missions of few hours. The graph corresponding to mission 2 is shown in figure 2 (bandwidth value statement is defined in the following).

Although graph sizes are similar, urban environment significantly changes the graph connectivity structure, which alternate long and short edges. Also, radio reception rates are stated as a heterogeneous model, to make the problem more difficult (with less symmetries). On mission 2 for example, parameter $C2$ associated to the following edges were set to a rate of 0.5 : [(1,2), (1,4), (1,8), (2,4), (2,11), (4,8), (4,9), (4,10), (9,12), (9,13), (10,11), (10,12), (11,12), (12,13)]. The other edges were maintained at 1, as well as all $C1$ parameters. Over each graph, we generate multiple problems, on which we compare the three probe strategies detailed above. Each problem can be defined as:

minimize **mission duration** D
subject to **minimal bandwidth requirement** Bw_{min}.

For each mission benchmark, two levels of bandwidth limitations Bw_{min} are experimented, weakly ($40KB/s$) vs strongly constrained ($400KB/s$). For the six resulting cases, we run a serie of 10 executions over a different number of mandatory waypoints, from 0 to 5, chosen randomly (but being the same for all 10 executions and algorithms). In order to average the bias resulting from the graph structure and mandatory waypoints, the 10 executions differ in the starting and ending nodes definition. Those are selected on both extremities of a graph diameter to make each instance difficult enough. All 10 executions use different diameters to provide a heterogeneous set of instances. This gives 120 instances per benchmark, and 360 in total.

To limit solving time, a timeout has been set at 500 seconds. Experiments were run on a $2GHz$ entry-level laptop with $2GB$ of memory. In [20], we show that probe preprocessing can be neglected with respect to the complete solving. Figures 6 and 7 show the computation time needed to find the optimal solution for the two different bandwidth limitations and the three missions. X-Axis represents the series of runs, differing in the number of mandatory waypoints. For each serie, the three algorithms are represented : basic probe in blue, distance probe in green and bi-metric probe in red. Y-axis represents average solving time over the ten runs.

Concerning low bandwidth requirements, performances are obviously good on under-constrained instances (no or one mandatory waypoint). At the opposite, instances with more than two or three waypoints get over-constrained, then the solution space get rapidly cut by an overkill of constraint propagation and it is also easy to find solutions. In between, the instances with two or three mandatory waypoints are more difficult to solve. In general, this is because the solution space is sparse and some exploration is required to improve suboptimals in the B&B loop. On these instances, the three solving methods behave quite similarly. For all of them, optimality can be proven on all these weakly-constrained bandwidth instances within the 500 seconds.

On strongly-constrained examples, the problem is amplified with the strong bandwidth constraints and similar behavior is characterized. However, the solving behaviour differs in three aspects:

- The distance probe behaves differently over the three mission benchmarks;
- Proof of optimality can be provided only for most benchmark instances. Only the hardest instances of the second benchmark exhibit difficulties. Figure 8 gives the number of timeouts on the Y-axis, where timeouts start to occur with two mandatory waypoints.
- The most difficult instances are now for the series with four or five waypoints where optimality cannot be proven. Some difficulties subsist for series with two or three.

Interestingly enough, sorting search variables with bandwidth metrics does not improve the behaviour of the *bi-metric* probe compared to the *basic* one. In addition, another interesting property is shown. Over the easiest instance (mission 1), *distance* probe appears to be less efficient in comparison to

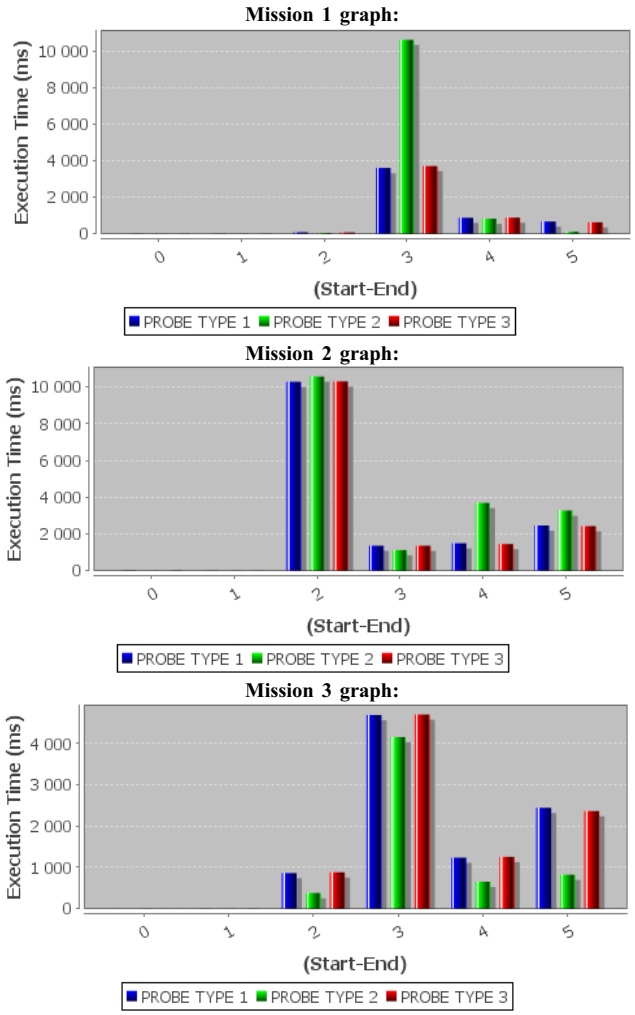


Fig. 6. Finding optimal solutions on weakly constrained problems

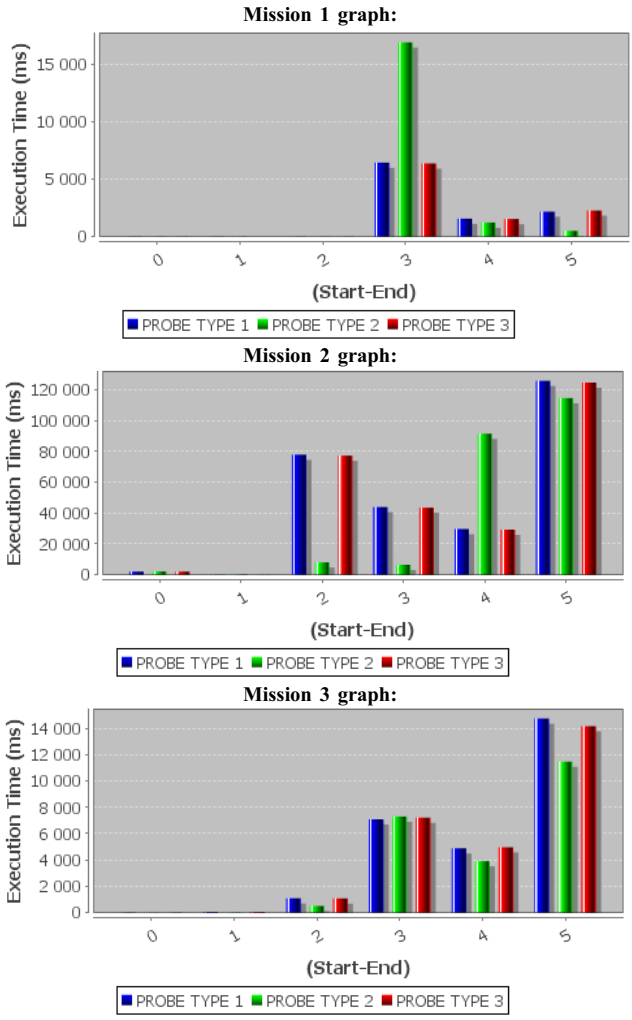


Fig. 7. Finding optimal solutions on strongly constrained problems

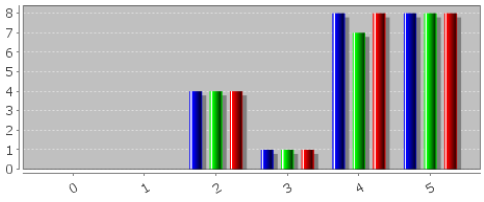


Fig. 8. Timeouts on mission 2 graph

the other strategies. Indeed, computation time is approximately 2.5 times worse both in wealky and strongly bandwidth constrained instances in the series involving 3 mandatory waypoints. Over mission 3 (of average difficulty), *distance* probe is always slightly better that the other strategies. Over mission 2, the most complex problem, results need a thorough analysis. While *distance* probe does not bring better results on the weakly constrained instance, its behaviour on the strongly constrained instances reveals a significant improvement in comparison to the other strategies. Graphics must be carefully read : in the series involving 4 mandatory waypoints, *distance* probe seems to have worse efficiency (computation time is 3 times higher). However, this is due to the fact that the probe allows to solve one more problem, as we can see in figure 8. As average computation times are computed only on solutions that were *proved* to be optimal, the value shown in optimal finding solving time is not computed over the same number of runs (and the "extra" solved case takes a lot of time to find the optimal solution, that leads to a higher average value). As a result, the *distance* probe is in fact better on *every* strong constrained instance of mission 2.

VI. Discussion and Future Work

As we showed in the results, the *bi-metric* probe brought no improvement in the search strategy. An interesting way of considering both distance and bandwidth metrics could be to introduce bandwidth metric in ACO distance estimates of the precomputation algorithm. Its cost fonction should then evaluate both metrics with distinctive weights.

	Mission 1	*Mission 2*	*Mission 3*
Average connectivity	3.30	3.36	3.09
Distance CV	0.49	0.26	0.53

Fig. 9. Mission Graph Properties

The analysis of probe construction strategies presented above show interesting properties : the *distance* probe seems to be better adapted to the most complex problems while *basic* probe (and in extension the *bi-metric* probe) seems to better address simple problems. Then, a question arises: knowing that problem sizes are quite the same (see in figure 5), why are the the computation times (and thus the instance complexities) so variable?

Is is rather hard to exactly determine a graph complexity. However, with a little intuition, it is possible to find properties allowing to estimate this complexity. We mainly distinguished two simple properties : the first one is the *average connectivity factor* (the average number of edges connecting nodes). Indeed, the higher the connectivity factor, the higher the alternative solutions while choosing the next variable during the search. As a consequence, more solutions are evaluated and complexity increases. The second property that can be used is the *coefficient of variation* (CV) of edge distances (even if we are not talking about distributions here). This intuition relies on *branch & bound* mechanism knowledge. When a first valid solution is found, its cost function value is used to bound the search in the next evaluations. If value distances are sensibly different (and thus CV is high), it may be easy to discard a potential solution if the cost of the new selected variable is much higher. However, if all costs are quite the same, the solver may need to go deep in the search tree before invalidating an alternative solution and pruning the tree. Therefore, a low edge distance CV and a high average connectivity factor will have an impact by increasing problem complexity. Table 9 summarizes these property values for each problem instance. The obtained values corroborate well the previous explanations : for example, mission 2 presents the lowest CV value and the highest connectivity factor over the three instances. These findings lead us to an interesting point : by evaluating complexity before the beginning of the solving, it would be possible to choose the best strategy according to the problem shape. This mechanism would be an interesting alternative to currently existing work of that kind [21]. This will be part of our future work.

VII. Conclusion

Future combat vehicle will have to balance mobility, mission objectives and mobile network management. To adress this issue, we have proposed an original approach to tackle Bandwidth-constrained Vehicule navigation Planning (BVP) problems with possible mandatory waypoints.

It combines an exact solving method with an ACO metaheuristic guidance. Hybridization is achieved using a probe backtrack search that is built on a relaxed solution of the problem, returned by the metaheuristic. This pre-processing step, consisting of ranking variables, allows the backtrack method of the complete solver to select the most promising variables first.

Variants of probe strategies are quite equivalent, although the *distance* probe provides some better performances, but this depends on problem structure. The global approach presented yields performances close to operational expectations.

References

[1] C. Guettier, P. Sechaud, J. Yelloz, G. Allard, I. Lefebvre, P. Peteuil, P. Ponthereau, F. Cuisinier, and J. Martinet, "Improving tactical capabilities with net-centric systems: the phoenix'08 experimentation," in *Proceedings of IEEE MILCOM Military Communications Conference*, 2009.

[2] R. Hassin, "Approximation schemes for the restricted shortest path problem," *Mathematics of Operations Research*, vol. 17, no. 1, pp. 36–42, 1992.

[3] M. Gondran and M. Minoux, *Graphes et Algorithmes*. Editions Eyrolles, 1995.

[4] M. Fox and D. Long, "Automatic synthesis and use of generic types in planning," in *Proceedings of the Artificial Intelligence Planning System*. AAAI Press, 2000, pp. 196–205.

[5] ——, "Pddl 2.1: An extension to pddl for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, 2003.

[6] P. Laborie and M. Ghallab, "Planning with sharable resource constraints," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Menlo Park, California, USA, 1995.

[7] N. Muscettola, "Hsts: Integrating planning and scheduling," in *Technical Report CMU-RI-TR-93-05*, Robotics Institute, Carnegie Mellon University, 1993.

[8] M. Abramson, P. Kim, and B. Williams, "Executing reactive, model-based programs through graph-based temporal planning," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Seattle, WA, USA, 2001.

[9] R. Goldman, K. Haigh, D. Musliner, and M. Pelican, "Macbeth: A multi-agent constraint-based planner," in *Proceedings of the 21st Digital Avionics Systems Conference*, vol. 2, 2002, pp. 7E3:1–8.

[10] N. Meuleau, C. Plaunt, D. Smith, and T. Smith, "Emergency landing planning for damaged aircraft," in *Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference*, 2009.

[11] E. Aarts and J. Lenstra, *Local Search in Combinatorial Optimization*. Princeton University Press, 1997.

[12] V. Cerny, "A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, pp. 41–51, 1985.

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

[14] M. Dorigo and L. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[15] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems, Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[16] S. Koenig, X. Sun, and W. Yeoh, "Dynamic fringe-saving a*," in *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 2, 2009, pp. 891–898.

[17] A. Botea, M. Mller, and J. Schaeffer, "Near optimal hierarchical path-finding," *Journal of Game Development*, vol. 1, no. 1, 2004.

[18] Y. Deville and P. V. Hentenryck, "An efficient arc consistency algorithm for a class of csp problems," in *Proceedings of the 12th International Joint Conference on Artificial intelligence (IJCAI)*, vol. 1, 1991, pp. 325–330.

[19] H. E. Sakkout and M. Wallace, "Probe backtrack search for minimal perturbations in dynamic scheduling," *Constraints Journal*, vol. 5, no. 4, pp. 359–388, 2000.

[20] F. Lucas, C. Guettier, P. Siarry, A. de La Fortelle, and A.-M. Milcent, "Constrained navigation with mandatory waypoints in uncertain environment," *International Journal of Information Sciences and Computer Engineering (IJISCE)*, vol. 2, to appear.

[21] W. Ruml, "Incomplete tree search using adaptive probing," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, vol. 1. Morgan Kaufmann Publishers Inc., 2001, pp. 235–241.

Hybrid Solving Technique for Vehicle Planning with Communication Constraints

François Lucas and Christophe Guettier
Sagem Defense & Security
27, Rue Leblanc, 75012 Paris, France
Email: {francois.lucas,christophe.guettier}@sagem.com

***Abstract*—The complexity of modern engagements and the numerous communication links made available increase the difficulties of mission planning and execution. In particular, finding unit waypoints while meeting frequencies allocation is a hard operational problem. Continuity of communications among the command chain must be guaranteed on the move despite tactical mobility, networks availability, path loss, jamming, mission updates and versatile threats. In previous works, we proposed a problem formulation relying on vehicle plan optimization, that satisfies network connectivity and operational constraints. This paper addresses hybrid search techniques to solve Bandwidth-constrained Vehicle Planning (BVP) on-line problem instances. It is modeled using constraint-based formulations to express both network connectivity and operational requirements (missions, waypoints, terrain). Vehicle planning is then solved and optimized according to a combination of a metaheuristic, namely Ant Colony Optimisation (ACO) and Constraint Programming (CP) techniques. Various forms of optimization strategies are compared and evaluated on realistic scenarios, outperforming previous results.**

I. Introduction

In modern Network Centric Warfare (NCW), ground units are able to share tactical data such as observation reports, images or videos. The network is also a powerful communication vector allowing to update mission objectives and to enforce the situation awareness (threats and obstacles) of these actors. Behind the NCW concept, the challenge is consequently to ensure the communication continuity in spite of mission contingencies. Since spectrum allocation is a difficult issue in operations, network users must adapt their manoeuvres and tactics to the available bandwidth.

This paper addresses vehicle planning issues, managing constraints composed of mission objectives and bandwidth requirement in a dynamic environment. In this problem, vehicles are able to connect to the tactical network through different waveforms : tactical radios (UHF, VHF), high bandwidth local access (like *WiMax*) or satellites. Given a final position and a list of non-sequenced intermediate objectives to reach, the optimization problem consists of finding the path that minimizes the overall mission time while ensuring a minimum bandwith to the tactical network all along this path. This problem is called the Bandwidth-constrained Vehicle Planning (BVP) problem. Due to the insertion of *reception quality* factors, modelling the ground specificities and the deployment of networking relays, communication constraints make this problem harder to solve. Indeed, determining the shortest path may not lead to the fastest one as it may pass through an area with no or low reception quality. This effect is due to radio properties, whose available bandwidth decreases when vehicle speed increases.

In general, network frequency allocation and vehicle plans are considered separately and always defined at mission preparation time. Frequency allocation is statically defined for a large period of time, while unit command has some degree of freedom to adapt its course of action. In this paper, we consider dynamic handover of vehicle communications, coupled with on-line navigation planning. This approach can efficiently propose vehicle planning alternatives when contingencies occur or communications are perturbated. The goal is to provide to the user (vehicule of company commander, platoon leader, relaying vehicle ...) a decision support functionality by solving a BVP problem. The planner advises on-line the best route to follow under specified mission constraints, and according to available bandwidth on the different network accesses. System efficiency criteria are *solution optimality* and *reactivity*: it must give the best solution in execution times close to those of mission needs ($\leq$ 10 seconds). This approach can also be applied to unmanned vehicles, where navigation plans must be updated whenever the mission changes, the environment evolves, or the network access becomes too limited.

This paper brings new results over this problem, which was already presented in MILCOM'10 [1]. In our first work, we proposed a hybrid approach in which a meteheuristic-based algorithm was used to guide the search of a complete Constraint Programming (CP) solver. The results showed interesting performances, although not applicable in highly reactive situations. From that time, numerous improvements have been made on this approach, both on the logical CP model and the preprocessing algorithm.

II. Navigation and Bandwidth Management

A. Problem context

With new high performance observation sensors, tactical units need to support more and more bandwidth demands. This is already the case for soldiers and officers (mounted or dismounted), but also for specific units and associated vehicles (Joint Terminal Attack Controllers), Unmanned Aircraft/Ground Vehicles (fig. 1). Nowadays vehicles or dismounted soldiers employ widely HF, VHF, UHF, satellites links according to situation awareness and mission objectives.

Fig. 1. This long endurance autonomous UAV can perform network relaying or/and observation missions. In both cases, network access in several frequency bands is a critical problem.

Different kinds of networks are available: fixed or deployable infrastructures as well as tactical mobile networks (see section II-E for further details). To handle communication perturbations, the network operator must cope with jamming, frequency plan, path loss and tactical movements. Sometime under critical conditions, the operator must deal with real-time communication handover on the different available networks, and must provide continuity of communication on the move to its unit.

B. Bandwidth-constrained Vehicle Planning

The Bandwidth-constrained Vehicle Planning (BVP) problem consists in optimizing an itinerary from the current vehicle location to an objective waypoint. This problem can occur for reccon and contact units at the tactical edge, or during logistics and medevac missions. Mandatory waypoints can be imposed to express secondary objectives to fulfill or other mission constraints (such as imposed check points). The problem is to minimize mission duration, while ensuring a minimum radio bandwidth.

The following situation is inspired from a real case (fig. 2). From location S, a vehicle is given a goal G and a list of mandatory waypoints to reach (mission objectives, identified with gray nodes). The optimal path must go through all the mandatory waypoints in a minimal time while satisfying a continuous communication capacity threshold on the itinerary. However, the constraint can compete with the metric: the bandwidth can decrease as a function of speed, depending on the radio model. Moreover, radio reception quality depends on edges, waypoints and the underlying terrain.

The BVP Problem is NP-hard and exposes a combinatorial problem structure. Finding the best mandatory waypoint sequence can be reduced into a Traveling Salesman Problem (TSP) instance. Compared to TSP state of the art, this work adresses small and composite problem instances where short response times are needed.

C. Input Data and Replanning

In our problem, progression axis are modeled as a graph structure over the area of operation. A vertex is a geographical point of tactical interest (for example crossroads, bridges, foothill tops, dots) and an edge a potential progression axis (for example a road or a meadow) for the unit. Radio propagation is modeled as a constant weight over edges, assimilated to path loss as a ratio over best possible reception (see subsection II-E for more details). A travel distance is associated to each graph edge, and can be given by terrain analysis. These data are provided by C2 systems at mission preparation time, but can be updated during mission execution:

- path loss updates due to various sorts of radio jamming. Also, path loss may have been overestimated.
- new navigation points may be available and existing ones can be forbidden.
- mission objectives and/or mandatory navigation points may be updated.

When input parameters change, replanning time is critical for continuity of communication. Alternate path can be found either to increase communication capacity or to reduce the duration of the tactical move.

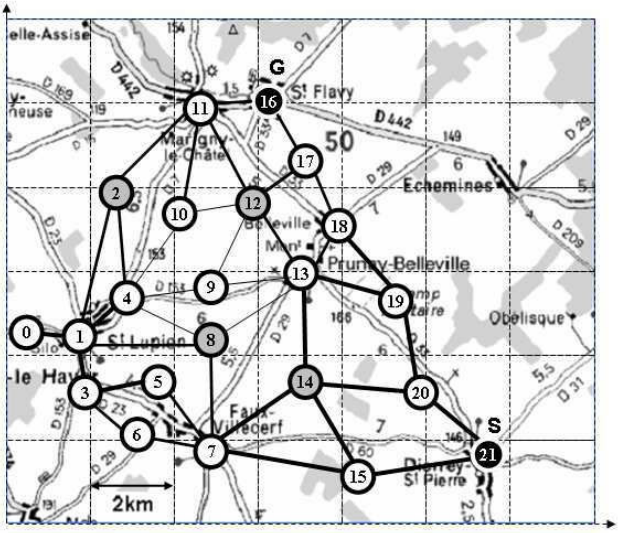


Fig. 2. Example of BVP problem with particularized bandwidth constraints on edges.

D. Problem Formalization

Formally, the trafficability graph G is defined as a couple $G = (V, E)$ where V is the set of vertices and E is the set of edges. V_m is a subset of V that represents the set of mandatory waypoints. A valid path starts from vertex v_s and reaches vertex v_g after having passed through all mandatory waypoints. The BVP problem cas be consequently expressed as the following optimization problem:

$$\min_{p=\in P'(v_s,v_g,V_m)} \sum_{e\in p} t(e) \tag{1}$$

where

$$P'(v_s, v_g, V_m) = \{p \in P(v_s, v_g, V_m) \mid \forall e \in p, b(e) \geq B_{min}\} \tag{2}$$

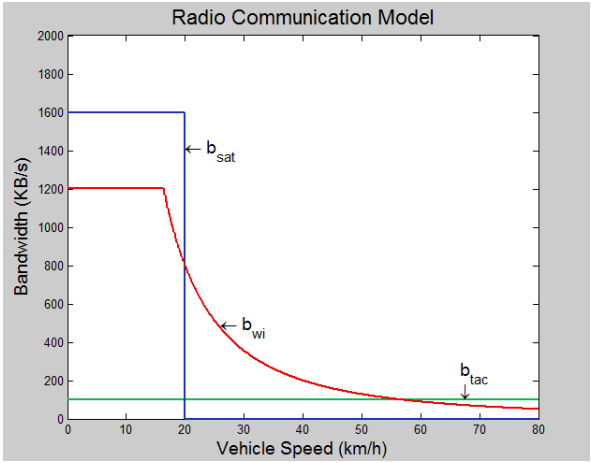


Fig. 3. Different types of radio communication system capabilities.

$P(v_s, v_g, V_m)$ denotes the set of paths from v_s to v_g, visiting all V_m nodes. $b(e)$ represents the maximum available communication bandwidth, expressed below. B_{min} is the required minimum bandwidth, specified as an input of the problem. $t(e)$ denotes the time variable of an edge e, defined as:

$$\forall e \in E_p,\ t(e) = \frac{d(e)}{s(e)} \tag{3}$$

where $s(e)$ is the vehicle speed along the edge e. $s(e)$ is defined in $[0, S_{max}]$ range, where S_{max} is an input parameter.

E. Bandwidth metric

Three main wireless communication technologies may be used by ground units:

- tactical mobile network in HF / VHF / low UHF bands: they provide reliable links, according to terrain and speed (green curve).
- deployable infrastructure (highest UHF) provides highest capacity (derived from civil standards such as *WiMax*) but decrease over speed, distance (red curve) and path loss is more sensitive to terrain structure. These infrastructures can be used close to forward operation bases, where antenae can be deployed.
- fixed infrastructure, where the most representative are satellite-based communications, providing high bandwidth but with a very limited mobility (blue curve), due to tracking antenna.

In our model, bandwidth constraints are associated to edges. Considering that only one communication system is used at a time, the global bandwidth availability metric is a piecewise function. Assuming that $b_{tac}(e)$,$b_{wi}(e)$ and $b_{sat}(e)$ are tactical, high capacity and satellite available bandwidth for an edge $e \in p$ respectively, the global available bandwidth is defined as:

$$b(e) = \max\{b_{tac}(e), b_{wi}(e), b_{sat}(e)\} \tag{4}$$

Bandwidths are defined as follow for tactical network, deployable and fixed infrastructure, respectively with equations (5, 6 and 7):

$$b_{tac}(e) = C_1(e).B_{tac} \tag{5}$$

$$b_{wi}(e) = \min\left\{B_{wi}\ ,\ C_2(e).\frac{\alpha}{s_e^2}\right\} \tag{6}$$

$$b_{sat}(e) = B_{sat}.\left\lfloor \frac{\min\{s_e, S_{sat}\}}{s_e} \right\rfloor \tag{7}$$

$C1$ and $C2$ are communication quality rates defined in [0,1] (used to model radio path loss along edges) for tactical and *WiMAX* radio respectively. Parameter α is a utility constant. s_e is the vehicle speed value over edge e, S_{sat} is the maximum allowed speed for satellite communication, and B_{tac}, B_{wi} and B_{sat} are maximum bandwidth values for tactical, *WiMAX* and satellite communications respectively. In this model, satellite reception quality is supposed maximum everywhere on the ground.

The following parameter values were used for this model :

- $B_{tac} = 100$KB/s ;
- $B_{wi} = 1200$KB/s ;
- $B_{sat} = 1600$KB/s ;
- $S_{sat} = 20$km/h ;
- $\alpha = 320000$.

The model proposed above makes strong simplifications, as a fidel representation involve many environment parameters. However, it allows us to introduce complexity in the navigation problem. Moreover, the values that were selected (empirically) for this model may be seen as very optimistic, in comparison to effective current technologies.

F. Impact of radio constraints on planning

The radio communication constraints may heavily impact on navigation plans, depending on the B_{min} minimum bandwidth requirements. if B_{min} is high, and if there are areas in the environment where reception quality is low (low C_1 and C_2 edge values), a vehicle may need to decrease its speed to maintain required communication. Consequently, take a detour may save time as it may avoid worst path loss areas.

III. State of the Art

Both planning formalization and combinatorial search techniques are needed to tackle a BVP problem.

A. Planning Formalization

Domain-independant planning [2] based on Planning Domain Description Language (PDDL) formalisms [3] can deal with different sorts of complex actions, and can be somewhat adapted to vehicle planning. However, complex constraint formulations require additional solving mecanisms. At the opposite, different dedicated planners have been developed for UAV, UGV and vehicle planning: Ix-TeT [4], Heuristic Scheduling Testbed System (HSTS) [5], Reactive Model-based Programming Language (RMPL) [6]... Similarly, planning frameworks like Hierarchical Task Network (HTN)[7] or [8]

have been developed to tackle specific operational domains. So far, none of these approaches have addressed navigation and communication capacity management simultaneously.

B. Combinatorial Search for Vehicle Planning

Different search techniques have been developped to solve directly vehicule routings, with Operation Research[9], Local Search[10] and with metaheuristics Simulated Annealing [11], Genetic Algorithms [12]. More recently, Ant Colony Optimization (ACO) metaheuristics [13] have been shown to be efficient on discrete and possibly dynamic problems. However, most of these techniques cannot directly cope with our hybrid problem formulation.

Many heuristic search methods are based on the well-known A* [14] and also commonly used in vehicle planning. Several derivatives have been developed, such as Anytime A* [15], or other variants adapted to dynamic environments, divided into two categories: incremental heuristic searches[16] and real-time heuristic searches [17]. However, these simple path search approaches necessitate extensions to deal with bandwidth management and/or secondary objectives.

IV. Solving the BVP

This section summarizes the solving approach developed in [1] and improvements that were made since our last work. To fastly solve the BVP problem, we developed an approach mixing metaheuristics and Constraint Programming (CP), using a probe backtrack method as a hybridization mechanism.

A. The CP Solver

1) Path Logical Model: To model a path in the trafficability graph G, a flow model based on Kirchhoff laws is used. A path is expressed as a 1-flow from v_s to v_g. To do so, each edge $e \in E$ is given a flow variable $\varphi_e \in \{0, 1\}$. The vehicle will path through an edge if $\varphi_e = 1$, and will ignore it otherwise. This is formulated as the following constraint, where $\omega^-(v)$ and $\omega^+(v)$ represent the set of incoming and outgoing edges of vertex v respectively:

$$\sum_{e \in \omega^+(v_s)} \varphi_e = 1, \quad \sum_{e \in \omega^-(v_g)} \varphi_e = 1, \tag{8}$$

$$\forall v \in V \setminus \{v_s, v_g\}, \sum_{e \in \omega^+(v)} \varphi_e = \sum_{e \in \omega^-(v)} \varphi_e \leq 1 \tag{9}$$

Equation (9) ensures path connectivity and unicity while equation (8) imposes conditions on v_s and v_g nodes. Objectives can be expressed very easily as :

$$\forall v_m \in V_m, \sum_{e \in \omega^-(v_m)} \varphi_e = 1 \tag{10}$$

Through equation 10, the flow (i.e. the path) is constrained to pass through each node $v_m \in V_m$.

The flow model given above is not sufficient to ensure path consistency. Indeed, a flow cycle external to the path may not be excluded whereas the solution feasibility is compromised. In previous work, we consequently used time variables to prevent such problematic cases. Time values over each node of the path corresponded to the cumulated path time from v_s. Propagating these values allowed, as a path edge value is necessarily greater than zero, to introduce precedence constraints over nodes, thus removing cycles. However, in the BVP Problem, propagating time may lead to important computation time as bandwidth constraints are directly related to these constraints. A specific propagator, composed of new variables named id, was consequently set up:

$$id_{v_s} = 1 \tag{11}$$

$$\forall v \in V \backslash \{v_s\}, id_v = \sum_{e=(u,v) \in \omega^+(v)} \phi_e.(id_u + 1) \geq 0 \tag{12}$$

$$id_{v_g} = 1 + \sum_{e \in E} \phi_e \tag{13}$$

These variables ensure that the cumulated number of edges (or flows) composing the path from v_s to v_g is equal to the specified id at v_s minus 1. If a cycle is created outside the path, the equality does not hold and the solution is immediately discarded, without having to compute bandwidth values.

2) Radio Model: The radio model can be implemented directly in CP using equations of section 3. However, computing the maximum bandwidth for each time value may rapidly lead to important computation time. However, it is possible to strongly simplify the problem by adding cuts to time values, using problem knowledge:

$$\forall e \in E, t(e) \geq \begin{cases} 0 & \text{if } B_{min} > B_{sat} \\ \frac{d(e)}{S_{sat}} & \text{if } B_{sat} \geq B_{min} > B_{wi} \\ \frac{d(e)}{s_{wi}(e)} & \text{if } B_{wi} \geq B_{min} > B_{tac} \\ \frac{d(e)}{S_{max}} & \text{if } B_{min} \leq S_{tac}. \end{cases} \tag{14}$$

Knowing that

$$b_{wi}(e) = C_2(e).\frac{\alpha}{s(e)^2} \geq B_{min} \tag{15}$$

s_{wi} is expressed such that

$$s_{wi}(e) \leq \sqrt{C_2(e).\frac{\alpha}{B_{min}}} \tag{16}$$

Equation 14 allows, for each edge *independently*, to apply a lower bound on time values (equivalently to an upper bound on speed values). With these cuts, inconsistent time values will not be evaluated anymore. This equation remains an inequality (and not a strong equality) to keep genericity of our solving approach (time variables are expressed in seconds and may be constrained by other mission requirements).

B. Solving Techniques

1) Hybrid Solving with Constraint Programming: The CP framework proposes powerful constraint solving methods. The *SICStus Prolog* engine, through the CLP(FD) library, uses state-of-the-art discrete constraint optimization techniques:

- Arc Consistency-5 (AC-5) [18] for constraint propagation, that is managed by $CLP(FD)$ predicates. When

a variable domain get reduced, AC-5 propagates domain variables until a fixed point is reached.
- Variable filtering with correct values, using specific *labeling* predicates to instantiate problem domain variables. AC-5 being incomplete, value filtering guarantees the search completeness.
- Tree search with standard backtracking when variable instantion fails. It is associated to Branch and Bound (B&B) for cost optimization.

This framework is interesting since solving algorithm guarantee optimality and allow to prove optimality, even when optimization is subject to constraints. Solving execution is deterministic, which is important in most of critical applications, where scenarios may have to be reproduced. Lastly, search tree guarantees completness, since all variable instantations have been either explored or eliminated.

CP can be hybridized with many other search techniques. The probing technique described in **??** encapsulates metaheuristic to determine the CP optimization search. The metaheuristic provides partial solutions based on problem relaxation (mandatory waypoints, network capacity constraints). Then, the probe uses an estimate of constraint violations by the metaheuristic in order to build up the optimization search tree.

C. An Ant Colony approach to guide CP

The Ant Colony (ACO) metaheuristic is not described here. For more details, we refer to [1], [13].

1) Algorithm Description: In our last work, we developed an adaptation of ACO to fastly determine the shortest path from v_s to v_g visiting all V_m nodes. Optimization was made using an euclidean distance cost function. Once this relaxed solution was obtained (it is relaxed because bandwidth constraints were not considered), it was used through a *probe distance calculation* to organize flow variables of the CP search tree. In discussion, several probe distances metrics were discussed.

2) Search Guidance: Considering solely distances in ACO may lead to inefficient CP search guidance, as the shortest path may not be the fastest one (see section II-F). It is consequently necessary to take into account radio bandwidth in ACO optimization. To do so, ACO does not consider distance anymore, but the minimum time values over each node that do not violate the constraints. To do so, we used the equation 14 that was applied on each edge as an *equality* constraint. This way, the relaxed solution becomes a solution to the global problem which will necessary guide CP to a feasible solution. Moreover, it is now possible to used the relaxed solution time values as temporary upper bounds on CP time variables to fastly guide them towards lowest cost solutions.

V. Results

This section presents the results that were obtained using our modified approach. The probe distance that is employed in this paper is a *min bound* distance (see [1]). Compared our to previous work, one to two orders of magnitude were gained, which allows us to seriously consider using this solver in an embedded framework.

Mission Id	*Environment*	*Nodes*	*Edges*
1	urban	23	76
2	open	22	74
3	mixed	22	68

Fig. 4. Environment type and graph size

For our experimentations, three benchmarks inspired from realistic missions were considered. Table 4 depicts their graph characteristics. The graph corresponding to mission 2 is shown in figure 2 (bandwidth value statement is defined in the following).

Although graph sizes are similar, urban environment significantly changes the graph connectivity structure, which alternate long and short edges. Also, radio reception rates are stated as a heterogeneous model, to make the problem more difficult (with less symmetries). On mission 2 for example, parameter $C2$ associated to the following edges were set to a rate of 0.5 : [(1,2), (1,4), (1,8), (2,4), (2,11), (4,8), (4,9), (4,10), (9,12), (9,13), (10,11), (10,12), (11,12), (12,13)]. The other edges were maintained at 1, as well as all $C1$ parameters. Over each graph, we generated multiple problems, on which we ran our solver. Six series of instances were generated, each series corresponding to a fixed number of objectives (from 0 to 5). In each series, 10 instances (each corresponding to a different (v_s, v_g) couple) are evaluated. The following results present the average time for the CP solver, over each series, to find the optimal solution (T_1) and to prove the solution optimality (T_2). The optimality is proved when the search finishes. The following results do not consider preprocessing computation time as they can be neglected (less than 100ms in any case).

The maximum vehicle speed was set at $S_{max} = 60$km/h. The minimum required bandwidth was set at a high value, namely $B_{min} = 900$kB/s to strongly constrain the problem.To limit solving time, a timeout has been set at 500 seconds. Experiments were run on a *Samsung Q310* entry-level laptop with a 2GHz double-core microprocessor and 3GB of cache memory.

Results are detailed in table 5. We compared three approaches, each using the new CP solver: an approach using a simple Dijkstra algorithm to order CP flow variables (named SP for *shortest path*), our ACO adaptation (named ACO) without using time bounds mentioned in IV-C1, and our ACO adaptation with time bounds implementation (named ACO(2)).

Three preprocessing approaches are considered: SP (*shortest path*), ACO (without time bounds) and ACO(2) (implementing time bounds). Values between parenthesis shows the number of instances which were not able to be solved before the 500ms timeout (not considered in the average time computation in this case).

As a first comment, it can be seen that computation times are much lower than those obtained in our previous work. This result is mainly due to CP model modifications, in which the new propagator enables much faster path consistency check

$\|Vm\|$	SP Probe		ACO Probe		ACO(2) Probe	
	T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)	T_1 (ms)	T_2 (ms)
urban scenario 1						
0	142	344	143	346	17	304
1	59	260	171	386	43	294
2	443	737	185	483	129	539
3	1065	2002	5051	7041	4227	6495
4	794	1467	471	1107	152	717
5	550	1302	2670	3876	123	1138
open scenario 2						
0	96	234	301	781	322	730
1	1196	1441	337	925	123	823
2	131	507	923	1873	836	1794
3	28095	28861	2364	4433	1128	3365
4	16010	17297	1675	4945	1889	5238
5	46433	66748 **(3)**	8729	11587	2307	5300
mixed scenario 3						
0	240	519	1499	1810	9	268
1	436	814	1118	1667	173	645
2	433	812	1035	1449	190	578
3	569	1330	1658	2875	71	912
4	466	1221	372	1692	240	956
5	546	1341	4600	6148	333	1112

Fig. 5. Average time in milliseconds needed for the CP solver to find the best solution (T_1) and to prove the solution optimality (T_2).

and the time cut prevents from considering unfeasible time values. It can be noticed that even SP approach show very high performances on scenarios 1 and 3. Results are globally even better than those of ACO and ACO(2). However, on scenario 2, SP has much difficulties to fastly solve instances of series 3,4 and 5. It even fails at proving the solution optimality (and even the optimal solution) on 3 instances of series 5. Compared to SP, ACO is a bit less efficient on scenarios 1 and 3 but performs much better on scenario 3, that are solved in less than 12s on average for series 5 (worst case). Finally, temporary upper bounds applied to CP time variables using relaxed solution values show their relevance in terms of computation time savings, as ACO(2) significantly outperforms ACO on any case (even on worst cases such as series 5 of scenarios 2 and 3).

In conclusion, the ACO(2)+CP approach shows performances that are consistent with an embedded use for highly reactive replanning situations. Our experimentations showed that worst case computation times, on realistic scenarios, are lower than 5.5s in average (and lower that 8.8s in the worst case). This work brings much improvement to the approach previously proposed in MILCOM, and it is now possible to envisage inserting this preprocessing approach on multi-vehicle problems.

VI. Conclusion

Adapting dynamically vehicle planning thanks to frequency allocation, objectives and other constraints is a hard operational issue. The paper improves an original approach to tackle Bandwidth-constrained Vehicle Planning (BVP) problems with possible mandatory waypoints. This approach, combining an exact Constraint Programming (CP) solver with a metaheuristic-based preprocessing step, was initially proposed in MILCOM'10. Since that time, several important modifications were led in order to improve solving performance, and presented in this paper. In our CP model, the combination of a specific propagator and cuts applied to time variables allowed to significantly fasten the search exploration. In our preprocessing step, we took into account radio bandwidth in order to increase CP guidance efficiency and obtain a feasible solution regarding to the CP model. Finally, we took advantage of the relaxed solution time values to apply temporary upper bounds on time values to fastly guide them to a low cost value solution. The results that were obtained are now consistent with an on-line use, and outperform previous performances. This solving strategy showed that combination of CP and metaheurstics can take advantage of both approaches, leading to high performance solving times *and* optimality. We now plan to introduce this solving approach in a multi-vehicle context to manage the mission planning globally.

References

[1] F. Lucas and C. Guettier, "Automatic vehicle navigation with bandwidth constraints," in *In Proceedings of MILCOM 2010*, San Jose, CA, USA, November 2010.

[2] M. Fox and D. Long, "Automatic synthesis and use of generic types in planning," in *Proceedings of the Artificial Intelligence Planning System*. AAAI Press, 2000, pp. 196–205.

[3] ——, "Pddl 2.1: An extension to pddl for expressing temporal planning domains," *Journal of Artificial Intelligence Research*, vol. 20, 2003.

[4] P. Laborie and M. Ghallab, "Planning with sharable resource constraints," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Menlo Park, California, USA, 1995.

[5] N. Muscettola, "Hsts: Integrating planning and scheduling," in *Technical Report CMU-RI-TR-93-05*, Robotics Institute, Carnegie Mellon University, 1993.

[6] M. Abramson, P. Kim, and B. Williams, "Executing reactive, model-based programs through graph-based temporal planning," in *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, Seattle, WA, USA, 2001.

[7] R. Goldman, K. Haigh, D. Musliner, and M. Pelican, "Macbeth: A multi-agent constraint-based planner," in *Proceedings of the 21st Digital Avionics Systems Conference*, vol. 2, 2002, pp. 7E3:1–8.

[8] N. Meuleau, C. Plaunt, D. Smith, and T. Smith, "Emergency landing planning for damaged aircraft," in *Proceedings of the 21st Innovative Applications of Artificial Intelligence Conference*, 2009.

[9] M. Gondran and M. Minoux, *Graphes et Algorithmes*. Editions Eyrolles, 1995.

[10] E. Aarts and J. Lenstra, *Local Search in Combinatorial Optimization*. Princeton University Press, 1997.

[11] V. Cerny, "A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm," *Journal of Optimization Theory and Applications*, vol. 45, pp. 41–51, 1985.

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

[13] M. Dorigo and L. Gambardella, "Ant colony system: A cooperative learning approach to the traveling salesman problem," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 53–66, 1997.

[14] P. Hart, N. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems, Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[15] E. Hansen and R. Zhou, "Anytime heuristic search," *Journal of Artificial Intelligence Research*, vol. 28, pp. 267–297, 2007.

[16] S. Koenig, X. Sun, and W. Yeoh, "Dynamic fringe-saving a*," in *Proceedings of the 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, vol. 2, 2009, pp. 891–898.

[17] A. Botea, M. Mller, and J. Schaeffer, "Near optimal hierarchical path-finding," *Journal of Game Development*, vol. 1, no. 1, 2004.

[18] Y. Deville and P. V. Hentenryck, "An efficient arc consistency algorithm for a class of csp problems," in *Proceedings of the 12th International Joint Conference on Artificial intelligence (IJCAI)*, vol. 1, 1991, pp. 325–330.

A New Benchmark for the Energy-Constrained Fastest Path with Waypoints Problem

François Lucas[a], Christian Joubert[a], Christophe Guettier[b], Patrick Siarry[c], Arnaud de La Fortelle[a]

[a] *Mines ParisTech, CAOR Robotic Centre*
60 boulevard Saint Michel, 75272 Paris, France
{francois.lucas,arnaud.de_la_fortelle}@mines-paristech.fr
christian.joubert@polytechnique.org

[b] *Sagem Defense & Security*
100 avenue de Paris, 91344 Massy Cedex, France
christophe.guettier@sagem.com

[c] *Univ. of Paris Est-Créteil Val de Marne, LiSSi (EA 3956)*
61, avenue du Général de Gaulle, 94010 Créteil, France
siarry@u-pec.fr

Abstract

This paper presents a new benchmark for the Energy-constrained Fastest Path with Waypoints (EFP+W) problem. The goal of this specific problem is to minimize the travel time of a vehicle through a road network while meeting its energy limitations and ensuring the passage through a set of waypoints, whose order is not previously defined. This problem is NP-hard and can be reduced to several well-known problems of the Operations Research (OR) literature, that have never been considered together. A formal definition of the problem and a general description of the benchmark instances are presented in the paper. An overview of the best current solving approach and a summary of obtained results are also given. Instances can be found online at [9].

Introduction

This paper presents a new benchmark intended for the Operations Research field. It describes the Energy-constrained Fastest Path with Waypoints (EFP+W) problem, where a path of minimum traversal time must be found in a graph while taking into account the vehicle consumption and mandatory waypoints to pass through. Defined as a discrete problem, EFP+W is a NP-hard problem as it can be reduced to two well-known issues of the literature: the *Constrained Shortest Path Problem* (CSPP) and the Traveling Salesman Problem (TSP), that are both NP-hard.

This problem can be applied to both civilian and military applications. On the civilian side, share taxis management systems or single delivery vehicle planning can be mentioned. On the military side, missions consisting in rescuing

nationals abroad can also be modeled as a EFP+W. In both cases, the respect of energy constraints is essential (for ecological and economical reasons in the first examples, for tactical reasons in the latter). Moreover, in those applications, planning can be subject to change. For this reason, the focus is put on finding fast solving strategies to allow embedded systems to determine new solutions in a short amount of time.

The challenge of this benchmark is to find an efficient approach, in terms of computation time, able to jointly the different aspects of the problem. The EFP+W problem is formally defined in section 1. Section 2 presents how the EFP+W problem can be reduced to CSPP and TSP, and develops a state of the art for each of those well-known problems. Section 3 describes the instances that are currently published on-line and summarizes the assumptions that were made for modeling. Section 4 briefly presents the approach and performance results of the best current solver. Finally, section 5 presents the process to follow to submit your own results and compare your approach to the existing one.

1. The problem

1.1. Application example and problem difficulty

The problem presented below aims at developing an optimal path planner for military vehicles in mission. We consider the case of a single vehicle that must reach a final location from its current one, and pass through intermediate mandatory locations (also called *objectives* in the following). The problem is to find the route that minimizes the overall time duration, taking into account the maximal speed of the vehicle and its available amount of energy.

As an example, let us consider the problem of evacuating nationals from a country threatened by a civil war. Figure 1 illustrates a possible scenario of this problem. The initial location S of the vehicle is the entry point of the rescue zone. The objectives are the locations of nationals to rescue, marked by the R symbols. The final location G of the path is the extraction point (here a seaport). The environment is represented as a directed graph, where vertices are geographical locations and edges are possible progression axes, i.e. roads or open terrain. Edges are also given a *consumption factor* to model the "difficulty" for a vehicle to cross them (uphill road, soft ground...). The challenge is to find the route that minimizes the time needed to pick up all nationals and bring them to the extraction point while ensuring that the vehicle will not run out of fuel (or exceed a limited amount of fuel for the mission, as shown in the figure).

The complexity of this problem is introduced by the absence of priority order over waypoints. Thus, the number of possible combinations is potentially factorial with the number of waypoints. Complexity also arises when the consumption constraint is strong. As consumption impacts on the vehicle speed, and as it may vary over edges due to the consumption factor, the fastest path may not be the shortest one.

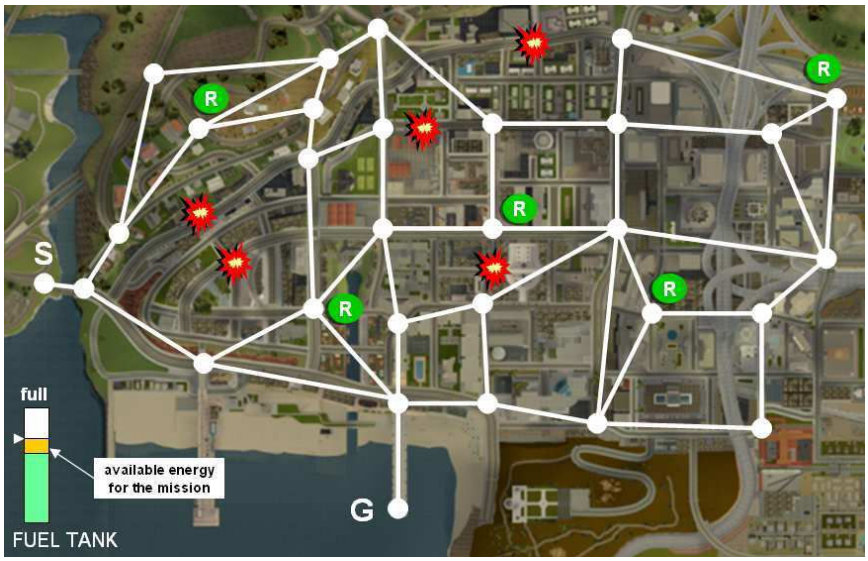


Figure 1: Illustration of a possible rescue mission scenario. S is the entry point of the rescue zone. R symbols designate nationals to rescue. G is the extraction point.

1.2. Assumptions

In our framework, the following assumptions are considered:

- the environment graph G defined below is connected ;
- if an edge from a vertex v_1 to a vertex v_2 exists, then an edge from v_2 to v_1 necessarily exists (but the consumption factor may be different) ;
- a vehicle can only pass **one time** through a node of G (and consequently an edge) ;
- the vehicle speed is supposed **constant** over along an edge ;
- for simplification, edge distances are determined as the euclidean distance between source and destination nodes (characterized by a (x, y) coordinate couple).

1.3. Problem formalization

1.3.1. Problem and cost function

The environment is represented as a graph $G = (V, E)$ where V is the set of vertices and E is the set of edges. $v_s \in V$ and $v_g \in V \setminus \{v_s\}$ designate the initial and desired final location of the vehicle respectively. We also define $V_m \subseteq V \setminus \{v_s, v_g\}$ the set of mandatory waypoints. To each edge $e \in E$ is associated a distance value $d(e)$ and a consumption factor $m(e)$.

The EFP+W problem can be expressed as a constrained minimization problem (problem formulation is voluntarily close to that of [1]):

$$\min_{p \in P'(v_s, v_g, V_m)} \sum_{e \in p} t(e) \tag{1}$$

where

$$P'(v_s, v_g, V_m) = \{p \in P(v_s, v_g, V_m) \mid \sum_{e \in p} c(e) \leq R_0\} \tag{2}$$

$P(v_s, v_g, V_m)$ denotes the set of paths from v_s to v_g, visiting all V_m nodes. $c(e)$ represents the consumption variable of an edge e, described in section 1.3.2. R_0 is the available energy resource for the mission. $t(e)$ denotes the time variable of an edge e, defined as:

$$\forall e \in E_p,\ t(e) = \frac{d(e)}{s(e)} \tag{3}$$

where $s(e)$ is the vehicle speed along the edge e. $s(e)$ is defined in $[S_{min}, S_{max}]$ range, S_{min} and S_{max} being an input of the problem.

A solution to the EFP+W is a path p with a discrete time value associated to each edge of p.

1.3.2. Energy constraints

The model defined below is simplistic in comparison to a real energy consumption law which involves a much larger number of parameters. However, it allows to roughly express what could be a vehicle consumption at constant speed, and to insert a non-linear capacity function in our problem.

The consumption model is expressed using an energy flow (energy consumed by unit of time). This energy flow is composed of two parameters :

- a constant flow value f_0, corresponding to the energy flow consumed when the vehicle is stationary (zero in case of an electrical engine) ;
- a instant flow value $f_i(e)$ corresponding to the energy flow consumed over edge e to move the vehicle.

For each edge $e \in p$, the energy flow $f(e)$ consumed is given by (unit conversions do not appear in the equations):

$$f(e) = f_0 + f_i(e) \tag{4}$$

where

$$f_i(e) = m(e).\frac{s(e)^2}{Q} \tag{5}$$

where Q is an utility constant. $m(e)$ is the consumption factor, corresponding to a percentage value, where 1 is a "flat road" (a downhill and an uphill road will have a consumption factor inferior and superior to 1 respectively). Each edge flow can be integrated over time to determine the effective energy consumption $c(e)$ of the vehicle needed to cross e:

$$\forall e \in p, c(e) = f(e).t(e) \tag{6}$$

Figure 2 shows a graphic representation of $c(e)$ depending on f_0 and $s(e)$ values.

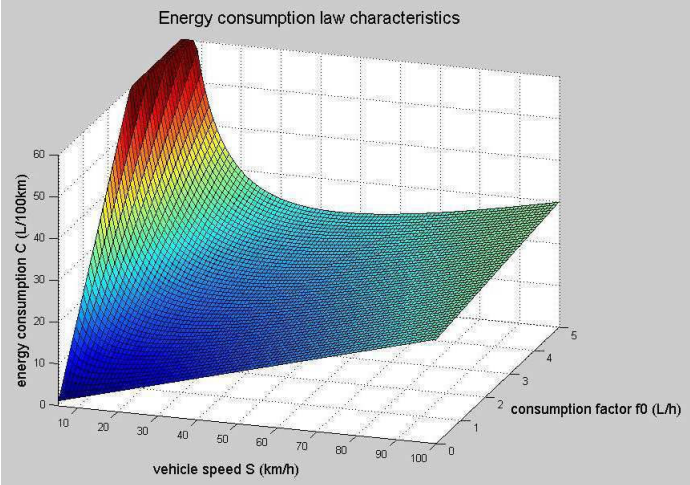


Figure 2: Energy consumption law (in L/100km), as a function of the constant factor f_0 and the vehicle speed s. for $Q = 400$ and $m = 100$, from definition in section 1.3.2. The s axis begins at a value of 5km/h, for readibility.

2. State of the Art

The EFP+W Problem is a NP-hard problem, as it can be reduced to two well-known issues of the literature: the *Constrained Shortest Path Problem* (CSPP) (formally defined as the *Shortest Weight-Constrained Path* in [12]) and the Traveling Salesman Problem (TSP) (formally defined as the *Minimum Traveling Salesperson* in [12]).

2.1. The Constrained Shortest Path Problem (CSPP)

It has to be noticed that the CSPP can be found under various other names in the literature: *Restricted Shortest Path*, *Minimum Cost Restricted Time Combinatorial Optimization*, *QoS Routing*, *Delay Constrained Least Cost Path* or *Weight-Constrained Shortest Path*.

A CSPP instance is defined as a graph $G = (V, E)$. Each edge $e \in E$ is given a *length* and *weight* value (also called *cost* and *delay* in telecommunications issues, respectively). Solving the CSPP consists in finding the minimum-length path $p^\star$ in G such that the sum of the weight values along $p^\star$ is lower than a given weight bound W. Formally:

$$p^\star = \min_{p=\in P'(s,g)} \sum_{e \in p} l(e) \tag{7}$$

with

$$P'(s,g) = \{\ p \in P(s,g) \mid \sum_{e \in p} w(e) \leq W\ \} \tag{8}$$

$P(s, g)$ is the set of paths in G from s to g. $l(e)$ and $w(e)$ are the length and weight values associated to an edge e, respectively.

The CSPP can be reduced in polynomial time to a EFP+W expression: considering the particular case in which $V_m = \emptyset$ and $S_{min} = S_{max}$, each edge $e \in E$ has a unique $\langle time, energy \rangle$ value couple. Consequently, a EFP+W instance is directly equivalent to a CSPP instance in which the time function t corresponds to the length function l, and the consumption function c corresponds to the weight function w. As the CSPP is a NP-hard problem, it leads that EFP+W is also NP-hard.

Polynomial approximation algorithms have been developed in the literature to fastly solve the CSPP. Two main recent approaches can be cited: geometric approaches based on lagrangian relaxation such as LARAC [1] and its derivatives [17], and algebraic approaches initiated by [13] on fully-polynomial time approximation schemes (FPTAS) and improved later [4, 5]. Approximation using the Simplex method [16] and cost-based filtering techniques [11] may also be mentioned.

2.2. The Traveling Salesman Problem (TSP)

A TSP instance is defined as a set C of *cities* and a distance value for each pair of cities in C. The solution to the TSP is a minimum-length tour, i.e. the shortest hamiltonian circuit in the complete graph formed by C. Formally, a tour is a permutation $\pi : [1..m] \rightarrow [1..m]$, and the solution to the TSP obtained by minimizing:

$$\pi^\star = \min_{\pi \in \Pi} \{ \ d(\{c_{\pi(m)}, c_{\pi(1)}\}) + \sum_{i=1}^{m-1} d(\{c_{\pi(i)}, c_{\pi(i+1)}\}) \ \} \tag{9}$$

where Π is the set of tours in C.

A TSP can be reduced in polynomial time to a EFP+W expression, by leaving aside energy constraints. The problem of minimizing time while passing through all mandatory waypoints amounts to searching for the minimum hamiltonian path from start to goal location in a complete graph formed by $C = V_m \cup \{v_s, v_g\}$ nodes. Distance values between cities can be computed using a shortest path algorithm. It leads to a TSP-like formulation, where a solution is covering path of C, i.e. a permutation $\pi : [1..m-1] \rightarrow [2..m]$ in which city $1 = v_s$ and city $m = v_g$. Thus the solution is obtained by minimizing:

$$\pi^\star = \min_{\pi \in \Pi} \{ \ \sum_{i=1}^{m-1} d(\{c_{\pi(i)}, c_{\pi(i+1)}\}) \ \} \tag{10}$$

This slightly modified TSP formulation remains NP-hard in terms of solving complexity.

Best existing exact solvers are heuristic-based branch and bound approaches, coupled with cut methods from linear programming [2]. Stochastic approaches such as metaheuristics [15, 10, 8] are also very effective in practice and can rapidly lead to a near-optimal solution.

3. Benchmark instances

3.1. Website

The benchmark instances described below are available online [9]. This webpage:

- summarizes the main information about the benchmark and problem definition ;
- describes the data structure (XML files) of benchmark instances and results ;
- shows the scores of current solvers ;
- allows any visitor to download the benchmark instances, and to submit his own results.

3.2. Instance description

The benchmarks that are presented now are available on-line under the name *Test Set 1*. The consumption model parameters are detailed in section 3.2.1. *Test Set 1* is composed of three benchmarks, which represent three different graphs inspired from real-case scenarios. Each contains six different problem instances of the same graph (see section 3.2.3).

3.2.1. Vehicle model parameters

The three benchmarks use the same consumption model parameters:

- $S_{min} = 0$km/h ;
- $S_{max} = 60$km/h ;
- $f_0 = 1$L/h ;
- $Q = 400$.

Q value was set to 400 to obtain a consumption equivalent to 15L/100km at a speed $S_{max} = 60$km/h (without considering constant consumption involved by f_0). f_0 was set to 1L/h to induce an infinite consumption law at null speed (see figure 2).

The initial resource factor R_0 is specifically defined for each instance of the benchmarks. Values were selected in $]R_s^\star, R_w^\star[$, where $R_s^\star$ is the energy resource of the optimal solution when the consumption constraints are relaxed (the vehicle then necessarily moves at S_{max} speed), and $R_w^\star$ is the minimum resource consumption for the mission (energy is minimized instead of time). Each value R_0 was set so that no solution can be found at highest speed (i.e. $s = S_{max}$ for every edge of the solution), that would make the problem much easier. Indeed, the objective was to obtain very constrained problems in order to increase solving challenges.

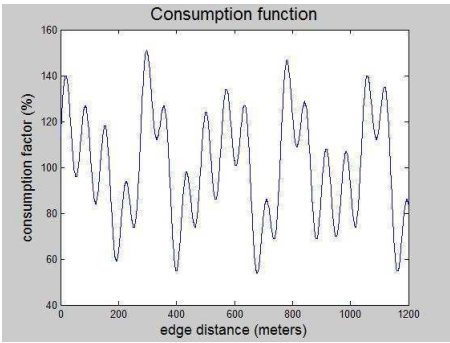


Figure 3: Generation function of edge consumption factor, function of its distance value.

3.2.2. Consumption factor value generation

The consumption factor values were generated with a law based on a combination of three basic trigonometric functions. Considering an edge $e \in E$:

$$f_1(e) = 100 + 70.(\sin(\frac{d(e)}{40}))$$

$$f_2(e) = 100 + 40.(\cos(\frac{d(e)}{25}))$$

$$f_3(e = 100 + 60.(\sin(\frac{d(e)}{11}))$$

Finally, $m(e)$ is expressed as the average value of f_1, f_2 and f_3:

$$m(e) = \frac{f_1(e) + f_2(e) + f_3(e)}{3}$$

Figure 3 shows a plot of the m coefficient for a distance in [0;1200] meters range.

The $m(e)$ values obtained, only depending on edge distance, are at no point realistic. It can be noticed that the three subfunctions $f_1(e)$, $f_2(e)$ and $f_3(e)$ have the same offset, in order to remain centered on the "neutral" consumption multiplier value 100. Period and amplitude factors have been chosen empirically in order to induce high variations.

3.2.3. Benchmark instances

The three benchmarks proposed below were simply named *Benchmark 1*, *Benchmark 2* and *Benchmark 3*. Each represent a different environment. Over those three benchmarks, six distinct instances have been created. Each instance is given specific v_s, v_g, V_m and R_0 parameters. Tables 1, 2 and 3 describe the chosen values for those parameters, on each instance.

Benchmark 1 : 23 vertices, 76 edges				
Mission ID	v_s	v_g	V_m	R_0
mission_1_1	7	11	$\emptyset$	475
mission_1_2	12	18	$\emptyset$	400
mission_1_3	13	5	$\{3, 8, 16\}$	700
mission_1_4	6	14	$\{7, 10, 15\}$	450
mission_1_5	11	8	$\{0, 4, 16, 19, 22\}$	800
mission_1_6	21	17	$\{2, 8, 10, 13, 18\}$	950

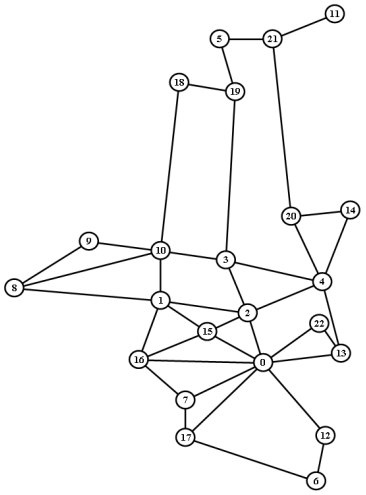


Table 1: *Benchmark 1* instance description.

Benchmark 2 : 22 vertices, 74 edges				
Mission ID	v_s	v_g	V_m	R_0
mission_2_1	1	18	$\emptyset$	750
mission_2_2	11	8	$\emptyset$	820
mission_2_3	9	21	$\{3, 4, 20\}$	600
mission_2_4	15	8	$\{3, 20, 21\}$	705
mission_2_5	11	9	$\{3, 7, 10, 14, 20\}$	1025
mission_2_6	15	1	$\{3, 5, 11, 13, 17\}$	1250

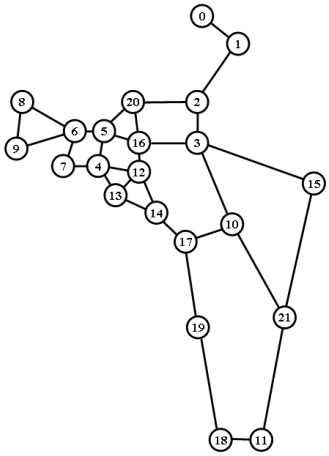


Table 2: *Benchmark 2* instance description.

Benchmark 3 : 22 vertices, 68 edges				
Mission ID	v_s	v_g	V_m	R_0
mission_3_1	21	2	$\emptyset$	1975
mission_3_2	6	16	$\emptyset$	2000
mission_3_3	0	21	$\{4, 14, 16\}$	2200
mission_3_4	15	18	$\{2, 5, 12\}$	2725
mission_3_5	20	16	$\{1, 2, 10, 14, 19\}$	3600
mission_3_6	7	19	$\{4, 6, 11, 14, 17\}$	3850

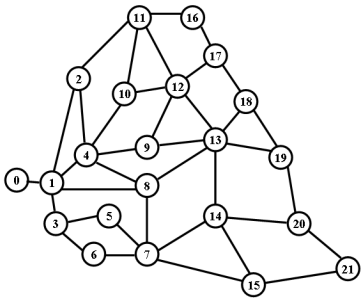


Table 3: *Benchmark 3* instance description.

3.3. Instance data structure

Instances have been specified using an XML format so that users may use it on any software architecture. Moreover, the XML schema named $Mission.xsd$ is available on-line (website URL is given in section 3.1), so that automatic parsers can be used to extract information. The root element has the type $tMission$.

4. Solving the EFP+W

To solve the EFP+W problem, a hybrid solver using metaheuristics and Constraint Programming (CP) was developed. We used previous work [7, 6] that we extended to address the non-linear energy constraints. The CP model and the approach are presented in the following of this section.

4.1. The CP solver

To solve this problem, a CP solver has been defined in Prolog language. CP was chosen for several reasons:

- the completeness of the search, allowing to find the optimal solution and to obtain an optimality proof ;
- the CP language semantics, allowing high level expression of complex problems ;
- the efficient arc-consistency algorithms, propagating problem constraints and therefore allowing to reduce the problem size by removing inconsistent value settings from the state space ;
- the ability to elaborate search strategies, in combination with a *branch & bound* algorithm, to efficiently solve the problem with *anytime* properties.

4.1.1. Problem modeling

The problem expressed in 1.3 was modeled as follows. A path in the environment graph $G = (V, E)$ is formally defined using a flow model through Kirchhoff laws. To do so, flow variables are associated to each edge $e \in E$. A flow is thus described as a unitary value flow:

$$\sum_{e \in \omega^+(v_s)} \phi_e = \sum_{e \in \omega^-(v_g)} \phi_e = 1 \tag{11}$$

$$\sum_{e \in \omega^-(v_s)} \phi_e = \sum_{e \in \omega^+(v_g)} \phi_e = 0 \tag{12}$$

$$\forall v \in V \setminus \{v_s, v_g\}, \sum_{e \in \omega^+(v)} \phi_e = \sum_{e \in \omega^-(v)} \phi_e \leq 1 \tag{13}$$

$$\forall v \in V_m, \sum_{e \in \omega^+(v)} \phi_e = \sum_{e \in \omega^-(v)} \phi_e = 1 \tag{14}$$

and ϕ_e variables are defined in $\{0, 1\}$ range. $\omega^+(v) \subset E$ (resp. $\omega^-(v) \subset E$) represents the set of edges that leave (resp. reach) a node v.

Equations 11, 12, 13 and 14 are necessary but not sufficient conditions, because cycles may be found outside of the path. To counter this drawback, additional variables denoted by *id* are added to graph nodes in V. These variables are used to propagate precedence constraints along the path:

$$id_{v_s} = 1 \tag{15}$$

$$\forall v \in V \setminus \{v_s\}, id_v = \sum_{e=(u,v) \in \omega^+(v)} \phi_e.(id_u + 1) \geq 0 \tag{16}$$

$$id_{v_g} = 1 + \sum_{e \in E} \phi_e \tag{17}$$

and id_v variables are defined in $\{0, n\}$ range where $n = |V|$ (a vehicle can only pass once through each node, according to the problem assumptions described in section 1.2). *id* variables can be replaced by time variables on nodes (expressing the cumulated time from v_s), but the constraint propagation induced can be much more time consuming than *id* values (due to the additional propagation over variables sharing constraints with time variables).

Edge time variables are defined as follows (unit conversions do not appear in the equation):

$$\forall e \in E, t_e = \begin{cases} \frac{d_e}{s_e} \text{ if } \phi_e = 1 \\ 0 \text{ otherwise.} \end{cases} \tag{18}$$

Energy constraints are introduced by expressing consumption variables on edges (unit conversions do not appear in the equation):

$$\forall e \in E, c_e = \begin{cases} f^0.t_e + \frac{m_e.s_e^2}{Q}.t_e \text{ if } \phi_e = 1 \\ 0 \text{ otherwise.} \end{cases} \quad (19)$$

c_e belongs to $[0, R_0]$ range and is bounded by the maximum consumption value spent by the vehicle if it crosses the edge e at S_{max} speed. The following equation is used to apply the constraint on the overall energy consumption:

$$\sum_{e \in E} \phi_e.c_e \leq R_0 \quad (20)$$

It can be noticed that both equations 18 and 19 depend on vehicle speed s. To limit the number of variables involved in the model, s variables can be removed by directly expressing c as a function of time t only. Reducing the number of variables in this case has two advantages : potentially reducing computation time (as each variable is subject to constraint propagation), and limiting rounding errors which are a major issue for this kind of problems. Equations 18 and 19 are replaced by the following ones in our CP model (unit conversions do not appear in the equation):

$$\forall e \in E, c_e = \begin{cases} f^0.t_e + \frac{m_e.d_e^2}{Q.t_e} \text{ if } \phi_e = 1 \\ 0 \text{ otherwise.} \end{cases} \quad (21)$$

Equation 21 is directly constraining time variables, so that equation 18 becomes useless. However, it is necessary to reintroduce constraints on maximum speed, as follows:

$$\forall e \in E, t_e \geq \phi_e.\frac{d_e}{S_{max}} \quad (22)$$

Finally, the optimization problem is expressed as :

$$\min \sum_{e \in E} \phi_e.t_e \quad (23)$$

where ϕ and t variables are the decision variables of the problem.

4.1.2. Problem solving and the use of "probe backtracking"

The problem presented above is solved using a branch & bound approach, in which flow variables and time variables are consecutively labeled. Studies over this approach have shown that flow variable sorting has a crucial impact on solving performances. Indeed, if a good solution is found early during the search, its cost value can be used as an upper bound for the remaining evaluations. Thus, the solver is potentially able to discard more potential solutions and save evaluations. The better the solution, the better the cut. In our approach, this statement was used by setting a preprocessing step, in which a relaxation of the problem is solved. The solution to this relaxation is not used directly in an attempt prior to the search. It is used to sort flow variables, by computing

an indicator corresponding to the minimum distance between each edge and the partial solution. The backtrack process, called *probe backtracking* in the literature [14], is then configured to select in priority edges that have the lowest distance value, so that flows of the closest edges will be set positive first. The bet that is made through this strategy is that a good solution to the global problem belongs to the close area around the partial solution.

To be relevant, this search strategy must use a relaxed problem that will efficiently guide the search. Moreover, the partial solution should be computed in a very short amount of time so that the global computation time will not be impacted. Previous research [7, 6] was performed on solving a relaxation consisting in finding the shortest path passing through all mandatory waypoints using a fast Ant Colony Optimization (ACO) [10] algorithm. This approach, mixing metaheuristics and CP, showed interesting performances. However, the problem did not consider energy constraints.

Yet, inserting additive constraints can lead to poor search performance. In our problem, shortest path solution to guide the search may be ineffective, as the fastest path may not be the shortest one. To counter this limit, it is necessary to simultaneously consider path finding and additive constraints in the preprocessing step. A new approach (it will be detailed in a next article), based on LARAC algorithm [1, 17], was developed. This algorithm implements a Lagrange relaxation over a series of minimum cost path to determine the best path satisfying the constraints. We adapted this approach by replacing the initial *Dijkstra* algorithm [3] by an ACO-based algorithm so that the minimum-cost path found integrates the requirements on imposed waypoints. Moreover, to tackle the interdependence between cost and energy functions (as both depend on speed), a multigraph is generated prior to the execution of the algorithm. This multigraph is built from the initial graph in which an arc is replaced by a set of arcs, each representing a possible time value in the CP model (with the corresponding consumption value). Then, before each minimum cost path is computed in our LARAC-ACO approach, a simple graph is extracted from the multigraph. This simple graph only contains arcs of minimum reduced cost (relatively to the Lagrange relaxation) and allows to reduce the problem to a classic CSPP instance, that LARAC is able to solve.

The solution to our LARAC-ACO approach is used to guide the search through flow variables, but also time variables. Indeed, time values are extracted from the partial solution and used as upper bounds on CP time variables. After all the solutions with bound restrictions have been considered, those constraints are relaxed so that the approach remains complete.

4.2. Solution precision

As the approach described above uses Integer Programming, rounding errors may have serious impacts on solution precision (especially for propagated variables, cumulating the errors). To counter this inconvenience, the following units were used:

- time is specified in seconds ;

Benchmark	Instance	Results (ms)		
		Preproc.	Opt. (ms)	Proof (ms)
Benchmark 1	mission_1_1	4	0 (189)	218
	mission_1_2	36	16 (155)	125
	mission_1_3	468	15 (306)	546
	mission_1_4	452	0 (224)	374
	mission_1_5	1120	0 (333)	717
	mission_1_6	1174	0 (363)	328
Benchmark 2	mission_2_1	40	47 (372)	437
	mission_2_2	3	16 (384)	359
	mission_2_3	517	31 (401)	1685
	mission_2_4	521	0 (502)	1997
	mission_2_5	1053	496738 (678)	–
	mission_2_6	1194	0 (787)	–
Benchmark 3	mission_3_1	82	47 (812)	749
	mission_3_2	81	0 (665)	561
	mission_3_3	596	2060 (2276)	–
	mission_3_4	529	187 (1798)	–
	mission_3_5	1225	496973 (2653)	–
	mission_3_6	1070	497300 (1568)	–

Table 4: Results of our best current approach on described instances. *Preproc.* is the computation time of the preprocessing step, *Opt.* is the time needed to find the optimal solution, and *Proof* is the time needed to obtain the optimality proof (time at which the search finishes). A 500s timeout has been set: "–" denotes instances for which the solver failed to prove the optimality within those 500s. Values indicated in parenthesis in *Opt.T.* cells are optimal solution values (or possibly suboptimal, when proof is not obtained).

- distance is in meters ;
- remaining energy is in centiliters ;

Those choices allowed to keep satisfactory solution precision, but have a non negligeable impact on problem complexity.

4.3. Current results

The table 4 summarizes the results that were obtained with our approach, on proposed benchmark instances. It was computed on a *Samsung Q310* laptop, equipped with an *Intel Core 2 Duo P3750* microprocessor running at 2GHz, and 3GB of cache memory. Preprocessing and global solver computation times are presented separately.

Results show very efficient performance on *Benchmark 1*. On *Benchmark 2*, the four first instances are solved with the same efficiency. However, on the last two instances, our approach fails at proving the optimality of the solution. On

instance *mission_2_5*, performance is really bad as the search is still improving optimization at 496,738 ms. Actually, this failure is due to the CP model that is slightly different from the one used by the LARAC-ACO solver. In CP, a path cannot pass through a node several times, whereas the LARAC-ACO model authorizes it. Consequently, when the CP search starts, the solution that consists in setting the flows of the partial solution at 1 is discarded. At that moment, the upper bounds on time variables cannot be set as the latter do not correspond to the solution previously found. Without these upper bounds constraints, the search is constrained to evaluate each time value combination to determine the best one, as soon as a satisfactory path (in terms of flows) is found. This case also appears for *mission_3_5* and *mission_3_6* instances, where the best solution is found just before the timeout (and certainly not the optimal, as a consequence). Finally, on *mission_2_6*, *mission_3_3* and *mission_3_4* instances, the optimal solution is found almost immediately but the search is not able to prove the optimality within 500 seconds. It may be due to the large number of alternative solutions that have to be considered by the branch & bound approach: if their costs are close to the optimal solution, they may not be easily discarded.

5. Submit your own results

Result Data Format.. An XML schema has been defined to format the results of an instance. It can be found on the benchmark website (see 3.1) with the name *Planning.xsd*. The root element is defined with type *tPlanning*.

On-Line submission.. Results shall be submitted at the e-mail address specified on the benchmark website. They shall be formatted using the result data format mentioned above. After consistency check and score computation, on-line result tables will be updated.

Rating.. The result evaluation method is described in [9], in the FAQ section. This evaluation is based on the number of problems solved, the distance of solutions to the optimal one, and the time needed to solve the instances. A relative score, whose reference is Sagem solving approach (described in section 4), is then computed and used for result ranking.

Conclusion

In this paper, we presented a new benchmark for the Energy-constrained Fastest Path with Waypoints (EFP+W) problem, intended for the OR community. It consists in finding a path in a graph, passing through a set of imposed nodes, that minimizes the overall traversal duration of a vehicle. The solution must also satisfy energy constraints, whose model is described. The non-linear property of the consumption model, function of the vehicle speed and path slope, and the absence of priority between waypoints make the problem hard to address.

The results currently obtained show interesting performance on proposed instances, but suffer from the slightly different modeling between the preprocessing solver and the global solver on some instances. Moreover, the solver may have difficulties to prove the optimality within satisfactory time.

References

[1] A.Jüttner, B.Szviatovszki, I.Mécs, and Z.Rajkó. Lagrange relaxation based method for the QoS routing problem. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies (INFOCOM)*, volume 2, pages 859–868, 2001.

[2] D.L. Applegate, R.M. Bixby, V. Chvátal, and W.J. Cook. *The Traveling Salesman Problem: A Computational Study*. Princeton University Press, 2006.

[3] E.W. Dijkstra. *A Short Introduction to the Art of Programming*. T.H. Eindhoven, 1971. EWD316.

[4] D.Lorenz and D.Raz. A simple efficient approximation scheme for the restricted shortest path problem. *Operations Research Letters*, 28:213–219, 1999.

[5] F.Ergün, Rakesh K. Sinha, and Lisa Zhang. An improved FPTAS for restricted shortest path. *Information Processing Letters*, 83:287–291, 2002.

[6] F.Lucas and C.Guettier. Automatic vehicle navigation with bandwidth constraints. In *Proceedings of MILCOM'10*, San Jose, CA, USA, November 2010.

[7] F.Lucas, C.Guettier, and P.Siarry. Hybridisation of constraint solving with an ant colony algorithm for vehicle on-line path planning. In *Proceedings of ICAPS 4th Workshop on Planning and Plan Execution for Real-World Systems*, Thessaloniki, Greece, September 2009.

[8] J.Watson, C.Ross, V.Eisele, J.Denton, J.Bins, C.Guerra, D.Whitley, and A.Howe. The traveling salesrep problem, edge assembly crossover, and 2-opt. *Proceedings of Lecture Notes in Computer Science*, pages 823–832, 1998.

[9] F. Lucas and C. Joubert. `http://www-roc.inria.fr/imara/dw/caor/benchmarks/missionplanning`.

[10] M.Dorigo and L.Gambardella. Ant colony system: A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997.

[11] M.Sellmann, T.Gellermann, and R.Wright. Cost-based filtering for shorter path constraints. In *Proceedings of CP03*, pages 694–708, 2003.

[12] P.Crescenzi and V.Kann. A Compendium of NP Optimization Problems. `http://www.csc.kth.se/~viggo/wwwcompendium`, 2005.

[13] R.Hassin. Approximation schemes for the restricted shortest path problem. *Mathematics of Operations Research*, 17(2):36–42, 1992.

[14] H. El Sakkout and M. Wallace. Probe backtrack search for minimal perturbation in dynamic scheduling. *Constraints*, 5(4):359–388, 2000.

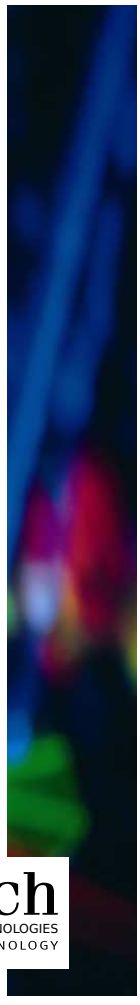
[15] V.Černý. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[16] Y.Xiao, K.Thulasiraman, and G.Xue. QoS routing in communication networks: Approximation algorithms based on the primal simplex method of linear programming. *IEEE Transactions on Computer*, 55(7):815–829, 2006.

[17] Y.Xiao, K.Thulasiraman, G.Xue, and A.Jüttner. The constrained shortest path problem: Algorithmic approaches and an algebraic study with generalization. *AKCE International Journal of Graphs and Combinatorics*, 2(2):63–86, 2005.

Des métaheuristiques pour le guidage d'un solveur de contraintes dédié à la planification automatisée d'itinéraires de véhicules

Résumé : Cette thèse, réalisée en collaboration avec Sagem Défense Sécurité, porte sur l'élaboration d'une stratégie de recherche efficace pour la résolution de problèmes de planification d'itinéraires de véhicules avec points de passage obligatoires. Ce document propose une approche originale, hybridant un algorithme de colonies de fourmis avec un solveur de Programmation par Contraintes existant. Le premier est utilisé pour résoudre rapidement une version relaxée du problème. La solution partielle obtenue est alors employée pour guider la recherche du second, par le biais d'une méthode de sonde, vers les zones les plus prometteuses de l'espace d'état. Cette approche permet ainsi de combiner la rapidité des métaheuristiques et la complétude de la programmation par contraintes. Dans un cadre applicatif, nous considérons en particulier les problèmes de planification avec contraintes de points de passage et de *capacité* (énergie, bande passante radio) appliquées au véhicule. Pour la gestion de capacités additives, l'algorithme de colonies de fourmis est couplé à une approche de relaxation lagrangienne en nombre entiers. Nous montrons expérimentalement que ces approches satisfont les exigences pour une utilisation du planificateur dans un cadre embarqué.
Mots clés : planification d'itinéraires, optimisation combinatoire, métaheuristiques, colonies de fourmis, programmation par contraintes, sonde, stratégie de recherche, relaxation lagrangienne



Metaheuristics for the guidance of a constraint solver dedicated to automated vehicle path planning

Abstract: This thesis, led in collaboration with Sagem Defence & Security, focuses on defining an efficient search strategy to solve vehicle path planning problems with mandatory waypoints. This document proposes an original approach, mixing an ant colony algorithm with an existing Constraint Programming solver. The former is used to fastly solve a relaxed version of the problem. The partial solution returned is then employed to guide the search of the latter, through a Probe Backtrack mechanism, towards the most promising areas of the state space. This approach allows to combine the metaheuristics solving fastness and the Constraint Programming completeness. In an applicative framework, this work addresses planning problems in which *capacity* constraints (energy, radio bandwidth) are applied to vehicles. To efficiently manage additive capacities, the ant colony algorithm is coupled with an integer lagrangian relaxation approach. We experimentally show that these approaches meet the requirements for an on-line use of the planner.
Keywords: path planning, combinatorial optimization, metaheuristics, ant colony, constraint programming, probe, search strategy, lagrangian relaxation

MINES ParisTech

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY