Méthodes de recherche arborescentes. Application à la résolution de problèmes d'ordonnancement et au calcul d'itinéraires multimodaux

Marie-José Huguet

Université de Toulouse

Habilitation à Diriger des Recherches

préparée au **Laboratoire d'Analyse et d'Architecture des Systèmes**

présentée à l'**Université Paul Sabatier**

par

Marie-José HUGUET

Maître de Conférences à l'Institut National des Sciences Appliquées de Toulouse

le 20 mai 2011

Méthodes de recherche arborescentes.
Application à la résolution de problèmes d'ordonnancement et au calcul d'itinéraires multimodaux

Jury

M.	Philippe Jégou, Professeur des Universités,	Rapporteur
M.	Eric Pinson, Professeur des Universités,	Rapporteur
M.	Francis Sourd, Chargé de Recherche CNRS (HDR), SCNF,	Rapporteur
Mme.	Claudette Cayrol, Professeur des Universités,	Examinateur
M.	Emmanuel Néron, Professeur des Universités,	Examinateur
M.	Pierre Lopez, Chargé de Recherche CNRS (HDR),	Directeur des recherches

Avant-Propos

J'exprime toute ma gratitude à Philippe Jégou, Eric Pinson et Francis Sourd d'avoir accepté d'être rapporteurs de mon habilitation à diriger des recherches. Je remercie également Claudette Cayrol et Emmanuel Néron pour leur participation à mon jury.

Je tiens à remercier particulièrement Pierre Lopez, à la fois pour son rôle de responsable du groupe Modélisation Optimisation et Gestion Intégrée de Systèmes d'Activités (MOGISA) mais surtout pour la collaboration menée tout au long de ces années, qui m'a permis d'aboutir à ce mémoire d'habilitation et qui je l'espère se poursuivra.

J'adresse de chaleureux remerciements aux membres du groupe MOGISA, anciens ou récents, pour les nombreux échanges de points de vue et pour la bonne ambiance apportée au quotidien. Je remercie plus particulèrement Christian Artigues, Cyril Briand, Jacques Erschler et Patrick Esquirol, avec qui j'ai eu beaucoup de plaisir à travailler ; sans oublier Christèle Mouclier dont l'efficacité simplifie de si nombreuses tâches. Un grand merci à Thierry Vidal de l'ENI de Tarbes pour toutes les discussions que nous avons eues, des pieds des Pyrénées à Toulouse.

Une grande partie des travaux présentés dans ce mémoire a été développée grâce aux doctorant(e)s que j'ai co-encadré(e)s et que je remercie.

J'exprime toute ma reconnaissance à mes collègues de l'INSA qui m'ont si chaleureusement incitée et encouragée à soutenir mon habilitation.

J'adresse mes pensées à Jean-Etienne Doucet, qui m'a fait découvrir l'informatique lorsque j'étais qu'étudiante et avec qui j'ai eu la chance d'enseigner par la suite.

Pour conclure, je souhaite remercier mon conjoint Jérôme et nos trois enfants, Arthur, Félix et Adèle, pour cette belle vie de tous les jours ; mes sœurs Cathy et Sylviane sur qui je sais pouvoir toujours compter ; ma mère et mon frère Jean-François qui sont présents malgré la distance ; mon père qui aurait été si fier d'assister à la présentation de mon travail.

Table des matières

Avant-Propos iii
Table des matières v

Introduction générale et positionnement des travaux de recherche 1

1 Méthodes à divergences 5
1.1 Généralités sur les méthodes de résolution arborescente 5
1.1.1 Problématique 5
1.1.2 Heuristiques d'instanciation 6
1.1.3 Principe général de la méthode avec retours-arrière chronologiques . 7
1.1.4 Propagation de contraintes 8
1.1.5 Retours-arrière intelligent et apprentissage 9
1.1.6 Redémarrage 10
1.1.7 Principe général des méthodes à divergences 10
1.2 Contributions apportées 17
1.2.1 Contributions pour les méthodes à divergences 17
1.2.2 Heuristiques basées sur la pondération de variables ou de valeurs . . 21
1.2.3 Méthodes à divergences pour l'optimisation 21
1.3 Applications à résolution de problèmes d'ordonnancement 22
1.3.1 Adaptations de la méthode CDDS pour les problèmes de flowshop hybride 23
1.3.2 Adaptation de la méthode CDDS pour les problèmes de jobshop flexible 26
1.3.3 Apport des heuristiques à pondération pour la résolution de jobshop avec contraintes de délais 30
1.4 Evaluation de l'impact des modes de comptage des divergences 31
1.5 Applications à la résolution de CSP 33
1.5.1 Apport des heuristiques à pondération pour la résolution de problèmes de car-sequencing 33
1.5.2 Apport des heuristiques à pondération pour la résolution de CSP aléatoires 35
1.6 Conclusion 36

2 Propagation de contraintes pour l'ordonnancement disjonctif 37
2.1 Contexte 37
2.2 Méthodes de résolution 38
2.3 Les problèmes de satisfaction de contraintes temporelles 39
2.4 Propagation de contraintes pour l'ordonnancement 42
2.4.1 Modélisation par contraintes de problèmes d'ordonnancement 42
2.4.2 Graphes utilisés pour la représentation de problèmes d'ordonnancement 43
2.4.3 Propagation de contraintes temporelles 45
2.4.4 Propagation de contraintes de ressources 45

2.5 Contributions 48
2.5.1 Approche temporelle des problèmes d'ordonnancement et d'affectation 48
2.5.2 Extension de propagation de contraintes ressources 49
2.5.3 Résultats expérimentaux 53
2.6 Conclusion 55

3 Algorithmes de recherche de plus courts-chemins multimodaux bi-objectif 57
3.1 Introduction 57
3.2 Les algorithmes de plus courts chemins 58
3.2.1 Cas des réseaux de transport monomodaux statiques 58
3.2.2 Cas des réseaux de transport monomodaux dépendants du temps . . 59
3.2.3 Cas des réseaux de transport multimodaux 60
3.3 Contributions apportées 62
3.3.1 Définition du problème 62
3.3.2 Règles de dominance 63
3.3.3 Algorithmes proposés 65
3.3.4 Approche bidirectionnelle 68
3.3.5 Variantes A^* 69
3.4 Conclusion 70

4 Bilan et Projet de Recherche 73
4.1 Bilan 73
4.2 Projet de recherche 75
4.2.1 Méthodes arborescentes 75
4.2.2 Problèmes d'ordonnancement et de transport 76

Bibliographie 79

Introduction générale et positionnement des travaux de recherche

Les travaux présentés dans ce mémoire ont été réalisés au LAAS-CNRS au sein des groupes OCSD (Organisation et Conduite de Systèmes Discrets) puis MOGISA (Modélisation, Optimisation et Gestion Intégrée de Systèmes d'Activités) créé en 2003 à partir d'un thème de recherche du groupe OCSD.
Les activités du groupe MOGISA se situent au carrefour d'expertises en Recherche Opérationnelle (plus précisément en optimisation combinatoire) et en Intelligence Artificielle (dans le domaine des problèmes de satisfaction de contraintes). Les travaux de recherche du groupe se concentrent sur des problèmes d'optimisation ou de décision dans lesquels des activités doivent être réalisées et sont soumises à des contraintes temporelles ainsi qu'à des contraintes liées à l'utilisation et la disponibilité de ressources.
Ces travaux s'intéressent plus spécifiquement à la modélisation et à la résolution de problèmes d'ordonnancement, de transport et de planification.
Les domaines d'application se situent dans les industries de production de biens (ateliers), dans les sociétés de services (transports, télécoms, soins, entretien, etc.) ou dans les administrations (planification de projets, d'emplois du temps, affectation de personnels, etc.). Une caractéristique commune et fréquente de ces domaines est leur forte composante socio-technique, l'homme restant au centre de ces systèmes d'activités avec souvent une multiplicité de rôles.

Au sein du groupe MOGISA, nous nous sommes intéressés au développement de méthodes de résolution de problèmes combinatoires, appelées méthodes de recherche, dont le but est d'explorer un graphe d'états pour déterminer un chemin allant d'un état initial du problème vers un état final en minimisant un cout (temps d'exécution ou fonction objectif par exemple).

Lorsque l'exploration d'un graphe d'états développe une arborescence de recherche on parle de méthode arborescente. Dans le contexte que nous avons considéré pour la résolution de problèmes d'ordonnancement, chaque nœud de l'arborescence (ie. chaque état) représente une instanciation partielle, c'est-à-dire une instanciation portant sur un sous-ensemble des variables du problème et les feuilles (ie. les états finaux) correspondent à des instanciations dites complètes dans le sens où elles concernent l'ensemble des variables.
Les méthodes arborescentes s'appuient sur une stratégie d'instanciation des variables du problème, appelée heuristique d'instanciation, permettant une instanciation progressive de l'ensemble des variables du problème et, pour les méthodes considérées dans ce mémoire, sur une stratégie de parcours en profondeur d'abord. En cas d'échec lors de l'exploration d'une branche de l'arborescence, elles effectuent un retour-arrière pour développer l'exploration d'une nouvelle branche.

Le développement de l'arborescence peut être complet, et dans ce cas, garantir l'optimalité des solutions trouvées ou l'absence de solutions, il peut également être incomplet et se limiter à l'obtention d'une solution que l'on espère de bonne qualité vis-à-vis d'une fonction objectif.
Les deux principes généraux guidant le développement d'une méthode arborescente sont tout d'abord d'éviter une énumération de l'ensemble de l'espace de recherche et d'autre part d'éviter les redondances dans le parcours lors des itérations successives.
Pour cela, les méthodes arborescentes peuvent intégrer différentes techniques, telles que de la propagation de contraintes ou des calculs de bornes sur la valeur de l'objectif, permettant d'anticiper l'inutilité de certains parcours et d'élaguer ainsi a priori des parties de l'arborescence. D'autres techniques, basées par exemple sur des retours-arrière intelligents ou de l'apprentissage, permettent quant à elles d'analyser ou de mémoriser a posteriori les raisons des échecs rencontrés afin de limiter les redondances du parcours.
Les heuristiques d'instanciation conditionnent également l'efficacité des méthodes arborescentes en guidant la recherche vers des zones succeptibles de contenir une solution admissible ou de bonnes solutions.
Pour la résolution de problèmes d'optimisation via une méthode arborescente des stratégies de parcours doivent par ailleurs être mises en œuvre afin d'assurer un bon compromis entre intensification et diversification de la recherche.

Nous avons également abordé la résolution de problèmes de plus courts chemins dans des réseaux de transport. Pour ces problèmes, un état initial du graphe d'états correspond au point origine d'un itinéraire, un état final correspond au point de destination et un état quelconque à un point sur le réseau de transport. Pour ce type de problème, le résultat attendu d'une méthode de recherche est le chemin permettant d'aller de l'origine à la destination (avec éventuellement un cout associé).
Dans ce contexte, afin d'améliorer leur efficacité, les méthodes d'exploration peuvent s'appuyer sur des heuristiques permettant de guider le choix du prochain nœud à visiter en fonction de son positionnement vis-à-vis de l'état final (on parle alors de recherche orientée ou A^*) ou sur des approches bidirectionnelles permettant d'explorer l'espace d'états en partant à la fois de l'état initial et de l'état final.
Les méthodes d'exploration du graphe d'états doivent, de plus, éviter de générer des redondances dans l'arborescence de recherche en utilisant par exemple un marquage des nœuds déjà visités.

Les travaux que nous avons menés ont porté sur la proposition de méthodes arborescentes et de méthodes de propagation de contraintes appliquées à la résolution de différents problèmes d'ordonnancement ainsi que sur le développement de modèles et de méthodes pour la résolution de problèmes de calculs d'itinéraires issus d'un contexte industriel.
Nos apports spécifiques concernent :

- le développement de méthodes de recherche originales basées sur le concept de divergences. Les méthodes à divergences proposées dans un contexte d'optimisation, pour la résolution de problèmes d'ordonnancement flexibles, réalisent un parcours incomplet de l'arborescence de recherche. Elles utilisent des techniques de propagation simples et des calculs de bornes pour élaguer l'arborescence ou des stratégies de parcours permettant de limiter l'application des divergences. Les méthodes à divergences proposées dans un contexte de décision combinent des techniques de propagation, différents modes d'application ou de comptage des divergences et des heuristiques d'instanciation intégrant des informations sur les échecs rencontrés.
- le développement de nouvelles techniques de propagation de contraintes de ressources pour la résolution de problèmes d'ordonnancement combinés à des problèmes d'af-

fectation ou prenant en compte des contraintes temporelles généralisées comme des durées variables, des gammes quelconques ou des délais entre tâches.

· la proposition de modèles et d'algorithmes pour la résolution d'un problème de calcul d'itinéraires multimodaux avec notamment l'extension de techniques d'accélération des méthodes de recherche telle que $A*$ ou les approches bidirectionnelles au cas multimodal.

Ce rapport correspond à la synthèse scientifique de mon mémoire d'habilitation. Il est constitué de quatre chapitres. Dans le chapitre 1, nous présentons les différentes composantes d'une méthode de recherche avant de focaliser sur le principe des méthodes à divergences. Nous donnons ensuite les différentes contributions que nous avons apportées que ce soit pour l'amélioration des méthodes à divergences ou sur la proposition de nouvelles heuristiques dynamiques d'instanciation se basant sur les échecs rencontrées lors de précédentes étapes de résolution. Enfin, nous exposons les applications de ces propositions pour la résolution de différents problèmes d'ordonnancement ou pour la résolution de problèmes de satisfaction de contraintes aléatoires.
Dans, le chapitre 2 nous exposons les principes des techniques de propagation de contraintes pour l'ordonnancement et nous présentons les contributions apportées en termes de nouvelles techniques de propagation de contraintes de ressources et l'apport de ces nouvelles techniques pour la résolution de différents problèmes d'ordonnancement.
Le chapitre 3 présente ensuite les problèmes de calcul d'itinéraires auxquels nous nous sommes intéressés et en particulier le problème de calcul de plus courts chemins multimodaux bi-objectifs pour lequel nous avons adapté certaines techniques d'accélération des méthodes de recherche au cas de réseaux de transport multimodaux.
Le chapitre 4 propose un bilan de mes activités de recherche ainsi qu'un ensemble de perspectives.

Chapitre 1

Méthodes à divergences

1.1 Généralités sur les méthodes de résolution arborescente

1.1.1 Problématique

Nous nous sommes intéressés à la résolution de problèmes combinatoires définis dans le formalisme des problèmes de satisfaction de contraintes (*constraint satisfaction problem* ou *CSP*) par un ensemble X de variables limitées par un ensemble de domaines discrets finis D et soumises à un ensemble de contraintes linéaires C. Dans un contexte d'optimisation, une fonction objectif f peut également être associée à la définition du problème, on parle dans ce cas de *constraint satisfaction optimisation problem* ou *CSOP*.

Dans la suite de ce chapitre, D_x représente le domaine d'une variable $x \in X$, une *instanciation élémentaire* (x, v) associe à une variable x une valeur v de son domaine D_x. Une instanciation partielle est un ensemble d'instanciations élémentaires. Une *instanciation partielle* I satisfait une contrainte $c \in C$ si les différentes valeurs prises par les variables de I respectent la contrainte c. Dans le cas contraire, on dit que l'instanciation partielle viole la contrainte c. On parle d'*instanciation complète* lorsque l'ensemble des variables de X sont instanciées. Une instanciation complète est *cohérente* si elle satisfait toutes les contraintes de l'ensemble C. Une *solution* est une instanciation complète et cohérente des variables de X. Une solution x^* est *optimale* si pour toute solution x, $f(x^*) \leq f(x)$ (pour un objectif de minimisation).
Résoudre un CSP (X, D, C) revient à déterminer une instanciation complète et cohérente (une solution). On peut également s'intéresser à la recherche de toutes les solutions. Résoudre un CSOP consiste à déterminer une solution optimale ou une solution de bonne qualité (ou toutes les solutions) par rapport à la fonction objectif considérée.
Pour cela, les méthodes étudiées dans ce travail parcourent l'espace de recherche (ou espace des solutions) du problème en développant une arborescence de recherche ou arbre de recherche par abus de langage. Le principe général de ces méthodes consiste à instancier progressivement les variables de X et à effectuer des retours-arrière lorsque survient une incohérence se traduisant par l'impossibilité d'instancier certaines variables car leur domaine s'est vidé au cours de la résolution.
Nous nous sommes plus particulièrement centrés sur deux grandes familles de méthodes que nous présentons plus en détail par la suite : les méthodes de type *backtrack chronologique* et les méthodes basées sur les *divergences*.

1.1.2 Heuristiques d'instanciation

Les méthodes de résolution (basées divergences ou de type backtrack chronologique), s'appuient sur une heuristique d'instanciation, aussi appelée stratégie de branchement, pour déterminer un ordre sur les variables à instancier et un ordre sur les valeurs que ces variables peuvent prendre. L'instanciation des variables s'opère en sélectionnant les variables l'une après l'autre en suivant l'ordre défini par l'heuristique d'instanciation. Pour chaque variable sélectionnée, l'ordre dans lequel vont être examinées les différentes valeurs est également déterminé par l'heuristique d'instanciation.
L'ordre sur les variables (ou sur les valeurs) peut être statique - dans ce cas l'ordre d'examen des variables (respectivement valeurs) est fixé avant la résolution du problème - ou il peut être dynamique - dans ce cas l'ordre d'examen des variables (valeurs) doit être établi au cours de la procédure de résolution.

L'heuristique d'instanciation vise donc à guider la méthode de recherche pour l'obtention rapide d'une solution (éventuellement de bonne qualité). Les heuristiques d'ordre sur les variables et d'ordre sur les valeurs s'appuient, en général, sur des principes opposés [143, 12]. L'heuristique d'ordre sur les variables exploite le principe dit *fail-first* dont le but est de minimiser la taille de l'arbre de recherche développé en faisant en sorte que les branches ne conduisant pas à une solution soit élaguées le plus rapidement possible [72]. L'heuristique d'ordre sur les valeurs utilise quant à elle le principe dit *succeed-first* afin de sélectionner une valeur ayant de bonnes chances d'appartenir à une solution et de limiter les retours-arrière dans l'arbre de recherche.

Parmi différentes heuristiques génériques basées sur ces principes, on peut citer :
- **l'heuristique de variables basée sur le domaine minimal** (dom) : elle sélectionne en premier la variable ayant le moins de valeurs possibles. Elle peut être définie de manière statique ou dynamique mais en pratique elle est utilisée de manière dynamique ;
- **l'heuristique de variables basée sur le degré** : elle sélectionne la variable impliquée dans le plus grand nombre de contraintes [45]. Elle peut être utilisée en statique (deg) ou en dynamique ($ddeg$) et dans ce dernier cas sélectionne en priorité la variable impliquée dans le plus grand nombre de contraintes par rapport à des variables non encore instanciées ;
- **l'heuristique de variables basée sur le degré pondéré** ($wdeg$) : cette heuristique mémorise des informations sur les échecs rencontrés au cours de la recherche de solution afin de les exploiter lors d'une prochaine étape de résolution par le biais de pondérations associées aux contraintes [24]. Ainsi, l'heuristique $wdeg$ donne la priorité aux variables impliquées dans les contraintes ayant le poids le plus important. Pour cela, initialement le poids de chaque contrainte est identique et il est incrémenté à chaque fois que celle-ci n'est pas satisfaite lors du développement des différentes branches de l'arbre de recherche. Le degré pondéré d'une variable x est alors défini comme la somme des poids des contraintes liant x à au moins une autre variable non encore instanciée.
- **l'heuristique de la valeur la moins contraignante** ($min - conflict$) : c'est une heuristique dynamique sélectionnant la valeur laissant le plus de valeurs possibles pour les autres variables non encore instanciées ;
- **les heuristiques de variables et de valeurs basées sur l'« impact »** ($impact$) : pour déterminer un ordre sur les variables, ce type d'heuristiques consiste à mesurer l'impact de l'instanciation d'une variable, c'est-à-dire la réduction de l'espace de recherche qu'elle engendre et à utiliser cette valeur de l'impact pour sélectionner la

variable ayant l'impact le plus élevé sur l'espace de recherche en se basant sur le principe *fail-first* [137]. Elle peut également être utilisée pour sélectionner la valeur ayant le plus faible impact sur l'espace de recherche en suivant dans ce cas le principe *succeed-first*.

Les méthodes de résolution qui utilisent des heuristiques d'ordre dynamique assurent, en général, une résolution plus rapide que celle obtenue par des méthodes basées sur des heuristiques d'ordre statique. En ce qui concerne la sélection de variables, les différentes heuristiques présentées précédemment peuvent également être combinées entre elles. On retrouve classiquement dans la littérature les heuristiques de variables suivantes :

- $dom \oplus ddeg$ également appelée *brelaz* (du nom de son auteur) : sélectionnant la variable ayant le plus petit domaine et en cas d'égalité la variable ayant le plus grand degré dynamique ;
- $dom/ddeg$ [21] : sélectionnant en priorité la variable ayant le plus petit rapport entre la taille du domaine et le degré dynamique ;
- $dom/wdeg$ [24] : donnant la priorité à la variable ayant le plus petit rapport entre la taille du domaine et le degré pondéré.

Les performances de l'heuristique $dom/wdeg$ ont été validées expérimentalement sur de nombreux problèmes (voir par exemple les travaux de [24, 85, 65]). Les raisons de ces bonnes performances sont expliquées par le fait que la pondération des contraintes permet de guider la recherche vers des parties difficiles du problème et de limiter les redondances lors de la recherche de solution (exploration répétée de mêmes sous-arbres de recherche aussi appelée *trashing*) ; de plus, cette heuristique s'avère être une alternative efficace vis-à-vis des méthodes exploitant des techniques de retours-arrière intelligents (voir ci-après).

En complément de ces différentes heuristiques, une méthode basée sur l'analyse des derniers conflits a été proposée dans [104, 105] afin d'améliorer l'efficacité du processus de résolution. Le principe du raisonnement basé sur les derniers conflits (LC) est de remonter dans l'arbre de recherche sur la variable ayant causé le dernier échec. Pour cela, cette variable est considérée comme devant être prioritairement instanciée quelle que soit la sélection effectuée par l'heuristique d'ordre sur les variables. Ce raisonnement a été généralisé à l'analyse des k derniers conflits (LC_k). Les expérimentations menées par les auteurs de cette méthode ont montré que combinée avec des heuristiques « classiques » telle que $dom \oplus ddeg$ ou $dom/ddeg$, l'utilisation de LC ou de LC_k améliorait les performances des méthodes. En revanche, ces améliorations ne sont pas toujours présentes lorsque le raisonnement basé sur les derniers conflits est couplé à l'heurisique $dom/wdeg$ car celle-ci permet déjà d'éviter de nombreuses redondances dans le parcours de l'arbre de recherche.

1.1.3 Principe général de la méthode avec retours-arrière chronologiques

La méthode avec retours-arrières chronologique ou « backtrack chronologique » (BT) (cf Algorithme 1) est une méthode de recherche arborescente. Un nœud de l'arborescence correspond à une variable x du problème et une branche issue de ce nœud représente l'instanciation de la variable x par une valeur de son domaine de telle sorte que cette l'instanciation soit cohérente avec les instanciations précédentes. La méthode BT parcourt l'ensemble des variables avec une stratégie *en profondeur d'abord* : elle instancie progressivement chaque variable du problème, sur la base d'une heuristique d'instanciation, jusqu'à obtention d'une solution ou détection d'une insatisfiabilité. Nous supposons également que la méthode BT inclut des mécanismes de propagation de contraintes : après chaque instanciation des propagations sont effectuées sur les variables non encore instanciées afin d'élaguer l'arbre de recherche et d'améliorer l'efficacité de la méthode de résolution.

Lorsque le domaine d'une variable devient vide, un échec apparaît et la méthode effectue un retour-arrière sur la dernière variable précédemment instanciée.

Algorithme 1 Backtrack Chronologique

Entrées: X, D, C {X : ensemble de variables, D : ensemble de domaines, C : ensemble de contraintes}
Sorties: Sol

if $X = \emptyset$ **then**
  Arrêt {Sol contient une solution}
else
  $x \leftarrow$Selectionner_Variable(X)
  for $v \in D_x$ **do**
    if Propagation_OK($X, D, C, Sol \bigcup \{(x, v)\}$) **then**
      $BT(X \backslash \{x\}, D, C, Sol \bigcup \{(x, v)\})$
      if Instanciation_Complete(Sol) **then**
        Arrêt
      end if
    end if
  end for
  $Sol \leftarrow \emptyset$
end if

La méthode BT stoppe dès qu'une solution est obtenue. Si le problème est insatisfiable elle développe la totalité de l'arbre de recherche.

1.1.4 Propagation de contraintes

La propagation de contraintes permet de vérifier des propriétés de *cohérence* (on parle aussi de *consistance*) des instanciations partielles courantes vis-à-vis des contraintes du problème. Pour cela, la propagation de contraintes se base sur des mécanismes de filtrage permettant de transformer le problème en un problème équivalent dans lequel des éléments (valeurs de certains domaines, simplification de contraintes, ...) ne pouvant figurer dans une solution ont été éliminés. La propriété de cohérence globale d'un problème étant dans le cas général difficile à établir, on se restreint en pratique à des niveaux de cohérence plus faible en considérant des sous-problèmes du problème initial (réduction des domaines, prise en compte de sous-ensembles de variables, ...). Ces différents niveaux de cohérence sont associés à des couts de calcul plus ou moins élevés [118, 112, 148, 44].

De manière générale, on dit qu'un problème est *k-cohérent* (ou *k-consistant*) si, pour toute instanciation I de $k-1$ variables $(x_1, ..., x_{k-1})$, il existe pour toute variable x_p ($p \in [k, n]$) une valeur $v_p \in D_p$ telle que l'instanciation $I \cup \{(x_p, v_p)\}$ vérifie les contraintes liant ces k variables.
Le niveau le plus élémentaire de cohérence est la *1-cohérence* ou *cohérence de domaines* qui consiste à vérifier l'ensemble des contraintes unaires pour toutes les valeurs des domaines des variables d'un problème : $\forall x_k \in X$ et $\forall v_k \in D_k$, l'instanciation (x_k, v_k) est cohérente. Par exemple, soit une variable x_k de domaine $D_k = \{1, 2, 3, 4, 5\}$ et soit la contrainte unaire $x_k < 4$, la vérification de la 1-cohérence permet par filtrage de réduire le domaine de cette variable à $D'_k = \{1, 2, 3\}$.

Un deuxième niveau de cohérence est celui de la *2-cohérence* vérifiant que $\forall x_i, x_j \in X$, $\forall v_i \in D_i$, il existe $v_j \in D_j$ telle que l'instanciation $\{(x_i, v_i), (x_j, v_j)\}$ vérifie les contraintes

binaires liant x_i et x_j.
Par exemple, soient deux variables x_i et x_j telles que $D_i = D_j = \{1, 2, 3, 4\}$ et soit la contrainte binaire $x_i > x_j + 1$, la vérification de la 2-cohérence permet de réduire le domaine de ces deux variables par filtrage à $D'_i = \{3, 4\}$ et $D'_j = \{1, 2\}$.
On parle également de k-cohérence forte lorsque la p-cohérence est vérifiée $\forall p = 1, ..., k$. Ainsi un problème vérifie la 2-cohérence forte s'il vérifie à la fois la 1-cohérence et la 2-cohérence.

Dans le cas de problèmes à contraintes binaires, on s'intéresse plus particulièrement aux propriétés d'*arc-cohérence* ou de *chemin-cohérence*. Un problème est dit arc-cohérent si et seulement toutes ses contraintes vérifient la 2-cohérence (et que les domaines des variables ne sont pas vides). La chemin-cohérence consiste à vérifier la propriété de cohérence d'une succession de contraintes binaires reliant des variables sur un chemin donné. Pour les problèmes à contraintes binaires, l'arc-cohérence et la 2-cohérence sont équivalentes ainsi que la chemin-cohérence et la 3-cohérence.

Intégrée dans une méthode de résolution, la propagation de contraintes permet donc de simplifier a priori la résolution d'un problème en réduisant l'espace de recherche à considérer, ces techniques sont qualifiées de *prospectives* ou *look-ahead*). Avant toute résolution la propagation de contraintes peut également dans certains cas apporter la preuve de l'incohérence globale du problème.
Dans la suite de ce chapitre, lorsqu'un problème n'admet pas de solution, on le qualifie de problème insatisfiable. Dans le cas contraire, c'est-à-dire lorsqu'une solution existe, le problème est dit satisfiable.

Les mécanismes de propagation classiquement utilisés au sein d'un algorithme de retours-arrière chronologique sont l'arc-cohérence [112, 111]. Celle-ci peut être réalisée sur l'ensemble des contraintes (méthode *MAC* pour *Maintening Arc-Consistency*) ou réalisée de manière plus locale en ne considérant que les contraintes liées à la dernière variable instanciée (méthode *FC* pour *Forward-Checking*).
Afin de résoudre des problèmes d'optimisation, la méthode de retours-arrière chronologique peut être intégrée au sein d'une recherche dichotomique faisant varier les bornes inférieure et supérieure de la fonction objectif à chaque itération.

1.1.5 Retours-arrière intelligent et apprentissage

Afin d'améliorer l'efficacité des méthodes de recherche, et en complément des techniques de propagation présentées précédemment, des techniques permettant d'améliorer la stratégie de retour-arrière qualifiées de techniques de *rétrospectives* ou *look-back* ont été proposées [42, 44].

Lorsqu'un échec survient lors de la recherche d'une solution, les algorithmes de retour-arrière chronologique remettent en cause la dernière variable instanciée pour lui affecter une autre valeur. Les algorithmes utilisant des retours-arrières intelligents permettent quant à eux de revenir plus haut dans l'arborescence de recherche sur une variable à l'origine de l'échec rencontré. Les méthodes de *backjumping* [42, 135] effectuent un retour-arrière sur une variable à l'origine de l'échec rencontré, les déductions effectuées entre la variable sur laquelle le retour-arrière est effectué et la variable ayant rencontré un échec sont perdues ce qui peut conduire à des explorations inutiles de l'espace de recherche par la suite. La méthode *dynamic backtracking* [61], quant à elle, ne remet en cause l'affectation que de la dernière variable responsable de l'échec sans remettre en cause les autres instanciations

(dans la mesure où elles restent valident).
Ces techniques de retours-arrière intelligents ont été combinées avec des techniques prospectives telles que MAC mais l'efficacité de la combinaison n'est pas probante et dépend des instances considérées [89].

Un autre axe pour améliorer les performances des techniques de retours-arrière intelligents est de déterminer et de mémoriser les raisons de chaque échec. On parle alors de *nogood-recording* ou de méthodes avec apprentissage. Un nogood représente un ensemble d'affectations source de l'échec rencontré et la mémorisation des nogood a pour but d'éviter de reproduire les mêmes échecs lors d'étapes de recherche ultérieures. Ainsi, une méthode de résolution basée sur les nogood utilise les informations mémorisées en plus des contraintes du problème pour déterminer une solution. Afin de limiter la complexité des algorithmes, différentes techniques permettant de limiter la taille des nogood ont été développées (en termes de nombre de variables, de pertinence par rapport à l'arbre de recherche en cours de développement).

1.1.6 Redémarrage

Une autre des techniques utilisées pour limiter l'exploration répétitive de mêmes parties de l'arbre de recherche consiste à utiliser des redémarrages [110, 99]. Les techniques de redémarrage (ou *restart*) permettent d'améliorer l'efficacité d'une méthode de recherche en y introduisant de l'aléatoire. Pour cela, si la méthode n'aboutit pas à une solution lorsqu'une condition appelée *condition de coupure* est atteinte (nombre de retours-arrière, nombre d'itérations etc.), elle est interrompue et redémarrée avec de nouveaux paramètres pour avoir un comportement différent. Les variations introduites concernent généralement les heuristiques d'instanciation. La condition de coupure est mise à jour à chaque redémarrage pour garantir la complétude de la méthode. L'introduction de redémarrages dans une méthode de résolution nécessite de déterminer leurs fréquences ; pour cela il faut établir une condition initiale de coupure ainsi qu'un taux d'accroissement de l'exploration. Ce taux d'accroissement correspond, par exemple, à une suite géométrique comme proposé dans [152].

1.1.7 Principe général des méthodes à divergences

Notion de divergence

Les méthodes de résolution basées sur les divergences se basent, comme la méthode BT, sur une instanciation progressive des variables d'un problème sur la base d'une heuristique d'instanciation. Cependant, elles offrent un parcours alternatif de l'arbre de recherche par rapport aux retours-arrière chronologiques.

La notion de divergence se définit par rapport à l'heuritique de choix de valeurs de référence et représente la contradiction des choix de cette heuristique. Ainsi, une divergence consiste en l'instanciation d'une variable avec une valeur différente de celle préconisée par l'heuristique de choix de valeur.
Par exemple, soit un problème à 4 variables binaires $X = \{x_1, x_2, x_3, x_4\}$ de domaine $D_i = \{a, b\}$ pour toute variable $x_i \in X$ et considérons une heuristique d'instanciation fournissant un ordre statique sur les variables, par exemple l'ordre lexicographique et un ordre statique sur les valeurs, par exemple « a avant b ». La figure 1.1 donne le nombre de divergences associées à chaque feuille de l'arbre de recherche de ce problème. La feuille à 0 divergence est celle pour laquelle toutes les variables ont été instanciées à la valeur a, les feuilles à 1 divergence sont celles pour lesquelles une seule variable a été instanciée par la

valeur b et ainsi de suite.
Par convention, sur cette figure (et sur les suivantes), la valeur privilégiée par l'heuristique correspond au fils gauche d'un nœud.

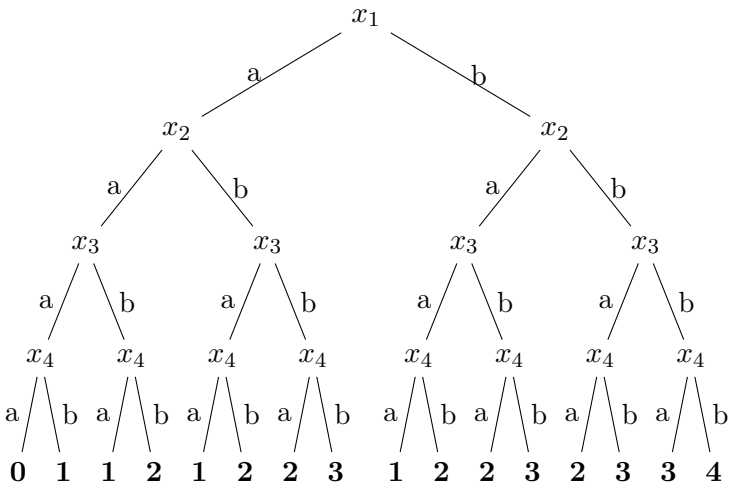
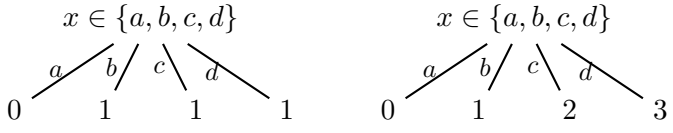


FIGURE 1.1 – Divergences sur un arbre à quatre variables binaires

Les méthodes à divergences ont été conçues initialement pour résoudre des problèmes à variables binaires et ont été adaptées par la suite pour des problèmes à variables non binaires [58]. Dans le cas de problèmes à variables n-aire, par convention, le fils le plus à gauche correspond au meilleur choix de l'heuristique de valeurs et tous les autres fils correspondent à des divergences. Deux modes de comptage des divergences sont ainsi classiquement envisagés :

- soit le premier fils (à gauche) correspond à 0 divergence et tous les autres fils à 1 divergence. On se ramène ainsi à un comptage binaire des divergences ;
- soit le premier fils (à gauche) correspond à 0 divergence et les autres fils sont associés à un nombre de divergences croissant au fur et à mesure qu'on se déplace vers la droite dans l'arbre. On procède alors à un comptage non binaire des divergences pour chaque nœud.

Les schémas de la figure 1.2 donne le nombre de divergences obtenus suite à l'instanciation d'une variable x par l'une des valeurs de son domaine en supposant qu'elles sont ordonnées de manière lexicographique. Sur le premier schéma, toute branche autre que celle correspondant à $x = a$ correspond à une divergence, sur le deuxième schéma, le nombre de divergences obtenu dépend de l'ordre des valeurs.



(a) Comptage binaire (b) Comptage non-binaire

FIGURE 1.2 – Modes de comptage binaire et non-binaire

L'intégration de techniques de propagation au sein de méthodes à divergences n'est pas détaillée dans la littérature. Pour cela, les deux approches présentées ci-dessous sont

envisageables.
Suite à des propagations, des valeurs éventuellement privilégiées par l'heuristique d'instanciation peuvent avoir été supprimées du domaine des variables. Si le comptage des divergences est effectué de manière dynamique, la valeur effectivement choisie par l'heuristique d'instanciation sera la première valeur non éliminée par propagation.
En revanche, si le comptage des divergences est effectué de manière statique, la suppression de valeurs dans le domaine de certaines variables peut rendre impossible le développement de certaines branches lors de la recherche de solutions pour un nombre de divergences donné.
A notre connaissance, il n'y a pas d'étude expérimentale comparant ces deux approches. Dans la suite de ce chapitre, lorsque nous considérons une méthode à divergences couplée avec des techniques de propagation, nous supposons que le comptage des divergences est effectué de manière dynamique.

Méthodes LDS, ILDS et DDS

Depuis une quinzaine d'années apparaissent de nouvelles stratégies de recherche basées sur le concept de divergence. L'idée générale est d'autoriser la recherche à s'écarter progressivement (en cas d'échec) d'une instanciation obtenue par une heuristique.
La première de ces méthodes est la *recherche à divergences limitées* (*Limited Discrepency Search* ou LDS), proposée en 1995 par W.D. Harvey et M.L. Ginsberg [73].

Parmi les nombreuses variantes de cette méthode, nous présenterons la méthode de *recherche à divergences limitées améliorée* (*Improved Limited Discrepancy Search* ou ILDS), permettant de limiter la redondance dans l'exploration des nœuds de l'arbre des solutions, proposée par R.E. Korf en 1996 [102], et plusieurs variantes et la méthode de *recherche à divergences limitées en profondeur* (*Depth-bounded Discrepancy Search* ou DDS) proposée en 1997 par T. Walsh [151] permettant de considérer prioritairement les divergences en priorité à un niveau élevé dans l'arbre de recherche.

La méthode originale et ses principales variantes sont présentées plus en détail par la suite mais toutes ces méthodes suivent un même principe général résumé dans la figure (1.3). Des variantes sur le test d'application des divergences et sur l'ordre d'exploration des nœuds fils conduisent à différentes méthodes à divergences.

L'idée de la méthode LDS est de construire, en premier, le chemin de l'arborescence recommandé par l'heuristique, ce qui aboutit à une feuille à 0 divergence, suivi (si la feuille à 0 divergence n'est pas une solution) de tous les chemins qui divergent de l'heuristique pour une seule variable, ce qui constitue les feuilles à une seule divergence, suivis de tous les chemins qui ont 2 divergences (si les feuilles à 1 divergence ne contiennent pas de solution) et ainsi de suite. Le nombre maximum de divergences possibles pour un problème donné est noté k_{max} et correspond dans le cas d'un problème à variables binaires à la profondeur maximale de l'arbre. Les algorithmes 2 et 3 donnent le principe de cette méthode tel que exposé par leurs auteurs.

Une première hypothèse faite par les auteurs de la méthode LDS est que l'heuristique d'instanciation est suffisamment performante pour guider la recherche de solution dans le parcours de l'arbre en suggérant de bonnes valeurs à sélectionner en premier. S'il y a des erreurs commises par cette heuristique, elles sont en petit nombre et effectuer progressivement des divergences par rapport à cette heuristique va permettre d'aboutir rapidement à une solution. Une seconde hypothèse faite pour la méthode LDS est que les erreurs effectuées par l'heuristique dans le haut de l'arbre de recherche sont plus importantes que les

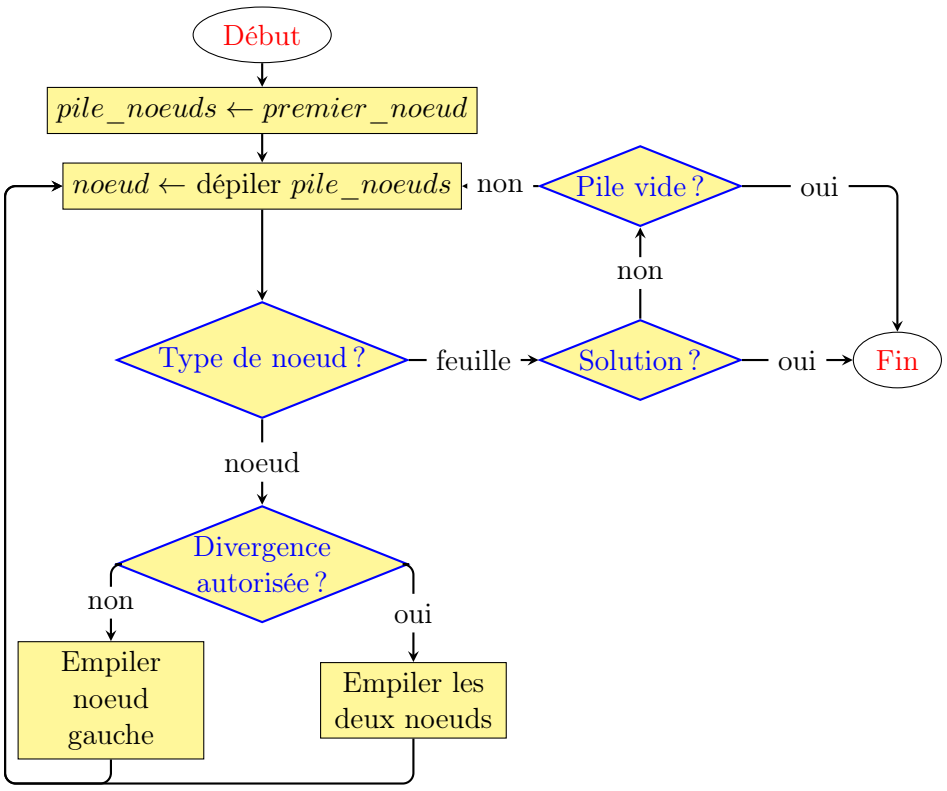


FIGURE 1.3 – Principe général des méthodes de résolution à divergences

Algorithme 2 LDS

ENTRÉES: X, D, C {X : ensemble de variables, D : ensemble de domaines, C : ensemble de contraintes}

1: $k_{max} \leftarrow$ profondeur maximale, $node \leftarrow$ Racine(X, D, C)
2: $k \leftarrow 0$, $Res \leftarrow NIL$
3: **while** $k \leq k_{max}$ and not Est_Solution(Res) **do**
4: $\quad Res \leftarrow$ LDS_Iteration$(node, k)$
5: **end while**
6: **return** Res

erreurs effectuées dans le bas de l'arbre car, d'une part, l'heuristique dispose de moins d'information pour effectuer des choix de manière pertinente. Dans ce cas, il est plus intéressant de commencer à appliquer les divergences en partant du haut de l'arbre de recherche.

La figure 1.4 illustre le parcours effectué par la méthode LDS pour l'exploration de l'ensemble des solutions d'un problème composé de 3 variables binaires. Les numéros placés au niveau des feuilles donnent l'ordre d'apparition de ces feuilles lors de la résolution. A l'itération $k = 0$, la feuille à 0 divergence est obtenue, lors de l'itération suivante ($k = 1$), les feuilles à 1 divergence ainsi que la feuille à 0 divergence sont atteintes, à l'itération $k = 2$, les feuilles à 2, 1 et 0 divergences sont explorées puis à l'itération $k = 3$, les feuilles à 3, 2, 1 et 0 divergences sont atteintes.

Comme on peut le noter sur cette figure, le parcours de l'arborescence effectué par la méthode LDS comporte de nombreuses redondances. En effet, à chaque itération, la méthode LDS visite non seulement les branches conduisant à des feuilles ayant exactement un nombre de divergences fixé mais également toutes les branches qui conduisent à des feuilles ayant un nombre de divergences inférieur à celui fixé.

Algorithme 3 LDS_Iteration($node, k$)

```
1: if Est_Solution(node) then
2:    return node
3: end if
4: if Sans_Fils(node) then
5:    return NIL
6: end if
7: if k = 0 then
8:    return LDS_Iteration(Fils_Gauche(node), 0)
9: else
10:   Res ← LDS_Iteration(Fils_Droit(node), k − 1)
11:   if Res ≠ NIL then
12:      return Res
13:   end if
14:   return LDS_Iteration(Fils_Gauche(node), k)
15: end if
```

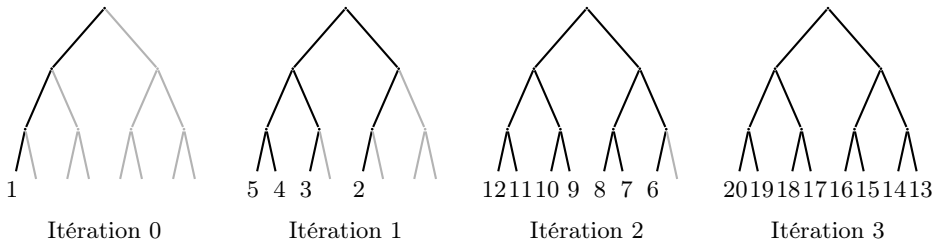


FIGURE 1.4 – Ordre de visite des feuilles d'un arbre à trois variables binaires avec LDS

Pour remédier aux redondances inhérentes à la méthode LDS, la méthode ILDS, proposée par R.E. Korf [102] en 1996, vise à limiter les feuilles explorées à chaque itération en considérant uniquement celles les feuilles ayant exactement le nombre de divergences fixé. Pour cela, la méthode ILDS effectue un test sur la profondeur du nœud avant toute itération supplémentaire. Le principe de l'algorithme ILDS est le même que celui de LDS (cf. algorithme 2) sauf qu'à chaque itération l'appel à la procédure ILDS_Iteration nécessite un paramètre supplémentaire correspondant à la profondeur maximale de l'arbre (k_{max}). L'algorithme 4 présente le principe de cette procédure ILDS_Iteration.

La figure 1.5 illustre le parcours effectué par la méthode ILDS pour l'exploration de l'ensemble des solutions d'un problème composé de 3 variables binaires. Comme précédemment, les numéros placés au niveau des feuilles donnent l'ordre d'apparition de ces feuilles lors de la résolution. Avec ILDS, seules les feuilles ayant exactement p divergences sont obtenues lors de l'itération $k = p$, ainsi chaque feuille n'est explorée qu'une seule fois. Mais il n'en est malheureusement pas de même pour les nœuds internes qui comme le montre la figure sont explorés plusieurs fois. Cette variante des méthodes à divergences présente donc toujours une certaine redondance que l'on ne peut éliminer car inhérente au principe même de la recherche à divergences.

Contrairement à LDS, la méthode ILDS applique en premier les divergences en bas de l'arborescence ce qui semble en contradiction avec la seconde hypothèse faite pour le développement de la méthode LDS. Dans [136], P. Prosser et C. Unsworth ont montré expérimentalement (sur des problèmes de car-sequencing et de circuits hamiltoniens) que

Algorithme 4 ILDS_Iteration($node, k, profondeur$)

```
1: if Est_Solution(node) then
2:    return node
3: end if
4: if Sans_Fils(node) then
5:    return NIL
6: end if
7: if profondeur > k then
8:    ILDS_Iteration(Fils_Gauche(node), k, profondeur − 1)
9:    if Res ≠ NIL then
10:      return Res
11:   end if
12: end if
13: if k > 0 then
14:   return LDS_Iteration(Fils_Droit(node), k − 1, profondeur − 1)
15: end if
```

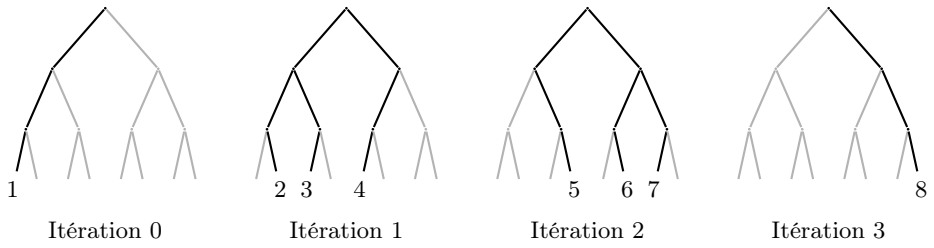


FIGURE 1.5 – Ordre de visite des feuilles d'un arbre à trois variables binaires avec ILDS

l'application des divergences à partir du haut de l'arbre de recherche ou à partir du bas avait finalement peu de conséquence sur les performances obtenues.
Une variante de la méthode ILDS, appelée RLDS (pour *Reverse LDS*) a été proposée par N. Prcovic [134], elle correspond à la méthode ILDS dans laquelle les divergences sont appliquées en haut de l'arborescence en priorité.
La méthode ILDS, proposée initialement pour des problèmes à variables binaires a été étendue dans [136] au cas de variables non binaires en considérant un comptage non binaire des divergences à chaque variable.

Dans la méthode DDS, proposée en 1997 par T. Walsh [151], l'idée de l'auteur est de considérer en priorité les divergences à un niveau élevé dans l'arbre de recherche plutôt qu'à un niveau bas afin de corriger les premières erreurs dans les instanciations commises par l'heuristique. Cette méthode s'appuie sur l'hypothèse, faite par les auteurs de LDS, que, dans la partie basse de l'arbre, l'heuristique d'instanciation fait généralement un choix qui n'influence pas trop la solution du problème. Ainsi, la méthode DDS augmente progressivement le nombre de divergences autorisé en restreignant leurs emplacements dans une partie supérieure de l'arbre paramétrée par une limite sur la profondeur autorisée pour les divergences. Cette limite sur la profondeur est relâchée au fur et à mesure que la recherche avance et que le nombre de divergences autorisées augmente. De plus, comme la méthode ILDS, la méthode DDS génère à chaque itération, les feuilles ayant exactement le nombre de divergences fixé.
Le principe général de la méthode DDS est identique à celui de la méthode LDS (cf. algorithme 2) sauf qu'à chaque itération DDS appelle la procédure $DDS\_Iteration$ dont le principe est donné dans l'algorithme 5.

Algorithme 5 DDS_Iteration($node, k$)

```
1: if Est_Solution(node) then
2:    return node
3: end if
4: if k = 0 then
5:    return DDS_Iteration(Fils_Gauche(node), 0)
6: end if
7: if k = 1 then
8:    return DDS_Iteration(Fils_Droit(node), 0)
9: end if
10: if k > 1 then
11:    Res ← DDS_Iteration(Fils_Gauche(node), k − 1)
12:    if Res ≠ NIL then
13:       return Res
14:    end if
15:    return DDS_Iteration(Fils_Droit(node), k − 1)
16: end if
```

Il existe de nombreuses autres variantes des méthodes à divergences que nous ne présenterons pas ici. Nous présentons une synthèse plus détaillée dans les travaux de thèse de A. Ben Hmida et W. Karoui [13, 91]. Notons que les méthodes à divergences ont initialement été proposées dans un contexte de satisfaction de contraintes. Dans la partie suivante de ce chapitre, nous présentons des adaptations des méthodes à divergences pour la résolution de problème d'optimisation.

Méthodes à divergences pour l'optimisation

La méthode LDS peut être adaptée pour la résolution de problèmes d'optimisation. Dans un contexte d'optimisation, la solution à 0 divergence fournie par l'heuristique d'instanciation a un certain cout vis-à-vis d'une fonction objectif. La méthode LDS peut alors être employée pour rechercher de meilleures solutions que la solution initiale en utilisant la notion de divergences mais, pour garantir l'optimalité de la solution trouvée il est nécessaire de développer l'arbre de recherche dans sa totalité (c'est-à-dire d'attendre le nombre maximum de divergence possible k_{max}). En pratique, la méthode LDS sera d'un très grand intérêt si elle réussit à trouver une solution de bonne qualité avec un nombre limité de divergences. Pour restreindre la recherche de solutions de meilleure qualité autour de la solution courante, il est possible de fixer un nombre de divergences limite inférieur à k_{max}. Cette première adaptation de la méthode LDS à l'optimisation revient en fait à l'exploration d'un voisinage d'une solution courante et à la sélection de la meilleure solution de ce voisinage ce qui en fait une méthode peu performante pour l'obtention de solutions de bonne qualité.

La méthode appelée « Climbing Discrepancy Search (CDS) » ou *recherche par progression de divergences*, proposée par [116] permet une exploration plus importante de l'espace de recherche des solutions. La méthode CDS débute l'exploration à partir d'une solution initiale obtenue à l'aide d'une heuristique d'instanciation. Cette solution est évaluée à l'aide de la fonction objectif considérée. La méthode CDS explore ensuite les feuilles ayant 1 divergence par rapport à cette solution initiale et les évalue à l'aide de la fonction objectif, puis, si aucune de ces feuilles n'améliore la solution courante, les feuilles à 2 divergences sont générées et évaluées. Le processus de recherche de solutions de meilleure qualité se

poursuit ainsi en augmentant progressivement le nombre de divergences autorisées. Quand une solution améliorant le cout de la solution courante est obtenue, celle-ci est mise à jour et le nombre de divergences autorisées est re-initialisé. Le processus d'exploration autour de cette nouvelle solution courante se déclenche de nouveau. La méthode CDS se termine quand un critère d'arrêt est atteint. L'algorithme 6 illustre cette méthode.

Algorithme 6 CDS

Entrées: X, D, C, f {X : ensemble de variables, D : ensemble de domaines, C : ensemble de contraintes, f : fonction objectif}
Sorties: Sol

```
1: k ← 0 {k est le nombre de divergences}
2: while not (Condition d'arret) do
3:    k ← k + 1
4:    S ← Generer_Feuilles(Sol, k) {Déterminer les solutions à k divergences}
5:    Sol' ← Meilleure_Feuille(S, f) {Sélectionner dans S la solution de meilleur cout par
      rapport à f}
6:    if Est_Meilleur(f(Sol'), f(Sol)) then
7:       Sol ← Sol'
8:       k ← 0
9:    end if
10: end while
```

La méthode CDS peut être vue comme une méthode de recherche à voisinage variable (*Variable Neighbourhood Search* ou VNS ou plus présisément *Variable Neighbourhood Descent*) [69]. En effet, les deux méthodes explorent un voisinage de plus en plus grand autour d'une solution tant que celle-ci n'est pas améliorée. L'intérêt de CDS réside dans la simplicité de la définition du voisinage d'une solution qui repose sur le principe de divergence et consiste à explorer un arbre de plus en plus grand autour d'une solution de référence.

1.2 Contributions apportées

Cette partie présente une synthèse des contributions apportées sur les méthodes à divergences et sur de nouvelles heuristiques génériques d'instanciation. Les applications de ces différentes contributions dans un contexte optimisation ou satisfaction sont détaillées dans les parties suivantes.

1.2.1 Contributions pour les méthodes à divergences

Synthèse des méthodes à divergences

Nous proposons de classer les méthodes de la littérature en fonction de la redondance qu'elles entrainent lors du parcours de l'arbre de recherche et en fonction de la stratégie d'application des divergences (en haut de l'arbre en priorité ou en bas de l'arbre en priorité) [133]. Les méthodes LDS, ILDS et RLDS ont été présentées dans la partie 1.1 de ce chapitre. Sur cette base, nous avons proposé une méthode à divergences supplémentaire appelée par la suite PRLDS (pour pseudo RLDS) qui correspond à une variante de LDS avec divergence en priorité en bas (voir tableau 1.1).

Comme RLDS, la méthode DDS présentée dans la partie 1.1 est une méthode non redondante dans laquelle les divergences sont appliquées en premier en haut de l'arbre de recherche. La différence entre DDS et RLDS réside dans la manière dont sont appliquées les divergences dans DDS en tenant compte de la profondeur de l'arbre. Nous avons également

Parcours \ Divergences	en priorité en haut	en priorité en bas
avec redondance	LDS, **red-DDS**	**PRLDS**
sans redondance	RLDS, DDS	ILDS

TABLE 1.1 – Classification des méthodes à divergences

proposé une variante redondante de DDS (avec divergence en haut) appelée red-DDS (voir tableau 1.1).

Méthodes à divergences et heuristiques d'instanciation

Dans un contexte de satisfaction de contraintes, les méthodes à divergences sans redondance supposent l'utilisation d'heuristiques d'instanciation statiques afin de garantir la complétude du processus de résolution. En effet, lors du passage d'une itération à k divergences à une itération à $k+1$ divergences, en l'absence de redondance de l'exploration, il n'y a pas de garantie que les méthodes à divergences explorent de nouvelles parties de l'espace de recherche.
Cependant, dans la littérature, les heuristiques d'instanciation efficaces pour la résolution de problèmes combinatoires sont en général des heuristiques dynamiques.
Ainsi, les méthodes à divergences avec redondance conservent tout leur intérêt car elles peuvent être utilisées avec des heuristiques de variables dynamiques sans perdre au niveau de la complétude des résultats obtenus : en cas d'insatisfiabilité d'un problème toutes les branches de l'arborescence seront bien générées.
Toutefois, afin de limiter en partie la redondance des méthodes à divergences, il est possible de jouer sur la valeur de l'incrémentation du nombre de divergences lors d'une nouvelle itération. Dans les méthodes d'origine, le nombre de divergences est incrémenté de 1 à chaque itération, pour avoir des méthodes à divergences présentant moins de redondances dans l'exploration des nœuds cette valeur d'incrément peut être plus importante sans que cela nuise à la complétude de la méthode.
Lorsque cet incrément a une valeur importante, la méthode développe moins d'itérations mais chacune d'entre elle est de taille importante et s'il y a une solution pour un faible nombre de divergences, des nœuds auront été parcourus inutilement. En revanche si la solution est obtenue pour un nombre assez important de divergences, elle sera obtenue avec moins de redondances sur les différents nœuds car il y aura moins d'itérations.

Reduction des redondances dans les méthodes à divergences pour les problèmes incohérents

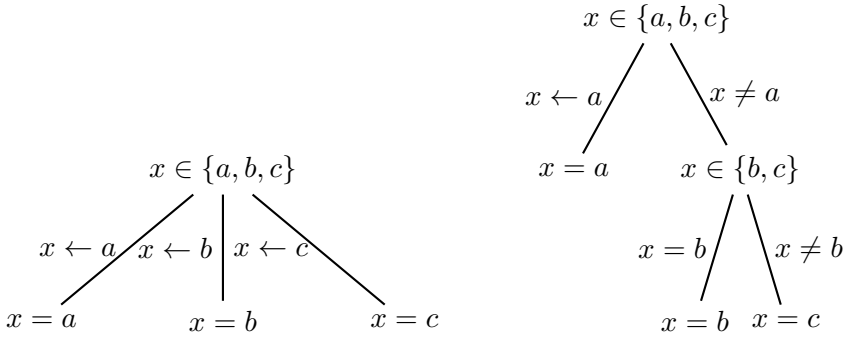
Nous avons proposé un mécanisme permettant de limiter les itérations des méthodes à divergences dans le cas de problèmes insatisfiables [98, 95]. Dans une méthode basée divergences, le nombre de divergences est incrémenté jusqu'à ce que le maximum de divergences autorisées par la recherche soit atteint. Pour les problèmes insatisfiables, dès lors que dans une itération de la méthode, un échec est détecté sans que le nombre maximum de divergences autorisées soit atteint il est inutile de poursuivre la recherche. En effet, un nombre plus important de divergences conduira à la même arborescence de recherche.
Ce mécanisme de réduction des redondances dans les méthodes à divergences s'avère très efficace [98, 95] par rapport aux méthodes à divergences classiques.

Mode de comptage des divergences

Dans le cas de problèmes à variables non binaires, les divergences peuvent être comptabilisées sur les variables soit de manière binaire : tout choix différent de celui déterminé par l'heuristique est considéré comme une divergence ; soit de manière non binaire : les différentes valeurs des variables sont ordonnées par l'heuristique, la première valeur correspond à 0 divergence, la deuxième valeur à 1 divergence, la troisième valeur à 2 divergences et ainsi de suite.

Comme dans [58], nous avons combiné le comptage binaire et le comptage non-binaire pour adopter un comptage mixte [91] dans lequel le comptage utilisé pour les divergences sur une variable dépend de la profondeur où elle se situe.

Par ailleurs, lorsque nous nous sommes intéressés à l'implémentation de méthodes à divergences au sein du solveur de contraintes MISTRAL (développé par Emmanuel Hébrard [74]), nous avons pu mettre en évidence un nouveau mode de comptage des divergences [133]. En effet, pour des raisons de performances [87], dans MISTRAL, comme dans tout solveur de contraintes récent, l'espace de recherche est représenté par un arbre binaire même si les variables sont n-aires. A chaque nœud de cet arbre, une des branches consiste à suivre le choix de l'heuristique c'est-à-dire à poser une instanciation alors que la deuxième branche consiste à poser la réfutation de cette instanciation.
La figure 1.6 compare le principe de branchement théorique et celui utilisé dans un solveur de contraintes sur une variable x n-aire pouvant prendre 3 valeurs ($D_x = \{a, b, c\}$). Dans la partie gauche de cette figure (ie. branchement théorique) une branche est générée pour chaque valeur possible de la variable x, l'arbre de recherche développé est alors n-aire. En revanche, dans la partie droite de la figure (ie. branchement du solveur), à un nœud donné une branche consiste à poser l'affectation d'une valeur à la variable (par exemple $x = a$) et l'autre branche consiste à poser sa réfutation (par exemple $x \neq a$).



(a) Branchement théorique

(b) Branchement binaire dans un solveur

FIGURE 1.6 – Différence entre explorations n-aire et binaire

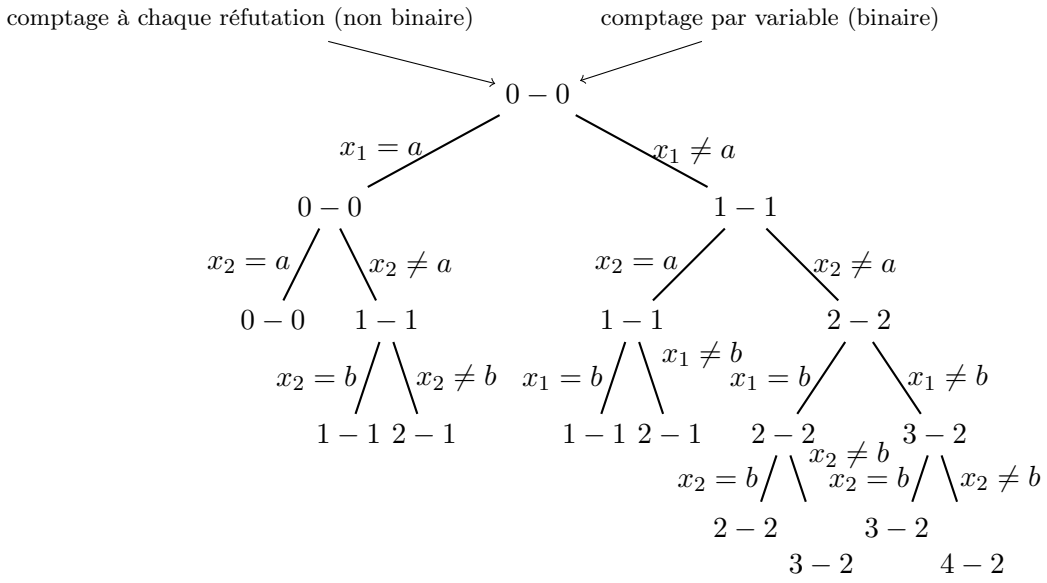
En raison de cette exploration d'un arbre binaire au sein de MISTRAL différentes implémentations des méthodes à divergences ont pu être proposées. Ces implémentations correspondent à différentes manières de comptabiliser les divergences sur les variables :

- *comptage à chaque réfutation* : le nombre de divergences est incrémenté à chaque réfutation. Cela correspond à un **comptage non binaire** des divergences en cas de variables n-aires ;
- *comptage par variable* : une réfutation sur une variable donnée ne sera comptée comme une divergence que si cette cette variable n'a pas encore eu de divergence. Cela cor-

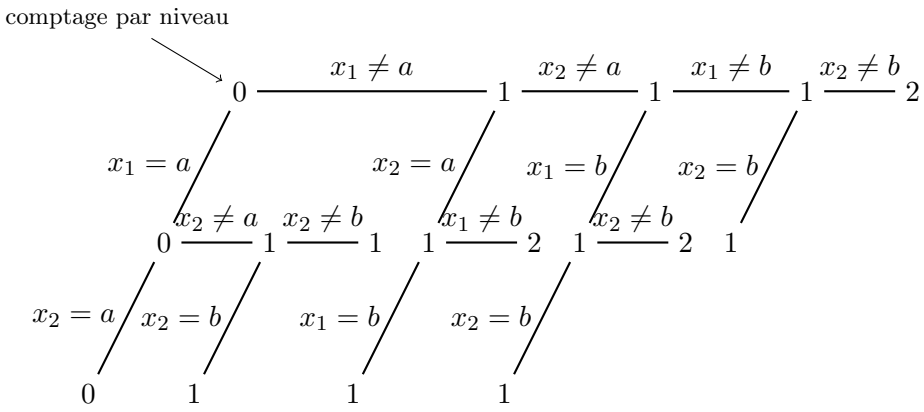
respond à un **comptage binaire** des divergences. Ce comptage et le précédent sont équivalents lorsque le problème ne comporte que des variables binaires ;

· *comptage par niveau dans l'arbre du solveur.* Dans l'arbre du solveur on ne change de niveau que lorsqu'une décision d'instanciation est prise. Avec ce mode de comptage, appelé **comptage par niveau**, plusieurs réfutations successives ayant donc lieu à un même niveau dans l'arbre du solveur ne correspondent qu'à une seule divergence, même si elles portent sur des variables différentes.

Ces différentes modes de comptage sont illustrés par la figure 1.7 représentant un arbre binaire développé par un solveur de contraintes. La première partie de cette figure (ie. Modes de comptage dans l'arbre binaire) fournit à chaque nœud de l'arbre de recherche le nombre de divergences trouvées avec le mode de comptage par réfutation (partie gauche du nœud) et avec le mode de comptage par variable (partie droite du nœud). Dans le solveur MISTRAL, une réfutation ne provoque pas de changement de niveau dans l'arbre binaire développé. La deuxième partie de cette figure (ie. Mode de comptage dans l'arbre du solveur) représente le même arbre binaire que précédemment mais organisé en fonction des niveaux. Sur cet arbre, on donne le nombre de divergences obtenues à chaque nœud avec le mode de comptage par niveau.



(a) Modes de comptage dans l'arbre binaire



(b) Mode de comptage dans l'arbre du solveur

FIGURE 1.7 – Différentes implémentations de comptabilisation des divergences

1.2.2 Heuristiques basées sur la pondération de variables ou de valeurs

Afin d'améliorer l'efficacité des heuristiques d'instanciation, nous avons proposé [97, 96] de mémoriser les échecs rencontrés lors de la résolution via des pondérations associées aux variables ou aux valeurs du problème.

Concernant la pondération des variables, l'heuristique proposée, appelée *wvar*, vise, lors d'une itération donnée, à considérer de manière prioritaire les variables sources d'échecs lors de précédentes itérations de la méthode de résolution. Pour cela, tous les poids des variables sont initialement fixés à une même valeur puis lors de la résolution, le poids d'une variable est incrémenté à chaque fois que cette variable entraine un échec de la méthode de résolution (ie. apparition d'un domaine vide).
Ce principe général de pondération des variables peut être combiné avec n'importe quelle autre heuristique dynamique d'ordre sur les variable. Par exemple, nous pouvons définir l'heuristique dynamique `dom⊕wvar` consistant à selectionner en priorité la variable ayant le plus petit domaine et en cas d'égalité la variable ayant rencontré le plus d'échecs. Nous pouvons également définir l'heuristique `dom/wvar` permettant de sélectionner en priorité la variable ayant à la fois le plus petit domaine et la plus grande pondération.

Sur le même principe, nous avons défini un mécanisme de pondération des instanciations (x, v) afin de prendre en compte les échecs survenant suite à des instanciations. Différentes incrémentations de ces pondérations peuvent être envisagées :

- lors d'un échec sur une variable x, incrémentation de la dernière instanciation effectuée (y, u) ayant entrainé un échec lors de la résolution ;
- lors d'un échec sur une variable x, incrémentation des instanciations supprimées (x, v) par la dernière instanciation effectuée (y, u) ;
- lors d'un échec sur une variable x, incrémentation des instanciations supprimées (x, v) par des instanciations autres que la dernière effectuée.

Ces différentes heuristiques à pondération sont envisagées de manière dynamique. Pour la résolution de problèmes de satisfaction de contraintes, elles sont employées au sein de méthodes à divergences redondantes ou au sein d'une méthode en profondeur d'abord avec retour-arrière chronologique.

1.2.3 Méthodes à divergences pour l'optimisation

Variantes de la méthode CDS

La méthode CDS a été proposée pour résoudre des problèmes d'optimisation en utilisant les méthodes à divergences. Elle peut ainsi être combinée avec différentes méthodes à divergences : LDS, ILDS, RLDS, PRLDS, DDS ou red-DDS. L'objectif de la méthode CDS est de fournir une solution approchée de bonne qualité pour un problème d'optimisation. Dans ce contexte-là, l'emploi de méthodes à divergences non redondantes semble plus pertinent car la complétude de la résolution n'est pas une nécessité.
Des heuristiques à pondération peuvent également être couplées à une méthode CDS, elles permettent alors lors de l'exploration de voisinages de sélectionner en priorité certaines valeurs mais aussi de sélectionner certaines variables sur lesquelles appliquer des divergences.

CDDS : une nouvelle méthode d'optimisation basée divergences

Dans un objectif de résolution de problèmes d'optimisation, nous avons proposé [20, 17, 19, 18, 14] une nouvelle méthode, appelée CDDS (Climbing Depth-bounded Discrepancy Search ou *recherche par progression de divergences limitées par la profondeur*). La méthode

CDDS s'appuie d'une part sur CDS pour explorer le(s) voisinage(s) d'une solution de référence et sur une méthode de type LDS dite *tronquée* dans le sens où les divergences sont limitées par un paramètre de profondeur. La variante considérée est sans redondance et considère en priorité les divergences dans le haut de l'arborescence. Par rapport au schéma de la figure 1.5 représentant l'exploration complète d'une arborescence à 3 variables binaires par ILDS, si on tronque l'application des divergences à la profondeur 1, les solutions numérotées 5, 6 et 8 ne sont pas générées. Ainsi cette méthode à divergences tronquée permet de limiter l'exploration des voisinages par la profondeur des divergences.

Au sein de la méthode CDS, la procédure $Generer\_Feuilles$ nécessite alors un paramètre supplémentaire donnant la profondeur maximale d jusqu'à laquelle les divergences sont autorisées. Avec la méthode CDDS, seules les solutions à k divergences situées au plus à une profondeur d sont déterminées. Ceci peut être envisagé de différentes manières :

- la profondeur d varie de 1 à k comme dans la méthode DDS. Dans ce cas, les branches à k divergences sont au plus à la profondeur k dans l'arbre de recherche. Ainsi, la recherche descend en profondeur dans l'arbre au fur est à mesure que la valeur k du nombre de divergences maximum augmente ;
- la profondeur d est limitée a priori par une valeur donnée (dépendant par exemple des caractéristiques du problème à résoudre). Dans ce cas, les branches à k divergences sont toutes au plus à la prodondeur d fixée dans l'arbre de recherche et seule la partie de l'arborescence située au dessus de cette profondeur d est explorée durant la résolution.

Améliorations de la méthode CDDS

Plusieurs mécanismes peuvent été greffés au principe de recherche de la méthode CDDS afin d'améliorer son efficacité en élaguant l'arbre de recherche parcouru. Nnous avons proposé d'une part d'intégrer des calculs de bornes inférieures de la fonction objectif (minimisation) à la méthode CDS afin d'élaguer des parties de l'espace de recherche [19, 20, 17]. D'autre part, dans [14, 18, 16, 83] nous avons défini des mécanismes permettant de limiter les points d'application des divergences. Ainsi certaines variables et/ou certaines valeurs sont considérées de manière prioritaire pour effectuer des divergences.

On peut trouver dans la littérature d'autres adaptations de la méthode CDS et de la méthode CDDS [58, 57]. Dans ces travaux, une méthode mixte entre CDS et CDDS est proposée. Cette méthode, appelé HD-CDS applique la méthode CDS jusqu'à un certain nombre de divergences. Une fois que ce nombre limite de divergences est atteint, seuls certains niveaux de l'arbre seront explorés avec ce nombre limite de divergences. Cette méthode a montré son efficacité pour la résolution de problèmes d'ordonnancement à machines parallèles avec contraintes de précédence et temps de préparation en considérant la minimisation de la durée totale de l'ordonnancement. De plus, des règles de dominance ont été introduites avec succès au sein d'une méthode à divergences afin d'élaguer certaines branches de l'arborescence.

1.3 Applications à la résolution de problèmes d'ordonnancement

Nous présentons dans cette partie, différentes applications des contributions exposées précédemment autour de la méthode CDS pour la résolution de problèmes d'ordonnancement. Les problèmes étudiés sont de type flowshop et jobshop et présentent des

contraintes spécifiques telles que des choix de ressources pour la réalisation des opérations, on parle alors de *problèmes d'ordonnancement avec flexibilité de ressources*, ou telles que des contraintes de délais minimum ou maximum entre certaines paires d'opérations, on parle alors de *problèmes d'ordonnancement avec time-lags*.

Les problèmes d'ordonnancement étudiés sont composés d'un ensemble J de n travaux ou jobs devant être réalisés sur un ensemble K de m ressources. Chaque job j comporte n_j opérations (ou tâches) devant être exécutées en séquence, notées $O_{ji}, \forall j \in J, \forall i \in [1, .., n_j]$. Chaque opération est réalisée sans interruption par une seule ressource. Les ressources sont disjonctives (on parle également de machines) et ne peuvent donc exécuter qu'une seule opération à la fois. Un job peut passer plusieurs fois par une même machine (recirculation) ou bien ne pas y passer du tout. Le flowshop est donc un cas particulier de jobshop.
Dans un problème de *flowshop*, tous les jobs passent sur les machines dans le même ordre, ie. la gamme de réalisation des jobs est unique. Dans un problème de *jobshop*, l'ordre de passage sur les machines peut varier d'un job à un autre, ie. les gammes de réalisation des jobs sont multiples.
Les machines sont dites *identiques* lorsque les durées des opérations ne dependent pas de la ressource allouée ; dans le cas contraire elles sont dites *indépendantes*.
Pour tous les problèmes d'ordonnancement considérés, l'objectif pris en compte pour la résolution est la minimisation de la date de fin au plus tard de réalisation de l'ensemble des jobs aussi appelé *makespan* ou $C_{\max}$.

Dans le cas d'ordonnancement flexible, la ressource nécessaire à l'éxécution d'une opération O_{ji} doit être choisie parmi un ensemble K_{ji} de m_{ji} ressources.
En ce qui concerne les contraintes de délais, aussi appelées *time lags*, elles correspondent dans les problèmes traités à une distance minimale ou maximale entre deux opérations consécutives d'un même job.

1.3.1 Adaptations de la méthode CDDS pour les problèmes de flowshop hybride

Pour le problème de flowshop avec flexibilité de ressources, aussi appelé *flowshop hybride* ou *Hybrid Flow Shop* (HFS), les ressources sont décomposées en étages. On dispose d'un ensemble E de l étages et chaque étage e comporte un ensemble K_e de m_e machines parallèles identiques. A chaque étage e chacun des jobs j est traité par une machine de e pendant une durée $p_{j,e}$ indépendante de la ressource allouée.
Résoudre un problème de flowshop hybride consiste à affecter dans chaque étage une machine à chaque opération de chaque job ainsi qu'à séquencer entre elles les différentes opérations des jobs. Le problème de HFS est NP-difficile même s'il ne contient que deux étages avec plus d'une machine sur un des étages [68].

Pour traiter ce problème à l'aide de la méthode CDDS proposée, nous avons défini deux types de variables [19] :

- $X_j^e \in J$: sélection de l'opération du j^{e} job à l'étage e ;
- $A_j^e \in K_e$: affectation de ressource pour l'opération du j^{e} job à l'étage e.

Différentes heuristiques d'instanciation ont été évaluées. Nous ne présentons ici que celle ayant conduit aux meilleurs résultats. Elle consiste à à procéder étage par étage. Puis à chaque étage, un job est sélectionné et l'opération correspondante de ce job est allouée à une machine donnée. Pour cela, l'heuristique d'instanciation se base sur :

- un ordre entre les jobs : la priorité est donnée au job correspondant à l'opération ayant la plus petite date de fin (*Earliest Start Time* ou EST). En cas d'égalité la priorité

est donnée au job ayant la plus grande durée (*Longest Job Duration* ou LJD) ;
- un ordre sur les machines : la machine sélectionnée est celle permettant la plus petite date de fin du job (*Earliest Completion Time* ou ECT).

Une fois les variables de sélection de job et d'allocation de ressources instanciées, un mécanisme de propagation de contraintes de type *Forward Checking* est appliqué. Il permet de mettre à jour les dates de début au plus tôt des opérations non encore ordonnancées ainsi que d'actualiser les dates de disponibilité des différentes machines.

Bien qu'il y ait deux types de variables, nous considérons que les divergences ne peuvent être appliquées que sur les variables de sélection des jobs. En effet, les machines étant identiques à chaque étage et l'heuristique choisissant la machine permettant de terminer au plus tôt, sélectionner une autre machine aurait un effet limité pour la minimisation du makespan. Dans nos travaux, faire une divergence pour la résolution du HFS consiste à sélectionner un autre job que celui préconisé par l'heuristique à un étage donné. Par ailleurs, nous avons considéré un mode de comptage binaire pour les divergences car celui-ci s'est avéré plus intéressant expérimentalement.
Afin de limiter l'exploration de l'arbre de recherche, nous avons intégré un calcul de bornes inférieures (problème de minimisation) au sein de la méthode CDDS au niveau de chaque nœud permettant la sélection d'un job suivi de l'affectation de ressource. La méthode CDDS avec utilisation de bornes inférieures ne diffère de la méthode CDDS qu'au niveau de la procedure $Generer\_Feuille$ qui compare à chaque nœud la borne inférieure à une borne supérieure fournie en paramètre (meilleure valeur du makespan obtenue lors d'une précédente itération). Dès que la valeur de la borne inférieure associée à un nœud atteind celle de la borne supérieure, le développement de la branche correspondante est stoppée.
Pour des raisons de simplicité et d'efficacité d'implémentation, la borne inférieure retenue pour le flowshop hybride à l étages est celle proposée dans [25, 132] même si elle n'est pas la plus performante. La méthode CDDS intégrant ce calcul de borne lors du développement de chaque nœud est notée CDDS^l par la suite.

Les expérimentations ont été réalisées sur un PC Intel Core 2 à 2,9 GHz et 2GB de RAM. Les algorithmes ont été implémentés en C. Les jeux de tests considérés sont ceux proposés par Vignier [149] et par Carlier et Néron [32].

Pour les instances de Vignier, le temps CPU alloué à la méthode CDDS^l est de 15 secondes (au delà de ce temps CPU, nous avons pu constater peu d'améliorations des résultats obtenus). Parmi les 36 instances les plus difficiles, CDDS^l les résout toutes contre seulement $27,78\%$ pour l'algorithme génétique de [149] et 75% pour le Branch and Bound de [123].

Pour les 76 instances de Carlier et Néron [32], le temps CPU alloué est de 120 secondes. Les premières expérimentations ont consisté à évaluer l'écart de l'heuristique d'instanciation par rapport aux bornes inférieures (LB) proposées dans [123] qui est de $2,30\%$.
Une deuxième étape d'expérimentation a permis d'évaluer les performances respectives de trois méthodes à divergences : CDDS, CDDS^l et LDS tronquée avec une limitation sur le temps CPU maximum alloué à la recherche. Pour ces trois méthodes, nous avons évalué l'amélioration en pourcentage qu'elles permettent par rapport au makespan de la solution initiale (cf tableau 1.2) ; pour réaliser cette comparaison les instances ont été séparées en deux groupes : les instances faciles (52 instances) et les instances difficiles (24 instances). Les trois méthodes étudiées parviennent à améliorer la solution initiale, et dans le cas de problèmes difficiles, c'est la méthode CDDS^l qui s'avère la plus performante suivi de CDDS et en dernière position LDS tronquée.

La dernière étape d'expérimentation a porté sur la comparaison de méthodes à di-

Méthodes	Problèmes faciles	Problèmes difficiles
LDS tronquée	9, 7	14, 7
CDDS	8, 0	17, 9
CDDS^l	8, 3	20, 2

TABLE 1.2 – Comparaison de méthodes à divergences pour le HFS : amélioration de la solution initiale (en %)

vergences avec d'autres méthodes de la littérature. Nous avons ainsi comparé les résultats obtenus par CDDS, CDDS^l, DDS, du Branch and Bound de Néron et. al [123] et une méta-heuristique basée sur les systèmes immunitaires artificiels (AIS) de Engin et Döyen [53] qui fournissait les meilleurs résultats pour les problèmes étudiés. Les résultats de ces méthodes sont calculés en termes d'écart (en pourcentage) à une borne inférieure (LB). Les résultats obtenus sont résumés dans le tableau 1.3, ils montrent l'efficacité des méthodes CDDS pour la résolution du HFS et en particulier l'efficacité de CDDS^l qui devance toutes les autres méthodes.

Méthodes	Problèmes faciles	Problèmes difficiles
Branch and Bound	2, 21	6, 88
AIS	1, 01	3, 12
LDS tronquée	1, 42	8, 01
CDDS	1, 10	4, 87
CDDS^l	0, 96	3, 06

TABLE 1.3 – Comparaison de différentes méthodes pour le HFS : écart à une borne inférieure (en %)

Les problèmes de HFS à 2 étages ont été également abordés [15]. Pour cela, nous avons intégré une heuristique d'instanciation spécifique et un calcul de borne inférieure dédiée à ce problème à 2 étages. L'heuristique d'instanciation est issue des travaux de Haouari et.al [70] et est une généralisation de la règle de Johnson. La borne retenue est celle proposée dans Haouari et M'Halla [71].
Afin de comparer l'influence respective de l'heuristique et de la borne inférieure nous avons intégré chacun de ces élements séparement au sein de la méthode CDDS. Ainsi, nous allons comparer :

- CDDS^2 : la méthode CDDS incluant une heuristique et un calcul de bornes dédiés au HFS à 2 étages ;
- CDDS^l : la méthode CDDS incluant une heuristique et un calcul de bornes dédiés au HFS général à l étages ;
- CDDS$^{l-LB(2HFS)}$: la méthode CDDS incluant une heuristique pour le HFS général et un calcul de bornes pour le HFS à 2 étages ;
- CDDS$^{2-LB(HFS)}$: la méthode CDDS incluant une heuristique pour le HFS à 2 étages et un calcul de bornes pour le HFS général.

Les expérimentations ont été menées sur trois types d'instances (A, B et C) générées de la même manière que [106]. Chaque type correspond à 560 instances. Le temps CPU alloué à la recherche de solution a été fixé expérimentalement à 15 secondes. Le tableau 1.4 compare les performances des quatre méthodes pour chaque type d'instances. Il donne l'écart (en pourcentage) à la borne inférieure dédiée au 2HFS, le pourcentage de problèmes résolus et, entre paranthèses, le temps de calcul (en seconde) nécessaire à la résolution de l'ensemble des instances de chaque type.
Pour la résolution de ces instances, la méthode utilisant une heuristique et un calcul de borne dédiés au HFS à 2 étages se montre la plus performante. De plus, ces résultats montrent que l'utilisation d'une heuristique dédiée au problème considéré a un plus grand

Méthodes	Type A	Type B	Type C
CDDS^2	0,19 - 90% (87)	0,17 - 91% (73)	0,26 - 86% (88)
$\text{CDDS}^{2-LB(HFS)}$	0,37 - 85% (120)	0,26 - 87% (87)	0,30 - 81% (115)
CDDS^l	0,96 - 69% (1633)	0,89 - 73% (141)	0,41 - 55% (207)
$\text{CDDS}^{l-LB(2HFS)}$	0,91 - 70% (152)	0,85 - 74% (134)	0,40 - 58% (191)

TABLE 1.4 – Comparaison de méthodes CDDS pour le 2HFS : écart à une borne inférieure (en %) et temps CPU total (secondes)

impact sur la qualité de la résolution qu'un calcul de borne inférieure spécifique.

Sur ces mêmes instances, nous avons également comparé la méthode CDDS^2-LB à une méta-heuristique de type recherche Tabou (TS) proposée dans Haouari et M'Hallah [71]. Les résultats obtenus montrent que la méthode CDDS^2 s'avère très efficace avec un écart moyen à la borne inférieur de $0,22\%$ sur toutes les instances contre un écart moyen de $0,82\%$ pour la méthode TS. La méthode CDDS^2 atteint la solution optimale pour $85,8\%$ des instances en un temps moyen de $2,26$ secondes par instance alors que la méthode TS atteint l'optimum pour seulement 35% des instances mais en un temps beaucoup plus bref ($0,03$ seconde en moyenne par instance en utilisant les coefficients de normalisation de Dongarra [50, 51].

L'utilisation de méthodes basées divergences pour la résolution de problèmes de flowshop hybride (dans le cas général à l étages ou dans le cas à 2 étages) s'est avérée très pertinente. Les résultats obtenus par la méthode CDDS proposée sont compétitifs avec les meilleures méthodes de la littérature et l'intégration d'un calcul de bornes inférieures au sein de la méthode permet d'en faire une des méthodes les plus performantes pour les instances traitées. L'intérêt de la méthode à divergences proposée réside dans l'exploration de voisinage de taille variable et dans l'utilisation de bornes permettant d'élaguer rapidement des voisinages ne conduisant pas à de bonnes solutions.

1.3.2 Adaptation de la méthode CDDS pour les problèmes de jobshop flexible

Les problèmes de jobshop avec flexibilité de ressources (*Flexible Job Shop* ou FJS) sont une généralisation des problèmes de jobshop classiques dans lesquels chaque opération doit être réalisée par une machine donnée à choisir parmi un ensemble de machines possibles. Contrairement au cas du flowshop hybride, on suppose ici que la durée p_{ji} de réalisation d'une opération O_{ji} dépend de la machine qui lui est affectée. Les machines sont alors dites non-reliées. Résoudre un FJS consiste à déterminer une allocation des machines aux différentes opérations et à déterminer la séquence des opérations en optimisant un objectif donné (ici la durée totale de l'ordonnancement). Le FJS est NP-difficile à partir de 2 machines et de 3 jobs [60].

Pour résoudre un FJS à l'aide de la méthode CDDS proposée nous avons défini deux types de variables [14] :

- X_r avec $1 \leq r \leq \sum_{j \in J} n_j$ représentant la sélection de la r^{e} opération, avec $X_r \in \{O_{ji}, j \in J, i \in n_j\}$;
- A_r avec $1 \leq r \leq \sum_{j \in J} n_j$ traduisant l'affectation d'une machine à la r^{e} opération, avec $A_r \in K_r$ où K_r représente l'ensemble des machines possibles pour la r^{e} opération.

Le principe général de l'heuristique d'instanciation proposée consiste à sélectionner tout d'abord une première opération (X_1) puis à lui allouer une ressource (A_1) et à fixer la date de début de l'opération sélectionnée au plus tôt compte tenu de la ressource allouée (en raison de l'objectif de minimisation de la durée totale de l'ordonnancement). Ce processus est répété jusqu'à ce que toutes les opérations soient placées dans l'ordonnancement. Pour

cela, l'heuristique d'instanciation se base sur différents ordres dont les performances ont été évaluées expérimentalement [18] :

- un ordre sur les opérations : la priorité est donnée à l'opération ayant la plus petite date de début au plus tôt (*Earliest Start Time* ou EST) et en cas d'égalité à l'opération ayant la plus petite date de fin au plus tard (*Earliest Due Date* ou EDD) ;
- un ordre sur les machines : la machine sélectionnée est celle offrant la plus petite date de fin de l'opération (*Earliest Completion Time* ou ECT).

Comme pour la résolution du problème de flowshop hybride, une fois les variables instanciées, un mécanisme de propagation de contraintes de type Forward Checking est appliqué afin d'actualiser les dates de début des opérations non encore sélectionnées ainsi que les dates de disponibilité des machines.

Les divergences considérées peuvent être appliquées sur les deux types de variables prises en compte : sélection d'opération et affectation de ressource. Une divergence sur une variable de sélection d'opération consiste à choisir une opération différente de celle proposée par l'heuristique, quelle que soit l'opération différente retenue, le choix ne correspond qu'à une seule divergence. Autrement dit, on effectue un comptage binaire des divergences sur les variables de sélection d'opération. Une divergence sur une variable d'affectation de ressource consiste à choisir une autre ressource que celle préconisée par l'heuristique, pour ces variables le mode de comptage des divergences est également binaire : toute autre ressource choisie correspond à une seule divergence.

Nous avons étudié dans [18] trois stratégies d'application des divergences :

- Stratégie S_1 : effectuer des divergences uniquement sur les variables de sélection d'opération ;
- Stratégie S_2 : effectuer des divergences uniquement sur les variables d'affectation de ressource ;
- Stratégie S_3 : combiner les divergences sur les deux types de variables en alternant les stratégies S_1 et S_2.

Nous avons comparé notre méthode CDDS avec ces trois stratégies pour résoudre différentes instances de FJS :

- un ensemble de 10 instances à machines non reliées proposées par Brandimarte [26], cet ensemble est noté **BRdata** par la suite. Le nombre de jobs de ces instances varie entre 10 et 20 et le nombre de machines varie entre 4 et 15. Le nombre de machines possibles par opération varie entre 1,43 et 4,10 avec une moyenne à 2,59.
- trois ensembles de 43 instances à machines équivalentes proposés par Hurink [86]. Les trois ensembles d'instances de [86] sont construits à partir d'instances classiques de jobshop de Fisher et Thompson (ft06, ft10 et ft20) et de Lawrence (la01 à la40). Ils se distinguent par le nombre moyen de ressources possibles pour chaque opération, ces ensembles sont notés par la suite **Edata**, **Rdata** et **Vdata**. Pour l'ensemble Edata, la flexibilité des ressources est faible car il y a en moyenne 1,15 machines possibles par opération ; elle est un peu plus élevée pour les instances Rdata (nombre moyen de machines par opération égal à 2) et elle est élevée pour les instances Vdata (nombre moyen de machines par opération comprise entre 2,5 et 7,5).

Un temps de calcul de 1800s a été alloué à la méthode CDDS développée en C. Les tests ont été effectués sur un Pentium IV à 3,2 GHz avec 448 Mo de RAM sous Windows XP.

Les résultats obtenus ont été comparés à trois méthodes parmi les plus performantes de la littérature : une méthode Tabou (TS) proposé par Mastrolilli et Gambardella [114]

et deux algorithmes génétiques (GA et hGA) proposés respectivement par Pezzella et. al [129] et par Gao et. al [59]. Les résultats de ces expérimentations sont résumés dans le tableau 1.5 où la première colonne donne le nom des types de problèmes, la deuxième colonne donne le nombre d'instances, la troisième colonne montre le nombre moyen de ressources par opération et les quatre dernières colonnes fournissent les résultats obtenus par les différentes méthodes CDDS, TS, GA et hGA en termes de déviation par rapport à une borne inférieure de la littérature [114], en gras on met en évidence le meilleur écart pour un jeu d'instances donné. Ces expérimentations ont montré l'avantage de la stratégie S_3 par rapport aux deux autres. Néanmoins, les résultats obtenus avec la méthode CDDS reste en retrait par rapport aux meilleures méthodes de la littérature (TS et hGA). Une autre limite de la méthode CDDS proposée réside dans sa consommation en temps de calcul.

Problèmes	Nb instances	Flexibilité	CDDS	TS	GA	hGA
BRdata	10	2,59	17,3	15,14	17,53	**14,92**
Edata	43	1,15	5,3	2,17	6,0	**2,13**
Rdata	43	2	2,5	1,24	4,42	**1,19**
Vdata	43	4,31	0,6	0,095	2,04	**0,082**
		Déviation moyenne	3,84	2,17	5,12	2,13

TABLE 1.5 – Comparaison de CDDS avec d'autres méthodes pour la résolution du FJS

Notons également que pour le FJS, il n'existe pas dans la littérature de formule analytique pour évaluer une borne inférieure du makespan. L'utilisation de ces bornes inférieures dont l'efficacité a été montrée dans le cas de la résolution de flowshop hybride ne peut pas être envisagée ici.

Pour faire face à ces difficultés rencontrées par la méthode CDDS, nous avons proposé :
- de définir des divergences de manière plus complexe qu'un simple changement de valeur pour une variable en considérant de manière agrégé différents changements d'affectation et de séquencement.
- d'effectuer des divergences uniquement sur certaines variables pouvant s'avérer pertinentes par rapport à une amélioration du critère considéré.

Des travaux préliminaires avaient été effectués dans le cadre d'une méthode LDS et avaient montré l'intérêt du guidage des choix de points de divergences [83]. Nous avons donc sélectionné des ensembles de variables sur lesquelles l'application de divergences pouvait avoir un intérêt du point de vue du critère que l'on cherche à optimiser (ie. la durée totale de l'ordonnancement). Pour cela, nous avons utilisé la notion de bloc défini dans [88] de la manière suivante : Un *bloc* est une succession d'au moins deux opérations critiques exécutées sur la même ressource et une opération est dite *critique* si elle appartient au chemin critique, c'est-à-dire au plus long chemin. Via cette notion de bloc, des propriétés entre solutions peuvent être définies. Soient x et x', deux solutions d'un problème d'ordonnancement telles que la valeur du critère C_{max} pour x' soit strictement inférieure à la valeur du critère C_{max} pour x :
- il existe dans x' au moins une opération d'un bloc B de x qui est exécutée sur une autre machine que dans la solution x ;
- il existe dans x' au moins une opération d'un bloc B de x (différente de la première opération de B) qui est exécutée avant la première opération de B ;
- il existe dans x' au moins une opération d'un bloc B de x (différente de la dernière opération de B) qui est exécutée après la dernière opération de B.

En exploitant ces propriétés, à partir d'une solution donnée, les divergences sont appliquées

sur les opérations des différents blocs du (des) chemin(s) critique(s) et vont consister soit à changer l'affectation pour une opération donnée soit à permuter la réalisation des autres opérations du bloc. Dans l'heuristique d'instanciation, les priorités des autres opérations restent inchangées dans le cas d'une application « statique » soit re-évaluées dynamiquement.

Cette nouvelle variante de la méthode CDDS, notée *CDDS-HDiv*, a été évaluée sur les instances précédemment utilisées (BRdata de Brandimarte ainsi que les ensembles Edata, Rdata et Vdata de Hurink) ainsi que d'autres instances :

- un ensemble de 18 instances proposées par [40], noté **DPdata**, comportant des problèmes à machines identiques et des problèmes à machines indépendantes. Le nombre de jobs varie entre 10 et 20, le nombre de machines entre 5 et 10 et le nombre moyen de machines par opération est compris entre 1,13 et 5,02.
- un ensemble de 21 instances à machines identiques, noté **BCdata**, définies par [10] construites à partir de trois problèmes classiques de jobshop (mt10, la24 et la40). Ces instances comportent de 10 à 15 jobs et de 11 à 18 machines. Le nombre moyen de machines par opération varie entre 1,07 et 1,3.

Les expérimentations ont été réalisées sur un PC Intel Core 2 Duo à 2,9 GHz et 2 GB de mémoire vive sous Windows XP. Les temps de calcul alloués à la méthode CDDS-HDiv étaient de 15 secondes sauf pour les instances **DPdata** pour lesquelles les temps limites ont été fixés à 200 secondes (les valeurs des durées opératoires de ces instances augmentent de manière significative le temps de calcul nécessaire à leur résolution). Les résultats obtenus sont résumés dans tableau 1.6. Ils montrent que la méthode CDDS-HDiv proposée obtient toujours de meilleurs résultats que la méthode GA de [129], elle est plus performante que la méthode hGA de [59] pour les instances **BCdata** et **DPdata** et plus performante que la méthode TS de [114] sur les instances **BRdata** et **DPdata** (pour les instances BCdata, elle est plus mauvaise sur seulement une seule instance). Globalement, il n'y a que pour les instances de **Hurink** (**Edata**, **Rdata** et **Vdata**) [86] que la méthode CDDS-HDiv n'améliorent aucune des méthodes existantes.

Problèmes	Nb instances	Flexibilité	CDDS-HDiv	TS	GA	hGA
BRdata	10	2,59	14,98	15,14	17,53	**14,92**
BCdata	21	1,17	22,54	**22,53**	29,56	22,61
DPdata	18	2,49	**1,94**	2,01	7,63	2,12
Edata	43	1,15	2,32	2,17	6,0	**2,13**
Rdata	43	2	1,34	1,24	4,42	**1,19**
Vdata	43	4,31	0,12	0,095	2,04	**0,082**
Déviation moyenne			4,61	4,56	8,25	4,54

TABLE 1.6 – Comparaison de CDDS-HDiv avec d'autres méthodes pour la résolution du FJS

Au niveau de la comparaison en temps de calcul des différentes méthodes, il faut noter que les résultats des méthodes TS, GA et hGA sont fournis en moyenne sur plusieurs exécutions contrairement à la méthode CDDS-HDiv. Néanmoins afin d'avoir une idée des temps de calcul des différentes méthodes nous donnons dans le tableau 1.7 quelques éléments de comparaison. Les temps de calcul sont exprimés en secondes et ont été normalisés avec les coefficients de [50, 51]. Les temps de calcul correspondent au temps nécessaire pour résoudre un ensemble d'instances et entre parenthèses on fournit également le temps moyen. Seules les méthodes CDDS-HDiv, TS et hGA ont pu être comparées car les auteurs de la méthode GA ne fournissent pas leur temps de calcul par type d'instances.

Au vu de l'ensemble de ces résultats, la méthode CDDS-HDiv s'avère être une approche pertinente par rapport aux autres méthodes de la littérature.

Problèmes	CDDS-HDiv	TS	hGA
BRdata	385 (96)	370 (74)	455 (91)
BCdata	1 012 (253)	1 780 (356)	4 105 (821)
DPdata	11 560 (2890)	12 335 (2467)	31 030 (6 206)
Hurink	5 570 (1 393)	82 840 (16 568)	353 725 (70 745)

TABLE 1.7 – Synthèse des résultats en termes de temps de calcul (en secondes)

1.3.3 Apport des heuristiques à pondération pour la résolution de jobshop avec contraintes de délais

Dans ce travail, nous nous sommes intéressés à l'utilisation d'une heuristique basée sur la pondération de variables au sein de la méthode CDS. L'objectif est de déterminer une solution optimale (ie minimiser le makespan) à un problème de jobshop avec contraintes de délais ou *time-lags* [94, 93]. Les problèmes considérés sont tels que les contraintes de time lags ne sont présentes qu'entre deux opérations d'un même job. On notera $TL^{min}_{(j,i,i+1)}$ (respectivement $TL^{max}_{(j,i,i+1)}$) le délai minimal (respectivement maximal) entre deux opérations consécutives $O_{j,i}$ et $O_{j,i+1}$, avec $i \in [1,..,n_j]$.

Une difficulté spécifique des problèmes de jobshop avec time-lags provient de la contrainte de time-lags maximum qui peut créer une impossibilité de déplacer une opération plus tard sur une ressource, ce qui resserre les créneaux disponibles pour l'insertion des opérations. Les problèmes contenant uniquement des contraintes liées aux time lags minimum sont moins problématiques car de telles contraintes peuvent être absorbées en rallongeant la durée de l'opération précédente.

L'heuristique d'instanciation utilisée est celle présentée dans le chapitre 2. Elle consiste à sélectionner progressivement les jobs et à placer au plus tôt l'ensemble des opérations du job sélectionné sur les machines en respectant les contraintes de disponibilité des ressources et de séquencement entre ces opérations. En cas d'insatisfiabilité pour le placement d'une opération sur une machine, celle-ci est décalée dans le temps. Ce décalage d'une opération donnée peut entraîner le décalage de l'opération précédente afin de respecter les contraintes de time lags. Dans le pire des cas, les différents jobs sont ordonnancés les uns à la suite des autres. Cette heuristique s'appuie sur différents classements des jobs (en fonction de leur durée, de leur time lags ou d'une composition entre les deux) qui sont statiques.
Lors du déroulement de la méthode CDS avec cette heuristique, les variables à instancier représentent la sélection des jobs, ce sont donc des variables n-aires dont le domaine est l'ensemble des jobs à placer dans l'ordonnancement. La sélection d'un job s'effectue en fonction d'un classement sur les jobs (durée, time lags, ...) et une divergence consiste alors à sélectionner un autre job.

Nous avons considéré une heuristique de pondération des valeurs permettant de classer les jobs entre eux. A tout job j, on associe initialement un même poids w_j.

Lors du placement sur les machines des opérations du job j sélectionné par l'heuristique, certaines opérations peuvent être décalées dans le temps compte tenu des contraintes de time lags. Ces décalages dans le temps des différentes opérations d'un job peuvent être interprétées comme des "échecs" par rapport au fait de pouvoir positionner ces opérations au plus tôt et vont entraîner des incrémentations du poids du job correspondant. Plusieurs modes d'incrémentation des poids ont été définis :

- chaque décalage d'une opération du job j provoque l'incrémentation de w_j ;
- w_j est incrémenté à chaque fois qu'une des opérations du job j n'est pas placée au plus tôt sur la machine qu'elle nécessite. Avec ce mode d'incrémentation des poids, il

y a au plus une incrémentation par opération (ou par machine si toutes les opérations demandent des machines distinctes).

- w_j est incrémenté quand il y a au moins un décalage sur une des opérations du job j. Ce mode d'incrémentation est un mode binaire : soit il y a au moins un décalage et le poids est incrémenté de 1 soit il n'y a aucun décalage des différentes opérations et le poids du job n'est pas incrémenté.

Lors d'une itération de la méthode CDS, la méthode recherche les solutions voisines ayant k divergences par rapport à la solution courante. Plusieurs feuilles vont être générées et lors du calcul de chaque feuille, différentes pondérations vont être associées aux jobs. Avant de passer à la prochaine itération, les différentes pondérations sur les jobs doivent être combinées entre elles en prenant pour chacun des jobs soit la somme de toutes les pondérations obtenues soit le maximum des pondérations obtenues. Lors du calcul des solutions à $k+1$ divergences, la méthode à divergences développée est une variante redondante réalisant des divergences en priorité en haut (variante LDS).

Les expérimentations ont été menées sur des instances de problèmes de jobshop avec time-lags proposées par Caumond et. al [37]. Les instances utilisées correspondent à des problèmes classiques modifiés pour y introduire des contraintes de time-lags. Seules les contraintes de time-lags maximum ont été introduits par [37] par le biais d'un coefficient représentant l'importance de la contrainte de time-lags (plus ce coefficient est grand et moins la contrainte de time-lags maximum est forte, ie. le problème se rapproche d'un problème de jobshop classique). De plus, dans les instances proposées, les valeurs des contraintes de time-lags maximum sont identiques pour toutes les opérations d'un même job.
Nous avons considéré par la suite 20 instances de Lawrence (de la01 à la20) et 7 coefficients de time-lags $\alpha \in \{0,25; 0,5; 1; 2; 3; 5; 10\}$. Les méthodes à divergences utilisées sont soit la méthode CDS sans pondération, soit la méthode CDS avec un des trois modes d'incrémentation des pondérations combiné avec un des deux modes de cumul des pondérations. Les résultats de ces méthodes sont comparées aux résultats obtenus par Ilog Scheduler dans un même temps limite de 200 secondes. Les résultats obtenus [94, 93] montrent que la méthode CDS sans pondération ainsi que la méthode CDS avec le deuxième mode d'incrémentation des pondérations (quelle que soit la manière de cumuler les pondérations) sont les meilleures variantes. Cependant elles ne sont plus performantes que Ilog Scheduler que pour 14 instances (sur les 140 testées) et pour ces 14 instances l'amélioration de l'écart à une borne inférieure est de 7,81% entre les méthodes à divergences et Ilog Scheduler.

1.4 Evaluation de l'impact des modes de comptage des divergences

Des expérimentations ont été effectuées pour comparer des méthodes à divergences intégrant un comptage binaire, non binaire ou mixte dans un contexte de satisfaction de contraintes. Les conclusions de ces expérimentations sont liées aux instances utilisées pour les tests (voir tableau 1.8). Le comptage non binaire s'est avéré très performant (couplé à des heuristiques de pondération) pour la résolution de CSP aléatoires en améliorant les performances d'une méthode MAC (utilisant les mêmes heuristiques). Pour la résolution de problèmes de car-sequencing, le comptage non binaire s'est également avéré plus performant mais l'écart avec le comptage binaire est moins important [98, 92].

Par rapport au mode de comptage mixte, les résultats obtenus sont mitigés : aucun

apport n'a pu être noté sur les instances de car-sequencing mais pour les instances de CSP aléatoires un mode de comptage mixte (non binaire dans le haut de l'arborescence et binaire dans le bas avec un changement de mode de calcul à la profondeur de 1/5) améliore nettement la qualité des résultats obtenus [91]. Des résultats plus détaillés sont présentés dans la partie 1.5.

Problèmes \ Modes de comptage	binaire	non-binaire	mixte	
Car sequencing		X	x (NB-N-1/5)	Satisfaction
CSP aléatoires		X	x	
Ordonnancement	X			Optimisation

TABLE 1.8 – Efficacité des modes de comptage

Pour la résolution de problèmes d'optimisation tels que des problèmes d'ordonnancement (avec des contraintes spécifiques sur la flexibilité des ressources ou de délais minimum et maximum entre opérations), le mode de comptage binaire s'est avéré plus compétitif. Ces résultats ont été confirmés par des travaux d'autres auteurs [57] pour la résolution de problèmes d'ordonnancement avec temps de préparation entre opérations et machines parallèles.

Avec le solveur MISTRAL, différentes méthodes à divergences (LDS, red-DDS et PRLDS) ont été implémentées avec trois modes de comptage des divergences (binaire, non binaire et par niveau) et comparées avec une méthode déjà implémentée et connue pour ses performances : parcours en profondeur d'abord avec retour-arrière chronologique (backtrack chronologique) et redémarrages.

Les premières expérimentations ont été réalisées sur 500 instances de quasi-groupes issues de la CSPlib (de taille 20×20 avec 166 trous) et sur 88 instances de quasi-groupes plus difficiles (taille 25×25 avec 264 trous). Pour toutes ces méthodes, les meilleurs résultats ont été obtenus avec l'heuristique dynamique *dom/wdeg* ce qui confirme les travaux de [24]. Le tableau 1.9 récapitule le classement en termes de temps de calcul de ces trois méthodes en fonction du mode de comptage des divergences. Le comptage non binaire est le plus performant pour les méthodes LDS et red-DDS (méthodes avec divergences en priorité en haut) alors que c'est le comptage par niveau qui s'avère le plus performant pour PRLDS (méthode avec divergences en priorité en bas). Cependant, les différentes méthodes à divergences testées n'obtiennent pas de meilleurs temps de calcul que la méthode à redémarrage de référence.

Méthodes \ Modes de comptage	niveau	non-binaire	binaire
LDS	2	1	3
red-DDS	2	1	3
PRLDS	1	2	3

TABLE 1.9 – Efficacité des modes de comptage pour la résolution de problèmes de quasigroup

Les méthodes à divergences LDS, red-DDS et PRLDS combinées avec les trois modes de comptage (binaire, non binaire et par niveau) ont été incluses dans la méthode CDS pour la résolution de problèmes d'ordonnancement de type jobshop (20 instances de Lawrence

et 15 instances plus difficiles de Taillard) disponibles sur la ORLib.
Le modèle de jobshop utilisé est un modèle à variables binaires, ainsi les modes de comptage binaire et non binaire reviennent au même. Les résultats de ces méthodes ont été évalué en termes d'obtention de l'optimum dans un temps limite imparti.

Pour les instances de Lawrence, les modes de comptage les plus efficaces sont le mode par niveau pour PRLDS et red-DDS et le mode binaire (ou non binaire) pour LDS. Cependant pour les instances de Taillard, c'est le mode de comptage par niveau qui est le plus performant pour l'ensemble des méthodes.
On peut également noter que les trois méthodes restent en retrait de la méthode à redémarrage, red-DDS étant la moins performante.
Les faibles performances de nos variantes de CDS doivent être pondérés par le fait que les méthodes à divergences utilisées sont des variantes redondantes alors que la nécessité de complétude n'est pas avérée pour la recherche d'une solution de bonne qualité par rapport à un objectif donné.

On peut remarquer au vu de ces expérimentations que le mode de comptage non-binaire semble être le plus performant pour les problèmes de satisfaction alors que pour les problèmes d'optimisation le comptage par niveau semble le plus adapté (nous n'avons pas pu comparé le comptage binaire et non-binaire dans ce cas car le modèle utilisé pour le problème de jobshop est un modèle à variables binaires).

En couplant les résultats obtenus sur les modes de comptage des divergences avec les variations de l'incrémentation sur le nombre de divergences, on peut noter que, pour les trois méthodes, le temps de résolution décroit en fonction de la valeur de cet incrément jusqu'à une valeur limite.
Par ailleurs, pour les méthodes LDS et red-DDS, le classement des méthodes en fonction des modes de comptage ne dépend pas de la valeur de l'incrément : le comptage non binaire est toujours plus performant que le comptage par niveau, lui-même plus performant que le comptage binaire. En revanche, pour la méthode PRLDS, cela ne se vérifie pas : en fonction de la valeur de l'incrément sur les divergences le classement entre les différents modes de comptage des divergences n'est pas le même (pour de petites valeurs d'incrémentation le comptage par niveau est le plus efficace mais pour de plus grandes valeurs, c'est le comptage non-binaire qui l'emporte).

1.5 Applications à la résolution de CSP

Cette partie est consacrée à l'évaluation de différentes variantes des méthodes à divergences dans un contexte de satisfaction de contraintes [91]. Pour cela nous avons considéré deux types de problèmes particuliers qui sont le car-sequencing et les CSP aléatoires.

1.5.1 Apport des heuristiques à pondération pour la résolution de problèmes de car-sequencing

Le problème de car-sequencing est un problème d'ordonnancement particulier dont la résolution vise à déterminer l'ordre selon lequel des voitures doivent être réalisées sur une chaîne de fabrication [75, 142, 23]. Ce problème a fait l'objet d'une compétition de la ROADEF en 2005 [144].
Le problème de car-sequencing se concentre sur une partie de la fabrication des véhicules, appelée montage, dans laquelle différents éléments vont être ajoutés aux véhicules au sein

de différentes unités de montage. Lors de cette étape, les voitures nécessitant des options particulières (air bag, toit ouvrant, etc.) vont entrainer une plus grande charge de travail dans l'unité correspondante. Ces véhicules doivent donc être répartis dans la séquence des véhicules à fabriquer de telle sorte que la charge de travail ne dépasse pas la capacité des différentes unités de montage. Chaque voiture ne demande pas toutes les options mais une liste d'options spécifiques. Chaque unité est caractérisée par une capacité limitée représentant le fait qu'elle peut produire au maximum r voitures avec l'option donnée sur une séquence de s voitures et nécessite un temps pour mettre au point l'option qui lui est associée.

Les instances de car-sequencing utilisées sont 70 instances satisfiables à 200 véhicules et 5 options de la CSPLib issues de [143]. Sur les 70 instances considérées, nous avons comparé l'apport de différentes heuristiques d'ordre sur les variables et sur les valeurs au sein de la méthode backtrack chronologique (BT) et au sein d'une méthode de recherche à divergences limitées (LDS). Dans ce contexte, les divergences ont été appliquées en priorité à partir du bas de l'arborescence. Les propagations utilisées au sein de CB et de LDS sont du type forward-checking dans le sens où l'on propage la modification aux variables voisines et qu'on ne répercute pas ces modifications sur le reste du CSP. Nous avons en effet choisi de nous limiter à des propagations plus simples que celles proposées dans [75] pour nous concentrer sur l'évaluation des méthodes et des heuristiques.
Les heuristiques de variable utilisés sont *Lexico* ou $wvar \oplus Lexico$ et les heuristiques de valeurs sont $MaxOpt$ (valeur correspondant à la voiture nécessitant le plus d'option) couplées à une des heuristiques de pondération de valeurs présentées en partie 1.2 et notées ici $wval1$, $wval2$ et $wval3$.

H. val. : $MaxOpt\oplus$ / H. var. :	*Lexico*	$wval1$	$wval22$	$wval3$	
Lexico	36 (3,08)	36 (3,08)	36 (3,08)	36 (3,08)	BT
$wvar \oplus Lexico$	37 (12,54)	36 (10,27)	36 (5,13)	36 (5,05)	
Lexico	41 (9,87)	41 (9,75)	41 (9,85)	41 (9,85)	LDS Binaire
$wvar \oplus Lexico$	59 (10,42)	59 (10,32)	59 (10,33)	59 (10,42)	
Lexico	38 (11,81)	42 (17,21)	47 (15,06)	46 (19,56)	LDS Non Binaire
$wvar \oplus Lexico$	65 (9,61)	65 (9,44)	65 (9,46)	65 (9,64)	

TABLE 1.10 – Synthèse des résultats obtenus pour la résolution de problèmes de car-sequencing

Les résultats obtenus sont fournis dans le tableau 1.10 où l'on donne le nombre de problème résolus (dans un temps limite fixé ici à 200 secondes) et le temps CPU moyen en secondes. En ce qui concerne la méthode LDS (avec un mode de comptage binaire ou non binaire), utiliser une heuristique à pondération de variables améliore les performances, mais ceci ne se vérifie pas pour les heuristiques à pondération de valeurs. Par rapport à la méthode BT, on peut également noter que pour le problème de car-sequencing les heuristiques de pondération de valeurs et de variables n'apporte rien. De plus, les résultats obtenus par la méthode BT sont améliorés par la méthode LDS avec comptage binaire ainsi qu'avec la méthode LDS avec comptage non-binaire. La méthode la plus performante étant, celle à comptage non-binaire. Les expérimentations menées sur des méthodes LDS avec comptage mixte n'ont pas été concluantes.
En augmentant le temps alloué maximum à 600s, la méthode LDS en binaire résout 62 problèmes (avec un temps de calcul moyen de 26,75s) et la méthode LDS en non binaire reste la plus performante en résolvant 67 problèmes avec un temps de calcul moyen de 21,76s.

1.5.2 Apport des heuristiques à pondération pour la résolution de CSP aléatoires

Pour l'obtention de CSP aléatoires, nous avons utilisé le générateur de CSP de modèle B [Frost 1996]. Ces instances de CSP sont caractérisées par : n le nombre de variables, d la taille des domaines, p_1 la densité du problème correspondant au rapport du nombre de contraintes dans le graphe de contraintes et du nombre de toutes les contraintes envisageables, et p_2 la dureté du problème correspondant au rapport du nombre de n-uplets incompatibles et du nombre de tous les n-uplets envisageables. Les instances de CSP aléatoires générées comportent soit 30 variables avec une taille de domaine fixe de 25 et une densité $p_1 = 0,16$ soit 40 variables avec une taille de domaine fixe de 20 et une densité $p_1 = 0,09$. Pour chaque triplet (n, d, p_1), on fait varier la dureté du problème p_2, de manière à obtenir des instances autour du pic de complexité et on considère 12 ensembles d'instances comportant chacun 100 CSP.

Les résultats obtenus sont résumés dans les tableaux 1.11 et 1.12 qui donnent pour chaque méthode le temps CPU moyen en millisecondes ainsi que le nombre moyen de nœuds développés (les moyennes sont effectuées sur les 100 problèmes générés pour les 12 valeurs de la dureté p_2). L'heuristique proposée *wvar* est comparée à l'heuristique de variables *wdeg* ; elles sont pour cela intégrées soit au sein de la méthode MAC (BT incluant des propagations de type arc-consistance) soit au sein d'une méthode LDS avec comptage binaire soit au sein d'une méthode LDS avec comptage non binaire. La propagation de contraintes utilisée dans les méthodes LDS est du type arc-consistance et est réalisée par le même algorithme que pour la méthode MAC. Les méthodes LDS employées s'appuient sur le mécanisme de restriction du nombre de divergences exposé dans la partie 1.2 afin d'interrompre la méthode en cas de détection d'une incohérence.

Méthodes \ Heur. Var.	*dom/wdeg*	*dom/wvar*
MAC	1 104,82 5 317 788	985,55 4 796 890
LDS Binaire	763,82 10 786 170	914,64 8 112 428
LDS Non Binaire	343,18 1 437 248	198,27 1 053 227

TABLE 1.11 – Synthèse des résultats obtenus pour la résolution de CSP aléatoires (30 - 25 - 0,16)

Méthodes \ Heur. Var.	*dom/wdeg*	*dom/wvar*
MAC	2 592,05 139 536 454	2 226,36 122 489 091
LDS Binaire	6 187,27 715 949 948	4 689,09 293 135 194
LDS Non Binaire	1 016,45 68 892 963	448,18 57 602 380

TABLE 1.12 – Synthèse des résultats obtenus pour la résolution de CSP aléatoires (40 - 20 - 0,09)

Ces expérimentations ont montré que l'heuristique *dom/wvar* proposée est nettement plus performante que l'heuristique *dom/wdeg* lorsqu'elle est intégrée dans une méthode LDS à comptage non binaire. Utilisée au sein de la méthode MAC, elle procure également des gains de performance mais ils sont d'une plus faible importance, en revanche si elle

est utilisée dans une méthode LDS avec comptage binaire, elle n'entraine pas forcément un gain de performance : par exemple elle nécessite un temps CPU plus important sur les CSP (30 - 25 - 0,16).
Le deuxième résultat obtenu, est le fait que la méthode LDS avec comptage non-binaire surpasse la méthode MAC pour la résolution de CSP aléatoires quelle que soit l'heuristique d'instanciation utilisée ($dom/wvar$ ou $dom/wdeg$). En revanche, ce résultat ne se retrouve pas pour la méthode LDS avec comptage binaire.

1.6 Conclusion

Dans ce chapitre, nous avons présenté nos contributions sur les méthodes de recherche arborescente et plus spécifiquement sur les méthodes à divergences.

La proposition d'une méthode à divergences dédiée à la résolution de problèmes d'optimisation nous a permis d'obtenir des résultats de très bonne qualité. Adaptée à la résolution de problèmes d'ordonnancement flexibles, la méthode proposée permet de concurrencer, voire de surpasser pour certaines instances, les meilleures méthodes de la littérature.
L'efficacité de la méthode proposée est due à la fois à l'efficacité de l'exploration des voisinages exprimés en termes de divergences, à l'utilisation d'heuristiques d'instanciation pertinentes pour le problème visé, à l'utilisation de bornes pour élaguer l'arbre de recherche ainsi qu'à la définition d'heuristiques pour guider l'application des divergences.

Une autre contribution a porté sur la proposition d'heuristiques d'instanciation basées sur la pondération de variables. Cette heuristique s'est avérée performante pour la résolution de problèmes de satisfaction de contraintes, qu'elle soit incluse dans une méthode à divergences ou dans une méthode basée sur des retours-arrière chronologiques.

Afin de généraliser les résultats obtenus sur les méthodes à divergences, les travaux que nous avons menés dernièrement se sont intéressés à l'intégration des contributions apportées au sein d'un solveur de contraintes dans l'objectif de concurrencer les méthodes existantes. Ces travaux ont permis de mettre en évidence un nouveau mode de comptage des divergences s'appuyant sur l'arbre de recherche développé par le solveur. Ce mode de comptage s'avère intéressant pour la résolution de problèmes d'optimisation combinatoire. Ces travaux d'intégration ont permis de synthétiser les méthodes de recherche à divergences et les résultats préliminaires obtenus, s'ils ne sont pas meilleurs que ceux fournis par les méthodes à retours-arrière chronologiques existantes fournissent des pistes de recherche intéressantes notamment sur les modes de comptage, sur le développement de méthodes à divergences non-redondantes pour l'optimisation ou sur l'intégration d'heuristiques à pondération de variables.

Chapitre 2

Propagation de contraintes pour l'ordonnancement disjonctif

2.1 Contexte

Un problème d'ordonnancement consiste en la réalisation d'un ensemble de tâches sur un ensemble de ressources, il peut être caractérisé plus finement en fonction de la nature des tâches, de la nature des ressources ainsi que de la nature des contraintes et des objectifs pris en compte.
Nous avons considéré des problèmes d'ordonnancement dans lesquels les tâches ne sont pas interruptibles et dans lesquels les ressources sont renouvelables, c'est-à-dire qu'elles redeviennent totalement disponibles après la réalisation d'une tâche. Les problèmes d'ordonnancement dits d'atelier sont des problèmes particuliers à ressources disjonctives dans lesquels les tâches correspondent à l'ensemble des opérations de travaux (ou *jobs*) à réaliser. Le séquencement de réalisation des opérations des différents jobs, ou gammes de fabrication, est linéaire. Lorsque l'ensemble des jobs dispose d'une gamme unique on parle de problèmes de *flowshop*, lorsque cette gamme diffère en fonction des jobs on parle de problèmes de *jobshop*.

Les contraintes communes aux problèmes d'ordonnancement que nous avons étudiés consistent d'une part en des contraintes temporelles portant sur l'ensemble des tâches :

- contraintes sur le positionnement relatif des tâches entre elles,
- contraintes sur les dates de début et/ou de fin de réalisation des tâches,
- contraintes sur les durées de réalisation des tâches qui peuvent être fixes ou comprises dans un intervalle de valeurs.

D'autre part les contraintes liées à l'utilisation de ressources que nous avons prises en compte sont les suivantes :

- les tâches sont mono-ressources, c'est-à-dire qu'elles ne nécessitent qu'une seule ressource pour leur réalisation,
- les ressources sont disjonctives, elles ne peuvent réaliser qu'une seule tâche à la fois,
- l'affectation des ressources aux tâches peut être connue ou non.

Pour la résolution de tels problèmes, nous nous sommes intéressés soit à l'objectif de minimisation de la durée totale de l'ordonnancement, ou *makespan*, soit à la variante décisionnelle de ce problème lorsque nous les avons abordés d'un point de vue uniquement satisfaction de contraintes.

Depuis plus de trente ans, de nombreux travaux ont traité de la représentation et de l'exploitation des contraintes en ordonnancement et ont montré leur efficacité dans la réso-

lution de nombreux problèmes d'ordonnancement NP-difficiles (ordonnancement d'atelier comme le jobshop, ordonnancement de projet avec contraintes de ressources, ...).

Les approches par contraintes de problèmes d'ordonnancement développent des modèles de représentation et des méthodes de traitement des contraintes issus de l'**intelligence artificielle** et plus particulièrement des problèmes de satisfaction de contraintes ou CSP (modélisation par contraintes, propagation de contraintes, stratégies de parcours dans des méthodes de recherche arborescente) le plus souvent en combinaison avec des méthodes issues de la **recherche opérationnelle** (théorie des graphes, programmation mathématique, programmation linéaire en nombres entiers, procédures par séparation et évaluation, etc.).
L'objectif est de mettre au point des outils facilitant l'interaction entre les modèles et les décideurs, en intégrant des méthodes d'analyse utiles dans un contexte d'aide à la décision ainsi que des algorithmes efficaces de résolution.

2.2 Méthodes de résolution

Dans le domaine des CSP, la méthode de recherche la plus usuelle est la méthode en profondeur d'abord avec retours-arrière chronologiques que nous avons présentée au chapitre 1. Cette méthode est une méthode de recherche dite complète (ou systématique) basée sur une énumération de l'ensemble des solutions. Elle instancie progressivement les variables d'un problème sur la base d'une heuristique d'instanciation.
Afin de limiter de futures infaisabilités lors de la résolution, cette méthode est couplée avec des mécanismes de propagation de contraintes telle que *Forward Checking*, limitant les propagations de type arc-cohérence aux variables voisines de celles instanciées ou *Maintening-Arc-Consistency* (MAC), appliquant l'arc-cohérence à l'ensemble des variables non encore instanciées.

Pour résoudre un problème d'optimisation combinatoire, il est possible d'exploiter les méthodes exposées dans le chapitre 1 en considérant la résolution successive d'un ensemble de CSP dans lesquels la fonction objectif sera contrainte par une valeur donnée (obtenue lors des itérations précédentes).
On peut alors utiliser une méthode de recherche dichotomique fixant à chaque étape une limite sur la valeur de la fonction objectif (voir algorithme 7). Si l'on considère un problème de minimisation, la recherche dichotomique débute avec une borne inférieure (LB) et une borne supérieure (UB) de l'objectif et se positionne au milieu de cet intervalle pour fixer une valeur limite de la fonction objectif, notée L. La méthode tente de résoudre le problème de l'existence d'une solution dont le cout est inférieur ou égal à L. Si c'est le cas, la recherche redémarre avec comme nouvelle valeur de borne supérieure celle de la solution trouvée sinon la borne inférieure est actualisé à $L + 1$. La méthode stoppe lorsque la borne inférieure et la borne supérieure sont égales.
Des méthodes de recherche linéaires augmentant ou diminuant progressivement la valeur de L sont également utilisées.

On peut également chercher une évaluation par défaut d'une borne inférieure d'une fonction objectif (dans le cas d'un problème de minimisation). Dans ce cas, on considère généralement un sous-ensemble du problème initial et on cherche s'il existe une solution à ce problème moins contraint pour une valeur L donnée de la fonction objectif. Si le sous-problème n'a pas de solution, alors le problème global n'en a pas non plus et la valeur de L peut donc être augmentée. Si le sous-problème a une solution, la valeur de L peut être diminuée. On obtient ainsi la plus grande valeur pour laquelle l'infaisabilité du problème

Algorithme 7 Recherche dichotomique

Entrées: X, D, C, f {X : ensemble de variables, D : ensemble de domaines, C : ensemble de contraintes, f :fonction objectif à minimiser}
Sorties: Sol, f^*

```
1: UB ← Borne_Sup(X, D, C, f) {Calculer une borne supérieure}
2: LB ← Borne_Inf(X, D, C, f) {Calculer une borne inférieure}
3: repeat
4:    L ← (LB + UB)/2
5:    Sol ← Trouver_Solution(X, D, C, f, L)
6:    if Est_Solution(Sol) then
7:       f* ← f(Sol)
8:       UB ← f*
9:    else
10:      LB ← L + 1
11:   end if
12: until LB = UB
```

n'a pas été prouvée.
Ces méthodes faisant évoluer progressivement les valeurs de bornes inférieures et/ou supérieures de la fonction objectif sont qualifiées de méthodes de calculs de *bornes destructives* [100].

Une autre variante pour la résolution de problèmes d'optimisation consiste à modifier la méthode de backtrack chronologique en lui associant deux fonctions d'évaluation pouvant être appliquées à chaque nœud (ie. sur des solutions partielles) et donnant des estimations de la borne inférieure et de la borne supérieure. Un échec survient lorsqu'à un nœud donné l'estimation de la borne supérieure est inférieure à l'estimation de la borne inférieure et un retour-arrière pourra alors être effectué. On parle également de méthode par séparation et évaluation ou *branch-and-bound*.

2.3 Les problèmes de satisfaction de contraintes temporelles

Nous avons donné dans le chapitre 1, les notions élémentaires sur les problèmes de satisfaction de contraintes ou CSP (variables, domaines, contraintes, instanciation, solution, propagation). Nous présentons dans cette partie, des CSP dédiés à la modélisation et à la résolution de problèmes temporels, appelés **TCSP** pour *Temporal Constraint Satisfaction Problems*, que nous avons plus particulièrement étudiés pour la résolution de problèmes d'ordonnancement.

Les TCSP permettent de représenter les contraintes temporelles d'un problème, deux types de modèles ont été proposés dans la littérature : des modèles basés sur des contraintes temporelles numériques ou quantitatives introduits par R. Dechter [43] et des modèles basés sur des contraintes qualitatives utilisant notamment l'algèbre d'intervalles de J.F. Allen [1] ou l'algèbre de points de M. Vilain et H. Kautz [150]. Des travaux intégrant les deux types de modélisation ont également été réalisés notamment dans [115].
Nous nous sommes intéressés aux TCSP manipulant des contraintes numériques définis par :

- un ensemble $X = \{x_1, ..., x_n\}$ de variables ;
- un ensemble de domaines correspondant chacun à un ensemble d'intervalles de valeurs

possibles. Si D_i est le domaine de la variable x_i alors $D_i = \{I_i^1 = [a_i^1, b_i^1], ..., I_i^n = [a_i^{n_i}, b_i^{n_i}]\}$. Ainsi pour une variable x_i on a un ensemble non conjonctif de contraintes unaires : $(a_i^1 \leq x_i \leq b_i^1) \vee .... \vee (a_i^{n_i} \leq x_i \leq b_i^{n_i})$;

- un ensemble de relations binaires contraignant la distance entre deux variables. Soient x_i et x_j deux variables, la contrainte binaire C_{ij} les reliant limite les valeurs possibles pour la distance entre ces deux variables à un ensemble d'intervalles de valeurs $\{I_{ij}^1, ..., I_{ij}^{n_{ij}}\}$. On a donc pour chaque relation binaire un ensemble non conjonctif de contraintes de la forme : $(a_{ij}^1 \leq x_j - x_i \leq b_{ij}^1) \vee .... \vee (a_{ij}^{n_{ij}} \leq x_j - x_i \leq b_{ij}^{n_{ij}})$.

L'ajout d'une variable fictive x_0 représentant l'origine temporelle permet d'écrire les équations des domaines sous forme de contraintes binaires.

Un TCSP se modélise par un graphe de contraintes, dans lequel les sommets représentent l'ensemble des variables $X \cup \{x_0\}$ et les arcs les contraintes binaires entre ces variables. Ils sont valués par l'ensemble des intervalles correspondant à chaque contrainte. La figure 2.1 présente un exemple de TCSP comportant 5 variables.

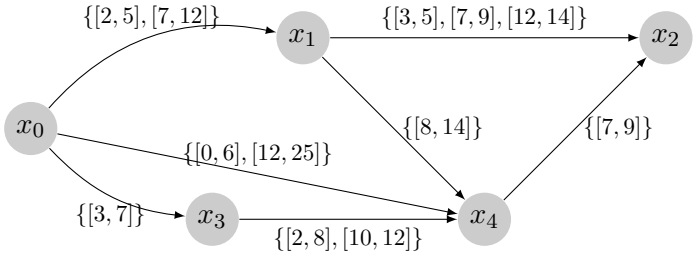


FIGURE 2.1 – Exemple de TCSP

Un cas particulier de TCSP, appelé **STP** (pour *Simple Temporal Problem* [43]), correspond au cas où chaque contrainte ne comporte qu'un seul intervalle. Ainsi le domaine d'une variable x_i est limité par un intervalle $[a_i, b_i]$ et la distance entre deux variables x_i et x_j est limitée par un seul intervalle $[a_{ij}, b_{ij}]$.
Le problème se représente alors par un graphe orienté valué dont les sommets correspondent aux variables du STP et les arcs représentent des relations de précédence entre deux variables et sont valués par la valeur minimale de cette précédence. Ainsi toute contrainte binaire $(a_{ij} \leq x_j - x_i \leq b_{ij})$ est représentée par deux arcs : un arc de x_i vers x_j valué par a_{ij} correspondant à la contrainte $x_j - x_i \geq a_{ij}$ et un arc de x_j vers x_i valué par $-b_{ij}$ correspondant à la contrainte $x_i - x_j \geq -b_{ij}$.

Un TCSP (ou un STP) est dit minimal lorsque les intervalles associés aux variables et aux contraintes binaires ne comportent aucune valeur inconsistante. L'obtention d'un TCSP minimal est un problème NP-difficile.
En revanche, l'obtention d'un STP minimal est un problème polynomial et revient à déterminer les plus longs chemins sur un tel graphe de tout sommet vers tout sommet. On peut l'effectuer en utilisant un algorithme de type Floyd et Warshall.
On peut également noter que l'arc-consistance d'un STP revient à appliquer un algorithme de plus long chemin de l'origine (ici x_0) vers tous les autres sommets du graphe, elle peut être réalisée à l'aide d'un algorithme de type Bellman-Ford. Si elle ne garantit pas l'obtention d'un STP minimal, elle permet cependant d'obtenir le domaine minimal pour les variables du problème.
Pour le cas général des TCSP, des algorithmes de consistance locale permettent de réduire le problème initial sans toutefois garantir la complétude du processus et donc l'obtention d'un TCSP minimal. L'algorithme de chemin-consistance (PC) permet d'actualiser

les intervalles de valeurs de toutes contraintes binaires. Il s'appuie pour cela sur deux opérations définies sur des ensembles d'intervelles. Soient $\{I_1, ..., I_p\}$ et $\{J_1, ..., J_q\}$ deux ensembles d'intervalles on définit :

- la composition $\{I_1, ..., I_p\} \otimes \{J_1, ..., J_q\} = \{K_1, ..., K_s\}$ où si $I_i = [a, b]$ et si $J_j = [c, d]$, $K_k = I_i \otimes J_j = [a+c, b+d]$ (on a $s \leq p.q$). Par exemple, la composition des contraintes C_{03} et C_{34} de l'exemple de la figure 2.1 donne $\{[3, 7]\} \otimes \{[2, 8], [10, 12]\} = \{[5, 19]\}$.
- l'intersection $\{I_1, ..., I_p\} \oplus \{J_1, ..., J_q\} = \{K_1, ..., K_r\}$ où $K_k = I_i \cap J_j$ pour tout i de 1 à p et pour tout j de 1 à q (on a $r \leq p + q$). Par exemple, l'intersection de la contrainte C_{04} de l'exemple avec la composition obtenue précédemment donne $\{[0, 6], [12, 25]\} \oplus \{[5, 19]\} = \{[5, 6], [12, 19]\}$.

Sur la base de ces deux opérations, l'ensemble d'intervalles associé à une contrainte C_{ij} est actualisé via toute autre variable x_k par la formule $C_{ij} \oplus (C_{ik} \otimes C_{kj})$ (dans le cas des STP, cette formule revient à l'algorithme de Floyd et Warshall pour un seul ensemble d'intervalles). Cependant, cet algorithme de chemin-consistance conduit potentiellement à une fragmentation des intervalles de valeurs. Des variantes ont été proposées afin de limiter cette fragmentation [139].
La première variante, appelée ULT, se ramène à un STP en considérant la borne inférieure et la borne supérieure des ensembles d'intervalles avant d'appliquer un algorithme de chemin-consistance. Une fois les propagations effectuées, les intervalles résultants sont intersectés avec les intervalles fragmentés initiaux. Le processus est ré-itéré jusqu'à ce que les contraintes n'évoluent plus. Une deuxième variante appelée LPC s'appuie sur une nouvelle opération d'intersection n'autorisant que des variations des bornes minimales et maximales des différents intervalles mais ne permettant pas la création de nouveaux intervalles. les algorithmes PC, ULT et LPC déterminent les mêmes bornes minimales et maximales pour chaque contrainte.
La différence entre ces algorithmes est dans la gestion des « trous »apparaissant dans les intervalles associés aux contraintes. L'algorithme PC provoque une augmentation du nombre de trous dans les domaines des contraintes, l'algorithme ULT conserve les trous présents initialement dans les contraintes, alors que l'algorithme LPC les agrandit.

Dans le courant des années 2000, des travaux se sont intéressés à une extension des TCSP appelée *Disjunctive Temporal Problem* ou *DTP* [3, 126, 145, 147]. Dans un DTP, les contraintes considérées sont appelées contraintes disjonctives et sont composées d'un ensemble d'inégalités de la forme : $x_{j_1} - x_{i_1} \leq v_1 \vee x_{j_2} - x_{i_2} \leq v_2 \vee ..... \vee x_{j_k} - x_{i_k} \leq v_k$. Les DTP permettent ainsi de pallier aux limites des TCSP qui ne permettaient pas l'expression de contraintes disjonctives sur plus de 2 variables.
Une première approche pour le traitement de ce type de problème temporel consiste en la résolution d'un problème de satisfiabilité [3, 4] (ou SAT) dans lequel les clauses correspondent aux contraintes et les littéraux aux différentes inégalités. La méthode de résolution proposée s'appuie sur des heuristiques d'instanciation dédiées (choix de la clause de plus petite taille et ayant le littéral impliqué dans le plus grand nombre de conflits). Une deuxième approche [126, 145, 147] s'appuie quant à elle sur une modélisation d'un méta-CSP dans lequel on associe une variable à chaque contrainte disjonctive (la valeur pouvant être prise par cette variable correspond alors à une des inégalités). Les heuristiques d'instanciation sont du type sélection de la variable de plus petit domaine en la couplant éventuellement avec des informations plus élaborées recueillies lors des étapes précédentes de résolution. Une instanciation d'un DTP correspond alors à un STP dont la consistance peut être prouvée en temps polynomial.
Les meilleurs algorithmes basés sur l'approche méta-CSP exploitent une combinaison de différentes techniques permettant d'élaguer la recherche arborescente (retours-arrières intelligents et apprentissage) [147].

L'approche SAT incluent également des techniques d'accélération de type retours-arrières intelligents et apprentissage ainsi que des techniques spécifiques des problèmes SAT. Elle s'est avérée la plus performante pour la résolution de DTP aléatoires et en particulier pour des problèmes difficiles [4].
Appliqué à la résolution de problèmes d'ordonnancement disjonctifs (jobshop) modélisés sous la forme de DTP, ces méthodes se sont avérées moins performantes que des algorithmes dédiés à la résolution de ces problèmes. Les raisons données sont liées d'une part à la généricité de l'approche DTP comparée à des méthodes dédiées à la résolution de jobshop, et d'autre part, sont liées à une différence de nature dans les problèmes considérés : il est relativement « simple » de trouver une solution pour un problème de jobshop mais il est difficile de trouver une solution optimale alors que pour les DTP l'obtention même d'une solution est difficile.

2.4 Propagation de contraintes pour l'ordonnancement

2.4.1 Modélisation par contraintes de problèmes d'ordonnancement

Soit $\mathcal{T} = [1, .., n]$ un ensemble de tâches à réaliser sur un ensemble $\mathcal{R} = [1, .., m]$ de ressources, nous exposons par la suite une modélisation générale des problèmes d'ordonnancement dans lesquels les tâches sont non interruptibles et les ressources renouvelables. Les modèles par contraintes de ces problèmes d'ordonnancement associent à chaque tâche i, deux variables st_i, ft_i représentant respectivement sa date de début (start time), sa date de fin (finish time). Par ailleurs, on peut limiter la réalisation de toute tâche i à une fenêtre temporelle $[r_i, d_i]$ où r_i représente la date de démarrage au plus tôt (release date) et d_i représente la date de fin au plus tard (due date). Ainsi, le domaine des variables st_i et ft_i est limité par $[r_i, lst_i]$ et $[eft_i, d_i]$ où lst_i est la date de début au plus tard (latest start time) et eft_i la date de fin au plus tôt (earliest finish time). La durée de réalisation notée pt_i (processing time) d'une tâche i correspond au temps s'écoulant entre sa date de début et de fin : $pt_i = ft_i - st_i$, elle est comprise dans l'intervalle $[\underline{pt}_i, \overline{pt}_i]$.
Lorsque la durée de réalisation des tâches est connue et fixe, une seule variable par tâche suffit, par exemple st_i, et on a par définition $ft_i = st_i + pt_i$.
Lorsque l'affectation des tâches aux ressources n'est pas connue, il est nécessaire d'associer à chaque tâche une variable supplémentaire res_i représentant la ressource qui lui est allouée. On note $\mathcal{R}_i \in \mathcal{R}$ l'ensemble des ressources possibles pour la tâche i. La durée des tâches peut être indépendante de la ressource allouée ou dépendre du choix d'affectation.

Les contraintes temporelles d'un problème d'ordonnancement sont tout d'abord issues des limites posées par les fenêtres de réalisation des tâches et qui restreignent le domaine de chaque variable. De plus, les tâches peuvent être reliées entre elles par différentes relations temporelles comme la succession, le chevauchement, l'inclusion, etc. Ces contraintes s'expriment dans le cas général par un ensemble d'inégalités linéaires aussi appelées inégalités de potentiels généralisées : $x_i - x_j \geq c_{ij}$ où c_{ij} est un entier représentant la valeur de la contrainte et où x_i et x_j appartiennent à $\{st_k, ft_k | k \in \mathcal{T}\}$.
La relation la plus simple est celle de succession stricte que l'on retrouve, par exemple, dans les problèmes d'ordonnancement d'atelier dans lesquels les opérations d'un même job doivent s'enchaîner les unes après les autres. Par exemple, la succession entre une tâche i et une tâche j, notée $i \prec j$ correspond à l'inégalité $st_j - ft_i \geq 0$.

Les contraintes de partage de ressources traduisent le fait que les ressources sont disponibles en quantité limitée constante notée Q_k ($\forall k \in \mathcal{R}$). On considère que les tâches nécessitent une quantité constante de ressource q_i^k ($\forall k \in \mathcal{R}, \forall i \in \mathcal{T}$).

Dans le cas de ressources disjonctives, $Q_k = 1$ et les tâches i devant être exécutée sur k nécessitent une quantité unitaire de ressource : $q_i^k = 1$. Les contraintes de partage de ressource s'expriment alors par une disjonction de contraintes de précédence entre toute paire de tâches requérant une même ressource disjonctive. De manière plus formelle, pour toute ressource disjonctive k et pour toute paire de tâches nécessitant cette ressource k on a la contrainte $(i \prec j) \vee (j \prec i)$. Une telle paire de tâches en conflit est appelée *paire de disjonction.*
Dans le cas de ressources cumulatives, il peut s'avérer nécessaire de caractériser des ensembles de tâches en conflit appelés également ensembles critiques de tâches. Soit une ressource k, et soit $T_k \subseteq \mathcal{T}$ l'ensemble des tâches utilisant k, $E_k \subseteq T_k$ est un ensemble critique de tâches si $\sum_{i \in E_k} q_i^k > Q_k$ et $\forall j \in E_k, \sum_{i \in E_k \setminus \{j\}} q_i^k \leq Q_k$. Au sein d'un ensemble critique, le séquencement d'une des tâches par rapport aux autres suffit à lever le conflit de partage de ressources.
Pour les contraintes de partage de ressources, on peut également caractériser les ensembles disjonctifs maximum associés à chaque ressource. Un tel ensemble correspond au plus grand sous-ensemble de tâches tel que chaque paire est une paire de disjonction. Chaque ensemble disjonctif maximal correspond alors à un problème d'ordonnancement à une machine.
Pour les problèmes mono-ressource, lorsque les ressources sont disjonctives, chaque ensemble T_k est un sous ensemble disjonctif maximal ; en revanche lorsque les ressources sont cumulatives, chaque T_k peut contenir plusieurs ensembles disjonctifs maximaux.

2.4.2 Graphes utilisés pour la représentation de problèmes d'ordonnancement

Les problèmes d'ordonnancement sont classiquement représentés par des graphes orientés valués appelés potentiels-tâches, dans lesquels chaque sommet correspond à une tâche. Cette modélisation suppose qu'il n'y a qu'une seule variable pour chaque tâche i, par exemple sa date de début st_i et que la durée de réalisation pt_i est fixée.
Une contrainte de succession entre deux tâches i et j, $i \prec j$, est représentée par la contrainte $st_j - st_i \geq pt_i$. Les arcs d'un graphe potentiels-tâches sont composés de deux ensembles d'arcs.

- Tout d'abord, le graphe se compose d'un ensemble conjonctif d'arcs représentant l'ensemble des contraintes temporelles (début au plus tôt, fin au plus tard, succession, etc.). Des sommets représentant les tâches fictives de début, noté S et de fin, noté F de l'ordonnancement sont ajoutés à ce graphe et sont connectés respectivement aux tâches sans prédécesseur et aux tâches sans successeur.
- Pour la représentation de contraintes de ressources disjonctives, un ensemble d'arcs dits disjonctifs sont ajoutés à ce graphe. Il permettent de représenter les paires de tâches en conflit pour l'utilisation d'une ressource.

Considérons un problème d'ordonnancement avec 3 tâches, a_1, a_2 et a_3, de durées respectives pt_1, pt_2 et pt_3 et telles que a_1 se termine au plus tard à d_1, a_3 débute au plus tôt à r_3, a_1 précède a_2 et a_1 et a_3 nécessitent la même ressource.
La figure 2.2 représente le graphe potentiels-tâches associés à ce problème dans lequel la paire de disjonction $\{a_1, a_3\}$ est modélisée par les arcs disjonctifs en pointillés.

Résoudre un problème d'ordonnancement modélisé par ce type de graphe nécessite de sélectionner une orientation pour chacun de ces arcs disjonctifs de telle sorte que le graphe correspondant ne comporte pas de circuit de longueur positive. La longueur du plus long chemin de S vers F donne la valeur du makespan de la solution obtenue.

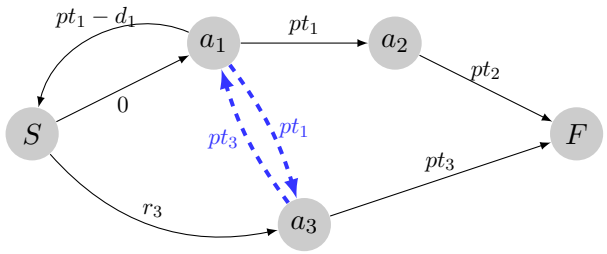


FIGURE 2.2 – Exemple de graphe potentiels-tâches

Afin de pouvoir modéliser des problèmes d'ordonnancement plus généraux tels que ceux modélisés par des inégalités de potentiels généralisées, nous avons proposé dans [55], d'utiliser un graphe appelé potentiels-bornes. Sur ce type de graphe, les sommets correspondent soit aux dates de début, soit aux dates de fin des tâches et les arcs conjonctifs représentent la distance temporelle minimale séparant deux de ces variables temporelles. Avec un graphe potentiels-bornes, il est par exemple possible de représenter les tâches dont la durée est comprise dans un intervalle donné par deux arcs allant du sommet de début vers le sommet fin valué par la durée minimale et par un arc en sens inverse valué par l'opposé de la durée maximale. Sur ce type de graphe, les arcs disjonctifs représentant les paires de tâches en conflit relient 4 sommets différents (fin de i et début de j ou fin de j et début de i pour la paire de tâches en conflit $\{i, j\}$).

Considérons le même exemple de problème d'ordonnancement à 3 tâches que précédemment mais pour lequel les durées respectives des tâches sont comprises dans les intervalles $[\underline{pt}_1, \overline{pt}_1]$, $[\underline{pt}_2, \overline{pt}_2]$ et $[\underline{pt}_3, \overline{pt}_3]$. Le graphe potentiels-bornes représentant ce problème est donné à la figure 2.3. Pour chaque tâche a_i, il y a deux variables st_i et ft_i. La paire de disjonction entre les tâches a_1 et a_3 est modélisée par les arcs disjonctifs en pointillés.

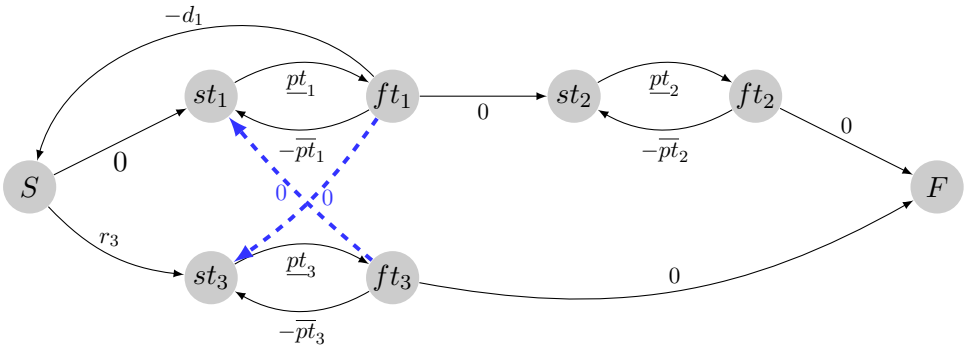


FIGURE 2.3 – Exemple de graphe potentiels-bornes

Pour résoudre un problème d'ordonnancement avec cette modélisation, il faut sélectionner un seul arc dans chaque paire de disjonction sans créer de cycle de longueur positive. Lorsque cette sélection est admissible, la longueur du plus long chemin de S à F donne la valeur du makespan.

Synthèse des notations

· $\mathcal{T} = [1, .., n]$: ensemble de tâches

- $\mathcal{R} = [1, .., m]$: ensemble de ressources
- st_i : date de début d'une tâche i, $st_i \in [r_i, lst_i]$
- ft_i : date de fin d'une tâche i, $ft_i \in [eft_i, d_i]$
- pt_i : durée d'une tâche i, $pt_i \in [\underline{pt}_i, \overline{pt}_i]$
- $\mathcal{R}_i \in \mathcal{R}$: ensemble des ressources possibles pour la tâche i
- res_i : ressource allouée à une tâche i, $res_i \in \mathcal{R}_i$
- T_k : ensemble des tâches utilisant la ressource k
- Q_k : quantité de ressource k disponible
- q_i^k : quantité de ressource k nécessaire à la réalisation d'une tâche i
- soit Ω un ensemble de tâches, $d_\Omega = \max_{i\in\Omega}(d_i)$, $r_\Omega = \min_{i\in\Omega}(r_i)$ et $\underline{pt}_\Omega = \sum_{i\in\Omega}(\underline{pt}_i)$

2.4.3 Propagation de contraintes temporelles

Les contraintes temporelles d'un problème d'ordonnancement correspondent (le plus souvent) à un ensemble conjonctif d'inégalités de potentiels de la forme $x_i - x_j \geq c_{ij}$ et le problème à résoudre s'apparente à un STP.
Les algorithmes dérivés de celui de Bellman-Ford dans sa version standard ou dans une version incrémentale telle que celle proposée par [38] permettent d'assurer l'arc-consistance aux bornes, c'est-à-dire qu'ils actualisent les valeurs minimales et maximales des dates de debut et de fin de chaque tâche sans chercher à vérifier la consistance pour chaque valeur de l'intervalle.
Dans d'autres articles [30, 103, 49] ainsi que dans les travaux que nous avons réalisés (par exemple [78, 84, 5]), les auteurs cherchent à obtenir la chemin-consistance aux bornes et exploitent pour cela un algorithme de type Floyd et Warshall dans une version standard ou incrémentale [117, 147, 131]. En utilisant ce type de propagation, en plus des actualisations sur les domaines des variables, de nouvelles contraintes de précédence sont déduites. Ces algorithmes s'appuient sur une matrice de distance $A = (a_{ij})$ (où a_{ij} correspond à la valeur de la contrainte entre les variables x_i et x_j : $x_j - x_i \geq a_{ij}$) et assurent la complétude des déductions sur l'ensemble conjonctif d'inégalités de potentiels. Dans cette matrice, les valeurs a_{x_0,x_i} donnent les valeurs au plus tôt des variables x_i, les valeurs a_{x_i,x_0} correspondent quant à elles aux valeurs au plus tard de ces variables : $x_i - x_0 \in [a_{x_0,x_i}, a_{x_i,x_0}]$. Pour n tâches, si la modélisation nécessite 2 variables par tâches, la matrice contient $n + 1$ lignes et colonnes (avec la variable origine).

2.4.4 Propagation de contraintes de ressources

Les propagations de contraintes de ressources que nous allons présenter par la suite permettent de déduire des ordres partiels entre tâches ou ensembles de tâches. Dans ce paragraphe, nous allons dresser un état de l'art (non exhaustif) des propagations les plus usuelles dans le cas de ressources disjonctives (même si certaines d'entre elles sont généralisées pour des ressources cumulatives) qui permettent de déduire des ordres partiels du type :

- précédences interdites entre tâches, notées $i \not\prec j$.
- tâches non premières ou non dernières parmi un ensemble de tâches, notées $i \not\prec \Omega$ ou $\Omega \not\prec i$ avec Ω un ensemble de tâches. Ces propagations sont appelées *not first/not last* dans la littérature [31, 33, 34, 125, 36, 6, 108].
- tâches premières ou dernières dans un ensemble de tâches, notées $i \prec \Omega$ ou $\Omega \prec i$ avec Ω un ensemble de tâches. Les propagations permettant d'obtenir ce type de déduction sont appelées *edge finding* ou *sélection immédiate* dans la littérature [33, 2, 34, 125, 29, 35].

Une fois ces ordres partiels obtenus, les ajustements sur les fenêtres temporelles ci-dessous peuvent être effectués :

$$\text{si } j \prec i \text{ alors } r_i \leftarrow \max(r_i, eft_j) \text{ et } d_j \leftarrow \min(d_j, lst_i) \tag{2.1}$$

$$\text{si } i \not\prec \Omega \text{ alors } r_i \leftarrow \max(r_i, \min_{j\in\Omega}(eft_j)) \tag{2.2}$$

$$\text{si } \Omega \not\prec i \text{ alors } d_i \leftarrow \min(d_i, \max_{j\in\Omega}(lst_j)) \tag{2.3}$$

$$\text{si } i \prec \Omega \text{ alors } d_i \leftarrow \min(d_i, \min_{\Omega'\subseteq\Omega}(d_{\Omega'} - \underline{pt}_{\Omega'})) \tag{2.4}$$

$$\text{si } \Omega \prec i \text{ alors } r_i \leftarrow \max(r_i, \max_{\Omega'\subseteq\Omega}(r_{\Omega'} + \underline{pt}_{\Omega'})). \tag{2.5}$$

Ces ajustements peuvent permettre par la suite de nouvelles déductions par propagation des contraintes temporelles. Le processus est alors ré-itéré jusqu'à ce qu'il n'y ait plus de modification dans l'ensemble des contraintes.
Lorsque l'on dispose d'une matrice de distance entre les variables temporelles du problème d'ordonnancement, les déductions de séquencements obligatoires (entre deux tâches ou entre une tâche et un ensemble) peuvent être directement intégrées. Les actualisations de fenêtres temporelles sont ensuite obtenues par propagation de contraintes temporelles.

Précédences interdites

La règle de déduction élémentaire de précédences interdites considère les paires de tâches en conflit pour l'utilisation d'une ressource (par exemple la paire $\{i, j\}$ provenant soit d'un ensemble critique soit d'un ensemble disjonctif maximum) et étudie les largeurs des fenêtres de temps disponibles en positionnant soit i avant j soit le contraire par rapport aux durées nécessaires de réalisation des deux tâches. Pour une ressource k et pour une paire de tâches en conflit $\{i, j\}$, sa formulation est donnée par la condition (2.6).

$$\text{si } d_j - r_i < \underline{pt}_i + \underline{pt}_j \text{ alors } i \not\prec j \text{ et donc } j \prec i \tag{2.6}$$

Une généralisation [30, 84, 103, 49] de cette condition de précédence interdite, se base sur la matrice de distance maintenue par un algorithme de chemin-consistance aux bornes. Dans ce cas, un test sur la valeur déduite dans la matrice de distance entre les variables st_j et ft_i permet de déduire une précédence interdite comme spécifié dans la condition (2.7).

$$\text{si } c_{st_j, ft_i} > 0 \text{ alors } i \not\prec j \text{ et donc } j \prec i \tag{2.7}$$

Il a été montré que cette formulation domine la formulation de l'équation (2.6).

En se basant sur le raisonnement énergétique [54, 107], une autre règle de précédence interdite peut être définie dans le cas de problème disjonctif. Cette règle de précédence interdite se base sur la consommation minimale de tâches sur un intervalle de temps donné $\Delta = [r_\Delta, d_\Delta]$, notée $\underline{w_i^\Delta}$ pour une tâche i. Le calcul de cette consommation minimale résulte des différentes positions relatives que peut prendre la tâche i ramenée à sa durée minimale par rapport à Δ : inclusion de i dans Δ, inclusion de Δ dans i, durée de i débordant sur Δ lorsque i est calée au plus tôt, durée de i débordant sur Δ lorsque i est calée au plus tard. D'où la formule :

$$\underline{w_i^\Delta} = \max[0, \min(\underline{pt}_i, d_\Delta - r_\Delta, eft_i - r_\Delta, d_\Delta - lst_i)]. \tag{2.8}$$

Pour une ressource k et pour toute paire de tâches en conflit $\{i, j\}$, si on considère l'intervalle $\Delta = [r_i, d_j]$, la condition (2.9) donne la formulation de la règle de précédence interdite en utilisant les consommations minimales des tâches autres que i et j s'exécutant sur la ressource k.

$$\text{si } d_j - r_i < \underline{pt}_i + \underline{pt}_j + \sum_{t \in T_k \setminus \{i,j\}} \underline{w}_t^{[r_i, d_j]} \text{ alors } i \not\prec j \text{ et donc } j \prec i \tag{2.9}$$

Il a été montré [107] que la règle de propagation (2.9) domine la règle (2.6).

Not First / Not Last

La règle de déduction de condition *not first* s'appuie sur les ensembles disjonctifs maximums. Soient E_k un tel ensemble disjonctif maximal pour une ressource k, $\Omega \subset E_k$ et $i \in E_k \setminus \Omega$, la règle *not first*, exprimée par la condition (2.10), teste si le positionnement de i calée au plus tôt peut être suivi du séquencement de l'ensemble des tâches de Ω sans que ceci n'entraîne un dépassement de la plus grande date de fin des tâches de cet ensemble. Si ce n'est pas le cas, la tâche i ne peut être placée avant celles de Ω :

$$\text{si } r_i + \underline{pt}_i + \underline{pt}_\Omega > d_\Omega \text{ alors } i \not\prec \Omega \tag{2.10}$$

De manière symétrique, la règle *not last*, donnée par la condition (2.11), teste si la tâche i peut être calée au plus tard précédée de l'ensemble des tâches de Ω sans que cela n'entraine un dépassement de la plus petite date de début au plus tôt des tâches de cet ensemble :

$$\text{si } d_i - \underline{pt}_i - \underline{pt}_\Omega < r_\Omega \text{ alors } \Omega \not\prec i \tag{2.11}$$

Des implémentations de ces règles de propagation peuvent être trouvées dans [7, 146].

Une dernière condition fournie en (2.12) permet de détecter la **non-insérabilité** [33] d'une tâche i dans un ensemble Ω, notée $i \notin \Omega$. Pour cela, elle teste la possibilité de séquencer l'ensemble des tâches de cet ensemble et de i à l'intérieur de la fenêtre temporelle de Ω.

$$\text{si } d_\Omega - r_\Omega < \underline{pt}_\Omega + \underline{pt}_i \text{ alors } i \notin \Omega \tag{2.12}$$

Ainsi, la tâche i est soit positionnée avant l'ensemble des tâches de Ω soit après. Cette déduction de non-insérabilité, couplée avec celles de *not first/not last* permet d'obtenir des séquencements obligatoires entre i et Ω.

Edge Finding

Comme pour la déduction de précédences interdites, il existe différentes formulations de règles de propagation permettant de déduire des séquencements obligatoires entre une tâche i et un ensemble Ω. Les différentes conditions permettant de détecter ces sélections immédiates ne se recouvrent pas les unes par rapport aux autres.
La formulation ci-dessous est issue des travaux de [33, 34], elle vérifie s'il est possible de séquencer les tâches de Ω et la tâche i dans la fenêtre temporelle délimitée par le début au plus tôt de Ω et la fin au plus tard entre celle de i et celle de Ω dans le cas de la condition (2.13) ou dans la fenêtre temporelle délimitée par la date de fin au plus tard de Ω et le début au plus tôt entre celui de Ω et celui de i pour la condition (2.14).

$$\text{si } r_\Omega + \underline{pt}_\Omega + \underline{pt}_i > \max(d_i, d_\Omega) \text{ alors } i \prec \Omega \tag{2.13}$$

$$\text{si } d_\Omega - \underline{pt}_\Omega - \underline{pt}_i < \min(r_i, r_\Omega) \text{ alors } \Omega \prec i \tag{2.14}$$

Des implémentations efficaces de ces règles de déductions peuvent être trouvées dans [34, 29].
Une formulation générale des règles recouvrant ces déductions de sélections immédiates, celles de *not first/not last* et permettant également d'autres déductions a été proposée dans [52, 28]. Pour cela, à partir de l'ensemble T_k de tâches en conflit pour l'utilisation d'une ressource k, on considère trois sous-ensembles : $\Omega', \Omega'' \subseteq \Omega \subseteq T_k$, la condition générale est alors la suivante :

$$\text{si } \max_{i \in \Omega \backslash \Omega', j \in \Omega \backslash \Omega'', i \neq j} (d_j - r_i) < \underline{pt}_\Omega \text{ alors } \exists i \in \Omega' \text{ première ou } \exists j \in \Omega'' \text{ dernière} \tag{2.15}$$

Le tableau 2.1 explicite les conditions obtenues en fonction des ensembles utilisés.

$\Omega\backslash\Omega'$	$\Omega\backslash\Omega''$	Déduction
$\Omega\backslash\{i\}$	Ω	$i \prec \Omega\backslash\{i\}$
Ω	$\Omega\backslash\{i\}$	$\Omega\backslash\{i\} \prec i$
$\{i\}$	$\Omega\backslash\{i\}$	$i \not\prec \Omega\backslash\{i\}$
$\Omega\backslash\{i\}$	$\{i\}$	$\Omega\backslash\{i\} \not\prec i$
$\Omega\backslash\{i\}$	$\Omega\backslash\{j\}$	$i \prec \Omega\backslash\{i\}$ ou $\Omega\backslash\{j\} \prec j$

TABLE 2.1 – Synthèse des différentes conditions obtenues par la condition (2.15)

Dans [35], d'autres règles de sélection immédiate ont été proposées, elles sont nommées *Latest Starting time of Last* ou LSL et *Earliest Finishing time of First* ou EFF. Ces déductions sont complémentaires des précédentes.
Pour une ressource k, à partir de l'ensemble T_k des tâches nécessitant cette ressource, pour tout ensemble $\Omega \subset T_k$ la règle LSL est donnée dans la condition (2.16) et la règle EFF est donnée par la condition (2.17).

$$\text{si } \max_{j \in \Omega' \subset \Omega} (d_j - \underline{pt}_j) < r_i + \underline{pt}_i \text{ alors } \Omega' \prec i \tag{2.16}$$

$$\text{si } \min_{j \in \Omega' \subset \Omega} (r_j + \underline{pt}_j) < d_i - \underline{pt}_i \text{ alors } i \prec \Omega' \tag{2.17}$$

Ces deux règles de déduction sont des généralisations de la règle de précédence interdite (2.6) à laquelle on peut se ramener lorsque l'ensemble des tâches en conflit avec la tâche i ne comporte qu'un seul élément.

2.5 Contributions

2.5.1 Approche temporelle des problèmes d'ordonnancement et d'affectation

Nous avons présenté dans [77, 80, 78, 82, 81] une modélisation et des méthodes de résolution de problèmes d'ordonnancement flexibles à ressources disjonctives en nous appuyant sur la formulation proposée pour les problèmes de satisfaction de contraintes temporelles. La flexibilité du problème réside dans le fait que la ressource allouée à la réalisation des tâches n'est pas fixée mais doit être déterminée parmi plusieurs ressources possibles, avec une durée de réalisation pour chaque ressource.
Le modèle retenu associe à chaque tâche i, deux variables st_i, ft_i représentant respectivement les dates de début et de fin. Les choix d'affectation pour chaque tâche i sont exprimés par un ensemble de durées possibles ainsi la durée de chaque tâche est comprise

dans un intervalle $[\underline{pt}_i, \overline{pt}_i]$ issu de ces choix d'affectation. Les contraintes de partage de ressources sont exprimées par des disjonctions d'inégalités mettant en jeu quatre variables du problème (début et fin de deux tâches). Par exemple, la contrainte $(i \prec j) \vee (j \prec i)$ s'exprime par la disjonction d'inégalités $(st_j - ft_i \geq 0) \vee (st_i - ft_j \geq 0)$. Cette modélisation peut s'exprimer via un graphe potentiels-bornes présenté en (2.4.2) et se compose d'un ensemble conjonctif de contraintes temporelles et d'un ensemble de disjonction d'inégalités. En raison des disjonctions considérées, le modèle proposé est une généralisation des TCSP et l'approche développée s'apparente à celle postérieure sur les DTP.
Nos travaux ont porté sur la proposition d'un algorithme de propagation de contraintes temporelles génériques permettant de traiter les disjonctions considérées. Cet algorithme est composé de :

- la chemin-consistance aux bornes pour l'ensemble conjonctif de contraintes réalisée par une méthode incrémentale ;
- la simplification de l'ensemble disjonctif d'inégalités en fonction des déductions obtenues sur l'ensemble conjonctif (élimination d'inégalités incohérentes ou élimination de disjonction satisfaite).

L'algorithme proposé est en fait une variante de l'algorithme ULT développé pour les TCSP à un STP avec disjonction.
Des expérimentations ont été menées sur des instances aléatoires, sur des instances de flowshop hybrides de [149, 122] ou de jobshop flexibles de [86]. Comme dans le cas des algorithmes pour les DTP appliqués à la résolution de problèmes d'ordonnancement, les résultats obtenus ne se sont pas avérés concluants par rapport à des méthodes de propagation dédiées pour l'ordonnancement et l'affectation que nous avons proposées et qui sont présentées ci-après.

2.5.2 Extension de propagation de contraintes ressources

Pour la résolution de problèmes d'ordonnancement disjonctifs avec contraintes temporelles généralisées telles que des durées variables (en liaison avec des choix d'affectation ou avec des délais entre tâches) nous avons proposé différentes extensions de certaines règles de propagation de contraintes de ressources. La modélisation retenue comprend un ensemble conjonctif d'inégalités de potentiels ainsi qu'un ensemble de paires de disjonction représentant les contraintes de partage de ressources. Selon les problèmes spécifiques considérés, elle nécessite une ou deux variables de décision par tâche (date de début seulement ou dates de début et de fin).

Chemin-consistance incrémentale

Pour les extensions de règles de propagation proposées, nous avons proposé [79, 5] l'utilisation d'un algorithme incrémental de chemin-consistance aux bornes pour la propagation de contraintes temporelles. Cet algorithme est appelé *Incremental Full Path Consistency* ou IFPC dans la littérature [145, 130] et son principe est donné dans l'algorithme 8. Dans cet algorithme, on suppose que la nouvelle contrainte ajoutée a'_{ij} n'est pas trivialement inconsistante ($a'_i j + a_{ji} \leq 0$) et qu'elle n'est pas couverte par une contrainte existante ($a'_{ij} > a_{ij}$).

Cet algorithme revient à celui présenté récemment par L. Planken [130] ayant une complexité temporelle au pire en $O(n^2)$. Dans cet article, Planken montre, de manière expérimentale, l'efficacité de la chemin-consistance incrémentale pour la résolution de problèmes d'ordonnancement par rapport à d'autres algorithmes de chemin-consistance. Après appli-

Algorithme 8 Chemin-consistance incrémentale

Entrées: A, a'_{ij} {A : matrice de distance représentant l'ensemble de contraintes, a'_{ij} :nouvelle contrainte}
Sorties: C, Consistance ou Inconsistance

```
1:  Add ← {a_ij} {Liste de contraintes à ajouter}
2:  while Add non vide do
3:    for k = 1 à n do
4:      if a_ik < a'_ij + a_jk then
5:        a_ik ← a'_ij + a_jk
6:        if a_ik > 0 then
7:          return Inconsistance
8:        else
9:          Add ← Add + {a_ik}
10:       end if
11:     end if
12:     if a_kj < a_ki + a'_ij then
13:       a_kj ← a_ki + a'_ij
14:       if a_kj > 0 then
15:         return Inconsistance
16:       else
17:         Add ← Add + {a_kj}
18:       end if
19:     end if
20:   end for
21: end while
22: return Consistance
```

cation de cet algorithme complet de chemin-consistance, les distances entre toutes paires de variables sont actualisées à leur plus grande valeur possible.

Nouvelles conditions pour la propagation de contraintes de ressources

La première extension proposée consiste à étendre la condition de précédence interdite (2.6) à la prise en compte des valeurs obtenues dans la matrice de distance. La condition étendue (2.7) a été présentée dans l'état de l'art sur la propagation de contraintes en ordonnancement et se retrouve indépendamment dans différents autres travaux [30, 84, 103, 49]. On peut noter que la complexité de cette condition étendue est la même que celle de la condition classique car elle teste les mêmes ensembles de tâches et l'accès aux valeurs stockées dans la matrice de distances est direct.

Le schéma de la figure 2.4 illustre l'application de cette condition pour deux tâches i et j en disjonction. Si la distance déduite de st_j vers ft_i est strictement positive et si i est avant j il y a un cycle de longueur positive dans le graphe. Cela permet de déduire que que $i \not\prec j$.

En suivant cette même idée, nous avons proposé [79, 5] une extension des conditions (2.16) et (2.17) basée sur les valeurs obtenues dans la matrice de distance après un calcul de chemin-consistance.
La généralisation de la condition LSL (2.16) cherche un minorant de la longueur des chemins de $(st_j)_{j\in\Omega'}$ à ft_i, pour tout $j \in \Omega'$. Si ce minorant est positif, alors la tâche i ne

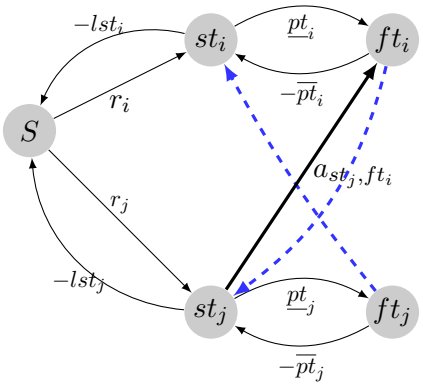


FIGURE 2.4 – Exemple d'application de la condition de précédence interdite étendue

peut précéder l'ensemble des tâches de Ω', d'où la conclusion : $\Omega' \prec i$. Ainsi la nouvelle formulation de la condition LSL s'exprime par la condition (2.18).

$$\text{si } \min_{j \in \Omega' \subset \Omega} (a_{st_j,ft_i}) > 0 \text{ alors } \Omega' \prec i \tag{2.18}$$

Le graphe de la figure 2.5 illustre l'application de la condition LSL étendue entre une tâche i et un ensemble Ω' composé des tâches $j_1, j_2, ..., j_k$. Les arcs testés par cette condition sont ceux en trait épais. Si l'un d'entre eux correspond à une valeur positive alors la tâche correspondante ne peut précéder i, si chacun des arcs à une valeur positive pour les différentes tâches de Ω' alors $\Omega' \prec i$.

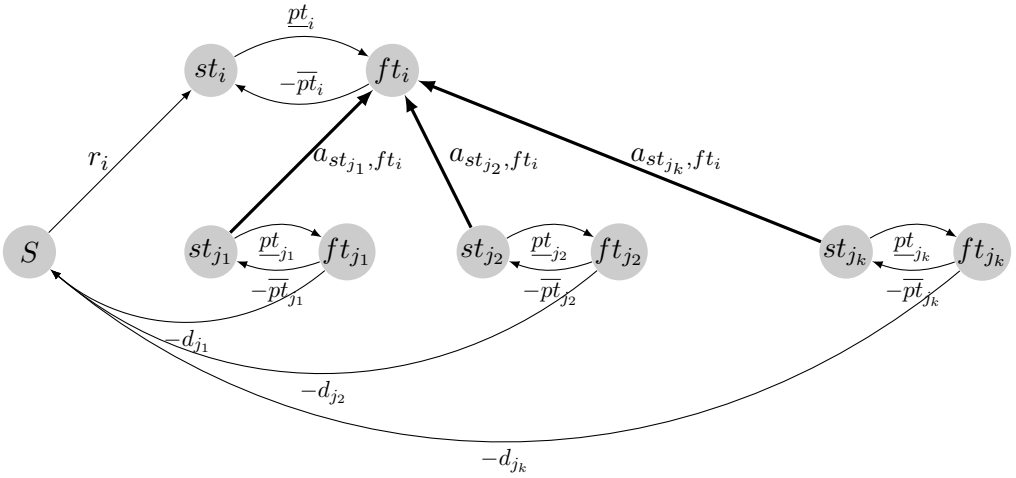


FIGURE 2.5 – Exemple d'application de la condition LSL étendue

La généralisation de la condition EFF (2.17) procède de manière symétrique et est fournie dans la condition (2.19)

$$\text{si } \min_{j \in \Omega' \subset \Omega} (a_{st_i,ft_j}) > 0 \text{ alors } i \prec \Omega' \tag{2.19}$$

Nous avons montré que ces deux extensions dominent les conditions dont elles sont issues [79, 5] et leur complexité est identique à celle des conditions classiques car elles considèrent les mêmes ensembles de tâches.

La dernière extension que nous avons proposée [84] concerne la précédence interdite basée sur un raisonnement énergétique (2.9). Pour cela, nous avons tout d'abord étendu la formulation de la consommation minimale (2.8). Avec cette formulation (2.8), le premier terme est inchangé et représente le cas où la tâche i est incluse dans l'intervalle Δ, le deuxième terme représente la distance entre le début et la fin de l'intervalle, le troisième et le quatrième terme représente le cas où la tâche est calée respectivement à gauche ou à droite de l'intervalle et exprime la distance entre le début (resp. fin) de l'intervalle et la fin (resp. début) de la tâche.

$$\underline{we}_i^{\Delta} = \max[0, \min(\underline{pt}_i, a_{st_\Delta, ft_\Delta}, a_{st_\Delta, ft_i}, a_{st_i, ft_\Delta})]. \tag{2.20}$$

Le calcul de la consommation minimale d'une tâche i sur un intervalle $\Delta = [t_1, t_2]$ est illustré sur la figure 2.6. Les arcs en trait épais représentent les quatre valeurs à tester pour les différentes positions de i sur l'intervalle considéré.

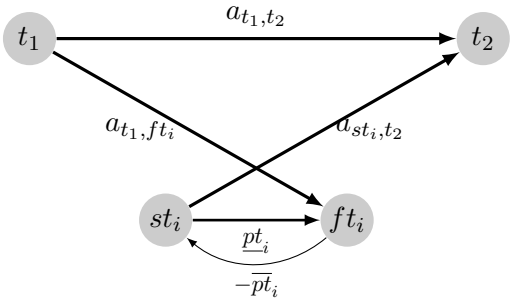


FIGURE 2.6 – Exemple de calcul de la consommation minimale étendue

La formulation de la condition de précédence interdite étendue est donnée dans la condition (2.21). Elle considère la paire de tâches $\{i, j\}$ en conflit pour une même ressource k et lorsque l'intervalle Δ correspond à $[st_i, ft_j]$, elle permet de conclure au séquencement interdit de i vers j en considérant le calcul de consommation étendue pour toute autre tâche k utilisant cette même ressource.

$$\text{si } a_{ft_j, st_i} < \underline{pt}_i + \underline{pt}_j + \sum_{t \in T_k \setminus \{i,j\}} \underline{we}_t^{[st_i, ft_j]} \text{ alors } i \not\prec j \text{ et donc } j \prec i \tag{2.21}$$

Nous avons montré [79, 5] que cette extension ne dominait pas la condition dont elle est issue. Chacune de ces conditions permet des déductions que l'autre ne trouve pas.

Une adaptation de cette condition dans le cas des STP avec incertitudes a également été réalisée [84]. Dans ce contexte, correspondant par exemple à des situations de planification de trajectoires, certaines distances sont dites *contingentes* dans le sens où leur valeur réelle est incertaine et ne peut être déterminée à l'avance. Vérifier la cohérence d'un STP avec incertitudes permet de garantir qu'il existe une instanciation des variables quelle que soient les valeurs des durées incertaines. On parle dans ce cas de *contrôlabilité dynamique*. Cette propriété est vérifiée hors-ligne par une extension d'un algorithme de chemin-consistance [119]. L'ajout de contraintes de partage de ressources introduit des disjonctions dans le modèle des STP avec incertitudes et on cherche alors à vérifier avant exécution du plan la propriété dite de *séquentiabilité dynamique* garantissant la faisabilité du plan quelle que soient les valeurs prises par les durées incertaines. Pour chercher à atteindre cette propriété de séquentiabilité dynamique, nous avons proposé une adaptation de l'extension de la condition de précédence interdite basée sur un raisonnement énérgétique.

Cette adaptation considère pour une paire de disjonction donnée $\{i, j\}$ les différentes possibilités de contingence et de non-contigence pour chacune des tâches et utilise en fonction de cela la durée maximale ou minimale de ces tâches.

2.5.3 Résultats expérimentaux

Problèmes d'ordonnancement flexibles

Nous avons considéré des problèmes d'ordonnancement dans lesquels la ressource devant être utilisée pour la réalisation d'une tâche doit être sélectionnée parmi un ensemble de ressources possibles et nous avons cherché à évaluer différentes modélisations et propagations sur ces problèmes.
Soit res_i, le choix d'affectation d'une tâche i, pour prendre en compte ces contraintes d'affectation, nous avons proposé différentes propagations spécifiques [82, 81] basée sur des propagations de contraintes de partage de ressources. Le premier type de propagation se base sur les propagations déduisant des précédences interdites. Le principe en est le suivant : soit une paire de tâches $\{i, j\}$ telle que i est affectée à une ressource r et j peut utiliser cette même ressource r, si $i \not\prec j$ et $j \not\prec i$ sur r alors $res_j \neq r$.
Le deuxième type de propagation se base sur le fait que les durées de réalisation dépendent de la ressource allouée. Si après propagation certaines durées s'avèrent impossibles pour une tâche, cela peut supprimer des choix d'affectation.

Nous avons cherché à déterminer des bornes inférieures pour le makespan de ces problèmes d'ordonnancement avec flexibilité de ressources [82, 81] en utilisant des propagations de contraintes de ressources classiques (2.6 et 2.9) et les différentes propagations de contraintes d'affectation proposées. Les instances considérées sont celles de flowshop hybrides de [149, 122] ou de jobshop flexibles de [86]. Les résultats obtenus ont montré tout d'abord l'intérêt de cette approche dédiée pour les problèmes d'ordonnancement avec flexibilité par rapport à l'approche purement temporelle ; de plus ils ont validé l'intérêt de la chemin-consistance en termes de calcul de borne (écart entre la borne inférieure déduite et l'optimum ou une borne supérieure de $2,57\%$ pour les 43 instances Edata de [86] avec amélioration de 5 bornes inférieures, contre un écart de plus de 32% en utilisant uniquement les propagations temporelles).
Cependant, pour les problèmes de jobshop flexibles présentant la plus grande flexibilité de ressource (instances Rdata et Vdata), les propagations proposées se sont avérées moins performantes. Enfin, concernant la propagation de contraintes d'affectation, elle s'est avérée sans intérêt pour les problèmes de flowshop hybrides, en revanche pour les problèmes de jobshop flexibles elle contribue à l'amélioration des bornes inférieures.

Problèmes d'ordonnancement avec gammes quelconques

Nous avons cherché à montrer l'intérêt des propagations étendues (2.7, 2.18 et 2.19) proposées par rapport aux propagations classiques [79]. Pour cela, nous avons considéré des problèmes d'ordonnancement disjonctif présentant des contraintes de gammes non linéaires permettant d'avoir des contraintes temporelles plus variées entre les tâches que les problèmes de jobshop ou de flowshop.
Les instances ont été obtenues à partir du générateur PROGEN utilisé en ordonnancement de projet [101]. Elles comportent de 20 à 70 tâches et de 3 à 4 ressources disjonctives et les caractéristiques du graphe de précédence varient en fonction d'une mesure dite de complexité de niveau faible, moyen ou élevé. Sur l'ensemble des instances générées, nous avons déterminé des bornes inférieures du makespan en utilisant soit des propagations classiques

soit des propagations étendues.
Les résultats obtenus montrent que les propagations étendues améliorent les bornes inférieures par rapport aux propagations classiques, le pourcentage d'instances dont la borne est améliorée augmente avec la complexité du réseau (de 38% d'instances améliorés pour une complexité faible à 62% pour une complexité élevée). L'amélioration obtenue est dans tous les cas de l'ordre de 5%.

Les expérimentations précédentes se sont limitées à des calculs de bornes inférieures. Les techniques de propagation proposées nécessitent d'être intégrées au sein d'une procédure de résolution afin de valider leur apport pour la résolution de problèmes d'ordonnancement. L'intérêt des extensions proposées est d'autant plus important que les problèmes d'ordonnancement présentent des caractéristiques temporelles variées (durées variables en fonction des affectations, contraintes de gammes quelconques, etc.) permettant de bénéficier pleinement des déductions pouvant être obtenues par chemin-consistance et des propagations de contraintes de ressources étendues exploitant les résultats de la chemin-consistance.

Problèmes d'ordonnancement avec contraintes de délais

Afin de poursuivre les travaux de validation des propagations étendues, nous nous sommes intéressés à un autre problème d'ordonnancement présentant des contraintes temporelles spécifiques appelées contraintes de délais ou *time-lags*. Dans un problème d'ordonnancement, une contraintes de time-lag permet de limiter la distance minimale et maximale entre deux tâches. Par exemple, cette contrainte permet d'exprimer que la distance entre les tâches i et j est comprise dans l'intervalle $[TL^{min}_{ft_i,st_j}, TL^{max}_{ft_i,st_j}]$ où $TL^{min}_{ft_i,st_j}$ est la valeur de time-lag minimum et $TL^{max}_{ft_i,st_j}$ la valeur de time-lag maximum.
De manière plus précise, nous avons considéré des problèmes de jobshop avec time-lags tels que les contraintes de time-lags ne soient présentes qu'entre deux tâches d'un même job. La valeur de time-lag minimum est nul et la valeur de time-lag maximum est positive.
Ce problème a notamment été étudié par [37] qui a proposé une heuristique d'insertion et un algorithme mémétique pour sa résolution ainsi que des jeux de test. Dans ces jeux de test, la valeur du time-lag maximum varie pour chaque instance de 0 à une valeur fonction de la durée des tâches du job. Ainsi, pour un même job, ces valeurs de time-lag maximum sont identiques. On peut noter que les problèmes les plus contraints sont ceux dits *sans attente*, pour lesquels les time-lags minimum et maximum sont nuls. Les problèmes ayant de grandes valeurs de time-lags maximums s'apparentent à des problèmes de jobshop classiques.

Une des caractéristiques de ce problème est la difficulté d'obtenir une solution réalisable autre que triviale (consistant à placer l'ensemble des jobs les uns à la suite des autres), de très mauvaise qualité vis-à-vis du makespan. L'heuristique d'insertion proposée dans la littérature place les tâches les unes après les autres (en fonction d'une règle de priorité) et effectue des réparations en cas de non respect des contraintes de time-lags lors du placement d'une tâche sur une ressource.
L'heuristique que nous avons proposée [5] se base sur la sélection d'un job et le placement de l'ensemble des tâches de ce job sur les ressources en utilisant les créneaux laissés disponibles par les placements précédents. Lorsque le placement d'une des tâches dans un créneau ne permet pas de respecter les contraintes de time-lag, elle est retardée ainsi qu'éventuellement les tâches la précédant. Dans le pire des cas, avec l'heuristique proposée comme avec celle de [37], on obtient un ordonnancement dans lequel les jobs sont placés les uns à la suite des autres. Notre heuristique a une complexité temporelle au pire exponentielle en fonction du nombre de ressources.

Les résultats obtenus ont montré que l'heuristique proposée était plus performante que celle de la littérature lorsque les valeurs de time-lags maximum étaient faibles et donc que le problème était fortement contraint par ces time-lags (écart à l'optimum de 32% pour notre heuristique contre 55% pour celle de la littérature). En revanche pour des problèmes moins contraints, l'heuristique de la littérature donne de meilleurs résultats (écart de plus de 34% pour notre heuristique contre 25% pour celle de la littérature).

En plus de cette heuristique d'insertion, nous avons utilisé une méthode de recherche dichotomique sur la valeur du makespan (voir algorithme 7) déterminant, à l'aide d'une recherche arborescente, s'il existe ou non une solution au problème pour une valeur donnée du makespan. Dans cette recherche arborescente, l'évaluation de chaque nœud est réalisée à l'aide de propagation de contraintes classiques ou étendues.
Les comparaisons effectuées ont porté sur la précédence interdite simple (2.6) et son extension (2.7), la précédence interdite basée sur le raisonnement énergétique (2.9) et son extension (2.21), ainsi que sur les conditions LSL et EFF et leurs extensions. De plus, nous avons également utilisé les règles de edge-finding (2.13 et 2.14) pour lesquelles nous ne disposons pas d'extension.
Les tests réalisés [5] ont permis de montrer que l'utilisation des extensions de la précédence interdite simple, de LSL et de EFF combinées avec les règles de edge-finding et la précédence interdite énergétique non étendue obtenait les meilleurs performances.
Sur 48 instances de petite taille (10 jobs et 5 ressources) nous avons pu en résoudre optimalement 41 grâce à cette combinaison de propagation, les instances les plus difficiles à résoudre étant celle sans-attente (time-lag maximum nul). En utilisant seulement les propagations non étendues avec le edge-finding seules 30 instances peuvent être résolues.

Ces expérimentations ont ainsi permis tout d'abord de montrer l'intérêt des extensions proposées, elles ont également mis en évidence que l'extension de la précédence interdite basée sur le raisonnement énergétique ne s'avérait pas performante. Enfin, elles ont montré que les extensions proposées n'étaient pas couvertes par les propagations de type edge-finding et qu'au contraire l'utilisation conjointe de edge-finding et des extensions proposées avaient un effet bénéfique sur les performances de la méthode.

Nous avons également comparé les résultats obtenus par notre méthode avec ceux obtenus par l'algorithme mémétique de [37]. Nous obtenons de meilleurs résultats que l'algorithme mémétique pour des instances de petites tailles, de son côté l'algorithme mémétique est plus efficace pour les instances sans attente et/ou de grandes tailles.

2.6 Conclusion

Dans ce chapitre nous avons exposé les travaux réalisés sur la modélisation de problèmes d'ordonnancement et sur le développement de nouvelles techniques de propagation de contraintes de ressources.
Des liens entre les CSP temporels et les problèmes temporels disjonctifs ont été proposés mais cette approche purement temporelle des problèmes d'ordonnancement ne permet pas pour le moment d'obtenir des méthodes de résolution efficaces.

Néanmoins, cette étude nous a permis de nous intéresser à des problèmes d'ordonnancement soumis à des contraintes temporelles variées (gammes quelconques, durées variables, délais entre tâches). Pour faire face aux caractéristiques temporelles de ces problèmes, nous avons donc proposé d'étendre certaines techniques de propagation classiques sur la base

d'une propagation complète des contraintes temporelles. Afin d'améliorer l'efficacité de nos méthodes, nous exploitons une propagation complète incrémentale des contraintes temporelles.
Les extensions que nous avons pu proposer concernent des conditions classiques s'exprimant sur des paires de tâches (même si les conclusions portent sur des ensembles de tâches). Les conditions basées sur des ensembles de tâches (début, durée ou fin d'un ensemble) ne peuvent pas être retrouvées à partir de la matrice de distance représentant l'ensemble des contraintes temporelles.
Les expérimentations que nous avons menées ont montré que les extensions proposées étaient performantes en particuliers pour des problèmes d'ordonnancement comportant de nombreuses contraintes temporelles.

Chapitre 3

Algorithmes de recherche de plus courts-chemins multimodaux bi-objectif

3.1 Introduction

Depuis quelques années, la multimodalité reflète une des principales évolutions du transport urbain ou inter-urbain de biens ou de passagers et son exploitation nécessite le développement de services suffisamment complets et performants afin de permettre aux passagers ou aux entreprises d'organiser au mieux leurs trajets. Malgré ce besoin, les outils existants les plus connus ne proposent généralement pas de réaliser des calculs d'itinéraires multimodaux. Un des principal frein au développement de ce type d'outils est lié à la difficulté d'accéder à l'ensemble des données des réseaux de transport considérées par la plupart des entreprises organisatrices de transport comme stratégiques. On peut néanmoins noter l'accélération des activités de recherche dans ce domaine et les évolutions législatives récentes concernant l'accès aux données devraient permettre l'émergence de nouveaux outils.
Les contributions que nous présentons dans ce chapitre ont été réalisées dans le cadre d'une thèse CIFRE.

Dans le domaine des transports, les méthodes de calculs d'itinéraires contribuent à deux activités qui sont :

- l'analyse de la performance des réseaux de transport s'appliquant lors de la phase de conception de ces réseaux afin de quantifier l'accessibilité de certains lieux. Cette analyse d'accessibilité est également appelé calcul d'*isochrones* et revient à déterminer les parties du réseaux de transport pouvant être atteinte avec un objectif à respecter, comme par exemple une durée maximale de transport.
- l'aide à la planification des itinéraires qui est généralement mise en œuvre lors de la phase d'exploitation des réseaux et qui s'appuie sur des outils de calculs de trajets point à point afin de répondre à des requêtes de transport émises par des utilisateurs du réseau.

Ces deux activités de calculs d'isochrones et de calcul d'itinéraires s'appuient sur des algorithmes de recherche de plus court chemin qui est l'un des problèmes de transport les plus anciens et les plus traités. Ce problème consiste à trouver le chemin partant d'un noeud origine O vers un noeud destination D et qui minimise un cout. Ce problème général peut prendre différentes formes selon :

- le type d'itinéraire recherché : chemin point à point de O vers D, chemin d'une origine

O vers tout autre sommet, chemin entre tout couple de sommet (un distancier) ;
- le nombre de chemins souhaités : 1 ou K plus courts chemins ;
- les objectifs à optimiser : un ou plusieurs objectifs à considérer.

Les méthodes de résolution varient selon la nature du réseau de transport (mono ou multimodal, statistique ou dépendant du temps, etc.) et selon l'intégration ou non de contraintes sur la nature des chemins obtenus.

Le problème sous-jascent à celui que nous avons étudié est celui de la recherche de plus courts chemins point à point sur des réseaux de transport multimodaux. Dans le contexte de réseaux de transport monomodal, la résolution du problème de plus court chemin a suscité le développement d'une multitude de méthodes afin de pouvoir maintenant aborder des réseaux à l'échelle d'un continent dans des temps de calcul très faibles. On peut notamment citer les approches bidirectionnelle, A^*, ou de hiérarchisation du réseau ainsi que des techniques de précalculs. De plus, de nombreuses méthodes ont été étendues au cas de problèmes dont les couts ou les temps de trajet dépendent du temps. En revanche, il existe relativement peu de travaux concernant les problèmes de plus court chemin point à point multimodaux.

3.2 Les algorithmes de plus courts chemins

3.2.1 Cas des réseaux de transport monomodaux statiques

Lorsque le réseau de transport est monomodal et que les valuations sont constantes (indépendantes du temps), l'algorithme de Dijkstra, datant de 1959, permet de résoudre le problème de plus court chemin d'un sommet vers tous les autres et par conséquent le problème de plus court chemin point à point. Il est adapté au cas où les valuations des arcs sont toutes positives ou nulles. Le principe de cet algorithme est le suivant : chaque sommet i est étiquetté par la longueur du plus court chemin entre O et i, notée π_i, par la valeur de son prédécesseur sur ce plus court chemin $pred_i$ et par un booléen $Marq_i$ valant $True$ si la valeur du plus court chemin est fixée. L'algorithme part du sommet de plus petit cout, le fixe ($Marq_i \leftarrow True$), met à jour le cout de ses successeurs : $\forall j \in Succ(i),\ \pi_j \leftarrow Min(\pi_j, \pi_i + trajet_{i,j})$ et met à jour $pred_j$ si π_j a été actualisé.
L'algorithme de Dijkstra est dit à **fixation d'étiquettes**, une fois qu'un sommet a été fixé (ie. a été sélectionné comme étant celui de plus petit cout), la valeur du plus court chemin de l'origine vers ce sommet est déterminée (elle ne peut diminuer car toutes les valuations sont positives ou nulles). L'algorithme stoppe quand le sommet destination est atteint (calcul de plus court chemin point à point) ou lorsque tous les sommets sont fixés (calcul de plus court chemin d'un sommet vers tous les autres). La complexité temporelle de cet algorithme réside dans la recherche du sommet de cout minimal. Avec une implémentation naïve, la complexité temporelle de l'algorithme de Dijkstra est au pire en $O(n^2)$, avec une structure de tas, elle passe en l'algorithme de Dijkstra est en $O((n+m).logn)$, c'est à dire en $O(m.log(n))$ lorsque $m > n$ (ce qui est le cas de réseaux de transport qui sont connexes).

Différentes améliorations de l'algorithme de Dijkstra ont été proposées dans la littérature pour aborder des graphes de très grande taille (ie. contenant des millions de sommets et d'arcs). Les améliorations les plus anciennes sont la **recherche bidirectionnelle** et la **recherche orientée** (Goal Oriented Search ou A*).
La recherche bidirectionnelle [124, 90] consiste à calculer les plus courts chemins en partant alternativement du sommet origine (recherche dite **en avant** ou **forward** sur le graphe G) et du sommet destination (recherche dite **en arrière** ou **backward** sur le graphe inverse G^{-1}). Lorsqu'un sommet est atteint par les deux sens de recherche, un chemin entre l'ori-

gine et la destination est trouvé. Ce chemin est optimal s'il est de plus petit cout que la plus petite des connexions potentielles non encore développées.
La recherche orientée aussi appelée A* [56] évite à l'algorithme de Dijkstra de visiter des sommets s'éloignant de la destination. Pour cela, le cout d'un sommet i correspond à la fois à la valeur du plus court chemin de l'origine vers i (π_i) mais aussi à une estimation de la valeur du plus court chemin de ce sommet vers la destination notée $e(i)$. L'algorithme sélectionne alors le sommet ayant le plus petit cout $\pi_i + e_i$. Si l'estimation vérifie de bonnes propriétés d'admissibilité (c'est à dire que plus un sommet est loin de la destination et plus son estimation est élevée) et de borne inférieure du plus court chemin, l'optimalité de l'algorithme est garantie. Sur des réseaux de transport, l'estimation utilisée classiquement est la distance euclidienne (couplée avec une vitesse de déplacement maximale pour obtenir une estimation en temps).
La combinaison de la recherche bidirectionnelle avec la recherche orientée a permis de réduire les temps de calcul d'un facteur 2 à 3 [48].
Un autre type d'accélération réside dans le pré-calcul d'un certain nombre de plus courts chemins. La technique la plus ancienne s'appuie sur le pré-calcul des plus courts chemins de tous les sommets vers un ensemble restreint de sommets donnés appelés *landmarks* et l'utilisation des valeurs calculés pour avoir une estimation d'un chemin point à point donné en s'appuyant sur l'inégalité triangulaire. L'utilisation des landmarks couplée à une recherche de type A^* a donné lieu à la méthode appelée ALT. Différents types de pré-calculs, liés à la topologie des graphes, ont également été considérés et ont permis d'améliorer l'efficacité des algorithmes sur des graphes de grande taille [62, 63].

Depuis une dizaine d'années, de nombreuses techniques d'accélération de l'algorithme de Disjkstra ont été proposées dont la synthèse est exposée dans [48]. Ces techniques s'appuient sur la topologie du graphe, le pré-calcul d'un certain nombre de plus courts chemins ou sur la combinaison de plusieurs de ces techniques. Ces améliorations ont permis de réduire à la fois les temps de calcul et l'espace mémoire nécessaire lors des pré-calculs de plus court chemin. Pour avoir un ordre de grandeur des résultats obtenus sur des graphes de très grande taille (de 18 à 33 millions de sommets pour 42 à 75 millions d'arcs), les temps de précalcul sont exprimés en dizaine de minutes et les temps de calcul d'un itinéraire sont exprimés en millisecondes.

3.2.2 Cas des réseaux de transport monomodaux dépendants du temps

Les réseaux de transport monomodaux dynamiques sont des réseaux dans lesquels les valuations (cout ou temps de trajet) dépendent du temps. Ils sont classés selon qu'ils respectent ou non la propritété dite FIFO : partir plus tôt d'un sommet i permet d'arriver plus tôt à un sommet j. Le problème de plus courts chemins **en temps de trajet** sur un réseaux de transport dépendant du temps a largement été étudié dans la littérature [39, 41, 46, 121, 120].

Un des modèles utilisés se base sur un graphe dit temporel dans lequel les valuations des arcs sont dépendantes du temps et représentés par des fonctions croissantes, ce qui permet de respecter la propriété FIFO. Dans le cas de problème de plus court chemin en temps de trajet point à point pour une date donnée sur des graphes temporels, l'algorithme de Dijkstra peut être directement étendu en considérant que les étiquettes dépendant du temps. La complexité au pire reste identique au cas de graphes statiques en supposant que les valuations dépendant du temps sont obtenues en $O(1)$.

Différentes accélérations existant dans le cas statique ont été adaptées avec succès dans

le cas de réseau dépendant du temps comme A*, le calcul de landmarks, certains pré-calculs [46, 121, 120]. La recherche bidirectionnelle est plus délicate à mettre en œuvre pour l'étape de recherche backward (on ne connait pas la date d'arrivée à la destination) ; dans [121], les auteurs développent une méthode bidirectionnelle dans laquelle la phase de recherche en forward est dépendante du temps et la phase de backward est indépendante du temps. Les résultats obtenus avec la méthode proposée montrent que la variante bidirectionnelle s'avère un peu moins rapide que la variante uni-directionnelle (méthode ALT en dépendant du temps) mais que si on s'autorise à perdre la garantie de l'optimalité des itinéraires la variante bidirectionnelle devient largement plus efficace.

Les travaux de Pallattino et al. [128] se sont intéressés au cas plus général où les arcs sont valués par des couts dépendant du temps et à la détermination de chemin de cout minimum. Ils s'appuient sur une modélisation du problème par un graphe espace-temps dans lequel, à partir d'un graphe $G(N, E)$ et d'un ensemble d'instants $T = (t_1, t_2, ..., t_q)$ on définit pour chaque sommet i de N, un ensemble de sommets $(i, t_k), \forall t_k \in T$ ainsi que les valuations des arcs (i, j), notées $a_{i,j}(t_q), \forall t_k \in T$. L'algorithme proposé est un extension de celui de Dijkstra ; il s'appuie sur la discrétisation du temps en l'ensemble T et partitionne les valuations des sommets pour chacun de ces instants. La modélisation basée sur un graphe espace-temps est cependant limitée à des problèmes pour lesquels le nombre d'instants reste limité comme par exemple des réseaux de transport en commun.

Des expérimentations ont été effectuées sur le réseau ferroviaire allemand pour comparer l'approche basée sur un graphe temporel et celle basée sur un graphe espace-temps [138] ; elles ont montré que les algorithmes s'appuyant sur un graphe temporel étaient plus performants. Cependant, lorsque ce graphe doit être étendu pour prendre en compte certaines caractéristiques liées notamment à des transferts de train, la représentation par graphe espace-temps s'avère plus compacte et l'écart entre les deux approches n'est plus aussi important.

3.2.3 Cas des réseaux de transport multimodaux

Problématique

La prise en compte de la multimodalité a été abordé tout d'abord d'un point de vue de la modélisation du réseau de transport. Deux modélisations ont été principalement développées : graphes multi-couches ou graphes multi-valués.
La modélisation par graphes multi-valués [153] est une représentation compacte dans laquelle les couts sont définis sur chaque arc pour chaque mode de transport ; elle nécessite le développement d'algorithmes dédiés de calculs d'itinéraires et a été adaptée pour des problèmes statiques et dynamiques.
Avec une modélisation par graphes multi-couches, le réseau de transport multimodal est représenté par un graphe $G(N, E, M)$ où M est l'ensemble des modes existant dans le réseau. Le graphe G correspond à un ensemble de M graphes associés chacun à un mode de transport (ie. une couche de transport). A chaque sommet $i \in V$, on associe une étiquette m_i donnant le mode auquel il appartient. De même à chaque arc (i, j), en plus de la valuation $trajet_{i,j}$ on associe une étiquette $m_{i,j}$ représentant le mode auquel cet arc est associé. Un arc (i, j) reliant deux sommets appartenant à un mode différent $(m_i \neq m_j)$ est appelé un arc de transfert. Dans la pratique, les transferts sont considérés comme un mode spécifique [109] que l'on ajoute à l'ensemble M des modes de transport effectifs.

La multimodalité fait surgir des contraintes spécifiques pour le problème de calcul d'iti-

néraires : en effet certains enchainements de modes s'avèrent irréalisables soit en raison de contraintes liées aux modes soit en raison de contraintes (ou de préférences) de voyageurs. Par exemple, utiliser de manière successive les modes « véhicule personnel, transport en commun puis véhicule personnel »est irréalisable car une fois le véhicule personnel quitté pour prendre un transport en commun il n'est plus accessible ailleurs dans le réseau. Un autre exemple peut traduire le cas d'un voyageur ne souhaitant pas prendre plusieurs fois le mode « métro ». Ces contraintes d'utilisation des différents modes de transport sont appelées **contraintes de viabilité** et un chemin respectant ces contraintes est un **chemin viable**.

Le problème de recherche de plus courts chemins multimodaux bi-objectifs le plus simple a été introduit par Pallotino et Scutella [128]. Le problème posé est polynomial et consiste à déterminer l'ensemble des plus courts chemins point à point non-dominés vis-à-vis d'un objectif de minimisation du temps de trajet et d'un objectif de minimisation du nombre de transferts. Pour résoudre ce problème, les auteurs proposent une extension d'un algorithme à fixation d'étiquettes. Une première extension de ce problème à la prise en compte de contraintes sur la viabilité des itinéraires a été introduite par Lozano et Storchi [109]. Pour cela, une modélisation des contraintes de viabilité basées sur un automate à états finis a été introduite et les auteurs proposent une extension de l'algorithme de [128] incluant les contraintes sur les modes. Une seconde extension de ce problème a été proposée par Bielli et.al [22] pour prendre en compte des temps de trajet dépendant du temps. Le problème posé est un peu différent dans le sens où les auteurs cherchent à déterminer les K meilleurs chemins avec une limite sur le nombre maximum de transferts autorisés, l'algorithme proposé est également une extension de celui de [128]. Une limite commune à toutes ces études est l'absence d'expérimentations sur des réseaux de taille réelle.

Des extensions de problèmes de plus courts chemins multi-objectifs NP-difficiles ont été proposées dans le cas multimodal par T. Grabener et al. [64]. Dans ces travaux, les contraintes de viabilité des itinéraires ne sont pas prises en compte et des extensions de l'algorithme de Martins [113] ont été proposées. En considérant uniquement la minimisation du temps de trajet et la minimisation du nombre de transferts, la méthode proposée a montré son efficacité sur des graphes de grande taille prenant en compte les modes de transport en commun et le vélo, néanmoins en l'absence de contraintes de viabilité, le nombre de solutions non-dominées est très faible (le mode vélo est dominant). Lorsque les auteurs prennent en compte un objectif supplémentaire (par exemple la minimisation du dénivelé), le problème devient NP-difficile et les temps de résolution sont nettement plus importants. En revanche, la taille de l'ensemble de solutions non-dominées augmente.

Modélisation des contraintes sur les modes

Une première modélisation des contraintes de viabilité a été proposée dans [109], elle consiste à définir un automate dont les états traduisent l'enchainement des modes de transport et dont les transitions représentent les évolutions autorisées pour l'enchainement de ces modes. Cet automate est défini par : $A = (S, s_0, F, M, \delta)$ où

- S est l'ensemble des états ;
- s_0 est l'état initial ;
- F est l'ensemble des états finaux ;
- M est l'ensemble des modes utilisés sur le graphe de transport ;
- δ est la fonction de transition definie de $M \times M \times S \to S$. $s' = \delta(m, m', s)$ donne l'état s' obtenu lorsque l'arc (i, j) est emprunté partant d'un sommet i du mode m dans l'état s et allant vers un sommet j du mode m'. Par convention, $\delta(m_i, m_j, s) = \emptyset$

signifie que l'evolution d'un nœud du mode m et à l'etat s via le mode m' n'est pas viable.

Un chemin origine destination est viable si le sommet destination est dans un des états finaux de A.

Avec cette approche, le calcul de plus courts chemins multimodal avec contraintes de viabilité s'appuie sur deux éléments : le graphe de transport et l'automate de viabilité.

Cette modélisation des contraintes de viabilité a été introduite pour résoudre des problèmes multimodaux bi-objectifs avec minimisation du temps de trajet et du nombre de transferts par [109]. Elle a été étendue au cas de problèmes avec temps de trajet dépendant du temps par [22].

Dans les algorithmes développés, lors de l'exploration des sommets, les étiquettes comportent une information liée à l'état de ce sommet dans l'automate. Ainsi, à chaque sommet on associe différentes étiquettes liées à ces états. Des règles de dominance basées sur les états de l'automate peuvent de plus être mises en œuvre.

Une seconde approche consiste à déterminer des plus courts chemins avec contraintes de labels ou *label-constraint shortest path problems* [9, 8, 141, 140]. Cette approche se base sur un graphe $G(N, E)$, sur un langage L et sur un alphabet Σ. A chaque arc (i, j) sont associés une valuation $a_{i,j}$ et un élément de Σ : $\sigma_{i,j}$. Le problème de plus court chemin avec contraintes de labels consiste à déterminer un plus court chemin point à point tel que le mot formé par la concaténation des labels le long de ce chemin soit un mot du langage L.

Le langage L peut être utilisé pour traduire les contraintes de viabilité d'un réseau multimodal. Dans ce cas, les éléments de l'alphabet correspondent aux différents modes étiquettant les arcs. La complexité des algorithmes dépend du langage L, lorsque L est un langage régulier ils restent polynomiaux et se basent sur un automate à états finis non déterministe pour la représentation de L.

Cette approche a été développée pour la résolution de problèmes multimodaux dans un contexte mono-objectif uniquement (minimisation du temps de trajet) et a également été considérée dans le cas de temps de trajet dépendant du temps [141, 140].

Dans les algorithmes proposés, différentes techniques d'accélération ont été mises en œuvre comme la recherche A^* et recherche bidirectionnelle dans [8]. Par ailleurs, une application restreignant les combinaisons de modes possibles [47, 127] et réduisant dont les mots reconnus par l'automate a permis d'exploiter des techniques de pré-calcul existant dans les graphes mono-modaux et les algorithmes proposés se sont avérés efficaces sur un cas pratique contenant une très forte proportion de réseau routier par rapport au réseau de transport en commun.

3.3 Contributions apportées

3.3.1 Définition du problème

Le problème central que nous avons étudié est celui de la recherche du plus courts chemins point à point bi-objectif viable appellé **BI-MM-V-SP : Bi-objective Multimodal Viable Shortest Path** dans le cas de graphe statique ou **BI-MM-V-TD-SP** lorsque le temps de trajet dépendent du temps. Pour ces problèmes, les deux objectifs que nous avons considérés sont la minimisation du temps de parcours et du nombre de transferts ou nombre de changements de modes.

Modélisation du graphe de transport. La modélisation retenue pour le graphe

de trabsport est basée sur un graphe multi-couches valué, noté $G_T = (N, E, M)$ où N est l'ensemble des nœuds, E l'ensemble des arcs et M l'ensemble des couches (ie. modes). Dans l'application pratique qui nous a guidé, ce graphe est composé :

- d'un sous-graphe représentant le *réseau routier*. Dans ce sous-graphe, des arcs peuvent être utilisés par les modes *piéton*, noté Ma (pour marche), ou *véhicule personnel*, noté VP. D'autres arcs peuvent n'être utilisés que par un seul mode (dans ce cas le temps de trajet associé au mode non permis est considéré comme infini). Nous avons supposé que les temps de trajet du réseau routier étaient statiques.
- d'un ensemble de sous-graphes représentant chacun les différentes *lignes de bus* , noté Bus, ou *de métro*, noté Me. Les valuations des différents arcs peuvent dépendre du temps et être représentées par des tables horaires ou par des fréquence de passage.
- d'un ensemble d'*arcs de transfert* assurant le passage entre les différents sous-graphes définis précédemment.

Cette modélisation (basée sur une décomposition en plusieurs couches des lignes de transport en commun) garantit l'hypothèse que le graphe de transport respecte la propriété FIFO (partir plus tôt d'un point i permet toujours d'arriver plus tôt en un point j).

Modélisation de la viabilité. La viabilité des itinéraires est représentée par un automate de viabilité $A = (S, M, \delta, F, s_0)$ défini précédemment. Pour l'application pratique, nous nous sommes limités à la prise en compte de 4 modes de transport $M = \{Ma, VP, Bus, Me\}$ auxquels s'ajoute le mode spécifique de transfert Tr. Les contraintes de viabilité considérées sont :

- le mode Me ne peut être utilisé qu'une seule fois, de plus l'origine et la destination ne sont pas dans le métro ;
- le mode VP ne peut être utilisé qu'à partir de l'origine, une fois quitté il ne peut être utilisé de nouveau et il ne permet pas d'atteindre la destination (il est nécessaire de se garer).

L'automate considéré comporte les 4 états définis dans le tableau 3.1, ses états finaux sont $F = \{s_1, s_4\}$.

Etats	Signification
s_0	état initial : aucun mode n'est encore utilisé
s_1	le mode métro n'a pas été utilisé
s_2	le mode véhicule n'a pas encore été quitté
s_3	le mode métro n'a pas encore été quitté
s_4	le mode métro a été quitté

TABLE 3.1 – Etats de l'automate de viabilité

Cet automate de viabilité est représenté à la figure 3.1. Par exemple, à partir de l'état initial s_0, utiliser les modes Ma ou Bus conduit dans l'état s_1 et utiliser le mode VP conduit dans l'état s_2. Les états finaux de cet automate (s_1, s_4) traduisent le fait que l'on ne peut arriver à la destination qu'en marchant ou en bus.

3.3.2 Règles de dominance

Définition des labels

Dans les algorithmes proposés, on associe à chaque nœud du graphe de transport une étiquette (ou label) (i, s, k) représentant un chemin partiel viable de k transferts partant du nœud origine, arrivant au nœud i dans l'état s. Deux valeurs correspondent à chaque label :

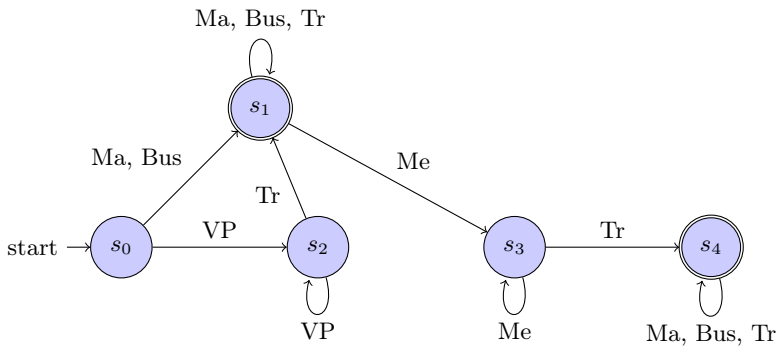


FIGURE 3.1 – Automate de viabilité considéré

- π_{is}^k : le temps de trajet le plus court depuis l'origine au nœud i dans l'état s et avec un nombre de transfert k ;
- $pred_{is}^k$: le label prédécesseur pour la reconstruction du chemin. Si $pred_{is}^k = (j, s', k')$ alors l'arc (j, i) est dans le chemin de cout minimum pour (i, s, k), l'état du chemin au nœud j est s', le nombre de transferts utilisé est k' (avec $k' = k$ si $m_i = m_j$, $k' = k - 1$ si $m_i \neq m_j$) et $s = \delta(m_j, m_i, s')$.

Pour le cas de graphes de transport dépendants du temps et vérifiant la propriété FIFO, il ne peut exister deux trajets de i vers j tel que l'un partant avant l'autre de i arrive après l'autre à j. Une seule valeur du temps minimal de trajet doit donc être stockée à chaque label (i, s, k) donné. Ainsi, pour chaque nœud $i \in N$ du graphe de transport étudié $G_T = (N, E, M)$, il y a plusieurs labels mais le nombre de labels à considérer est limité par le nombre d'états de l'automate de viabilité (au plus |S|) et par le nombre maximum de transferts, noté $K_{\max}$ avec $K_{\max} < |N|$. Il s'ensuit que le problème à résoudre est polynomial.

Régles de dominances proposées

Afin d'élaguer certains labels lors du déroulement des algorithmes nous avons proposé deux règles de dominance. La première, dite *dominance simple*, compare des labels en temps de trajet et en nombre de changements de modes. La seconde, dite *dominance états*, introduit en plus des comparaisons précédentes des relations entre les états de l'automate de viabilité.

Proposition 3.3.1 (Dominance simple) *Soient deux labels (i, s, k) et (i, s, k'). Si $k \leq k'$ et $t_{is}^k \leq t_{is}^{k'}$ alors (i, s, k) domine (i, s, k') qui peut être élagué.*

Ainsi un label d'un nœud i dans un état s domine un label pour ce même nœud i dans le même état s s'il correspond à la fois à moins de transfert et à une plus petite date d'arrivée. Avec cette première règle de dominance, un label (i, s, k) ne peut pas être dominé par un autre label (i, s', k') dès que $s \neq s'$. Pour cela nous avons proposé une deuxième règle de dominance qui renforce la règle 3.3.1 en s'appuyant sur une relation binaire $\preceq$ entre états. $s \preceq s'$ signifie que l'état s a au moins autant de possibilités d'extension que l'état s'. Plus précisément $s \preceq s'$ si pour toute paire de modes de transport m et m', une des conditions suivantes est vérifiée :

$$\delta(m, m', s') = \delta(m, m', s) \tag{3.1}$$

$$\delta(m, m', s) = s \ \ et \ \ \delta(m, m', s') = s' \tag{3.2}$$

$$\delta(m, m', s') = \emptyset \tag{3.3}$$

La première condition traduit le fait que chaque état atteignable depuis s' par une transition d'un mode m vers un mode m' l'est aussi depuis l'état s. Lorsque la deuxième condition est vérifiée, une transition d'un mode m vers un mode m' conserve l'état s' et conserve également l'état s. La troisième condition assure le fait qu'il ne peut y avoir de transition d'un mode m vers un mode m' en s' si elle n'est pas autorisée depuis s.
Par exemple, dans l'automate de viabilité de la figure 3.1 on a $s_1 \preceq s_4$.

Cette relation entre états, tout en gardant le même objectif, simplifie celle proposée dans [109], qui se limitait à des conditions sur les modes de transport contraints (tels le véhicule personnel et le métro) et qui comportait des ambiguités lors de la comparaison d'états n'ayant pas encore utilisés de mode contraint. Sur la base de cette relation entre états, la deuxième règle de dominance proposée est :

Proposition 3.3.2 (Dominance états) *Soient les labels (i, s, k) et (i, s', k'). Si $k \leq k'$, $t_{is}^k \leq t_{is}^{k'}$ et $s \preceq s'$ alors (i, s', k') est dominé et peut être élagué.*

Proposition 3.3.3 (Equivalence états) *Deux états s et s' tels que $s \preceq s'$ et $s' \preceq s$ sont dits équivalents car ils ont les mêmes possibilités d'extension. Ainsi, dans un automate de viabilité, deux états équivalents s et s' peuvent être fusionnés.*

3.3.3 Algorithmes proposés

Les différents algorithmes décrits ci-dessous permettent de résoudre les problèmes BI-MM-V-SP ou BI-MM-V-TD-SP considérés. Ils s'appuient sur un graphe multicouches pour représenter le réseau de transport et sur automate à états finis pour modéliser les contraintes de viabilité des itinéraires. Ces algorithmes déterminent l'ensemble des itinéraires non-dominés entre une origine et une destination ou depuis une origine vers tout autre sommet du graphe.

Principe général de fixation d'étiquettes

Les différents algorithmes que nous avons proposés s'appuient sur une liste de labels à explorer (implémentée par une structure de données adéquate). Ils étendent le label (i, s, k), de cout π_{is}^k minimal, vers l'ensemble de ces successeurs, si ce label n'est pas élagué par une des règles de dominance. Ainsi, pour chaque successeur j, l'algorithme détermine son état s' ($s' = \delta(m_i, m_j, s)$), son nombre de transfert k' ($k' = k$ si $m_i = m_j$, sinon $k' = k + 1$) et actualise le label (j, s', k') si besoin (ie. les valeurs $\pi_{js'}^{k'}$ et $pred_{js'}^{k'}$).

Adaptation de l'algorithme de [109]

L'algorithme proposé dans [109] et noté TLS (Topological Label Setting) permet de déterminer les plus courts chemins par nombre de transferts croissants et par temps de trajet décroissant.
Il itère sur le nombre de transferts (de 0 à K_{max}) et, à chaque itération k, il utilise deux listes de labels, notées Q_{now} et Q_{next} (implémentées par des tas binaires) pour mémoriser respectivement les labels à k transferts et ceux à $k + 1$ transferts. A la fin de l'itération k, les labels contenus dans Q_{next} sont transférés dans Q_{now} ce qui limite les redondances de l'algorithme. Lors des différentes itérations, afin d'appliquer les règles de dominance pour élaguer un label (j, s', k'), on a besoin de conserver la meilleure valeur précédemment trouvée pour tout nœud j à l'état s' (cette valeur a nécessairement été obtenue pour des nombres de transferts $k'' < k'$).

Les adaptations de cet algorithme que nous avons effectuées sont : d'utiliser la règle de dominance 3.3.2 proposée ; d'effectuer le test de dominance lorsqu'un label est extrait de Q_{now} et non pas avant l'ajout dans une des files Q_{now} ou Q_{next} afin d'améliorer les performances de l'algorithme et finalement d'utiliser une valeur $\pi^{k'}_{js'}$ pour chacune des files Q_{now} et Q_{next} et non pas une seule valeur $\pi_{js'}$ comme suggéré dans l'article (car à un nœud i, un chemin ayant k transferts pourrait ne pas être étendu s'il correspond à un plus long temps de trajet qu'un chemin à k+1 transferts).
Pour un graphe de transport $G_T = (N, E, M)$ avec $n = |N|$ et $m = |E|$ et un automate de viabilité comportant $|S|$ sommets, nous avons établi la complexité temporelle dans le pire des cas de cet algorithme qui est en $O(K_{max}.m.|S|.(|S| + log(n.|S|)))$ lorsque la règle de dominance basée sur les états est appliquée et en $O(K_{max}.m.|S|.(log(n.|S|)))$ sans utilisation de règle de dominance [66].
Un des inconvénients de cet algorithme est lié à la difficulté d'en proposer une version bidirectionnelle. En effet, l'obtention des solutions par valeurs croissantes du nombre de transferts pose des problèmes lors de la connexion des recherche en avant et en arrière car on ne peut pas se limiter à tester des connexions entre chemins partiels ayant un même nombre de transferts. Face à cette difficulté, nous avons proposé un nouvel algorithme dont l'extension en bidirectionnel est plus aisée.

Proposition d'un nouvel algorithme : MQLS

Le principe de l'algorithme MQLS (Multi-Queues Label Setting) que nous avons proposé est de déterminer l'ensemble des plus courts chemin dans l'ordre croissant du temps de trajet et donc dans l'ordre décroissant du nombre de transferts. Pour cela, il nécessite un ensemble $\mathcal{Q}$ de listes de labels (implémentées par des tas binaires) tel que chaque liste Q_k contienne les labels à k transferts. Avec l'algorithme MQLS, le label de cout minimal (i, s, k) est recherché dans l'ensemble des listes $\mathcal{Q}$. Puis, s'il n'est pas dominé, ses labels successeurs (j, s', k') sont calculés et insérés dans les listes $Q_{k'}$ correspondant à leur nombre de transferts. Lorsqu'un plus court chemin est déterminé d'une origine vers une destination, son nombre de transfert k^* permet de supprimer toutes les listes Q_k telles que $k \geq k^*$. L'algorithme stoppe lorsqu'il n'y a plus de labels dans les files Q_k, il ne nécessite donc pas de fixer a priori une valeur maximale du nombre de transferts K_{max}. Dans le pire des cas, sa complexité temporelle est en $O(K_{max}.m.|S|.(K_{max}.|S| + log(n.|S|)))$ avec la règle de dominance basée sur les états et en $O(K_{max}.m.|S|.(K_{max} + log(n.|S|)))$ sans règle de dominance. Sa complexité temporelle dans le pire des cas est moins bonne que celle de l'algorithme TLS ci-dessus.

Comparaisons expérimentales

Les expérimentations ont été menées sur une partie du réseau de transport de la ville de Toulouse dont les caractéristiques générales sont fournies dans le tableau 3.2.

Nous avons considéré que les modes de transport en commun (métro et bus) disposent de tables horaire ou de fréquence de passage et que les autres modes (marche et véhicule personnel) sont statiques. Le graphe dépendant du temps utilisé est le même que le graphe statique, cependant l'intégration des horaires réels des bus, dans la modélisation par graphe multi-couches, a conduit à éclater une ligne de bus en utilisant la notion de missions (ensemble d'itinéraires effectifs associés à une même ligne) et de courses (association des horaires aux différentes missions). Ceci a entrainé, d'une part, une augmentation du nombre nœuds et d'arcs pour le mode *Bus* et, d'autre part, une augmentation du nombre

	Graphe statique		Graphe dynamique	
Modes	**Nœuds**	**Arcs**	**Nœuds**	**Arcs**
Bus	6 170	6 646	9 384	9 047
Métro	75	72	75	72
Voirie (Marche et VP)	56 774	146 280	56 774	146 280
Transfert	-	6 370	-	28 017
Parking	29	-	29	-
Total	63 048	159 368	66 262	183 416

TABLE 3.2 – Caractéristiques du graphe de transport

d'arcs de transfert pour permettre les changements de modes.

A partir de ces deux graphes, nous avons généré aléatoirement 100 couples origines-destinations répartis dans les zones périphériques de l'agglomération.
Dans le graphe dépendant du temps, l'horaire de départ est fixé à 8h du matin, le nombre d'horaires est de 129 975 et la fréquence de passage du métro est de 3 minutes.

Tous les algorithmes déterminent l'ensemble des trajets non-dominés. Dans ces expérimentations, le nombre moyen de solutions non-dominées est de 5 pour un nombre de transferts moyen de 2. L'obtention de plusieurs solutions non-dominées pour un trajet donné, en termes de temps de trajet et de nombre de transferts, est en grande partie liée à l'interdiction de stationner un véhicule à la destination. Cette hypothèse, réaliste en mileu urbain, permet d'obtenir une plus grande variété de solutions. En l'absence de cette hypothèse, le mode VP domine les autres et il n'y a souvent qu'une seule solution non-dominée consistant à réaliser tout le trajet origine-destination en voiture (et donc avec 0 transfert). La distance moyenne des itinéraires trouvée est de 40 km (entre 30 et 50), ces itinéraires sont plus longs que ceux classiquement réalisés sur l'agglomération toulousaine mais ils permettent d'avoir un meilleur aperçu des performances relatives de nos algorithmes et des règles de dominance.

La figure 3.2 montre l'impact de la règle de dominance simple et de celle basée sur les états pour les différents algorithmes en termes de temps de calcul.

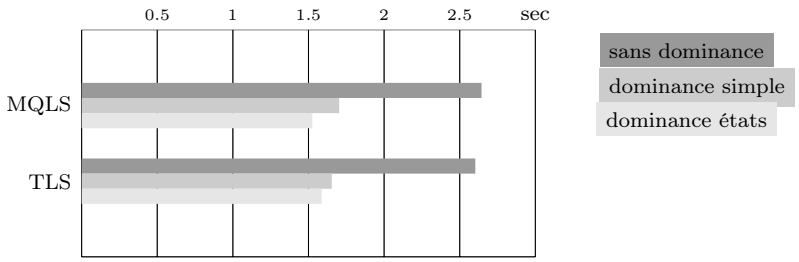


FIGURE 3.2 – Impact des règles de dominances sur le graphe statique

Ces résultats montrent l'apport significatif de la règle de dominance simple pour les algorithmes TLS et MQLS et met aussi en évidence l'apport de la règle de dominance basée sur les états.
Par ailleurs, bien que la complexité théorique au pire de MQLS soit moins bonne que celle de TLS (avec ou sans dominance), en pratique, lors de l'utilisation de la dominance basée sur les états, MQLS a un meilleur temps de calcul que TLS (ce qui n'est pas le cas sans dominance ou avec la dominance simple).

Ces résultats sont confirmés dans le cas d'un graphe dépendant du temps.

3.3.4 Approche bidirectionnelle

Afin d'améliorer les performances de nos algorithmes, nous avons développé une variante bidirectionnelle de l'algorithme MQLS, notée FB-MQLS dans lequel chaque phase de recherche (en avant et en arrière) s'appuie sur l'algorithme MQLS. Pour la phase de recherche en arrière, il est nécessaire d'ébalir un automate de viabilité traduisant la viabilité sur le graphe inverse. Pour cela, deux approches ont été étudiées : soit considérer l'automate inverse qui est alors non-déterministe ce qui peut générer des labels ne pouvant jamais atteindre l'origine et donc des calculs inutiles, soit déterminiser l'automate de viabilité inverse mais l'obtention de cet automate déterministe dédiés par des algorithmes peut conduire dans le pire des cas à un automate ayant un nombre exponentiel de sommets.
Un deuxième aspect à considérer pour la mise au point d'un algorithme bidirectionnel est la définition d'une condition d'optimalité garantissant que la connexion trouvée par les phases de recherche avant et arrière est un chemin optimale. Pour cela, nous avons proposé une condition [67] généralisant celle existant dans le cas mono-modal. Comme l'algorithme MQLS détermine les plus courts chemins indépendamment du nombre de transferts, il suffit de comparer la meilleure connexion trouvée à la meilleure connexion possible pour les labels restant dans les files de la recherche en avant et en arrière. Lorsqu'un plus court chemin est déterminé, son nombre de transerts k^* est déterminé et les files de la recherche en avant et en arrière correspondant à un nombre de transferts supérieur ou égal peuvent être supprimées.

Adaptation au cas dépendant du temps

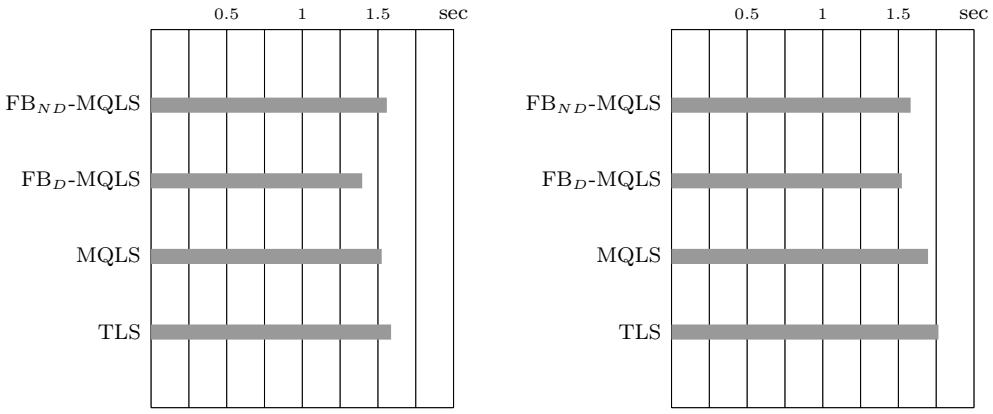
Les algorithmes TLS et MQLS dans le cas de graphes dépendants du temps fonctionnent sur le même principe que ceux proposés dans le cas statique et déterminent les plus courts chemins non-dominés entre une origine et une destination pour une date de départ t (ou d'une origine vers tout autre sommet). Le calcul des temps de trajet d'un nœud i vers nœud j appartenant à un des modes de transport en commun s'effectue sur la base d'une table mémorisant la date d'arrivée en j pour chaque instant d'une journée (avec une précision à la minute).

En revanche, la mise en oeuvre d'un algorithme bidirectionnel est plus problématique car on ne connait pas la date d'arrivée à la destination. Nous avons suivie la démarche proposée par [121] pour une évaluation par défaut du graphe inverse utilisé par la recherche en arrière. Lorsqu'une connexion est établie avec la recherche en avant, le temps de trajet réel du chemin origine destination peut alors être déterminé.

Comparaison expérimentales

La figure 3.3 donne l'impact de la recherche bidirectionnelle sur le graphe statique et sur celui dépendant du temps en incluant la règle de dominance basée sur les états. FB_{ND}-MQLS et FB_{D}-MQLS représentent respectivement les variantes de l'algorithme FB-MQLS avec un automate non déterministe ou un automate déterministe pour traduire les contraintes de viabilité dans la phase de recherche en arrière.

Les résultats obtenus montrent que les variantes bidirectionnelles améliorent les variantes mono-directionnelles et que l'utilisation d'un automate de viabilité déterministe conduit à l'algorithme le plus performant.



(a) Graphe statique (b) Graphe dépendant du temps

FIGURE 3.3 – Impact de la recherche bidirectionnelle

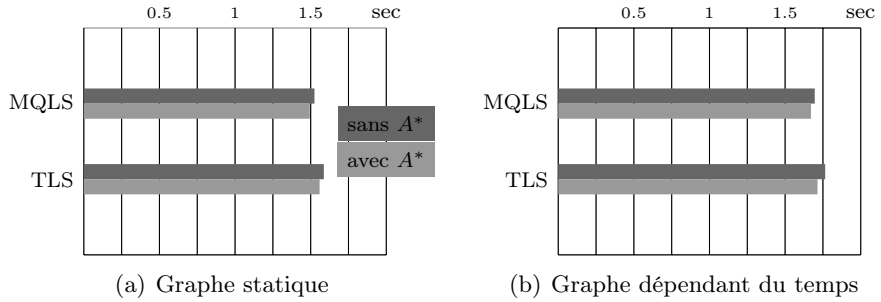
3.3.5 Variantes A^*

Les algorithmes TLS, MQLS ont été adaptés simplement pour utiliser le principe de la recherche A^*, en utilisant une estimation du temps de trajet d'un sommet à la destination. La fonction d'estimation retenue considère la distance à vol d'oiseau et la vitesse du mode le plus rapide. Des adaptations plus précises pourraient être effectuées pour tenir compte de l'élimination ou non du mode correspondant à la plus grande vitesse (véhicule personnel dans notre application).

En ce qui concerne l'algorithme bidirectionnelle FB-MQLS, dans les travaux développé nous avons introduits deux fonctions d'estimation correspondant chacune à un sens de parcours du graphe. La condition d'optimalité des connexions doit également être actualisée en fonction de ces estimations. Pour cela, nous avons proposé deux conditions de connexion : une condition exacte et une condition heuristique [66].

Comparaisons expérimentales

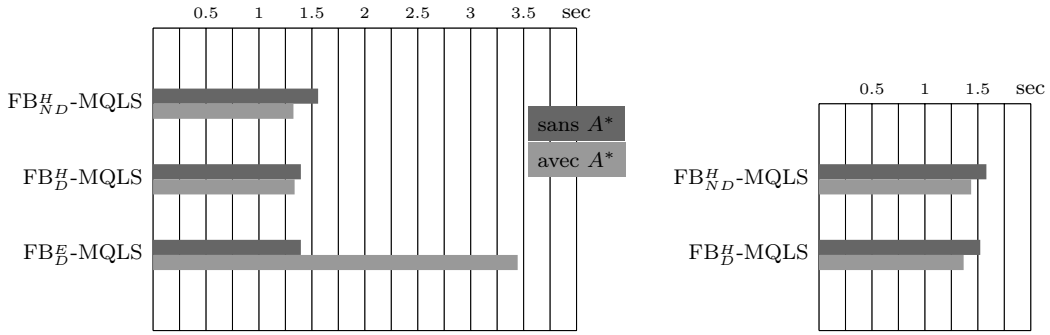
La figure 3.4 présente l'impact de la recherche A^* pour les algorithmes mono-directionnels TLS and MQLS incluant la règle de dominance basée sur les états à la fois sur le graphe statique et sur le graphe dépendant du temps. Nous avons pu noter que l'amélioration due à A^* est très faible. Pour l'algorithme TLS, cette amélioration est de 1.76% sur le graphe statique et de 2.89% sur le graphe dépendant du temps et pour l'algorithme MQLS, cette amélioration est de 2.03% sur le graphe statique et de 1.47% sur le graphe dépendant du temps.



(a) Graphe statique (b) Graphe dépendant du temps

FIGURE 3.4 – Impact de A^* pour les algorithmes mono-directionnels

Les dernières expérimentations ont porté sur l'impact de A^* dans les algorithmes bidirectionels (Figure 3.5). Pour cela, nous avons considéré dans l'algorithme bidirectionnel utilisant l'automate déterministe l'emploi de la condition exacte (algorithme FB_D^H-MQLS) ou de la condition heuristique (algorithme noté FB_{ND}^H-MQLS) pour la détection d'une connexion optimale entre les deux phases de recherche.
Dans le cas du graphe statique, l'utilisation de la condition exacte ralentit fortement le temps de calcul de l'algorithme bidirectionnel (il nécessite environ 2,5 fois plus de temps CPU par rapport à l'algorithme sans A^*). En revanche la condition heuristique permet d'améliorer le temps de calcul de 15% avec l'automate non-déterministe et de 4% avec l'automate déterministe tout en conservant dans nos tests l'ensemble des solutions non-dominées.
Les résultats sont similaires dans le cas du graphe dépendant du temps : la condition exacte ne s'avère pas efficace et la condition heuristique permet d'améliorer les temps de calcul tout en conservant, dans nos tests, l'ensemble des solutions non-dominées. Ces résultats relativement décevants de l'emploi du principe A^* sont en liaison avec ceux de la littérature [8], des adaptations de la fonction d'estimation devraient conduire à de meilleurs résultats.



(a) Graphe statique (b) Graphe dépendant du temps

FIGURE 3.5 – Impact de A^* pour les algorithmes bidirectionnels

3.4 Conclusion

Pour résoudre les problèmes de plus courts chemins multimodaux bi-objectifs avec minimisation du temps de trajet et du nombre de transferts nous avons proposé différents algorithmes à fixation d'étiquettes. Des règles de dominance ont été introduites afin d'élaguer l'exploration, une variante bidirectionnelle ainsi que des adaptations de la recherche A^* ont également été proposées.
Ces algorithmes s'appuient sur une modélisation d'un réseau de transport en graphe multicouches et utilisent un automate à états finis pour représenter les contraintes de viabilité des itinéraires. Nous avons proposé une nouvelle condition de dominance basée sur une relation entre états de cet automate permettant d'élaguer des étiquettes lors de la recherche d'un itinéraire.
Les expérimentations menées ont permis de mettre en évidence l'intérêt de l'algorithme mono-directionnel proposée par rapport à des travaux de la littérature ainsi que l'apport de l'extension de cet algorithme en bidirectionnel. En revanche, l'intégration de la recherche A^* ne conduit pas à de très bons résultats en raison potentiellement de la difficulté de guider la recherche avec l'utilisation des contraintes de viabilité.
Les algorithmes proposés ont permis le développement d'un logiciel industriel de calcul

d'itinéraires multimodaux couplé avec un système d'information géographique.

Chapitre 4

Bilan et Projet de Recherche

4.1 Bilan

Dans ce mémoire nous avons présenté notre contribution sur le développement de méthodes de recherche arborescente pour la résolution de problèmes combinatoires avec un focus particulier sur les problèmes d'ordonnancement à ressources disjonctives et sur les problèmes de calculs d'itinéraires multimodaux bi-objectif.
Les points forts de nos travaux de recherche sont les suivants :

- Proposition d'une nouvelle méthode basée sur le concept de divergences
La méthode de résolution proposée, Climbing Depth-bounded discrepancy Search, est une méthode arborescente incomplète pour la résolution de problèmes d'optimisation combinatoire. Elle est basée sur l'exploration progressive de voisinages, de taille de plus en plus grande, définis via le concept de divergences. L'efficacité de cette méthode est liée à l'intégration de bornes sur la fonction objectif, sur l'utilisation d'heuristiques d'instanciation de bonne qualité et sur des stratégies d'application des divergences à certaines parties de l'espace de recherche.
Des adapations de cette méthode ont été réalisées pour la résolution de problèmes d'ordonnancement flexibles (flowshop hybrides et jobshop flexibles) et les résultats expérimentaux obtenus attestent de l'efficacité de son efficacité par rapport aux méthodes de la littérature.

- Proposition d'une nouvelle heuristique d'instanciation basée sur une pondération des variables
Le mécanisme de pondération de variables que nous avons introduit permet une certaine mémorisation des échecs rencontrés lors du déroulement d'une méthode arborescente et produit ainsi un certain apprentissage des caractéristiques du problème à résoudre. Cette nouvelle heuristique a été intégrée au sein de deux méthodes arborescentes (recherche en profondeur avec retour-arrière chronologique ou recherche à divergences limitée). Son efficacité a été montré pour la résolution de problèmes de statisfaction de contraintes de type car-sequencing, jobshop ou aléatoires.

- Etude des différents paramètres composants une méthode à divergences
Nous avons étudié l'impact de différents paramètres comme le mode de comptage des divergences, l'ordre d'application des divergences (en priorité en haut ou en bas de l'arborescence), l'incrémentation du nombre de divergences, l'heuristique d'instanciation, la propagation de contraintes. Cette étude a permis de développer un ensemble de méthodes à divergences intégrées au sein d'un solveur de contraintes. Les premières expérimentations menées ont mis en évidence l'influence du mode de comptage des divergences (mais aussi

du pas d'incrémentation) en fonction des types de problèmes à résoudre (optimisation ou satisfaction) mais ces premiers résultats restent à confirmer.

- Proposition de nouvelles techniques de propagation de contraintes de ressources
Pour la résolution de problèmes d'ordonnancement à ressources disjonctives soumis à des contraintes temporelles généralisées, nous avons proposé d'étendre certaines techniques de propagation classiques en se basant sur des déductions obtenues sur un graphe de contraintes temporelles. Les propagations étendues proposées les plus efficaces vérifient des propriétés de dominance vis-à-vis des propagations classiques, et leur apport pratique a été mis en évidence dans des méthodes de calculs de bornes appliquées à la résolution de problèmes d'ordonnancement avec durée variable ou de problèmes d'ordonnancement flexibles. Ces techniques de propagation ont par ailleurs été intégrées au sein d'une méthode arborescente de type Branch-and-Bound pour la résolution de problèmes d'ordonnancement avec contraintes de délais et leur efficacité a également été mise en évidence.

- Proposition de nouvelles méthodes pour un problème de plus courts chemins multimodaux bi-objectif
Nous nous sommes intéressés à la résolution d'un problème de calcul d'itinéraires de complexité polynomiale encore peu étudié consistant à déterminer les plus courts chemins mulitmodaux viables minimisant le temps de trajet et le nombre de transfert. Le réseau de transport est représenté par un graphe multi-couches et la modélisation des contraintes de viabilité s'effectue à l'aide d'un automate à états finis. Pour la résolution de ce problème, nous avons proposé de nouveaux algorithmes à fixation d'étiquettes, des conditions de dominance permettant d'éliminer des labels ainsi que des adaptations de la recherche bidirectionnelle et de la recherche orientée. Pour mettre en œuvre une recherche bidirectionnelle, un automate représentant la viabilité d'un chemin pris à l'envers doit également être pris en compte. Les expérimentations réalisés ont montré l'intérêt des algorithmes proposés et en particulier de l'approche bidirectionnelle.

4.2 Projet de recherche

En prenant appui sur ce bilan, nos perspectives de recherche s'organisent en deux axes de recherche. Le premier axe concerne le développement de méthodes de recherche arborescentes génériques pour la résolution de problèmes combinatoires et le second axe prote sur l'étude et la résolution de nouveaux problèmes d'ordonnancement et de transport. Ces différentes perspectives sont précisées dans les points ci-dessous.

4.2.1 Méthodes arborescentes

Les premiers travaux que nous allons développer sont issus directement des résultats que nous avons obtenus sur les méthodes à divergences. Tout d'abord, nous allons poursuivre l'intégration de différents résultats obtenus sur les méthodes à divergences afin de proposer un cadre de résolution basé divergences complet et générique à la fois dans un contexte d'optimisation et de satisfaction.
Plus spécifiquement, des travaux d'intégration des heuristiques à pondération sont à finaliser, des travaux complémentaires sur l'impact du mode de comptage des divergences sont à réaliser et des propositions permettant de généraliser les stratégies de séléction des points de divergences doivent être élaborées.

De plus, des études complémentaires doivent être développées afin de mieux caractériser les propriétés des heuristiques dynamiques d'instanciation et d'étudier l'impact de cette dynamicité sur la redondance de l'exploration. Pour garantir la complétude de la

méthode, les méthodes à divergences implémentées inclues une redondance dans le parcours de l'arborescence pour pallier à l'aspect dynamique des heuristiques d'instanciation. Selon le type d'heuristique dynamique considérée, cette redondance du parcours peut ne pas s'avérer utile. Par ailleurs, la complétude de la méthode n'est pas forcément nécessaire dans un contexte d'optimisation.
Par ailleurs, une étude de l'impact de différentes pistes d'intégration des techniques de propagation de contraintes dans une méthode à divergences est à mener Nous avons pour le moment envisagé une seule piste d'intégration des techniques de propagation dans une méthode à divergences consistant à comptabiliser les divergences de manière dynamique. Cette intégration si elle permet de s'appuyer efectivement sur l'état courant de la méthode de recherche ne prend pas en compte le fait que des valeurs peuvent avoir été supprimées par propagation. Une autre approche d'intégration serait de comptabiliser les divergences de manière statique. Des études comparatives de ces deux approche doit être réalisée et couplée à celles sur les modes de comptage des divergences.
Pour évaluer ces différents travaux de recherche nous nous intéresserons à la résolution de problèmes d'optimisation combinatoire ou de satisfaction de contraintes avec un focus particuliers sur la résolution de problèmes d'ordonnancement ou de transport.

Les travaux présentés par M. Milano et A. Roli en 2002 [116] proposent un formalisme liant les méthodes à divergences et les méthodes de recherche par voisinages (Recuit Simulé, Tabou, ...). En partant à la fois de cette contribution et de nos travaux sur les méthodes à divergences, nous proposons d'exploiter, voire d'étendre, ce formalisme afin enrichir le cadre des méthodes de recherche à voisinages.

4.2.2 Problèmes d'ordonnancement et de transport

Ordonnancement multi-objectif

Nous envisageons d'étendre l'application des méthodes à divergences dans un contexte multi-objectif en raison des caractéristiques des méthodes à divergences permettant la manipulation d'ensembles de solutions pour un nombre de divergences donné. Cette spécificité des méthodes à divergences peut permettre d'obtenir de manière efficace un ensemble de solutions non dominées.
Pour ces travaux, nous nous appuyerons en priorité sur les résultats que nous avons obtenus sur les méthodes à divergences appliquées à la résolution de problèmes d'ordonnancement mono-objectif.
Dans le contexte multi-objectif, nos travaux porteront notamment sur la définition des types de divergences et sur la localisation de points de divergences en fonction des objectifs considérés.

Approches par contraintes de problèmes d'ordonnancement

Afin d'enrichir les mécanismes de propagation de contraintes de ressources, une des pistes que nous explorerons est l'obtention de nouvelles conditions permettant des propagations de type not first/not last ou de type edge finding basées sur les déductions du graphe de contraintes temporelles. En effet, même si l'extension des conditions classiques de type Not First / Not Last ou de type Edge Finding n'est pas directement réalisable en raison du fait que ces règles mettent en jeu des ensembles de tâches et non pas des paires de tâches, nous pensons qu'il serait judicieux de proposer d'autres conditions (de type Not First / Not Last ou de type Edge Finding) en exploitant les résultats des propagations des contraintes temporelles.

De plus, nous travaillerons à l'intégration de l'ensemble de ces nouvelles techniques de propagation de contraintes de ressources au sein de méthodes arborescentes génériques, comme celles à divergences, afin d'obtenir une validation plus poussée de nos propositions.

Ordonnancement dans les systèmes embarqués critiques

Le contexte applicatif visé est celui des problèmes d'ordonnancement dans les systèmes embarqués critiques et ces activités seront développées via un projet LAAS impliquant différents groupes de recherche du laboratoire.
Dans ce contexte, nous aborderons à la fois la problèmatique de l'obtention d'une solution de configuration (ie. une solution à un problème d'ordonnancement et d'affectation) et la problématique de reconfiguration de l'ordonnancement de tâches en cas de pannes (par exemple pannes de processeurs) qui consiste à basculer sur une nouvelle solution d'ordonnancement et d'affectation réalisable.
Pour la résolution de problème d'ordonnancement et d'affectation, le contexte applicatif met en évidence des contraintes temporelles spécifiques (comme la périodicité des tâches ou la variabilité de la durée d'exécution). Nous proposons de partir des travaux menés sur les problèmes de satisfaction de contraintes avec incertitudes et sur ceux que nous avons menés pour la résolution conjointe de problèmes d'ordonnancement et d'affectation afin de proposer de nouvelles méthodes de résolution.
Le problème de reconfiguration peut-être abordé hors-ligne et consiste alors à définir un ensemble de solutions au problème d'ordonnancement et d'affectation posé, suffisament éloignées les unes des autres pour éviter d'être touchées par une même panne et présentant également de bonne qualité de robustesse vis-à-vis d'un ensemble d'aléas. Les approches par contraintes sont une piste privilégiée pour aborder cette problématique de reconfiguration via notamment le raisonnement sur des distances entre solutions.

Problèmes de plus courts-chemins multimodaux multi-objectif

Les travaux sur le développement de méthodes de résolution pour des problèmes de plus courts chemins multimodaux multi-objectifs vont être poursuivis. L'objectif est d'une part de pouvoir adresser des réseaux de transport de grande taille et d'autre part de prendre en compte une grande variété de problèmes multimodaux et muti-objectifs.

Les travaux que nous avons menés ouvrent de nombreuses pistes de recherche. A court terme, des études expérimentales et théoriques doivent être menées pour mieux comprendre les impacts de la recherche A^* sur ce type de problème, pour mesurer l'impact de l'automate de viabilité sur les performances des algorihmes, pour permettre une meilleure prise en compte des caractéristiques temporelles des différents modes afin d'améliorer l'efficacité des algorithmes et pour développer d'autres algorithmes bidirectionnels.

Afin d'étendre la taille des réseaux de transport pouvant être manipulés par nos méthodes, nous nous intéresserons à l'extension de techniques d'accélération développées ces dernières années pour le problème de plus court chemin monomodal afin de les appliquer à la résolution d'un problème de plus courts chemins multimodal bi-objectif de complexité polynomiale (tel que minimisation du temps de trajet et du nombre de changement de modes).
Les travaux menés devront permettre tout d'abord de mieux cerner les apports respectifs de la recherche bidirectionnelle et de la recherche A^* ainsi que leurs interactions avec les contraintes de viabilité sur les modes restreignant certaines possibilités d'exploration dans le graphe, de plus des calculs de bornes peuvent être améliorés en utilisant les informations

sur l'état courant du trajet (et donc sur les possibilités d'extension restantes). Puis des techniques d'accélération plus récentes basées notamment sur des précalculs seront abordées dans un contexte multimodal.

D'autre part, sur la base d'une analyse des différents objectifs pouvant être introduits par la prise en compte d'un réseau multimodal, nous caractériserons différentes familles de problèmes de calcul d'itinéraires multimodaux et multi-objectifs afin de pouvoir proposer des méthodes de résolution adaptées.

Problèmes de tournées de véhicules

Afin d'étendre l'éventail des problèmes combinatoires considérés et en raison des nombreuses applications rencontrées, nous développerons des travaux sur la résolution de problèmes de tournées de véhicules fournissant à la fois un cadre pratique des méthodes développées et permettant leur enrichissement.
Des travaux plus particuliers porteront sur la résolution des problèmes de tournées dans un environnement dynamique. Et nous nous intéresserons plus en détails à la notion de stabilité entre différentes solutions consécutives de tournées. Comme dans le contexte des problèmes de reconfiguration de solution d'ordonnancement, nos travaux porteront plus précisément sur la définition de distance entre solutions et sur le développement de méthodes exploitant cette notion de distance.

Les deux axes de recherche présentés dans ce projet (méthodes de recherche arborescente et résolution de problèmes d'ordonnancement et de transport) s'appuient sur les activités de recherche que nous avons menées jusqu'à présent et sont complémentaires. D'une part les apports que nous ferons au niveau méthodologique contribuera à l'amélioration de la résolution de problèmes combinatoires. D'autre part, l'étude de problèmes d'ordonnancement ou de transport, en prenant en compte soit de nouveau contexte applicatif soit de nouvelles contraintes, permettra de dégager des concepts génériques qui enrichiront les méthodes de résolution.

Bibliographie

[1] J.F. Allen : Maintening knowledge about temporal intervals. *Communications of the ACM*, 26(11):832–843, 1983.

[2] D. Applegate et W. Cook : A computational study of the job-shop scheduling problem. *ORSA Journal on Computing*, 3(2):149–156, 1991.

[3] A. Armando, C. Castellini et E. Giunchiglia : SAT-based procedures for temporal reasoning. *In 5th European Conference on Planning (ECP-99), Durham (UK)*, volume 1809 de *Lecture Notes in Computer Science*, pages 97–108, 1999.

[4] A. Armando, C. Castellini, E. Giunchiglia, M. Idini et M. Maratea : Tsat++ : an open platform for satisfiability modulo theories. 125(3):25–36, 2005.

[5] C. Artigues, MJ. Huguet et P. Lopez : Generalized disjunctive constraint propagation for solving the job shop problem with time lags. *Engineering Applications of Artificial Intelligence*, 24(2):220–231, 2010.

[6] Ph. Baptiste et C. Le Pape : A theoretical and experimental comparison of constraint propagation techniques for disjunctive scheduling. *In Proceedings of the 14th IJCAI, Montreal (Canada)*, 1995.

[7] Ph. Baptiste et C. Le Pape : Edge-Finding constraint propagation algorithms for disjunctive and cumulative scheduling. *In Proceedings of the 15th Workshop of the U.K. Planning Special Interest Group, Liverpool (UK)*, 1996.

[8] C. Barett, K. Bisset, M. Holzer, G. Konjevod, M. Marathe et D. Wagner : Engineering label-constrained shortest-path algorithms. *In 4th International Conference on Algorithmic Aspects in Information and Management, (AAIM 2008), Shanghai (China)*, volume 5034 de *Lecture Notes in Computer Science*, pages 27–37, 2008.

[9] C. Barett, R. Jacob et M. Marathe : Formal-language-constrained path problems. *SIAM Journal on Computing*, 30(3):809–837, 2000.

[10] J.W. Barnes et J.B. Chambers : Flexible job shop scheduling by tabu search. Rapport technique, Graduate Program in Operations and Industrial Engineering, The University of Texas at Austin, Technical Report Series, ORP96-09, 1996. http ://www.cs.utexas.edu/users/jbc/.

[11] J. Beck, P. Prosser et R. Wallace : Variable ordering heuristics show promise. *In Proceedings CP'2004, Toronto (Canada)*, pages 711–715, 2004.

[12] A. Ben Hmida : *Méthodes arborescentes pour la résolution de problèmes d'ordonnancement flexible.* Thèse de doctorat, Faculté des Sciences de Tunis (Tunisie) et Université de Toulouse-INSA (France), 2009.

[13] A. Ben Hmida, M. Haouari, MJ. Huguet et P. Lopez : Discrepancy search for the flexible job shop problem. *Computers & Operations Research*, 37:2192–2201, 2010.

[14] A. Ben Hmida, M. Haouari, MJ. Huguet et P. Lopez : Solving two stage hybrid flow shop using climbing depth-bounded discrepancy search. *Computers & Industrial Engineering*, 60:320–327, 2011.

[15] A. Ben Hmida, M. Haouari, MJ. Huguet et P. Lopez : Discrepancy and backjumping heuristics for flexible job shop scheduling. *In 11th International Workshop on Project Management and Scheduling (PMS'08), Istanbul (Turkey)*, page 4p., April 28-30, 2008.

[16] A. Ben Hmida, MJ. Huguet, P. Lopez et M. Haouari : Solving hybrid flow shop scheduling problems by discrepancy-based methods. *In 10th International Workshop on Project Management and Scheduling (PMS'2006), Poznan (Pologne)*, pages 168–174, April 26-28, 2006.

[17] A. Ben Hmida, MJ. Huguet, P. Lopez et M. Haouari : Climbing depth-bounded discrepancy search for solving flexible job shop scheduling problems. *In 3rd Multidisciplinary International Scheduling Conference : Theory and Applications (MISTA 2007), Paris (France)*, pages 217–224, August 28-31, 2007.

[18] A. Ben Hmida, MJ. Huguet, P. Lopez et M. Haouari : Climbing depth-bounded discrepancy search for solving hybrid flow shop problems. *European Journal of Industrial Engineering*, 1:223–243, July, 2007.

[19] A. Ben Hmida, MJ. Huguet, P. Lopez et M. Haouari : Adaptation of discrepancy-based methods for solving hybrid flow shop problems. *In IEEE International Conference on Service System and Service Management, (SSSM), Troyes (France)*, pages 1120–1125, October 25-27, 2006.

[20] C. Bessiere et JC. Régin : MAC and combined heuristics : two reasons to forsake FC (and CBJ ?) on hard problems. *In Proceedings CP'1996, Cambridge, MA, (USA)*, pages 61–75, 1996.

[21] M. Bielli, A. Boulmakoul et H. Mouncif : Object modelling and path computation for multimodal travel systems. *European Journal of Operational Research*, 175(3):1705–1730, 2006.

[22] S. Boivin, M. Gravel, M. Krajecki et C. Gagné : Résolution du problème de car-sequencing à l'aide d'une approche de type fc. *In Premières Journées Francophones de la Programmation par Contraintes (JFPC'05), Lens (France)*, pages 11–20, Juin 2005.

[23] F. Boussemart, F. Hemery, C. Lecoutre et L. Sais : Boosting systematic search by weighting constraints. *In Proceedings ECAI'2004, Valencia (Spain)*, pages 146–150, 2004.

[24] S.A. Brah et J.L. Hunsucker : Branch and Bound algorithm for the flow shop with multiprocessors. *European Journal of Operational Research*, 51(1):88–99, 1991.

[25] P. Brandimarte : Routing and scheduling in a flexible job shop by tabu search. *Annals of Operations Research*, 41:157–183, 1993.

[26] P. Brucker : Scheduling and constraint propagation. *Discrete Applied Mathematics*, 123:227–256, 2002.

[27] P. Brucker, B. Jurisch et A. Kramer : The job-shop problem and immediate selection. *Annals of Operations Research*, 50:73–114, 1994.

[28] P. Brucker et S. Knust : A linear programming and constraint propagation-based lower bound for the rcpsp. *European Journal of Operational Research*, 127:355–362, 2000.

[29] J. Carlier : *Ordonnancement à contraintes disjonctives.* Thèse de doctorat, Thèse de troisième cycle, Université Paris VI (France), 1975.

[30] J. Carlier et E. Néron : An exact method for solving the multiprocessor flowshop. *RAIRO-Operations Research*, 34:1–25, 2000.

[31] J. Carlier et E. Pinson : A practical use of jackson's preemptive schedule for solving the job-shop problem. *Annals of Operations Research*, 26:269–287, 1990.

[32] J. Carlier et E. Pinson : Adjustment of heads and tails for the job-shop problem. *European Journal of Operational Research*, 78:146–161, 1994.

[33] Y. Caseau et F. Laburthe : Improving CLP scheduling with task intervals. *In Proceedings International Conference on Logic Programming, Santa Margherita (Italy)*, pages 369–383, 1994.

[34] Y. Caseau et F. Laburthe : Improving Branch and Bound for jobshop scheduling with constraint propagation. *In Proceedings Eighth Franco-Japanese - 4th Franco-Chinese Conference on Combinatorics and Computer Science, Brest (France)*, 1995.

[35] A. Caumond, P. Lacomme et N. Tchernev : A memetic algorithm for the job-shop with time-lags. *Computers & Operations Research*, 35(7):2331–2356, 2008.

[36] A. Cesta et A. Oddi : Gaining efficiency and flexibility in the simple temporal problem. *In Proceedings Third International Workshop on Temporal Representation and Reasoning (TIME-96), Los Alamitos (USA)*, pages 45–50, 1996.

[37] I. Chabini : Discrete dynamic shortest path problems in transportation applications : Complexity and algorithms with optimal run time. *Transportation Research Records*, 1645:170–175, 1998.

[38] S. Dauzere-Peres et J. Paulli : An integrated approach for modelling and solving the general multiprocessor job-shop scheduling problem using tabu search. *Annals of Operations Research*, 70:281–306, 1997.

[39] B.C. Dean : *Continuous-Time Dynamic Shortest Path Algorithms.* Thèse de doctorat, Master's thesis, Massachusetts Institute of Technology (USA), 1999.

[40] R. Dechter : Enhancement schemes for constraint processing : backjumping, learning and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.

[41] R. Dechter : Temporal constraint networks. *Artificial Intelligence*, 49:61–91, 1991.

[42] R. Dechter : *Constraint processing.* Morgan Kaufmann, San Francisco, 2003.

[43] R. Dechter et I. Meiri : Experimental evaluation of preprocessing techniques in constraint satisfaction problems. *In Proceedings IJCAI'1989, Detroit, Michigan, USA*, pages 271–277, 1989.

[44] D. Delling : *Engineering and Augmenting Route Planning Algorithms.* Thèse de doctorat, PhD dissertation, Universitat Karlsrhue (Germany), 2009.

[45] D. Delling, T. Pajor et D. Wagner : Accelerating multi-modal route planning by access-nodes. *In Proceedings of the 17th Annual European Symposium on Algorithms (ESA'09), Copenhagen (Danemark)*, volume 5757 de *Lecture Notes in Computer Science*, pages 587–598, 2009.

[46] D. Delling, P. Sanders, D. Schultes et D. Wagner : *Engineering Route Planning Algorithms*, volume 5515 de *Lecture Notes in Computer Science*, pages 117–139. Springer-Verlag, J. Lerner, D. Wagner and K.A. Zwieg (eds) édition, 2009.

[47] S. Demassey, Ch. Artigues et Ph. Michelon : Constraint-propagation-based cutting planes : An application to the resource-constrained project scheduling problem. *INFORMS Journal on Computing*, 17(1):52–65, 2005.

[48] J. Dongarra : Performance of various computers using standard linear equations software. Rapport technique, Computer Science Department, University of Tennessee, Knoxville, Tennessee, 1998.

[49] J. Dongarra : Performance of various computers using standard linear equations software. Rapport technique, Computer Science Department, University of Tennessee, Knoxville, Tennessee, 2009.

[50] U. Dorndorf, T.P. Huy et E. Pesch : *Project Scheduling*, chapitre A survey of interval capacity consistency tests for time- and resource-constrained scheduling. Kluwer Academic Publishers, 1998.

[51] O. Engin et A. Döyen : A new approach to solve hybrid flow shop scheduling problems by artificial immune system. *Future Generation Computer Systems*, 20: 1083–1095, 2004.

[52] J. Erschler, P. Lopez et C. Thuriot : Raisonnement temporel sous contraintes de ressources et problèmes d'ordonnancement. *Revue d'Intelligence Artificielle*, 5(3):7–36, 1991.

[53] P. Esquirol, MJ. Huguet et P. Lopez : Modeling and managing disjunctions in scheduling problems. *Journal of Intelligent Manufacturing*, 6:133–144, 1995.

[54] H. Farenny : *Recherche Heuristiquement Ordonnée dans les graphes d'états*. Masson, 1995.

[55] B. Gacias, C. Artigues et P. Lopez : Parallel machine scheduling with precedence constraints and setup times. *Computers & Operations Research*, 37(12):2141–2151, 2010.

[56] B. Gacias, C. Artigues et P. Lopez : Tree and local search for parallel machine scheduling problems with precedence constraints and setup times. *In Proceedings PMS'2008, Istanbul (Turkey)*, page 4p., April 28-30, 2008.

[57] J. Gao, L. Sun et M. Gen : A hybrid genetic and variable neighborhood descent algorithm for flexible job shop scheduling problems. *Computers & Operations Research*, 35(9):2892–2907, 2008.

[58] M.R. Garey, D.S. Johnson et R. Sethi : The complexity of flow shop and job shop scheduling. *Mathematics of Operations Research*, 1, 1976.

[59] M. Ginsberg : Dynamic Backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

[60] A. V. Goldberg et C. Harrelson : Computing the shortest path : A^* search meets graph theory. *In 16th ACM-SIAM Symposium on Discrete Algorithms (SODA'05), Vancouver (Canada)*, 2005.

[61] A. V. Goldberg, H. Kaplan et R. Werneck : Reach for A^* : Efficient point-to-point shortest path algorithm. *In SIAM Workshop on Algorithms Engineering and Experimentation (ALENEX'06), Miami (USA)*, pages xxx–xxx, 2006.

[62] T. Grabener, A. Berro et Y. Duthen : Calcul d'itinéraire multimodal et multiobjectif : application au transport urbain. *In 11e Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Toulouse*, pages 12–13, 2010.

[63] D. Grimes et R.J. Wallace : Learning to identify global bottlenecks in constraint satisfaction search. *In Proceedings FlAIRS'2007*, pages 592–598, 2007. AAAI Press.

[64] F. Gueye : *Algorithmes de recherche d'itinéraires en transport multimodal*. Thèse de doctorat, Université de Toulouse-INSA (France), 2010.

[65] F. Gueye, C. Artigues, MJ. Huguet, F. Schettini et L. Dezou : Bi-objective multimodal time-dependent shortest viable path algorithms. *In Seven Triennial Symposium on Transportation Analysis, Tromso (Norway)*, page 4p., June 20-24, 2010.

[66] J.N.D. Gupta : Two stage hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 39:359–364, 1988.

[67] P. Hansen et N. Mladenovic : Variable neighborhood search : Principles and applications. *European Journal of Operational Research*, 130:449–467, 2001.

[68] M. Haouari, L. Hidri et A. Gharbi : Optimal scheduling of a two hybrid flow shop. *Mathematical Methods of Operations Research*, 64:107–124, 2006.

[69] M. Haouari et R. M'Hallah : Heuristic algorithms for the two-stage hybrid flowshop problem. *Operations Research Letters*, 21:43–53, 1997.

[70] R. Haralick et G. Elliot : Increasing tree search efficiency for constraint satisfaction problems. *Artificial Intelligence*, 14:263–313, 1980.

[71] W. D. Harvey et M. L. Ginsberg : Limited discrepancy search. *In 14th International Joint Conference on Artificial Intelligence (IJCAI'95)*, volume 1, pages 607–615, 1995.

[72] E. Hebrard : MISTRAL : a constraint satisfaction library. Rapport technique, 2008.

[73] P. Van Hentenryck, H. Simonis et M. Dincbas : Constraint satisfaction using constraint logic programming. *Artificial Intelligence*, 58:113–159, 1992.

[74] MJ. Huguet et J. Erschler : Non-conjunctive graph for constraint modelling and propagation in scheduling problems. Rapport technique, Rapport LAAS n°98189, 19 pages, 1998.

[75] MJ. Huguet et P. Lopez : Combining resource assignment and disjunctive scheduling : a constraint-based approach. Rapport technique, Rapport LAAS n°99288, 16 pages, 1999.

[76] MJ. Huguet et P. Lopez : Extension de mécanismes de propagation de contraintes pour l'ordonnancement. Rapport technique, Rapport LAAS n°05446, 25 pages, 2005.

[77] MJ. Huguet et P. Lopez : An integrated contraint-based model for task scheduling and resource assignement. *In International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'99), Ferrara (Italy)*, page 5p., February 1999.

[78] MJ. Huguet et P. Lopez : Approche par contraintes de problèmes d'ordonnancement de tâches et d'affectation de ressources. *In CIFA 2000 : Première Conférence Internationale Francophone d'Automatique, Lille*, pages 236–241, Juillet 2000.

[79] MJ. Huguet et P. Lopez : Mixed task scheduling and resource allocation problems. *In International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'00), Paderborn (Germany)*, volume 2, pages 71–79, March 2000.

[80] MJ. Huguet, P. Lopez et A. Ben Hmida : A limited discrepancy search method for solving disjunctive scheduling problems with resource flexibility. *In Ninth International Workshop on Project Management and Scheduling (PMS'04), Nancy, (France)*, pages 299–302, April, 2004.

[81] MJ. Huguet, P. Lopez et T. Vidal : Dynamic task sequencing in temporal problems with uncertainty. *In Workshop on On-Line Planning and Scheduling in Sixth International Conference on AI Planning & Scheduling (AIPS'02), Toulouse (France)*, pages 41–48, April 2002.

[82] T. Hulubei et B. O'Sullivan : Search heuristics and heavy-tailed behaviour. *In Proceedings CP'2005*, pages 328–342, 2005.

[83] E. Hurink, B. Jurisch et M. Thole : Tabu search for the job shop scheduling problem with multi-purpose machines. *Operations Research-Spektrum*, 15:205–215, 1994.

[84] J. Hwang et D. G. Mitchell : 2-way vs d-way branching for CSP. *In Proceedings CP'2005, Barcelona (Spain)*, pages 343–357. Springer, 2005.

[85] B. Jurisch : *Scheduling jobs in shops with multi-purpose machines.* Thèse de doctorat, PhD dissertation, Universitat Osnabruck (Germany), 1992.

[86] N. Jussien, R. Debruyne et P. Boizumault : Maintaining arc-consistency within dynamic backtracking. *In Principles and Practice of Constraint Programming (CP 2000), Singapore*, numéro 1894 de Lecture Notes in Computer Science, pages 249–261. Springer-Verlag, 2000.

[87] H. Kaindl et G. Kainz : Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Resarch*, 7:283–317, 1997.

[88] W. Karoui : *Méthodes à divergences pour la résolution de problèmes de satisfaction de contraintes et d'optimisation combinatoire.* Thèse de doctorat, Faculté des Sciences de Tunis (Tunisie) et Université de Toulouse-INSA (France), 2010.

[89] W. Karoui, MJ. Huguet et P. Lopez : Impact des modes de comptage sur les méthodes à base de divergences. *In 10e Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Nancy*, page 2p., 10-12 février 2009.

[90] W. Karoui, MJ. Huguet, P. Lopez et M. Haouari : Limited discrepancy heuristic for scheduling problems with time lags. *In 12th International Workshop on Project Management and Scheduling, PMS2010, Tours (France)*, page 4p., April 26-28, 2010.

[91] W. Karoui, MJ. Huguet, P. Lopez et M. Haouari : Climbing Discrepancy Search for flowshop and jobshop scheduling. *In International Symposium on Combinatorial Optimization, ISCO2010, Hammamet (Tunisia)*, volume 36 de *Electronic Notes in Discrete Mathematics*, pages 821–828, March 24-26 2010.

[92] W. Karoui, MJ. Huguet, P. Lopez et W. Naanaa : Apport de MDS à la résolution de CSP et applications aux problèmes de job shop et de carrés latins. *In 8e Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Grenoble*, pages 20–23, 20-23 Février 2007.

[93] W. Karoui, MJ. Huguet, P. Lopez et W. Naanaa : Amélioration par apprentissage d'une méthode de recherche arborescente. *In 7e Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF), Lille*, page 2p., 6-8 Février 2006.

[94] W. Karoui, MJ. Huguet, P. Lopez et W. Naanaa : Amélioration par apprentissage de la recherche à divergences limitées. *In 1ères Journées Francophones de Programmation par Contraintes (JFPC), Lens*, pages 109–117, 8 - 10 juin 2005.

[95] W. Karoui, MJ. Huguet, P. Lopez et W. Naanaa : YIELDS : a yet improved limited discrepancy search for CSPs. *In 4th Int. Conf. on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'07), Brussels (Belgium),*, volume 4510 de *Lecture Notes in Computer Science*, pages 99–111, May 23-26, 2007.

[96] H. Kautz, E. Horvitz, Y. Ruan, C. Gomes et B. Selman : Dynamic restart policies. *In Proceedings AAAI'2002, Edmonton, Alberta, Canada*, pages 674–681, 2002.

[97] R. Klein et A. Scholl : Computing lower bounds by destructive improvement - an application to resource-constrained project scheduling. *European Journal of Operational Research*, 112:322–346, 1999.

[98] R. Kolisch et A. Sprecher : Psplib - a project scheduling library. *European Journal of Operational Research*, 96:205–216, 1996.

[99] R.E. Korf : Improved limited discrepancy search. *In AAAI-96*, pages 286–291, 1996.

[100] Ph. Laborie : Algorithms for propagating resource constraints in AI planning and scheduling : Existing approaches and new results. *Artificial Intelligence*, 143:151–188, 2003.

[101] C. Lecoutre, L. Sais, S. Tabary et V. Vidal : Last conflict-based reasonning. *In Proceedings ECAI'2006*, pages 133–137, 2006.

[102] C. Lecoutre, L. Sais, S. Tabary et V. Vidal : Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173:1592–1614, 2009.

[103] C.Y. Lee et G.L. Vairaktarakis : Minimizing makespan in hybrid flow-shop. *Operations Research Letters*, 16:149–158, 1994.

[104] P. Lopez : *Approche énergétique pour l'ordonnancement de tâches sous contraintes de temps et de ressources.* Thèse de doctorat, Thèse de doctorat, Université Paul Sabatier, Toulouse (France), 1991.

[105] P. Lopez et P. Esquirol : Consistency enforcing in scheduling : A general formulation based on energetic reasoning. *In Proceedings PMS'96*, pages 155–158, 1996.

[106] A. Lozano et G. Storchi : Shortest viable path algorithm in multimodal networks. *Transportation Research Part A*, 35:225–241, 2001.

[107] M. Luby, A. Sinclair et D. Zuckerman : Optimal speedup of las vegas algorithms. *Information Processing Letters*, 47(4):173–180, 1993.

[108] A. K. Mackworth et E. C. Freuder : The complexity of some polynomial network consistency algorithms for constraint satisfaction problems. *Artificial Intelligence*, 25(1):65–74, 1985.

[109] A.K. Mackworth : Consistency in networks of relations. *Artificial Intelligence*, 8:99–118, 1977.

[110] E.Q.V. Martins : On a multicriteria shortest path problem. *European Journal of Operational Research*, 16:236–245, 1984.

[111] M. Mastrolilli et L.M. Gambardella : Effective neighbourhood functions for the flexible job shop problem. *Journal of Scheduling*, 3:3–20, 2000.

[112] I. Meiri : Combining qualitative and quantitative constraints in temporal reasoning. *Artificial Intelligence*, 87:343–385, 1996.

[113] M. Milano et A. Roli : On the relation between complete and incomplete search : an informal discussion. *In CPAIOR'02*, pages 237–250, 2002.

[114] R. Mohr et T.C. Henderson : Arc-consistency and path-consistency revisited. *Artificial Intelligence*, 28:225–233, 1986.

[115] U. Montanari : Networks of constraints : Fundamental properties and applications to picture processing. *Information Sciences*, 7:95–132, 1974.

[116] P. Morris, N. Muscettola et T. Vidal : Dynamic control of plans with temporal uncertainty. *In 17th International Joint Conference on A.I. (IJCAI-01)*, pages 494–499, 2001.

[117] G. Nannicini : *Point to Point Shortest Paths on Dynamic Time-Dependent Road Networks*. Thèse de doctorat, Ecole Polytechnique (France), 2009.

[118] G. Nannicini, D. Delling, D. Schultes et L. Liberti : Bidirectional A^* search for time-dependent fast paths. *In 7th International Workshop on Experimental Algorithms (WEA 2008),Provincetown MA (USA)*, volume 5038 de *Lecture Notes in Computer Science*, pages 334–346, 2009.

[119] E. Neron : *Du flow-shop hybride aux problèmes cumulatifs*. Thèse de doctorat, Thèse de doctorat, Université de Technologie de Compiègne (France), 1999.

[120] E. Neron, Ph. Baptiste et J.N.D. Gupta : Solving hybrid flow shop problem using energetic reasoning and global operations. *Omega*, 29(6):501–511, 2001.

[121] T.A.J. Nicholson : Finding the shortest route between two points in a network. *The Computer Journal*, 9(3):275–280, 1966.

[122] W. Nuijten : *Time and resource constrained scheduling - A constraint satisfaction approach*. Thèse de doctorat, PhD Thesis, Eindhoven University of Technology (Netherlands), 1994.

[123] A. Oddi et A. Cesta : Incremental forward checking for the disjunctive temporal problem. *In Proceedings European Conference on Artificial Intelligence (ECAI-2000), Berlin, (Germany)*, pages 108–112, 2000.

[124] T. Pajor : *Multi-Modal Route Planning*. Thèse de doctorat, PhD dissertation, Universitat Karlsrhue (Germany), 2009.

[125] S. Pallottino et M. G. Scutella : *Shortest path algorithms in transportation models : Classical and innovative aspects*, pages 245–281. Kluwer Academic Publishers, p. marcotte and s. nguyen (eds) édition, 1998. Equilibrium and Advanced Transportation Modelling.

[126] F. Pezzella, G. Morganti et G. Ciaschetti : A genetic algorithm for the flexible job-shop scheduling problem. *Computers & Operations Research*, 35(10):3202–3212, 2008.

[127] L. R. Planken : Incrementally solving the STP by enforcing partial path consistency. *In* Ruth Aylett et Ivan Petillot, éditeurs : *27th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2008)*, pages 87–94, 2008.

[128] L.R. Planken, M. M. de Weerdt et N. Yorke-Smith : Incrementally solving STNs by enforcing partial path consistency. *In Proceedings of the Twentieth International Conference on Automated Planning and Scheduling (ICAPS-10), Menlo Park, CA, (USA)*, pages 129–136. AAAI Press, 2010.

[129] M.-C. Portmann, A. Vignier, D. Dardihac et D. Dezalay : Branch and Bound crossed with G.A. to solve hybrid flow shops. *International Journal of Production Economics*, 43:27–37, 1992.

[130] C. Pradel : Développement et test de méthodes à divergences au sein du solveur de contraintes mistral. Rapport technique, Université de Toulouse (INSA), Juin 2010. stage M2R et Ingénieur.

[131] N. Prcovic : Quelques variantes de LDS. *In Journées Nationales sur les Problèmes NP-Complets (JNPC'02), Nice (France)*, pages 195–208, 2002.

[132] P. Prosser : Hybrid algorithms for the constraint satisfaction problem. *Computational Intelligence*, 9(3):268–299, 1993.

[133] P. Prosser et C. Unsworth : LDS : testing the hypothesis. Rapport technique, Tech Report Number TR-2008-273, Dept of Computing Science, University of Glasgow, 2008.

[134] P. Refalo : Impact-based search strategies for constraint programming. *In Proceedings CP'2004*, pages 557–571, 2004.

[135] F. Schulz : *Timetable Information and Shortest Paths.* Thèse de doctorat, PhD dissertation, Universitat Karlsrhue (Germany), 2005.

[136] E. Schwalb et R. Dechter : Processing disjunctions in temporal constraint networks. *Artificial Intelligence*, 93:29–61, 1997.

[137] H.D. Sherali, et C. Jeenanunta : The approach dependent, time-dependent, label constrained shortest path problem. *Networks*, 48(2):57–67, 2006.

[138] H.D. Sherali, A.G. Hobeika et S. Kangwalklai : Time-dependent, label-constrained shortest path problems with applications. *Transportation Science*, 37(3):278–293, 2003.

[139] B. Smith : Succeed-first or fail-first : A case study in variable and value ordering heuristics. *In 3rd Conference on the Practical Applications of Constraint Technology (PACT'97), London (UK)*, pages 321–330, 1997.

[140] B.M. Smith : Suceed-first or fail-first : a case study in variable and value ordering. Rapport technique, Report 96.26, University of Leeds, 1996.

[141] C. Solnon, V. D. Cung, A. Nguyen et C. Artigues : The car sequencing problem : overview of state-of-the-art methods and industrial case-study of the roadef'05 challenge problem. *European Journal of Operational Research*, 191(3):912–927, 2008.

[142] K. Stergiou et M. Koubarakis : Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence*, 120:81–117, 2000.

[143] P. Torres et P. Lopez : On not-first/not-last conditions in disjunctive scheduling. *European Journal of Operational Research*, 127(2):332–343, 2000.

[144] I. Tsamardinos et M.E. Pollack : Efficient solution techniques for disjunctive temporal reasoning problems. *Artificial Intelligence*, 151:43–89, 2003.

[145] E. Tsang : *Foundations of constraint satisfaction.* Academic Press Ltd, London, 1993.

[146] A. Vignier : *Contribution à la résolution des problèmes d'ordonnancement de type monogamme, multimachines (flow shop hybride).* Thèse de doctorat, Université de Tours (France), 1997.

[147] M. Vilain et H. Kautz : Constraint propagation for temporal reasoing. *In Proceedings AAAI'1986, Philadelphia, PA*, pages 377–382, 1986.

[148] T. Walsh : Depth-bounded discrepancy search. *In IJCAI-97*, pages 1388–1395, 1997.

[149] T. Walsh : Search in a small world. *In Proceedings IJCAI'1999*, pages 1172–1177, 1999.

[150] A. Ziliaskopoulos et W. Wardell : An intermodal optimum path algorithm for multimodal networks with dynamic arc travel times and switching delays. *European Journal of Operational Research*, 125(3):486–502, 2000.

Titre

Méthodes de recherche arborescentes. Application à la résolution de problèmes d'ordonnancement et de calculs d'itinéraires multimodaux.

Résumé

Les travaux présentés dans ce document traitent de méthodes arborescentes pour la résolution de problèmes combinatoires d'optimisation ou de décision. Le premier chapitre présente les contributions que nous avons apportées pour les méthodes de résolution dites " à divergences ". Ces contributions concernent les modes de comptage des divergences pour les problèmes à variables discrètes, le développement d'une heuristique dynamique à pondération de variables, ainsi que, dans un contexte d'optimisation, l'utilisation de bornes ou d'heuristiques pour la sélection des points de divergences. Ces différentes contributions sont illustrées sur des problèmes d'ordonnancement ou sur des problèmes de satisfaction de contraintes. Le deuxième chapitre traite de propagation de contraintes pour la résolution de problèmes d'ordonnancement disjonctifs en présence de contraintes temporelles généralisées. Des extensions de méthodes de propagation de contraintes efficaces dans ce contexte sont proposées et des applications à la résolution de différents problèmes d'ordonnancement sont également présentées. Le troisième chapitre s'intéresse à un problème de calcul d'itinéraires point à point sur des réseaux de transport multi-modaux. La prise en compte de la multi-modalité fait surgir à la fois de nouvelles contraintes permettant d'exprimer si la séquence de modes d'un itinéraire conduit ou non à une solution admissible, mais aussi de nouveaux objectifs comme la minimisation du nombre de changement de modes. Le problème étudié (minimisation du temps de trajet et du nombre de transferts) est polynomial et différentes variantes basées sur le principe de l'algorithme de Dijkstra sont présentées et évaluées sur un cas réel.

Mot-clefs

Méthodes à divergences, Propagation de contraintes temporelles, Propagation de contraintes de ressources, Ordonnancement flexible, Ordonnancement avec délais minimum et maximum, Plus courts chemins multi-modaux

---

Title

Tree search methods. Application for the solving of scheduling problems and multimodal itineraries problems

Abstract

The works presented in this document deal with tree search methods for the resolution of combinatorial optimization problems or of decision problems. The first chapter presents the contributions which we used for the solving methods "with discrepancies". These contributions concern the modes of counting the discrepancies for problems with discrete variables, the development of a dynamic heuristic with weighting variables, as well as, in a context of optimization, the use of bounds or heuristics for the selection of the points of discrepancies. These various contributions are illustrated by scheduling problems or by constraint satisfaction problems. The second chapter deals with constraint propagation for disjunctive scheduling problems involving generalized temporal constraints. Extensions of propagation techniques, effective in the presence of generalized temporal constraints, are proposed and applications in the resolution of various scheduling problems are also presented. The third chapter is interested in calculating point to point itineraries on multi-modal transport networks. The consideration of the multi-modality induces, at once, new constraints to express whether a sequence of modes of an itinerary does or does not lead to an admissible solution, but also induces new objectives such as the minimization of the number of transfers. The studied problem (minimization of the travel time and of the number of transfers) is polynomial and various variants based on the principle of the Dijkstra's algorithm are presented and evaluated by a real case.

Keywords

Discrepancy search, Temporal Constraint Propagation, Resource Constraint Propagation, Flexible Scheduling, Scheduling with Time-Lags Constraints, Multi-Modal Shortest Paths