

Compositions et hybridations pour l'optimisation combinatoire appliquée

Sophie Demassey

▶ **To cite this version:**

Sophie Demassey. Compositions et hybridations pour l'optimisation combinatoire appliquée. Recherche opérationnelle [math.OC]. Université côte d'azur, 2017. tel-01569853

HAL Id: tel-01569853

https://minesparis-psl.hal.science/tel-01569853

Submitted on 27 Jul 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



Distributed under a Creative Commons Attribution 4.0 International License

UNIVERSITÉ DE NICE - SOPHIA ANTIPOLIS

École Doctorale STIC
Sciences et Technologies de l'Information et de la Communication

Compositions et hybridations pour l'optimisation combinatoire appliquée

Mémoire de Synthèse présenté à l'Université de Nice Sophia Antipolis

pour l'obtention d'une

Habilitation à Diriger les Recherches

Spécialité Informatique

par

Sophie Demassey

soutenue le 30 juin 2017

Devant la commission d'examen composée de :

Christian Artigues	Directeur de recherche, LAAS Toulouse (rapporteur)
Nicolas Beldiceanu	Professeur, Mines Nantes
Dominique Feillet	Professeur, Mines St-Étienne (rapporteur)
Christelle Guéret	Professeur, Université d'Angers (rapporteur)
Nadia Maïzi	Professeur, Mines ParisTech (invitée)
Jean-Charles Régin	Professeur, Université de Nice Sophia Antipolis

Table des matières

1 Introduction **1**
- 1.1 Décomposition hybride . . . 1
- 1.2 Paradigmes déclaratifs pour la décomposition . . . 4
  - 1.2.1 Optimisation déclarative versus impérative . . . 4
  - 1.2.2 Méthodes de décomposition en programmation mathématique . . . 5
  - 1.2.3 Contraintes globales en Programmation Par Contraintes . . . 9
- 1.3 Organisation du document . . . 11

2 Contributions choisies **13**
- 2.1 Scalabilité et flexibilité dans BtrPlace . . . 14
  - 2.1.1 Contrainte de vector-packing paramétrable . . . 14
  - 2.1.2 Ordonnancement consommateur/producteur . . . 15
  - 2.1.3 Contraintes utilisateur . . . 16
  - 2.1.4 Réparation des conflits . . . 17
  - 2.1.5 Contributions . . . 18
- 2.2 Décomposition et recherche opérationnelle pour Interval-Amongs . . . 21
  - 2.2.1 Preuve de complexité . . . 21
  - 2.2.2 Décompositions pour le filtrage . . . 23
  - 2.2.3 Contributions . . . 24
- 2.3 Génération de colonnes hybride pour la planification de personnel . . . 25
  - 2.3.1 Décomposition couverture-ordonnancement . . . 26
  - 2.3.2 Modèle de contraintes en ordonnancement riche . . . 28
  - 2.3.3 Contrainte de langage et d'optimisation pour les règles d'ordonnancement 28
  - 2.3.4 Applications . . . 30
  - 2.3.5 Branchement orienté contrainte . . . 34
  - 2.3.6 Contributions . . . 35
- 2.4 MultiCostRegular : hybridation pour le filtrage et la modélisation . . . 36
  - 2.4.1 Filtrage par relaxation lagrangienne . . . 37
  - 2.4.2 Agrégation et relaxation automatiques de règles de séquencement . . . 40
  - 2.4.3 Contributions . . . 47

3 Réflexions **51**
- 3.1 Les contraintes globales : essence de la décomposition et de l'hybridation . . . 51
  - 3.1.1 Contraintes spécifiques, classification et hybridation . . . 52
  - 3.1.2 Filtrage générique et flexibilité . . . 55
- 3.2 Optimisation par contraintes . . . 60
  - 3.2.1 Contraintes globales d'optimisation . . . 61
  - 3.2.2 Au-delà du filtrage . . . 64
  - 3.2.3 Intégration à une méthode d'optimisation . . . 67

4 Perspectives **71**

A Curriculum Vitae **75**

Chapitre 1

Introduction

Mes travaux ont un objectif pratique commun : résoudre de manière efficace et flexible des problèmes d'optimisation fortement combinatoires, concrets, de grandes tailles. Ma principale contribution à cet objectif est de développer des solutions basées sur la *décomposition structurelle ou sémantique* des problèmes et *hybridant des techniques* issues de différents *paradigmes déclaratifs* de l'optimisation discrète. Ce document référence certains travaux consécutifs à ma thèse [Demo3] qui participent de cet objectif. Ce chapitre présente l'approche générale retenue et ses motivations, puis précise l'objet des travaux et l'organisation du document.

1.1 Décomposition hybride

Les problèmes d'optimisation appliqués reposent le plus souvent sur une combinaison de sous-structures élémentaires, définies par des sous-ensembles de contraintes portant sur des sous-ensembles de variables. La *résolution par décomposition* (sous-entendue *structurelle*) désigne ici l'ensemble des méthodes coordonnant plusieurs algorithmes associés à ces sous-structures, par opposition aux méthodes monolithiques où l'intégralité du problème est spécifié en bloc en entrée d'un algorithme « boîte noire ». Selon le degré d'intrication, il existe plusieurs avantages à exploiter la décomposition structurelle d'un problème pour sa résolution :

Scalable. [1] La décomposition s'inscrit dans le principe « divide-and-conquer » : quand la complexité algorithmique est supra-linéaire, il est plus rapide de résoudre deux petits sous-problèmes plutôt qu'un grand. Quand le problème est NP-difficile, il reste souvent plus rapide de combiner les solutions d'une série de sous-problèmes, même non disjoints. Les méthodes de résolution de problèmes combinatoires exploitent pratiquement toutes ce principe : procédures arborescentes (branch-and-bound en programmation mathématique, backtracking en programmation par contraintes, DPLL en programmation logique), recherches locales, algorithmes de graphes s'appuient, par exemple, sur la décomposition spatiale de l'ensemble des solutions. Imprimer une décomposition structurelle peut s'avérer plus efficace, ou simplement complémentaire, en communiquant à l'algorithme de résolution une information – les sous-structures du problème – qu'il ne peut généralement pas inférer.

Modulaire. La décomposition permet d'appliquer à chaque sous-problème la technique de résolution la plus appropriée. La variété des paradigmes, techniques et algorithmes existants pour résoudre les problèmes combinatoires résulte de (et confirme) l'ignorance d'une solution universelle. Plus la solution est générique, moins elle est à même de résoudre certains problèmes spécifiques. Ainsi, appliquer une méthode monolithique à

1. Dans ce document, on emploie à dessein l'anglicisme *scalabilité* au lieu de *passage à l'échelle*

un problème hétérogène impose de choisir un cadre de résolution conservatif – le plus large possible pour traiter tous les sous-problèmes – et potentiellement peu performant – ignorant les caractéristiques de certains sous-problèmes. On désigne par *décomposition hybride* les solutions qui combinent des techniques issues de différents paradigmes.

Robuste. La résolution modulaire est robuste à une évolution de la définition du problème quand celle-ci touche les sous-problèmes de façon indépendante. C'est une qualité essentielle en pratique où les problèmes varient dans le temps, et d'une application à l'autre. Certaines formes de décomposition sont même robustes à des variations dynamiques, c'est à dire en cours de résolution.

Réutilisable. Aussi du point de vue du développeur, la décomposition autorise la réutilisabilité des composants, comme les mêmes sous-structures élémentaires se retrouvent dans de nombreuses problématiques.

Fonctionnelle. Les sous-structures reflètent la décomposition sémantique d'un problème. Du point de vue de l'utilisateur, la décomposition autorise une spécification plus haut niveau du problème, orientée sémantique plutôt que méthode, à l'inverse des solutions monolithiques, déclaratives ou impératives, qui peuvent nécessiter un encodage artificiel des différentes sous-structures afin de les imbriquer au sein d'un modèle unique.

Au-delà de la révérée rapidité d'exécution, ces attributs me semblent essentiels pour promouvoir les méthodes avancées de l'optimisation combinatoire dans leur fonction propre : la résolution de problèmes techniques et sociétaux concrets.

*
* *

Cependant, ces attributs ne s'obtiennent pas seuls et la mise en œuvre de méthodes de décomposition présente certains verrous :

— Par définition, la décomposition structurelle d'un problème est propre au problème. Traiter un problème par décomposition nécessite une analyse fine pour identifier les composants – i.e. les sous-problèmes et les outils de résolution associés – et le schéma global de collaboration ; ces choix étant interdépendants et dépendants des critères de performance retenus. Il est nécessaire par exemple de décider du grain de décomposition, afin de balancer la taille des sous-problèmes avec la rapidité d'exécution : plus riche, le sous-problème offrira un meilleur potentiel d'inférence au détriment d'un traitement plus lourd.
— La décomposition nécessite de sélectionner une combinaison d'algorithmes parmi la multitude existante. Dans le cadre de la décomposition hybride, l'éventail des possibilités est redoublé et la tâche de sélection d'autant plus ardue. On s'autorise en effet à piocher dans différents paradigmes de résolution, composés de techniques dissemblables et animés par des communautés qui tendent, de manière opportune, à s'ignorer.
— Quand les algorithmes sont inexistants ou inadaptés au problème et au schéma de collaboration, il est nécessaire de les redévelopper selon des critères particuliers, comme par exemple, l'incrémentalité quand l'algorithme doit être appelé de manière itérative dans le schéma de résolution. De plus, s'il a le souci de réutilisabilité des composants,

le développeur doit également jongler entre efficacité et généricité et trouver la balance entre ces buts opposés.

— Enfin, les codes génériques de schémas collaboratifs n'ont pas les niveaux de standardisation et d'efficacité offerts par les solutions monolithiques actuelles. La mise en œuvre pratique d'un tel schéma requiert donc son implémentation complète ou, au mieux, l'ajustement des nombreux paramètres d'exécution. La complexité d'implémentation des méthodes de décomposition hybrides est d'autant plus grande qu'elle fait communiquer des composants logiciels qui ont rarement été conçus pour coexister.

*
* *

Mes travaux, notamment ceux relatés dans ce manuscrit [2], s'attachent à lever ces différents verrous. Leurs apports, effectifs ou potentiels, sont d'ordre théorique (analyse et conception d'algorithmes), méthodologique (validation de méthodes) et appliqué (résolution de problèmes) :

Applications. Pour divers problèmes d'optimisation (allocation, ordonnancement, packing) issus de divers contextes fonctionnels (planification de personnel, logistique ferroviaire ou portuaire, gestion des centres de données), nous avons élaboré des méthodes de décomposition adaptées, dans un double objectif. Il s'agit, d'une part, de produire de nouvelles solutions performantes pour les problématiques étudiées. Dans le terme « performance », nous intégrons les notions d'efficacité (qualité des solutions et rapidité d'exécution) et de flexibilité (robustesse aux variations du problème). D'autre part, ces travaux se veulent preuves de concept, pour susciter l'intérêt de la décomposition et de l'hybridation dans des domaines qui n'y sont habituellement pas sensibilisés. C'est le cas, par exemple, de l'emploi des automates pour la modélisation des règles de travail ou de la programmation par contraintes pour la gestion flexible des centres de données.

Développement de composants. Pour mettre en œuvre ces solutions, nous avons conçu et développé des composants algorithmiques et logiciels pour résoudre mais aussi pour aider à modéliser des sous-problèmes. Bien qu'élaboré dans le cadre d'une solution dédiée à un problème, chaque composant a vocation à être réutilisé dans d'autres contextes. Dans nos développements, nous avons donc généralement considéré conjointement efficacité et réutilisabilité. Par exemple, nous avons systématiquement implémenté et mis à disposition toutes les contraintes globales que nous avons conçues dans des solveurs de contraintes open-source. Comme précédemment, au-delà de proposer une solution à un problème donné, certains de ces travaux viennent valider une approche méthodologique, comme employer des méthodes de programmation linéaire pour le filtrage de contraintes globales.

Mise en œuvre. L'efficacité des solutions repose en grande partie sur la communication entre les composants logiciels. Les prototypes que nous avons développé ont valeur aussi de démonstrateurs de faisabilité de l'hybridation de solveurs. Nous avons également employé des techniques d'accélération, originales pour certaines comme l'agrégation dynamique de contraintes en génération de colonnes. Comme il n'existe générale-

2. réalisés dans le cadre de différentes collaborations, d'où l'usage du « nous » ci-après

ment pas de caractérisation théorique de leur impact, seule une validation expérimentale sur une grande variété de cas d'études permet d'arbitrer la valeur de ces techniques. Nos travaux participent donc de cette validation.

1.2 Paradigmes déclaratifs pour la décomposition

Comme leur nom ne l'indique pas, les *méthodes de décomposition* désignent moins la manière de décomposer un problème que le *schéma de collaboration* des algorithmes appliqués aux sous-problèmes pour dériver une solution globale. Il en existe dans tout paradigme de l'optimisation combinatoire : par exemple, la décomposition arborescente de graphes ou les metaheuristiques multi-étapes. Mes travaux se sont principalement articulés autour de la notion de décomposition dans les paradigmes déclaratifs de la programmation mathématique et de la programmation par contraintes.

1.2.1 Optimisation déclarative versus impérative

La programmation mathématique et la programmation par contraintes, aussi la programmation logique et ses extensions (SAT, SMT, ASP), offrent une approche déclarative à la résolution des problèmes d'optimisation combinatoire : le problème est spécifié par *modèle* dans un langage standardisé plus ou moins haut niveau, en entrée d'un *solveur* – un moteur de résolution exacte ou approchée générique – plus ou moins paramétrable. Ces approches se distinguent des algorithmes de type metaheuristique [3] ou programmation dynamique par exemple, qui doivent, à chaque mise en œuvre sur un problème donné, être re-développés en fonction de l'encodage particulier des solutions et des contraintes.

L'avantage de l'optimisation déclarative est donc de s'abstraire de tout ou partie du développement algorithmique, en ayant recours à l'algorithme implémenté dans un solveur choisi, en fonction de la taxinomie du modèle, parmi la profusion des solutions logicielles existantes, propriétaires ou libres. Elle offre ainsi un gain en temps de développement, sécurité, reproductibilité et flexibilité. Elle permet aussi à l'utilisateur de se focaliser sur la spécification du problème en adoptant un niveau d'abstraction au plus proche de la sémantique.

Les solveurs, de programmation mathématique ou logique notamment, ont progressé en performance de manière spectaculaire ces dernières années, mais ils ne sont évidemment pas à même de résoudre tout problème combinatoire en temps raisonnable. Le choix du modèle est déterminant sur la performance du solveur mais il n'est généralement pas suffisant.

L'approche par décomposition se présente en alternative pour dépasser ces limites, mais, comme pour les metaheuristiques, la mise en œuvre requiert alors un développement spécifique au problème traité. Par exemple, un solveur de programmation par contraintes qui est, comme on le verra ensuite, ce qui se rapproche le plus d'une solution générique de décomposition, ne peut être directement appliqué à un modèle décomposé donné si les différents composants n'y sont pas tous implémentés (sous forme de *contraintes globales*). Les algorithmes par défaut de ces solveurs sont aussi souvent limités quant à leurs performances et nécessitent alors des adaptations profondes, plus que du simple paramétrage, pour atteindre un

3. hormis de nouvelles hyper-heuristiques (voir [Bur+09]) qui ambitionnent de s'auto-composer

certain niveau d'efficacité sur un problème donné. En programmation mathématique, des solveurs génériques de *génération de colonnes* existent (Coin-OR BCP [RL01], Symphony [RG05] et SCIP GCG [GL10] par exemple) mais présentent une limitation plus forte encore à savoir qu'il ne s'agit que de frameworks dans lesquels il est nécessaire d'implémenter l'algorithme de résolution des sous-problèmes (sous forme de générateurs de variables et de contraintes). Néanmoins, les solutions génériques de décomposition sont un domaine de recherche très actif actuellement dans les différentes communautés (voir par exemple [Puc+11]). Ainsi, avec la regain d'intérêt récent pour la décomposition de Benders (voir par exemple [FLS16]), notamment sur les problèmes stochastiques (voir par exemple [Bol+16]), des frameworks commerciaux ont été avancés : le langage de modélisation AIMMS offre un cadre concis à l'utilisateur pour spécifier sa décomposition ; Cplex dans sa dernière version (12.7) supporte la décomposition de Benders non-hybride (de programmation linéaire) et applique, par défaut, la décomposition classique (variables entières dans le programme maître, fractionnaires dans le programme esclave) dans le cas des programmes linéaires en nombres entiers.

Nous avons tenté de maintenir un maximum de déclarativité dans toutes les méthodes que nous avons développées pour bénéficier des avantages notamment en termes de flexibilité, et ce de deux manières. D'une part, nous avons travaillé sur des outils d'aide à la modélisation en exploitant l'expressivité d'autres paradigmes, à savoir les graphes et les automates, et en les couplant à des solutions d'optimisation. D'autre part, nous avons appliqué des schémas de collaboration propres à la programmation mathématique ou par contraintes, de sorte que les composants reposent sur un modèle et sa résolution par un solveur générique approprié. Nous présentons brièvement ces schémas dans les sections suivantes.

1.2.2 Méthodes de décomposition en programmation mathématique

En programmation mathématique, un modèle dit *étendu* présente une structure par blocs dans la matrice des coefficients, résultant du fait que les variables de décision n'entrent en jeu que dans un nombre restreint de contraintes, traduisant une division hiérarchique, temporelle, géographique, ou logique du problème. Relâcher les contraintes ou fixer les variables couplantes permet de séparer le modèle en plusieurs modèles mathématiques indépendants. La solution obtenue par juxtaposition n'est alors pas assurée d'être réalisable (dans le premier cas) ou optimale (dans le second) mais peut être la base d'une approche heuristique rapide.

Cette résolution en phases séquentielles est une approche par décomposition élémentaire. Elle est suffisante quand le problème est séparable mais pas si les composants sont en interaction. Les méthodes de résolution génériques pour des décompositions primales (génération de coupes [Ben62]), duales (génération de colonnes [FF58] et relaxation lagrangienne [Geo74]), mixtes (cross-decomposition [Van83]) ou spécialisées (bi-level [IG98], rolling-horizon [BPR96]) constituent des schémas de collaboration entrelacée ou intégrée. Elles procèdent par résolution itérative de sous-problèmes *esclaves*, générés dynamiquement et commandés par un *programme maître* qui assure la convergence de la méthode vers une solution optimale sans jamais considérer la globalité des contraintes ou variables relâchées.

Ces méthodes existent de longue date mais elles n'ont donné lieu, et seulement pour certaines, à des implémentations nombreuses sur des applications concrètes que plus tardivement. Bien que génériques, ces schémas sont en effet complexes à instancier. Il a fallu des travaux

d'application pionniers pour provoquer leur diffusion dans des domaines applicatifs devenus privilégiés (typiquement, la génération de colonnes pour le transport [DDS92] ou la décomposition de Benders pour l'optimisation stochastique [FL82]). La mise en en œuvre efficace de ces méthodes a également attendu la conception plus récente d'ingrédients algorithmiques supplémentaires (par exemple, la stabilisation [Du +99] pour remédier aux lenteurs de convergence dues aux dégénérescences).

Initialement, ces méthodes ont souvent été présentées comme une alternative à la relaxation continue pour calculer des bornes de meilleure qualité aux programmes linéaires en nombres entiers (PLNE), la borne du dual lagrangien étant toujours au moins aussi bonne que la relaxation continue. Elles sont toujours utilisées de la sorte pour la résolution des problèmes les plus difficiles. Elles sont aussi privilégiées sur les problèmes de grandes tailles pour leur mode de résolution dynamique. En effet, les méthodes de décomposition appliquées à un problème donné ne considèrent, à tout moment, qu'une partie d'un modèle étendu du problème, tandis que les solveurs monolithiques standards de type *branch-and-cut* nécessitent en entrée un modèle éventuellement *compact*, mais complet. Malgré leur capacité en termes de scalabilité, ces solveurs clé-en-main s'avèrent inutilisables quand le modèle complet est trop large.

Le lecteur pourra se référer à [RG10] par exemple pour une description de ces approches. Dans nos travaux, nous avons employé deux méthodes de décomposition parmi les plus usitées : la relaxation lagrangienne et la génération de colonnes. Nous donnons ci-après une brève définition de ces méthodes dans le contexte restreint de la PLNE, les principes et le vocabulaire associés, utiles à la compréhension de nos contributions.

1.2.2.1 Relaxation lagrangienne

Principe. La *relaxation lagrangienne* [Geo74] consiste à *dualiser* un ensemble de contraintes d'un modèle mathématique (dit *problème primal*) ; c'est à dire, à les relâcher et à en pénaliser leur violation dans l'objectif suivant une pondération donnée (les *multiplicateurs lagrangiens*). Pour un problème de minimisation, elle fournit une borne inférieure de l'optimum :

$$z^\lambda = \min_{x \in X} \left(cx + \lambda(d - Ex)\right) \leq \min_{x \in X} \left(cx \mid Ex \geq d\right), \quad \lambda \geq 0. \tag{P_λ}$$

Dans le contexte de la PLNE[4], X dénote ici un ensemble de solutions discrètes dans un polyèdre défini par des inégalités linéaires. Le problème du *dual lagrangien* consiste à déterminer la pondération associée à la relaxation lagrangienne donnant la meilleure borne (maximale) :

$$u = \max_{\lambda \geq 0} z^\lambda. \tag{L}$$

Méthodes de résolution. Il s'agit de maximiser alors une fonction $\lambda \mapsto z^\lambda$ concave linéaire par morceaux : l'intuition est que, sauf aux points de rupture, une solution optimale $x^\lambda \in X$ de la relaxation lagrangienne (P_λ) reste optimale dans un voisinage de λ. La fonction z^λ coïncide avec le plan $\lambda \mapsto cx^\lambda + \lambda(d - Ex^\lambda)$ dans ce voisinage et se situe en-dessous ailleurs. Vu ainsi, le dual lagrangien est souvent résolu par l'algorithme du sous-gradient ou ses variantes : la

4. La relaxation lagrangienne s'établit dans un contexte plus général, non linéaire ou continu.

recherche des multiplicateurs optimaux est réalisée en faisant évoluer λ, itérativement de pas en pas, le long de ces plans supports (i.e. dans la direction du sous-gradient $d - Ex^{\lambda}$).

Le dual lagrangien peut également être formulé comme un programme linéaire étendu, sur un ensemble implicite de contraintes :

$$u = \max_{y,\lambda \geq 0}(y \mid y \leq cx + \lambda(d - Ex), \forall x \in X). \tag{D}$$

Comme au plus $|\lambda| + 1$ contraintes, correspondant à une base S de X (ou aux plans supports de la fonction concave ci-dessus), suffisent à sa définition, ce programme linéaire peut être résolu en un nombre fini d'étapes par génération progressive des contraintes suivant la *méthode de Kelley* [Kel60] : à chaque itération $k \geq 1$, on cherche une nouvelle contrainte séparant l'ensemble réalisable de (D) de la solution (y^k, λ^k) optimale du programme linéaire restreint à k contraintes, appelé le *programme maître* ; il s'agit donc de résoudre un *sous-problème* consistant à déterminer un élément $x \in X$ tel que $y^k > cx + \lambda^k(d - Ex)$. S'il existe, la contrainte associée appelée *coupe*, est ajoutée au programme restreint. Sinon, la solution duale (y^k, λ^k) est réalisable et donc optimale pour (D). L'instabilité de la séquence des solutions duales calculées – due à la *dégénérescence* du programme maître, quand il possède une infinité de solutions optimales – et la taille croissante du programme linéaire à chaque itération, sont causes de la lenteur réputée de cette approche. Des variantes, les *méthodes des faisceaux* notamment, résolvent un programme maître de taille contrôlée, mais rendu plus complexe suivant diverses techniques de stabilisation dans le but d'accélérer la convergence.

Quelque soit l'approche (sous-gradient, faisceaux, génération de contraintes), la résolution du dual lagrangien consiste à faire varier la pondération selon la direction donnée par un problème maître, et résoudre à chaque itération la relaxation lagrangienne associée.

Bornes, solutions et filtrage. Par dualité forte sur le programme maître restreint à une base $\{x^S, s \in S\}$ de X, on obtient une formulation étendue primale $(\widetilde{P})$ exprimant la convexification partielle des contraintes (non dualisées) du programme original.

$$\begin{aligned} u &= \max_{y,\lambda \geq 0}(y \mid y \leq cx^s + \lambda(d - Ex^s), \forall s \in S) \\ &= \min_{\theta \geq 0}(\sum_s cx^s\theta_s \mid \sum_s \theta_s = 1, \sum_s (d - Ex^s)\theta_s \geq 0) \\ &= \min_{x \geq 0}(cx \mid Ex \geq d,\ x \in conv(X)). \end{aligned}$$

Cette expression permet de caractériser la valeur de la borne du dual lagrangien : celle-ci est toujours au moins aussi bonne que la borne de la relaxation continue.

L'approche lagrangienne est avantageuse quand un modèle présente un ensemble de contraintes couplantes ou compliquantes, quand les sous-problèmes lagrangiens résultant de la dualisation de ces contraintes sont faciles à résoudre. Trop faciles et la borne du dual lagrangien n'est pas meilleure que celle de la relaxation continue : notamment si la formulation de X est *idéale* (le polyèdre coïncide avec l'enveloppe convexe des éléments discrets qu'il contient). Mais même en cela, le dual lagrangien peut être plus rapide à résoudre que la relaxation continue. C'est le cas par exemple pour la formulation étendue du Traveling Salesman Problem (TSP)

dont la relaxation continue, contenant un nombre exponentiel de contraintes de sous-tours, ne peut être formulée (et résolue) directement.

Autrement, la borne du dual lagrangien peut être strictement meilleure que la borne de relaxation continue, et l'approche lagrangienne est ainsi souvent employée pour bénéficier de la qualité de cette borne. Dans de rares cas, les optimaux des problèmes primal et dual coïncident. Dans le cas contraire, la relaxation lagrangienne peut être embarquée dans un algorithme exact de branch-and-bound avec une stratégie de branchement adaptée : à chaque noeud, il s'agit de résoudre une ou un ensemble de contraintes dualisées et violées par la solution de relaxation lagrangienne courante, en veillant à ne pas modifier la structure des sous-problèmes lagrangiens.

Plus fréquemment encore, la relaxation lagrangienne est employée de manière heuristique couplée à une recherche locale autour des solutions des sous-problèmes lagrangiens pour recouvrir leur faisabilité vis à vis des contraintes dualisées.

Finalement, la solution du dual lagrangien (L), les multiplicateurs optimaux λ, et la solution du primal convexifié $(\widetilde{P})$, sont sources d'information sur l'optimum primal et peuvent être exploités de différentes manières. La technique de *variable fixing* permet notamment d'identifier des valeurs d'affectation des variables qui n'appartiennent à aucune solution optimale. On l'illustre ici dans le cas simple où les variables sont binaires ($X \subseteq \{0,1\}^n$). Si z^* dénote l'optimum primal et $\bar{z}$ une borne supérieure connue, alors toute solution réalisable x de valeur inférieure à cette borne satisfait la relation suivante :

$$f^\lambda(x) = \lambda d + (c - \lambda E)x \leq cx < \bar{z}, \qquad \forall \lambda \geq 0.$$

La fonction f^λ atteint son minimum sur l'ensemble des vecteurs binaires $\{0,1\}^n$ au point e défini par composantes par :

$$e_i = \begin{cases} 0 & \text{si } c_i - \lambda E_i \geq 0, \\ 1 & \text{si } c_i - \lambda E_i < 0. \end{cases}$$

En testant les valeurs opposées des composantes de e une à une, on peut déduire par contradiction celles n'entrant dans aucune solution optimale x du problème :

$$\begin{array}{l} \text{si } c_i - \lambda E_i < 0 \text{ et } f^\lambda(e) - (c_i - \lambda E_i) \geq \bar{z} \text{ alors } x_i \neq 0 \text{ (ou } x_i = 1), \\ \text{si } c_i - \lambda E_i > 0 \text{ et } f^\lambda(e) + (c_i - \lambda E_i) \geq \bar{z} \text{ alors } x_i \neq 1 \text{ (ou } x_i = 0). \end{array}$$

Ces tests peuvent être effectués à chaque itération de la résolution du dual lagrangien, mais ils nécessitent une bonne estimation par excès $\bar{z}$ de l'optimum pour se déclencher. C'est cette propriété, en plus du calcul de la borne inférieure, dont nous faisons usage dans l'algorithme de filtrage de la contrainte globale `multicost-regular` en section 2.4.

1.2.2.2 Génération de colonnes

La relaxation lagrangienne est une approche de décomposition à laquelle s'appliquent différentes approches de résolution ; la *génération (progressive) de colonnes* désigne à l'inverse une approche de résolution applicable à des programmes linéaires naturellement décomposés, étendus avec un grand nombre de variables, ou bien obtenus par le principe de *décomposition de Dantzig-Wolfe* [DW60] sur un modèle linéaire compact :

Dans cette variante de l'algorithme primal du simplexe, seule une partie des variables (les colonnes du tableau) est exprimée ; les autres sont considérées hors base et fixées à zéro dans les solutions basiques calculées :

$$z^k = \min\{\sum_{j\in J} c_j x_j \mid \sum_{j\in J} a_{ij}x_j \geq b_i \forall i \in I, x_j \geq 0 \ \forall j \in J, x_j = 0 \forall\ j \in J_k\} \ \text{ avec } J_k \subseteq J \qquad (M^k)$$

À certaines itérations de l'algorithme, une nouvelle variable basique entrante (de coût réduit négatif) est générée à la volée, par la résolution d'un sous-problème (S^k), et ajoutée au modèle linéaire, le programme maître (M^k).

$$J^{k+1} = J^k \cup J' \ \text{ avec } J' \subseteq \{j \in J \mid c_j - \sum_{i\in I} y_i^k a_{ij} > 0\} \qquad (S^k)$$

avec y^k une solution duale de (M^k). La recherche de colonnes entrantes est équivalente à la recherche de contraintes violées dans le dual du problème global :

$$u = \max\{\sum_{i\in I} y_i b_i \mid \sum_{i\in I} y_i a_{ij} \leq c_j \ \forall j \in J, y_i \geq 0 \ \forall i \in I\} \qquad (D)$$

La décomposition de Dantzig-Wolfe est en fait duale au modèle linéaire du dual lagrangien et la génération de colonnes coïncide avec la méthode de Kelley appliqué à ce modèle.

Pour un problème d'optimisation combinatoire, à variables discrètes, la génération de colonnes produit la solution optimale d'une relaxation continue pour un modèle étendu. L'intérêt est que cette solution a un coût souvent bien meilleur que l'optimum de la relaxation continue du modèle compact correspondant. Elle peut être intégrée dans un branch-and-bound, appelé alors *branch-and-price* [Bar+98], où la génération de colonnes est appelée à chaque nœud de l'arbre de recherche. Cette méthode de résolution exacte est souvent coûteuse et elle nécessite une stratégie de branchement adaptée de façon à ne pas modifier la structure des sous-problèmes. Une méthode alternative de résolution approchée consiste à ne pas générer de nouvelles colonnes pendant le parcours de l'arbre de recherche.

Un traitement plus complet de l'approche par génération de colonnes est proposé dans [LD05].

1.2.3 Contraintes globales en Programmation Par Contraintes

La décomposition sémantique est l'essence même de la programmation par contraintes. Un problème y est spécifié comme une conjonction de contraintes sur un ensemble de variables. Chaque contrainte est associée à un algorithme de filtrage capable de détecter des instantiations des variables incohérentes (ou inconsistantes) avec sa définition, puis de supprimer ces valeurs du domaine de définition des variables. Les algorithmes de filtrage procèdent ainsi de manière indépendante mais coordonnée par un algorithme de propagation, communiquant via le domaine des variables, pour tenter d'inférer des inconsistances relatives à la conjonction de plusieurs contraintes. Les contraintes sont donc les briques élémentaires dans le schéma de décomposition que propose la programmation par contraintes.

En capturant la sous-structure des problèmes, les contraintes globales font l'originalité de l'approche de résolution par programmation par contraintes et sa puissance en termes d'expressivité, de flexibilité, d'efficacité ; en encapsulant des algorithmes dédiés communiquant

par propagation sur le domaine des variables, elles offrent un cadre transparent d'hybridation permettant de combiner les techniques les plus appropriées à chaque sous-structure. D'autres l'ont soigneusement écrit ; je renvoie donc le lecteur aux essais de Régin [Rég04 ; Rég11] et Beldiceanu [Belo3 ; BCR], par exemple, pour un exposé plus complet sur le sujet. Nous rappelons ici les définitions des notions employées dans ce document.

Définition 1 (Réseau de contraintes) *Un modèle de programmation par contraintes ou* réseau de contraintes *est défini par un ensemble de* variables $X = \{X_1, \ldots, X_n\}$, *chaque variable* X_i *prenant ses valeurs dans un ensemble discret* D_i, *son* domaine, *et un ensemble de* contraintes $\mathcal{C}$. *Une contrainte est une relation portant sur un sous-ensemble des variables, définie de manière implicite ou explicite par l'ensemble des combinaisons de valeurs autorisées pour ses variables. Une instanciation (affectation) des variables* satisfait *la contrainte si la combinaison des valeurs affectées appartient à cet ensemble. Une solution du réseau de contraintes est une instanciation complète des variables* X, *chacune à une valeur de son domaine, qui satisfait l'ensemble des contraintes.*

L'*algorithme de filtrage* ou *propagateur* associé à une contrainte est un algorithme chargé d'identifier les instanciations d'une variable à une valeur de son domaine qui ne possèdent pas de *support* dans la contrainte, i.e. qui ne peuvent être étendues à une instanciation complète satisfaisant la contrainte. L'algorithme est alors chargé de supprimer (*filtrer*) ces valeurs *incohérentes* ou *inconsistantes* du domaine de la variable. Le propagateur assure la *consistance d'arc* – il est *complet* – s'il est capable d'identifier et de supprimer toutes les valeurs inconsistantes des domaines des variables. Pour les variables *bornées*, i.e. dont les domaines sont maintenus comme des intervalles de valeurs discrètes, la *consistance aux bornes* est un niveau de filtrage suffisant qui ne considère – pour le filtrage et dans les supports – que les valeurs des bornes des intervalles.

La résolution d'un réseau de contraintes s'effectue classiquement par l'algorithme de *backtracking* : un moteur de propagation appelé à chaque nœud d'un arbre de recherche. Le moteur de propagation appelle tour à tour les propagateurs des contraintes jusqu'à un point fixe au moment duquel plus aucune réduction de domaines n'est inférée. Quand tous les domaines sont réduits à des singletons, l'instanciation correspondante est solution du problème ; si le domaine d'une variable est vide alors le réseau de contraintes est globalement inconsistant. Autrement, une *heuristique de recherche* ou *stratégie de branchement* choisit une variable à instancier à une valeur de son domaine ; l'instanciation est posée, ouvrant ainsi un nouveau *nœud* d'un arbre de recherche et le moteur de propagation est appelé à nouveau jusqu'au point fixe afin de propager cette nouvelle réduction de domaine. Si le domaine d'une variable devient vide, la dernière décision de branchement prise est invalidée. On effectue alors un *backtrack* : l'état des domaines des variables dans le nœud parent est restauré, la valeur invalidée est supprimée du domaine de la variable ; puis le processus est relancé : propagation, branchement et/ou backtrack.

Pour un problème d'optimisation, l'une des variables, appelée Z, porte la valeur de coût à minimiser par exemple. L'algorithme de backtracking est modifié de sorte, qu'à chaque solution réalisable trouvée, le coût z^* de la solution (l'*incumbent*) est enregistré ; une nouvelle contrainte $Z < z^*$ est ajoutée au réseau et le parcours de l'arbre de recherche se poursuit par un backtrack, ou s'arrête à la racine.

1.3 Organisation du document

L'ensemble de mes travaux, depuis ma thèse en 2003, a été réalisé au travers de diverses collaborations. En plus des collaborations extérieures, j'ai été associée à des équipes de recherche aux diverses orientations : programmation mathématique et ordonnancement au Laboratoire Informatique d'Avignon, programmation par contraintes à l'École Polytechnique de Montréal et aux Mines de Nantes, optimisation des systèmes énergétiques au CMA de Mines de Paris. Mes travaux couvrent diverses applications : ordonnancement de projet, placement 2D/3D, gestion de centre de données, planification de personnel, opération et dimensionnement des réseaux de distribution d'eau et des micro-grids, optimisation ferroviaire, distribution des hubs en logistique. Ils empruntent et mixent diverses techniques : programmation linéaire/non-linéaire/en nombres entiers, génération de coupes/colonnes, relaxation lagrangienne, programmation par contraintes, chemins et flots dans les graphes, automates.

Plutôt que les égrener, je me concentre, dans ce manuscrit, sur quelques questions centrales, communes de près ou de loin à tous ces travaux : la composition/décomposition de modèles (1) et l'hybridation de techniques (2) pour augmenter la flexibilité et l'ergonomie des solveurs (3), leur scalabilité et rapidité (4). J'ai ainsi consigné dans le premier chapitre quatre de mes études parmi les plus significatives : le modèle de contraintes du gestionnaire de ressources dans les centre de données BtrPlace, la contrainte globale `interval-amongs`, la génération de colonnes hybride par programmation par contraintes pour la planification de personnel, et la contrainte automate d'optimisation `multicost-regular` pour la planification de personnel sur-contraint. Chaque étude est décrite de manière concise – je donne les références aux codes et aux publications dont elles sont tirées – et accompagnée d'un récapitulatif de la nature des apports.

Je qualifie ces études de significatives, non pas qu'elles soient toutes impactanctes, mais dans le sens où elles apportent, ou ont apporté au moment de leur publication, des contributions originales et où elles illustrent au mieux mon propos. Il s'agit également de contributions plus personnelles, réalisées comme principale investigatrice ou à mon initiative (pour la dernière extraite de la thèse de Julien Menana que j'ai encadrée) au sein d'une collaboration.

Dans le second chapitre, je propose une réflexion transverse et plus générale, nourrie notamment, mais pas seulement, de ces travaux. Elle s'articule en deux parties. La première est consacrée à la programmation par contraintes vue comme un outil pratique d'implémentation de la décomposition et de l'hybridation via les contraintes globales. Je distingue, même si la frontière est floue, contraintes spécifiques et contraintes universelles et devise de l'intérêt ou des difficultés qu'elles présentent en terme d'efficacité et de flexibilité. La seconde partie est consacrée plus précisément au critère d'optimalité et les façons de mieux l'appréhender dans les approches de programmation par contraintes grâce à la décomposition et l'hybridation : de manière interne avec les contraintes globales d'optimisation dont le filtrage s'inspirent généralement des méthodes de recherche opérationnelle et avec les heuristiques de recherche, ou de manière externe en intégrant le module de programmation par contraintes dans une méthode de décomposition de la programmation mathématique ou dans une recherche locale.

J'évoque enfin mes perspectives de recherche avec des études que j'ai débuté plus récemment : persévérer dans l'hybridation des techniques mais me concentrer sur les applications dans le domaine de l'énergie/climat, et répondre à deux questions, qui sont à la fois prégnantes

dans ce domaine et attachées au principe de décomposition, et que sont la prise en compte des incertitudes des données et le couplage des échelles de temps pour un traitement plus réaliste des problématiques d'optimisation combinatoire de court à très long terme.

Chapitre 2

Contributions choisies

Ce chapitre présente quatre de mes études les plus pertinentes, et leurs contributions méthodologiques à différents contextes applicatifs :

1. le modèle scalable et adaptatif de contraintes de BtrPlace, un gestionnaire de ressources dans les centres de données ;
2. la recomposition d'une conjonction de contraintes à l'aide d'outils de la recherche opérationnelle et son application à une contrainte globale de cardinalité particulière `interval-among`s identifiée dans des problèmes de localisation ;
3. une approche de génération de colonnes hybride pour la planification de personnel où les règles de travail impactant le séquencement possible des tâches sont modélisées dans le sous-problème de programmation par contraintes de manière agrégée par une contrainte automate d'optimisation `cost-regular` ;
4. une relaxation lagrangienne appliquée au filtrage de `multicost-regular`, la généralisation multi-objectif de `cost-regular` avec des vecteurs de coûts combinant également des règles de cardinalité et des règles souples, ainsi qu'un outil d'aide à la formulation et à l'agrégation de ces règles appliqués à des problèmes de planification de personnel sur-contraints.

J'ai choisi de présenter ces quatres études car elles illustrent mes principaux sujets de de recherche :

— composition/décomposition : du bon dimensionnement des contraintes globales (2,3,4) ; relâcher une contrainte globale coûteuse en la re-composant (2,3,4) ; appliquer les méthodes de décomposition de la programmation mathématique (3,4) ;
— hybridation : appliquer des outils de la recherche opérationnelle pour concevoir des algorithmes de filtrage (2,3,4) ; de l'intérêt des contraintes globales d'optimisation (3,4) ;
— flexibilité et ergonomie : composition automatique de modèles (1,4) ; des contraintes spécifiques (1,2) aux contraintes universelles (2,3,4) ;
— scalabilité et rapidité (1,2,3,4).

Les quatre sections ci-après présentent un résumé plus ou moins bref de ces quatre études, en les recentrant sur ces quatre sujets. Ces travaux ont tous été détaillés dans des publications et leurs implémentations sont librement accessibles. Les références sont indiquées. Je discute également à et en quoi ces travaux ont contribué en marge des quatres sujets centraux, qui eux, sont développés dans le chapitre suivant.

2.1 Scalabilité et flexibilité dans BtrPlace

Je contribue depuis sa création en 2009 à la solution open-source de gestion opérationnelle des centres de données BtrPlace [btr]. Elle implémente, au-dessus du solveur de contraintes Choco, un ordonnanceur responsable, à intervalles réguliers, du placement et de la migration de machines virtuelles (VM) sur les machines physiques du centre de données. Le problème de placement y est naturellement modélisé comme un problème de Vector Packing [Gar+76], où il s'agit de répartir un ensemble d'objets (les applications ou machines virtuelles (VM)) requérant différentes ressources (CPU, mémoire, etc.) sur un ensemble de conteneurs (les serveurs ou machines physiques (PM)) offrant ces ressources cumulatives en quantités limités. Ce contexte applicatif présente deux particularités. D'une part, les instances sont le plus souvent (mais pas toujours) faciles car peu chargées. En contrepartie, des instances variées et larges sont considérées, de 2 à plusieurs dizaines de milliers de serveurs, et de 1 à plusieurs dizaines de VM par serveur. Cette particularité exige une implémentation réfléchie du modèle de contraintes. L'ordonnanceur se doit également d'être réactif, et la résolution, donc, rapide. D'autre part, l'ordonnanceur doit pouvoir être spécialisé à la volée pour prendre en compte les besoins courants ou instantanés relatifs à l'administration du centre de données, et les besoins spécifiques d'exécution des applications soumises par les clients.

BtrPlace est conçue dans ce double objectif de scalabilité et de flexibilité. J'ai participé à cet objectif, avec Fabien Hermenier concepteur et principal développeur de BtrPlace, à travers plusieurs contributions dont les quatres dernières qui sont décrites ci-dessous : l'étude théorique inédite du problème composé du (re)placement et de la migration, la conception du modèle d'optimisation sous contraintes correspondant, la conception d'heuristiques pour sa résolution, le développement d'une contrainte globale de vector-packing paramétrable en fonction du ratio difficulté/taille de l'instance, la conception d'une contrainte globale d'ordonnancement producteur/consommateur spécifique au problème de migration, la modélisation d'une vingtaine de *contraintes utilisateurs*. Ces travaux ont été en partie publiés dans [HDL11a ; HDL11b ; DHK15] et le code source est disponible à `https://github.com/btrplace/scheduler`.

2.1.1 Contrainte de vector-packing paramétrable

Le problème de placement sous-jacent à BtrPlace est un problème de vector-packing :

Définition 2 (placement de VMs) *Il s'agit d'affecter un ensemble $\mathcal{V}$ de VMs (les objets) à un ensemble $\mathcal{P}$ de PMs (les conteneurs) en fonction d'un ensemble $\mathcal{R}$ de ressources disponibles (les dimensions). Chaque VM $v \in \mathcal{V}$ requiert pour son exécution une quantité w_{vr} de chaque ressource $r \in \mathcal{R}$. Chaque PM $p \in \mathcal{P}$ a une capacité c_{pr} en chaque ressource $r \in \mathcal{R}$. L'affectation $M : \mathcal{V} \rightarrow \mathcal{P}$ est réalisable si les contraintes de ressources sont satisfaites :*

$$\sum_{v \in M^{-1}(p)} w_{vr} \leq c_{pr} \quad \forall p \in \mathcal{P}, r \in \mathcal{R}.$$

Ces conditions pourraient être modélisées au moyen d'une contrainte globale `bin-packing` pour chaque ressource. Une telle contrainte est disponible dans Choco où elle met en oeuvre [1]

1. du moins dans Choco 2.0

plusieurs algorithmes de filtrage proposés par Shaw [Sha04]. Dans BtrPlace, nous avons modélisé ces conditions en une contrainte globale unique `vector-packing` qui met en oeuvre les mêmes principes de filtrage, mais dont les algorithmes ont été re-développés dans un souci de scalabilité.

Comme pour `bin-packing` [Sha04], la contrainte `vector-packing` repose sur des variables d'affectation des objets aux conteneurs ($x_v \in \mathcal{P}$ désigne le conteneur où est affecté la VM v) et elle introduit des variables $l_{pr} \in [0, c_{pr}]$ indiquant la charge totale des objets affectés au conteneur p dans la dimension r. Le filtrage associe deux propagateurs en boucle jusqu'au point fixe : la propagation des sommes globales $\sum_p l_{pr} = \sum_v w_{vr}$ et la propagation des *knapsack* $\sum_{v|x_v=p} w_{vr} = l_{pr}$ sur chaque conteneur p.

Dans `vector-packing`, les propagateurs sont itérés sur les différentes dimensions $r \in \mathcal{R}$. Les dimensions étant indépendantes, elle réalise alors la même propagation qu'un système de $|\mathcal{R}|$ contraintes `bin-packing` mais elle fait l'économie des structures de données internes rendondantes et du temps de calcul lié au mécanisme de propagation. Pour des instances de très grande taille, cela a un impact non négligeable sur les temps de résolution car le nombre d'évènements de propagation est divisé par le nombre de dimensions.

Une optimisation plus complexe à implémenter et à maintenir et visant une meilleure scalabilité repose sur la réalisation d'une propagation événementielle incrémentale, i.e. un réveil limité des propagateurs en fonction de l'évènement reçu sur le domaine d'une ou plusieurs variables. Elle s'accompagne de l'emploi de structures backtrackables pour la mémorisation de calculs, dont une implémentation du parcours des conteneurs en tas (heap) de manière à accéder directement aux conteneurs de plus grande capacité résiduelle et à éviter le parcours des autres pour le premier propagateur., ainsi que d'une gestion fine des boucles sur les objets, les conteneurs et les dimensions. Cette optimisation limite fortement le nombre d'opérations inutiles, d'autant plus quand les nombres de conteneurs et d'objets dépassent le millier.

Enfin, notre implémentation modulaire permet à l'utilisateur de débrancher le coûteux propagateur de knapsack quand le nombre d'objets augmente. Le défaut de propagation est alors souvent compensé par le gain en temps de calcul. En effet, ce propagateur se déclenche rarement dans de nombreuses instances pour lesquelles la taille des items est faible comparée à la taille des conteneurs. De plus, le critère d'optimisation considéré par défaut, minimisant la durée de reconfiguration, tend à déployer les VMs sur l'ensemble des serveurs et donc à diminuer la charge moyenne de ces derniers.

Ainsi, ce code, comparé à l'implémentation originale de la contrainte `bin-packing` de Choco 2, avait divisé les temps de résolution [2] par 10 sur le jeu de test de BtrPlace (de 500 à 2000 serveurs et de 1000 à 10000 VMs), bien que triplant le nombre de backtracks.

2.1.2 Ordonnancement consommateur/producteur

En plus de calculer un nouveau placement, BtrPlace doit planifier les différentes actions de reconfiguration nécessaires à passer du placement courant au nouveau placement calculé. Les actions de démarrage et d'extinction des VMs et des PMs ainsi que les actions de transfert des VMs ayant des durées non nulles (et généralement proportionnelles à la mémoire utilisée), il

2. backtracking tronqué à la première instance trouvée

s'agit d'ordonnancer les différentes actions de manière à ce que la configuration reste viable durant la reconfiguration. Typiquement, les VMs hébergées sur une PM à éteindre doivent être transférées avant l'extinction de la PM, ou encore une VM ne peut démarrer sur une PM qu'une fois que les ressources suffisantes soient disponibles, éventuellement après éteindre ou transférer une ou plusieurs VMs actuellement hébergées sur la PM. L'objectif par défaut de BtrPlace consiste à minimiser la somme des durées des actions de reconfiguration. Cet objectif traduit en effet la volonté de minimiser la période d'instabilité du système. Il porte sur le problème couplé du placement et de la reconfiguration puisque la durée du plan de reconfiguration est évidemment dépendant du placement.

Parmi les actions de reconfiguration permises sur les VMs, BtrPlace traite deux types d'actions de transfert entre PMs : la migration à froid et la migration à chaud [Cla+05]. Dans le second cas, il s'agit d'une opération continue durant laquelle la VM poursuit son exécution sur la PM d'origine durant le transfert de ces données vers la PM destination. La PM d'origine est alors libérée lorsque le transfert est complété et la VM démarrée sur la PM destination. L'occupation simultanée des ressources des deux PMs donne un modèle d'ordonnancement original.

Pour traiter le problème de reconfiguration, nous avons implémenté une contrainte globale portant sur l'ensemble des PMs, des VMs et des dimensions, et où chaque VM est modélisée par deux tâches, l'une consommatrice de ressources (sur le serveur destination), l'autre productrice (sur le serveur origine). Cette contrainte est une alternative spécifique et suffisante par rapport à une conjonction de contraintes `cumulatives` pour chaque dimension, trop génériques et coûteuses. Elle est munie d'un propagateur basique incomplet mais qui néanmoins domine la propagation d'une conjonction de contraintes sur l'ensemble des VMs. Cet algorithme est aussi délicat comme il encapsule tous les cas particuliers inhérents, tels par exemple, le cas d'une VM dont la demande en ressources change sans qu'elle ne soit migrée. Comme pour la contrainte `vector-packing`, un soin a été apporté à l'implémentation pour assurer la scalabilité de cette contrainte d'ordonnancement.

2.1.3 Contraintes utilisateur

L'ordonnanceur de BtrPlace est axé sur la liberté de spécialisation : le problème de base peut être augmenté dynamiquement de contraintes utilisateur spécifiées via l'API du système. Celles-ci sont alors automatiquement transcrites dans le modèle Choco avant la prochaine résolution. Ces contraintes traduisent différents besoins ou préférences en terme de placement exprimés à la fois par les administrateurs du centre de données et par les clients qui soumettent les VMs. Un administrateur requiert, par exemple, de façon permanente d'isoler les VMs d'administration des VMs clients sur des PMs distinctes pour une question de sécurité, ou bien, de façon temporaire, de libérer une certaine PM en prévision d'une opération de maintenance. Un client peut lui exiger de manière contractuelle, par exemple, d'héberger l'ensemble de ses VMs sur des PMs proches en terme de latence réseau pour améliorer l'intercommunication, ou encore d'héberger l'ensemble de ses VMs replicas sur des PMs toutes distinctes pour une question de tolérance aux pannes.

BtrPlace propose une vingtaine de contraintes de ce type, issues de systèmes publics comme Amazon EC2 ou de systèmes privés. Ces contraintes peuvent être exprimées par l'utilisateur

dans un langage haut niveau à travers l'API ; elles sont alors traduites en une ou plusieurs contraintes globales ajoutées, avec éventuellement de nouvelles variables, au modèle central sous Choco.

J'ai, en particulier, travaillé sur cette modélisation. Dans une première approche [HDL11a], nous invoquions des contraintes sur des variables ensemblistes, mais les avons abandonnées dans un second temps pour des raisons de performance. La contrainte *lonely* de l'API par exemple permet d'isoler un groupe de VMs donné sur des serveurs propres, de manière à ce qu'aucune VM hors de ce groupe ne puisse être hébergé sur ces mêmes serveurs. Cette contrainte est utile aux administrateurs du centre de données pour isoler les VMs de service, qui gèrent le centre, des VMs des clients. Nous l'avons, dans un premier temps, modélisée au moyen d'une contrainte `disjoint` sur les ensembles de serveurs hébergeant les VMs. Nous l'avons substituée ensuite [HDL11b] par une contrainte ad-hoc (intégrée par la suite dans Choco) assurant la disjonction de deux listes de variables entières, les variables d'affectation du groupe de VMs d'une part, et des autres VMs d'autre part.

2.1.4 Réparation des conflits

En plus d'algorithmes de filtrage, nous avons équipé chaque contrainte utilisateur d'un algorithme (un *checker*) vérifiant si elle est satisfaite par une affectation complète donnée et retournant, dans le cas contraire, un sous-ensemble de VMs candidates à migrer pour résoudre la violation. Cet algorithme est invoqué en prétraitement de la résolution dans le but de réduire la taille du modèle. En effet, chaque résolution dans BtrPlace consiste à réparer la configuration courante de manière à minimiser l'effort de reconfiguration. Pour une raison de scalabilité et de performance, la résolution peut être invoquée en mode *repair* où un sous-ensemble des affectations des VMs aux PMs dans la configuration courante, qui satisfont toutes les nouvelles contraintes du problème sont fixées a priori. Seules les affectations conflictuelles peuvent alors être remises en question. Ce mode de résolution est analogue à une itération d'une recherche locale autour de la configuration courante. À noter que le voisinage est également évalué de manière heuristique, l'algorithme de backtracking étant tronqué à la première solution trouvée dans un temps limité configurable (5 minutes par défaut).

L'identification d'un ensemble d'affectations conflictuelles minimal est réalisée de manière heuristique en analysant chacune des contraintes individuelles. Toutes les VMs dont l'affectation courante apparaît dans la violation d'une contrainte donnée, sont ajoutées au modèle. La contrainte $spread(V)$ par exemple, assure que les VMs de l'ensemble V sont toutes affectées à des PMs disjointes deux-à-deux. Cette contrainte utilisateur est proposée dans de nombreux systèmes, dont VMWare [EF10] et OpenStack [Sof], à des fins de tolérance aux pannes. Dans BtrPlace, cette contrainte est modélisée par une contrainte `alldifferent` [3]. Le checker associé vérifie si toutes les VMs de V sont couramment affectées à des PMs distinctes deux-à-deux. Dans le cas contraire, seules les VMs colocalisées sont retenues comme candidates potentielles à la migration. Les autres VMs, si elles n'apparaissent dans aucun autre conflit, seront fixées à leur affectation courante à la prochaine résolution.

3. quand la contrainte porte uniquement sur la configuration cible ; si elle doit être satisfaite aussi pendant la reconfiguration, elle est alors modélisée par des contraintes de précédence réifiées

Cette approche ne garantit pas de retourner un ensemble conflictuel ni suffisant, ni minimal. Nous avons cependant conçu les checkers individuels de manière à sélectionner des ensembles conflictuels réduits pour limiter la taille du problème, mais pas trop pour donner une latitude au solveur de réparer le conflit.

2.1.5 Contributions

Si l'apport théorique de ces travaux est relativement limité, leur contribution est importante en terme de valorisation de la programmation par contraintes et du principe de décomposition en contraintes globales.

Expressivité et extensibilité. Premièrement, ces travaux contribuent à affirmer la force d'expressivité de la programmation par contraintes donnée par ce principe. Formaliser en un programme mathématique, par exemple, le sous-problème de planification de la reconfiguration, incluant le modèle des différents types d'action, dont la migration à chaud et sa consommation de ressources particulière, est une tâche particulièrement ardue aboutissant à des modèles d'ordonnancement lourds et certainement peu performants. La programmation par contraintes offre un cadre déclaratif pour la résolution du problème entier, mais en encapsulant ce sous-problème au sein d'une contrainte globale dédiée, nous nous affranchissons de l'effort de modélisation dans un formalisme donné et avançons directement un traitement algorithmique, même simple et incomplet.

De même, chaque contrainte utilisateur est transcrite au sein du modèle par une contrainte globale ou, dans de rares cas, par un ensemble de contraintes globales associées à de nouvelles variables induites. La transcription se fait de manière systématique à partir d'un langage haut niveau. La définition des contraintes globales (ou leur redéfinition en l'occurrence ici quand nous avons abandonné les variables ensemblistes) et leur implémentation (i.e. le choix de l'algorithme de filtrage) n'impactent pas le modèle cœur qui devient alors potentiellement extensible à l'infini.

Concernant l'expressivité de l'approche et son extensibilité via de nouvelles contraintes utilisateur, on doit cependant distinguer le cas où le modèle repose sur une contrainte globale existante, implémentée dans le moteur de contraintes employé, et le cas où il est nécessaire de concevoir et d'implémenter une nouvelle contrainte globale. La modélisation est accessible à des utilisateurs non initiés à la programmation par contraintes, ou du moins à l'algorithmique, dans le premier cas, mais plus difficilement dans le second cas. En particulier, cela soulève l'importance de l'ergonomie des solveurs de contraintes et notamment de la facilité de définir et d'implémenter de nouvelles contraintes. Le choix du solveur Choco dans BtrPlace a par exemple été motivé par son ergonomie, autant ou plus que par ses performances.

Flexibilité et robustesse. C'est aussi le principe d'isolation des contraintes globales qui fournit à l'approche sa flexibilité dynamique. Elle est nécessaire dans ce contexte d'optimisation en quasi-temps réel, où la définition du problème est spécialisée à chaque nouvelle résolution. Les contraintes utilisateurs sont ainsi spécifiées en ajoutant au modèle les contraintes globales associées, sans que l'algorithme général de résolution en soit modifié.

Par ailleurs, l'ensemble du solveur est paramétrable (le choix du mode de résolution *rebuild/repair*, les critères d'optimalité et les heuristiques de recherche, l'algorithme de filtrage de `vector-packing`) pour pallier à la variété des cas d'application potentiels : des centres de données privés de 10 serveurs aux publics de milliers de serveurs ; de la gestion d'une charge moyenne dépassant rarement les 3/4 de la disponibilité à l'absorption de pics de consommation soudains ; de la stabilité des opérations de calcul à la dynamique des applications web ; etc. Ces paramètres n'influent pas ou peu les uns sur les autres. Ainsi, les algorithmes de filtrage minimaux du packing et de l'ordonnancement, qui sont suffisants pour des taux de charge moyenne, peuvent être renforcés individuellement sans impacter le modèle global. De même, le mode heuristique de réparation des conflits peut être levé pour redonner de la liberté de choix à la méthode dans les cas de surcharge.

Une limite actuelle de BtrPlace est qu'il nécessite un paramétrage manuel. Celui-ci peut être réalisé par des expérimentations préalables à l'installation sur un centre de données stable. En revanche, le paramétrage n'est pas réactif en cas d'un changement subit de contexte. Une piste serait de mettre en place un système d'apprentissage afin d'automatiser également ce paramétrage, mais ceci nécessite notamment l'obtention de bases de test, si possible issues d'observations réelles.

Scalabilité. La scalabilité, nous l'avons obtenue principalement de trois manières propres à – ou du moins facilitées par – la décomposition en contraintes globales : (1) en limitant les redondances, en mémoire et temps de calcul, en agrégeant notamment les dimensions dans la contrainte de `vector-packing` et dans la contrainte d'ordonnancement, (2) en paramétrant l'algorithme de filtrage de `vector-packing` (à la fois le raisonnement et les structures de données) en fonction du rapport difficulté/taille des instances, (3) en implémentant un mode de réparation, relâchant dans la solution courante une sous-instantiation identifiée conflictuelle pour au moins l'une des contraintes du problème.

Nos expérimentations [4] sur des centres de données simulés, mais basés sur des observations réelles, attestent de la capacité de BtrPlace à traiter d'instances de grande taille. La figure 2.1 présente ainsi les temps de résolution (en secondes) pour deux scénarios de reconfiguration sur un centre de données de 5.000 PMs exécutant des applications web 3-tiers dont le nombre de VMs associées varie de 10.000 à 30.000 (soit un taux de consolidation de 3 à 6 VMs par PM, et un taux d'utilisation des ressources CPU+RAM variant de 36% à 73%). Le scénario `LI` prévoit une augmentation de 30% de la charge CPU de 10% des applications (soit 5% d'augmentation de la charge totale) et le scénario `NR` une reconnection du réseau impliquant la fermeture de 5% des PMs (soit le taux de maintenance des centres de données de Google en cas de panne réseau). Toutes les instances sont résolues dans les deux scénarios, du moins pour les ratios de consolidation de 3 à 4 VMs par PM que l'on observe couramment en pratique. Le scénario `NR` est plus facilement résolu, notamment car l'heuristique de réparation réduit la taille du modèle par un facteur 20 contre un facteur 10 pour les instances `LI`. Dans `NR`, seules les VMs hébergées sur les PMs en maintenance doivent être réaffectées tandis que, dans `LI`, il s'agit de l'ensemble des VMs hébergées sur les PMs surchargées.

La figure 2.2 montre que la présence de contraintes utilisateur impacte aussi de manière

4. L'analyse détaillée est publiée dans [DHK15].

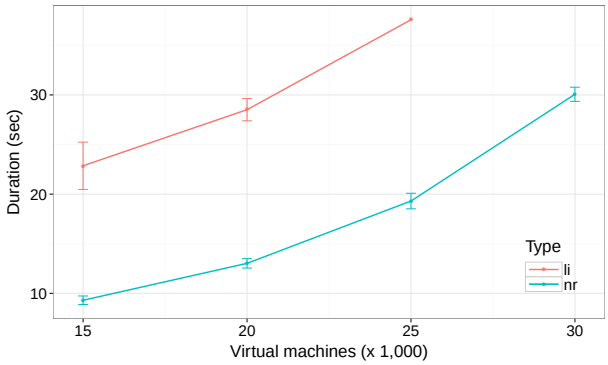
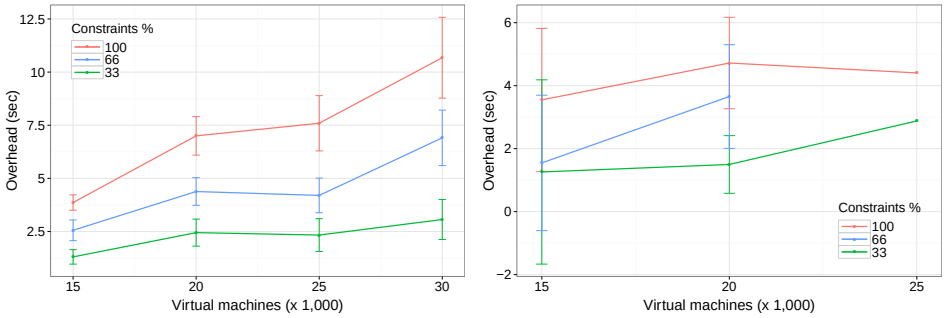


FIGURE 2.1 – Temps moyen de résolution en fonction du nombre de VMs pour les deux scénarios NR et LI



(a) NR (b) LI

FIGURE 2.2 – Impact des contraintes utilisateurs sur les temps de résolution sur les instances NR (gauche) et LI (droite)

acceptable les temps de résolution. Des contraintes utilisateur, liées à une demande en haute disponibilité, sont associées à 1/3 (vert), 2/3 (bleu) ou toutes (rouge) les applications web. Elles sont spécifiées par les contraintes utilisateur *spread* et *among*, couplant entre 2 et 10 VMs pour chacune des applications. Le surcoût en temps de calcul lié à l'ajout des contraintes utilisateur sont, dans les pires cas, de 11 secondes (34%) pour le scénario NR et 4 secondes (11%) pour le scénario LI. Ce faible surcoût s'explique par la dominance des contraintes de ressources ainsi que par la réduction de l'espace recherche permise par ces contraintes.

En sachant gérer des systèmes de plus d'un millier de PMs et de dizaine de milliers de VMs, BtrPlace participe à contrer l'idée reçue répandue du manque de scalabilité et de performance de la programmation par contraintes en optimisation. BtrPlace doit aussi sa notoriété à la flexibilité qu'elle offre par la définition des contraintes utilisateur. BtrPlace est une solution open-source, aujourd'hui intégrée dans différentes solutions commerciales dont notamment Nutanix qui l'a choisi pour ces raisons de performance (supérieure à des heuristiques ad-hoc), de flexibilité et d'extensibilité. Dans ce contexte, elle tourne quotidiennement sur un millier de clouds privés d'entreprise.

2.2 Décomposition et recherche opérationnelle pour Interval-Amongs

Réalisée en collaboration avec Gilles Chabert en marge du projet européen FP7 Angels, cette étude publiée dans [CD12] a été motivée par une application de localisation par des réseaux de capteurs. Ce problème se modélise par `interval-amongs`, une conjonction particulière de contraintes de cardinalité `among` portant sur un même ensemble de variables X (les positions des objets), et sur des ensembles de valeurs spécifiés par des intervalles V_i (les zones d'observation des capteurs). Chaque contrainte `among` compte/contraint les nombres minimal et maximal de variables X qui prennent leur valeur dans un ensemble V_i.

Définition 3 (`interval-amongs`) *Soient $X = (X_j)_{j=1}^n \in \mathbb{Z}^n$ un tuple de n variables, $V = (V_i)_{i=1}^m \subseteq \mathbb{Z}^m$ un ensemble de m intervalles de valeurs entières, et des valeurs de capacité minimum $\underline{k} = (\underline{k_i})_{i=1}^m \in \mathbb{Z}^I$ et maximum $\overline{k} = (\overline{k_i})_{i=1}^m \in \mathbb{Z}^I$, alors la contrainte `interval-amongs`$(X, V, \underline{k}, \overline{k})$ est satisfaite si et seulement si, pour tout $i \in \{1, \dots, m\}$, le nombre de variables de X prenant leur valeur dans l'intervalle V_i est compris entre $\underline{k_i}$ et $\overline{k_i}$, ou formellement :*

$$\texttt{interval-amongs}(X, V, \underline{k}, \overline{k}) \iff \underline{k_i} \leq \text{card}(\{j \in \{1, \dots, n\} \mid X_j \in V_i\}) \leq \overline{k_i},\ \forall i \in \{1, \dots, m\}.$$

Sans perte de généralité, l'union des domaines de X peut être assimilé à l'ensemble Σ des entiers de 1 à p avec $p \leq 2m$.

2.2.1 Preuve de complexité

S'il est prouvé [Rég05] que le problème de satisfaisabilité d'un système de contraintes `among` quelconques est $\mathcal{NP}$-complet, il devient polynomial dans certains cas pratiques. Nous prouvons que la satisfaisabilité (et par conséquence, la consistance aux bornes) de `interval-amongs` est dans $\mathcal{P}$, et $\mathcal{NP}$-complet en dimensions supérieures (quand les zones-intervalles deviennent des aires et des volumes).

La preuve de complexité en dimension 1 repose sur une reformulation de `interval-amongs` en un système (P_L) d'inégalités linéaires sur les seules variables duales du modèle : les variables de cardinalité y_s associées aux valeurs $s \in \Sigma$. La preuve de la reformulation repose à son tour sur un modèle de flot dans un certain graphe d'affectation :

Lemme 1 *$x \in X$ satisfait `interval-amongs`$(x, V, \underline{k}, \overline{k})$ si et seulement s'il existe un vecteur $y \in \mathbb{Z}_+^p$ solution de :*

$$(P_L): \quad \underline{k_i} \leq \sum_{s \in V_i} y_s \leq \overline{k_i}, \qquad \forall i \in I, \tag{2.1}$$

$$L_{[a,b]} \leq \sum_{s \in [a,b]} y_s, \qquad \forall a \leq b \in \Sigma, \tag{2.2}$$

$$\sum_{s \in \Sigma} y_s \leq n, \tag{2.3}$$

où, pour chaque intervalle non-vide de valeurs $[a, b]$ de Σ, $L_{[a,b]}$ dénote le nombre de variables à valeurs dans $[a, b]$: $L_{[a,b]} = card\{j \in J \mid X_j \subseteq [a, b]\}$.

Preuve. La condition nécessaire est évidente. La condition suffisante repose sur l'existence d'un flot dans le graphe biparti orienté d'instantiation $G = (J \cup \Sigma, E, c)$ sur l'ensemble des arcs $E = \{(j,s) \in J \times \Sigma \mid s \in X_j\}$ de capacité $c_e = 1$ sur chaque arc $e \in E$, augmenté d'une source u et d'un arc (u,j) de capacité 1 pour chaque variable $j \in J$, d'un puits v et d'un arc (s,v) de capacité y_s pour chaque valeur $s \in \Sigma$. À toute solution x correspond un flot

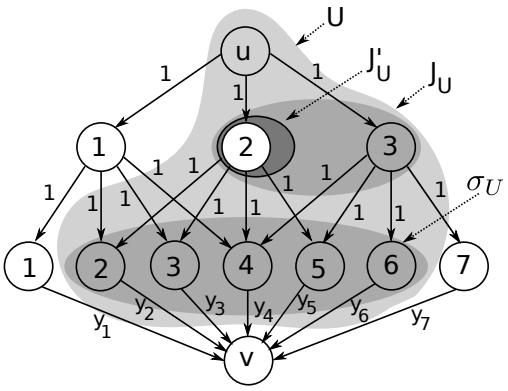


FIGURE 2.3 – **Le graphe du modèle de flot pour 3 variables, 7 valeurs et un exemple de coupe, U en gris clair, les ensembles de noeuds $J \cap U$ et $\Sigma \cap U$ en gris moyen, et J'_U in gris foncé.**

compatible de valeur n de u vers v dans ce graphe, avec $x_j = s$ si et seulement si le flot est non nul sur l'arc $(j,s) \in E$. L'existence d'un tel flot est conditionnée, d'après le théorème de Hoffman, par le fait que toute (u,v)-coupe dans ce graphe est de capacité au moins n. Une (u,v)-coupe (U,V) est composée des arcs de trois sous-graphes engendrés respectivement par les ensembles bi-partis de nœuds : $(u, J \cap V)$, $(J \cap U, \Sigma \cap V)$ et $(\Sigma \cap U, v)$. La capacité totale du premier sous-ensemble est $n - \text{card}(J \cap U)$ et la capacité du deuxième est au moins égale à $\text{card}(J \cap U) - \text{card}(J'_U)$ où J'_U désigne l'ensemble des noeuds parmi J dont au moins un arc sortant appartient à la coupe. On montre alors que la capacité totale du troisième sous-ensemble $\sum_{s \in \Sigma \cap U} y_s$ est au moins égale à $\text{card}(J'_U)$. Pour ce faire, on distingue la partition en intervalles entiers de $\Sigma \cap U = [a_1, b_1] \cup \ldots \cup [a_r, b_r]$. On a alors :

$$\begin{aligned}
\text{card}(J'_U) &= \text{card}\{j \in J \mid X_j \subseteq \Sigma \cap U\} && \text{par définition de } J'_U \\
&= \sum_{l=1}^{r} \text{card}\{j \in J \mid X_j \subseteq [a_l, b_l]\} && \text{car } X_j \text{ est un intervalle} \\
&= \sum_{l=1}^{r} L_{[a_l,b_l]} \leq \sum_{l=1}^{r} \sum_{s \in [a_l,b_l]} y_s = \sum_{s \in \Sigma \cap U} y_s && \text{d'après (2.2).}
\end{aligned}$$

Le système (P_L) peut être réécrit comme la conjonction de contraintes de la forme $L'_{[a,b]} \leq \sum_{s \in [a,b]} y_s \leq U'_{[a,b]}$ pour tout intervalle d'entiers $[a,b] \subseteq \Sigma$. Par un changement de variables défini récursivement par $z_0 = 0$, $z_b = \sum_{s=1}^{b} y_s$ pour tout $b \in \Sigma$, on identifie (P_L) à un réseau de contraintes temporelles [DMP91] :

$$(P_T) : \qquad z_b - z_a \leq d_{ab}, \qquad \forall a, b \in \{0\} \cup \Sigma. \tag{2.4}$$

Une solution entière de (P_T) peut être déterminée en calculant un plus court chemin dans le graphe orienté complet sur les nœuds numérotés de 0 à p et valué par d_{ab} sur chaque arc (a, b) ; l'existence d'une telle solution étant conditionnée par l'absence de cycle négatif dans ce graphe. La construction du graphe et la recherche d'un cycle négatif (ou d'un plus court chemin) s'effectuent en temps polynomial, par l'algorithme de Floyd-Warshall par exemple.

2.2.2 Décompositions pour le filtrage

La preuve ci-dessus décrit un algorithme polynomial de filtrage assurant la consistance aux bornes sur la contrainte `interval-amongs` : il s'agit d'intégrer le test de satisfaisabilité, l'algorithme de Floyd-Warshall ici, dans une boucle de *shaving* pour tester une par une la consistance des bornes. Cette solution présente une complexité en $O(n^2m^4)$ (soit n^2m appels, dans le pire cas, d'un algorithme de complexité $O(p^3)$) trop élevée pour un usage pratique.

La question d'un algorithme complet plus rapide est ouverte, cependant, on peut déduire des reformulations précédentes des résultats de filtrage plus immédiats :

La décomposition classique en m contraintes `among` est directement implémentable dans les nombreux solveurs qui fournissent cette contrainte. La consistance aux bornes sur cette décomposition peut être assurée en $O(n^2m^2)$.

Une reformulation intermédiaire à (P_L) consiste en une décomposition de `interval-amongs` en m contraintes de somme (2.1) et une contrainte globale de cardinalité `global-cardinality`. Ces contraintes classiques sont également disponibles dans de nombreux solveurs et offrent une alternative directe à la décomposition orthogonale précédente.

Enfin, l'algorithme de Floyd-Warshall appliqué au réseau de contraintes temporelles (P_T) en assure la consistance aux bornes. En conjonction avec une contrainte globale de cardinalité, pour la propagation sur les variables originales x du problème, il offre un filtrage en $O(nm^3 + n^2m)$. À noter que Floyd-Warshall est implémenté comme propagateur des contraintes de précédence en ordonnancement dans certains solveurs. Il doit être cependant réimplémenté dans ce cas-ci, car les poids d_{ab} du graphe sous-jacent doivent être mis à jour lors d'une modification des bornes des variables x.

On désigne ces trois reformulations de `interval-amongs`, respectivement, par *décomposition among*, *décomposition somme-cardinalité*, et *décomposition cardinalité*.

Il est facile d'exhiber des instances de `interval-amongs` inconsistantes aux bornes, mais sur lesquelles la consistance aux bornes sur l'une des décompositions est atteinte. Autrement dit, la borne-consistance sur `interval-amongs` est strictement plus forte que sur chacune de ces décompositions. La décomposition somme-cardinalité est également dominée par la décomposition cardinalité. En revanche, les borne-consistances sur les décompositions *among* et *cardinalité* sont incomparables, et la conjonction des deux est strictement dominée par `interval-amongs`.

Si les décompositions *among* et *cardinalité* sont formellement incomparables, la seconde semble capturer un peu plus la globalité du problème dans le sens où elle n'est constituée que de 2 contraintes globales au lieu de m. Cette dominance a été clairement observée de manière empirique sur notre protocole de test. Nous avons généré 100 instances réalisables avec $n = m$ pour toutes les valeurs de n comprises entre 10 et 32, implémenté le propagateur

de cardinalité et comparé les deux formulations sur ces 2300 instances dans l'algorithme de backtracking standard du solveur Choco 2.1.2. En choisissant $n = m$, nous assurons que la propagation dans les deux cas est de même complexité $O(n^4)$. Nous mesurons alors le nombre d'instances pour lesquelles une première solution est trouvée plus rapidement, sur une échelle logarithmique, par l'un ou l'autre des solveurs. Les résultats sont représentés par la figure 2.4. Le nombre d'instances pour lesquelles la décomposition *cardinalité* est plus de 100 fois plus

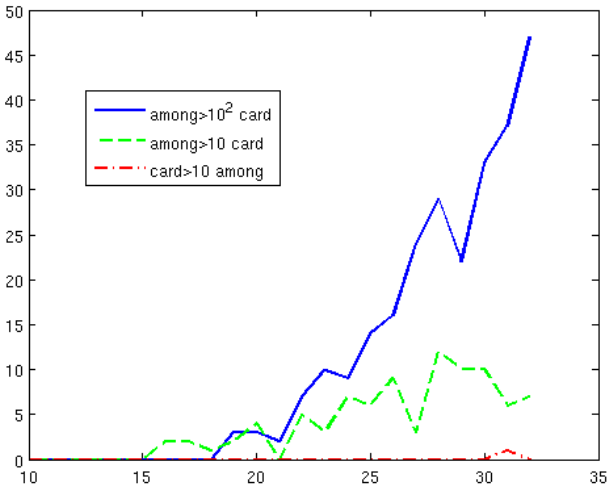


FIGURE 2.4 – Comparaison des temps de résolution pour les décompositions *among* et *cardinalité* sur 100 instances de taille n, n variant de 10 à 32. Les trois courbes continue/bleu, discontinue/verte et pointillée/rouge indiquent le nombre d'instances sur lesquelles la décomposition *cardinalité* est respectivement plus de 100 fois plus rapide, 10 fois plus rapide, 10 fois plus lente.

rapide augmente rapidement avec n, et représente la moitié des instances pour $n = 32$. Pour les autres instances, soit la décomposition est plus de 10 fois plus rapide, soit une première solution est rapidement trouvée par les deux décompositions (non représenté). Aucune instance n'a été résolue 100 fois plus rapidement par la décomposition *among*, et seulement une instance (pour $n = 31$) a été résolue 10 fois plus vite par cette décomposition.

2.2.3 Contributions

Notre première contribution ici est d'ordre applicatif, avec la définition d'une nouvelle contrainte globale de cardinalité et d'un filtrage qui s'avère en pratique plus efficace que la décomposition naturelle (la conjonction de `among`). Pour autant, cet article paru dans la conférence dédiée CPAIOR n'a reçu aucune citation en 4 ans : (1) ses domaines d'application sont nettement plus limités que ceux de `global-cardinality` ou moins étudiés que ceux (principalement le car-sequencing) de `sequence`[5] ; (2) il s'agit d'une alternative de modélisation, plus efficace peut-être, mais moins naturelle. Cette application montre l'intérêt des contraintes spécifiques mais aussi la difficulté posée par leur diffusion.

5. nous n'avons nous-même, faute de temps, pas abouti sur le problème de localisation qui a motivé ces travaux

La seconde contribution est d'ordre méthodologique : notre argumentation – basée sur des reformulations duales, des résultats de théorie des graphes, des relaxations – est dans la lignée de [Rég05 ; Bes+05a ; Bra+07 ; Mah+08] pour diverses conjonctions de `among`, mais elle présente des spécificités. Ainsi, notre première reformulation étend celle présentée dans [Bes+05b] pour une contrainte `among`, composée d'un système de *contraintes de capacité* sur les variable duales $(y_s)_{s\in\Sigma}$ et d'un système de *contraintes de channelling* entre les variables x et y. Contrairement à [Rég05 ; Bra+07], le système de contraintes capacité+channelling n'est pas Berge-acyclique de sorte que le filtrage sur le modèle dual n'assure pas la consistance d'arcs sur le modèle primal.

Par ailleurs, comme décrit dans [Rég11], les contraintes de cardinalité sont associées à des problèmes d'affectation et se posent naturellement en application d'un algorithme de flot dans le graphe d'affectation. Ainsi, pour la contrainte `sequence`, Maher et al. [Mah+08] dérivent du système de contraintes temporelles de [Bra+07] un modèle de flot sur lesquels ils appliquent un filtrage incrémental similaire à `global-cardinality` de Régin [Rég96]. Cet algorithme ne s'applique en revanche pas à notre modèle de flot (figure 2.3), où les y représentent les capacités des arcs et non les valeurs du flot. Enfin, le réseau de contraintes temporelles (P_T) est un graphe complet. Le réduire à un problème de flot comme dans [Mah+08] requiert une structure spécifique et utiliser la variante incrémentale de l'algorithme de Johnson comme dans [Bra+07] n'est pas une meilleure alternative à Floyd-Warshall dans ce cas. Toutes ces études illustrent la difficulté d'employer des techniques de graphes et de recherche opérationnelle pour la conception de contraintes globales, ainsi que l'intérêt des décompositions pour les contraintes coûteuses. Elles montrent également que si l'approche générale est adaptable, il est nécessaire parfois de modifier intégralement le raisonnement dès qu'une variation, même minime de la contrainte est considérée.

Enfin, les algorithmes de filtrage que nous proposons pour `interval-amongs` reposent sur le principe de recomposition : partant d'une conjonction de contraintes, nous la transposons en une autre conjonction de contraintes, équivalente dans la sémantique mais pas dans la force de filtrage. Ce principe générique de recomposition de contraintes ne nécessite pas de mettre en œuvre un algorithme dédié complexe comme il repose en partie sur le moteur de propagation du solveur. En revanche, la complexité ici est d'identifier le modèle alternatif et éventuellement de caractériser sa capacité de filtrage et de la comparer au modèle initial. Dans ce même article [CD12], nous étudions des problèmes de localisation en 2 ou 3 dimensions et prouvons que la contrainte `interval-amongs` devenait $\mathcal{NP}$-difficile dans ces cas. Nous montrons aussi que la décomposition naturelle suivant chacune des dimensions inhibe fortement la propagation. Dans ce contexte, il serait tout à fait légitime d'étudier une recomposition alternative de cette contrainte $\mathcal{NP}$-difficile, mais s'abstraire de la décomposition naturelle par dimensions n'est pas une tâche évidente. Je reviens sur ce principe de recomposition dans le chapitre suivant (section 3.1.2.1).

2.3 Génération de colonnes hybride pour la planification de personnel

Cette section présente mes travaux de post-doc réalisés en 2005 à Polytechnique Montréal puis au sein de la société Omega Optimisation, en collaboration avec Louis-Martin Rousseau

et Gilles Pesant. Ces travaux publiés dans [DPR05 ; DPR06] portent sur la résolution d'un problème de planification de personnel multi-activités avec des règles d'ordonnancement complexes par une approche déclarative à la fois efficace et flexible pour s'adapter à tout type de règles métier.

2.3.1 Décomposition couverture-ordonnancement

Génération de colonnes. La planification de personnel consiste à concevoir les horaires des employés d'un service de façon à couvrir une charge de travail sur un horizon de temps donné tout en minimisant le coût de réalisation (coût du travail, pénalités de sous/sur-charge, pénalités d'insatisfaction des employés, etc.). Une grande variété de problèmes existent et diffèrent sensiblement selon les contextes d'application comme les centres d'appel, le personnel hospitalier (nurse scheduling) ou les équipages dans le transport aérien ou routier (crew scheduling). Notre étude porte sur une application plutôt générale, quand l'horizon temporel peut être discrétisé, basée sur deux cas d'études réels de conception des emploi du temps d'employés de commerce.

Ce problème présente deux facettes orthogonales : (1) l'allocation des ressources (humaines) aux activités à chaque pas de temps afin de couvrir la charge et minimiser les coûts d'affectation et (2) l'ordonnancement des activités qui forment l'horaire de chaque employé soumis à différentes contraintes réglementaires ou personnelles. Le problème d'allocation de ressources se modélise, classiquement depuis les travaux de Dantzig [Dan54], par un programme linéaire en variables binaires de type *set-covering* consistant à sélectionner un sous-ensemble d'horaires couvrant la charge parmi l'ensemble $\mathcal{S}$ exhaustif des horaires valides, c'est à dire qui satisfont les règles d'ordonnancement. Soit $b^s_{at} \in \{0,1\}$ une variable binaire indiquant si l'activité a est travaillée au temps t dans l'horaire valide $s \in \mathcal{S}$ (i.e. $b^s_{at} = 1 \iff s(t) = a$), $c^s = \sum_t \sum_a b^s_{at} c_{at}$ le coût de l'horaire s, et r_{at} la charge minimale attendue pour l'activité a au temps t, alors le modèle s'écrit :

$$\min \quad \sum_{s \in \mathcal{S}} c^s x_s \tag{2.5}$$

$$s.t. \quad \sum_{s \in \mathcal{S}} b^s_{at} x_s \geq r_{at} \qquad \forall\, a \in A, \forall\, t \in [1..T], \tag{2.6}$$

$$x_s \geq 0 \qquad \forall\, s \in \mathcal{S}, \tag{2.7}$$

$$x_s \in \mathbb{Z} \qquad \forall\, s \in \mathcal{S}. \tag{2.8}$$

Contrairement à une formulation compacte qui mêlerait contraintes de couverture et contraintes d'ordonnancement, ce modèle explicite les isole. Reposant sur le pré-calcul de l'ensemble $\mathcal{S}$ des horaires valides, il offre la possibilité de prendre en compte des règles d'ordonnancement difficiles voir impossibles à modéliser dans le cadre restreint de la programmation mathématique. Si l'ensemble des horaires valides est trop conséquent, il peut être construit de manière partielle et itérative. La génération de colonnes permet notamment de résoudre la relaxation continue du programme en générant progressivement des horaires valides jusqu'à produire un sous-ensemble optimum. À chaque itération, il s'agit d'identifier un ou plusieurs horaires valides $s \in \mathcal{S}$ de coût réduit négatif $\sum_t c_{s(t)t} - \lambda_{s(t)t} < 0$ avec $(\lambda_{at})_{a \in A, t \in [1..T]}$ les valeurs duales associées aux contraintes de couverture (2.6) dans le programme maître restreint.

Heuristiques La valeur optimale de la relaxation continue fournit une borne inférieure du problème. Une solution réalisable (discrète) peut ensuite être obtenue simplement en arrondissant à l'entier supérieur la solution fractionnaire. Cependant, comme les contraintes de couverture impliquent chacune un grand nombre de variables (de l'ordre de $|A|^{n-1}$), la solution relaxée est potentiellement fortement fractionnaire et le surcoût de l'arrondi non négligeable. La solution relaxée peut également être arrondie progressivement de manière constructive (à chaque itération, on ajoute un horaire apparaissant dans un maximum de contraintes de couverture non satisfaite) puis améliorée par recherche locale.

Une autre approche heuristique directement implémentable consiste à résoudre le programme linéaire en variables binaires restreint aux seuls horaires générés par la génération de colonnes. La résolution exacte peut cependant être longue étant données la taille du problème, la densité de la matrice de contraintes et la forte symétrie du problème de par l'existence de nombreux horaires valides pratiquement identiques.

Branch-and-Price robuste L'obtention d'une solution avec certificat d'optimalité nécessite d'étendre la génération de colonnes à la racine de l'arbre de recherche, à un branch-and-price, avec génération de colonnes à chaque noeud pour générer les horaires optimaux manquants et évaluer exactement le nœud. Le choix de la stratégie de branchement est délicate dans un branch-and-price car celle-ci ne doit pas, dans l'idéal, complexifier le sous-problème. La stratégie de séparation classique $x_s \leq k$ et $x_s \geq k+1$ est par exemple peu adaptée. D'une part, l'arbre de recherche est fortement non balancé. D'autre part, l'ajout d'une contrainte $x_{s^*} \leq k$ dans le programme maître nécessite de contraindre explicitement le sous-problème afin qu'il ne génère pas à nouveau l'horaire s^*. En effet, elle modifie la définition du coût réduit ($rc_s + \mu_{s^*}$ avec $\mu_{s^*} \geq 0$ la valeur duale associée à la contrainte de branchement), de sorte que la condition $rc_s + \mu_{s^*} \geq 0$ ne prévient plus la condition $rc_s < 0$. Étudiées dans le cadre des problèmes de tournée ou de packing, les stratégies de branchement dite robustes visent à ne pas augmenter la complexité du sous-problème. Ces stratégies sont basées sur la reformulation du programme maître en un modèle de flot et consistent à brancher sur des sommes de variables. Une telle reformulation s'applique dans le cas présent :

$$\min \quad \sum_{s \in \mathcal{S}} c^s x_s \tag{2.9}$$

$$s.t. \quad \sum_{b \in A} f^t_{ab} \geq r_{at} \qquad \forall\, a \in A, \forall\, t \in \{1, \ldots, T\}, \tag{2.10}$$

$$f^t_{ab} = \sum_{s \in \mathcal{S}} \delta^s_{at} \delta^s_{b(t+1)} x_s \qquad \forall\, a, b \in A, \forall\, t \in \{1, \ldots, T\}, \tag{2.11}$$

$$x_s \geq 0, x_s \in \mathbb{Z} \qquad \forall\, s \in \mathcal{S}, \tag{2.12}$$

$$f^t_{ab} \geq 0, f^t_{ab} \in \mathbb{Z} \qquad \forall\, a, b \in A, \forall\, t \in \{1, \ldots, T\}. \tag{2.13}$$

Dans ce programme linéaire, une variable f^t_{ab} précise le nombre d'employés affectés à une activité a au temps t et à une activité b au temps $t+1$. Brancher sur une variable de ce type est effectivement robuste et particulièrement adapté à notre modèle de contraintes où elle se traduit par la suppression d'arcs dans l'automate de `cost-regular` (voir ci-dessous). En revanche, l'argument usuel de complétude de la recherche est insuffisant. En effet, il est

facile d'exhiber des cas où toutes les variables de flot sont entières mais pas les variables x, par exemple pour $n = 4$, affecter un demi employé à chacun des horaires (a_1, a_3, a_1, a_3), (a_1, a_3, a_2, a_3), (a_2, a_3, a_1, a_3), (a_2, a_3, a_2, a_3) donne une couverture entière où toutes les variables de flot sont entières. Cette stratégie de branchement peut donc être employée en début de recherche et complétée éventuellement en branchant sur les x fractionnaires restants.

2.3.2 Modèle de contraintes en ordonnancement riche

Dans une approche exacte de génération de colonnes, le sous-problème à chaque itération consiste à déterminer une séquence d'activités de coût d'affectation minimal et satisfaisant l'ensemble des règles d'ordonnancement qui sont, pour la plupart, définies par des séquences d'activités (ou *motifs*) interdites ou par des restrictions sur le nombre d'occurrences des activités. Le problème s'apparente à un problème de plus court chemin avec contraintes de ressources (RCSPP) dans un certain graphe dont la topologie traduit les règles de motifs, et la pondération des arcs, les occurrences des activités [Des+95]. Si la résolution du problème, répétée dans le cadre de la génération de colonnes, est envisageable par programmation dynamique, la construction initiale du graphe peut être un défi en soi selon la singularité des règles. En pratique, à la manière des approches heuristiques, les règles difficiles ou impossibles à traduire dans le graphe sont appliquées après calcul, en éliminant les horaires invalides avant de les intégrer dans le programme maître.

La programmation par contraintes offre une alternative pour la modélisation et la résolution du sous-problème. Son expressivité lui permet d'intégrer les contraintes d'ordonnancement les plus diverses. L'approche de génération de colonnes par programmation par contraintes est alors doublement robuste à la variation des règles d'ordonnancement dans différents contextes de planification de personnel. En effet, ces règles se retrouvent encapsulées dans le sous-problème de génération de colonnes, puis dans des contraintes dédiées du modèle. Cette flexibilité avait motivé les premières applications de cette approche au problème du crew scheduling [Cap+98 ; YMD05], un problème de planification de personnel proche de celui considéré ici mais qui comprend principalement des règles de motifs simples.

2.3.3 Contrainte de langage et d'optimisation pour les règles d'ordonnancement

Notre approche combine à la fois les aspects déclaratifs et flexibles de la programmation par contraintes avec l'efficacité de la programmation dynamique dans le contexte où les règles de motifs prédominent. En les agrégeant au sein d'une nouvelle contrainte globale `cost-regular`, nous proposons d'automatiser la construction initiale du graphe des horaires à partir de l'ensemble des règles de motifs présentes, puis de filtrer le graphe durant la résolution en maintenant les plus courts (et plus longs) chemins. `cost-regular` est en effet la variante d'optimisation de la contrainte `regular` [Pes04], une contrainte qui assure qu'une séquence de variables forme un mot appartenant au langage régulier spécifié par un automate déterministe fini :

Définition 4 *Un* automate Π *est un multi-graphe orienté* (Q, Δ) *dont les arcs sont étiquetés par les* symboles *(i.e. les éléments) d'un* alphabet *(i.e. un ensemble non-vide)* Σ. *Une* transition $(q_1, \sigma, q_2) \in \Delta \subseteq Q \times \Sigma \times Q$ *correspond à un arc de* q_1 *à* q_2 *étiqueté* σ. *L'automate est* déterministe fini *si* Δ *est fini*

et s'il n'existe pas deux transitions issues d'un même sommet portant la même étiquette. Les sommets Q sont appelés des états *et on distingue deux sous-ensembles particuliers les états initiaux I (unique si déterministe) et les états acceptants A. Un* mot *(i.e. une séquence de symboles) sur Σ est* reconnu *par Π s'il correspond à la séquence des labels des arcs d'un chemin depuis l'état initial vers un état acceptant dans Π. L'ensemble des mots reconnus par Π est $\mathcal{L}(\Pi)$ le* langage *reconnu par Π. Voir [HMU01] par exemple pour plus de détail sur la théorie des automates.*

Définition 5 *Soient $(X_1, \dots, X_n)$ une suite de variables de domaines finis dans un ensemble discret Σ et Π un automate déterministe fini sur l'alphabet Σ, alors $\texttt{regular}((X_i)_{i=1}^n, \Pi)$ est satisfaite si et seulement si $(X_i)_{i=1}^n$ est un mot reconnu par Π.*

Comme les mots reconnus par `regular` sont de longueur fixe (n), l'algorithme de filtrage [Pes04] opère sur un graphe acyclique Π^n obtenu en répliquant le graphe Π en n couches. Un arc étiqueté σ dans la couche i du graphe correspond alors à l'affectation de la valeur σ à la variable X_i. L'algorithme procède en deux passes avant/arrière à travers les n couches du graphe pour éliminer les arcs n'appartenant à aucun chemin menant vers un état final dans la dernière couche. La valeur σ est supprimée du domaine de X_i quand plus aucun arc de la couche i n'est étiquetée σ.

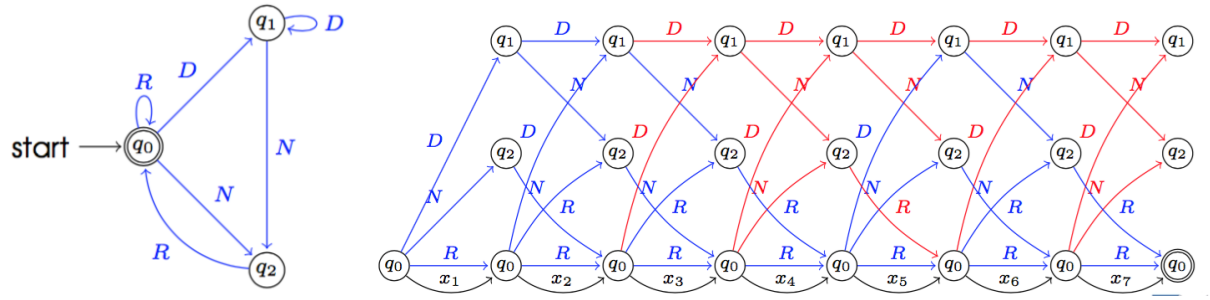


FIGURE 2.5 – À gauche, un automate déterministe fini Π avec un état initial et final q_0 sur l'alphabet $\Sigma = \{R, N, D\}$. À droite, l'automate déployé Π^7 reconnaissant uniquement les mots de longueur 7. Les arcs en rouge n'appartiennent à aucun chemin vers l'état final et sont donc supprimés. En conséquence, les valeurs D et N sont filtrées du domaine de la variable x_7, par exemple.

Définition 6 *Soit $c_i : \Sigma \to \mathbb{R}$ des fonctions de coût d'affectation pour chaque $i \in \{1, \dots, n\}$ et Z une variable entière bornée, alors $\texttt{cost-regular}((X_i)_{i=1}^n, Z, \Pi, (c_i)_{i=1}^n)$ est satisfaite si et seulement si $(X_i)_{i=1}^n$ est un mot reconnu par Π et si $Z = \sum_1^n c_i(X_i)$. À noter que des coûts différents peuvent être appliqués en fonction de la transition empruntée, et non pas seulement en fonction du label de la transition. Chaque fonction c_i doit alors être spécifiée sur l'ensemble $\Delta \subseteq Q \times \Sigma$.*

Dans cette variante d'optimisation [DPR05], un coût $c_i(\sigma)$ est associé à chaque arc étiqueté σ dans la couche i du graphe Π^n. L'algorithme de filtrage calcule alors par programmation dynamique (toujours en deux passes avant/arrière) les plus courts et plus longs chemins du graphe afin de maintenir les bornes du domaine de la variable de coût Z. En même temps, il effectue la back-propagation du coût vers les variables d'état : (i) à chaque nœud, sont calculés et mémorisés les valeurs et prédécesseurs des plus courts et long chemins depuis l'état initial

I dans la couche 1 vers les états finaux A de la couche $n+1$, à la suite de quoi les bornes de Z peuvent être mises à jour ; (ii) les arcs qui n'appartiennent à aucun chemin de longueur comprise entre l_0 et u_0 sont identifiés et supprimés à la volée ; (iii) quand plus aucun arc de la couche i n'est étiqueté σ, la valeur σ est supprimée du domaine de la variable X_i.

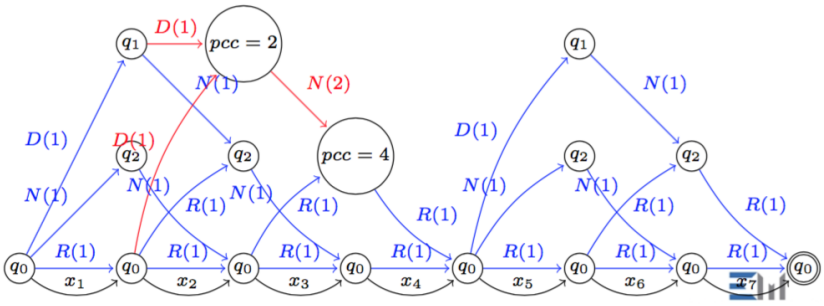


FIGURE 2.6 – Filtrage de `cost-regular` sur la variante avec coût de l'automate de la figure 2.5 et une variable de coût Z de borne inférieure 9 : l'arc rouge de la couche 3 est supprimé car il n'appartient à aucun chemin de longueur supérieure à 9. Les arcs rouges de la couche 2 sont supprimés en conséquence car ils n'appartiennent à aucun chemin vers un état final.

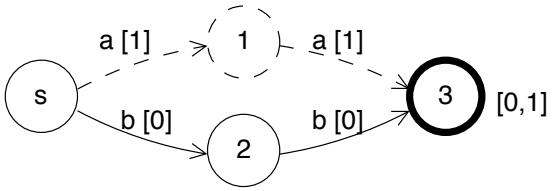
Comme pour `regular`, la complexité de l'algorithme est celle du premier parcours des chemins qui est linéaire en n et dans le nombre de transitions de Π. Les chemins sont ensuite mis à jour de manière incrémentale sur le graphe Π^n réduit.

Un cas particulier de `cost-regular` est la contrainte `knapsack` [Trio1]. En conséquence, maintenir la consistance d'arc est un problème NP-difficile et, à moins que P=NP, cela ne peut se faire au mieux qu'en temps pseudo-polynomial, i.e. polynomial en les les valeurs des bornes de la variable Z. De fait, `cost-regular` assure un niveau de consistance hybride. La définition de `cost-regular` révèle une décomposition naturelle en la conjonction d'une contrainte `regular` liée à une contrainte `knapsack` au moyen de n contraintes `element` chacune associant la valeur d'une variable X_i à son coût $c_i(X_i)$. `cost-regular` domine alors cette conjonction de contraintes quand la consistance aux bornes est appliquée aux variables de coût. En revanche, si la consistance d'arc est appliquée à `knapsack` (en temps pseudo-polynomial donc), les deux modèles sont incomparables comme le montrent les deux exemples ci-dessous. Le niveau de consistance réalisé par `cost-regular` est en fait équivalent à la consistance d'arc appliquée à la conjonction de contraintes suivante :

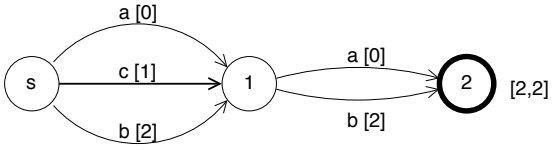
$$\texttt{regular}((X_i)_{i=1}^n, \Pi) \ \wedge \ Z \geq \sum_1^n c_i(X_i) \ \wedge \ Z \leq \sum_1^n c_i(X_i).$$

2.3.4 Applications

`regular` et `cost-regular` dérivent de contraintes qui ont été définies spécifiquement dans le contexte du séquencement d'activités, telles que `stretch` [Peso1] et `sequence` [RP97]. Via les langages réguliers, elles permettent d'exprimer une grande variété de règles de motifs. Les deux applications ci-après illustrent cette expressivité et la généricité de l'approche globale.



$x_1 \in \{a, b\}, x_2 \in \{a, b\}, z \in [0, 1]$.



$x_1 \in \{a, b, c\}, x_2 \in \{a, b\}, z \in [2, 2]$.

FIGURE 2.7 – À gauche, l'algorithme de `cost-regular` supprime les transitions étiquetées a et donc la valeur a des domaines de X_1 et X_2 réalisant la consistance d'arc. Ces valeurs ne sont pas filtrées par la conjonction de contraintes. À droite, en revanche, l'affectation $X_1 = c$ n'est pas consistante avec les bornes de Z mais les chemins, respectivement minimal et maximal, passant par la transition étiquetée c sont consistants avec les bornes, respectivement maximale $(1 \leq 2)$ et minimale $(3 \geq 2)$, de Z.

Un magasin de chaussures. Le premier cas d'études que nous avons traité présentait l'ensemble de règles suivantes (sur l'horizon d'une journée au pas de 15 minutes) :

1. une activité travaillée a une durée minimale de 1 heure ;
2. un déjeuner a une durée de 1 heure ;
3. une pause a une durée de 15 minutes ;
4. les repos sont en début et fin de journée ;
5. deux activités travaillées différentes ne sont pas consécutives ;
6. pauses, repos et déjeuner ne sont pas consécutifs ;
7. F_t définit l'ensemble des activités interdites à chaque pas de temps t ;
8. un horaire comprend entre 3 et 8 heures d'activités travaillées ;
9. tout horaire plein (au moins 6 heures d'activités travaillées) prévoit exactement un déjeuner et deux pauses ;
10. tout horaire partiel (de moins de 6 heures d'activités travaillées) prévoit exactement une pause.

Notre modèle de contraintes comprend 96 variables d'état X_t prenant leurs valeurs dans l'ensemble A des activités travaillées augmenté des activités p (pause), l (déjeuner) et o (repos). Les 6 premières règles se modélisent par un automate que l'on peut construire systématiquement pour toute valeur de $|A|$. La règle suivante définit des contraintes unaires $X_t \notin F_t$. Les trois dernières règles nécessitent de compter les valeurs de pause, déjeuner et repos, au moyen d'une contrainte `global-cardinality` par exemple associée à une contrainte logique pour chacune des deux dernières règles. En distinguant les sous-problèmes de génération d'horaires en temps plein et en temps partiel dans la génération de colonnes, on peut également se dispenser des contraintes logiques qui complexifient significativement le modèle. [6]

Un coût est associé à chaque activité travaillée à chaque pas de temps et, comme présenté dans le modèle de set-covering ci-dessus ((2.5)-(2.8)), il s'agit de satisfaire une courbe de charge

6. Ces instances ont été reprises dans différents travaux en faisant notamment l'impasse sur ces règles temps plein/partiel.

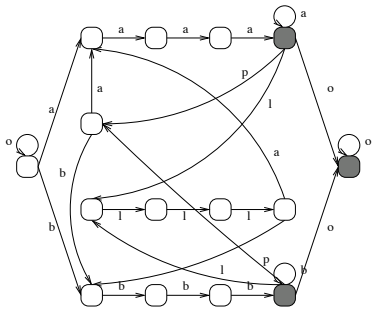


FIGURE 2.8 – Automate modélisant les 6 premières règles de l'exemple pour $A = \{a, b\}$.

minimale pour chaque activité travaillée tout en minimisant la somme cumulée des coûts d'affectation. Au niveau du sous-problème de génération de colonnes, la matrice des coûts spécifiée correspond aux coûts réduits d'affectation.

Notre implémentation générique du schéma de résolution reprend certains éléments connus pour accélérer la convergence de la génération de colonnes (qui se trouve ici particulièrement dégénérée) : le sous-problème n'est pas résolu à l'optimum, sauf éventuellement à la dernière itération pour la preuve d'optimalité et plus d'une solution sont ajoutées à chaque itération (typiquement 50 colonnes maximum). Ces éléments sont facilement applicables à la génération de colonnes par programmation par contraintes : il s'agit de gérer le sous-problème comme un problème de satisfaction et d'appliquer une stratégie de recherche compatible avec le critère d'optimisation. L'arbre de recherche est alors tronqué en mémorisant les 50 premières solutions réalisables rencontrées. Des techniques de *restart* peuvent également être employées pour diversifier les solutions. La principale difficulté d'implémentation d'un schéma de génération de colonnes par programmation par contraintes est inhérente au solveur et réside dans le lancement itératif du modèle de contraintes. Chaque itération de la génération de colonnes relance la résolution du sous-problème à zéro. Pourtant, seule la matrice des coûts réduits, soit les pondérations de `cost-regular`, est modifiée dans le modèle de contraintes et certaines informations obtenues lors d'une résolution (le résultat de la propagation initiale sans les coûts par exemple) pourraient être exploitées dans les itérations suivantes pour les accélérer. Nous n'avions pas mis en œuvre cette optimisation dans l'implémentation réalisée au-dessus des solveurs Ilog Cplex 9.0 et Ilog Solver 6.0.

Le nombre d'activités est limité en réalité dans ce problème, mais nous avons généré à partir des courbes de demande fournies, des instances fictives avec un nombre croissant d'activités (10 instances de chaque pour un nombre d'activités travaillées variant de 1 à 10) pour tester la scalabilité de l'approche exacte, à savoir le comportement de la borne inférieure obtenue par génération de colonnes et la solution optimale obtenue par le branch-and-price robuste décrit ci-avant. Il s'agit du benchmark *Shoe* employé ci-après. Dans nos expérimentations de l'époque (2005), la génération de colonnes à la racine convergeait en moyenne en moins d'une seconde pour les petites instances et en moins d'une minute pour les plus grandes (voir le détail dans la table 2.1). Le branch-and-price retournait un certificat d'optimal en moins de 2 heures seulement pour 20 des plus petites instances (avec $|A| \leq 3$). On notait alors que pour ces 20 instances la borne inférieure de la génération de colonnes à la racine était en fait égale

à l'optimum entier. Ceci laisse présumer de la qualité de la solution fractionnaire obtenue par génération de colonnes et de son intérêt pratique comme base d'une recherche locale pour une résolution approchée rapide du problème. Un branch-and-bound appliqué aux seules colonnes générées à la racine permettait d'obtenir en moins d'une heure des solutions réalisables avec un gap d'optimalité inférieur à 5% pour les petites instances et 10% pour les plus grandes.

Un réseau d'agences bancaires. Dans ce cas d'étude réel, les activités travaillées représentent les différentes agences annexes auxquelles les employés peuvent être affectés ainsi que les deux types d'activités (back-office et front-office) de l'agence centrale. Une activité non-travaillée supplémentaire (en plus de repos, pause, dîner et souper) est considérée pendant le transfert d'une agence à l'autre. La planification se fait à la semaine, avec un pas de 15 minute, et considère des règles différentes pour les horaires de temps plein (35h par semaine) et de temps partiel (entre 20h et 30h par semaine). Parmi l'ensemble considéré, on trouve les règles suivantes :

1. la durée de transfert entre deux agences est donnée ainsi que la durée des repas
2. on ne peut travailler après 17h plus d'une fois par semaine
3. une pause est requise dans la journée d'un temps partiel travaillée pendant au moins 5h et au plus 6h
4. il y a au plus 1 transfert par jour et toujours d'une annexe vers le central
5. il y a au plus 1 changement d'activité par jour au sein de l'agence centrale

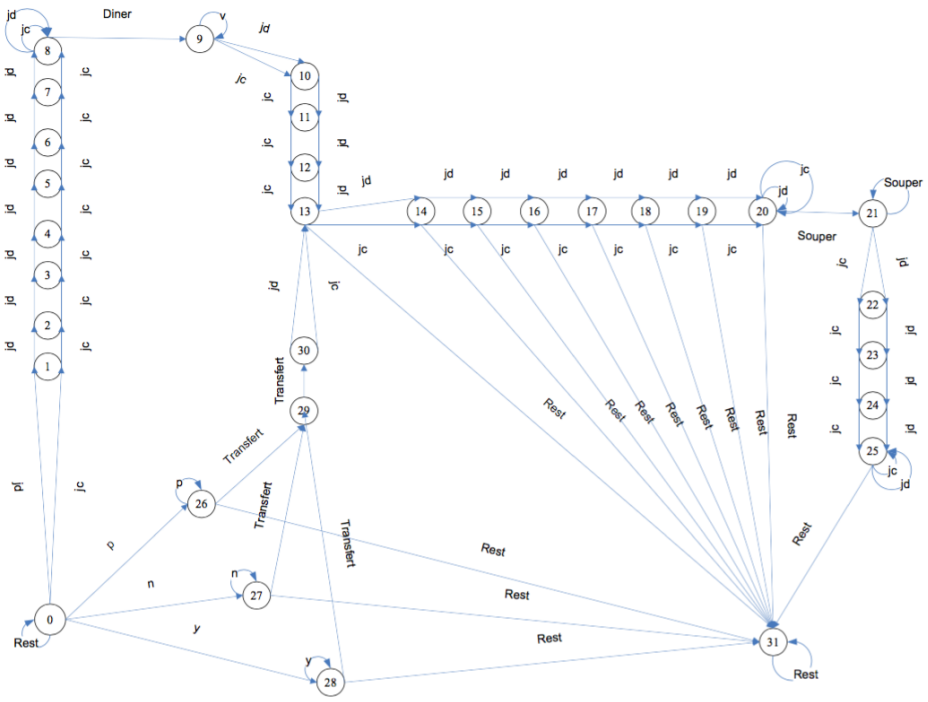


FIGURE 2.9 – Automate modélisant une journée temps plein pour un réseau d'agences bancaires.

La figure 2.9 décrit l'automate utilisé pour représenter une journée d'un temps plein. Nos expérimentations montraient qu'il était préférable de représenter chaque journée indépendamment (sur des séquences de 96 variables donc) plutôt que d'employer une contrainte globale sur l'ensemble de la semaine (480 variables) bien que l'automate-semaine soit une simple concaténation des automates-jours et que la décomposition inhibe les règles de balance du nombre d'heures travaillés par jour.

Dans ce contexte, des pénalités de sous-charge et de sur-charge sont considérées en plus des coûts d'affectation. Le problème maître de génération de colonnes se présente alors comme un problème de set-partitionning. Le modèle de coût du sous-problème n'est lui pas impacté.

Ici la taille du problème et la forme spécifique du problème maître font que la génération de colonnes ne converge pas en temps raisonnable. Dans notre implémentation générique, celle-ci est alors tronquée et une solution entière est calculée par branch-and-bound (tronqué également) sur les seules colonnes générées. Elle a permis d'obtenir, de manière systématique et en temps bien plus limité, des solutions similaires à celles générées manuellement par l'administration bancaire.

2.3.5 Branchement orienté contrainte

Notre implémentation bénéficie de quelques optimisations tout en restant la plus générique possible. L'un des ingrédients les plus impactants réside dans la stratégie de branchement du sous-problème d'optimisation par contraintes. C'est en effet un élément important de la solution puisque le sous-problème est résolu itérativement, en moyenne plus d'une centaine de fois sur les instances de la table 2.1 par exemple. Hormis à la dernière itération où la preuve d'optimalité est requise, il s'agit à chaque itération de produire une cinquantaine de solutions de coûts réduits les plus négatifs possibles. Cet aspect nous a mené à exploiter le support de la contrainte d'optimisation `cost-regular`, centrale dans notre modèle, à la fois pour réduire l'espace de recherche par filtrage mais également pour guider la recherche.

En effet, la contrainte `cost-regular` maintient en interne un plus court chemin dans le graphe en couche Π^n. Dans notre application, un tel chemin correspond à un horaire « pratiquement réalisable et optimal » : il est valide pour les règles modélisées par l'automate mais potentiellement pas pour les règles modélisées dans d'autres contraintes (notamment les règles de comptage modélisée par une `global-cardinality`) et il est de coût minimum pour l'ensemble des horaires de ce type. De sorte, on dispose à chaque nœud de l'arbre de recherche d'une solution relâchée du modèle de contraintes, à l'instar de la solution de relaxation continue en programmation linéaire en nombres entiers. De même que la solution de relaxation continue est intensivement exploitée dans les stratégies de branchement de type *strong branching*, les plus couramment utilisées dans les implémentations de branch-and-bound actuelles, nous proposons d'exploiter la solution relâchée de `cost-regular` dans notre stratégie de branchement, et plus précisément dans l'heuristique de choix de valeur. Il s'agit d'instancier, à chaque branchement, une variable X_i à la valeur de l'arc dans la couche i du plus court chemin dans le graphe Π^n, soit la valeur de X_i si la solution optimale de `cost-regular` était réalisable pour le modèle complet.

Nos résultats expérimentaux sur les instances *shoe*, partiellement reproduits dans la table 2.1, montrent l'intérêt indéniable de cette heuristique sur le temps de calcul en com-

paraison à une heuristique classique consistant à sélectionner le couple variable-valeur de coût minimal.

TABLE 2.1 – Résolution des instances *Shoe* de la section 2.3.4 (10 instances par groupe où le nombre d'activité travaillé $|A|$ varie de 1 à 10) par génération de colonnes hybride. Le sous-problème de contraintes est résolu avec l'heuristique de choix de la valeur de coût réduit minimum (*min rc*) ou le choix de la valeur donnée par `cost-regular`. Toutes les instances sont résolues avec la seconde en moins d'une heure mais pas avec la première. Temps moyen de la génération de colonne (*CPU*) et d'un sous-problème (*CPU CP*), écart moyen de la borne inférieure à la borne supérieure obtenue par branch-and-bound sur les seules colonnes générées à la racine ($\Delta LB/UB$), nombre d'itérations de la génération de colonnes (*#iter*), nombre de colonnes générées (*#col*).

$\|A\|$	*min rc*		*`cost-regular` heuristic*				
	CPU	CPU CP	CPU	CPU CP	$\Delta LB/UB$	#iter	#col
1	1.9	0.1	0.4	0.02	2.5%	19	889
2	6.1	0.1	3.7	0.03	2.8%	48	2340
3	16.7	0.2	2.0	0.03	2.3%	52	2550
4	92.9	0.6	12.5	0.04	3.3%	103	5063
5	108.4	0.7	6.2	0.04	5.1%	86	4288
6	355.6	1.6	13.8	0.05	4.7%	130	6493
7	9/10 résolues		18.4	0.06	6.1%	137	6839
8	9/10 résolues		25.4	0.07	6.0%	155	7736
9	0/10 résolues		25.9	0.07	7.3%	155	7741
10	0/10 résolues		42.0	0.09	8.7%	179	8974

L'idée d'exploiter le support d'une contrainte globale, en dehors du filtrage, pour guider la recherche, a été proposée par FOCACCI, LODI et MILANO [FLM99] notamment dans le but de mimer les solveurs d'optimisation qui exploitent les informations des relaxations (valeurs de bornes, fonctions de gradient) pour guider les heuristiques de décision pendant la recherche. Cette idée a été peu ou pas suivie jusqu'à très récemment [BCH15b]. À travers nos travaux, nous sommes convaincus de son intérêt, notamment dans le contexte des contraintes globales difficiles et d'optimisation. Nous revenons sur ce type de contraintes et discutons notamment de ce point dans la section 3.2.2.2.

2.3.6 Contributions

Encore une fois, les contributions de ces travaux sont d'ordre à la fois applicatif, méthodologique et technique :

Concernant l'application de planification de personnel, nous avons proposé une approche de résolution axée à la fois sur l'efficacité et la flexibilité pour une classe de problèmes relativement peu étudiée définie par un horizon de temps discretisé et des règles d'ordonnan-

cement complexes. Ces travaux ont essaimé dans la communauté « contraintes » avec, par exemple, l'intégration de contraintes de langage dans une recherche locale [QR10] ou linéarisées et intégrées dans un programme mathématique en nombres entiers [CGR11]. Plusieurs études expérimentales, dont celles-ci, ont été menées sur les instances *shoe* que nous avons créées mais elles traitent de variantes du problème dans le modèle de couverture (minimisation du nombre d'employés [KS08], minimisation des pénalités de sous ou sur-couverture [QR10 ; CGR11 ; CGR13]) ou dans les règles de travail (une pause par demi-journée maximum [QR10], absence des temps partiels [QW07]), prévenant la possibilité d'une comparaison expérimentale juste. Dans [CGR13], Côté et al. adoptent un schéma de résolution très proche du notre, pour un problème plus général où les règles de travail des employés peuvent être différentes. Dans cette méthode, le sous-problème est directement résolu par programmation dynamique mais sur un graphe obtenu depuis une autre contrainte globale de langage, la contrainte `grammar` dont le langage est spécifié non pas par un automate fini mais par des règles de production. La comparaison expérimentale avec nos résultats publiés dans [DPR06], bien qu'abusive car les modèles de couverture sont différents (sans parler de l'évolution logicielle entre les deux expérimentations), semble indiquer que le Branch-and-Price de [CGR13] est plus scalable que le notre mais cela est principalement dû, je pense, à une stratégie de branchement plus intelligente. En intégrant cette optimisation dans notre approche, la comparaison expérimentale permettrait en sorte d'évaluer le surcoût lié à la résolution du modèle de contraintes – comme celui-ci repose principalement sur la programmation dynamique de `regular` – et donc le surcoût lié à l'hybridation. Ce surcoût pourrait alors être mis en balance avec la souplesse apportée face à la complexité de formuler le graphe des règles de travail et de mettre en œuvre l'algoritme de programmation dynamique. De même, la communauté « recherche opérationnelle » avait et a depuis présenté de nombreuses approches de Branch-and-Price par programmation dynamique, souvent sur des variantes du problème de planification de personnel et des instances distinctes. Il serait intéressant de voir si ce schéma d'hybridation serait compétitif en efficacité *et* en adaptabilité.

D'un point de vue méthodologique et technique, notre étude fait partie des quelques applications du couplage de la génération de colonnes et de la programmation par contraintes qui ont été réalisées au début des années 2000 et a participé ainsi à valider la méthode en mettant plus encore l'accent sur son potentiel de flexibilité. Avec la définition de `cost-regular`, elle a participé également à l'essor des contraintes de langage vers la fin des années 2000 qui ont aussi cherché à étendre l'expressivité des contraintes de langage rationnel mais en les généralisant aux langages d'ordre supérieur (`grammar` [QW06 ; Sel06]). Enfin, cette contrainte appartient aux classes restreintes des contraintes d'optimisation et des contraintes NP-difficiles. Ces aspects sont discutés plus avant dans les sections du chapitre suivant sur les méthodes de décomposition hybrides et les contraintes de langage et d'optimisation.

2.4 MultiCostRegular : hybridation pour le filtrage et la modélisation

Cette section présente des travaux issus de la thèse de Julien Menana [Men11] que j'ai encadrée de 2008 à 2011 et parus en partie dans [MD09]. Ces travaux sont motivés par et font directement suite à mes travaux de post-doc sur la planification de personnel présentés

dans la section précédente, où ils poussent plus avant les concepts d'agrégation et d'aide à la modélisation, empruntant pour cela aux outils de la recherche opérationnelle et des automates.

Dans nos précédents travaux, nous proposions d'agréger les règles d'ordonnancement au sein d'une même contrainte globale, afin d'établir une meilleure interaction et accélérer la résolution. En représentant ces règles par un modèle compact d'automate, nous proposions par ailleurs, à la suite de [LP04], d'aider à la prise en main d'une telle solution d'optimisation par un public non (totalement) initié. Nos expérimentations montraient cependant parfois un défaut de propagation entre la contrainte `cost-regular` et la contrainte `global-cardinality`, limitant le nombre d'occurrences des activités dans un horaire. Dans le cadre de la thèse de Julien Menana, nous avons travaillé à corriger ce défaut en agrégeant règles d'ordonnancement et règles d'occurrences au sein d'une seule contrainte globale. Nous avons abouti à la définition de `multicost-regular`, une meta-contrainte à la vaste expressivité, dont la portée dépasse le cadre de la planification de personnel.

`multicost-regular` étend `cost-regular` à un automate multi-pondéré et à plusieurs variables de coûts, permettant d'associer différents coûts ou compteurs à l'affectation des variables d'état.

Définition 7 *Soient* $c_i^k : \Sigma \rightarrow \mathbb{R}$ *des fonctions de coût d'affectation et* Z_k *des variables entières bornées pour* $k \in \{0, \ldots, p-1\}$, $i \in \{1, \ldots, n\}$, *alors* `multicost-regular`$((X_i)_{i=1}^n, (Z_k)_{k=0}^{p-1}, \Pi, (c_i^k)_{(i,k)=(1,0)}^{(n,p-1)})$ *est satisfaite si et seulement si* $(X_i)_{i=1}^n$ *est un mot reconnu par* Π *et si* $Z_k = \sum_1^n c_i^k(X_i)$ *pour tout* $k \in \{0, \ldots, p-1\}$.

NP-difficile, cette contrainte est équipée d'un algorithme de filtrage incomplet, présenté dans la section 2.4.1, basé sur une méthode de décomposition de la programmation mathématique : la relaxation lagrangienne. Nous avons également assortie cette meta-contrainte de procédures de modélisation, basées sur la manipulation des automates pondérés. Adaptées au contexte de la planification de personnel sans toutefois lui être réservées, ces procédures sont présentées en section 2.4.2.

2.4.1 Filtrage par relaxation lagrangienne

La contrainte `multicost-regular` s'identifie à un problème de plus court (et de plus long) chemin avec $p-1$ contraintes de ressources dans le graphe en couches Π_n, support de `regular` et `cost-regular` (voir 2.5). Ce problème d'optimisation combinatoire classique (RCSPP) est $\mathcal{NP}$-difficile. Il reçoit principalement deux traitements dans la littérature : la programmation dynamique et la relaxation lagrangienne. Nous avons naturellement envisagé les deux approches pour la conception d'un algorithme de filtrage.

La programmation dynamique ne semblait pas pertinente dans une version incrémentale, compatible avec l'algorithme de backtracking. En effet, après un ou une série de backtracks, l'état des domaines des variables doit être rétabli, i.e. l'effet du filtrage de la contrainte annulé. Ce peut être fait de deux manières : par enregistrement de l'état ou par recalcul. Le recalcul est coûteux en temps et l'enregistrement n'est pas envisageable ici car la programmation dynamique repose sur l'accumulation de « labels » intermédiaires à chaque nœud du graphe, et la présence des contraintes de ressources dans le RCSPP démultiplie le nombre de labels enregistrés. À noter que dans le cas du problème de plus court chemin simple, la programmation

dynamique ne génère qu'un label par nœud. Elle est ainsi à la base du filtrage de la contrainte `cost-regular`.

La relaxation lagrangienne se prête particulièrement bien au filtrage : l'approche, formalisée par Sellmann [Sel04], consiste à appliquer la technique de fixation des variables basée sur les coûts réduits [Wol98] (ce qui correspond ici à supprimer des arcs du graphes) à chaque itération de la résolution du dual lagrangien qui, en parallèle, resserre les bornes de la variables d'optimisation. Nous avons ainsi développé un algorithme de filtrage basé sur la relaxation lagrangienne pour la contrainte `multicost-regular`.

2.4.1.1 Application

Un modèle linéaire en variables binaires pour le RCSPP/RCLPP dans le graphe Π_n est le suivant :

$$\min / \max \sum_{i=1}^{n} \sum_{q\in Q} \sum_{\sigma\in\Sigma} c_i^0(\sigma) x_{iq\sigma} \tag{2.14}$$

$$\text{s.t. } \underline{Z_k} \leq \sum_{i=1}^{n} \sum_{q\in Q} \sum_{\sigma\in\Sigma} c_i^k(\sigma) x_{iq\sigma} \leq \overline{Z_k} \qquad \forall k \in \{1,\dots,p-1\} \tag{2.15}$$

$$\sum_{q\in Q} \sum_{\sigma\in\Sigma|(q,\sigma,q')\in\Delta} x_{i-1q\sigma} = \sum_{\sigma\in\Sigma} x_{iq'\sigma} \qquad \forall i \in \{2,\dots,n\}, q' \in Q \tag{2.16}$$

$$\sum_{\sigma\in\Sigma} x_{1q\sigma} = 1 \qquad \forall q \in I \tag{2.17}$$

$$\sum_{q\in Q} \sum_{\sigma\in\Sigma|(q,\sigma,q')\in\Delta} x_{nq\sigma} = 1 \qquad \forall q' \in A \tag{2.18}$$

$$x_{iq\sigma} \in \{0,1\} \qquad \forall i \in \{1,\dots,n\}, q \in Q, \sigma \in X_i. \tag{2.19}$$

Les contraintes de conservation du flot (2.16)-(2.18) déterminent l'existence d'un chemin de I vers A de longueur n (unique car $|I| = 1$), tel que $x_{iq\sigma} = 1$ si et seulement si $(q,\sigma,q') \in \Delta$ est le i-ème arc.

Sous la relation $x_{iq\sigma} = 1 \iff X_i = \sigma$, il existe une correspondance 1-à-1 entre les solutions de `multicost-regular` et les solutions réalisables de RCSPP/RCLPP dont la valeur objectif est dans l'intervalle de définition de Z_0.

Le sous-problème lagrangien associé à des multiplicateurs lagrangiens positifs quelconques $\lambda \in \mathbb{R}^{2p-1}$ s'obtient par dualisation des contraintes compliquantes de ressources (2.15). Il s'agit alors d'un simple problème de plus court (respectivement long) chemin dans le graphe Π_n et fournit une borne inférieure $\underline{z_0^\lambda}$ (respectivement supérieure $\overline{z_0^\lambda}$) de l'optimum du RCSPP (respectivement RCLPP) :

$$\underline{z_0^\lambda} = \min \sum_{i=1}^{n} \sum_{q\in Q} \sum_{\sigma\in\Sigma} (c_i^0(\sigma) + \sum_{k=1}^{p-1} (\lambda_+^k - \lambda_-^k) c_i^k(\sigma)) x_{iq\sigma} + \sum_{k=1}^{p-1} (\lambda_-^k l_k - \lambda_+^k u_k) \quad s.t. \quad (2.16)-(2.18),\ x \in \{0,1\}^{nQX}.$$

Le calcul de ces bornes fournit donc un premier filtrage des bornes de la variable Z_0 :

$$\underline{z_0^\lambda} \leq Z_0 \leq \underline{z_0^\lambda}.$$

L'ordre des ressources étant arbitraire, on peut l'intervertir pour appliquer le même raisonnement sur les autres variables Z_k, $\forall k \in \{1, \ldots, p-1\}$. Par ailleurs, la back-propagation consiste ici à identifier et supprimer les arcs du graphe Π_n qui n'appartiennent à aucun chemin de coût compris entre $\underline{Z_0}$ et $\overline{Z_0}$. L'algorithme de filtrage de `cost-regular` [DPR05] peut ainsi être mis en œuvre pour le filtrage des bornes et la back-propagation : il calcule les plus courts et plus longs chemins de Π_n avec la matrice de coûts $(c^0 + \sum_{k=1}^{p-1}(\lambda_+^k - \lambda_-^k)c^k)$.

L'algorithme de filtrage que nous proposons pour `multicost-regular` consiste ainsi en l'appel à celui de `cost-regular` pour différentes valeurs de k et de λ. D'un point de vue algorithmique, il est à noter que les calculs de plus court/long chemins pour différentes valeurs de k et λ, s'effectuent toujours sur le graphe Π_n dont seuls les coûts changent. Pour le choix des multiplicateurs λ, nous appliquons une variante simple de l'algorithme du sous-gradient avec des paramètres fixés de manière empirique. L'intérêt de l'approche de filtrage par relaxation lagrangienne est qu'il n'est pas nécessaire d'atteindre les multiplicateurs lagrangiens optimaux pour filtrer. On s'abstrait ainsi d'un éventuel défaut de convergence de l'algorithme du sous-gradient. Dans notre implémentation, nous avons ainsi fixé le nombre maximal d'itérations du sous-gradient, i.e. le nombre d'appels à `cost-regular`, mais l'algorithme termine généralement seul en 5 ou 6 itérations.

2.4.1.2 Évaluation

Nous avons évalué la performance de `multicost-regular`, implémentée dans Choco 2, pour la construction d'horaires en planification de personnel, en utilisant les instances du benchmark *Shoe* détaillé en section 2.3.4. L'horaire d'un employé est défini par une séquence de 96 périodes affectées à une activité travaillée prise dans un ensemble A ou une activité non travaillée (repas, pause, repos). Il est soumis à un ensemble de règles d'ordonnancement et d'occurrences. La contrainte `multicost-regular` internalise ces règles d'occurrences, alors qu'elles étaient externalisées dans une contrainte `global-cardinality` dans notre précédent modèle basé sur `cost-regular`. Le modèle `multicost-regular` est ainsi spécifié sur le même automate que `cost-regular` (représenté figure 2.8 pour $|A| = 2$) mais avec 3 pondérations en plus du coût d'affectation que l'on souhaite minimiser.

Nous avons comparé ces deux modèles sur des instances générées pour $|A|$ variant entre 1 et 50, la taille de l'automate variant alors de 11 états et 15 transitions à 207 états et 505 transitions. Nous avons généré 20 instances pour chaque valeur de $|A|$ en modifiant la matrice des coûts d'affectation et testé les deux modèles sous l'algorithme de backtracking par défaut de Choco 2. La table 2.2 présente ces résultats repris de [MD09] : le pourcentage d'instances résolues en moins de 10 minutes, le temps moyen pour la preuve d'optimalité, le temps moyen pour trouver la meilleure solution, le nombre de nœuds de branchement ouverts. Ces résultats expriment une claire supériorité du modèle intégré en une contrainte `multicost-regular`. On voit notamment que les temps de résolution augmentent logiquement avec la taille de l'automate, mais de manière nettement plus modérée que pour le modèle décomposé.

	Modèle CR + GCC				Modèle MCR			
n	résolus	preuve(s)	meilleure(s)	#nœuds	résolus	preuve(s)	meilleure(s)	#nœuds
1	100%	0.3	0.2	225	100%	0.0	0.0	41
2	100%	0.6	0.3	393	100%	0.1	0.1	68
4	100%	2.9	2.3	1199	100%	0.2	0.1	67
8	100%	17.9	13.2	3597	100%	0.3	0.2	52
10	100%	50.0	47.7	7615	100%	0.4	0.4	63
15	100%	58.1	47.1	6233	100%	0.8	0.7	63
20	100%	58.1	44.0	4577	100%	1.2	1.0	64
30	80%	153.1	127.4	6080	100%	1.8	1.5	62
50	40%	460.0	65.6	6747	100%	5.0	4.8	65

TABLE 2.2 – Comparaison expérimentale du modèle intégré MCR en une contrainte `multicost-regular` et du modèle décomposé CR+GCC en deux contraintes `cost-regular` et `global-cardinality`.

2.4.2 Agrégation et relaxation automatiques de règles de séquencement

Les contraintes de langage sont des meta-contraintes en ce sens qu'elles généralisent, de manière exacte ou relâchée, différents types de contraintes. Je reviendrai plus en détail sur cet aspect dans la section 3.1.2.3. A contrario, par rapport à des contraintes dédiées, la difficulté est plus grande pour l'utilisateur de représenter son problème par le descripteur du langage de la contrainte, en l'occurrence dans le cadre de `multicost-regular` de concevoir un automate multi-pondéré. Ajoutant à la difficulté, cette représentation n'est souvent pas unique, comme ici où certaines restrictions peuvent s'exprimer par différents automates ou par une pondération. La complexité de l'algorithme étant proportionnelle à la taille de l'automate déployé, il s'agirait dans l'idéal de spécifier un automate de taille minimale reconnaissant uniquement les mots du langage de la taille de la séquence de variables. Cependant, cette représentation n'est pas fonctionnelle comme l'automate est généralement ainsi moins lisible et, a fortiori, plus difficilement modifiable par l'utilisateur.

Les automates sont cependant de formidables outils de représentation et de manipulation des langages, pour lesquels on dispose notamment d'algorithmes d'intersection, de minimisation et de construction à partir d'expressions rationnelles. Dans le cadre de la thèse de Julien Menana, nous avons développé plusieurs procédures basées sur ces outils afin d'aider l'utilisateur à modéliser tout ou partie de son problème au sein d'une contrainte `multicost-regular`. Elles automatisent trois étapes de modélisation et réalisent par ce biais un prétraitement du modèle pour accélérer sa résolution :

— traduction de règles de séquencement, avec ou sans comptage, en automates pondérés individuels ;
— traduction des fonctions régulières de pénalités de violation en pondérations dans le cadre de règles souples ;
— agrégation par intersection et minimisation en un automate multi-pondéré.

Ces trois étapes, décrites en détail dans le manuscrit de thèse [Men11], sont résumées ci-dessous. Une implémentation de cet outil d'aide à la modélisation, basée sur la bibliothèque Java d'algorithmes sur les automates *dk.brics* [bri], est disponible à `https://github.com/sofdem/chocoETP`. Elle supporte plusieurs langages plus ou moins haut niveaux de spécification des règles à modéliser et est ainsi compatible avec 5 différents benchmarks de la

planification de personnel et notamment du Nurse Scheduling Problem (NSP).

2.4.2.1 Règles de séquencement

Pour illustrer le principe de construction automatique d'automates pondérés à partir de règles de séquencement, prenons l'exemple d'un problème de Nurse Scheduling où chaque infirmier, chaque jour, doit être affecté à un quart de travail – M(atin) ou S(oir) – ou mis au R(epos). L'horaire hebdomadaire d'un infirmier se représente alors comme un mot $X_1X_2 \dots X_7$ sur l'alphabet $\{M, S, R\}$. Les règles du travail et les préférences personnelles régissent le séquencement possible des activités d'un infirmier et définissent ainsi le langage de l'ensemble des horaires possibles. Le problème de planification se pose, avant même d'évoquer la complexité de résolution, sur la difficile modélisation de ces règles. En pratique, c'est souvent l'infirmière en chef du service qui construit ces horaires et elle y passe un temps considérable. Elle pourrait adopter un outil de planification à la condition uniquement que celui-ci ait une ergonomie suffisante pour exprimer ces règles de séquencement. Notre outil de modélisation des règles exprimées dans un langage haut-niveau vise cet objectif.

Expressions rationnelles. Les règles les plus simples portant sur la (non-)affectation d'une activité à une date fixe se traduisent par des contraintes unaires (*repos le dimanche* $X_7 = R$, *congé le mardi matin* $X_2 \neq M$). Les autres règles consistent pour la plupart à forcer ou limiter l'apparition ou le nombre d'apparitions d'un motif d'activités dans l'horaire d'un infirmier, à date fixe ou variable dans un intervalle de temps donné. Souvent, ce motif peut facilement s'écrire sous la forme d'une expression rationnelle. On utilise ci-après les éléments de syntaxe suivants pour décrire une expression rationnelle :

- « | », l'opérateur de choix ($M|S$ *matin ou soir*)
- « $*$ », l'opérateur de répétition universel ($R*$ *une séquence de zéro ou plusieurs repos*)
- « $\{k\}*$ », l'opérateur de répétition limité (R^2 *deux repos consécutifs*)
- $A = (M|S|R)$, un littéral quelconque de l'alphabet : A^* est l'ensemble des mots sur l'alphabet et $A^{\{k\}}$ l'ensemble des mots de taille k.
- « $\sim$ », l'opérateur de complémentarité : $\sim L$ est $A^* \setminus L$ l'ensemble des mots sur l'alphabet exceptés ceux appartenant au langage L

Le motif $(M|S)^*R$ désigne, par exemple, une séquence de quart travaillés de longueur quelconque, éventuellement nulle, suivie d'un repos. Une règle forçant ou interdisant l'apparition d'un tel motif dans l'horaire peut se traduire au moyen d'une expression rationnelle représentant le langage auquel le mot $X_1X_2 \dots X_7$ doit appartenir. Par exemple, la règle « *un repos obligatoire entre mercredi et samedi* » se traduit par l'expression rationnelle $A^{\{2\}}(M|S)^*RA^*A$. La règle « *un repos obligatoire dans la semaine* » se traduit par $A * RA^*$. À partir d'une expression rationnelle, les méthodes morphologique ou des dérivées, par exemple, permettent de dériver un automate déterministe fini équivalent Π. La règle s'exprime alors dans un modèle de contraintes par `regular`$((X_i)_{i=1}^7, \Pi)$. À noter que les implémentations de `regular`, `cost-regular` et `multicost-regular` de Choco permettent de spécifier directement le langage par une expression rationnelle suivant la syntaxe décrite ci-dessus.

Compteurs et pondérations. Les deux règles précédentes peuvent alternativement se modéliser par une contrainte `cost-regular`$((X_i)_{i=1}^7, Z, \Pi, (c_i)_{i=1}^7)$ avec Z une variable bornée $Z \in [1,7]$ comptant le nombre de Repos (et forçant au moins une occurrence), Π l'automate universel (i.e. A^*) acceptant tous les mots sur l'alphabet, et une pondération nulle sur toutes les transitions ($c_i(a) = 0 \ \forall i = 1,\dots,7 \ \forall a = M,S,R$) exceptées : $c_i(R) = 1 \ \forall i = 3,\dots,6$ pour la première règle, et $\forall i = 1,\dots,7$ pour la seconde règle. Dans notre implémentation actuelle de traduction automatique, l'expression rationnelle est préférée au compteur quand la borne inférieure (resp. supérieure) du nombre d'occurrences d'une activité est contrainte à au moins 2 (resp. à au plus $n-2$). Ainsi la règle « *au plus deux repos dans la semaine* » est traduite par une variable de compteur $Z \in [0,2]$ plutôt que par l'expression rationnelle $(M|S)^*(M|S|R)(M|S)^*(M|S|R)(M|S)^*$.

Stretch. D'autres règles s'expriment difficilement par une expression régulière, mais répondent à un automate que l'on peut facilement construire de manière algorithmique. Ainsi, la règle générale dite de *stretch* « *la longueur d'une suite d'activité a est comprise entre l et u* » et son équivalent à date fixe « *la longueur d'une suite d'activité a à partir du jour i est comprise entre l et u* » se modélisent par une `regular` associée aux automates génériques suivant :

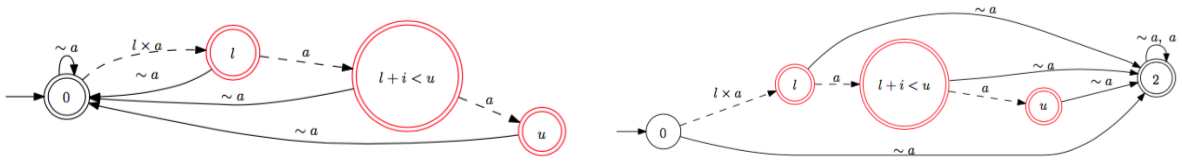


FIGURE 2.10 – Automates pour les règles sur la longueur des stretchs à date variable (gauche) ou fixe (au temps l à droite).

À ces automates, on peut également associer une pondération, pour compter le nombre d'occurrences d'un stretch par exemple : sur les automates précédents, il s'agit d'ajouter une fonction de coût 1 sur les transitions sortantes du stretch et de coût 0 sur toutes les autres transitions. Toute règle limitant le nombre d'occurrences d'un stretch peut alors être modélisée par une variable bornée associée à la séquence par une contrainte `cost-regular`.

Forcer ou interdire un motif. Ce principe de construction d'automates, via éventuellement une expression régulière, nous l'avons généralisé à l'ensemble des règles forçant ou interdisant l'apparition d'un motif rationnel quelconque ou limitant le nombre d'occurrences. Soit, par exemple, le motif $\sigma = (M|R)RR$, « *un matin ou un repos suivi de deux jours de repos* ».

Forcer l'apparition du motif, à date variable ou à date fixe, consiste à modéliser l'ensemble des mots/horaires contenant ce motif, soit, respectivement : $A^*\sigma A^*$ « *deux jours de repos consécutifs sont nécessaires et précédés par un repos ou un quart de matin* », ou $A^{\{4\}}\sigma$ « *le week-end est nécessairement chômé et précédé d'un quart du matin ou d'un repos le vendredi* ».

La forme générale d'interdiction est plus délicate comme elle repose sur l'opérateur de complémentarité : $\sim\sigma$ désigne l'ensemble des mots sur l'alphabet différents de σ, tandis que $\sim(A^*\sigma A^*)$ désigne l'ensemble des mots ne contenant pas le motif σ. Elle traduit ici la règle « *un*

quart de soir est nécessaire avant deux jours de repos consécutifs. L'équivalent à date fixe $A^{\{k-1\}} \sim (\sigma A^*)$ désigne l'ensemble des mots ne contenant pas le motif au temps k.

Il faut noter la subtile différence entre la règle forçant l'apparition du motif σ (*deux jours de repos consécutifs sont nécessaires et un quart de soir avant est interdit*) et la règle *interdisant un quart de soir avant deux repos consécutifs* et qui correspond à l'expression rationnelle $\sim(\sim(A^*(M|R))RRA^*)$. Du point de vue de l'utilisateur (même aguerri), formuler la seconde est notablement plus complexe que la première. Nous avons ainsi construit l'expression générale pour les règles du types « *si un motif σ apparaît alors il est directement* (ou *indirectement*) *suivi* (ou, *précédé*) *du motif σ'*, soit respectivement :

$$\sim(A^*\sigma \sim(\sigma' A^*)), \sim(A^*\sigma \sim(A^*\sigma' A^*)), \sim(\sim(A^*\sigma')\sigma A^*), \sim(\sim(A^*\sigma' A^*)\sigma A^*).$$

Limiter le nombre d'occurrences d'un motif. De la même manière, l'expression $\sim(A^*SMA^*SMA^*SMA^*)$ limite par exemple le nombre d'occurrences du motif SM (*quart de soir suivi d'un quart de matin*) à au plus 3. Cependant, la taille de l'automate correspondant devient dépendante de la borne. Pour le comptage de motif en général, nous proposons, comme pour le comptage de stretch d'avoir recours à un automate pondéré. Il s'agit alors d'affecter un coût 1 aux transitions sortantes du motif apparaissant dans l'automate, et un coût nul aux autres transitions. La difficulté ici est de construire, de manière automatique et dans le cas général, un automate universel dans lequel on puisse identifier le motif et ses transitions sortantes. Des algorithmes efficaces, tels Aho-Corasick, identifient un motif dans un texte, mais ils sont difficilement applicables à un automate. Nous proposons ici la méthode constructive générale suivante pour compter un motif σ. Soit α un littéral n'appartenant pas à l'alphabet, on considère l'automate construit à partir de l'expression régulière :

$$(A^*(\sigma\alpha^*)^*)^*.$$

Cet automate présente nécessairement une transition α à un état reconnaissant la fin du motif σ. Il suffit alors d'affecter un coût de 1 aux transitions menant à un tel état puis de supprimer les transitions α. L'opération de suppression est sûre (i.e. l'automate obtenu est universel) du fait que nous avons concaténé le motif α^*, et non α, dans l'expression régulière précédente. Surtout, l'opération reste valide quand elle doit être exécutée après intersection ou minimisation de l'automate (voir section 2.4.2.3). On associe au final à ce nouveau coût, une variable compteur dont le domaine est borné par les nombres d'occurrences min et max du motif.

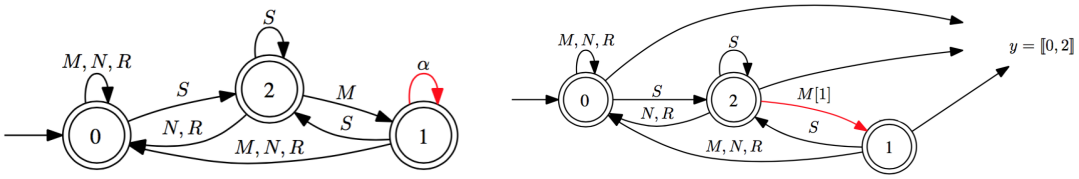


FIGURE 2.11 – Automate universel identifiant le motif SM, obtenu à partir de l'expression régulière (gauche), puis après l'opération de pondération limitant l'apparition du motif 2 fois au plus (droite).

2.4.2.2 Relaxation et pénalités de violation

Dans les problèmes sur-contraints, les règles dites *souples* sont autorisées à ne pas être satisfaites. Des pénalités, que l'on souhaite minimiser, sont alors encourues pour chaque violation et selon le degré de la violation. Évaluer le coût de pénalité pour une solution consiste à identifier dans la séquence un motif témoin de la violation, d'en compter les occurrences ou le degré de violation, et d'y associer un coût via la fonction de pénalité donnée.

Dans le cas où l'on pénalise le nombre d'occurrences d'un motif régulier quelconque, une contrainte `cost-regular` permet, comme décrit précédemment, de modéliser ce nombre d'occurrences via un automate pondéré. Ce nombre y est alors lié à la valeur de la pénalité z au moyen d'une contrainte externe de la forme $z = f(y)$ avec f la fonction de pénalité. Aucun critère de régularité sur la fonction f n'est requis par cette modélisation mais elle influera sur l'efficacité de propagation de cette seconde contrainte. Par ailleurs, quand la fonction n'est pas injective (ce qui est généralement le cas), cette décomposition d'une règle souple en deux contraintes peut inhiber la back-propagation du coût vers les variables d'état.

Dans certains cas, il est possible et donc préférable d'endogénéiser le coût directement dans la contrainte `cost-regular`. Par exemple, une variante souple des règles de stretch consiste à relaxer les bornes sur la taille autorisée des stretchs. On définit une borne supérieure « dure » u et une borne supérieure « souple » l sur la longueur d'un stretch d'une activité a : les stretchs de a de longueur supérieure à u sont interdits, et ceux de longueur supérieure ou égale à l sont permis mais occasionnent une pénalité typiquement proportionnelle avec le dépassement de la borne l. On définit, par exemple, une fonction de pénalité f quadratique qui associe à chaque longueur i de stretch comprise entre l et u un coût 2^{i-l}. Cette règle souple, internalisant le calcul de la valeur de pénalité, peut être modélisée par une contrainte `cost-regular` avec l'automate pondéré représenté par la figure 2.12.

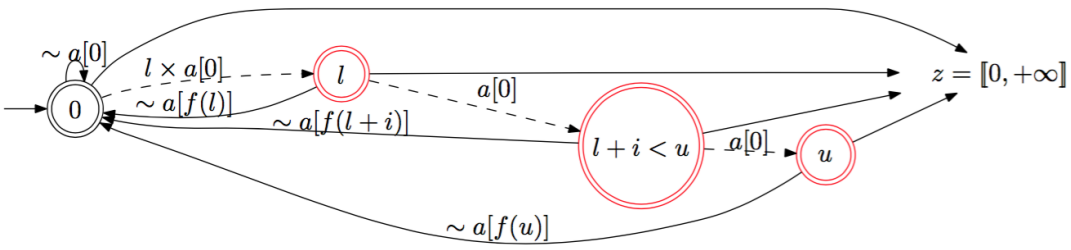


FIGURE 2.12 – Automate autorisant uniquement les stretchs de a de longueur $l + i$ avec i compris entre 0 et $u - l$ et associant une pénalité fonction de cette longueur $f(l + i)$.

De la même façon que pour les règles « dures », nous avons développé des procédures de modélisation automatique adaptées à différents types de règles de séquencement souples et de fonctions de pénalités.

2.4.2.3 Conjonction des règles

L'objectif de modéliser chaque règle individuelle par un automate est de pouvoir les conjuguer par composition en un automate multi-pondéré unique, et donc en une contrainte globale

unique `multicost-regular`.

Intérêt. Le gain espéré en efficacité de résolution est multiple. Premièrement, la composition est opérée au moment de la modélisation, en pré-traitement de la résolution à proprement dite (le backtracking). Pour autant, elle participe activement, en temps négligeable, au processus de résolution comme elle peut déduire de l'information de la conjonction des règles : typiquement, de nouveaux motifs interdits voire des affectations interdites ou obligatoires.

Deuxièmement, elle supprime de la redondance puisque l'automate élémentaire propre à chaque règle modélise l'ensemble des solutions du problème, i.e. qui satisfont toutes les règles. Ces solutions sont représentées de manière unique dans l'automate composé. Cela se traduit en terme de mémoire (et donc d'opérations) : la somme des tailles des automates individuels est bien supérieure à la taille de l'automate composé. À titre d'exemple sur une instance de la littérature du Nurse Scheduling (voir table 2.3), la réduction présente un facteur 10 environ (40402 arcs contre 4768).

Contrat	Analyse	$\sum$AFD	Π	Avant	Π_n
Temps complet	#Nœuds	5782	682	411	230
	# Arcs	40402	4768	1191	400
Temps partiel	#Nœuds	4401	385	791	421
	# Arcs	30729	2689	2280	681

TABLE 2.3 – Illustration sur l'instance GPost de la réduction des graphes (nombre de nœuds et d'arcs) avant résolution : l'ensemble des automates élémentaires avant agrégation (ΣAFD), l'automate agrégé (Π), l'automate déployé dans les phases d'initialisation de `multicost-regular`, avant puis arrière (Π_n).

Troisièmement, même si l'algorithme de `multicost-regular` n'offre qu'un filtrage partiel, on peut espérer un filtrage plus important que sur la conjonction de règles individuelles modélisées chacune par une contrainte `regular`, `cost-regular` ou `global-cardinality`. En effet, la table 2.2 fait état, en moyenne sur les 140 instances *Shoe* résolues par les deux modèles (avec 20 activités au plus), d'un nombre 57 fois moindre de noeuds explorés dans l'arbre de recherche jusqu'à la preuve d'optimalité pour le modèle `multicost-regular` comparé au au modèle partiellement composé d'une `cost-regular` et d'une `global-cardinality`.

Quatrièmement, le temps de propagation d'une `multicost-regular` (hors initialisation) est celui d'une `cost-regular` à un facteur constant près, et il est proportionnel à la taille de l'automate déployé en n couches (n étant la longueur de la séquence de variables). La taille de cet automate Π^n après l'initialisation est généralement bien en-deça de n fois la taille de l'automate Π spécifié. Elle peut même être inférieure à la taille de l'automate spécifié comme dans l'exemple de la table 2.3 où on observe une nouvelle réduction par un facteur 10 environ (400 arcs contre 4768). Cette observation est plus probable encore sur un automate composé de nombreuses règles car celui-ci est, par construction, déjà partiellement déployé. Le nombre d'états est donc similaire après déploiement et le filtrage initial contribue encore à supprimer les chemins de longueur n impossibles (i.e. ne menant pas à un état final). Par conséquent, en plus d'un filtrage supérieur, on peut espérer un temps de propagation d'une `multicost-regular`

équivalent ou moindre à celui de la conjonction des règles individuelles, hormis quand la décomposition inhibe à ce point le filtrage que le point fixe est atteint dès la première itération de la propagation des contraintes individuelles. On observe ainsi dans la table 2.2 un temps de résolution par noeud équivalent pour les deux modèles.

Opérations sur les automates. Contrairement aux langages d'ordre supérieur, en particulier les langages non-contextuels, les langages rationnels sont clos sous les opérations d'intersection, d'union, de concaténation, et de complémentarité. De plus, on dispose d'algorithmes efficaces pour effectuer ces opérations sur les automates (non-pondérés) décrivant les langages.

Ainsi, dans notre procédure d'agrégation, l'ensemble des règles individuelles modélisées en automates sont agrégées l'une après l'autre, par intersection des automates, en un automate unique. On applique alors à l'automate résultant, l'algorithme de minimisation de Hopcroft, implémentable en $O(|Q| \log |Q|)$, qui produit un automate de taille minimale reconnaissant le même langage.

La procédure débute par l'intersection des automates non-pondérés. En pratique, nous employons l'opération d'union sur le complémentaire plutôt que l'opération d'intersection. En effet, l'opération d'intersection réalise le produit des automates $\Pi_1 \times \Pi_2$ où tous les couples d'états (Q_1, Q_2) sont créés. Une transition $((q_1, q_2), \sigma, (r_1, r_2))$ est retenue alors si et seulement si les transitions (q_1, σ, r_1) et (q_2, σ, r_2) existent respectivement dans Π_1 et Π_2. Les opérations de complémentarité et union sont en revanche moins coûteuses : l'opération de complémentarité consiste simplement à inverser états finals et non-finals, tandis que l'opération d'union consiste à ajouter un nouvel état initial et les transitions correspondant aux transitions initiales dans l'un ou l'autre des automates réunis. L'inconvénient est que l'union produit un automate non-déterministe tandis que les opérations de minimisation et de complémentarité ne s'appliquent qu'à des automates déterministes. De plus, l'opération de déterminisation, qui doit être appliquée à l'automate final avant les dernières opérations de complémentarité et de minimisation, produit un automate avec potentiellement un nombre exponentiel d'états. À travers toutes nos expérimentations, nous n'avons cependant pas rencontré un tel cas et le temps de calcul de l'agrégation obtenue par union et complémentarité sont largement inférieurs par rapport à l'intersection directe.

Opérations sur les automates multi-pondérés. Concernant l'intersection des automates pondérés, notre définition diffère de la définition de la littérature. La définition originale d'un automate pondéré est en réalité plus large que celle utilisée ici : il s'agit d'un automate équipé de poids sur les transitions appartenant à un demi-anneau quelconque. Dans notre définition des automates (mono-)pondérés, seul le demi-anneau tropical $(\mathbb{R} \cup \{-\infty, +\infty\}, \min, +)$ est considéré. Dans sa définition originale, l'intersection de deux automates pondérés (Π_1, c_1) et (Π_2, c_2) sur ce demi-anneau résulte en un automate pondéré sur le même demi-anneau reconnaissant l'ensemble des mots reconnus à la fois par Π_1 et Π_2 et leur attribuant un coût égal à la somme de leur coûts initiaux $c_1 + c_2$. Dans notre définition, les pondérations sont indépendantes et doivent être concaténées et non pas additionnées. L'automate résultat de cette intersection est un automate que l'on dit multi-pondéré comme il se place dans un demi-anneau tropical en dimension supérieure (ici $\mathbb{R}^2$) : il reconnaît l'ensemble des mots reconnus à la fois par Π_1 et

Π_2 et leur attribue deux coûts distincts c_1 et c_2. Cette définition se généralise à l'intersection d'automates multi-pondérés, l'automate résultant de l'intersection étant défini sur le demi-anneau tropical de dimension $p_1 + p_2$, la somme des dimensions des demi-anneaux tropicaux sur lesquels sont définis les automates Π_1 et Π_2.

Nous avons ainsi développé un algorithme d'intersection propre à cette définition. L'algorithme est détaillé dans [Men11]. En bref, celui-ci applique l'opération d'intersection classique sur les automates non-pondérés aux automates initiaux modifiés par marquage des transitions. Le marquage consiste à agréger la valeur du symbole et des poids d'une transition en un nouveau symbole. L'opération d'intersection conserve alors les transitions en regardant la valeur des symboles originaux au lieu des symboles modifiés. Après minimisation de l'automate résultant, les symboles originaux des transitions sont restaurés et les pondérations concaténées.

2.4.3 Contributions

Meta-contrainte d'optimisation. La contrainte globale `multicost-regular` présente plusieurs propriétés originales qui lui confèrent un caractère unique : c'est une meta-contrainte dans laquelle on connaît l'expression de plus d'une centaine de contraintes [BCR] et dont on peut dériver facilement certaines variantes, souples ou d'optimisation notamment, en modifiant les automates ou en ajoutant des dimensions de coût ; c'est une contrainte d'optimisation min/max et multi-objectif dans le sens où elle contrôle et exploite à la fois les bornes inférieures et supérieures de plusieurs variables de coût indépendantes ; c'est une contrainte globale qui offre à l'instar de `cost-regular`, en plus du filtrage et du calcul de bornes, un service pour la stratégie de branchement (les supports des plus courts et longs chemins suivant les différentes dimensions de coût) ; c'est une contrainte NP-difficile pour laquelle nous avons proposé un des premiers algorithmes de filtrage basés sur la relaxation lagrangienne, un filtrage incomplet, difficile à caractériser et non-monotone mais efficace en pratique. Ces différentes propriétés sont discutées plus en détail dans le chapitre suivant.

Aide à la modélisation. Nous avons aussi équipé la contrainte globale `multicost-regular` d'un outil d'aide à la modélisation. Comme pour le langage dédié des contraintes utilisateurs de BtrPlace (section 2.1), ces travaux visent à accroître l'ergonomie des contraintes globales et leur utilisation pratique et appliquée. La perspective de la modélisation des contraintes globales est relativement peu considérée dans la littérature ; à noter tout de même les travaux sur l'acquisition de contraintes globales présentés dans le workshop annuel *Constraint Modelling and Reformulation* ou encore les applications directes dans les API des solveurs (pour aider à modéliser les contraintes en extension par exemple). Un service d'aide à la modélisation des contraintes globales n'a de sens en effet que dans le cas de contraintes génériques. Dans le cas de `multicost-regular`, ce service est rendu possible par la représentation sous forme d'automates qui s'accompagne d'une palette d'outils pour leur manipulation. Un tel service serait par exemple plus difficile à automatiser dans le cas des contraintes définies par des grammaires non-contextuelles (qui n'offrent pas cette palette, notamment car l'intersection n'est pas une opération close dans ces langages [HMU01]) mais il est envisageable pour les contraintes de graphes ou de diagrammes de décision [Had+08 ; PR15].

Intégration des fonctions de pénalité. Deux autres contributions relatives à `multicost-regular` sont détaillées dans [Men11] et résumées ci-dessous.

La première concerne la conception d'une variante souple de `multicost-regular`, appelée `soft-multicost-regular`, intégrant les fonctions de pénalité associées aux règles souples. En effet, dans la section 2.4.2, nous montrons que pour certaines règles souples, la fonction f de pénalité et la variable z du coût de violation étaient modélisées en dehors de `multicost-regular` et liées à elle par la variable y de compteur de violations : $z = f(y)$. Dans `soft-multicost-regular`, des fonctions de pénalités, éventuellement non-monotones, peuvent être associées aux différents compteurs et une variable globale réalise la somme totale de coûts. Le filtrage par relaxation lagrangienne s'applique de manière similaire à `multicost-regular` sur le programme linéaire ((2.14)-(2.19)) modifié. En plus de calculer les plus courts/longs chemins, il s'agit dans chaque sous-problème d'obtenir le minimum pour chaque fonction de pénalité (additionnée d'une constante différente à chaque itération). Si la minimisation peut s'exécuter dans tous les cas en représentant la fonction de pénalité en extension, le calcul peut évidemment être optimisé en connaissance de la régularité de la fonction. À noter aussi que l'algorithme de sous-gradient ne converge pas nécessairement en présence d'une fonction non-convexe mais le principe de filtrage reste valable, même si potentiellement moins performant.

Nos premières expérimentations (sur une variante souple des plus grandes instances *Shoe* avec 50 activités de la section 2.3.4) présentent des temps de résolution similaires, légèrement meilleurs ($6,1s$ contre $6.3s$), avec `soft-multicost-regular` par rapport au modèle composé d'une `multicost-regular` et de deux fonctions de pénalité modélisées par des contraintes en extension. En revanche, le nombre de noeuds dans l'arbre de recherche est supérieur (62 contre 50). Cela indique que les sous-problèmes lagrangiens sont de moins bonne qualité dans `soft-multicost-regular` et que le gain de temps est principalement dû à l'absence de mécanisme de communication entre les contraintes. Dans `soft-multicost-regular`, les variables de compteur causent une indirection entre les variables d'états et les variables de coût et les deux relations compteur-états et compteur-coût sont relâchées dans les sous-problèmes lagrangiens ce qui explique probablement la plus faible capacité de filtrage.

Si nous n'avons pas poussé l'étude de `soft-multicost-regular`, faute de temps, nous pensons qu'elle reste une voie à approfondir pour plusieurs raisons : en terme de modélisation d'abord, puisqu'elle permet d'intégrer toute forme de fonctions de pénalité ; en terme d'implémentation, elle démontre la capacité du filtrage par relaxation lagrangienne à s'étendre facilement à la variante souple des contraintes ; en terme d'efficacité car, comme on a vu sur nos premières expérimentations, l'économie du mécanisme de propagation en présence de nombreuses fonctions de pénalité peut parfois compenser un filtrage moins performant ; en terme de potentiel, puisque différentes optimisations sont envisageables, notamment en exploitant – ce qui n'est pas fait actuellement – la régularité des fonctions de pénalités.

Recherche locale et programmation par contraintes pour le Nurse Scheduling. Notre dernière contribution est le développement d'une méthode complète de résolution du problème de Nurse Scheduling basée sur une hybridation de programmation par contraintes et de recherche locale. La double gageure du Nurse Scheduling est que (1) les règles de travail possibles à consi-

dérer sont innombrables et hétérogènes et (2) il peut s'agir de purs problèmes d'optimisation où toutes les règles sont souples.

Le premier aspect peut faire défaut aux meta-heuristiques de par leur manque de déclarativité. L'ajout d'une nouvelle règle ou d'un nouveau type de règles nécessite, au mieux de reparamétrer, au pire d'implémenter (puis configurer, optimiser, tester) de nouveaux modules. Dans notre approche, un modèle de contraintes est employé où l'horaire de chaque employé est régi par une `multicost-regular` et par les contraintes de pénalité associées, et où des contraintes `global-cardinality` assurent de manière transversale la couverture des activités à chaque pas de temps.

Le second aspect implique qu'une affectation quelconque des tâches forme une solution réalisable. S'il est favorable aux meta-heuristiques puisque tout mouvement de recherche locale ou toute opération génétique est alors valide, cet aspect est en revanche désavantageux pour un modèle de contraintes car le filtrage ne repose alors plus que sur la back-propagation des coûts. De ce fait, nous proposons d'activer notre modèle de contraintes où `multicost-regular` assure, du moins en partie, cette back-propagation, au sein d'une recherche locale à voisinage large : une solution est améliorée progressivement de proche en proche dans son voisinage, défini en fixant une partie des variables à leurs valeurs dans la solution. À chaque itération, un voisinage est exploré de manière complète par backtracking sur le sous-ensemble des variables restantes.

Cette forme d'hybridation a initialement été proposée par Shaw [Sha98] puis largement reprise dans la communauté recherche opérationnelle où les voisinages sont explorés par différentes méthodes. Elle est facile à implémenter mais nécessite de définir avec soin la forme et la taille des voisinages de manière à ce qu'ils soient compatibles avec le modèle de contraintes. Nous l'avons adaptée ici en privilégiant l'incrémentalité du filtrage de `multicost-regular` afin de neutraliser sa complexité temporelle à la fois dans la définition des voisinages et dans leur parcours. Ainsi, le voisinage d'une solution est défini en sélectionnant un ensemble de périodes et en défixant les variables d'affectation de tous les employés à ces périodes. Au début de chaque itération, toutes les `multicost-regular`, associées chacune à un employé, sont propagées, mais de manière limitée car la pré-affectation d'une partie de la séquence réduit grandement la taille du graphe déployé sous-jacent à chaque contrainte. On choisit également de ne relâcher que les activités sur des périodes contigues car les interactions y sont plus fortes et qu'on limite ainsi le parcours des graphes.

Une fois le voisinage défini, il est parcouru par backtracking suivant la stratégie de branchement. Comme avec `cost-regular` (voir section 2.3.5), cette stratégie exploite des informations internes à `multicost-regular`. Nous les exploitons non seulement dans l'heuristique de choix de valeur, mais également dans l'heuristique de choix de variable, comme proposé par [FLM99], pour sélectionner, à chaque instant t (les variables d'affectation étant fixées période après période), le prochain employé à affecter suivant une notion de *regret* : la différence entre le coût de la meilleure affectation et le coût de la seconde meilleure. En effet, notre implémentation de `multicost-regular` offre accès en lecture à tous les plus courts/longs chemins dans son graphe interne passant par n'importe quel arc, ce dans les différentes dimensions du coût. Nous avons, dans nos heuristiques, expérimenté différentes combinaisons de ces informations. La plus performante, dans ce contexte, consistait à sélectionner l'employé de regret

maximal, le regret étant approché par $\min_{a \neq a^*}(z^0_{(t,a)} - z^0)$: la différence entre la valeur du plus court chemin (dans la première dimension) passant par un arc étiqueté a dans la couche t et la valeur du plus court chemin (qui passe par un arc étiqueté a^*).

choix de :	e		$e, a \in \mathcal{A}$		$e, a \in \mathcal{A} \setminus \{R\}$	
	first Z	best Z	first Z	best Z	first Z	best Z
$Z_e(\lambda)$	112	86	141	92	104	83
$Z^C_{e,\lambda}$	111	86	141	92	104	83
$Z_{e,\lambda}$	186	158	168	97	102	84
$Z_{e,\lambda} - Z_e$			155	99		
$regret^C_{e,(t,a)}$	76	70			76	69

FIGURE 2.13 – Exemples de stratégies de branchement sur les variables d'affectation et et leur impact sur la valeur de la première (*first Z*) et de la meilleure (*best Z*) solution obtenue en 2 minutes sur l'instance PATAT/sprinto2 de nurse scheduling. Les heuristiques sont appliquées au le choix de l'employé e (colonne 1), ou au le choix de l'employé et de l'activité (colonne 2), éventuellement hors activité *repos* (colonne 3). Elles reposent, de haut en bas, sur : la valeur de la borne lagrangienne minimale, le coût minimal (dans la première dimension) du plus court chemin (dans la première dimension), la somme minimale des coûts (sur toutes les dimensions) du plus court chemin (dans la première dimension), une approximation du regret basé sur les plus courts chemins (dans la première dimension).

Chapitre 3

Réflexions

Dans ce chapitre, nous discutons des questions et propositions centrales de ma recherche : la décomposition et l'hybridation pour l'efficacité et la flexibilité en optimisation combinatoire. Elles sont abordées à travers différents aspects que j'ai traités dans mes travaux, parmi ceux résumés dans le chapitre précédent et d'autres. La première partie 3.1 est consacrée à la programmation par contraintes globales, vue comme un outil pratique d'implémentation de la décomposition et de l'hybridation. La seconde partie 3.2 porte sur le traitement du critère d'optimalité.

3.1 Les contraintes globales : essence de la décomposition et de l'hybridation

Régin [Rég11] définit une *contrainte globale* comme une conjonction de contraintes. En pratique, une contrainte globale est une conjonction particulière présentant une structure de laquelle on peut dériver un propagateur dédié *pertinent* (qui filtre davantage que la consistance d'arc sur la conjonction) et efficace (qui exploite la connaissance de la structure pour s'affranchir de l'évaluation explicite de toutes les valeurs et pour accélérer la recherche de support).

Modéliser un problème dans le cadre de la programmation par contraintes (globales), c'est donc (1) trouver le degré de décomposition juste, dans le but d'accélérer le mécanisme de propagation à l'intérieur de chaque composant, et (2) modéliser le composant par une contrainte globale si elle existe, concevoir et implémenter un propagateur sinon. Le choix de la décomposition repose sur différents facteurs : la difficulté et la prépondérance d'un composant du problème, la disponibilité ou non d'une contrainte globale dans le solveur choisi pour modéliser ce composant, l'opportunité de développer une nouvelle contrainte dans ce but, l'extensibilité du modèle quand la flexibilité est prise en compte au même titre que la performance. On distingue grossièrement – la frontière est floue – deux types de contraintes globales :

Les contraintes spécifiques sont appliquées à une structure réduite précise, à laquelle on peut associer un propagateur efficace et pertinent dédié à cette structure. Elles sont d'autant plus performantes dans la résolution, que l'algorithme est appliqué à une partie réduite du problème. Elles posent question quant à leur usage : doit-on/peut-on/comment classifier toutes les contraintes existantes ? implémenter efficacement toutes les variantes possibles ? hybrider en invoquant des techniques propres à d'autres paradigmes de résolution pour la conception de propagateurs performants ?

Les contraintes universelles agrègent des contraintes hétérogènes ou résultent de stratégies génériques de filtrage. L'algorithme associé est plus faiblement efficace et/ou pertinent mais, par son raisonnement global ou par le simple fait d'agrégation, il supplante en performance le mécanisme de propagation appliqué à la conjonction de contraintes correspondante. Elles gagnent

en flexibilité comme elles ne nécessitent pas de développements supplémentaires quand elles sont étendues aux variantes d'un problème. En retour, de par leur généricité, elles rendent plus complexe la tâche de spécification d'un problème dans leur formalisme propre.

3.1.1 Contraintes spécifiques, classification et hybridation

3.1.1.1 Classification des contraintes

Comment se retrouver dans le bestiaire des contraintes existantes et de toutes les variantes possibles ? À chaque problème et chaque sous-problème correspond une contrainte globale. Potentiellement, celle-ci a déjà été étudiée et un ou plusieurs propagateurs ont été proposés. Certaines structures communes et facilement reconnaissables ont donné lieu à des contraintes centrales connues, notamment `alldifferent` et `global-cardinality`, dont on retrouve une implémentation dans tous solveurs de contraintes. D'autres contraintes ont été étudiées dans un contexte applicatif précis et sont implémentées dans un certain solveur. Pour autant, elles sont réutilisables dans d'autres contextes mais il n'est pas facile alors de les identifier, encore moins s'il s'agit de variantes auxquelles un algorithme donné peut être adapté ou non.

Contrairement à d'autres paradigmes déclaratifs (programmation mathématique, SAT), la programmation par contraintes offre un langage de spécification riche, avec pour objectif de réduire l'effort de modélisation. En l'état, elle requiert de l'utilisateur d'appréhender ce langage extensible à l'infini et perd de ce fait en ergonomie en transférant une partie de la complexité dans l'acte de modélisation. Ce foisonnement ne doit pas devenir un défaut et peut ne pas le devenir si, d'une part, le langage des contraintes globales s'accompagne d'un dictionnaire à l'usage de l'humain qui cherche à identifier et assembler les composants pour définir son problème. D'autre part, l'apprentissage automatique de contraintes est un axe de recherche à développer, dont les techniques doivent également s'appuyer sur des dictionnaires et classifications des contraintes globales.

Dès 2000, Beldiceanu a initié la tâche de répertorier les contraintes globales parues dans la littérature, ou inférées de celles-ci, au sein d'un *catalogue de contraintes globales* [BCR]. Il définit chaque contrainte en langages naturel et mathématique, l'instancie sur un exemple et référence les algorithmes associés. Il explicite des relations entre les contraintes et les classifie par mots-clés selon l'usage, le type d'algorithme et d'autres caractéristiques encore. Ces informations sont également exprimées en version électronique, en prolog, dans un format interprétable par machine. Trois types de reformulation sont enfin proposées (en propriétés de graphes, automates multi-pondérés, et formules logiques) dans le but d'offrir une définition concise et des propagateurs systématiques.

Un tel outil est, je pense, primordial pour la diffusion de la programmation par contraintes, pour accompagner la recherche sur l'apprentissage de contraintes, ou encore pour fédérer une communauté large autour de l'algorithmique relative à la programmation par contraintes. J'y contribue dans la forme depuis 2006 en créant et maintenant le site web traduit automatiquement depuis les documents source (latex, prolog). Le but initial était de mettre à disposition la version électronique tout en offrant une lecture plus aisée du catalogue papier. Le catalogue répertoriait 270 contraintes en 2006 et il en contient 443 aujourd'hui. Pour faciliter la recherche, j'ai enrichi la version web d'outils et de lexiques (sur les noms, les mots-clés, les références et

sur divers caractéristiques) et d'une traduction entre les prédicats du catalogue et les prédicats utilisés dans différents solveurs (Choco, Gecode, Jacop, Sicstus). S'il rencontre l'intérêt des utilisateurs (le site a en moyenne 50 pages vues par jour), le catalogue reçoit peu de contributions. Il me semble que la question de la classification des contraintes globales mériterait un traitement plus collectif de la part de la communauté des chercheurs, des développeurs et des utilisateurs. Une encyclopédie au format participatif « à la Wikipedia » aurait aussi une grande valeur.

3.1.1.2 Contexte d'hybridation

Avec le principe d'isolation des propagateurs dans les contraintes globales et leur communication à travers les domaines des variables par le mécanisme de propagation, la programmation par contraintes offre un cadre général pratique pour hybrider des techniques issus de tout paradigme, en associant à chaque sous-problème la technique la plus adaptée.

Ainsi, de nombreux propagateurs proposés dans la littérature s'appuient sur des concepts et techniques de la recherche opérationnelle et de la théorie des graphes, ou encore de la géométrie algorithmique, par exemple : les règles d'ordonnancement (edge-finding [Pin88 ; CP90] par exemple) pour les contraintes `cumulative` et `disjunctive`, le couplage dans un graphe biparti pour les contraintes `alldifferent` [Rég94] et `global-cardinality` [Rég96], l'algorithme *sweep* pour des contraintes de placement telles `diffn` [BC01]. Avec `interval-amongs`, nous avons procédé à différentes reformulations, en systèmes linéaires, en un graphe de flot et en un graphe de distance, pour établir la preuve de complexité et des algorithmes de filtrage. Les propagateurs de `cost-regular` et `multicost-regular` reposent, respectivement, sur la programmation dynamique et la relaxation lagrangienne.

Une tâche ardue. Les contraintes globales facilitent la tâche d'hybridation à l'usage, mais pas à la conception.

La première difficulté à la création d'une contrainte globale est d'identifier les techniques disponibles dans tout paradigme algorithmique et sur lesquelles baser le filtrage. Elle est plus aisée si la structure considérée correspond à un problème connu, comme dans l'exemple de `multicost-regular` dont le problème associé de plus court chemin avec ressources, est couramment résolu par relaxation lagrangienne. Dans tous les cas, la tâche d'identification nécessite un travail de recherche et de compréhension dans une bibliographie plus ou moins distante.

La deuxième difficulté est de concevoir un propagateur à partir de l'algorithme identifié. Il peut s'agir d'un algorithme qui détermine une solution, qui vérifie une propriété, ou qui calcule une quantité, et qui est généralement appelé de manière isolée sur des données fixes. Au contraire, un propagateur consiste à détecter des infaisabilités et à supprimer les valeurs n'appartenant à aucune solution ; il est appelé de manière itérée au sein du mécanisme de propagation, lui-même appelé de manière répété dans l'arbre de recherche, sur des attributs variables.

La troisième difficulté est de rendre le propagateur rapide, dans son application isolée et répétée. Une approche consiste à exploiter le principe d'incrémentalité, c'est à dire en permettant de propager rapidement un évènement simple, de retrait d'une valeur d'un domaine

par exemple. Une autre approche consiste à changer de structure de données pendant la recherche quand cela permet une économie de calcul au moment où la taille des domaines des variables ou que le fractionnement des domaines énumérés passent un certain seuil. De fait, la connaissance d'un algorithme de résolution ne donne pas les clefs pour la conception d'un propagateur efficace. C'est ainsi, par exemple, que nous avons écarté la programmation dynamique, l'approche la plus commune de résolution du problème de plus court chemin sous contraintes de ressources, pour le filtrage de `multicost-regular` car son coût en mémoire nous est apparu incompatible avec le besoin de sauvegarder l'état de la contrainte d'un noeud de l'arbre de recherche à un autre.

Quand il est décidé d'une approche incrémentale, la dernière difficulté réside dans son implémentation, en maintenant correctement la mémoire interne de la contrainte par exemple, et dans son évaluation de manière exhaustive, alors que les multiples cas pathologiques peuvent ne se manifester que dans de rares configurations conjuguant variables, contraintes et ordre de propagation. Si les dernières difficultés de conception et de mise en œuvre ne sont pas propres aux propagateurs hybrides, elles sont probablement plus fortes dans ces cas où les propagateurs sont adaptés de techniques pointues.

Il existe des solutions informelles face aux difficultés ci-dessus, comme établir des passerelles entre les communautés pour faciliter l'identification de techniques applicables (le catalogue de contraintes peut jouer un rôle dans ce sens) ou prévoir des interfaces ergonomiques d'implémentation de contraintes globales dans les solveurs. Cependant, réduire la complexité théorique et pratique des propagateurs restera un enjeu difficile, au cas par cas, tant que ne sera inventé, si jamais il existe, l'ultime propagateur universel auto-adaptable.

Une communication limitée. Dans la forme d'hybridation proposée par la programmation par contraintes, la communication entre les composants se fait à travers les domaines des variables. Le produit cartésien des domaines forme la relaxation du problème que les propagateurs des contraintes œuvrent à resserrer. C'est une forme générique d'hybridation qui, en contrepartie, semble limitée.

Différentes voies pour intensifier la communication entre les composants sont envisageables. Une première voie consiste naturellement à automatiser l'agrégation des contraintes. La composition d'automates pour `multicost-regular` (section 2.4) en est une illustration. Des techniques plus avancées d'apprentissage de contraintes permettraient de reconnaître automatiquement des combinaisons de contraintes et la manière de les agréger comme dans [BCP07] par exemple. Une autre voie passe par le partage, et donc, l'accès à la mémoire interne des contraintes. Ainsi, les heuristiques de recherche basées sur des éléments internes aux contraintes, telles que celle de `cost-regular` (section 2.3), peuvent être combinées entre elles de différentes manières ad-hoc ou génériques. Une nouvelle voie attractive a émergé plus récemment avec les différents travaux sur la relaxation de modèles par des diagrammes de décision [And+07 ; HVH10 ; BHH11 ; BC16 ; PR15] ou par relaxation et décomposition lagrangienne [F+14 ; HQR15 ; BCH15a ; CGS16] visant à renforcer la relaxation formée par le produit cartésien des domaines. Enfin, une recherche active sur des cadres d'hybridation qui généralisent celui de la programmation par contraintes, ou bien qui lui emprunte ses contraintes globales en les combinant dans un autre cadre que la propagation, par reformulation automatique

dans la programmation logique, la programmation linéaire, la recherche locale, a donné lieu à différentes implémentations dont SIMPL [AHY04], G12/Cadmium [DDS08], Essence [Fri+08], SCIP [Ach09], Comet CML [FM12].

3.1.2 Filtrage générique et flexibilité

Concevoir des algorithmes ad-hoc pour des contraintes spécifiques est une tâche laborieuse sans fin. C'est un reproche fait depuis l'origine des contraintes globales, notamment par la communauté Intelligence Artificielle, de devoir redévelopper entièrement la moindre varainte [Bel03]. Dans [Rég11], Régin présente deux schémas génériques de filtrage pour toute conjonction de contraintes : la première consiste à identifier et à ajouter des contraintes induites par la conjonction et la seconde consiste à appliquer un algorithme de consistance d'arc généralisé (*shaving* ou la version améliorée GAC-Schema [BR97]) à la conjonction. Par ailleurs, les meta-contraintes fournissent des algorithmes génériques applicables, après reformulation, à des familles entières de contraintes ou à certaines conjonction de contraintes hétérogènes.

Ces approches sont plus directement implémentables, à défaut d'offrir un filtrage à la fois complet et rapide. Sur un problème donné, elles peuvent être notamment employées pour identifier des conjonctions de contraintes formant un sous-problème dur central : si améliorer la propagation dans cette conjonction aboutit à réduire le nombre de backtracks, alors il peut être rentable d'élaborer un algorithme dédié. Surtout, elles apportent une flexibilité et une extensibilité pratiques, qualités attendues de la programmation par contraintes.

3.1.2.1 Recomposition de contraintes

Avec `interval-amongs`, nous avons employé une variante au premier schéma proposé par [Rég11], consistant à recomposer une conjonction de contraintes en une autre qui lui est substituée. Nous avons ainsi proposé deux décompositions alternatives (cardinalité et somme-cardinalité) à la décomposition naturelle (among) et montré qu'elles étaient incomparables en terme de filtrage. Ce principe s'applique bien aux contraintes de cardinalité dont les modèles duaux, basés sur les variables de cardinalité, offrent des recomposition tout indiquées. Nous avons également prouvé que la contrainte `interval-amongs` en dimensions supérieures devient $\mathcal{NP}$-difficile et que la décomposition naturelle par dimension inhibe fortement la propagation. La recomposition serait une alternative valable, cependant s'abstraire des dimensions est une tâche ardue.

Le principe de recomposition n'est pas nouveau. Il est par exemple employé pour la contrainte `automaton` [Bel+05], équivalente à `regular`. Dans `automaton`, des variables additionnelles $Q_1, Q_2, \ldots, Q_{n+1}$ sont associées aux états intermédiaires de la séquence de variables $X_1, X_2, \ldots, X_n$. La contrainte automate se décompose alors en n contraintes ternaires de la forme $(Q_i, X_i, Q_{i+1}) \in \Delta$ ou Δ désigne l'ensemble des transitions de l'automate spécifié. La conjonction étant Berge-acyclique, la propagation de cette recomposition assure la consistance d'arc. Dans sa version pondérée, la recomposition doit être augmentée d'une (ou plusieurs) variable de coûts associée à chaque état et liées par une contrainte de `knapsack`. Les contraintes partageant alors plus d'une variable, la recomposition n'est plus complète et son niveau de consistance est incomparable à celui de `cost-regular` ou `multicost-regular` (voir l'exemple

de la figure 2.7).

Ainsi, la principale difficulté de ce schéma générique de filtrage par recomposition est d'ordre théorique : elle réside dans l'identification de la décomposition alternative. Lorsqu'il s'agit de contraintes référencées, le catalogue de contraintes[BCR] présente souvent des reformulations comme des conjonctions de contraintes, elles-même référencées, toutes identiques (au mot-clé *systems of constraints*) ou non (voir la section *reformulation*).

Le principe de recomposition nécessite ensuite une comparaison formelle et empirique de la capacité de filtrage pour savoir si la recomposition peut préférablement se substituer à la conjonction initiale. Certaines recompositions impliquent par exemple un nombre quadratique de variables ou de contraintes qui n'est pas compatible avec des instances de grande taille, mais qui peut être avantageux dans d'autres cas.

En contrepartie, l'approche par recomposition est accessible et portable car elle réduit drastiquement le travail de mise en œuvre, sauf à redévelopper les contraintes apparaissant dans la nouvelle conjonction si elles n'existent pas déjà.

Enfin, cette approche introduit dans certains cas de nouvelles variables qui sont uniquement contraintes dans la nouvelle conjonction. Ces variables additionnelles ont l'inconvénient d'augmenter la taille du modèle. Toutefois, elles exposent une information interne à la contrainte globale (les cardinalités dans `interval-amongs` ou les états intermédiaires dans `automaton`), qui peut être exploitée, par exemple, dans la stratégie de recherche pour guider et accélérer la résolution. Ces variables peuvent également faciliter la définition de nouvelles contraintes induites par la conjonction de la contrainte globale considérée et des autres contraintes du problème.

3.1.2.2 Meta-contraintes

Une meta-contrainte offre un langage déclaratif et un formalisme particulier pour spécifier en les généralisant des familles hétérogènes de contraintes et un propagateur qui exploite cette structure.

Les contraintes en extension généralisent toute contrainte dont l'ensemble des supports peut être explicité sous forme de table. Si elles sont en théorie les plus génériques, leur usage est limité par la taille de l'ensemble des supports. Des études ont été menées sur différentes formes compactes, automates, diagrammes de décision, systèmes linéaires ou formules logiques, constructibles sans expliciter l'ensemble des supports et de la résolution desquels on dérive du filtrage.

Avec le catalogue de contraintes [BCR], Beldiceanu a produit un dictionnaire de reformulations pour une majorité de contraintes sous formes d'automates et de propriétés de graphes. Dans le premier cas, les propagateurs de `automaton`, `regular`, `cost-regular` et `multicost-regular` s'appliquent en fonction de la présence ou non de compteurs. La seconde reformulation consiste à identifier une contrainte à un réseau dynamique de contraintes binaires (les arcs du graphe) sur le même ensemble de variables (les sommets), associé à un ensemble de propriétés satisfaites par le graphe dans son état final (quand toutes les variables sont instanciées) et portant sur l'occurrence de certains paramètres comme le nombre de composantes connexes par exemple. Filtrer se traduit ici par statuer sur l'appartenance ou non d'un arc au graphe final. Avec Nicolas Beldiceanu, Thierry Petit et Mats Carlsson nous avons exhibé

dans [Bel+06] des règles de filtrage pouvant se déclencher lorsque les propriétés sont satisfaites mais les arcs pas tous encore fixés.

Naturellement, les algorithmes pour ces contraintes hautement génériques sont incomplets et/ou coûteux dans leurs applications les plus lourdes et ils ne sont pas tous triviaux à implémenter. Cependant, une fois disponibles dans un solveur, ils peuvent être employés en place de chaque contrainte pour laquelle une reformulation est possible. De plus, si dériver un algorithme dédié à une contrainte pour une de ses variantes peut être complexe, ne serait-ce que dans la mise en œuvre de l'algorithme dérivé, il est souvent plus évident d'obtenir la reformulation de la variante à partir de celle de la contrainte originale.

Traduire une contrainte ou un ensemble de contraintes dans un format générique donné nécessite cependant une certaine expertise de la part de l'utilisateur. Comme précédemment, les meta-contraintes déportent la difficulté de mise en œuvre d'un algorithme vers la spécification d'un modèle. Des outils d'aide à la modélisation sont alors essentiels : des dictionnaires de reformulation comme dans le catalogue de contraintes, qui nécessitent de l'utilisateur d'avoir au préalable identifié les contraintes globales attachées à son problème ; ou des outils tels des langages de spécification, comme l'outil d'aide à la modélisation de `multicost-regular`, dédiés ou non à un domaine d'application.

3.1.2.3 Contraintes de langage

Les contraintes automates sont des meta-contraintes particulièrement expressives car elles reposent sur un format, les automates finis, créé pour et dédié à la modélisation compacte (des langages formels) : nous avons montré comment elles généralisent diverses contraintes de séquencement et de cardinalité ; par ailleurs, le catalogue de contraintes référence [1] 59 reformulations de contraintes en automates non-pondérés, 55 en automates pondérés, 24 en automates multi-pondérés.

En plus d'être expressives, les contraintes automates ont la propriété d'être extensibles. on peut parfois facilement modifier l'automate associé à une contrainte pour obtenir une variante de celle-ci. De plus, les pondérations permettent de dériver des variantes souples ou valuées, d'optimisation, d'évaluation, d'énumération ou de comptage ; où la fonction de pénalité/coût est totalement ou partiellement intégrée à la contrainte (voir section 2.4.2.2).

La difficulté pour l'utilisateur est de caractériser une expression régulière ou un automate fini pour spécifier le langage voulu, sachant qu'il n'y a pas unicité. Cette difficulté est aussi accrue dans le cadre de `multicost-regular`, où certaines restrictions peuvent s'exprimer soit par un automate, soit par un coût. La complexité de l'algorithme, proportionnelle à la taille de l'automate déployé, doit être prise en compte dans le choix de la représentation : dans l'idéal, il s'agit d'identifier l'automate de taille minimale reconnaissant uniquement les mots du langage de taille n. Cette représentation n'est cependant pas fonctionnelle comme l'automate est généralement ainsi moins lisible, a fortiori, plus difficilement modifiable par l'utilisateur. Là encore, l'avantage du formalisme des automates finis est qu'il est composable en machine, comme il s'accompagne d'une vaste palette d'outils algorithmiques que nous avons étendue aux automates multi-pondérés. Nous avons proposé de mettre au service de l'utilisateur ces outils pour l'aider à composer son modèle en un automate de taille minimale (voir section 2.4.2).

1. dans la version 2014 avec 423 contraintes

La composabilité est un atout en terme de modélisation mais aussi en terme de performance de la résolution. Ainsi, une contrainte automate unique peut avantageusement modéliser de multiples restrictions sur une même séquence de variables au lieu de multiplier les contraintes. Le processus de résolution s'en trouve accéléré comme les interactions entre ces règles sont gérées, non plus dans la propagation des contraintes simples, mais dans le filtrage de la contrainte composée ; voire même en amont, au moment de l'intersection des langages.

Pour autant, les propagateurs de `cost-regular` et `multicost-regular` sont, de par leur généricité, lourds et incomplets. Une piste future serait d'identifier des cas particuliers où ces performances pourraient être améliorées. Il s'agirait alors de spécialiser la contrainte et de la recombiner (en modifiant l'ordre des variables par exemple) en amont, voire en cours de résolution.

Les contraintes `grammar` [QW06 ; Selo6] ont été proposées afin d'étendre la représentation des langages rationnels avec `regular` aux langages non-contextuels. Les premiers sont en effet strictement inclus dans les seconds : voir l'exemple connu d'un langage non-contextuel et non-régulier constitué de tous les mots formés par la concaténation d'une séquence d'un symbole a et d'une séquence d'un symbole b de même longueur. En réalité les deux contraintes ont une expressivité semblable car elles tronquent le langage spécifié à l'ensemble des mots d'une longueur donnée (égale à l'arité de la contrainte). Tout langage non-contextuel tronqué ainsi est rationnel. Autrement dit, il n'est théoriquement pas de contrainte implémentable dans `grammar` qui ne le soit dans `regular`. Pour autant la taille de l'automate modélisant par exemple les expressions parenthésées correctement balancées et d'une longueur fixe explose rapidement avec cette longueur. A contrario, l'ajout des pondérations étend strictement l'expressivité, ceci en conservant, voire en augmentant, la compacité.

L'apport de `grammar` en terme de modélisation réside ainsi plutôt dans son langage de spécification sous forme, non plus d'automates finis, mais de règles de production. De ces règles, on dérive un graphe de taille cubique en le nombre de variables et sur lequel deux types de propagateurs s'appliquent, par programmation dynamique [Selo6] ou par une représentation par un graphe AND/OR [QW07].

Une étude comparative entre les contraintes de grammaire et les contraintes automates a été proposée dans [QR10] sur un exemple particulier : une simplification des instances de planification de personnel *shoe* (voir section 2.3.4) résolue par recherche locale à voisinage large au-dessus d'un modèle basé sur l'une ou l'autre contrainte. La comparaison expérimentale montre que, quand le nombre d'activités augmente ($|A|$ variant de 1 à 10), la taille du graphe sous-jacent à `grammar` devient rapidement (dès $|A| = 2$) inférieure à celle du graphe de `regular` et jusqu'à près de trois fois plus petite (pour $|A| = 10$). Les nœuds du graphe de `grammar` sont en effet liés aux sous-séquences de mots possibles alors que les nœuds du graphe déployé de `regular` portent des informations accumulés depuis le début jusqu'à la fin de la séquence. Les temps de résolution moyens pour un voisinage deviennent également à l'avantage de `grammar` mais dans une proportion moins significative : de $9\mu s$ pour `grammar` contre $5\mu s$ pour `regular` pour $n = 1$ jusqu'à $234\mu s$ pour `grammar` contre $378\mu s$ pour `regular` pour $n = 10$. En effet, les deux algorithmes procèdent en temps proportionnel à la taille du graphe, l'algorithme de `grammar` calculant un arbre là où `regular` calcule un chemin.

Concernant l'expressivité, les auteurs avancent qu'elle est à l'avantage de `grammar` arguant qu'il suffit de 14 règles de production pour capturer l'ensemble des règles de séquencement de

ce problème alors que l'automate équivalent, quand le nombre d'activités augmente, nécessite des « milliers d'états » [2]. Si la facilité pour un utilisateur de déclarer des règles de production ou un automate est une question plutôt subjective, en revanche, les possibilités d'un traitement machine des automates finis sont plus étendues que pour les automates à pile (qui reconnaissent les langages non-contextuels), car les opérations disponibles, développées dans le cadre des grammaires formelles, sont plus performantes. Par exemple, comme l'ensemble des langages non-contextuels n'est pas clos pour l'intersection [HMU01], il n'existe pas d'algorithmes aussi efficaces et directement exploitables pour automatiser la conjonction de contraintes `grammar`. Il semble ainsi qu'il soit plus ardue dans le contexte des contraintes `grammar` d'aboutir à une procédure complète d'aide à la modélisation comme celle que nous avons développé au-dessus du formalisme des automates pondérées pour `multicost-regular`.

Les travaux de Côté et al. [CGR11] sur la linéarisation des contraintes de langage `regular` et `grammar` (avec notamment la preuve d'intégralité pour l'une [Pes+09]) pour la planification de personnel ont un intérêt au-delà de ce contexte. Il en découle en effet un outil de reformulation automatique en un système d'équations linéaires sur des variables fractionnaires, qui s'applique directement à l'ensemble des contraintes globales dont on dispose d'une reformulation sous `regular` ou `grammar`. Il serait intéressant d'étudier l'intégration des pondérations dans ces linéarisations, ainsi que les applications, hors planification de personnel, d'un tel outil qui fait le pont d'un paradigme (la programmation par contraintes) à un autre (la programmation linéaire).

3.1.2.4 Flexibilité/efficacité

Comme dit précédemment, les approches génériques de filtrage sont utiles, par exemple, dans un travail préliminaire pour identifier les sous-problèmes difficiles dont il faut soigner le traitement pour accélérer la résolution du problème global. Autrement dit, elles aident à déterminer le bon grain de décomposition d'un problème.

Elles sont également importantes en « optimisation flexible » où la facilité d'implémentation devient un critère prépondérant. On a montré que la généricité (ajout/recomposition de contraintes, meta-contraintes) déporte la complexité de la mise en œuvre algorithmique vers la modélisation. C'est un avantage dans une solution qui doit pouvoir être augmentée par des utilisateurs non-experts en algorithmique, comme dans BtrPlace par exemple. Il leur est plus facile de spécifier un sous-problème (une contrainte utilisateur dans le cadre de BtrPlace) comme une conjonction de contraintes que de concevoir et d'implémenter un algorithme dédié.

Mais, apporter de la généricité se fait au détriment de la force ou de la rapidité de filtrage. Par exemple, dans la recomposition de contraintes – autrement dit, quand le moteur de propagation est employé en place d'un propagateur dédié, le filtrage n'est pas complet et le temps de résolution dépend des mécanismes de propagation qui peuvent être coûteux quand la communication entre les contraintes de la conjonction (via les variables communes) est intense. Le filtrage peut cependant être accéléré avec un surcoût de mise en œuvre modéré, lorsque, notamment, il s'agit d'une conjonction de contraintes toutes identiques. En effet, dans ce cas, les contraintes partagent vraisemblablement une mémoire commune et il est parfois possible

2. en réalité, l'automate *non-déployé* que l'utilisateur doit spécifier est celui représenté dans la figure 2.8 et contient $4 * n + 7$ états, soit 47 états tout au plus dans les instances considérées.

d'identifier l'impact précis qu'aura un évènement de filtrage d'une contrainte sur les autres. Alors, il peut être avantageux d'itérer sur le filtrage des contraintes de la conjonction et de l'intégrer au sein d'une contrainte globale, à la manière dont nous avons procédé avec la contrainte `vector-packing` de BtrPlace. Celle-ci ajoute une boucle externe, sur les différentes dimension de ressources, au filtrage incrémental de `bin-packing` tout en mettant en commun la mémoire partagée, comme les listes des items et des conteneurs par exemple. La mise en œuvre de l'algorithme est simple dans le sens où il reprend principalement l'algorithme de la contrainte locale. Cette approche, si elle n'offre pas de filtrage supplémentaire, économise à la fois le temps de calcul lié au mécanisme de propagation et l'espace de la mémoire partagée. Par ailleurs, cette première implémentation peut ensuite être facilement améliorée : accélérée, dans `vector-packing` par exemple, où les boucles sur les ressources puis sur les conteneurs sont inversées dans certaines opérations pour bénéficier de conditions d'arrêt précoces ; ou augmentée avec des conditions particulières de filtrage dues à la conjonction des contraintes qui seraient potentiellement identifiées.

Concernant les performances des meta-contraintes, on a vu que pour gagner en expressivité et généricité, on aboutit rapidement à des contraintes $\mathcal{NP}$-difficile. Pour améliorer ces performances, là encore, l'apprentissage serait une piste pour aboutir à des contraintes auto-adaptatives. Il s'agirait de détecter de manière automatique, avant ou au cours de la résolution, quand un modèle générique répond à un cas particulier pour lequel on a identifié une amélioration de la performance de l'algorithme : en changeant un paramètre, une structure de donnée, voir l'algorithme complet. En attendant, quand différentes implémentations d'une contrainte sont possibles, avec leurs avantages et défauts respectifs, il est important d'offrir à l'utilisateur de pouvoir choisir facilement son implémentation par simple paramétrage, à l'image de la contrainte `vector-packing`, de manière à ce qu'il puisse adapter facilement le modèle de contraintes à la classe d'instances particulières qui l'occupe.

3.2 Optimisation par contraintes

La programmation par contraintes est réputée pour être inefficace en présence d'un critère de coût à optimiser, quand on la compare principalement à la programmation mathématique. Cette affirmation est fausse en partie : de nombreux problèmes d'ordonnancement en sont un contre-exemple (bien qu'une raison à cela est que la programmation mathématique n'excelle simplement pas sur ces problèmes du fait de la difficulté de linéariser les modèles ou de mauvaises relaxation continues). Mais effectivement, le schéma de résolution classique d'un problème d'optimisation par un solveur de contraintes est une variante de l'algorithme de backtracking pour les problèmes de satisfaction de contraintes qui prend faiblement en compte le critère d'optimisation.

Dans cette dernière partie, je reviens sur l'emploi de différentes formes de décomposition et d'hybridation pour renforcer le traitement d'un objectif à optimiser en parallèle d'un modèle flexible fait de contraintes globales. Une voie consiste à intégrer ce traitement en interne dans le processus de résolution de la programmation par contraintes par le biais des contraintes globales dédiées à l'optimisation et des stratégies de recherche. Une autre voie adopte un traitement externe en intégrant le modèle de contraintes dans un processus de résolution dédié

à l'optimisation tel que les méthodes de décomposition de la programmation mathématique ou la recherche locale et les metaheuristiques.

3.2.1 Contraintes globales d'optimisation

La fonction objectif d'un problème d'optimisation associe à certaines variables de décision, un coût dépendant de la valeur prise par la variable et, typiquement, en effectue la somme. Classiquement, en programmation par contraintes, le coût associé à une variable X_i est modélisé par une variable Z_i. Une contrainte `element`, qui associe à chaque valeur possible j du domaine de X_i la valeur du coût a_{ij} correspondant, modélise alors la relation $X_i = j \iff Z_i = a_{ij}$. La fonction objectif est elle modélisée par une contrainte somme sur les variables de coûts $\texttt{sum}(Z)$. L'algorithme de backtracking est modifié de sorte que, à chaque nouvelle solution réalisable trouvée de coût z^*, une nouvelle contrainte $\texttt{sum}(Z) < z^*$ (pour un objectif de minimisation) est ajoutée au modèle forçant le calcul par la suite de solutions strictement meilleures. Ces indirections via la contrainte de somme et les contraintes en extension préviennent une propagation forte de la nouvelle borne supérieure globale $z^* - 1$ et la réduction des domaines des variables de décisions X. Les contraintes d'optimisation visent à supprimer ces indirections en les englobant.

3.2.1.1 Définition

Une contrainte globale se présente comme un ensemble de variables d'état et des restrictions sur les affectations simultanées de ces variables d'état. Elle est dite *d'optimisation* si on ajoute une *variable de coût* – une le plus souvent, mais il peut y en avoir plusieurs – et une *fonction de coût* liant les affectations des variables d'état aux valeurs de la variable de coût. Surtout, elle doit être équipée d'un propagateur filtrant le domaine des variables d'état à partir du domaine de la variable de coût par l'application inverse de la fonction de coût. Ce principe est appelé la *back-propagation*. Les *contraintes globales souples* sont un cas particulier des contraintes d'optimisation dans lequel la fonction de coût et les restrictions sur les variables d'état sont liées. Plus précisément, la fonction de coût mesure un degré de satisfaction/violation de la restriction. Il existe, par exemple, des variantes souples de `alldifferent` pour différentes mesures de violation de la condition que toutes la variables d'état doivent prendre des valeurs différentes deux-à-deux, comme la distance de Hamming ou le nombre de contraintes binaires de différence violées [PRB01]. On distingue également les contraintes d'optimisation des contraintes de cardinalité, par exemple, où la restriction porte simultanément sur les variables d'état et la variable de coût, comme la contrainte `among` où les affectations des variables d'état sont uniquement restreintes par les bornes de la variable de cardinalité.

Parmi les contraintes globales d'optimisation existantes, hors contraintes souples et de cardinalité, on peut citer par exemple les variantes pondérées des contraintes d'affectation `alldifferent` [FLM99] et `global-cardinality` [Rég99], des contraintes de packing [Trio1 ; CO10], des contraintes de chemins et circuits [GSW05 ; Ben+12] et les contraintes de langage avec pondérations. Dans la plupart des cas ci-dessus, la fonction de coût correspond à la somme de valeurs associées à l'affectation de chacune des variables $(f(X) = \sum_{i=1}^{n} a_{iX_i})$ et la variable de coût est une variable bornée, liée à la fonction de coût par la relation $Z \geq f(X)$.

La back-propagation consiste alors à calculer une borne inférieure L de la fonction $f(X)$ sur le domaine de X : si la borne supérieure de Z est strictement inférieure à L, $\overline{Z} < L$, alors la contrainte est irréalisable ; sinon, il s'agit d'identifier et de filtrer les affectations $X_i = j$ telles que $L \leq \overline{Z} < L_{|X_i=j}$ où $L_{|X_i=j}$ désigne la borne inférieure de la fonction $f(X)$ sur le sous-ensemble des tuples de X tels $X_i = j$.

Ces contraintes sont alors employées pour des problèmes de minimisation dont le critère d'optimisation est modélisé directement par la variable de coût Z de la contrainte. La back-propagation permet de propager plus efficacement la borne de la meilleure solution réalisable trouvée z^* aux variables d'état de la contrainte en identifiant et supprimant des tuples qui n'appartiennent à aucune solution strictement meilleure.

3.2.1.2 Minimisation et/ou maximisation

Le traitement asymétrique des variables d'état et de coût dans ce type de contraintes globales se justifie dans cet usage comme il n'est pas nécessaire ici de maintenir la stricte égalité $Z = f(X)$ et donc la borne inférieure de la variable Z. Par ailleurs, cette tâche, qui nécessite le calcul d'une borne supérieure de $f(X)$, n'est généralement pas duale à l'autre, voire l'une peut être facile (calculer un plus court chemin) et l'autre $\mathcal{NP}$-difficile (calculer un plus long chemin).

La contrainte `cost-regular` est originale – et la contrainte `multicost-regular` unique car elle traite plusieurs coûts – en ce sens qu'elle maintient à la fois les bornes inférieures et supérieures de la variable de coût. Dans ce cas, les calculs de plus court et plus long chemins sont en effet faciles comme ils se font dans des graphes acycliques. En fait, comme le montre le contre-exemple de la figure 2.7, la contrainte `cost-regular` assure la consistance hybride (variables discrètes/variables bornées) sur la conjonction de contraintes $\texttt{regular}(X, \Pi, Z) \wedge Z \leq f(X) \wedge Z \geq f(X)$ mais pas sur la conjonction $\texttt{regular}(X, \Pi, Z) \wedge Z = f(X)$.

Le maintien des deux bornes de la variables de coût, quand cela est possible, a plusieurs avantages : cela permet de traiter indifféremment un critère de minimisation ou de maximisation sans impacter le reste du modèle (il n'est pas nécessaire d'inverser les coûts par exemple) ; la valeur de la borne inférieure L calculée en interne à la contrainte est exposée au niveau du modèle en mettant à jour la borne inférieure de Z : $\underline{Z} := \max(\underline{Z}, L)$; les variables d'état et de coût sont plus fortement liées ce qui permet de casser la symétrie sur les valeurs de coût due à la contrainte $Z \geq f(X)$; la variable de coût n'est plus enfin une variable particulière de la contrainte globale, de sorte qu'elle peut être employée en dehors de la modélisation de la fonction objectif d'un problème (par exemple, comme une variable de cardinalité comme on l'a montré dans le cadre de `multicost-regular`).

3.2.1.3 Relaxations pour le filtrage

Une grande majorité des propagateurs des contraintes globales d'optimisation existantes sont issus de la recherche opérationnelle. Une première catégorie emploie la programmation dynamique comme pour `cost-regular` (voir section 2.3) et `knapsack` [Trio1] avant elle. Une seconde catégorie exploite, comme expliqué section 2.4, le principe de *variable fixing* [Wol98] par les coûts réduits, ces derniers étant obtenus soit directement par pro-

grammation linéaire (`cumulative` [HY02],`bin-packing` [CO10]) ou par relaxation lagrangienne (`shorter-path` [GSW05], `multicost-regular`, `weighted-circuit` [Ben+12]), soit par un modèle de graphe résolu par un algorithme spécialisé (`cost-gcc` [Rég99], `alldifferent` [FLM02]).

Programmation linéaire. L'application d'un solveur linéaire a l'avantage de la généricité et de la composabilité. Le principe de linéarisation automatique de Refalo [Refoo] via la transformation $X_i = j \iff x_{ik} = 1$ pourrait permettre, par exemple, d'obtenir un programme linéaire en variables binaires pour de nombreuses contraintes auxquelles on peut ajouter toute fonction linéaire de coût. Basées sur des variables communes, les programmes associés à différentes contraintes d'un modèle peuvent être composés. Le propagateur procède alors en résolvant la relaxation continue du programme linéaire en nombres entiers et à déduire des coûts réduits des variables, celles qui peuvent être fixées.

L'approche peut en fait être appliquée à la formulation linéaire de n'importe quelle relaxation du problème complet. Dans [DF15] par exemple, nous avons développé une contrainte redondante, dédiée au problème de minimisation du nombre de relocations des conteneurs empilés sur des quais de chargement (Container Relocation Problem ou CRP). Cette contrainte s'appuie sur une formulation linéaire en nombres entiers d'une relaxation originale du problème. Elle englobe une partie des éléments qui sont décomposés dans le modèle de contraintes auquel elle est ajoutée, pour un supplément de déductions. De plus, elle fournit une borne inférieure du problème global à chaque noeud de l'arbre de recherche à la manière d'un branch-and-bound en programmation mathématique. Dans nos expérimentations, il s'est avéré avantageux de résoudre la relaxation en nombres entiers plutôt qu'en continu, le surcoût de calcul étant rentabilisé par la meilleure qualité de la borne. En revanche, afin d'appliquer le filtrage sur les coûts réduits, il est nécessaire de résoudre ensuite la relaxation continue afin d'obtenir ces coûts réduits.

Différentes optimisations algorithmiques sont envisageables, par exemple : ne pas évaluer de manière exhaustive l'ensemble des variables du modèle en identifiant a priori celles qui ne peuvent être fixées pour une valeur de coût réduit donné ; exploiter pour le filtrage des informations obtenues directement depuis la résolution en nombres entiers (les variables fixées dans le prétraitement ou les coûts réduits à la racine du branch-and-bound par exemple) plutôt que de relancer la résolution en continue ; en cas d'irréalisabilité de la relaxation, obtenir du solveur linéaire un sous-ensemble irréalisable irréductible (IIS) de variables et leurs affectations incompatibles, et le traduire comme une nouvelle contrainte, appelée *nogood*, ajoutée au modèle de contraintes pour interdire cette affectation incompatible dans la suite de la recherche ; inversement, enrichir le modèle linéaire de coupes qui pourraient être déduites des contraintes globales du modèle.

Mais le principal défaut d'une telle approche relève de sa mise en œuvre coûteuse. Une autre piste d'amélioration consiste donc à optimiser l'appel du solveur linéaire au réveil de la contrainte par le moteur de propagation. Il s'agirait par exemple, de synchroniser les modèles linéaire et par contraintes afin de limiter le coût de communication, notamment dans l'intégration des déductions de filtrage au modèle linéaire ; et plus généralement, de synchroniser la relaxation linéaire avec les décisions de branchement, à la manière de ce qui se fait dans

les solveurs de programmation linéaire en nombres entiers, afin de rendre la résolution plus incrémentale. En effet, dans notre implémentation [DF15], le modèle linéaire est reconstruit intégralement à chaque backtrack et aucune information n'est conservée d'une résolution à l'autre. Enfin, les réveils du solveur linéaire, qui est une opération coûteuse, devraient être limités (au plus une fois par noeud ou à certains noeuds) et ajustés en fonction de conditions qui en évaluent le potentiel de filtrage.

Relaxation lagrangienne. Comme vue à la section 2.4, la relaxation lagrangienne offre une solution de filtrage, incomplète mais efficace et générique, pour ces contraintes $\mathcal{NP}$-difficile dont le modèle linéaire présente des équations couplantes ou complicantes. Formalisée par Sellmann [Sel04], elle n'a été mise en œuvre que sur un nombre limité de contraintes [CL97 ; Sel03 ; KBN05 ; MD09 ; CP12], toutes ou presque relatives au problème de plus court chemin avec contraintes de ressources dans des graphes divers. Dans ce contexte, en particulier, les sous-problèmes lagrangiens sont des problèmes de plus court chemin, qui ne nécessitent pas le recours à un solveur mathématique et sur lesquels on peut facilement appliquer un processus incrémental (par la suppression et l'ajout d'arcs).

Un autre intérêt de l'approche est qu'il n'est pas nécessaire de résoudre le dual lagrangien à l'optimum comme chaque résolution des sous-problèmes lagrangiens peut engendrer du filtrage, même si ce sont les sous-problèmes quasi-optimaux qui engendrent le plus de filtrage. De sorte, la résolution du dual lagrangien peut se faire de manière heuristique, rapide, par un simple algorithme de sous-gradient par exemple. Ici, en revanche, une paramétrisation est alors nécessaire pour une raison de performance ; qui peut être statique, dynamique (l'utilisateur peut modifier ces paramètres à la déclaration de la contrainte), voir automatique.

Un reproche fait à cette approche est la perte de la monotonicité [ST09]. Reposant sur des choix heuristiques, elle peut en effet générer moins de filtrage, après réduction de domaine (dans un arbre de recherche) ou suite à l'introduction de nouvelles variables ou contraintes. Cette perte ne garantit plus alors la reproductibilité de la résolution et elle rend sensibles alors l'analyse expérimentale des modèles, la correction de bugs et les mécanismes de recalcul et d'explications. Cette difficulté rejoint en fait les problèmes de variabilité des performances observée dans les solveurs modernes de programmation linéaire en nombres entiers [Dan08 ; LT13], dû à la multiplication des traitements heuristiques et aléatoires dans les différents composants du branch-and-bound : prétraitement, heuristiques de branchement, simplexe, génération de coupes. Un traitement à moindre de coût est de fixer la graine des générateurs de nombres aléatoires, mais cela peut rester insuffisant pour répondre à tous les défauts liés à la non-monotonicité. Malgré tout, l'intérêt pratique de concevoir des contraintes $\mathcal{NP}$-difficile, qui repose nécessairement sur des algorithmes approchés, n'est plus à démontrer et vient relativiser les défauts que leur mise en œuvre engendre.

3.2.2 Au-delà du filtrage

La PPC n'est a priori pas équipée pour optimiser et moins encore pour effectuer la preuve d'optimalité. Pour autant, elle est probablement l'alternative la plus sérieuse pour les problèmes difficiles pour la programmation mathématique, typiquement les problèmes d'ordonnancement [Lod10 ; Sch+13], où savoir caractériser l'optimum a un intérêt autant pratique –

quand un point de l'objectif se traduit en montants financiers importants par exemple – que théorique – pour estimer la valeur d'une heuristique plus rapide et scalable. Sur des problèmes pratiques, quand l'optimum n'est pas requis, elle apporte la flexibilité dont les approches impératives telles que les metaheuristiques manquent.

Pour être compétitive avec ces approches, il est nécessaire de prendre en compte de manière plus intensive le critère d'optimisation dans l'algorithme de backtracking, notamment en s'inspirant, puisque les deux méthodes sont similaires, des branch-and-bound modernes en programmation linéaire en nombres entiers. Ces derniers ont en effet bénéficié d'améliorations considérables sur ces 10 dernières années par l'ajout de composants qui sont pour la plupart portables à la programmation par contraintes.

3.2.2.1 Bornes et coupes

Les contraintes globales d'optimisation sont une manière par exemple d'implémenter le calcul de borne inférieure (pour un problème de minimisation) et la génération de coupes, ici le filtrage. Quand les contraintes d'optimisation sont généralement employées pour modéliser le critère d'optimisation sur un sous-problème, les travaux récents sur la décomposition lagrangienne [HQR15 ; BCH15a] et notre contrainte pour le CRP [DF15] s'en distinguent en ce sens que, dédiées au critère d'optimisation et redondantes, elles viennent doubler les contraintes de satisfaction du modèle comme elles englobent une relaxation du modèle global. On pourrait alors qualifier de *contraintes d'optimalité* ces contraintes équipées, non pas seulement d'un back-propagateur, mais également de différents services sur lesquels appuyer des techniques d'accélération de la recherche propres à l'optimisation.

3.2.2.2 Stratégies de branchement

Ainsi, les heuristiques de recherche ou stratégie de branchement sont un autre composant prépondérant dans la performance des branch-and-bound modernes. Les stratégies en programmation linéaire en nombres entiers s'orientent en fonction de la borne ou de la solution de la relaxation continue. À la suite de Focacci et al. [FLM99], nous avons appliqué ce principe (voir section 2.3.5) en donnant accès aux calculs internes des contraintes d'optimisation `cost-regular` et `multicost-regular` dont, principalement, la solution optimale de la relaxation considérée par le propagateur (le plus court chemin du graphe), mais également la meilleure solution possible dans la relaxation pour n'importe quelle instantiation donnée (le plus court chemin passant par un arc correspondant à l'instantiation). À partir de ces informations, nous avons défini différentes heuristiques de recherche, notamment les heuristiques basées sur la notion de regret. Nos expérimentations ont montré le fort impact de ces heuristiques sur l'accélération de la recherche.

Deux questions se posent à ce sujet. La première concerne la composabilité de tels services. Un modèle est en effet composé de plusieurs contraintes globales et il peut être contre-productif d'orienter la recherche sur la base des informations de l'une des contraintes uniquement : cela peut engendrer un manque de diversification en concentrant la recherche autour des solutions du sous-problème défini par la contrainte globale que, potentiellement, les autres contraintes du modèle tarderaient (profondément dans l'arbre de recherche) à éliminer. Il est

nécessaire alors de pouvoir composer les heuristiques. Pour le problème de Nurse Scheduling, nous avons ainsi examiné des heuristiques basées sur différentes compositions des informations des contraintes `multicost-regular` centrales au modèle. La composition manuelle de telles heuristiques est facile à mettre en œuvre mais l'infinité des possibilités de combinaison nécessiterait une étude expérimentale poussée pour identifier, éventuellement de manière automatique, les combinaisons les plus favorables.

La seconde question porte sur la stratégie de parcours de l'arbre de recherche. Celle-ci est basée sur la recherche en profondeur (DFS) uniquement en programmation par contraintes tandis qu'elle est principalement basée sur la recherche du meilleur d'abord (BFS) en programmation mathématique, exploitant plus encore les heuristiques de type *strong-* ou *reliability-branching* qui évaluent la force de toutes les décisions de branchement ouvertes avant de sélectionner la meilleure. Ce parcours de recherche en largeur a un coût, en mémoire surtout, puisque le nombre de nœuds ouverts est bien plus important, mais le gain en performance est à la mesure. Il serait intéressant d'implémenter les parcours BFS dans les solveurs de contraintes et imaginer alors les services qui pourraient être offerts par les contraintes d'optimalité pour implémenter efficacement ces heuristiques de *strong-branching*. Pareillement, il s'agirait d'identifier comment combiner les heuristiques basées sur les contraintes globales avec les approches de recherche dynamique en programmation par contraintes basées sur les nogoods qui se sont avérées performantes récemment pour la résolution de problèmes d'optimisation en ordonnancement (voir par exemple [Sch+13]).

3.2.2.3 Heuristiques primales

Les heuristiques primales sont un autre composant fondamental des branch-and-bound modernes. Il s'agit de la recherche systématique à chaque nœud ou à certains nœuds de l'arbre de recherche de solutions réalisables de bonnes qualités afin notamment de descendre la borne supérieure (toujours en cas de minimisation) plus fréquemment qu'aux seules feuilles de l'arbre de recherche. Ce principe peut être adapté à la programmation par contraintes de différentes manières.

Les heuristiques génériques de type *diving* ou *local branching* par exemple, consistent à effectuer la descente d'une branche de l'arbre de recherche sans backtracking possible. Celles-ci peuvent être implémentées directement dans le backtracking du solveur de contraintes. Les solutions (relâchées) optimales données par les contraintes globales d'optimisation du modèle peuvent être alors employées pour guider la descente.

L'implémentation d'heuristiques spécifiques à un problème peut également être envisagée au sein d'une contrainte globale dédiée, sans toucher ainsi au moteur de résolution. Une telle contrainte a cependant un statut particulier : elle ne participe pas à la réduction de domaine et son réveil doit être limité, à une occurrence par nœud par exemple. Elle est aussi dédiée à un problème et on perd ainsi la notion de réutilisabilité propre aux contraintes globales. Dans la contrainte d'optimalité que nous avons développée pour le CRP [DF15], nous proposons par exemple d'employer une recherche gloutonne à partir de la solution de relaxation linéaire. La réalisabilité de la solution obtenue est validée par l'appel des *checkers* des autres contraintes du modèle. On pourrait imaginer un schéma plus générique de recherche locale autour de la solution relâchée optimale calculée par une des contraintes globales d'optimisation (ou composée

depuis plusieurs de ces contraintes), où le voisinage serait obtenu par réplication du modèle complet centré autour d'une instantiation partielle des variables issue de la solution relâchée.

3.2.2.4 Prétraitement

Le prétraitement des modèles linéaires (par fixation de variables, ajout/suppression/reformulation de contraintes) est l'ingrédient le plus important de l'amélioration des performances des branch-and-bound modernes [Ach+16]. À noter que certaines de ces techniques reposent sur des déductions logiques à la manière d'une résolution par contraintes. Des techniques similaires existent au niveau des contraintes globales dans les solveurs de contraintes : il s'agit par exemple du réveil initial de la contrainte globale, avec la construction des structures de données internes durant laquelle des réductions de domaines peuvent être réalisées ; ou encore, quand plusieurs propagateurs sont disponibles pour une même contrainte globale, de l'association du propagateur le plus adapté à l'instance particulière modélisée par l'utilisateur. L'outil d'agrégation des automates pondérés que nous avons développé pour la contrainte `multicost-regular` ou les techniques d'apprentissage de contraintes sont, comme dit précédemment, ce qui se rapproche le plus de la reformulation semi-automatisée des contraintes. Les opportunités d'améliorer de manière automatique les modèles de contraintes en présence d'un critère d'optimisation sont encore à étudier.

3.2.3 Intégration à une méthode d'optimisation

La section précédente présente des idées d'amélioration du backtracking en contraintes pour mieux tenir compte du critère d'optimisation. Cette dernière section s'attache à une autre voie, qui est en fait complémentaire à celle-ci, consistant à intégrer dans une méthode dédiée à l'optimisation, approchée ou exacte, le modèle de contraintes d'un sous-problème sur lequel on requiert un certain degré d'expressivité et de flexibilité.

3.2.3.1 Recherche locale à voisinage large

Les metaheuristiques sont les méthodes les plus répandues en optimisation combinatoire appliquées. Elles semblent bénéficier souvent de l'image tenace de la non scalabilité des approches déclaratives de la programmation mathématique ou par contraintes. Elles ne requièrent pas de solveur (commercial coûteux), ni de connaissance pointue en (bonne) modélisation et elles reposent sur des schémas algorithmiques plutôt faciles à implémenter. Bien que ces schémas soient génériques, leur mise en œuvre sur un problème donné repose sur un encodage, qui lui est propre, des solutions, des mouvements locaux, des contraintes du problème qui définissent ces solution et des fonctions coûts. De fait, les approches metaheuristiques doivent être implémentées intégralement ou presque, avec les problèmes de bugs et de maintenance que cela entend. De plus, une modification de la définition du problème se traduit par la réimplémentation d'une partie de l'algorithme, éventuellement accompagnée d'un reparamétrage fastidieux.

C'est en cela que metaheuristiques et approches déclaratives sont le plus différentes et complémentaires – plutôt que dans l'opposition résolution approchée/exacte. L'idée de les hybrider s'est largement répandue, motivée par exploiter leurs forces respectives. Les recherches

locales à voisinage large (LNS) [Sha98] sont l'une des formes d'hybridation les plus étudiées : le voisinage d'une solution est alors décrit par un modèle centré autour de la solution partiellement relâchée et son exploration réalisée par un solveur, tandis que le parcours non exhaustif rapide de l'espace de recherche, de solution en solution, suit le principe de la recherche locale, guidé par le critère d'optimisation. Cette approche repose ainsi sur un modèle déclaratif des contraintes du problème, dans la définition des voisinages, au lieu d'un traitement algorithmique direct, et le solveur de programmation mathématique ou par contraintes sera d'autant plus rapide que le voisinage à explorer est de petite taille.

Comme il est décrit dans la section 2.4.3, nous avons appliqué une telle recherche locale hybridée à la programmation par contraintes pour le problème de Nurse Scheduling. Dans de nombreux benchmarks, notamment celui de la compétition PATAT 2010, toutes les règles de travail sont souples, autrement dit, toutes les affectations satisfaisant la charge globale de travail sont réalisables. Les metaheuristiques sont particulièrement performantes, offrant un parcours non exhaustif de cet ensemble de recherche aussi vaste, et dans ce cas où la définition du voisinage est simple et où des fonctions évaluent rapidement le coût de pénalité pour chacune des contraintes souples. Remplacer la définition du voisinage par un modèle de contraintes, basé sur l'outil de modélisation de `multicost-regular`, a un premier objectif de répondre au besoin de flexibilité et d'extensibilité dans la description des règles de travail pour les problèmes de planification de personnel sur-contraints (au-delà des benchmarks et des compétitions qui n'évaluent généralement pas ce besoin).

L'autre objectif est de pouvoir mieux exploiter la présence de contraintes dures dans la définition du problème. En effet, les contraintes dures peuvent être néfastes aux metaheuristiques qui les traitent, soit au niveau de la définition des voisinages qui s'en retrouve complexifiée, soit au moyen d'une fonction de coût infini, multipliant alors le nombre de non-solutions évaluées inutilement. Inversement, les contraintes souples sont handicapantes pour un modèle de contraintes car le filtrage, ne reposant plus que sur la back-propagation de la somme des coûts de pénalité, est souvent limité (de nombreuses solutions ont un coût global équivalent), et il dépend très fortement de la stratégie de branchement. Ainsi, l'approche par contraintes peut rester compétitive dans ce contexte particulier si on soigne la back-propagation et la stratégie de branchement qui doit être guidée par les coûts. Incrémenter progressivement la valeur du coût global jusqu'à trouver une première solution réalisable (optimale donc) est par exemple envisageable si on dispose d'une bonne borne inférieure. Plus généralement, brancher sur les variables de coût est une manière d'intensifier la back-propagation. Par ailleurs, dans nos expérimentations sur le Nurse Scheduling, la stratégie de branchement basée sur la notion de regret calculé par les contraintes `multicost-regular` s'est avérée la plus performante. Trouver la combinaison optimale de ces stratégies demanderait de pousser l'étude expérimentale plus encore.

En décomposant l'espace de recherche en une succession de voisinages à explorer, la recherche locale se présente comme une méthode décomposition spatiale et son efficacité dépend en grande partie du grain de décomposition, i.e. la taille des voisinages, choisi. Implémenter le voisinage d'une solution dans un modèle de contraintes consiste trivialement à fixer une partie des variables du modèle à leur valeur dans la solution. Choisir ce sous-ensemble de variables et notamment sa taille est, en revanche, une tâche importante complexe qui nécessite de bien mettre en balance la complexité du modèle obtenu et la diversité des solutions qu'il engendre.

Dans le contexte du Nurse Scheduling, on retrouve par exemple deux types de mouvements locaux : l'échange d'activités entre personnels ou entre périodes. Elles se traduisent dans une recherche à voisinage large, respectivement par : altérer totalement les horaires d'un sous-ensemble d'employés, ou altérer partiellement sur un sous-ensemble de périodes les horaires de tous les employés. La première approche est coûteuse pour notre modèle car elle nécessite de repropager intégralement les contraintes `multicost-regular` associées aux employés sélectionnés. La seconde approche sera coûteuse si le nombre des périodes choisies est trop important. Dans nos expérimentations, nous avons fixé ce nombre de manière quasi-aléatoire à chaque itération de la recherche locale. Une telle recherche locale à voisinages variables pourrait bénéficier d'être mieux dirigée en réglant la taille des voisinages en fonction de la procédure globale ou bien de la solution dont le voisinage est considéré. Par exemple, si une solution est pénalisée plus lourdement par une contrainte souple du problème, elle pourrait être relâchée relativement à cette contrainte particulière. Un tel service pourrait être implémenté au niveau des contraintes globales du modèle, à la manière de ce que nous avons proposé avec les modèles des contraintes utilisateurs de BtrPlace qui, pour chacune, offre d'identifier un ensemble minimal d'instantiations des variables à relâcher pour lever la violation de la contrainte (voir section 2.1.4). Un tel service est également envisageable au niveau d'une contrainte agrégée comme `multicost-regular` : chaque pénalité étant associée à une dimension du vecteur de coût, les instantiations responsables de la pénalité sont facilement identifiables dans le graphe déployé, portées par les arcs de plus haut coût dans la dimension correspondante. Une définition réfléchie et automatisée des voisinages est donc envisageable dans une approche par contraintes et elle offre un potentiel d'amélioration du LNS, à condition cependant que le surcoût lié au choix du voisinage soit limité et qu'une part d'aléatoire reste présente pour assurer une certaine diversification dans la recherche.

On notera enfin que le schéma de recherche local à voisinage large hybridé à la programmation par contraintes repose sur quelques éléments qu'il est facile d'implémenter au-dessus d'un solveur de contraintes. J'ai ainsi développé dans le solveur Choco 2 [3], au-dessus de et comme alternative à la méthode de backtracking, un moteur de recherche locale généralisant celle que nous avions appliquée au Nurse Scheduling. Le code de quelques centaines de lignes automatise la procédure globale et permet à l'utilisateur de spécifier la stratégie de branchement pour l'exploration du voisinage, la stratégie de *débranchement* pour la définition du voisinage autour d'une solution, et quelques paramètres de diversification et de convergence.

3.2.3.2 Méthodes de décomposition

Les méthodes de décomposition en programmation mathématique offrent un cadre d'hybridation naturel : les sous-problèmes sont souvent résolus par programmation dynamique en génération de colonnes ou approchés par simulation, par exemple, en décomposition de Benders. Le couplage de la programmation par contraintes et des méthodes de décomposition de la programmation mathématique fait partie des travaux fondateurs de l'hybridation de la programmation par contraintes et des outils de la recherche opérationnelle. Les schémas de couplage à la génération de colonnes [Jun+99], à la relaxation lagrangienne [Sel04], à la décomposition de Benders [Tho01 ; EW01 ; HO03] ont été inventés et expérimentés sur quelques cas

3. réimplémenté depuis dans les nouvelles versions de Choco

d'études principalement dans la première moitié des années 2000. Ces travaux ont été à la base des efforts de rapprochement des communautés de recherche opérationnelle et de programmation par contraintes, mais ils ont relativement peu essaimé depuis, comparés aux *matheuristiques* et LNS hybrides.

Outre le fait que les méthodes approchées sont généralement considérées comme suffisantes dans un contexte applicatif, la difficulté théorique et pratique des méthodes de décomposition hybrides est en effet probablement plus grande : mettre en œuvre une génération de colonnes ou définir une contrainte globale sont des tâches très différentes et elles sont souvent plus complexes que définir un mouvement de recherche locale. Surtout, leur implémentation nécessite de coupler des solveurs qui n'ont pas vocation initialement à être compatibles. À mesure que les solveurs s'améliorent, leur fonctionnement devient souvent plus opaque, complexifiant encore un peu plus la tâche de synchronisation. Il y a une étude intéressante à mener ici sur les possibilité de synchronisation des solveurs, démarrage à chaud.

Une seconde raison, non scientifique, est liée à l'éloignement des deux communautés scientifiques. Si celles-ci ne sont plus cloisonnées, les chercheurs qui se revendiquent des deux à la fois ne sont aujourd'hui pas beaucoup plus nombreux que 15 ans auparavant, et il n'est pas rare de lire des revues de littérature sur un problème donné qui ignorent les travaux menés dans l'une ou l'autre des communautés. Suivre l'évolution et maîtriser les outils développés de part et d'autre présente une difficulté supplémentaire qui, au pire, n'est pas considérée et, au mieux, qui n'apparaît pas comme rentable.

Ces méthodes sont pourtant propices à l'optimisation de systèmes complexes dont certains composants sont difficilement linéarisables, ou dont l'obtention d'un certificat d'optimalité est requis. L'approche déclarative et modulaire leur confère expressivité, flexibilité et réutilisabilité. À ce propos, il serait intéressant de voir, par exemple, si la méthode de génération de colonnes hybride que nous avons développée pour les problèmes de planification de personnel (voir section 2.3) s'appliquerait avantageusement, autant en termes de temps de calcul qu'en temps de développement, aux nombreuses approches de génération de colonnes de la littérature sur ces problèmes pour lesquelles des modules de programmation dynamique ont été spécifiquement développées.

Par ailleurs, l'évolution des performances de ces méthodes est directement liée à celle des solveurs de programmation mathématique et par contraintes dont les progrès ont été considérables ces dernières années. Elle est aussi liée aux problèmes de convergence et d'instabilité typiques de ces approches, et pour lesquels différents instruments correctifs (stabilisation, agrégation de contraintes, etc.) ont été proposés. La compatibilité de ces correctifs avec le sous-problème de contraintes, leur adaptation éventuelle et leur impact n'ont peu ou pas été étudiés dans le cadre des méthodes de décomposition hybrides.

Enfin, il sera intéressant de voir si le regain d'intérêt actuel pour les méthodes de décomposition, notamment pour la génération de colonnes et surtout la décomposition de Benders, s'accompagne d'un renouveau de la recherche et des applications des méthodes de décomposition hybridées à la programmation par contraintes.

Chapitre 4

Perspectives

Décomposition, hybridation, optimisation et flexibilité sont toujours des mots-clés dans les perspectives de recherche dont j'ai entamé les travaux. À ceux-là s'ajoutent d'autres domaines et contextes d'application et d'autres paradigmes de résolution et outils.

Énergie/climat. Concernant les domaines d'application, mon affectation en 2014 au Centre de Mathématiques Appliquées (CMA) a été motivée par et m'amène à concentrer mes travaux autour des problématiques d'optimisation dans le domaine de l'énergie et du climat : *concentrer* car, si les travaux sur les techniques de résolution se nourrissent des expériences transverses dans différents domaines applicatifs, trop diversifier les sujets d'étude est une approche coûteuse en temps d'appropriation et de veille ; et le domaine de l'*énergie/climat* car, au-delà de mes convictions personnelles, c'est un domaine vaste et un sujet actuel avec une portée importante, qui soulève des problématiques d'optimisation qui évoluent, se multiplient et se complexifient. Aussi, ces problématiques présentent certaines caractéristiques techniques auxquelles je porte un intérêt particulier.

Programmation non-linéaire. Ainsi, une caractéristique des problématiques du domaine est la présence de phénomènes physiques non-linéaires, typiquement les flux d'énergie. Traduire exactement ces phénomènes dans une méthode d'optimisation est impossible mais, en contrepartie, cela permet d'envisager différentes approches de modélisation et donc de résolution, afin de répondre au problème réel de manière la plus fidèle possible (ou requise). Historiquement, ces phénomènes étaient souvent modélisés par approximation en des fonctions linéaires par morceaux, substituant la discrétisation à la non-linéarité. Avec les progrès rapides des solveurs de programmation linéaire en nombres entiers, des problèmes de taille de plus en plus importante ont pu être résolus. De manière concomitante, la même communauté cherche maintenant à appliquer aux solveurs non-linéaires, les ingrédients qu'elle a développés pour les solveurs linéaires. Le récent essor des solveurs quadratiques et/ou convexes, par exemple, permet aujourd'hui d'attaquer les problématiques d'optimisation dans l'énergie par de nouvelles approches.

Avec Gratien Bonvin, dont j'encadre la thèse depuis fin 2014, nous avons ainsi proposé dans [Bon+17] une nouvelle approche pour le problème de contrôle optimal dans les réseaux de distribution d'eau potable. Un réseau muni de châteaux d'eau et de pompes peut être vu comme un système de stockage de l'énergie, car il permet de décorréler les moments d'approvisionnement des consommateurs (par un système gravitaire depuis les châteaux d'eau) des temps de pompage (où l'électricité est consommée afin d'alimenter les châteaux d'eau). Étant donnés une courbe de demande en eau et un tarif variable de l'électricité sur l'horizon d'une journée par exemple, le problème consiste à ordonnancer l'allumage des pompes de manière

à minimiser le coût énergétique tout en satisfaisant la demande. Ce problème présente deux facettes non-linéaires : (i) les coûts de fonctionnement des pompes, les pertes de charge dans les tuyaux et l'élévation de la charge par les pompes sont des fonctions non-linéaires du débit de l'eau mais qui sont correctement approximées par des fonctions quadratiques, et (ii) les évènement discrets d'allumage des éléments actifs (pompes et valves). Ce problème est traité de différentes manières dans la littérature et, parmi les méthodes exactes [1], on trouve : la relaxation continue, l'approximation linéaire par morceaux, ou encore la décomposition de Benders où les flux d'eau sont simulés dans le problème esclave. Notre approche consiste à relâcher l'égalité dans les fonctions d'équilibre des charges de façon à obtenir un programme en nombres entiers à contraintes quadratiques convexes qu'on peut maintenant implémenter dans les versions récentes des solveurs Gurobi et Cplex. Grâce à la performance de ces nouveaux solveurs, l'approche domine les autres solutions dans nos expérimentations sur différents benchmarks de la littérature.

Dans une autre approche, qui semble plus prometteuse encore, nous proposons de précalculer l'équilibre des charges en fonction de toutes les combinaisons d'allumage possibles des pompes, à la manière des méthodes de relaxation préemptive en ordonnancement. Le problème se traduit alors en un modèle linéaire continu qui décide des moments et durées d'allumage des combinaisons pour satisfaire la demande et minimiser le coût. Pour être scalable, une méthode de décomposition, typiquement la génération de colonnes, est envisagée afin de générer de manière progressive seul un sous-ensemble de combinaisons. Il s'agirait d'une méthode de décomposition hybride où l'équilibre des flux serait calculé par simulation dans le sous-problème.

Échelles temporelles. Une autre caractéristique significative du domaine énergie/climat est la présence de problèmes d'optimisation à tous les niveaux de la décision (opérationnel/court-terme, tactique/moyen-terme et stratégique/long-terme) et leurs forte intrication. Par exemple, le coût opérationnel d'un réseau de distribution d'eau dépend de son dimensionnement : si les pompes sont sur-dimensionnées, le pompage coûtera plus cher. Inversement, le dimensionnement d'un réseau d'eau doit être établi en fonction de la prévision de son coût opérationnel. À ce jour, les problèmes de dimensionnement sont traités sur des réseaux d'eau sans éléments actifs, donc sans coût opérationnel. À terme, les travaux de Gratien Bonvin pour accélérer le contrôle optimal ont vocation à être intégrés dans une solution de dimensionnement pour des réseaux avec éléments actifs.

La recherche opérationnelle s'applique le plus souvent à résoudre des problèmes opérationnels justement. Les problèmes d'optimisation à moyen et long terme sont moins étudiés. À mesure que les méthodes se perfectionnent, elles permettent de traiter des problèmes de plus en plus gros et complexes et les problèmes relatifs au dimensionnement des systèmes ou à la tarification, qui nécessitent d'évaluer conjointement investissement et fonctionnement, en font partie. Les méthodes de décomposition telles Benders sont les plus indiquées pour réaliser cette évaluation conjointe.

Le dimensionnement des systèmes énergétiques à l'échelle régionale, nationale ou interna-

1. i.e. une méthode qui résoud à l'optimum un certain modèle mathématique, nécessairement imparfait, du problème réel et de ses phénomènes physiques

tionale est une problématique qui se pose à très long terme : 20 ou 50 ans pour tenir compte de la durée de vie des composants, 50 ans ou plus si on souhaite évaluer les impacts environnementaux et les orientations des choix énergétiques. À cette échelle où les données deviennent des inconnues, il n'est plus question de prévision mais de prospective : il s'agit d'éclairer les décideurs quant aux choix politiques, technologiques et économiques, comme évaluer les conditions de sortie du nucléaire [MA14] ou du doublement de la part de marché des énergies renouvelables en 2050 [Kra+16]. Parmi les outils de la modélisation prospective, se distinguent les approches *bottom-up* qui partent d'une description technico-économique détaillée du système, comprenant tous les flux énergétiques et leurs vecteurs, des producteurs aux consommateurs, les échanges internes et avec l'extérieur, les processus de transformation et leurs coûts, etc. Un scénario énergétique, c'est à dire un ensemble de conditions sur l'état du système, est appliqué. Le modèle retourne alors un chemin possible de transition du système pour atteindre un état satisfaisant ces conditions à l'horizon considéré, et de moindres coûts d'investissement et de fonctionnement. Une approche de programmation linéaire pour résoudre ces problèmes de planification long-terme, la famille des modèles MARKAL/TIMES, a été développé par l'ETSAP (Energy Technology Systems Analysis Program) un consortium international sous l'égide de l'Agence internationale de l'énergie. Le CMA est le représentant français de ce consortium et il développe, maintient et enrichit ses propres modèles, dont le modèle TIMES-FR du système multi-énergétique de la France. Ce contexte applicatif original de la programmation mathématique porte de très nombreuses voies d'amélioration de l'outil, notamment dans l'expressivité du modèle et dans la vitesse de résolution. Je m'intéresse, entre autres, à l'intégration d'éléments discrets dans le modèle (comme les niveaux étagés de production d'une centrale par exemple) et à comment corriger leur impact sur les temps de résolution, par décomposition temporelle ou spatiale.

Incertitudes. Finalement, l'incertitude sur les données prévisionnelles est une caractéristique du domaine de l'énergie/climat et de l'optimisation à moyen et long terme. L'optimisation robuste est une évolution de l'optimisation déterministe pour prendre en compte ces incertitudes. Il en résulte des modèles de grande taille mais qui supportent une décomposition bi-niveau évidente comme minimiser un critère sous un degré d'incertitude maximal. L'intérêt pour les modèles d'optimisation robuste a redoublé récemment en partie car, comme pour les modèles de programmation non-linéaire et d'optimisation à moyen/long terme, leur taille et leur complexité commencent à être abordables pour les solveurs matures modernes. Avec la thèse d'Arnold N'Goran, débutée en février 2017, nous aborderons la problématique du contrôle optimal des micro-grids sous la double exigence de robustesse vis-à-vis : (i) de l'incertitude sur les données prévisionnelles de production des énergies intermittentes, de demande et de tarification, et (ii) d'une variété de systèmes de micro-grids et de critères d'optimalité.

Annexe A

Curriculum Vitae

Parcours universitaire

- **2000 - 2003 : Doctorat Informatique, Optimisation Combinatoire**
  Université d'Avignon – Laboratoire Informatique d'Avignon FRE 2487

  — Titre du mémoire : *Méthodes hybrides de programmation par contraintes et de programmation linéaire pour le problème d'ordonnancement de projet à contraintes de ressources*
  — Directeurs : Christian Artigues (MCF), Philippe Michelon (Prof.)
  — Rapporteurs : Philippe Baptiste (CR-HDR, CNRS), Maurice Queyranne (DR-Prof., UBC Vancouver)
  — Examinateurs : Jacques Carlier (Prof., UTC), Claude Le Pape (HDR, ILOG S.A.), Alain Quilliot (Prof., U. Clermont-Ferrand II)
  — Mention : Très Honorable avec Félicitations

- **1999 - 2000 : DEA Informatique, Optimisation Combinatoire**
  Université d'Aix-Marseille II – Laboratoire Informatique d'Avignon FRE 2487

  — Titre du mémoire : *Borne inférieure pour l'ordonnancement de projet à moyens limités*
  — Directeurs : Christian Artigues (MCF), Philippe Michelon (Prof.)
  — Mention : Très Bien (1ère/16)

- **1997 - 1998 : DEA Mathématiques Fondamentales, Algèbre Commutative**
  Université d'Aix-Marseille I – Laboratoire d'Analyse, Topologie et Probabilités UMR 6632

  — Titre du mémoire : *Élasticité dans les anneaux de Dedekind*
  — Directeur : Paul-Jean Cahen (Prof., Aix-Marseille III)
  — Mention : Bien (2ème/12)

Parcours professionnel

- **depuis mai 2014 : Maître-Assistante** *École des Mines de Paris*
  — équipe *Centre de Mathématiques Appliquées* (dir. Nadia Maïzi)
  — enseignements : *Mastère Spécialisé Optimisation des Systèmes Énergétiques*

- **sept. 2006 - avril 2014 : Maître-Assistante** *École des Mines de Nantes*
  — équipe *TASC* (dir. Nicolas Beldiceanu) du *Laboratoire Informatique Nantes-Atlantique - CNRS UMR 6241 ; INRIA Rennes*
  — enseignements : *Département Informatique* (L2-M2, ∼150h/an) et *Master Operational Research and Optimization* de l'Université de Nantes (M2, 30h/an)
  — responsable du Master *Génie Informatique pour l'Aide à la Décision* (2006-2011, +200h/an) ; responsable de la refonte (2007-2008, +500h)
  — **avril 2012 - avril 2014 : congé parental**

- **sept. 2005 - août 2006 : Maître-Assistante Associée** *École des Mines de Nantes*
  — équipe *Contraintes Discrètes* (dir. Nicolas Beldiceanu) du *Laboratoire Informatique Nantes-Atlantique - CNRS FRE 2729*
  — enseignements : *Département Informatique* (L2-M2, ∼100h)

- **mars 2005 - juil. 2005 : Post-Doctorat** *Omega Optimisation Inc., Montréal, Canada*
  — société de services et d'édition de logiciels en optimisation
  — emploi conjointement subventionné par le programme canadien MITACS

- **sept. 2004 - fév. 2005 : Post-Doctorat** *École Polytechnique de Montréal, Canada*
  — séjour au *Centre de Recherche sur les Transports (CIRRELT)* sur invitation de Louis-Martin Rousseau (Prof.) et de Gilles Pesant (Prof.)

- **oct. 2003 - août 2004 : ATER** *Université d'Avignon*
  — équipe *Recherche Opérationnelle et Optimisation* (dir. Philippe Michelon) du *Laboratoire Informatique d'Avignon - LIA FRE 2487*
  — enseignements : *IUP Génie Informatique et Mathématique* (L3-M2, 192h)

- **oct. 2000 - déc. 2003 : Doctorante/Monitrice ESR** *Université d'Avignon*
  — équipe *Recherche Opérationnelle et Optimisation* (dir. Philippe Michelon) du *Laboratoire Informatique d'Avignon - LIA FRE 2487*
  — enseignements : *IUP Génie Informatique et Mathématique* et *Faculté des Sciences* (L1-M2, 64h/an)

Encadrements

Doctorats

- Julien Menana, *Automates et contraintes pour la planification de personnel*, soutenue en octobre 2011.
  Directeur : Narendra Jussien, Responsable scientifique : Sophie Demassey
- Aurélien Merel, *Évaluation biobjectif de la capacité d'infrastructures ferroviaires par génération de colonnes hybride*, soutenue en octobre 2012.
  Directeur : Xavier Gandibleux, Responsables scientifiques : Sophie Demassey, Xavier Gandibleux
- Xavier Libeaut, *Multi-flots dynamiques avec synchronisation de ressources*, soutenue en décembre 2013.
  Directeur : Éric Pinson, Responsables scientifiques : Éric Pinson, Jorge Mendoza, Sophie Demassey
- Gratien Bonvin, *Optimisation énergétique de réseaux de distribution d'eau : du court au long terme*, démarrée en décembre 2014.
  Directeur : Nadia Maïzi, Responsable scientifique : Sophie Demassey
- Rémy Doudard, *Arbitrage des solutions de flexibilité pour le climat*, démarrée en octobre 2015.
  Directeur : Nadia Maïzi, Responsables scientifiques : Edi Assoumou, Sophie Demassey
- Arnold N'Goran, *Contrôle optimal et gestion énergétique d'une station d'énergie autonome par Optimisation Robuste*, CIFRE Bertin Technologie démarrée en février 2017.
  Directeur : Nadia Maïzi, Responsables scientifiques : Sophie Demassey, Sébastien Thiry

Stages de recherche

- Dimitra Ignatiadis (2017), Mastère Spécialisé OSE, Mines ParisTech
- Yvann Nzengue (2015), Mastère Spécialisé OSE, Mines ParisTech
- Rachid Kalèche (2008), M2 international ORO, Université de Nantes
- Thierry Garaix (2004), DEA Université d'Avignon, en co-direction à 40% avec C. Artigues
- 1 stage ISIMA, 2 stages M1 Informatique

Animation Scientifique

Expertise (comités de programme)

— International Conference on Constraint Programming **CP** senior PC ('17), technical track ('15), application track ('12, '13), doctoral program ('11, '14)
— International Joint Conference on Artificial Intelligence **IJCAI**'16
— International Conference on Integration of AI and OR in CP **CPAIOR** ('11, '12)
— International Conference on Automated Planning and Scheduling **ICAPS**'12
— Journées Francophones de PPC **JFPC** ('07, '08, '09, '15, '16)

— Workshops internationaux MLS+CP'05, MELO ('11, '16)
— Conférences nationales ROADEF ('12,'15,'16,'17), JDIR'09

Animation de groupes de travail

– 2016-2017 : vice-présidente de la **ROADEF** (Société française de Recherche Opérationnelle et d'Aide à la Décision) en charge des relations extérieures
– 2009-2012 : membre initiateur du comité de pilotage du **Groupement Ligérien en RO** (LigéRO), projet émergence régional
– 2004-2009 : coordinatrice du groupe de travail **Contraintes et RO** (GdRs ALP et RO)

Diffusion et logiciels

– **Global Constraint Catalog** : conception/maintenance du site en ligne `http://sofdem.github.io/gccat/` (env. 50 visites/jour)
– **ChocoETP** : développements (Java/LGPL) `https://github.com/sofdem/chocoETP`
– contributions à la bibliothèque **Choco** (Java/BSD) `http://choco.emn.fr`
– contributions aux gestionnaires de ressources **Entropy** (Java/LGPL) puis **BtrPlace** (Java/LGPL) `http://btrp.inria.fr/` ; au prototype du projet FP7 **DC4Cities**

Organisation de colloques

– membre des comités d'organisation de la conférence internationale CP'06, des conférences nationales ROADEF'03, JFPC'08, du workshop international MLSCP'05
– organisation du prix Jeune Chercheur de la ROADEF '16,'17
– organisation de 3 journées thématiques *Contraintes et RO* et de 5 sessions spéciales aux congrès ROADEF et JFPC
– animation des *Jeudis de l'Optimisation* des Mines de Nantes (2007-2009)

Responsabilités collectives

– 2014 - : organisation du séminaire scientifique mensuel du CMA de Mines Paristech
– 2000 - 2004 : représentante élue des personnels non-permanents au comité scientifique et au comité de direction du Laboratoire Informatique d'Avignon
– rédaction, conception et maintenance des sites web : des équipes TASC du LINA (2007-2011) et Optimisation Combinatoire du LIA (2001-2003), des groupe de travail *LigéRO* (2010-2012) et *Contraintes et RO* (2004-2008), du congrès *Roadef'03* et du workshop *MLS+CP'05*

Enseignements

Animation pédagogique

- 2006-2011 : responsable du Master GIPAD de l'École des Mines de Nantes
- 2007-2008 : responsable de la refonte du Master GIPAD de l'École des Mines de Nantes
- 2004 : membre du comité de pilotage de la refonte des enseignements à l'IUP GMI de l'Université d'Avignon dans le cadre de la mise en place du LMD

Heures d'enseignement dispensées

	établissement	niveau	cours	TD/TP	projet
Optimisation Combinatoire			**177**	**270**	**135**
Programmation linéaire en nombres entiers *	Mines Paris	MS	15	72	
Programmation non-linéaire en nombres entiers *	Mines Paris	MS	6	12	
Études bibliographiques *	Mines Paris	MS			10
Advanced Integer Linear Programming *	FST Nantes	M2	90		
APP Recherche Opérationnelle *	Mines Nantes	M2	40	104	
Programmation par Contraintes *	Mines Nantes	M2		20	
Projet Industriel *	Mines Nantes	M2			110
Méthodes hybrides PPC/RO *	Mines Nantes	M2	5		
Algorithmes de Graphes	Mines Nantes	M1	5		
Séries Génératrices *	Mines Nantes	L3	10		
Graphes et Algorithmes	IUP Avignon	M1	3	40	
Optimisation Combinatoire	FST Avignon	L2	3	22	
Programmation par Contraintes *	IUP Avignon	L2			15
Programmation et algorithmique				**330**	**123**
Programmation et Algorithmique (Java)	Mines Nantes	L2		87	
Programmation Objet (Java)	Mines Nantes	L2		40	108
Structures de données	Mines Nantes	L3		52	
Programmation et Algorithmique (C++)	IUP Avignon	L2		48	15
Programmation et Algorithmique Avancée (C++)	IUP Avignon	M1		50	
Programmation Objet (Java)	IUP Avignon	M1		26	
Programmation et Algorithmique (C)	FST Avignon	L1		27	
Informatique fondamentale et ingénierie				**252**	**85**
Intégration Physique-Informatique *	Mines Nantes	L2		70	
Bases de Données (mySQL)	Mines Nantes	L3		50	
Projets Scientifiques *	Mines Nantes	L3			55
Systèmes d'Exploitation, (Shell, C)	IUP Avignon	L3		26	
Bases de Données (Oracle, PL/SQL)	IUP Avignon	L3		32	30
Technologies Internet (Apache, PHP, mySQL)	IUP Avignon	M2		44	
Initiation à l'Informatique	FST Aix-M II	L1		30	

FST=Faculté des Sciences et Techniques (ou UFR Sciences) de Nantes, Avignon, ou Aix-Marseille II
* *conception/responsabilité du cours*

Publications

Les publications sont téléchargeables en ligne : http://sofdem.github.io/biblio.html
Profil Scholar : http://scholar.google.com/citations?user=aLalTi4AAAAJ

Revues internationales

[1] G. Bonvin, S. Demassey, C. Le Pape, V. Mazauric, N. Maïzi, A. Samperio (2017). *A mathematical programming approach of pump scheduling for load management in branched water networks*, **Applied Energy** in *Clean, Efficient and Affordable Energy for a Sustainable Future*, 185 (2), 1702–1711.

[2] N. Beldiceanu, M. Carlsson, S. Demassey, E. Poder (2011). *New filtering for the* `cumulative` *constraint in the context of non-overlapping rectangles*, **Annals of Operations Research**, 184 (1) : 27–50.

[3] N. Beldiceanu, M. Carlsson, S. Demassey, T. Petit (2007). *Global Constraint Catalog : past, present and future*, **Constraints**, special issue on *global constraints*, 12 (1) : 21–62.

[4] S. Demassey, G. Pesant, L.-M. Rousseau (2006). *A* `cost-regular` *based hybrid column generation approach*, **Constraints**, *article invité, version étendue de [***cpaior05***]*, 11 (4) : 315–333.

[5] S. Demassey, C. Artigues, Ph. Michelon (2005). *Constraint-propagation-based cutting planes : an application to the Resource-Constrained Project Scheduling Problem*, **INFORMS Journal on Computing**, 17 (1) : 52–65.

[6] Ph. Baptiste, S. Demassey (2004). *Tight LP bounds for Resource Constrained Project Scheduling*, **OR Spectrum**, 26 (2) : 251–262.

Conférences internationales avec sélection et actes

[7] F. Hermenier, G. Giuliani, A. Milani, S. Demassey (2017). *Scaling Energy Adaptive Applications for Sustainable Profitability*, in proceedings of the 23rd International European Conference on Parallel and Distributed Computing (**Euro-Par'17**), Santiago de Compostela, Spain. (accepté)

[8] G. Bonvin, A. Samperio, C. Le Pape, V. Mazauric, S. Demassey, N. Maïzi (2015). *A heuristic approach to the water networks pumping scheduling issue*, in proceedings of the 7th International Conference on Applied Energy (**ICAE'15**), Abu Dhabi, UAE. Energy Procedia 75 : 2846-2851. **Taux : 71%**.

[9] G. Chabert, S. Demassey (2012). *The conjunction of* `Interval-Among` *constraints*, in proceedings of the 9th International Conference on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (**CPAIOR'12**), Nantes, France. Lecture Notes in Computer Science 7298 : 113-128. **Taux d'acceptation : 40%**.

[10] F. Hermenier, S. Demassey, X. Lorca (2011). *Bin-Repacking Scheduling in virtualized datacenters*, in proceedings of the 17th International Conference on Principle and Practice of Constraint Programming (**CP'11**), Perugia, Italie. Lecture Notes in Computer Science 6876 : 27–41. Application track. **Taux : 35%**.

[11] A. Merel, X. Gandibleux, S. Demassey (2011). *A collaborative combination between column generation and ant colony optimization for solving set packing problems*, in proceedings of the 9th Metaheuristics International Conference (**MIC'11**), Udine, Italie.

[12] J. Menana, S. Demassey (2009). *Sequencing and counting with the* `multicost-regular` *constraint*, in proceedings of the 6th International Conference on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (**CPAIOR'09**), Pittsburgh, USA. Lecture Notes in Computer Science 5547 : 178–192. **Taux : 47%**.

[13] N. Beldiceanu, M. Carlsson, S. Demassey, T. Petit (2006). *Graph properties based filtering*, in proceedings of the 12th International Conference on Principle and Practice of Constraint Programming (**CP'06**), Nantes, France. Lecture Notes in Computer Science 4204 : 59–74. **Taux : 29%**.

[14] S. Demassey, G. Pesant, L.-M. Rousseau (2005). *Constraint programming based column generation for employee timetabling*, in proceedings of the 2nd International Conference on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (**CPAIOR'05**), Prague. Lecture Notes in Computer Science 3524 : 140–154. **Taux : 26%**.

Édition de livre et chapitres de livres référés

[15] S. Demassey, F. Hermenier, V. Kherbache (2015). Chapitre *Dynamic Packing with Side Constraints for Datacenter Resource Management* du livre *Optimized Packings and Their Applications*, G. Fasano et J.D. Pintér (eds.), Springer Optimization and Its Applications vol. 105.

[16] C. Artigues, S. Demassey, E. Néron (2008). **Resource-Constrained Project Scheduling : models, algorithms, extensions and applications**, ISTE/Wiley.

[17] S. Demassey (2008). Chapitre 3 *Mathematical programming formulations and lower bounds for the RCPSP* du livre *Resource-Constrained Project Scheduling : Models, Algorithms, Extensions and Applications*, C. Artigues, S. Demassey, E. Néron (eds.), ISTE/Wiley.

[18] E. Néron, C. Artigues, Ph. Baptiste, J. Carlier, S. Demassey, Ph. Laborie (2006). Chapitre 7 *Lower bounds computation for RCPSP* du livre *Perspectives in modern project scheduling*, J. Weglarz et J. Józefowska (eds.), Springer, International Series in Operations Research and Management Science 92 : 167–204.

[19] C. Artigues, S. Demassey (2005). Chapitre 5 *Ordonnancement de projet* du livre *Gestion de la production et ressources humaines*, P. Baptiste, V. Giard, A. Hait, F. Soumis (éds.), Presses Internationales Polytechnique, Montréal.

Conférences nationales avec sélection et actes

[20] J. Menana, S. Demassey (2009). *Séquencer et compter avec la contrainte* `multicost-regular`, 5èmes Journées Francophones de Programmation par Contraintes (**JFPC'09**), Orléans, 125–134. *Version française de [***cpaior09***].*

[21] A. Merel, X. Gandibleux, S. Demassey, R. Lusby (2009). *An improved upper bound for the Railway Infrastructure Capacity Problem on the Pierrefitte-Gonesse junction*, actes longs du 10ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'09**), Nancy, 62–76.

[22] N. Beldiceanu, M. Carlsson, S. Demassey, T. Petit (2006). *Filtrage basé sur des propriétés de graphes*, 2èmes Journées Francophones de Programmation par Contraintes (**JFPC'06**), Nîmes, 25–34. *Version française de [***cp06***].*

Manifestations internationales avec actes à diffusion restreinte

[23] S. Demassey, F. Hermenier (2014). *BtrPlace : Flexible VM Management in Data Centers*, Conference on Optimization & Practices in Industry (**PGMO-COPI'14**) Paris-Saclay, France, 3p.

[24] A. Merel, X. Gandibleux, S. Demassey (2011). *Assessing railway infrastructure capacity by solving the saturation problem with an improved column generation algorithm*, IAROR Conference : 4th International Seminar on Railway Operations Modelling and Analysis (**RailRome'11**), Rome, Italie.

[25] A. Merel, X. Gandibleux, S. Demassey (2011). *Towards a realistic evaluation of Railway Infrastructure Capacity*, 9th World Congress on Railway Research (**WCRR'11**), Lille, France.

[26] S. Demassey, C. Artigues, Ph. Baptiste, Ph. Michelon (2004). *Lagrangean relaxation for the RCPSP*, in proceedings of the 9th International Workshop on Project Management and Scheduling (**PMS'04**), Nancy, France, 380–384.

[27] S. Demassey, C. Artigues, Ph. Michelon (2002). *A hybrid constraint propagation-cutting plane algorithm for the RCPSP*, in proceedings of the 4th International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimisation Problems (**CPAIOR'02**), Le Croisic, France, 321–331. *Étude préliminaire à [***joc05***].* Taux : 27/35.

[28] S. Demassey, C. Artigues, Ph. Michelon (2001). *Comparing lower bounds for the RCPSP under a same hybrid constraint-linear programming approach*, in proceedings of the International Workshop on Cooperative Solvers in Constraint Programming (**CoSolv'01**) held in conjunction with the International Conference on Constraint Programming (CP'01), Paphos, Chypre, 109–123. *Étude préliminaire à [***joc05***].*

[29] S. Demassey, C. Artigues, Ph. Michelon (2001). *A new LP based lower bound for the RCPSP*, 5th Workshop on Models and Algorithms for Planning and Scheduling Problems (**MAPSP'01**), Aussois, France. *Étude préliminaire à [***joc05***].*

Manifestations internationales et nationales référées

[30] G. Bonvin, S. Demassey. (2017). *Dimensionnement des réseaux gravitaires de distribution d'eau potable par relaxation convexe et décomposition spatiale*, 18ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'17**), Metz, France.

[31] G. Bonvin, S. Demassey. (2017). *Relaxation convexe pour la planification de pompage dans les réseaux de distribution d'eau potable*, 18ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'17**), Metz, France.

[32] G. Bonvin, S. Demassey (2016). *Convex Relaxation for Water Distribution Systems*, 28th European Conference on Operational Research (**EURO 2016**), Poznan, Poland.

[33] A. Havel, S. Demassey (2016). *Robust Optimisation for a Smart Grid*, 28th European Conference on Operational Research (**EURO 2016**), Poznan, Poland.

[34] G. Bonvin, S. Demassey. (2016). *Relaxation convexe pour la planification du pompage dans un réseau branché de distribution d'eau*, 17ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'16**), Compiègne, France.

[35] G. Bonvin, S. Demassey (2015). *Energy efficiency in water supply systems : variable speed drives vs pumping scheduling*, 27th European Conference on Operational Research (**EURO'15**), Glasgow, GB.

[36] S. Demassey, D. Feillet (2015). *Hybridation de programmation linéaire et de programmation par contraintes pour le problème de déplacement de conteneurs*, 16ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'15**), Marseille, France.

[37] X. Libeaut, S. Demassey, J. E. Mendoza, É. Pinson (2012). *Problème de multiflots dynamiques avec contraintes de synchronisation de ressources*, 13ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'12**), Angers, France.

[38] A. Merel, X. Gandibleux, S. Demassey (2012). *Vers la prise en compte d'un critère d'équité dans l'évaluation de la capacité d'infrastructures ferroviaires*, 13ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'12**), Angers, France.

[39] J. Menana, S. Demassey (2010). *Weighted automata, constraint programming, and large neighborhood search*, Nurse Rostering Competition at the 8th International Conference for the Practice and Theory of Automated Timetabling (**PATAT'10**), Belfast, Ireland.

[40] A. Merel, S. Demassey, X. Gandibleux (2010). *Un algorithme de génération de colonnes pour le problème de capacité d'infrastructure ferroviaire*, 11ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'10**), Toulouse, France.

[41] J. Menana, S. Demassey, N. Jussien (2009). *Relaxation lagrangienne pour le filtrage d'une contrainte-automate à coûts multiples*, 10ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'09**), Nancy, France.

[42] S. Demassey, G. Pesant, L.-M. Rousseau (2005). *Constraint programming based column generation for employee timetabling*, Optimization Days (**JOPT'05**), Montréal, Canada.

[43] T. Garaix, C. Artigues, S. Demassey (2005). *Bornes basées sur les ensembles interdits pour le problème d'ordonnancement de projet à moyens limités*, 6ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'05**), Tours, France.

[44] S. Demassey (2004). *Resolution search : intelligent backtracking for the RCPSP*, 17th European Conference on Combinatorial Optimization (**ECCO'04**), Beyrouth, Liban.

[45] S. Demassey, S. Gueye, Ph. Michelon, C. Artigues (2003). *Application de resolution search au RCPSP*, École d'Automne de Recherche Opérationnelle (**EARO'03**), Tours, France.

[46] C. Artigues, S. Demassey, Ph. Michelon (2002). *A hybrid constraint propagation-cutting plane procedure for the RCPSP*, International Conference on Operations Research (**OR'02**), Klagenfurt, Allemagne.

[47] S. Demassey, C. Artigues, Ph. Michelon (2002). *Bornes inférieures pour le RCPSP : une approche hybride PPC/PL*, 4ème congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision (**ROADEF'02**), Paris, France.

[48] Ph. Michelon, S. Demassey, C. Artigues (2002). *Bornes inférieures pour le RCPSP : une approche hybride PPC/PL*, 3ème Conférence Internationale de Recherche Opérationnelle (**CIRO'02**), Marrakech, Maroc.

[49] S. Demassey, C. Artigues, Ph. Michelon (2001). *Borne inférieure pour le problème d'ordonnancement de projet à contraintes de ressources*, 3èmes Journées Francophones de Recherche Opérationnelle (**Francoro'01**), Québec, Canada.

[50] C. Artigues, S. Demassey, Ph. Michelon (2000). *A linear programming based approach for resource constrained project scheduling*, International Symposium on Mathematical Programming (**ISMP'00**), Atlanta, USA.

Séminaires invités

[51] S. Demassey (2012). *Constraints and Automata*, I3S – Université de Nice. Séminaire du pôle MDSC.

[52] S. Demassey (2012). *Minimum Set Covering*, I3S – Université de Nice. Séminaire de l'équipe CEP.

[53] S. Demassey (2006). *Experiments with Resolution Search*, NATO Workshop on hybrid methods and branching rules in combinatorial optimization (**Hybrid'06**), Montréal, Canada.

[54] S. Demassey (2005). *Resolution Search and intelligent backtrackings*, Concordia University, Montréal, Canada. Séminaire invité par V. Chvátal.

[55] S. Demassey (2005). *Méthodes hybrides PL/PPC pour la résolution exacte des problèmes combinatoires difficiles*, LINA – École des Mines de Nantes. Séminaire invité par N. Beldiceanu.

[56] S. Demassey (2005). *Resolution search et backtrackings intelligents*, CRT – Université de Montréal, Canada. Séminaire invité par L.-M. Rousseau.

[57] S. Demassey (2003). *Resolution search et backtrackings intelligents*, 1ère journée du groupe de travail Programmation Mathématique (GdR ALP), 5 décembre 2003, Paris.

Mémoires

[58] S. Demassey (2003). *Méthodes hybrides de programmation par contraintes et de programmation linéaire pour le problème d'ordonnancement de projet à contraintes de ressources*, thèse de doctorat, Université d'Avignon, soutenue le 18 décembre 2003.

[59] S. Demassey (2000). *Borne inférieure pour l'ordonnancement de projet à moyens limités*, mémoire de DEA Informatique - Optimisation Combinatoire, Université d'Aix-Marseille II.

[60] S. Demassey (1998). *Élasticité dans les anneaux de Dedekind*, mémoire de DEA Mathématiques Fondamentales - Algèbre Commutative, Université d'Aix-Marseille I.

Bibliographie

[Ach+16] Tobias Achterberg, Robert E Bixby, Zonghao Gu, Edward Rothberg et Dieter Weninger. *Presolve reductions in mixed integer programming*. Rapp. tech. Technical Report 16-44, ZIB, Takustr. 7, 14195 Berlin, 2016 (cf. p. 67).

[Ach09] Tobias Achterberg. « SCIP : solving constraint integer programs ». In : *Mathematical Programming Computation* 1.1 (2009), p. 1–41 (cf. p. 55).

[AHY04] Ionuţ Aron, John N Hooker et Tallys H Yunes. « SIMPL : A system for integrating optimization techniques ». In : *Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2004, p. 21–36 (cf. p. 55).

[And+07] Henrik Reif Andersen, Tarik Hadzic, John N Hooker et Peter Tiedemann. « A constraint store based on multivalued decision diagrams ». In : *Principles and Practice of Constraint Programming*. Springer. 2007, p. 118–132 (cf. p. 54).

[Bar+98] Cynthia Barnhart, Ellis L Johnson, George L Nemhauser, Martin WP Savelsbergh et Pamela H Vance. « Branch-and-price : Column generation for solving huge integer programs ». In : *Operations research* 46.3 (1998), p. 316–329 (cf. p. 9).

[BC01] Nicolas Beldiceanu et Mats Carlsson. « Sweep as a generic pruning technique applied to the non-overlapping rectangles constraints ». In : *Principles and Practice of Constraint Programming*. Sous la dir. de T. Walsh. T. 2239. LNCS. Springer-Verlag, 2001, p. 377–391 (cf. p. 53).

[BC16] David Bergman et Andre A Cire. « Decomposition Based on Decision Diagrams ». In : *Integration of AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer. 2016, p. 45–54 (cf. p. 54).

[BCH15a] David Bergman, Andre A Cire et Willem-Jan van Hoeve. « Improved Constraint Propagation via Lagrangian Decomposition ». In : *Principles and Practice of Constraint Programming*. Springer. 2015, p. 30–38 (cf. p. 54, 65).

[BCH15b] David Bergman, Andre A Cire et Willem-Jan van Hoeve. « Lagrangian bounds from decision diagrams ». In : *Constraints* 20.3 (2015), p. 346–361 (cf. p. 35).

[BCP07] Christian Bessiere, Remi Coletta et Thierry Petit. « Learning Implied Global Constraints. » In : *International Joint Conference on Artificial Intelligence*. 2007, p. 44–49 (cf. p. 54).

[BCR] Nicolas Beldiceanu, Mats Carlsson et Jean-Xavier Rampon. *Global constraint catalog*. http ://sofdem.github.io/gccat/ (cf. p. 10, 47, 52, 56).

[Bel+05] Nicolas Beldiceanu, Mats Carlsson, Romuald Debruyne et Thierry Petit. « Reformulation of Global Constraints Based on Constraint Checkers ». In : *Constraints* 10.3 (2005) (cf. p. 55).

[Bel+06] Nicolas Beldiceanu, Mats Carlsson, Sophie Demassey et Thierry Petit. « Graph properties based filtering ». In : *Principles and Practice of Constraint Programming*. Springer Berlin Heidelberg. 2006, p. 59–74 (cf. p. 57).

[Bel03] Nicolas Beldiceanu. « Contraintes globales : aspects algorithmiques et déclaratifs ». Habilitation à diriger des recherches. 2003 (cf. p. 10, 55).

[Ben+12] Pascal Benchimol, Willem-Jan van Hoeve, Jean-Charles Régin, Louis-Martin Rousseau et Michel Rueher. « Improved filtering for weighted circuit constraints ». In : *Constraints* 17.3 (2012), p. 205–233 (cf. p. 61, 63).

[Ben62] Jacques F Benders. « Partitioning procedures for solving mixed-variables programming problems ». In : *Numerische mathematik* 4.1 (1962), p. 238–252 (cf. p. 5).

[Bes+05a] C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan et T. Walsh. « Among, Common and Disjoint Constraints ». In : *CSCLP : Recent Advances in Constraints*. T. 3978. Lecture Notes in Computer Science. 2005, p. 29–43 (cf. p. 25).

[Bes+05b] C. Bessière, E. Hebrard, B. Hnich, Z. Kiziltan et T. Walsh. « The Range and Roots Constraints : Specifying Counting and Occurrence Problems ». In : *International Joint Conference on Artificial Intelligence*. 2005, p. 60–65 (cf. p. 25).

[BHH11] David Bergman, Willem-Jan van Hoeve et John N Hooker. « Manipulating MDD relaxations for combinatorial optimization ». In : *Integration of AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer. 2011, p. 20–35 (cf. p. 54).

[Bol+16] Natashia Boland, Matteo Fischetti, Michele Monaci et Martin Savelsbergh. « Proximity Benders : a decomposition heuristic for stochastic programs ». In : *Journal of Heuristics* 22.2 (2016), p. 181–198 (cf. p. 5).

[Bon+17] Gratien Bonvin, Sophie Demassey, Claude Le Pape, Nadia Maïzi, Vincent Mazauric et Alfredo Samperio. « A convex mathematical program for pump scheduling in a class of branched water networks ». In : *Applied Energy* 185 (2017), p. 1702–1711 (cf. p. 71).

[BPR96] Matthew H Bassett, Joseph F Pekny et Gintaras V Reklaitis. « Decomposition techniques for the solution of large-scale scheduling problems ». In : *AIChE Journal* 42.12 (1996), p. 3373–3387 (cf. p. 5).

[BR97] Christian Bessiere et Jean-Charles Régin. « Arc consistency for general constraint networks : preliminary results ». In : *International Joint Conference on Artificial Intelligence*. 1997 (cf. p. 55).

[Bra+07] S. Brand, N. Narodytska, C.G. Quimper, P. Stuckey et T. Walsh. « Encodings of the sequence constraint ». In : *Principles and Practice of Constraint Programming*. T. 4741. Lecture Notes in Computer Science. 2007, p. 210–224 (cf. p. 25).

[bri] brics. *dk.brics.automaton*. http://www.brics.dk/automaton/ (cf. p. 40).

[btr] btrPlace. *BtrPlace : An Open-Source flexible virtual machine scheduler*. http://www.btrplace.org/ (cf. p. 14).

[Bur+09] Edmund K Burke, Mathew R Hyde, Graham Kendall, Gabriela Ochoa, Ender Ozcan et John R Woodward. « Exploring hyper-heuristic methodologies with genetic programming ». In : *Computational intelligence*. Springer, 2009, p. 177–201 (cf. p. 4).

[Cap+98] Alberto Caprara, Paolo Toth, Daniele Vigo et Matteo Fischetti. « Modeling and solving the crew rostering problem ». In : *Operations research* 46.6 (1998), p. 820–830 (cf. p. 28).

[CD12] Gilles Chabert et Sophie Demassey. « The conjunction of Interval-Among constraints ». In : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*. T. 7298. LNCS. 2012, p. 113–128 (cf. p. 21, 25).

[CGR11] Marie-Claude Côté, Bernard Gendron et Louis-Martin Rousseau. « Grammar-based integer programming models for multiactivity shift scheduling ». In : *Management Science* 57.1 (2011), p. 151–163 (cf. p. 36, 59).

[CGR13] Marie-Claude Côté, Bernard Gendron et Louis-Martin Rousseau. « Grammar-based column generation for personalized multi-activity shift scheduling ». In : *INFORMS Journal on Computing* 25.3 (2013), p. 461–474 (cf. p. 36).

[CGS16] Geoffrey Chu, Graeme Gange et Peter J Stuckey. « Lagrangian Decomposition via Sub-problem Search ». In : *Integration of AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer. 2016, p. 65–80 (cf. p. 54).

[CL97] Yves Caseau et Francois Laburthe. « Solving Small TSPs with Constraints. » In : *ICLP*. T. 97. 1997, p. 104 (cf. p. 64).

[Cla+05] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt et Andrew Warfield. « Live Migration of Virtual Machines ». In : *Networked Systems Design & Implementation - Volume 2*. NSDI'05. Berkeley, CA, USA : USENIX Association, 2005, p. 273–286 (cf. p. 16).

[CO10] Hadrien Cambazard et Barry O'Sullivan. « Propagating the bin packing constraint using linear programming ». In : *Principles and Practice of Constraint Programming*. Springer. 2010, p. 129–136 (cf. p. 61, 63).

[CP12] Hadrien Cambazard et Bernard Penz. « A constraint programming approach for the traveling purchaser problem ». In : *Principles and Practice of Constraint Programming*. Springer. 2012, p. 735–749 (cf. p. 64).

[CP90] Jacques Carlier et Eric Pinson. « a practical use of Jackson's preemptive schedule for solving the job shop problem. » In : *Annals of Operations Research* 26 (1990) (cf. p. 53).

[Dan08] Emilie Danna. *Performance variability in mixed integer programming. Presentation slides from MIP workshop in New York City*. 2008 (cf. p. 64).

[Dan54] George B Dantzig. « Letter to the Editor : A Comment on Edie's "Traffic Delays at Toll Booths" ». In : *Journal of the Operations Research Society of America* 2.3 (1954), p. 339–341 (cf. p. 26).

[DDS08] Gregory J Duck, Leslie De Koninck et Peter J Stuckey. « Cadmium : An implementation of ACD term rewriting ». In : *International Conference on Logic Programming*. Springer. 2008, p. 531–545 (cf. p. 55).

[DDS92] Martin Desrochers, Jacques Desrosiers et Marius Solomon. « A new optimization algorithm for the vehicle routing problem with time windows ». In : *Operations research* 40.2 (1992), p. 342–354 (cf. p. 6).

[Dem03] Sophie Demassey. « Méthodes hybrides de programmation par contraintes et programmation linéaire pour le probleme d'ordonnancement de projeta contraintes de ressources ». Thèse de doctorat. Université d'Avignon et des Pays de Vaucluse, 2003 (cf. p. 1).

[Des+95] Jacques Desrosiers, Yvan Dumas, Marius M Solomon et François Soumis. « Time constrained routing and scheduling ». In : *Handbooks in operations research and management science* 8 (1995), p. 35–139 (cf. p. 28).

[DF15] Sophie Demassey et Dominique Feillet. « Hybridation de programmation linéaire et de programmation par contraintes pour le problème de déplacement de conteneurs ». In : *ROADEF'15-16ème congrès annuel de la Société française de Recherche Opérationnelle et d'Aide à la Décision*. 2015 (cf. p. 63–66).

[DHK15] Sophie Demassey, Fabien Hermenier et Vincent Kherbache. « Optimized Packings and Their Applications ». In : Optimization and Its Applications 105. Springer, 2015. Chap. Dynamic Packing with Side Constraints for Datacenter Resource Management (cf. p. 14, 19).

[DMP91] R. Dechter, I. Meiri et J. Pearl. « Temporal constraint networks ». In : *Artificial Intelligence* 49.1-3 (1991), p. 61–95 (cf. p. 22).

[DPR05] Sophie Demassey, Gilles Pesant et Louis-Martin Rousseau. « Constraint programming based column generation for employee timetabling ». In : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*. Sous la dir. de Roman Barták et Michela Milano. T. 3524. LNCS. Springer-Verlag, 2005, p. 140–154 (cf. p. 26, 29, 39).

[DPR06] Sophie Demassey, Gilles Pesant et Louis-Martin Rousseau. « A cost-regular based hybrid column generation approach ». In : *Constraints* 11.4 (2006), p. 315–333 (cf. p. 26, 36).

[Du +99] Olivier Du Merle, Daniel Villeneuve, Jacques Desrosiers et Pierre Hansen. « Stabilized column generation ». In : *Discrete Mathematics* 194.1-3 (1999), p. 229–237 (cf. p. 6).

[DW60] George B Dantzig et Philip Wolfe. « Decomposition principle for linear programs ». In : *Operations research* 8.1 (1960), p. 101–111 (cf. p. 8).

[EF10] Duncan Epping et Denneman Frank. *VMware vSphere 4.1 HA and DRS technical deepdive*. 2010 (cf. p. 17).

[EW01] Andrew Eremin et Mark Wallace. « Hybrid Benders decomposition algorithms in constraint logic programming ». In : *Principles and Practice of Constraint Programming*. Springer. 2001, p. 1–15 (cf. p. 69).

[F+14] Daniel Fontaine, Pascal Van Hentenryck et al. « Constraint-based lagrangian relaxation ». In : *Principles and Practice of Constraint Programming*. Springer. 2014, p. 324–339 (cf. p. 54).

[FF58] Lester Randolph Ford Jr et Delbert R Fulkerson. « A suggested computation for maximal multi-commodity network flows ». In : *Management Science* 5.1 (1958), p. 97–101 (cf. p. 5).

[FL82] PM França et HPL Luna. « Solving stochastic transportation-location problems by generalized Benders decomposition ». In : *Transportation Science* 16.2 (1982), p. 113–126 (cf. p. 6).

[FLM02] Filippo Focacci, Andrea Lodi et Michela Milano. « Optimization-oriented global constraints ». In : *Constraints* 7.3 (2002), p. 351–365 (cf. p. 63).

[FLM99] F. Focacci, A. Lodi et M. Milano. « Cost-Based Domain Filtering ». In : *Principles and Practice of Constraint Programming –CP'99* (1999), p. 189–203 (cf. p. 35, 49, 61, 65).

[FLS16] Matteo Fischetti, Ivana Ljubić et Markus Sinnl. « Redesigning Benders Decomposition for Large-Scale Facility Location ». In : *Management Science* (2016) (cf. p. 5).

[FM12] Daniel Fontaine et Laurent Michel. « A high level language for solver independent model manipulation and generation of hybrid solvers ». In : *Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2012, p. 180–194 (cf. p. 55).

[Fri+08] Alan M Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martínez-Hernández et Ian Miguel. « Essence : A constraint language for specifying combinatorial problems ». In : *Constraints* 13.3 (2008), p. 268–306 (cf. p. 55).

[Gar+76] Michael R Garey, Ronald L Graham, David S Johnson et Andrew Chi-Chih Yao. « Resource constrained scheduling as generalized bin packing ». In : *Journal of Combinatorial Theory, Series A* 21.3 (1976), p. 257–298 (cf. p. 14).

[Geo74] Arthur M Geoffrion. « Lagrangean relaxation for integer programming ». In : *Approaches to integer programming*. Springer, 1974, p. 82–114 (cf. p. 5, 6).

[GL10] Gerald Gamrath et Marco E Lübbecke. « Experiments with a generic Dantzig-Wolfe decomposition for integer programs ». In : *International Symposium on Experimental Algorithms*. Springer. 2010, p. 239–252 (cf. p. 5).

[GSW05] Thorsten Gellermann, Meinolf Sellmann et Robert Wright. « Shorter path constraints for the resource constrained shortest path problem ». In : *Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2005, p. 201–216 (cf. p. 61, 63).

[Had+08] Tarik Hadzic, John N Hooker, Barry O'Sullivan et Peter Tiedemann. « Approximate compilation of constraints into multivalued decision diagrams ». In : *Principles and Practice of Constraint Programming*. Springer. 2008, p. 448–462 (cf. p. 47).

[HDL11a] Fabien Hermenier, Sophie Demassey et Xavier Lorca. « Bin repacking scheduling in virtualized datacenters ». In : *Principles and Practice of Constraint Programming*. Springer Berlin Heidelberg. 2011, p. 27–41 (cf. p. 14, 17).

[HDL11b] Fabien Hermenier, Sophie Demassey et Xavier Lorca. *Bin Repacking Scheduling in Virtualized Datacenters - Back to Work*. Rapport technique : addendum. Université de Nice Sophia Antipolis, 2011 (cf. p. 14, 17).

[HMU01] John E Hopcroft, Rajeev Motwani et Jeffrey D Ullman. *Introduction to automata theory, languages, and computation*. Addison-Wesley, 2001 (cf. p. 29, 47, 59).

[HO03] John N Hooker et Greger Ottosson. « Logic-based Benders decomposition ». In : *Mathematical Programming* 96.1 (2003), p. 33–60 (cf. p. 69).

[HQR15] Minh Hoàng Hà, Claude-Guy Quimper et Louis-Martin Rousseau. « General bounding mechanism for constraint programs ». In : *Principles and Practice of Constraint Programming*. Springer. 2015, p. 158–172 (cf. p. 54, 65).

[HVH10] Samid Hoda, Willem-Jan Van Hoeve et John N Hooker. « A systematic approach to MDD-based constraint programming ». In : *Principles and Practice of Constraint Programming*. Springer. 2010, p. 266–280 (cf. p. 54).

[HY02] John N Hooker et Hong Yan. « A relaxation of the cumulative constraint ». In : *Principles and Practice of Constraint Programming*. Springer. 2002, p. 686–691 (cf. p. 63).

[IG98] Ramaswamy R Iyer et Ignacio E Grossmann. « A bilevel decomposition algorithm for long-range planning of process networks ». In : *Industrial & Engineering Chemistry Research* 37.2 (1998), p. 474–481 (cf. p. 5).

[Jun+99] Ulrich Junker, Stefan E. Karisch, Niklas Kohl, Bo Vaaben, Torsten Fahle et Meinolf Sellmann. « A Framework for Constraint Programming Based Column Generation ». In : *Principles and Practice of Constraint Programming*. 1999, p. 261–274 (cf. p. 69).

[KBN05] Mohand Ou Idir Khemmoudj, Hachemi Bennaceur et Anass Nagih. « Combining arc-consistency and dual Lagrangean relaxation for filtering CSPs ». In : *Integration of Artificial Intelligence (AI) and Operations Research (OR) Techniques in Constraint Programming*. Springer. 2005, p. 258–272 (cf. p. 64).

[Kel60] James E Kelley Jr. « The cutting-plane method for solving convex programs ». In : *Journal of the society for Industrial and Applied Mathematics* 8.4 (1960), p. 703–712 (cf. p. 7).

[Kra+16] Vincent Krakowski, Edi Assoumou, Vincent Mazauric et Nadia Maïzi. « Feasible path toward 40–100% renewable energy shares for power supply in France by 2050 : A prospective analysis ». In : *Applied Energy* 171 (juin 2016), p. 501–522 (cf. p. 73).

[KS08] Serdar Kadioglu et Meinolf Sellmann. « Efficient Context-Free Grammar Constraints ». In : *AAAI*. 2008, p. 310–316 (cf. p. 36).

[LD05] Marco E Lübbecke et Jacques Desrosiers. « Selected topics in column generation ». In : *Operations Research* 53.6 (2005), p. 1007–1023 (cf. p. 9).

[Lod10] Andrea Lodi. « Mixed integer programming computation ». In : *50 Years of Integer Programming 1958-2008* (2010), p. 619–645 (cf. p. 64).

[LP04] Gilbert Laporte et Gilles Pesant. « A general multi-shift scheduling system ». In : *Journal of the Operational Research Society* 55.11 (2004), p. 1208–1217 (cf. p. 37).

[LT13] Andrea Lodi et Andrea Tramontani. « Performance variability in mixed-integer programming ». In : *Theory Driven by Influential Applications*. INFORMS, 2013, p. 1–12 (cf. p. 64).

[MA14] Nadia Maïzi et Edi Assoumou. « Future prospects for nuclear power in France ». In : *Applied Energy* 136 (2014), p. 849–859 (cf. p. 73).

[Mah+08] M.J. Maher, N. Narodytska, C-G. Quimper et T. Walsh. « Flow-Based Propagators for the SEQUENCE and Related Global Constraints ». In : *Principles and Practice of Constraint Programming*. T. 5202. Lecture Notes in Computer Science. 2008, p. 159–174 (cf. p. 25).

[MD09] Julien Menana et Sophie Demassey. « Sequencing and counting with the multicost-regular constraint ». In : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimisation Problems*. Springer Berlin Heidelberg. 2009, p. 178–192 (cf. p. 36, 39, 64).

[Men11] Julien Menana. « Automates et programmation par contraintes pour la planification de personnel ». Thèse de doctorat. École des Mines de Nantes, 2011 (cf. p. 36, 40, 47, 48).

[Pes+09] Gilles Pesant, Claude-Guy Quimper, Louis-Martin Rousseau et Meinolf Sellmann. « The polytope of context-free grammar constraints ». In : *Integration of AI and OR Techniques in Constriant Programming for Combinatorial Optimization Problems*. Springer. 2009, p. 223–232 (cf. p. 59).

[Pes01] Gilles Pesant. « A filtering algorithm for the stretch constraint ». In : *Principles and Practice of Constraint Programming*. Springer. 2001, p. 183–195 (cf. p. 30).

[Pes04] Gilles Pesant. « A Regular Language Membership Constraint for Finite Sequences of Variables ». In : *Principles and Practice of Constraint Programming*. 2004, p. 482–495 (cf. p. 28, 29).

[Pin88] Eric Pinson. « Le problème de job-shop ». Thèse de doctorat. Paris 6, 1988 (cf. p. 53).

[PR15] Guillaume Perez et Jean-Charles Régin. « Efficient Operations On MDDs For Building Constraint Programming Models ». In : *International Joint Conference on Artificial Intelligence*. Buenos Aires, Argentina, juil. 2015 (cf. p. 47, 54).

[PRB01] Thierry Petit, Jean-Charles Régin et Christian Bessiere. « Specific filtering algorithms for over-constrained problems ». In : *Principles and Practice of Constraint Programming*. Springer. 2001, p. 451–463 (cf. p. 61).

[Puc+11] Jakob Puchinger, Peter J Stuckey, Mark G Wallace et Sebastian Brand. « Dantzig-Wolfe decomposition and branch-and-price solving in G12 ». In : *Constraints* 16.1 (2011), p. 77–99 (cf. p. 5).

[QR10] Claude-Guy Quimper et Louis-Martin Rousseau. « A large neighbourhood search approach to the multi-activity shift scheduling problem ». In : *Journal of Heuristics* 16.3 (2010), p. 373–392 (cf. p. 36, 58).

[QW06] Claude-Guy Quimper et Toby Walsh. « Global grammar constraints ». In : *Principles and Practice of Constraint Programming*. Springer. 2006, p. 751–755 (cf. p. 36, 58).

[QW07] Claude-Guy Quimper et Toby Walsh. « Decomposing global grammar constraints ». In : *Principles and Practice of Constraint Programming*. Springer. 2007, p. 590–604 (cf. p. 36, 58).

[Refo0] Philippe Refalo. « Linear formulation of constraint programming models and hybrid solvers ». In : *Principles and Practice of Constraint Programming*. Springer. 2000, p. 369–383 (cf. p. 63).

[Rég04] Jean-Charles Régin. « Modélisation et Contraintes globales en programmation par contraintes ». Habilitation à diriger des recherches. Université Nice Sophia Antipolis, 2004 (cf. p. 10).

[Rég05] J-C. Régin. « Combination of Among and Cardinality Constraints ». In : *Integration of AI and OR Techniques in Constraint Programming*. T. 3524. Lecture Notes in Computer Science. 2005, p. 288–303 (cf. p. 21, 25).

[Rég11] Jean-Charles Régin. « Global constraints : A survey ». In : *Hybrid optimization*. Springer, 2011, p. 63–134 (cf. p. 10, 25, 51, 55).

[Rég94] Jean-Charles Régin. « A filtering algorithm for constraints of difference in CSPs ». In : *AAAI '94 : national conference on Artificial intelligence (vol. 1)*. Seattle, Washington, United States : American Association for Artificial Intelligence, 1994, p. 362–367 (cf. p. 53).

[Rég96] J-C. Régin. « Generalized Arc Consistency for Global Cardinality Constraint ». In : *13th conference on Artificial intelligence*. AAAI'96. 1996, p. 209–215 (cf. p. 25, 53).

[Rég99] Jean-Charles Régin. « Arc consistency for global cardinality constraints with costs ». In : *Principles and Practice of Constraint Programming*. Springer. 1999, p. 390–404 (cf. p. 61, 63).

[RG05] Ted K Ralphs et Menal Güzelsoy. « The SYMPHONY callable library for mixed integer programming ». In : *The next wave in computing, optimization, and decision technologies*. Springer, 2005, p. 61–76 (cf. p. 5).

[RG10] Ted K Ralphs et Matthew V Galati. « Decomposition methods for integer programming ». In : *Wiley Encyclopedia of Operations Research and Management Science* (2010) (cf. p. 6).

[RL01] TK Ralphs et L Ladányi. *COIN/BCP user's manual*. 2001 (cf. p. 5).

[RP97] Jean-Charles Régin et Jean-François Puget. « A filtering algorithm for global sequencing constraints ». In : *Principles and Practice of Constraint Programming* (1997), p. 32–46 (cf. p. 30).

[Sch+13] Andreas Schutt, Thibaut Feydy, Peter J Stuckey et Mark G Wallace. « Solving RCPSP/max by lazy clause generation ». In : *Journal of Scheduling* 16.3 (2013), p. 273–289 (cf. p. 64, 66).

[Sel03] Meinolf Sellmann. « Cost-based filtering for shorter path constraints ». In : *Principles and Practice of Constraint Programming*. Springer. 2003, p. 694–708 (cf. p. 64).

[Sel04] Meinolf Sellmann. « Theoretical foundations of CP-based lagrangian relaxation ». In : *Principles and Practice of Constraint Programming*. Springer. 2004, p. 634–647 (cf. p. 38, 64, 69).

[Sel06] Meinolf Sellmann. « The theory of grammar constraints ». In : *Principles and Practice of Constraint Programming*. Springer. 2006, p. 530–544 (cf. p. 36, 58).

[Sha04] Paul Shaw. « A Constraint for Bin Packing ». In : *Principles and Practice of Constraint Programming*. Sous la dir. de Mark Wallace. T. 3258. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2004, p. 648–662 (cf. p. 15).

[Sha98] Paul Shaw. « Using constraint programming and local search methods to solve vehicle routing problems ». In : *Principles and Practice of Constraint Programming*. Springer. 1998, p. 417–431 (cf. p. 49, 68).

[Sof] OpenStack Cloud Software. *OpenStack Cloud Software*. `http://www.openstack.org/` (cf. p. 17).

[ST09] Christian Schulte et Guido Tack. « Weakly monotonic propagators ». In : *Principles and Practice of Constraint Programming*. Springer. 2009, p. 723–730 (cf. p. 64).

[Tho01] Erlendur S Thorsteinsson. « Branch-and-check : A hybrid framework integrating mixed integer programming and constraint logic programming ». In : *Principles and Practice of Constraint Programming*. Springer. 2001, p. 16–30 (cf. p. 69).

[Tri01] M. Trick. « A dynamic programming approach for consistency and propagation for knapsack constraints ». In : *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*. 2001 (cf. p. 30, 61, 62).

[Van83] Tony J Van Roy. « Cross decomposition for mixed integer programming ». In : *Mathematical programming* 25.1 (1983), p. 46–63 (cf. p. 5).

[Wol98] Laurence A Wolsey. *Integer programming*. T. 42. Wiley New York, 1998 (cf. p. 38, 62).

[YMD05] Tallys H Yunes, Arnaldo V Moura et Cid C De Souza. « Hybrid column generation approaches for urban transit crew management problems ». In : *Transportation Science* 39.2 (2005), p. 273–288 (cf. p. 28).