TECHNOSUP

Les FILIÈRES TECHNOLOGIQUES des ENSEIGNEMENTS SUPÉRIEURS

INFORMATIQUE

Programmation par contraintes

Démarches de modélisation pour l'optimisation



Éric BOURREAU
Philippe LACOMME
Matthieu GONDRAN
Marina VINOT

ellipses

TECHNOSUP

Les FILIÈRES TECHNOLOGIQUES des ENSEIGNEMENTS SUPÉRIEURS

INFORMATIQUE

Programmation par contraintes

Démarches de modélisation pour l'optimisation

Éric BOURREAU Matthieu GONDRAN
Philippe LACOMME Marina VINOT

Enseignants-chercheurs au LIMOS Clermont-Ferrand
et au LIRMM Montpellier

ellipses

Avec l'un des co-auteurs, dans la même collection :

- De la programmation linéaire à la programmation par contraintes (B) 336p.
  E. BOURREAU, M. GONDRAN, Ph. LACOMME, M. VINOT
- Programmation linéaire avancée (C) 312 p. G. FLEURY, Ph. LACOMME
- Nouvelles technologies d'accès aux données (C) 384 p.
  Ph. LACOMME, R. PHAN, L. REN, N.TCHERNEV
- Bases de données NoSQL et Big Data (C) 336 p . Ph . LACOMME, S. ARIDHI, R. PHAN
- Les web services, Java, PHP, API JavaScript, Androïde SDK 384 p.
  J. FONTANEL, P. LACOMME, L. REN
- Téléphones et montres connectés, Outils de développement (C) 336 p.
  Ph . LACOMME, R. PHAN

Dans la même collection :

- Programmer en langage C, avec exercices corrigés (B) 240 p. J. LE MAITRE
- Belle programmation et langage C (C) 192 p. Y. NOYELLE
- Programmation avec le langage Python (C) 336 p X. DUPRE
- Exercices de programmation fonctionnelle en OCaml (A) 264 p.
  F. CARRIER, P. LAFOURCADE, L. MONIER

ISBN 9782340-092198
©Ellipses Édition Marketing S.A., 2020
32, rue Bargue 75740 Paris cedex 15



Le Code de la propriété intellectuelle n'autorisant, aux termes de l'article L. 122-5.2° et 3°a), d'une part, que les « copies ou reproductions strictement réservées à l'usage privé du copiste et non destinées à une utilisation collective », et d'autre part, que les analyses et les courtes citations dans un but d'exemple et d'illustration, « toute représentation ou reproduction intégrale ou partielle faite sans le consentement de l'auteur ou de ses ayants droit ou ayants cause est illicite » (art. L. 122-4).
Cette représentation ou reproduction, par quelque procédé que ce soit constituerait une contrefaçon sanctionnée par les articles L. 335-2 et suivants du Code de la propriété intellectuelle.

www.editions-ellipses.fr

PREFACE

C'est un grand honneur que m'ont fait les auteurs en me sollicitant pour préfacer leur ouvrage. Je souhaite commencer par les en remercier très chaleureusement.

Ce livre se concentre sur les problématiques de modélisation de problèmes d'optimisation combinatoire en s'appuyant sur la Programmation Par Contraintes. Il présente, pour divers problèmes d'optimisation, les conséquences de choix de modélisation ou de stratégies d'exploration sur l'efficacité d'un modèle de Programmation Par Contraintes. Il met également en avant l'intérêt d'hybrider Programmation Par Contraintes et méthodes de recherche locale afin de passer à l'échelle pour résoudre des problèmes combinatoires.

Les auteurs privilégient une démarche par l'exemple et s'appuient sur plusieurs problèmes dont les spécificités et les modèles mathématiques sont explicités en détail. Pour chaque problème, différents modèles de Programmation Par Contraintes et plusieurs stratégies de résolution sont proposés et analysés, notamment au vu de résultats expérimentaux. Au travers de ces exemples riches, les auteurs montrent également comment exploiter les solveurs de Programmation Par Contraintes (et en particulier Choco) pour mettre en place des méthodes de recherche locale.

Le premier chapitre introduit différents concepts en Programmation Par Contraintes ainsi que divers principes d'exploration d'un arbre de recherche tout en s'appuyant sur le solveur Choco.

Dans le second chapitre, les auteurs se concentrent sur le problème « académique » de tournées de véhicules pour lequel ils présentent de manière progressive plusieurs modèles de Programmation Par Contraintes. Différentes stratégies d'exploration de l'espace de recherche sont étudiées que ce soit par suppression de solutions équivalentes par symétrie ou par des techniques d'instanciation de variables.

Le chapitre trois étend le problème de tournées de véhicules en y ajoutant des contraintes de gestion de stocks au niveau des clients et producteurs et un contexte multi-périodes. Après un positionnement de ce problème dans la littérature, un enrichissement des modèles du chapitre précédent est exposé afin de prendre en compte cette nouvelle variété de contraintes. Les auteurs montrent comment utiliser la Programmation Par Contraintes pour réaliser une construction incrémentale (par période) de solutions dans un schéma de résolution de type glouton.

Le quatrième chapitre s'intéresse à un problème d'ordonnancement (de type Job-Shop), il montre comment la Programmation Par Contraintes peut être utilisée pour explorer le voisinage de solutions. Ce processus de recherche locale est ensuite intégré au sein d'une métaheuristique à population.

Le cinquième chapitre considère des problèmes de tournées avec contraintes de synchronisation. Différentes modélisations sont proposées, en particulier une approche par décomposition du problème ainsi qu'un processus de recherche locale s'appuyant sur la Programmation Par Contraintes.

Le choix des exemples, alternant des problèmes académiques, comme les tournées de véhicules ou l'ordonnancement d'ateliers, et des problèmes étendus, comme les tournées de véhicules avec contraintes de gestion de stock ou contraintes de synchronisation, permet d'illustrer la variété des questionnements qui se posent pour la résolution de problèmes d'optimisation combinatoire. Les lecteurs peuvent tirer profit des nombreux codes mis à disposition pour approfondir la compréhension des nombreuses problématiques abordées dans l'ouvrage.

J'adresse toutes mes félicitations aux auteurs pour avoir rédigé cet ouvrage et je les remercie vivement de mettre l'ensemble de leur travail à la disposition des étudiants, chercheurs et enseignants-chercheurs.

Prof. Marie-José Huguet

INSA Toulouse, LAAS-CNRS

AVANT-PROPOS

La Programmation Par Contraintes (PPC) permet de concevoir "facilement" des modélisations efficaces pour des problèmes d'optimisation. Cet ouvrage introduit des démarches de modélisations, sur des problèmes précis, prenant en compte les spécificités et les forces de la PPC.

Le principe de la PPC est d'explorer les combinaisons possibles d'affectations de valeurs aux variables en prenant en compte, à la fois, le domaine de chaque variable et les contraintes qui lient ces variables.

De nombreux supports (livres et cours) présentent les méthodes de PPC et proposent des notions théoriques indispensables et parfois complexes, mais il n'existe pas de guide des bonnes pratiques pour concevoir des modèles de PPC efficaces pour des problèmes d'optimisation.

Il ne s'agit pas dans ce livre, de donner des recommandations universelles sur comment "réaliser des modèles efficaces", mais de montrer concrètement au travers de 4 exemples, comment procéder.

Les objectifs poursuivis au travers de ce livre sont ambitieux car :

- le livre a comme objectif de présenter des modélisations de problèmes d'ordonnancement et de transport qui tirent profit d'approches habituellement utilisées dans les métaheuristiques ou les heuristiques ou encore dans des modélisations de type graphes disjonctifs ;
- le livre n'aborde pas la partie théorique : ce livre n'est pas un cours théorique sur la PPC mais il rappelle les fondamentaux ;
- le livre constitue un guide pratique destiné aux ingénieurs, aux chercheurs et aux étudiants désireux de prendre conscience des concepts généraux permettant d'obtenir des modèles efficaces de PPC.

Le livre s'articule autour de 4 exemples qui "couvrent" des problèmes variés d'optimisation avec des problèmes de Job-Shop, des problèmes de transport (VRP), des problèmes de transport multi-périodes avec stocks (IRP) et des problèmes de synchronisation de tournées.

Le **chapitre 1** est une introduction aux outils utilisés en PPC et aux notions de base de la PPC. La notion de contraintes globales est introduite avec les rappels indispensables à une compréhension poussée des concepts utilisés par la suite. Des extraits de code Java sont aussi présentés dans ce chapitre, pour illustrer la manipulation de la librairie Choco.

Le **chapitre 2** s'intéresse à la modélisation d'un problème de transport, le VRP (Vehicle Routing Problem). Ce problème pose de nombreuses difficultés aux solveurs linéaires à cause des contraintes de sous-tours. La modélisation proposée en PPC est basée sur un

ensemble de vecteurs, dépendant les uns des autres et offrant une modélisation efficace et une gestion ingénieuse pour éviter les contraintes de sous-tours.

Le **chapitre 3** présente une modélisation d'un problème d'IRP (Inventory Routing Problem). Il s'agit d'un problème de VRP multi-périodes avec des stocks qui permettent des livraisons anticipées à des clients. L'objectif est de déterminer une solution minimisant les coûts de stockage et les coûts de transport. La modélisation proposée réalise une extension de la modélisation du VRP introduite au chapitre 2, et montre aussi comment définir une approche incrémentale de résolution.

Le **chapitre 4** introduit une nouvelle modélisation du Job-Shop à l'aide de la notion de vecteur de Bierwirth. Les vecteurs de Bierwirth sont utilisés dans de nombreuses métaheuristiques, car ce vecteur définit un ordre topologique et permet d'obtenir un graphe disjonctif acyclique modélisant une solution du problème. Introduite en 1996, cette notion est adaptée à la PPC et permet, sur les instances de Lawrence de définir une approche hybride GRASP×ELS/PPC permettant d'obtenir en quelques secondes une solution de bonne qualité, même sur les instances avec 15 jobs et 10 machines.

Le **chapitre 5** aborde la problématique de synchronisation de visites dans les problèmes de tournées. Ces problèmes de synchronisation apparaissent, par exemple, dans la gestion des personnels de santé qui doivent se rendre simultanément chez un patient, ou successivement chez plusieurs patients. Synchroniser les visites revient à modifier des tournées existantes, et à modifier des dates de passage pour obtenir une solution du problème.

Remarques importantes

1) Dans ce livre, le séparateur décimal utilisé est le "." et le livre reprend, généralement, les conventions anglo-saxonnes. Des anglicismes sont aussi utilisés, dont par exemple le terme "processing time" en lieu et place de "durée de traitement", car ils sont les plus répandus dans la communauté scientifique.

2) Le lecteur familier de la programmation linéaire utilise naturellement les minuscules pour désigner les variables (ainsi x_{ij} est privilégiée à X_{ij} pour les variables) et les majuscules sont utilisées pour désigner des données du problème (ainsi Q désigne une donnée c'est-à-dire une constante). Il s'agit d'un usage de la communauté "programmation linéaire". La communauté "contrainte" a un usage inverse. Ainsi dans un modèle de Programmation Par Contraintes, X_{ij} fait référence à une donnée et q à une variable.

3) Les chapitres peuvent être lus dans n'importe quel ordre à l'exception du chapitre 3 qu'il est préférable de consulter après la lecture du chapitre 2.

4) Les tests ont été réalisés avec NetBeans 8.2, JDK 8 (version 1.8.0_144), JRE 1.8 (version 1.8.0_144) sur un PC avec Windows 7, un processeur Xéon 3.70 GHz et 32 Go de mémoire. En fonction des versions des logiciels, des caractéristiques de la machine utilisée pour les tests, il est possible que des différences puissent apparaître dans les tests numériques proposés dans les chapitres, en particulier lors des tests faisant référence à des temps de calcul ou à des convergences.

5) Les exemples traités sont à visée pédagogique uniquement mais constituent une base qui peut servir de point de départ à des travaux plus poussés.

Les exemples traités de ce livre sont présentés dans le tableau suivant :

Les exemples traités dans ce livre

	Ch. 1	Ch. 2	Ch. 3	Ch. 4	Ch. 5
Exemples introductifs	X				
VRP		X			
IRP			X		
Job-Shop				X	
Synchronisation de tournées					X

Les programmes accompagnant ce livre sont disponibles sur internet à l'adresse suivante :

http://www.isima.fr/~lacomme/PPC_Avancee/index.php

Les programmes disponibles sur le site web sont libres, vous pouvez les redistribuer et/ou les modifier selon les termes de la Licence Publique Générale GNU publiée par la Free Software Foundation (version 2 ou bien toute autre version ultérieure choisie par vous).

Vous pouvez contacter les auteurs en utilisant les adresses suivantes :

Éric Bourreau : Eric.Bourreau@lirmm.fr

Matthieu Gondran : gondran@isima.fr

Philippe Lacomme : placomme@isima.fr

Marina Vinot : vinot@isima.fr

REMERCIEMENTS

Un immense merci aux collègues anonymes et aux membres de nos familles qui, ayant relu et annoté des versions préliminaires de ce document, ont contribué, ainsi, à sa rédaction. Un grand merci à Hélène, Damien et Raksmey nos collègues qui ont lu et corrigé les versions antérieures.

Qu'ils trouvent tous, ici, l'expression de nos remerciements.

Clermont-Ferrand, le 15/05/2019.

Éric Bourreau, Matthieu Gondran, Philippe Lacomme et Marina Vinot

À QUI S'ADRESSE CE LIVRE ?

- Aux **étudiants de L2/L3/Master** qui suivent un parcours en Informatique. En effet, il introduit les notions de base en modélisation pour la Programmation Par Contraintes et contient de nombreux exemples. Les différents chapitres apportent des éléments de mise en œuvre tant sur la modélisation que la résolution de différentes problématiques combinatoires.

- Aux **élèves d'écoles d'ingénieurs en Informatique** qui ont un cursus comprenant une partie optimisation. Ce livre constitue pour eux une introduction sur la réalisation de modèles efficaces en PPC ; ils en trouveront les éléments essentiels. Pour eux, ce livre est un résumé des connaissances minimales à avoir en PPC, dans le cadre de leur travail d'ingénieur. Ce livre constitue donc une synthèse des bonnes pratiques, un mémento auquel ils peuvent se référer dans leur travail d'analyse, et de conception d'outils d'aide à la décision.

- Aux **ingénieurs généralistes** qui peuvent profiter des apports de la Programmation Par Contraintes pour la résolution de problèmes d'optimisation dans leurs projets et qui cherchent des exemples et une démarche de mise en œuvre. La lecture de cet ouvrage éclaire les ingénieurs sur les nouveaux outils de modélisation issus de la PPC.

- Aux **enseignants responsables de cours de programmation linéaire, de cours d'optimisation et de cours de Programmation Par Contraintes** qui sont à la recherche d'exemples non triviaux pour illustrer leurs TP et leurs TD. Les exemples proposés, ici, peuvent aussi servir à la réalisation de nouveaux cours et permettent de faire le lien entre les notions vues en cours d'optimisation et celles abordées dans les cours de PPC : en effet, les exemples proposés relèvent de la thématique "optimisation".

TABLE DES MATIERES

Chapitre 1 .....Notions fondamentales en PPC
1.1 Le solveur Choco 11
1.2 Principes de la PPC 12
1.3 Contraintes globales de Choco 21
1.4 Paramétrage du solveur Choco 35
1.5 Conclusion 39
1.6 Références 39

Chapitre 2 .....Modélisations de problèmes de type VRP
2.1 Problème du VRP 41
2.2 Amélioration du modèle : règles à respecter 58
2.3 Création des modèles 59
2.4 Utilisation de stratégies de branchement existantes 79
2.5 Démarrer le solveur à partir d'une solution existante 86
2.6 Prise en compte de la topologie des instances 87
2.7 Conclusion 90
2.8 Références 90

Chapitre 3 ..... Modélisations de problèmes de type IRP
3.1 Problème de l'IRP 91
3.2 Modélisation d'un IRP à seul produit 97
3.3 Des instances d'IRP 121
3.4 Définition d'un schéma de type métaheuristique gloutonne 131
3.5 Performances du modèle sur les instances abs 146
3.6 Conclusion 148
3.7 Références 149

Chapitre 4 .....Modélisations de problèmes de type Job-Shop
4.1 Job-Shop 151
4.2 Modélisation avec des vecteurs de Bierwirth 158
4.3 Modélisation PPC avec des vecteurs de Bierwirth 162
4.4 Schéma itératif basé sur la PPC 172
4.5 Évaluation numérique de l'approche 177
4.6 Définition d'un schéma global d'optimisation 178
4.7 Conclusion 182
4.8 Références 182

Chapitre 5 .....Modélisations de problème de type VRP avec synchronisation
5.1 Synchronisation de tournées 185
5.2 Modélisation de problèmes de tournées : rappels 194
5.3 Les données du problème 195
5.4 Modélisation des contraintes de synchronisation 199
5.5 Modélisation complète du problème 200
5.6 Évaluation numérique du modèle 202
5.7 Utilisation du modèle pour des instances de grande taille 205
5.8 Réalisation d'un modèle efficace 207
5.9 Conclusion 228
5.10 Références 229

Index 231

CHAPITRE 1

Notions fondamentales en PPC

Objectifs : Ce chapitre introduit les concepts de base de la Programmation Par Contraintes (PPC) ainsi que quelques notions théoriques indispensables à l'élaboration de modèles efficaces. Les principes présentés sont communs à un grand nombre de solveurs PPC, mais les exemples donnés dans ce chapitre sont réalisés avec le solveur Choco. Toutefois ces concepts sont facilement transposables à d'autres solveurs. La majorité des exemples et des éléments de base en PPC sont empruntés à (Bourreau et al. 2019) paru chez le même éditeur. Ce chapitre n'est pas un cours sur la PPC, mais un rappel des briques fondamentales.

1.1 Le solveur Choco

Choco est un solveur open source de PPC maintenu par une équipe de chercheurs et d'ingénieurs principalement issus de l'École des Mines de Nantes. La version la plus récente à l'heure où ce livre est écrit, est Choco 4.10.0. Le site de référence de Choco se trouve à l'adresse **http://www.choco-solver.org/**. Il comprend une documentation, des exemples, les codes sources et un lien de téléchargement pour la librairie (Figure 1-1).

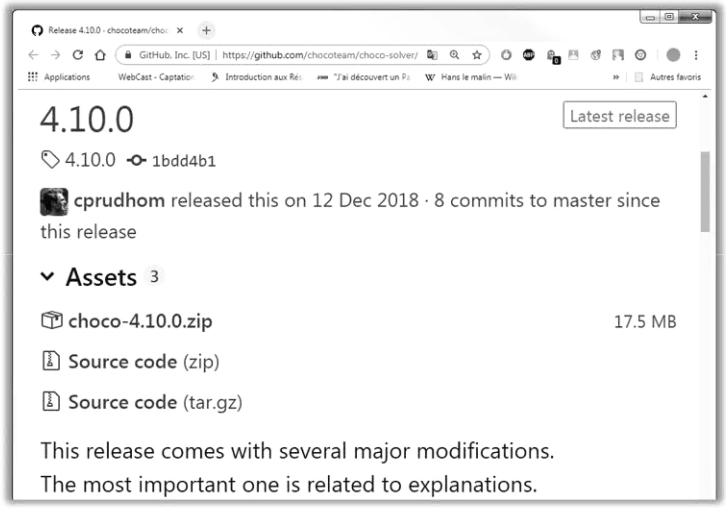


Figure 1-1. Téléchargement via Github

L'utilisation de ce solveur nécessite un environnement Java et celui adopté dans ce livre est NetBeans : **https://NetBeans.org/downloads/**

Afin d'utiliser le solveur PPC Choco à partir d'un projet NetBeans, il est nécessaire d'ajouter les trois librairies .jar dans les "Libraries" (dans la majorité des cas, seul le fichier **choco-solver-4.10.0.jar** est suffisant). L'ajout des librairies Choco s'effectue avec un clic droit sur "Libraries" puis "Add jar/Folder" (Figure 1-2). Les trois librairies sont les suivantes :

- **"choco-solver-4.10.0-sources.jar"**
- **"choco-solver-4.10.0-with-dependencies.jar"**
- **"choco-solver-4.10.0.jar"**

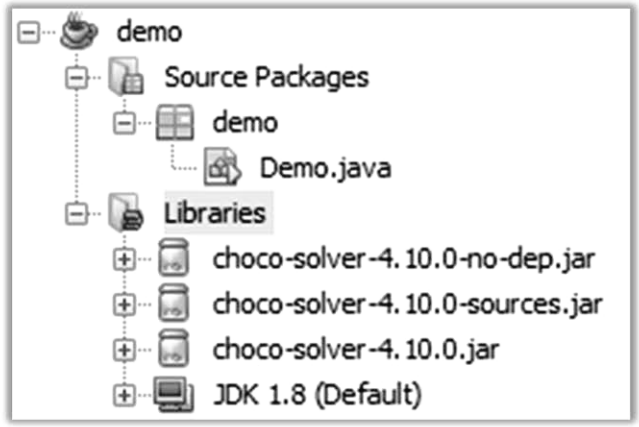


Figure 1-2. Ajout des librairies dans le projet

1.2 Principe de la PPC

1.2.1 Introduction (Régin, 2004)

La PPC est une technique de résolution issue de l'Intelligence Artificielle (IA) dont les premiers éléments datent des années 70. En PPC, un problème se définit par des variables et des contraintes. Une variable est dotée d'un domaine qui donne la liste des valeurs possibles que peut prendre la variable alors qu'une contrainte exprime une propriété qu'un ensemble de variables doit satisfaire. Par exemple, $x + y = 5$ est une contrainte liant les domaines des deux variables x et y.

Le **filtrage** consiste à supprimer du domaine des variables les valeurs qui ne peuvent pas appartenir à une solution du problème d'après les contraintes. Ainsi sur l'exemple précédent ($x + y = 5$), si $dom(x)$={1, 2, 3, 4} et $dom(y)$={1, 2, 3, 4, 5, 6}, alors après filtrage (et donc prise en compte de la contrainte $x + y = 5$) les domaines sont réduits à : $dom(x)$={1, 2, 3, 4} et à $dom(y)$={1, 2, 3, 4}. L'affectation de la valeur 5 ou de la valeur 6 à la variable y est impossible, car aucune solution ne peut respecter la contrainte $x + y = 5$ si $y = 5$ ou $y = 6$.

Lorsqu'un domaine est modifié (c'est-à-dire des valeurs retirées du domaine), il faut parcourir l'ensemble des contraintes dans lesquelles la variable apparaît (généralement, une variable est impliquée dans plusieurs contraintes). La suppression d'une ou plusieurs

valeurs d'un domaine peut aboutir à de nouvelles déductions : ce mécanisme est appelé **propagation**.

Par exemple, si la contrainte $x < y$ est aussi présente (en plus de $x + y = 5$), la suppression de 5 dans le domaine de y entraîne la suppression de 4 dans le domaine de x. De nouveau, la suppression de 4 dans le domaine de x doit être propagé sur $x + y = 5$ et implique que la valeur 1 n'appartient plus au domaine de y. Au final, après 3 activations de contraintes et la suppression de 4 valeurs, les domaines sont $dom(x)$={1, 2, 3} et $dom(y)$={2, 3, 4}.

La recherche d'une solution consiste à affecter successivement une valeur aux différentes variables, et à chaque affectation à utiliser les mécanismes de filtrage et de propagation. L'affectation successive d'une valeur aux variables revient à parcourir un arbre de recherche dans lequel il est possible de distinguer 3 types de nœuds :

- Les nœuds dont les domaines des variables sont réduits à une seule valeur sont des feuilles de l'arbre, modélisant une solution du problème.
- Les nœuds pour lesquels, le domaine d'une variable est de taille nulle sont des nœuds à partir desquels il est inutile de continuer l'énumération car aucune solution (affectation réalisable) ne peut découler des précédents branchements. Au niveau de ces nœuds, il faut opérer un retour arrière (backtrack) et remettre en cause une affectation précédemment réalisée.
- Enfin, il reste les nœuds pour lesquels une partie des variables ont été affectées et à partir desquels il faut poursuivre l'énumération. L'ensemble de ces notions est repris et détaillé dans les paragraphes suivants.

À noter que dans la suite de ce livre, l'exposé est limité à des variables dont les domaines sont entiers et finis. Il est cependant possible d'appliquer tous ces concepts sur des domaines continus, ensemblistes ou logiques. Un exemple dans le chapitre 3 utilise des variables réelles.

1.2.2 Variable – Domaine – Contrainte

En programmation linéaire un problème se définit avec des contraintes linéaires, des variables entières et réelles. Par exemple, le programme linéaire ci-dessous se compose de 3 variables et de 3 contraintes (toutes linéaires).

$$\text{Minimiser } f(x_1, x_2, x_3) = 4x_1 + 2x_2 + x_3$$

$$\begin{aligned} \text{sous les contraintes :} \quad & x_1 + x_2 \geq 1.1 && (1)\\ & x_2 + x_3 \geq 3.2 && (2)\\ & x_1 + x_3 \geq 4.3 && (3)\\ & x_i \in \mathbb{R},\ i \in \{1, 2, 3\} \end{aligned}$$

Résoudre ce problème en PLNE (Programmation Linéaire en Nombres Entiers) se fait classiquement avec l'algorithme du Simplex et revient à énumérer une partie des sommets du polyèdre.

En Programmation Par Contraintes, une variable est définie par un domaine de valeurs possible. La définition des contraintes est très différente puisqu'un modèle PPC peut contenir des contraintes linéaires, non linéaires, logiques, etc.

Le problème de **coloriage de graphe** est l'exemple type pour illustrer le principe de la PPC. Les différentes couleurs sont représentées par des entiers compris entre 1 et c où c est une constante définissant le nombre maximal de couleurs utilisables. Les sommets sont modélisés par les variables C_i à valeur dans $[1\,;c]$. x_{ij} est une donnée et vaut 1 si une arête existe entre deux sommets i et j du graphe et 0 sinon.

Le modèle donc se compose de deux contraintes :

$$\forall i = 1..n \qquad C_i \in [1\,;c] \qquad (1)$$

$$\forall i = 1..n, \forall j = 1..n\ /x_{ij} = 1 \qquad C_i \neq C_j \qquad (2)$$

L'intervalle $[1\,;c]$ définit le domaine de la variable C_i. La contrainte (2) exprime simplement que, si un arc relie le sommet i au sommet j, les deux sommets ne peuvent pas avoir la même couleur.

Ainsi, si la couleur 2 est affectée à la variable C_i alors la contrainte (2) propage le fait que la valeur 2 doit être supprimée des valeurs possibles pour la variable C_j (Figure 1-3).

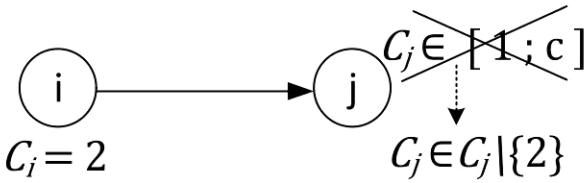


Figure 1-3. Propagation de la contrainte

1.2.3 Arbre de recherche

Un problème modélisé en PPC est composé d'un ensemble de variables $V = \{X_i\}_{i=1,n}$, où n est le nombre de contraintes ; d'un ensemble de domaines $D = \{D_i\}_{i=1,n}$ où chaque domaine D_i est associé la variable X_i ; et d'un ensemble C de contraintes $C = \{C_i\}_{i=1,m}$ où m est le nombre de contraintes.

Un solveur PPC parcourt l'arbre de recherche associé au problème en considérant qu'un nœud (ou un état) de l'arbre est caractérisé par :

- un ensemble de variables X_k $(k = 1..n_1)$, qui possèdent des domaines D_k comprenant une unique valeur. Ces variables sont affectées d'une valeur v_k appartenant au domaine initial D_k et représente une solution partielle ;
- un ensemble de variables X_j $(j = 1..n_2)$, dont les domaines D_j contiennent plusieurs valeurs ;
- avec $n_1 + n_2 = n$ où $n = |V|$ est le nombre total de variables.

Une affectation est **compatible** si toutes les contraintes sont respectées. Un nœud peut représenter une affectation partielle compatible du problème, mais une affectation partielle n'est pas nécessairement compatible. En effet, la vérification complète d'une contrainte ne peut se faire que lorsque toutes les variables sont instanciées.

Maintenir et vérifier la **consistance** des domaines de chaque variable pour l'ensemble des contraintes ne suffit pas toujours pour obtenir une consistance globale des domaines des variables.

C'est le cas dans l'exemple de la section 1.2.1, où les domaines $dom(x)$={1, 2, 3} et $dom(y)$={2, 3, 4} sont compatibles pour chaque contrainte. En effet, les solutions de la forme $\{x,y\}$ peuvent être valides pour la contrainte d'égalité ($\{1;4\},\{2;3\},\{3;2\}$) et pour la contrainte $x < y$ ($\{1;2\},\{1;3\},\{1;4\},\{2;3\},\{2;4\},\{3;4\}$), bien que la valeur $\{2\}$ devrait être supprimée du domaine de y si on avait une vision globale. En effet, si $y = \{2\}$, le seul couple acceptable pour la contrainte d'égalité est $\{3;2\}$, or celui-ci n'est pas présent dans la liste des couples compatibles de la contrainte $x < y$. Une remarque similaire existe pour $x = \{3\}$: si $x = \{3\}$, l'unique couple possible est $\{3;2\}$ (la somme 3 + 2 fait bien 5), mais celui-ci ne satisfait pas la contrainte $x < y$.

Lorsqu'une variable X_i possède un domaine vide (toutes ses valeurs de départ ont été supprimées/filtrées), alors aucune affectation ne permet d'obtenir une solution : les valeurs déjà affectées aux autres variables n'offrent pas la possibilité de terminer l'affectation.

Si une valeur a été affectée à chacune des variables, l'affectation est **complète**. Une affectation **compatible** (les contraintes sont satisfaites) et **complète** (toutes les variables sont affectées) est une **solution** du problème.

En résumé, un nœud de l'arbre de recherche modélise une solution partielle dans laquelle les domaines de certaines variables sont des singletons (valeur d'affectation connue) alors que d'autres variables ont des domaines réduits par filtrage et propagation. La recherche de solutions d'un problème consiste à parcourir les nœuds de l'arbre. Cinq principes gouvernent le parcours d'un arbre de recherche :

- Le branchement : à chaque nœud n, une variable est sélectionnée et une valeur de son domaine lui est affectée, donnant naissance à un nœud $n + 1$. Il y a $|V^n|$ variables, avec un domaine comprenant au minimum deux valeurs, sur lesquelles le solveur peut brancher que de variables. Pour chaque variable $v \in V^n$, il y a $|D_v^n|$ choix de valeurs possibles, où D_v^n est le domaine de la variable v sur le nœud n. Soit v la variable sur laquelle le solveur branche, avec la valeur m.
- Le filtrage : les algorithmes de filtrages vérifient les domaines des variables V^c ayant une contrainte en commun avec v et suppriment les valeurs impossibles.
- La propagation : les algorithmes de propagations réduisent les domaines des variables ayant une contrainte en commun avec une variable de l'ensemble V^c.
- Le backtrack : si une variable a un domaine vide, ou une contrainte est non respectée à cause du choix de branchement, alors un backtrack est nécessaire car le nœud $n + 1$ est à une affectation partielle dont aucune solution ne peut découler de ce nœud. Un backtrack permet de remonter dans l'arbre au nœud n et de réaliser d'autres choix de branchement afin de parcourir l'arbre. Si toutes les variables sont affectées à une valeur, alors le nœud est une solution. Le solveur réalise un backtrack pour chercher d'autres solutions.

Les principes de filtrages et de propagations sont parfois regroupés sous le terme générique "d'inférence".

1.2.4 Exemple du Job-Shop en PPC

Le Job-Shop est un problème classique d'ordonnancement où des pièces (ou jobs) sont à fabriquer sur des machines, sachant qu'une machine ne peut traiter qu'une seule pièce à

la fois. Chaque pièce est composée d'une gamme définissant l'ordre des machines que doit parcourir la pièce ainsi que la durée du traitement fait par chaque machine.

Un exemple, tiré de (Bourreau et al, 2019), illustre le problème du job-shop. Les gammes et les temps de passage sur les machines (ou processing times) pour chaque pièce sont introduits par le Tableau 1-1. Par exemple, la première opération de la pièce P1 est réalisée sur la machine M2 pendant 21 unités de temps. Ensuite P1 est traitée sur la machine M1 pendant 53 unités de temps (seconde opération de P1) ; et enfin P1 passe sur la machine M3 pendant 95 unités de temps. La pièce P2 commence son parcours sur la machine M1 pendant 21 unités de temps, puis passer sur la machine M2, etc.

Tableau 1-1. Gammes et temps de traitement des différentes pièces

PIECE\OPERATION	1	2	3
P1	M2(21)	M1(53)	M3(95)
P2	M1(21)	M2(52)	M3(16)
P3	M2(39)	M3(98)	M1(42)

Une solution du problème de job-shop est la planification de chaque opération, en respectant les gammes de chaque pièce et en prenant en compte qu'une machine ne peut traiter qu'une seule opération à la fois. Trouver un ordonnancement des opérations correspond à fixer les dates de début $St_{i,j}$ et de fin des opérations $Ft_{i,j}$ de chaque opération $O_{i,j}$.

L'objectif du job-shop est donc de déterminer les dates de début et les dates de fin de chacune des opérations qui permettent de minimiser le makespan sachant que le makespan est la date de fin de la dernière opération sur la dernière machine.

Modélisation PPC du problème de job-shop

Les données du problème :

- m le nombre de machines
- n le nombre de pièces à traiter (également appelées jobs)
- $O_{i,j}$ l'opération j de la pièce i
- $m_{i,j}$ la machine utilisée pour l'opération $O_{i,j}$
- $p_{i,j}$ la durée de traitement (processing time) de l'opération $O_{i,j}$
- D_{max} la taille de l'horizon de planification

Les variables de décisions utilisées en PPC :

- $St_{i,j}$ date de début de traitement de l'opération $O_{i,j}$
- $Ft_{i,j}$ date de fin de traitement de l'opération $O_{i,j}$
- C_{max} makespan : date maximale des $Ft_{i,j}$

Le modèle se compose des sept contraintes suivantes :

$$\forall i = 1..n, \forall j = 1..m \qquad Ft_{i,j} \in [0\,;D_{max}] \qquad (1)$$

$$\forall i = 1..n, \forall j = 1..m \qquad St_{i,j} \in [0\,;D_{max}] \qquad (2)$$

$$\forall i = 1..n, \forall j = 1..m \qquad Ft_{i,j} = St_{i,j} + p_{i,j} \qquad (3)$$

$$\forall i = 1..n, \forall j = 2..m \qquad St_{i,j} \geq Ft_{i,j-1} \qquad (4)$$

$$\forall i = 1..n, \forall j = 1..m, \forall u = 1..n, \forall v = 1..m \text{ avec } m_{i,j} = m_{u,v} \quad St_{i,j} \geq Ft_{u,v} \text{ ou } St_{u,v} \geq Ft_{i,j} \qquad (5)$$

$$\forall i = 1..n \quad C_{max} \geq Ft_{i,m} \qquad (6)$$

$$min(C_{max}) \qquad (7)$$

1.2.5 Énumération de l'arbre de recherche : principe avec l'exemple du job-shop

L'objectif de cette section est d'illustrer le principe des inférences et des branchements sur l'exemple du job-shop, lorsque celui-ci est modélisé sous la forme de graphe disjonctif (Roy and Sussmann, 1964).

Une itération correspond à un nouveau branchement (sélection d'une variable plus choix de la valeur), et plusieurs inférences peuvent être mises en évidence. Chacune des inférences est illustrée par un nouveau graphe disjonctif afin de modéliser les nouveaux domaines des différentes variables.

La nomenclature des différents graphes est de la forme $G_{itération}^{(étape)}$ afin d'aider le lecteur à associer chaque graphe à l'itération et à l'étape que celui-ci modélise. Le graphe disjonctif final obtenu à l'issue d'une itération i (c'est-à-dire de toutes les inférences possibles) est noté $G_i^{(*)}$.

La Figure 1-4 présente le principe global du parcours avec les différentes itérations et les différentes étapes. Toutes les étapes sont présentées en détail dans (Bourreau et al., 2019).

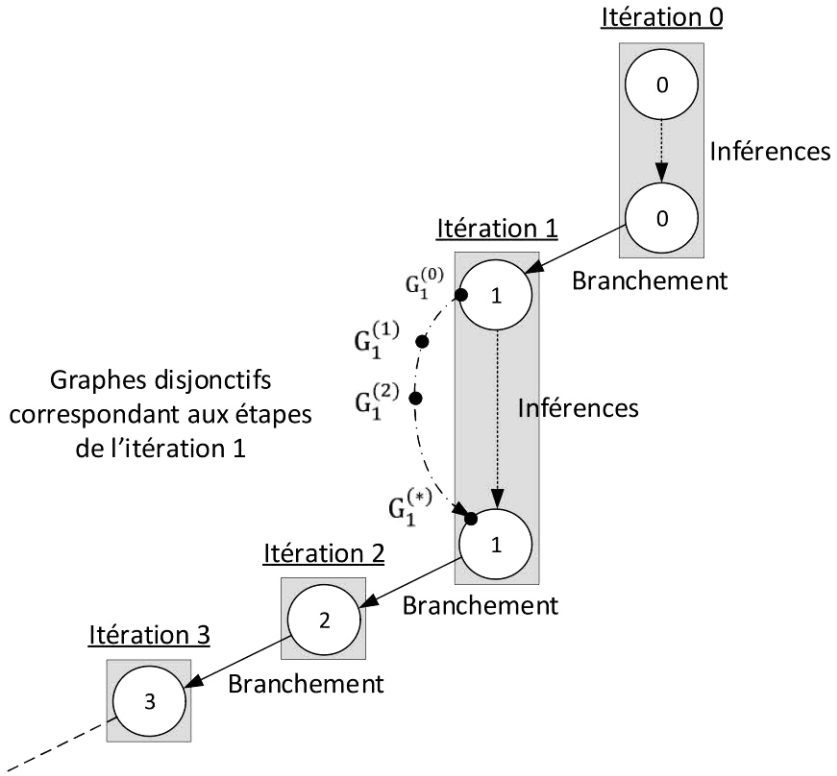


Figure 1-4. Principe de l'arbre de recherche (Bourreau et al., 2019)

Graphe à la racine : itération 0

La Figure 1-5 présente le graphe disjonctif initial $G_0^{(0)}$ où $D_{max} = 300$. La numérotation des sommets sous la forme $O_{i,j}$ est également rappelée sur cette figure. Les valeurs des domaines de chaque variable $St_{i,j}$ et $Ft_{i,j}$ sont les valeurs initiales, les inférences liées aux contraintes (liant notamment les $St_{i,j}$ et les $Ft_{i,j}$) ne sont pas encore vérifiées. Sur la Figure 1-5, pour chaque sommet $O_{i,j}$ le domaine de la date de début $St_{i,j}$ (première ligne) et le domaine de la date de fin $Ft_{i,j}$ (deuxième ligne) sont initialement défini par l'intervalle $[0\,;D_{max} = 300]$.

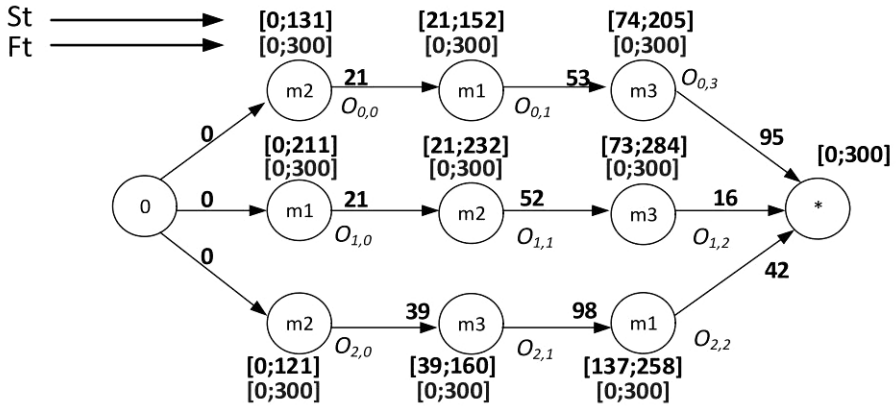


Figure 1-5. Le graphe disjonctif $G_0^{(0)}$

Même sans branchement, les domaines des variables $St_{i,j}$ et $Ft_{i,j}$ peuvent être réduit en considérant les durées de traitement d'une même pièce sur les différentes machines composant sa gamme. Par exemple, la pièce numérotée 0 (première ligne du graphe) passe successivement sur les machines $m2$, $m1$ et $m3$ avec les durées de traitement respectif 21, 53 et 95. La pièce a donc un temps de traitement minimal égal à $21 + 53 + 95 = 169$. Par conséquent, si la date de fin $Ft_{0,3} \leq 300$ (d'après son domaine $Ft_{0,3} = [0\,;D_{max} = 300]$), alors l'opération $O_{0,0}$ doit débuter dans l'intervalle $St_{0,0} = [0\,;300 - 169 = 131]$. En s'intéressant à toutes les opérations $O_{i,j}$, le même raisonnement peut être appliqué en prenant en compte les opérations $O_{i,j'}$ qui suivent $O_{i,j}$ avec $j' > j$.

De plus, la borne inférieure des domaines des variables peut également être améliorée : par exemple, l'opération $O_{0,1}$ ne peut pas débuter à la date 0 puisque l'opération $O_{0,0}$ qui précède à une durée de 21, d'où le domaine $St_{0,1} = [21\,;152 = 300 - (53 + 95)]$.

La Figure 1-6 présente le graphe final $G_0^{(*)}$ des inférences à la racine de l'arbre où l'ensemble des domaines sont mis à jour avec des raisonnements similaires à ceux présentés précédemment.

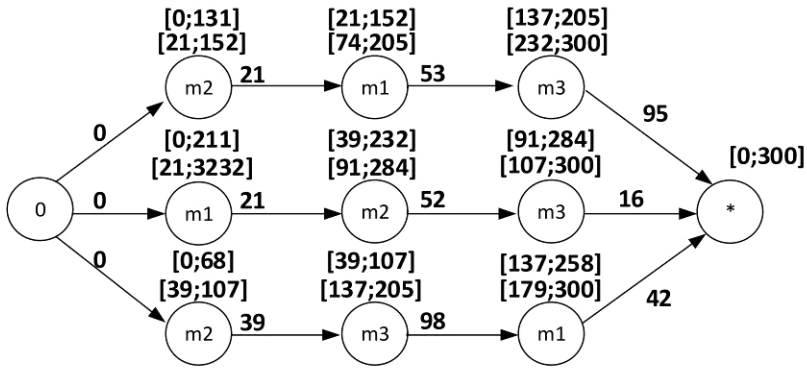


Figure 1-6. Le graphe disjonctif $G_0^{(*)}$ (après inférence)

La première décision de branchement consiste à :

- Sélectionner la variable possédant le plus petit domaine (stratégie par défaut de Choco, correspondant à **FirstFail()**) : $St_{0,2}$.
- Affecter la plus petite valeur du domaine (stratégie par défaut de Choco) à la variable $St_{0,2}$, cette valeur est 137.
- Inférer (filtrer et propager) cette information vers les autres variables du problème.

De ce branchement découle le graphe $G_1^{(0)}$ de la Figure 1-7.

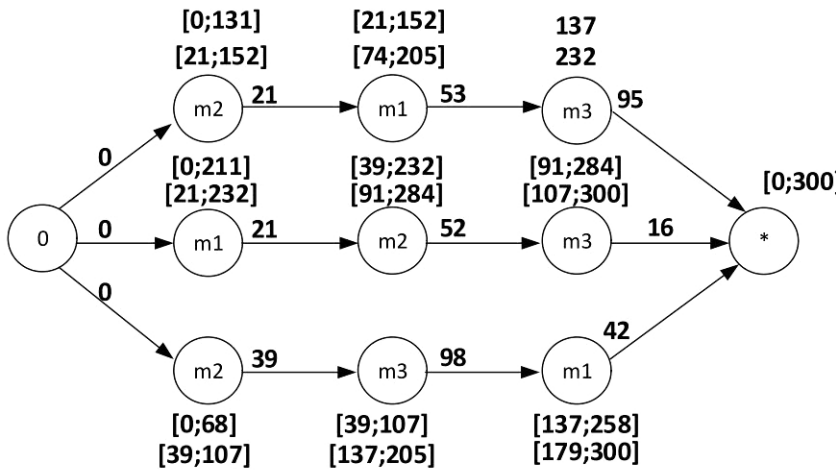


Figure 1-7. Le graphe disjonctif $G_1^{(0)}$

En choisissant de brancher avec $St_{0,2} = \{137\}$ et d'après les contraintes conjonctives entre les opérations de la pièce 0, le solveur déduit que $St_{0,0} = [0; 63]$ car 137-53-21 = 63. D'autres déductions peuvent être faites après la réduction des domaines des opérations $O_{0,0}$ et $O_{0,1}$, ces déductions concernent également les autres opérations utilisant les machines $m1$ et $m2$. Pour la machine $m2$, l'opération $O_{0,0}$ est soit avant, soit après l'opération $O_{1,1}$, de même entre les opérations $O_{0,0}$ et $O_{2,0}$. Certains cas deviennent impossibles et permettent la réduction des domaines. La Figure 1-8 présente le graphe $G_1^{(*)}$ à la fin de toutes les inférences et de toutes les réductions de domaine.

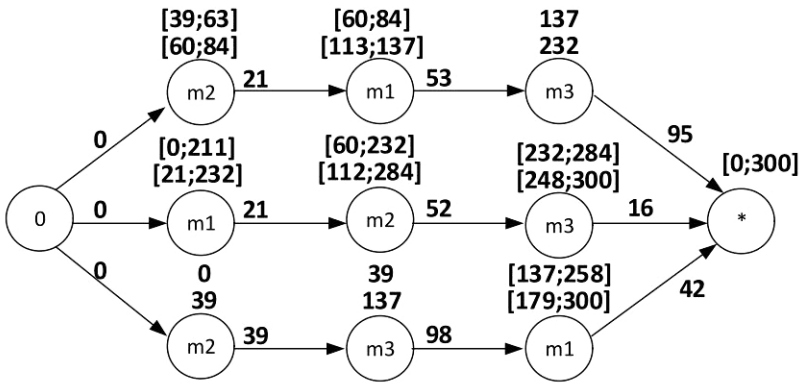


Figure 1-8. Le graphe disjonctif $G_1^{(*)}$

Après cette étape d'inférences, le processus est itéré jusqu'à ce que toutes les variables soient affectées (le domaine de chaque variable se réduit à un singleton). Sur cet exemple "simple", aucun backtrack n'est nécessaire (Figure 1-9) et le solveur obtient "directement" une solution de valeur 248.

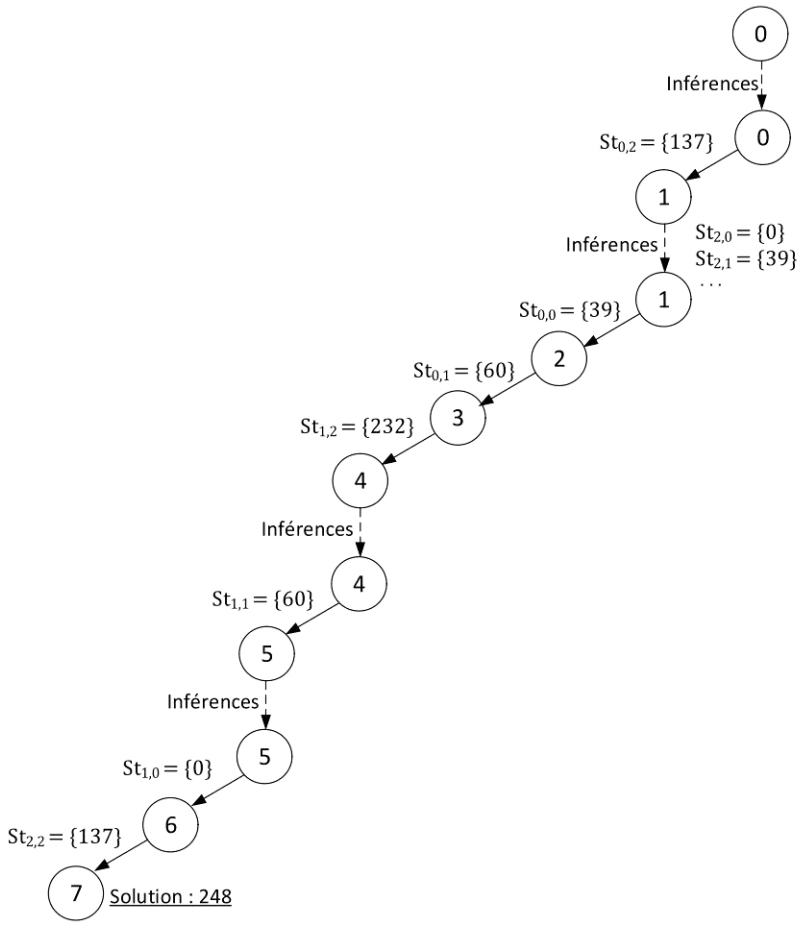


Figure 1-9. Trajectoire suivie par le solveur dans l'arbre de recherche

1.2.6 Arbre de recherche : les points clés

La résolution d'un problème de PPC est combinatoire dans le sens où il faut tester plusieurs affectations de valeurs aux variables pour obtenir une solution. Trouver la bonne solution, c'est-à-dire la solution minimisant un critère nécessite encore plus de puissance de calcul, car il s'agit de trouver la feuille modélisant la solution optimale. La manière de parcourir l'arbre de recherche doit être adaptée aux problèmes modélisés et deux éléments doivent être pris en compte :

- la capacité à obtenir une solution le plus rapidement possible en faisant le moins possible de backtracks ;
- la capacité à explorer efficacement le reste du graphe lorsqu'une première solution a été obtenue.

Ces deux éléments reposent sur la définition d'une politique adaptée :

- pour sélectionner la variable ;
- pour choisir une valeur dans le domaine de la variable.

1.3 Contraintes globales de Choco

1.3.1 Enjeux de la modélisation

Un modèle PPC est défini par un ensemble de variables, un ensemble de domaines et un ensemble de contraintes. Afin d'obtenir une modélisation efficace d'un problème, un grand soin doit être porté aux définitions des contraintes. En effet, un solveur PPC est plus rapide si les contraintes permettent une bonne propagation, dans ce cas des pans entiers de l'arbre de recherche ne sont pas explorés, une solution peut être obtenue après peu de branchement, et le solveur peut découvrir très tôt si les branchements déjà effectués permettront de trouver ou non une solution faisable. Il faut donc apporter une attention particulière aux points suivants :

- des modifications réalisées par un filtrage doivent pouvoir impacter d'autres variables par propagation à travers les contraintes ;
- plus des modifications des domaines des variables ont lieu tôt dans l'arbre de recherche, plus des parties importantes de l'arbre ne sont pas parcourues ;
- certaines contraintes, à privilégier, engendrent des modifications importantes des domaines et contribuent donc à réduire significativement l'arbre de recherche ;
- les contraintes doivent donc être aussi peu locales que possible ;
- il est préférable d'utiliser des contraintes globales qui par définition encapsulent d'autres contraintes et ont des algorithmes de filtrage et propagation souvent très puissants permettant de nombreuses réductions des domaines. Toutefois, ces algorithmes ont une complexité qui peut être élevée.

L'enjeu de la modélisation est de trouver un compromis entre les contraintes qui permettent d'obtenir de fortes réductions des domaines mais qui utilisent des algorithmes ayant une complexité élevée, et les contraintes qui ont un algorithme de filtrage et de propagation de plus faible complexité. Le risque des contraintes avec des algorithmes ayant une complexité élevée est de passer beaucoup de temps à chercher des réductions des domaines, à chaque nœud de l'arbre de recherche, sans résultat significatif.

1.3.2 Définition et intérêt

Plusieurs types de contraintes élémentaires peuvent être utilisés en programmation par contraintes, les contraintes binaires (à deux variables, comme $X < Y$), ternaires (à trois variables, comme **arithm()** avec $X + Y = Z$) ou alors n-aire (avec un ensemble de n variables, comme **sum()** ou **allDifferent()**).

Une contrainte globale peut être une union de contraintes élémentaires qui facilitent la modélisation et accélèrent la résolution, car elles incluent en plus des algorithmes efficaces de propagation. C'est l'exemple de la contrainte globale **allDifferent()** qui permet d'affecter une valeur différente à chaque variable. Cette contrainte réalise plus de réductions de domaines que la réécriture simple des inégalités entre tous les couples de variables. Ce principe des contraintes globales est illustré sur un problème d'affection. Il est montré sur cet exemple que l'ensemble des inférences possibles ne peuvent pas être obtenues par une somme d'inférences locales.

1.3.3 Notion de consistance locale et globale

Cette notion est illustrée en reprenant un exemple de David Savourey et disponible dans son cours "Programmation par Contraintes". D. Savourey considère un problème de coloriage de quatre pièces (P1, P2, P3, P4) pour lesquelles trois couleurs sont possibles : B pour bleu, R pour rouge et J pour jaune. La Figure 1-10 illustre ce problème dans lequel chaque variable (les pièces) est représentée par une ellipse comprenant les valeurs possibles d'affectation.

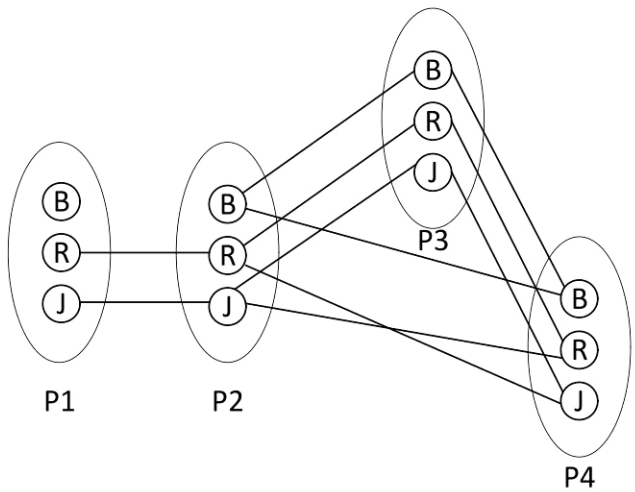


Figure 1-10. Problème de coloriage : couleurs compatibles

Un arc dans le graphe (Figure 1-10) représente un choix de couleurs compatibles. Ainsi l'arc entre P1 de couleur R et P2 de couleur R montre que le choix simultané de R pour P1 et P2 est possible. Le graphe montre par exemple, qu'il n'est pas possible d'affecter R à P1 et J à P2 puisqu'aucun arc ne relie la couleur R de P1 à la couleur J de P2. Il est possible d'analyser le réseau pour identifier des affectations de couleur à une pièce P qui sont localement inconsistantes c'est-à-dire qui ne peuvent pas permettre de propager ce choix vers les autres pièces. Ainsi, la couleur B peut être supprimée des couleurs possibles pour P1 puisque cette couleur n'est compatible avec aucune couleur de P2 (Figure 1-11).