Programmation par Contraintes (Modélisation)

Notes de cours

Nadjib Lazaar
Université Paris-Saclay
lazaar@lisn.fr

Introduction

La programmation par contraintes (PPC) est une approche déclarative permettant de modéliser et résoudre des problèmes combinatoires complexes. En définissant un ensemble de variables, domaines et contraintes, cette approche facilite l'expression et la résolution automatique des problèmes sans expliciter un algorithme particulier.

1 Historique et Motivation

La PPC trouve ses origines dans les années 1960-1970, notamment en intelligence artificielle, en programmation logique et en recherche opérationnelle. À cette époque, les chercheurs exploraient différentes approches pour résoudre des problèmes combinatoires de manière plus efficace qu'avec les méthodes traditionnelles de recherche exhaustive. L'idée fondamentale était d'exploiter des contraintes pour réduire l'espace de recherche et accélérer la résolution des problèmes.

Dans les années 1980, les premiers langages dédiés à la PPC ont émergé, notamment **CLP (Constraint Logic Programming)**, qui combinait la logique déclarative du langage Prolog avec la gestion des contraintes. Cette avancée a permis de mieux structurer et d'automatiser la résolution de nombreux problèmes en intelligence artificielle et en optimisation.

Dans les années 1990-2000, la PPC a connu une industrialisation progressive avec l'apparition de solveurs génériques tels que *ILOG Solver*, *ECLiPSe* et *Choco*. Ces outils ont facilité l'adoption de la PPC dans des domaines variés comme la planification, la logistique et l'optimisation des ressources.

Aujourd'hui, la PPC est largement utilisée dans l'industrie grâce à des solveurs performants comme *Gecode*, *Google OR-Tools* et *IBM Ilog CP Optimizer*. Ces solveurs permettent d'exprimer des modèles de manière déclarative et

d'utiliser des techniques avancées de propagation de contraintes et de recherche heuristique pour trouver rapidement des solutions optimales. La PPC continue d'évoluer, notamment avec son intégration dans des approches hybrides combinant PPC et apprentissage automatique.

2 Comparaison des Paradigmes de Programmation

La PPC appartient au paradigme déclaratif, qui se distingue des autres paradigmes de programmation.

La programmation déclarative, et en particulier la PPC, se distingue par sa capacité à exprimer un problème sous forme de relations logiques et de contraintes, laissant le solveur déterminer la meilleure solution possible. Cette approche est particulièrement utile pour les problèmes combinatoires et d'optimisation où les méthodes traditionnelles seraient trop coûteuses en temps de calcul.

Paradigme	Description	Exemples de Langages
Impératif	Décrit comment résoudre un problème à l'aide d'instructions successives	C, Java, Python (mode impératif)
Orienté Objet	Structure le code en objets et interactions entre eux	Java, C++, Python (mode objet)
Fonctionnel	Basé sur l'évaluation de fonctions sans état mutable	Haskell, Lisp, Scala
Déclaratif	Spécifie le problème à résoudre sans détailler comment le résoudre	Prolog, SQL, MiniZinc

TABLE 1 – Comparaison des Paradigmes de Programmation

3 Fondements de la PPC

Un problème de satisfaction de contraintes (CSP) est défini par un **ensemble fini de variables**, chacune prenant des valeurs dans un **domaine fini**, et un **ensemble de contraintes** exprimant des relations entre ces variables.

- **Variables** : Un CSP est défini par un ensemble fini de variables $X = \{x_1, x_2, ..., x_n\}$.
- **Domaines** : Chaque variable x_i possède un domaine fini $D(x_i)$ contenant les valeurs qu'elle peut prendre.
- **Contraintes** : Les contraintes sont des relations entre les variables, restreignant les combinaisons de valeurs possibles.

3.1 Types de Contraintes

Les contraintes peuvent être définies de différentes manières :

- **Contraintes d'arité 1 (unaires)** : elles concernent une seule variable, par exemple $x_1 \neq 3$.
- **Contraintes d'arité 2 (binaires)** : elles relient deux variables, par exemple $x_1 \neq x_2$.
- **Contraintes d'arité k (n-aires)** : elles concernent plus de deux variables, par exemple $x_1 + x_2 + x_3 \leq 10$.

3.2 Modes de Définition des Contraintes

Les contraintes peuvent être spécifiées de plusieurs manières :

- **En extension** : on liste explicitement les combinaisons autorisées. Exemple pour $x_1, x_2 \in \{1, 2, 3\}$:

$$C = \{(1, 2), (2, 3), (3, 1)\}$$

  signifie que seules ces paires de valeurs sont autorisées pour x_1 et x_2.
- **En intention** : on définit une relation logique ou mathématique. Exemple :

$$x_1 + x_2 = x_3$$

  impose que la somme de x_1 et x_2 soit égale à x_3.
- **Contraintes globales** : elles portent sur un grand nombre de variables et facilitent la propagation des contraintes. Exemple :

$$\texttt{allDifferent}(x_1, x_2, ..., x_n)$$

  impose que toutes les variables aient des valeurs distinctes.

Exemple de CSP

Considérons un problème de coloriage de carte où nous devons colorier trois régions A, B, et C avec trois couleurs différentes (Rouge, Vert, Bleu), en respectant la contrainte suivante :

— Deux régions adjacentes ne peuvent pas avoir la même couleur.

Formulation du CSP :

— **Variables** : $X = \{A, B, C\}$

— **Domaines** : $D(A) = D(B) = D(C) = \{Rouge, Vert, Bleu\}$

— **Contraintes** : $A \neq B$, $B \neq C$, $A \neq C$

La résolution consistera à attribuer une couleur à chaque région tout en respectant les contraintes définies.

Chaque contrainte restreint l'espace des solutions, et la solution finale du CSP est une affectation des variables satisfaisant **toutes** les contraintes simultanément.

4 Modélisation PPC : Stratégies de Modélisation

La modélisation est une étape cruciale en programmation par contraintes (PPC), car elle détermine l'efficacité de la résolution du problème. Une bonne modélisation repose sur plusieurs stratégies :

4.1 Identification des Variables et Domaines

La première étape consiste à définir les variables et leurs domaines :

— **Quels sont les éléments clés de la solution ?** Il s'agit des entités du problème que nous devons représenter sous forme de variables.

— **Quelles valeurs ces éléments peuvent-ils prendre ?** Chaque variable doit être associée à un domaine fini de valeurs possibles.

4.2 Définition des Contraintes

Une fois les variables définies, il faut établir les relations entre elles sous forme de contraintes :

— **Quelles sont les relations entre les variables ?** Par exemple, certaines variables peuvent être égales, différentes, ou respecter une relation arithmétique.

— **Exprimer les contraintes sous une forme mathématique ou logique.** Exemple : $x_1 + x_2 \leq 10$, ou encore $\texttt{allDifferent}(x_1, x_2, x_3)$.

4.3 Choix du Niveau de Granularité

Il est important d'adapter la modélisation en fonction du problème :

— **Une variable représente-t-elle un sous-problème ou une valeur individuelle ?** Parfois, une seule variable peut représenter un ensemble d'éléments.
— **Faut-il regrouper certaines variables pour simplifier le modèle ?** Une bonne structuration des variables peut améliorer l'efficacité de la résolution.

4.4 Optimisation de la Modélisation

Une modélisation efficace vise à réduire la complexité du problème :
— **Éviter les redondances.** Ne pas définir de contraintes inutiles qui alourdiraient le modèle.
— **Réduire la complexité en limitant le nombre de contraintes.** Une simplification du modèle améliore la vitesse de recherche de solutions.
— **Penser à la propagation des contraintes pour guider la recherche.** Une bonne propagation permet d'éliminer rapidement les valeurs incohérentes et d'accélérer la résolution du CSP.

5 Espace de Recherche en PPC

L'espace de recherche d'un problème de satisfaction de contraintes (CSP) représente l'ensemble des affectations possibles des variables du problème. Il joue un rôle crucial dans l'efficacité de la résolution et peut être optimisé à l'aide de différentes techniques.

5.1 Définition de l'Espace de Recherche

L'espace de recherche est défini comme le produit cartésien des domaines des variables :

$$E = D(x_1) \times D(x_2) \times ... \times D(x_n)$$

Ainsi, si chaque variable possède un domaine de d valeurs et qu'il y a n variables, alors la taille de l'espace de recherche est d^n.

5.2 Réduction de l'Espace de Recherche

L'espace de recherche brut peut être immense, mais il est réduit par les contraintes qui filtrent les affectations non valides. Plusieurs techniques permettent d'optimiser cette réduction :
— **Propagation de contraintes** : élimination des valeurs impossibles avant d'explorer l'espace de recherche.
— **Heuristiques de choix de variable et de valeur** : stratégies pour explorer efficacement l'espace de recherche.
— **Décomposition du problème** : division en sous-problèmes plus petits pour restreindre l'espace de recherche.

— **Recherche locale** : exploration partielle de l'espace pour trouver rapidement des solutions approximatives.

5.3 Impact des Contraintes sur l'Espace de Recherche

Les contraintes influencent directement la taille et la structure de l'espace de recherche :

— **Contraintes strictes** : réduisent fortement l'espace de recherche, facilitant la résolution mais risquant d'éliminer des solutions viables.
— **Contraintes souples** : laissent plus d'options ouvertes mais rendent la recherche plus complexe.
— **Contraintes globales** : permettent d'exprimer des relations complexes en une seule contrainte et améliorent l'efficacité de la résolution.

Une bonne gestion de l'espace de recherche est essentielle pour résoudre efficacement un CSP, en évitant une exploration exhaustive et en favorisant des techniques de recherche intelligentes.

6 Cassure de Symétrie en PPC

La cassure de symétrie est une technique essentielle en PPC permettant de réduire l'espace de recherche en éliminant des solutions équivalentes. Une symétrie apparaît lorsqu'il existe plusieurs solutions identiques modulo un simple réarrangement des valeurs.

6.1 Pourquoi Briser les Symétries ?

Les problèmes symétriques possèdent plusieurs solutions équivalentes, ce qui peut ralentir la recherche en explorant inutilement des configurations redondantes. En ajoutant des contraintes supplémentaires pour éliminer ces duplications, on réduit le temps de calcul et on améliore l'efficacité des solveurs.

6.2 Méthodes de Cassure de Symétrie

Différentes approches existent pour gérer les symétries :

— **Ajout de contraintes de symétrie** : on impose des restrictions pour forcer une seule représentation parmi les solutions symétriques. Exemple : fixer une variable pivot ($x_1 < x_2$) pour éviter des permutations inutiles.
— **Réduction de l'espace de recherche** : on adapte les heuristiques de recherche pour éviter d'explorer plusieurs fois les mêmes configurations.
— **Filtrage dynamique** : suppression des solutions symétriques au cours de la recherche grâce à des algorithmes spécialisés.

6.3 Exemple de Cassure de Symétrie

Prenons l'exemple du problème des n-reines, où l'on cherche à placer n reines sur un échiquier de taille $n \times n$ sans qu'aucune ne s'attaque. Ce problème possède plusieurs solutions symétriques par rotation ou réflexion. On peut briser ces symétries en imposant que la première reine soit placée dans la moitié gauche de l'échiquier :

Exemple de Cassure de Symétrie

Problème des n-reines : Contraintes de cassure de symétrie appliquées :
- Fixer la première reine sur la première ligne.
- Exiger que la colonne de la première reine soit inférieure à $n/2$.
- Éliminer les solutions obtenues par rotation ou réflexion.

L'application de ces techniques permet de réduire le nombre de configurations explorées et d'accélérer la résolution du CSP.

7 Contraintes Redondantes en PPC

Les contraintes redondantes sont des contraintes ajoutées à un CSP qui ne modifient pas l'ensemble des solutions valides mais permettent d'accélérer la résolution en renforçant la propagation.

7.1 Pourquoi Ajouter des Contraintes Redondantes ?

Bien qu'une contrainte redondante ne change pas l'ensemble des solutions, elle peut :
- Améliorer la propagation des contraintes et réduire l'espace de recherche.
- Éliminer plus rapidement des valeurs incohérentes et éviter des explorations inutiles.
- Faciliter la convergence vers une solution en apportant des informations supplémentaires au solveur.

7.2 Exemples de Contraintes Redondantes

- **Somme des variables** : Dans un problème où l'on impose que x_1, x_2, x_3 prennent des valeurs différentes ($\texttt{allDifferent}(x_1, x_2, x_3)$), une contrainte redondante comme $x_1 + x_2 + x_3 = 6$ peut améliorer la propagation.
- **Symétrie implicite** : Dans un problème de placement d'objets sur une grille, une contrainte imposant que la somme des indices des objets soit pair peut accélérer la recherche sans modifier l'ensemble des solutions.
- **Optimisation de la propagation** : Dans un problème de planification, ajouter des contraintes de précédence supplémentaires peut limiter les

affectations explorées par le solveur.

L'ajout de contraintes redondantes doit être fait avec précaution, car un trop grand nombre de contraintes peut ralentir la résolution au lieu de l'accélérer. Une bonne analyse du problème permet d'identifier les contraintes les plus pertinentes à ajouter.

8 Contraintes de Liaison (Channeling Constraints)

Les contraintes de liaison, aussi appelées *channeling constraints*, permettent d'introduire des variables auxiliaires pour faciliter la modélisation d'un problème en établissant une correspondance entre différentes représentations du même concept.

8.1 Pourquoi Utiliser des Contraintes de Liaison ?

L'ajout de variables auxiliaires et de contraintes de liaison est utile pour :
- Faciliter l'expression de certaines contraintes complexes.
- Améliorer la propagation des contraintes et accélérer la recherche de solutions.
- Offrir une alternative équivalente qui simplifie la résolution du problème.

8.2 Types de Contraintes de Liaison

Les contraintes de liaison sont souvent utilisées pour relier différentes formulations d'un problème :
- **Correspondance entre variables booléennes et entières** : On peut introduire une variable booléenne b_i pour indiquer si une variable entière x_i prend une valeur donnée.

$$b_i = 1 \iff x_i = v$$

- **Représentation alternative d'une contrainte** : On peut définir deux ensembles de variables représentant la même information de manière différente, avec des contraintes de liaison assurant leur cohérence.
- **Décomposition d'une contrainte globale** : Une contrainte complexe peut être transformée en plusieurs contraintes locales liées par des variables auxiliaires.

L'utilisation de contraintes de liaison est une technique puissante pour structurer et simplifier un problème de satisfaction de contraintes tout en optimisant sa résolution.

9 Contraintes Globales en Modélisation

Les contraintes globales sont des contraintes prédéfinies qui capturent des structures fréquentes dans les CSP. Elles permettent de modéliser efficacement des relations complexes en évitant d'énumérer toutes les contraintes élémentaires équivalentes.

9.1 Pourquoi Utiliser des Contraintes Globales ?

Les contraintes globales offrent plusieurs avantages en modélisation :

— Elles permettent d'exprimer un problème de manière plus compacte et lisible.

— Elles améliorent la propagation des contraintes, réduisant ainsi l'espace de recherche.

— Elles évitent d'avoir à formuler des ensembles de contraintes équivalentes de manière explicite.

9.2 Principales Contraintes Globales pour la Modélisation

Voici quelques contraintes globales couramment utilisées en modélisation PPC :

— `allDifferent`$(x_1, x_2, ..., x_n)$: impose que toutes les variables aient des valeurs distinctes. Cette contrainte est utile pour les problèmes de planification et d'ordonnancement.

— `sum`$(x_1+x_2+...+x_n = S)$: impose une somme donnée sur un ensemble de variables. Elle est fréquemment utilisée dans les problèmes d'affectation et d'optimisation.

— `element`$(i, [v_1, v_2, ..., v_n], x)$: impose que la variable x prenne la valeur située à la position i d'une liste. Cette contrainte est utile pour modéliser des choix dépendant d'un indice.

— `circuit`$(x_1, x_2, ..., x_n)$: impose qu'un ensemble de variables définisse un circuit hamiltonien, couramment utilisé dans les problèmes de routage.

— `globalCardinality`$(x_1, x_2, ..., x_n, [a_1, ..., a_m], [b_1, ..., b_m])$: restreint la fréquence d'apparition des valeurs dans un ensemble de variables. Elle est souvent employée dans les problèmes de répartition équilibrée.

Les contraintes globales sont des outils puissants qui permettent de rendre la modélisation plus intuitive et d'améliorer les performances des solveurs en PPC.