Programmation par Contraintes (Résolution)

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 Introduction à la Résolution en Programmation par Contraintes (PPC)

La PPC est une approche déclarative de la résolution de problèmes combinatoires, dans laquelle un problème est modélisé sous forme de variables, de domaines (ensembles de valeurs possibles pour chaque variable) et de contraintes (relations qui doivent être respectées entre les variables). L'un des avantages majeurs de cette approche est que l'algorithme de résolution n'a pas besoin d'être explicitement défini ; il est plutôt basé sur des techniques automatisées qui explorent l'espace des solutions possibles.

La résolution d'un problème en PPC repose sur la capacité à trouver une ou plusieurs valeurs pour chaque variable qui respectent toutes les contraintes imposées. Cette recherche est effectuée à l'aide de solveurs qui utilisent des algorithmes spécialisés pour naviguer efficacement dans l'espace des solutions et optimiser la recherche de solutions valides.

2 Méthodes de Résolution

Les techniques de résolution en PPC varient en fonction de la nature du problème et des contraintes. Les principales méthodes sont :

— **Backtracking (Retour en arrière) :** Il s'agit de l'une des méthodes les plus simples mais efficaces pour résoudre des problèmes de PPC. L'idée est de construire la solution progressivement, en vérifiant à chaque étape si les contraintes sont respectées. Si une contrainte est violée, l'algorithme revient en arrière pour essayer une autre possibilité.
— **Propagation de contraintes :** Cette technique consiste à propager les effets des contraintes tout au long de l'espace de recherche pour réduire le domaine des variables et éliminer les solutions invalides de manière proactive.
— **Recherche locale :** Ces techniques, comme les algorithmes de descente ou les algorithmes génétiques, tentent de trouver des solutions en explorant le voisinage d'une solution donnée, en itérant localement et en cherchant à améliorer la solution à chaque étape.
— **Décomposition du problème :** Cette approche consiste à diviser un problème complexe en sous-problèmes plus simples, permettant une résolution plus rapide et plus ciblée.

3 Résolution par Backtracking

L'algorithme du backtracking (ou retour en arrière) est une méthode de recherche utilisée pour résoudre les problèmes combinatoires. Il consiste à explorer un espace de solutions de manière exhaustive et à revenir en arrière lorsqu'une solution partielle ne peut plus mener à une solution valide.

L'idée fondamentale est de construire une solution progressivement, étape par étape, en vérifiant à chaque étape si la solution partielle est valide par rapport aux contraintes du problème. Si une étape mène à une situation où les contraintes ne sont plus satisfaites, l'algorithme revient en arrière et essaie une autre possibilité. Le processus se poursuit jusqu'à ce qu'une solution valide soit trouvée ou que toutes les possibilités aient été explorées.

Algorithme 1 : Backtracking (BT)

```
Input : ⟨X, D, C⟩ (réseau de contraintes), I (instanciation partielle)
Output : true si une solution est trouvée, sinon false
if I est une instanciation complète then
    return true ;                    // Une solution a été trouvée
end
Sélectionner une variable X_i ∉ I;
foreach v ∈ D(X_i) do
    if I ∪ {X_i ← v} est localement consistant then
        if BT(⟨X, D, C⟩, I ∪ {X_i ← v}) then
            return true
        end
    end
end
return false ;                  // Aucune solution n'a été trouvée
```

3.1 Description de l'Algorithme

1. **Entrée** : L'algorithme prend en entrée un réseau de contraintes $\langle X, D, C \rangle$, où X est un ensemble de variables, D est l'ensemble des domaines des variables, et C est l'ensemble des contraintes. L'algorithme prend également une instanciation partielle I des variables, qui représente l'état actuel de la solution en construction.
2. **Cas de base** : Si l'instanciation I est complète (toutes les variables sont affectées), alors une solution a été trouvée, et l'algorithme retourne **true**.
3. **Sélection de variable** : Si l'instanciation est incomplète, l'algorithme sélectionne une variable non encore instanciée (non affectée) parmi l'ensemble des variables X.
4. **Parcours des valeurs** : L'algorithme parcourt toutes les valeurs possibles v dans le domaine $D(X_i)$ de la variable sélectionnée X_i.
5. **Vérification de la consistance locale** : Avant d'affecter une valeur à X_i, l'algorithme vérifie si l'instanciation $I \cup \{X_i \leftarrow v\}$ est localement consistante, c'est-à-dire si toutes les contraintes sont satisfaites sous cette affectation partielle.
6. **Appel récursif** : Si l'instanciation est localement consistante, l'algorithme appelle récursivement BT pour résoudre le problème avec cette nouvelle affectation. Si la récursion retourne **true**, cela signifie qu'une solution a été trouvée, et l'algorithme retourne **true**.
7. **Backtracking** : Si aucune valeur ne permet de mener à une solution, l'algorithme revient en arrière (backtrack) et essaie une autre valeur pour la variable précédente.
8. **Retour** : Si toutes les possibilités ont été explorées sans succès, l'algorithme retourne **false**, indiquant qu'aucune solution n'a été trouvée.

4 Les Consistances Locales en PPC

En PPC, la **consistance locale** désigne une propriété où une partie du réseau de contraintes est rendue cohérente, sans garantir que l'ensemble du problème soit globalement résolu. Différents niveaux de consistance locale existent, parmi lesquels la **consistance d'arc (AC)** et la **consistance de bornes (BC)**.

4.1 Consistance d'Arc (AC)

La **consistance d'arc (AC)** est une forme de consistance locale, plus forte que la BC. Un problème est arc-consistant si, pour chaque contrainte binaire $C(X_i, X_j)$ et chaque valeur $v_i \in D(X_i)$, il existe au moins une valeur $v_j \in D(X_j)$ telle que la contrainte soit satisfaite. Si ce n'est pas le cas, v_i est éliminée de $D(X_i)$.

L'algorithme *AC-3* est souvent utilisé pour établir la consistance d'arc en parcourant les contraintes et en supprimant les valeurs inconsistantes. Bien que plus coûteuse que la BC, l'AC permet une meilleure réduction de l'espace de recherche.

4.2 Consistance de Bornes (BC)

La **consistance de bornes (BC)** est une autre forme de consistance locale qui ne considère que les valeurs extrêmes des domaines des variables. Une contrainte binaire $C(X_i, X_j)$ est bornes-consistante si, pour les valeurs minimales et maximales de X_i, il existe au moins une valeur de X_j respectant la contrainte.

Contrairement à l'AC, qui vérifie toutes les valeurs d'un domaine, la BC se limite aux bornes, ce qui la rend plus rapide mais aussi moins efficace en termes de filtrage des valeurs inconsistantes.

Dans le cas des variables booléenes, la BC et l'AC deviennent **équivalentes**. En effet, si un domaine est limité à deux valeurs, vérifier la BC revient à examiner toutes les valeurs possibles, ce qui est exactement ce que fait l'AC.

4.3 Autres Niveaux de Consistance Locale

D'autres formes de consistance locale existent, notamment :

— **SAC (Singleton Arc Consistency)** : Vérifie que chaque valeur d'une variable peut être étendue en une solution partielle consistante.

— **RPC (Restricted Path Consistency)** : Renforce la consistance d'arc en prenant en compte des contraintes reliant indirectement trois variables.

— **k-consistency** : Un problème est **k-consistant** si toute instanciation valide de $k-1$ variables peut être étendue à une k-ième variable en respectant les contraintes.

4.4 Consistance Globale et Consistance de Domaine

La **consistance globale** signifie que l'ensemble du problème est rendu localement consistant à tous les niveaux de propagation, maximisant ainsi la réduction de l'espace de recherche.

La **consistance de domaine** est une propriété plus faible garantissant que chaque valeur d'un domaine respecte au moins une contrainte du problème.

4.5 Contraintes Globales et Propagateurs

Les **contraintes globales** portent sur plusieurs variables et permettent de modéliser des relations complexes. Par exemple, la contrainte `AllDifferent` impose que toutes les variables aient des valeurs distinctes.

Pour maintenir un niveau de consistance sur ces contraintes, les solveurs utilisent des **propagateurs**, qui appliquent des algorithmes spécialisés pour

garantir une certaine consistance locale (AC, BC ou autres). Ces techniques améliorent l'efficacité du filtrage et réduisent significativement le temps de recherche.

5 Les Stratégies

Une fois que le problème a été modélisé, il est nécessaire de choisir des stratégies efficaces pour explorer l'espace des solutions. Parmi les stratégies les plus couramment utilisées en PPC, on retrouve :

- **Heuristiques de Choix de Variable et de Valeur :** Les **heuristiques** sont des stratégies utilisées pour guider la recherche dans l'espace de recherche en PPC. Elles permettent de prendre des décisions plus éclairées concernant l'ordre de sélection des variables et les valeurs à attribuer à ces variables. L'objectif est de réduire le temps de calcul et d'augmenter les chances de trouver rapidement une solution en privilégiant les choix les plus prometteurs.

  1. **Heuristiques de Choix de Variable :** Le choix de la variable à affecter est crucial dans la résolution des problèmes de PPC. Une mauvaise sélection peut mener à une exploration inefficace de l'espace de recherche. Voici quelques heuristiques courantes pour le choix de la variable : **Domaine minimal (Min Dom)** : Cette heuristique consiste à choisir la variable ayant le plus petit domaine ; **Plus de contraintes (Degree Heuristic)** : privilégie la variable qui est impliquée dans le plus grand nombre de contraintes.
  2. **Heuristiques de Choix de Valeur :** Une fois une variable sélectionnée, l'heuristique suivante consiste à choisir la valeur à lui attribuer. Comme pour le choix de la variable, le bon choix de valeur peut grandement influencer l'efficacité de la recherche. Voici quelques heuristiques de choix de valeur : **Valeur la plus contraignante (Most Constrained Value)** : Cette heuristique consiste à choisir la valeur qui est la plus contraignante pour les autres variables, c'est-à-dire la valeur qui réduit le plus le domaine des variables voisines ; **Valeur aléatoire (Random Value)** : Cette heuristique choisit une valeur de manière aléatoire dans le domaine de la variable.

- **No-good :** Les *no-good* sont des ensembles de décisions menant inévitablement à un conflit ou à une contradiction. Lorsqu'un *no-good* est détecté, on peut immédiatement élaguer toutes les branches de l'espace de recherche contenant cette configuration. Par exemple, dans un problème de Sudoku, un *no-good* pourrait être une combinaison de valeurs violant la contrainte d'unicité des chiffres dans une ligne.
- **Détection de Conflit :** La détection de conflit identifie les violations de contraintes, générant ainsi des *no-good* pour éviter de revisiter ces configurations. Dès qu'un conflit est détecté, l'algorithme revient en arrière et utilise les *no-good* pour éliminer les solutions conflictuelles lors

des itérations futures.

— **Consistance :** L'application de différents niveaux de consistance, comme la consistance de domaine ou l'arc-consistance, permet de réduire le domaine des variables en éliminant les valeurs invalides, ce qui réduit le nombre de possibilités à explorer.