Programmation par contraintes pour les technologies logicielles

Narendra Jussien
École des Mines de Nantes
4, rue Alfred Kastler – BP 20722
F-44307 Nantes Cedex 3
email: `jussien@emn.fr`

Résumé

La programmation par contraintes, discipline au carrefour de l'intelligence artificielle, de la recherche opérationnelle et de l'analyse numérique a fait ses preuves pour la résolution de problèmes combinatoires complexes dans le domaine de l'aide à la décision. De nouveaux champs d'applications apparaissent, en particulier dans le domaine du génie logiciel. Au travers de quelques exemples (notamment l'identification et la correction de motifs de conception approchés et la réalisation d'applications dans un contexte anytime), nous montrons les apports et les perspectives de l'utilisation de la programmation par contraintes dans le domaine du génie logiciel.

1 Introduction

La programmation par contraintes, discipline au carrefour de l'intelligence artificielle, de la recherche opérationnelle et de l'analyse numérique, a maintenant fait ses preuves pour la résolution de problèmes combinatoires complexes dans le domaine de l'aide à la décision. Grâce à des propositions récentes d'extension, de nouveaux champs d'applications apparaissent.

Dans cet article, nous présentons la programmation par contraintes et quelques unes de ses extensions et montrons au travers de deux exemples leur application au domaine du génie logiciel.

2 Programmation par contraintes

La programmation par contraintes est un sujet de recherche au carrefour des travaux de divers domaines comme les mathématiques discrètes, l'analyse numérique, l'intelligence artificielle, la programmation mathématique (ou plus généralement la recherche opérationnelle) ou le calcul formel.

La communauté *contraintes* peut s'enorgueillir de réussites marquantes dans des domaines d'applications réellement diversifiés : les problèmes combinatoires, l'ordonnancement, l'analyse financière, la simulation et la synthèse de circuits intégrés, le diagnostic de pannes, l'aide à la décision, la biologie moléculaire, la résolution de problèmes géométriques, ...

La programmation par contraintes est actuellement en plein essor industriel grâce, en particulier, à deux acteurs majeurs : les sociétés ILOG et COSYTEC qui ont développé, par exemple, des logiciels d'optimisation pour la gestion du trafic aérien, utilisés par de nombreux aéroports, des logiciels d'emploi du temps et de rotation de personnels,...

2.1 Analyse d'un succès

Ces résultats positifs ont été obtenus pour différentes raisons :
- l'hypothèse de base de la programmation par contraintes, la recherche de solutions dans un domaine borné, est souvent naturellement vérifiée dans la pratique où les inconnues correspondent à des grandeurs physiques ;
- les outils et méthodes sont d'application très générale, ils ne se restreignent pas à une catégorie particulière de problème ;
- par principe même, les méthodes sont très appropriées à une implémentation distribuée, ce qui permet d'en améliorer sensiblement l'efficacité ;
- les méthodes viennent compléter d'autres approches plus sensibles au nombre d'inconnues ou à la structure des équations, comme les systèmes algébriques.

2.2 Le cadre général

Un problème de satisfaction de contraintes (CSP) [Tsa93] est représenté par un triplet $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ où $\mathcal{V}$ est un ensemble de variables prenant leur valeur dans un domaine précisé pour chaque variable (leur ensemble est $\mathcal{D}$) et sur lesquelles portent des contraintes $\mathcal{C}$. Une solution pour CSP est une affectation complète des variables (c.-à-d. l'attribution d'une valeur unique à chaque variable) telle que toutes les contraintes soient vérifiées.

La résolution d'un CSP met en œuvre de manière entrelacée des étapes de filtrage (réduction sophistiquée des domaines des variables pour éliminer les portions de l'espace de recherche ne pouvant faire partie d'une solution) et des étapes d'énumération (prise de décision pour se rapprocher d'une solution). Un solveur de contrainte est un système programmable qui réalise le contrôle de cet entrelacement en faisant appel à divers composants de filtrage (les contraintes elles-mêmes) et de contrôle de l'énumération. Les travaux de la communauté *contraintes* portent sur ces deux étapes cruciales et sur les techniques permettant de les améliorer.

3 Extensions de la programmation par contraintes

Longtemps spécialisée dans la résolution de problèmes combinatoires complexes dans le domaine de l'aide à la décision, la programmation par contraintes s'ouvre maintenant à de nouveaux horizons. Des extensions à la programmation par contraintes classique existent maintenant pour prendre en compte de nouvelles problématiques.

3.1 Explications et analyse de programmes avec contraintes

L'introduction de la notion d'explication dans le contexte de la programmation par contraintes est relativement récente [Jus01]. Il s'agit en fait de conserver une trace limitée et lisible de l'activité passée du solveur de contraintes. L'intégration d'explications dans un solveur de contraintes permet, entre autres :

- d'unifier une grande majorité des techniques existantes dans un cadre générique permettant de caractériser simplement ces méthodes afin de faciliter le choix d'une technique donnée pour un problème donné ;
- de développer des algorithmes capables de gérer dynamiquement les modifications du problème avant, pendant ou après sa résolution avec une certaine stabilité dans les résultats obtenus ;
- de rendre compte à l'utilisateur, de manière compréhensible et lisible, de l'activité du solveur de contraintes.

Cette extension définit un nouveau paradigme de programmation par contraintes.

3.2 Préférences pour les problèmes avec contraintes

Le cadre des CSP valués (VCSP) [BMR+99] forme une extension des CSP permettant de manipuler des problèmes sur-contraints ou pour lesquels des pondérations sont spécifiées entre solutions. Un VCSP peut être défini comme un CSP $(\mathcal{V}, \mathcal{D}, \mathcal{C})$ auquel on adjoint une structure de valuation $\mathcal{S}$ et une fonction de valuation φ. La structure de valuation $\mathcal{S}$ est un triplet $(E, \succ, \otimes)$, où E est un ensemble de valuations, $\succ$ un ordre total sur E (avec $\top$ et $\bot$ les élément maximaux et minimaux de E) et $\otimes$ un opérateur binaire clos sur E. La fonction de valuation φ, définie de $\mathcal{C}$ vers E, associe une valuation à chaque contrainte dénotant son importance relative.

Soit $\mathcal{A}$ une affectation de toutes les variables et $\mathcal{C}_{unsat}(\mathcal{A})$ l'ensemble des contraintes non satisfaites par $\mathcal{A}$. La valuation de $\mathcal{A}$ est l'agrégation des valuations de toutes les contraintes de $\mathcal{C}_{unsat}(\mathcal{A})$: $\varphi(\mathcal{A}) = \otimes\, \varphi(c)$ pour $c \in \mathcal{C}_{unsat}(\mathcal{A})$.

Résoudre un VCSP revient à trouver une affectation complète des variables avec une valuation minimale. Des techniques spécifiques sont utilisées en fonction de l'opérateur d'agrégation retenu : $\wedge$ pour les CSP classique, *max* pour les CSP *possibilistes*, $\cup$ sur des multi-ensembles pour les CSP *lexicographiques*, Σ pour les CSP *additifs*. Cette dernière catégorie est la plus souvent étudiée. Il s'agit de plus de la catégorie la plus rétive algorithmiquement parlant.

3.3 Approches hybrides

L'extension de la programmation par contraintes aux algorithmes hybrides emprunte les mécanismes à la fois des recherches de type arborescent et des méthodes approchées de type recherches locales.

Les méthodes arborescentes sont utilisées pour produire des solutions optimales et donner une preuve d'optimalité. Mais, du fait de leur comportement potentiellement exponentiel, elles peuvent être très coûteuses en temps de calcul. De plus, de telles méthodes ne sont souvent pas capables de fournir rapidement une bonne solution. Au contraire, les méthodes approchées (comme le recuit simulé ou les recherches taboues) peuvent produire rapidement de très bonnes solutions grâce à leur façon opportuniste d'explorer l'espace

de recherche. Mais, ces méthodes ne peuvent pas toujours se sortir facilement d'optima locaux et peuvent perdre beaucoup de temps à explorer un voisinage inintéressant.

Les algorithmes hybrides sont un compromis entre les deux types d'approches. Ils combinent efficacement les avantages de la propagation de contraintes fournis par les méthodes arborescentes et de l'exploration opportuniste des recherches locales. Ces techniques mettant harmonieusement en œuvre deux ou plusieurs approches différentes pour résoudre un même problème donnent de bons résultats.

4 Une première application : résolution de problèmes en ligne

Il est indiscutable d'affirmer aujourd'hui que la programmation par contraintes est capable de satisfaire à la demande de l'industrie pour résoudre des problèmes hors-ligne (problème fixé et temps de résolution libre). Dans un contexte d'optimisation *en ligne*[1], les problèmes évoluent dynamiquement suivant les modifications de l'environnement externe et leur résolution doit respecter des contraintes temporelles fortes.

Même si la programmation par contraintes classique ne semble pas en mesure de répondre à ce type de demande, l'utilisation de ses extensions permet de fournir un algorithme utilisable dans un contexte où il peut être interrompu à tout moment et garantissant les propriétés suivantes : production rapide de solutions de bonne qualité et améliorations graduelles de ces solutions compte-tenu du temps qui lui a été attribué (c'est un contexte dit *anytime*). Il s'agit ainsi d'obtenir un bon profil de performance pour un tel algorithme interruptible.

Dans ce contexte, l'utilisation d'un VCSP (pour modéliser la notion de *qualité* d'une solution partielle) et d'un algorithme hybride paraît tout à fait appropriée. Par exemple, l'algorithme VNS/LDS+CP [LB01] est un algorithme hybride récent qui sur une base de recherche locale utilise des voisinages de tailles variables dont l'exploration est assurée par une méthode efficace de type LDS [HG95] (une exploration partielle mais bien répartie de l'espace de recherche) aidée par la propagation de contraintes pour résoudre des VCSP.

Son utilisation sur des instances réelles de problèmes d'affectation de fréquence radio [CdGL$^+$99] montre qu'une telle hybridation est une réponse efficace pour résoudre un problème dans un contexte anytime.

5 Une seconde application : le système PTIDEJ

L'extension de la programmation par contraintes aux explications a été utilisée pour produire le système PTIDEJ (Pattern Trace Identification, Detection and Enhancement in Java [AACGJ01]). PTIDEJ est un système automatique pour l'identification de micro-architectures similaires à des patrons de conception dans du code source orienté objet. Une micro-architecture décrit la structure d'un sous-ensemble des classes d'un programme orienté objets.

La production de code source de qualité est un enjeu important pour l'industrie du logiciel. Un code source de qualité facilite l'évolution et la maintenance : ajout de

[1] L'École des Mines de Nantes est partenaire du projet RNRT EOLE (Environnement d'Optimisation en LignE) avec Thalès RT, Bouygues, l'ONERA et Cosytec.

nouvelles fonctionnalités, correction de bogues, adaptation à de nouvelles plates-formes d'exécution, intégration dans des bibliothèques de classes, ... En programmation par objets, un code source de qualité se distingue par deux aspects : des algorithmes efficaces et énoncés clairement, respectant des conventions et des idiomes d'écriture, et une architecture de classes *élégante*. Les patrons de conception (*design patterns*) [GHJV94] sont des exemples de micro-architectures élégantes qui résument l'expérience de développeurs expérimentés.

Néanmoins, il n'est pas aisé d'écrire directement du code source qui respecte les patrons de conception. C'est pourquoi les patrons de conception sont souvent présents dans le code source sous des formes approchées (c.-à-d. des micro-architectures similaires mais non parfaitement identiques à celles proposées par les patrons de conception). Il n'existe pas d'outils pour retrouver dans du code source ces formes approchées de patrons de conception pour en proposer une amélioration à l'utilisateur – pour les rendre conformes aux micro-architectures élégantes proposées par les patrons de conception.

PTIDEJ est un tel outil. Son but est d'identifier des micro-architectures dans du code source. L'idée de PTIDEJ est de définir un problème de satisfaction de contraintes dont la structure (les variables et les contraintes) représente le patron de conception recherché et dont la sémantique (les domaines des variables et la sémantique des contraintes) est déduite du code source objet. La résolution de ce problème permet de découvrir les micro-architectures correspondant exactement au patron de conception cherché. Pour trouver les formes approchées, l'utilisation d'un solveur de contraintes avec explications permet de déterminer précisément pourquoi la structure exacte n'a pas été trouvée et donc de guider l'utilisateur vers les contraintes incriminées.

L'intérêt de PTIDEJ, mis à part le fait qu'il s'agisse véritablement du premier outil capable d'identifier des micro-architectures approchées [GJ01], est sa capacité à expliquer ses réponses. Cette capacité est intéressante dans la mesure où le codage et le développement d'applications sont souvent considérés comme un art (alors que la tendance est vers l'industrialisation du logiciel) et où les systèmes complètement automatiques ne sont pas toujours bien reçus par les utilisateurs potentiels de tels systèmes : les programmeurs.

6 Conclusion

Nous avons présenté rapidement la programmation par contraintes et trois de ces extensions. L'intérêt de la programmation par contraintes en se limite pas, loin s'en faut, à la résolution de problèmes d'optimisation combinatoire dans le domaine de l'aide à la décision où elle a largement fait ses preuves. En effet, les deux applications décrites de ses extensions (un système automatique d'identification de micro-architectures et un outil de résolution de problèmes en ligne) montrent que la programmation par contraintes a son mot à dire dans le domaine du génie logiciel.

Références

[AACGJ01] Hervé Albin-Amiot, Pierre Cointe, Yann-Gaël Guéhéneuc, and Narendra Jussien. Instantiating and detecting design patterns : Putting bits and pieces together. In *16th IEEE conference on Automated Software Engineering (ASE'01)*, pages 166–173, San Diego, USA, November 2001. IEEE Computer Society Press.

[BMR+99] S. Bistarelli, U. Montanari, F. Rossi, T. Schiex, and G. Verfaillie H. Fargier. Semiring-based csps and valued csps : Frameworks, properties, and comparison. *Constraints*, 4(3) :199–240, 1999.

[CdGL+99] B. Cabon, S. de Givry, L. Lobjois, T. Schiex, and J.P. Warners.W.K. Radio link frequency assignment. *Constraints*, 4(1) :79–89, 1999.

[GHJV94] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[GJ01] Yann-Gaël Guéhéneuc and Narendra Jussien. Using explanations for design-patterns identification. In *IJCAI'01 Workshop on Modelling and Solving problems with constraints*, pages 57–64, Seattle, WA, USA, August 2001.

[HG95] William D. Harvey and Matthew L. Ginsberg. Limited discrepancy search. In *Proceedings of IJCAI'95*, Montreal, August 1995.

[Jus01] Narendra Jussien. Programmation par contraintes avec explications. In *7ièmes Journées nationales sur la résolution pratique de problèmes NP-complets (JNPC'01)*, pages 147–158, June 2001.

[LB01] Samir Loudni and Patrice Boizumault. A new hybrid method for solving constraint optimization problems in anytime contexts. In *Proceedings of the Thirteenth IEEE International Conference on Tools with Artificial Intelligence (ICTAI'2001)*, pages 325–332, Dallas, USA, November 2001. IEEE Computer Society.

[Tsa93] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.