Search is not Dead!

Christophe lecoutre

University of Artois & CNRS, CRIL

lecoutre@cril.fr

Résumé

La tendance actuelle de la communauté de programmation par contraintes (PPC) à utiliser largement la technologie SAT (c'est-à-dire la traduction des contraintes et/ou le raisonnement à partir des clauses) est intéressante (et s'est avérée efficace dans bon nombre de situations), mais elle éclipse quelque peu l'intérêt de la PPC. Si l'on considère les résultats des dernières compétitions XCSP3 (2022, 2023, 2024 ; par exemple, voir www.cril.fr/XCSP24), on peut observer qu'un solveur PPC "pur" comme ACE peut être comparativement très efficace pour trouver des bornes de bonne qualité. En fait, notre position est que "search is not dead", une référence à l'exposé invité à la conférence CP'13 par Peter Stuckey qui affirmait que "search is dead, long live proof". Il nous semble que se concentrer uniquement sur la preuve dès le début (du processus de résolution) n'est pas nécessairement la bonne approche, en particulier avec les nouveaux progrès réalisés pour conduire la recherche de solutions (notamment, le solution-based phase saving, et les trois heuristiques complémentaires actuelles basées sur les conflits).

Mots-clés

recherche arborescente, heuristiques, contraintes

Abstract

The current trend in the CP (Constraint Programming) community to widely use SAT technology (i.e., translating constraints and/or reasoning from clauses) is interesting (and has been shown to be effective in a number of situations) but somewhat detracts from the value of CP. Considering the results of the last (2022, 2023, 2024) XCSP3 competitions (e.g., see www.cril.fr/XCSP24), one can observe that a pure CP solver like ACE can be comparatively efficient for finding good quality bounds. Actually, our position is that "search is reborn", a reference to the invited talk at the conference CP'13 by Peter Stuckey who claimed that "search is dead, long live proof". It seems to us that focusing only on proof from the start (of the solving process) is not necessarily the right approach, especially with the new advances made in search (notably, solution-based phase saving, and the three current complementary state-of-the art conflict-based heuristics).

Keywords

tree search, heuristics, constraints

1 Introduction

La programmation par contraintes (PPC) [18, 12, 11] est une technologie utile pour modéliser et résoudre des problèmes combinatoires sous contraintes. D'une part, on peut utiliser une bibliothèque comme PyCSP^3 [16] pour modéliser les problèmes qui se posent dans divers domaines d'application (par exemple, l'ordonnancement, la planification, la cryptographie, la bio-informatique, la chimie organique, etc.) Les instances de problèmes peuvent alors être directement générées à partir de modèles et de données particulières. D'autre part, pour résoudre les instances (notamment représentées au format XCSP^3 [6, 7]), on peut utiliser un solveur de contraintes comme ACE, dont il est question dans le présent document. ACE est un solveur de contraintes open-source, développé en Java, qui se focalise sur les variables entières (incluant les variables booléennes ou 0/1), différentes forme de contraintes table, les contraintes globales les plus importantes, les heuristiques de recherche état-de-l'art et l'optimisation (mono-critère).

2 Vers une recherche robuste en PPC

ACE est un solveur complet, qui effectue une exploration en profondeur de l'espace de recherche, avec retours-arrières. À chaque étape (nœud de l'arbre de recherche), une décision est prise (une affectation de variable ou une réfutation de valeur) et un processus de filtrage est exécuté (mécanisme de propagation de contraintes). Pour résoudre le problème de la forte variabilité des distributions de temps d'exécution [9], la recherche est relancée régulièrement, en suivant une progression géométrique. L'ordre dans lequel les variables sont choisies au cours de la construction de la branche courante de l'arbre de recherche est déterminé par une heuristique de choix de variables ; une heuristique générique classique est `dom/wdeg` [5], combinée à un mécanisme simulant une certaine forme de sauts en arrière intelligents [15]. L'ordre dans lequel les valeurs sont choisies lors de l'affectation des variables est déterminé par une heuristique de choix de valeurs ; pour l'optimisation, il est fortement recommandé d'utiliser d'abord, si disponible, la valeur de la dernière solution trouvée [20, 8].

Pour l'optimisation (instances du problème COP, Constraint Optimization Problem), on utilise une stratégie basée sur la mise à jour de la borne chaque fois qu'une solution est trouvée ; il s'agit d'une sorte de stratégie de descente (apparentée au Branch and Bound), dont le

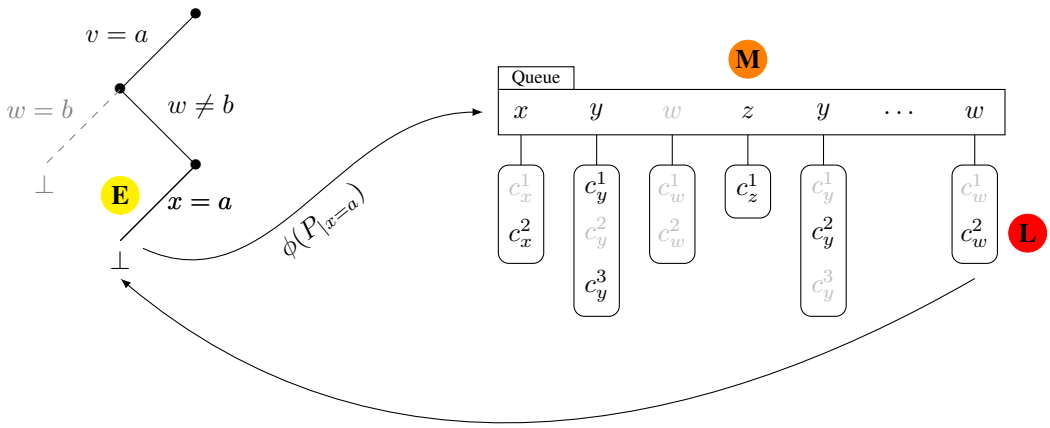

FIGURE 1 – Illustration de trois moments clés de la collecte d'informations concernant les conflits : ils correspondent au traitement précoce (Early), intermédiaire (Midway) et tardif (Late) des conflits.

principe est équivalent (dans l'hypothèse d'un problème de minimisation) à l'ajout d'une contrainte objectif spéciale $\texttt{obj} < \infty$ au réseau de contraintes (bien qu'elle soit initialement trivialement satisfaite), et à mettre à jour la limite de cette contrainte chaque fois qu'une nouvelle solution est trouvée. Cela signifie qu'à chaque fois qu'une solution S est trouvée avec un coût $B = \texttt{obj}(S)$, la contrainte objectif devient $\texttt{obj} < B$. Par conséquent, cette stratégie de descente progressive fournit une séquence de solutions de plus en plus intéressantes, jusqu'à prouver l'optimalité de la dernière solution trouvée.

Au moment de la rédaction de ce document, nous pensons que ACE est un solveur compétitif en raison des ingrédients suivants :

- redémarrage fréquent de la recherche
- enregistrement des nogoods de la branche courante au moment du redémarrage [14]
- pondération de contraintes pour la sélection des variables [5, 10, 21, 17, 4]
- raisonnement à partir du dernier conflit [15]
- solution(-based) phase saving [20, 8]

Fait important, dans [4], nous avons montré que trois manières d'exploiter les conflits pour guider la recherche présentent des comportements relativement complémentaires. Cela peut s'expliquer par le fait que les informations sont extraites à différents moments : au tout début du processus conduisant à un conflit (c'est-à-dire au moment de la décision), pendant la propagation de contraintes ou au moment où le dernier propagateur (algorithme de filtrage) est sollicité. On peut alors parler d'approches telles que le traitement opérationnel des conflits est *early* (E), *midway* (M) ou *late* (L). Ceci est illustré par la figure 1 où une nouvelle décision $x = a$ est prise, lors de la résolution d'un réseau de contraintes P, dans la continuité de deux décisions prises précédemment $v = a$ et $w \neq b$. Dans notre scénario, l'exécution de la propagation de contraintes ϕ sur (l'état actuel de) P après avoir assigné la valeur a à x, c'est-à-dire $\phi(P|_{x=a})$, révèle une nouvelle situation conflictuelle (désignée par $\bot$). Le traitement précoce de ce nouveau conflit consiste à considérer la variable x impliquée dans la décision comme cause principale. C'est le principe de l'heuristique `frba/dom` [17]. Le traitement intermédiaire de ce conflit consiste à considérer que toutes les variables ayant joué un rôle (c'est-à-dire ayant été choisies) au cours de la propagation ont contribué à l'échec. C'est le principe qui sous-tend l'heuristique `pick/dom` [4]. Le traitement tardif de ce conflit consiste à considérer la dernière contrainte (ici, c_w^2) provoquant l'échec (c'est-à-dire supprimant la dernière valeur d'un domaine) comme l'objet d'intérêt. C'est le principe de la pondération des contraintes, comme dans `wdeg/dom` [5, 10, 21].

L'intérêt (la complémentarité) de ces trois heuristiques est démontré lors de la compétition XCSP^3 2024 (voir [3] et www.cril.fr/XCSP24), à laquelle deux versions de ACE [13] ont participé :

- ACE, avec son comportement par défaut (une seule heuristique de choix de variables est utilisée : $\texttt{wdeg}^{cacd}$ [21] qui est un raffinement de `dom/wdeg`), et la première valeur (i.e., plus petite) est systématiquement choisie,
- ACE-mix, qui exploite les trois principales heuristiques de choix de variables mentionnées ci-dessus (ainsi que $\texttt{wdeg}^{cacd}$) et trois heuristiques de choix de valeurs (sélection de la valeur la plus petite, la plus grande, ou aléatoirement dans le domaine de la variable sélectionnée).

Il est clair que ACE-mix bénéficie de la complémentarité de ces heuristiques puisque cette version obtient de bien meilleurs scores que ACE dans les différents tracks de la compétition. Sans surprise, cela montre que la recherche devient (beaucoup) plus robuste lorsque l'on utilise des heuristiques complémentaires (ici, basées sur les conflits).

En fait, nous pensons que la tendance actuelle de la communauté PPC à utiliser largement la technologie SAT (c'est-à-dire traduire les contraintes en clauses et/ou raisonner à partir de clauses) est intéressante (et s'est avérée efficace dans

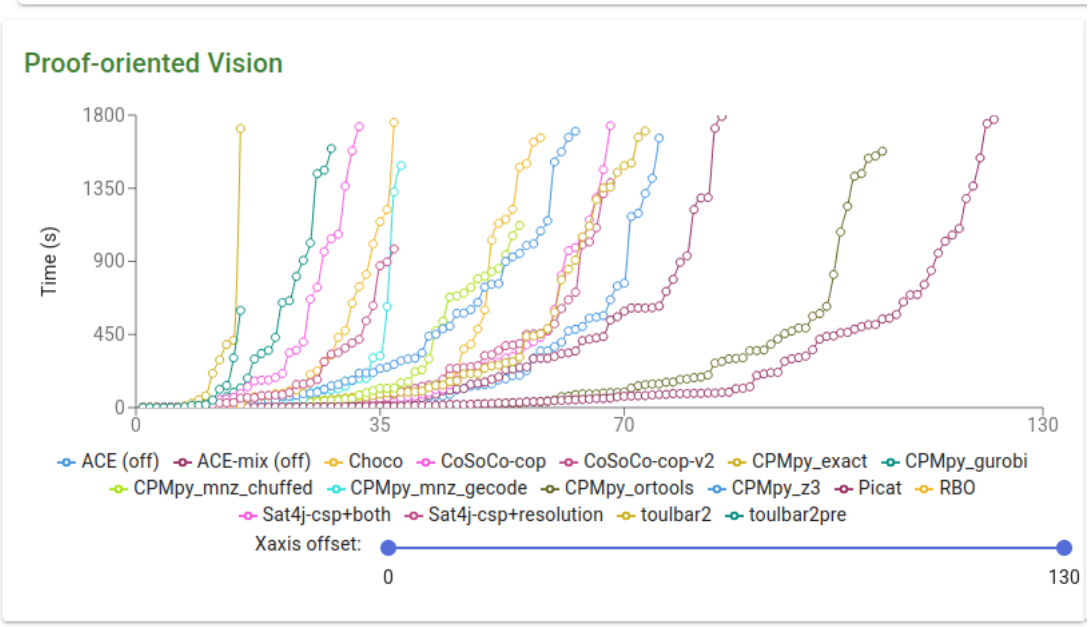

FIGURE 2 – Pour le track COP de la compétition XCSP^3 2024, avec une vision orientée preuve, Picat surpasse CPMpy_ortools, qui lui-même surpasse ACE-mix.

bon nombre de situations), mais qu'elle éclipse quelque peu la valeur de la PPC. En ce qui concerne les problèmes d'optimisation (main track COP), sur le site des résultats de la compétition XCSP^3 2024 (voir www.cril.fr/XCSP24), deux cactus plots sont visibles :

- un cactus plot, illustré à la figure 2, qui donne une vue "orientée preuve" des résultats (c'est à dire que seule les preuves d'optimalité comptent) : les approches basées sur SAT (et/ou sur la relaxation) sont très efficaces pour prouver l'optimalité ;
- un cactus plot, illustré à la figure 3, qui donne une vue "orientée recherche" (c'est à dire que les preuves d'optimalité sont ignorées) : les approches basée principalement sur la PPC sont très efficaces pour trouver des bornes de bonne qualité.

3 Search, Reborn !

Même des versions moins robustes de ACE, comme celles soumises en 2022 (voir [1] et www.cril.fr/XCSP22) et 2023 (voir [2] et ww.cril.fr/XCSP23), étaient déjà très compétitives pour le main track COP. La vision orienté recherche montre déjà comparativement les très bonnes performances de ACE. En fait, notre position est que *search is not dead* (ou *search is reborn*), en référence à la conférence invitée [19] à CP'13 par Peter Stuckey qui affirmait que "search is dead, long live proof". Comme la résolution d'un problème d'optimisation passe par trois étapes :

- la *phase d'accrochage* pendant laquelle une première solution doit être trouvée,
- la *phase de descente* pendant la quelle une solution optimale doit être trouvée, après un cheminement possiblement très long depuis la première solution,
- la *phase de preuve* pendant laquelle l'optimalité doit être prouvée,

il nous semble que se focaliser uniquement sur la preuve dès le départ n'est pas nécessairement la bonne approche, surtout avec les nouvelles avancées réalisées pour conduire la recherche (solution-based saving [20, 8] et les trois heuristiques complémentaires basées sur les conflits). Pour la phase de descente, la recherche telle qu'elle est effectuée par un solveur PPC classique peut, sous réserve d'utiliser des mécanismes de recherche complémentaires, devenir extrêmement compétitive (comme le montre ACE-mix).

Remerciements

Ce travail a bénéficié du soutien de l'ANR (France 2030), dans le cadre du projet MAIA (ANR-22-EXES-0009)

Références

[1] G. Audemard, C. Lecoutre, and E. Lonca. Proceedings of the 2022 XCSP3 competition. Technical Report arXiv :2209.00917, CoRR, 2022. https://arxiv.org/abs/2209.00917.

[2] G. Audemard, C. Lecoutre, and E. Lonca. Proceedings of the 2023 XCSP3 competition. Technical Report arXiv :2312.05877, CoRR, 2023. https://arxiv.org/abs/2312.05877.

[3] G. Audemard, C. Lecoutre, and E. Lonca. Proceedings of the 2024 XCSP3 competition. Technical Report arXiv :2412.00117, CoRR, 2024. https://arxiv.org/abs/2412.00117.

[4] G. Audemard, C. Lecoutre, and C. Prud'Homme. Guiding backtrack search by tracking variables during

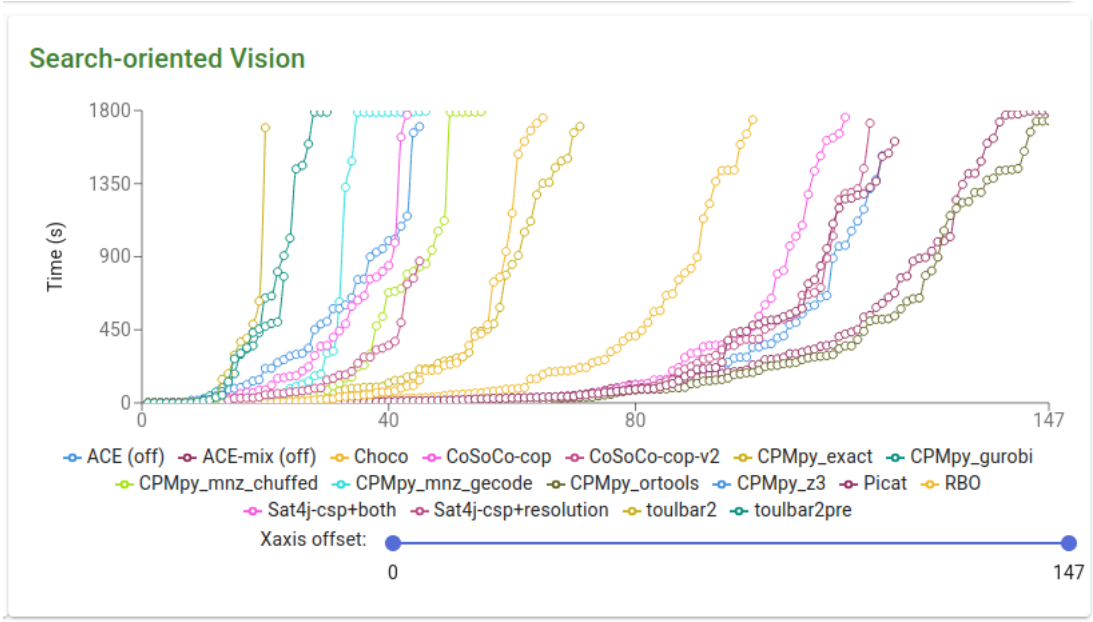

FIGURE 3 – Pour le track COP de la compétition XCSP^3 2024, avec une vision orientée recherche, ACE-mix et CPMpy_ortools sont assez proches et surpassent Picat.

constraint propagation. In *Proceedings of CP'23*, pages 9 :1–9–7, 2023.

[5] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. In *Proceedings of ECAI'04*, pages 146–150, 2004.

[6] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3 : an integrated format for benchmarking combinatorial constrained problems. Technical Report arXiv :1611.03398, CoRR, 2016. https://arxiv.org/abs/1611.03398.

[7] F. Boussemart, C. Lecoutre, G. Audemard, and C. Piette. XCSP3-core : A format for representing constraint satisfaction/optimization problems. Technical Report arXiv :2009.00514, CoRR, 2020. https://arxiv.org/abs/2009.00514.

[8] E. Demirovic, G. Chu, and P. Stuckey. Solution-based phase saving for CP : A value-selection heuristic to simulate local search behavior in complete solvers. In *Proceedings of CP'18*, pages 99–108, 2018.

[9] C. Gomes, B. Selman, N. Crato, and H. Kautz. Heavy-tailed phenomena in satisfiability and constraint satisfaction problems. *Journal of Automated Reasoning*, 24 :67–100, 2000.

[10] D. Habet and C. Terrioux. Conflict history based search for constraint satisfaction problem. In *Proceedings of SAC'19*, pages 1117–1122, 2019.

[11] D. E.. Knuth. *Art of Computer Programming, Volume 4, Fascicle 7 : Constraint Satisfaction*. Addison Wesley, March 2025.

[12] C. Lecoutre. *Constraint Networks : techniques and algorithms*. International Scientific and Technical Encyclopedia (ISTE Ltd) - John Wiley Inc., June 2009. 592 pages. ISBN : 9781848211063.

[13] C. Lecoutre. ACE, a generic constraint solver. Technical Report arXiv :2302.05405, CoRR, 2023. https://arxiv.org/abs/2302.05405.

[14] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Recording and minimizing nogoods from restarts. *Journal on Satisfiability, Boolean Modeling and Computation (JSAT)*, 1 :147–167, 2007.

[15] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Reasoning from last conflict(s) in constraint programming. *Artificial Intelligence*, 173(18) :1592–1614, 2009.

[16] C. Lecoutre and N. Szczepanski. PyCSP^3 : Modeling combinatorial constrained problems in Python. Technical Report arXiv :2009.00326, CoRR, 2020. https://arxiv.org/abs/2009.00326.

[17] Hongbo Li, Minghao Yin, and Zhanshan Li. Failure based variable ordering heuristics for solving CSPs. In *Proceedings of CP'21*, pages 9 :1–9 :10, 2021.

[18] F. Rossi, P. van Beek, and T. Walsh, editors. *Handbook of Constraint Programming*. Elsevier, 2006.

[19] P. Stuckey. Those who cannot remember the past are condemned to repeat it. In *CP'13*, pages 5–6, 2013. www.youtube.com/watch?v=lxiCHRFNgno.

[20] J. Vion and S. Piechowiak. Une simple heuristique pour rapprocher DFS et LNS pour les COP. In *Proceedings of JFPC'17*, pages 39–45, 2017.

[21] H. Wattez, C. Lecoutre, A. Paparrizou, and S. Tabary. Refining constraint weighting. In *Proceedings of ICTAI'19*, pages 71–77, 2019.