À propos de la bibliothèque de modèles XCSP

Arnaud Malapert[1] **Christophe Lecoutre**[2]

[1] Université Nice Sophia Antipolis, CNRS, I3S, UMR 7271, 06900 Sophia Antipolis, France
[2] Université d'artois, CNRS, CRIL, UMR 8188, rue de l'université, 62307 Lens cedex, France
arnaud.malapert@unice.fr lecoutre@cril.fr

Résumé

Les bibliothèques de modèles (modèles et instances) sont couramment utilisées pour évaluer les solveurs et les algorithmes en programmation par contraintes. Elles sont d'un intérêt certain pour l'expérimentation : vaste corpus de modèles, comparaison et reproductibilité. Toutefois, ces bibliothèques doivent être régulièrement examinées avec attention pour évaluer la pérennité de leur intérêt. Dans cet article, nous discutons de la classification des modèles, sous l'angle de la comparaison de solveurs. Ces réflexions sont étayées par les résultats d'un portfolio de solveurs utilisant les solveurs Choco2 et AbsCon sur la bibliothèque XCSP 2.1.

1 Introduction

La communauté Programmation Par Contraintes (PPC) dispose de trois principales bibliothèques pour l'évaluation des solveurs : MiniZinc [1], XCSP [2] et la CSPLib [3]. Les bibliothèques MiniZinc et XCSP sont construites autour d'un langage qui permet la représentation de nombreux problèmes sous contraintes. Tandis que MiniZinc permet de séparer modèles et données (tout comme par exemple, OPL), XCSP et FlatZinc permettent de représenter les instances de modèles (dites "à plat").

D'autres communautés proposent des bibliothèques de modèles, par exemple, la MIPLib ou la SATLib. Cependant, la définition d'un langage de modélisation/représentation commun est l'objet d'un plus grand consensus en SAT et MIP qu'en PPC. En effet, en PPC, les contraintes et algorithmes diffèrent significativement d'un solveur à l'autre. De plus, les performances d'un même solveur peuvent varier énormément sur deux modèles logiquement équivalents (et une reformulation efficace, voire automatisée, reste une tâche complexe). Finalement, une modélisation fine adaptée à un solveur peut considérablement améliorer les résultats. Un certain consensus existe malgré tout, notamment sur les principales contraintes globales.

Les bibliothèques de modèles sont couramment utilisées lors de compétitions de solveurs. Les solveurs sont alors évalués le plus souvent sur l'aspect *black-box* et les participants fournissent des efforts importants pour la reformulation automatique du problème ainsi que la configuration automatique du solveur. Par ailleurs, les participants proposent généralement des portfolio lorsque le parallélisme est autorisé.

Ces bibliothèques sont d'un intérêt certain pour l'expérimentation : vaste corpus de modèles et d'instances, comparaison d'algorithmes et reproductibilité des expériences. La bibliothèque la plus aboutie est la MIPLIB qui propose aussi un ensemble de scripts pour la résolution des instances et la compilation des résultats. D'un autre côté, la CSPLib propose des séries de *problèmes* (énoncés et données) pour lesquelles les jeux d'instances et les résultats sont compilés. Cette approche est très répandue en Recherche Opérationnelle : OR-library, TSP, PSP, etc. La ROADEF organise ainsi une compétition de résolution de problèmes où les participants proposent fréquemment des modèles très différents selon les technologies utilisées.

Un problème récurrent est la classification des instances. On peut distinguer deux pratiques courantes (l'une n'excluant pas l'autre) : classification qualitative et classification quantitative. Une classification *qualitative* regroupe les instances selon des caractéristiques communes définies à priori. Une classification *quantitative* regroupe les instances selon leur niveau de difficulté. La seconde pratique est très répandue dans les autres communautés, mais à notre connaissance, les classifications de MiniZinc et XCSP sont qualitatives.

L'intérêt d'une classification croisée, que nous re-

1. http://www.minizinc.org/
2. http://www.cril.fr/~lecoutre/benchmarks.html
3. http://www.csplib.org/

cherchons, est assez évident, à savoir, a) améliorer le processus de développement des solveurs (tests de correction et de performance), b) centrer les compétitions sur les modèles difficiles, c) proposer un corpus de modèles pertinents pour évaluer les nouvelles approches.

2 Bibliothèque XCSP

Le format XCSP 2.1 permet de représenter un réseau de contraintes en XML : instances des cadres CSP (Constraint Satisfaction Problem), Weighted CSP (WCSP) et Quantified CSP (QCSP). Ce format ne permet pas de représenter naturellement les problèmes d'optimisation [4]. Ceux-ci sont généralement transformés en un ou plusieurs CSPs (bornes supérieures ou preuve d'optimalité). Chaque réseau peut impliquer des contraintes en extension, en intension, ou même des contraintes globales (allDifferent, weightedSum, etc). Pour plus d'information, le lecteur pourra se référer au manuel de référence du format XCSP 2.1 [5].

Classification qualitative De nombreux modèles au format XCSP 2.1 ont été rassemblés ces dernières années. On peut les distinguer selon qu'ils soient structurés ou aléatoires. Cependant, cette classification a été affinée en définissant les catégories suivantes : **REAL** (applications réelles), **PATT** (motifs réguliers et avec composantes aléatoires), **ACAD** (académique sans composantes aléatoires), **QRND** (aléatoire et faiblement structuré), **RAND** (complètement aléatoire). On peut approximativement considérer que les modèles sont classés des plus structurés (**REAL**) aux moins structurés (**RAND**).

Choix des instances Nous ne traiterons que les CSPs des catégories **REAL** (1266) et **ACAD** (576), soit 1842 instances parmi presque 10000. De manière subjective, nous avons exclu les autres catégories. La catégorie **PATT** est principalement constitué de COPs transformés en séries d'instances CSP. Les catégories **QRND** et **RAND** ne sont composées que de contraintes en extension, et ces contraintes apparaissent assez fréquemment dans **REAL** et **ACAD**.

3 Classification quantitative

Nous classerons les instances grâce à un portfolio de solveurs Choco2 et AbsCon, chacun effectuant une résolution indépendante (pas de communication). Tous les solveurs utilisent un algorithme standard de retour arrière et l'heuristique de sélection de la valeur minimale.

4. XCSP 3.0 en cours de développement va y remédier.

Solveurs Choco2. `wdeg` : heuristique basée sur le ratio "domaine sur degré pondéré" [1] ; `wdeg-re` : `wdeg` avec une politique de redémarrage à croissance géométrique ; `impact` : heuristique basée sur les impacts [4] ; `impact-re` : `impact` avec redémarrages. Nous n'appliquerons ni reformulation, ni configuration automatique, une approche *blindbox* plutôt que *blackbox*. Aucun *nogood* n'est enregistré lors des redémarrages [2] et certaines parties de l'arbre de recherche peuvent donc être explorées plusieurs fois.

Solveurs AbsCon. En plus des configurations précédentes, `activity` : heuristique basée sur l'activité [3] ; `activity-re` : `activity` avec redémarrages ; `ddeg` : heuristique basée sur le ratio "domaine sur degré dynamique" ; `ddeg-re` : `ddeg` avec redémarrages. Les *nogoods* sont toujours enregistrés lors d'un redémarrage.

Environnement de test On alloue un cœur sur un cluster à chaque solveur pour une durée limitée, **Choco2** processeurs Intel E5-2670 (2.60 GHz – 8 cœurs) et 12 heures ; **AbsCon** processeurs Xéon (2.66 GHz – 2 coeurs) et 1000 secondes.

Nous proposons une classification en fonction des résultats du portfolio de solveurs. Le *virtual worst solver* (VWS) est le temps de résolution du pire solveur, i.e., le temps de résolution séquentielle avec le pire choix. Le *virtual best solver* (VBS) est le temps de résolution du meilleur solveur, i.e., le temps de résolution d'un portfolio parallèle. Les instances **faciles** sont classées avec le **VWS** : **training** (moins de 10 secondes) et **easy** (moins de 1000 secondes). Les instances **difficiles** sont classées avec le **VBS** : **medium** (moins de 1000 secondes) ; **challenging** (pas de timeout) ; **hard** (timeout ou memory exception

Ces catégories forment une partition des instances. La catégorie d'une instance est la première catégorie dont la condition est validée. Ainsi, une instance **medium** est résolue en moins de 1000 secondes par le VBS et en plus de 1000 secondes par le VWS (si ce n'était pas le cas pour cette seconde condition, elle serait **easy**). Bien sûr, une telle classification doit être étendue pour compiler des résultats venant de protocoles expérimentaux plus variés (solveurs, matériels, etc). Cependant, si le protocole est discutable pour identifier les modèles difficiles, il permet certainement d'identifier des modèles faciles.

4 Satisfaction de contraintes

La figure 1 permet de visualiser les performances globales du portfolio et celles des portfolio restreints à Choco2 et AbsCon. Pour chaque portfolio, le nombre d'instances résolues par le VBS (resp. VWS) est tracé

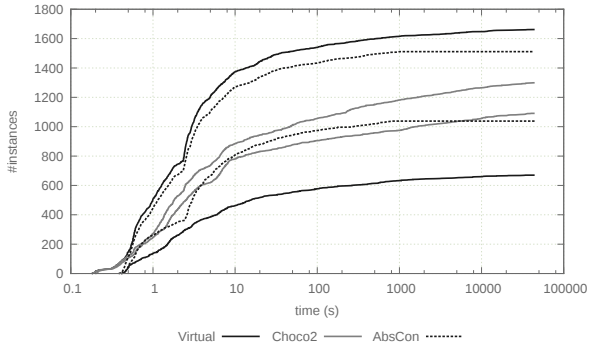

FIGURE 1 – Performances du portfolio.

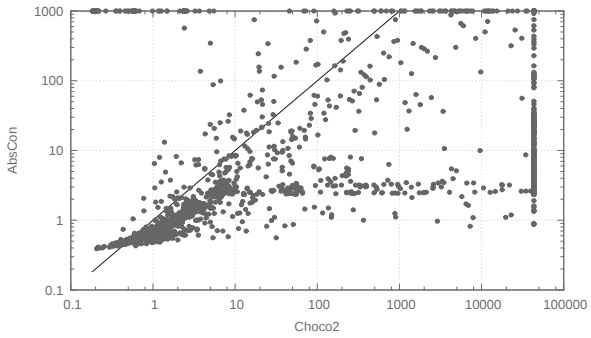

FIGURE 2 – Choco2 versus AbsCon

comme une fonction du temps de résolution. On peut observer que *75% des instances sont résolues en moins de 10 secondes par le VBS et 25% par le VWS*. Par ailleurs, plus le portfolio a de solveurs, plus l'écart entre le VBS et le VWS est grand. Le portfolio AbsCon résout plus d'instances que le portfolio Choco2, mais ces deux solveurs ne résolvent pas les mêmes instances (c.f. l'écart avec le VWS). Ainsi, l'intérêt du portfolio est démontré pour mesurer la difficulté des modèles.

La figure 2 décrit plus en détail les résultats des portfolio Choco2 et AbsCon. Chaque instance est représentée par un point dont la coordonnée x est le temps de résolution du portfolio Choco2 et la coordonnée y est le temps de résolution du portfolio AbsCon. Les échelles des deux axes sont logarithmiques. Tous les points situés sous (resp. au dessus de) la diagonale ($x = y$) représentent des instances résolues plus rapidement avec AbsCon (resp. Choco2). Les points formant une ligne verticale à droite (resp. horizontale en haut) sont les instances résolues seulement avec AbsCon (resp. Choco2). La figure confirme que Choco2 et AbsCon ont des comportements différents.

La figure 3 montre l'influence positive des redémarrages. Chaque triplet (instance, solveur, branchement) est représenté par un point dont la coordonnée x est le temps de résolution sans redémarrage et la coordonnée y est le temps de résolution avec redémarrage. Une

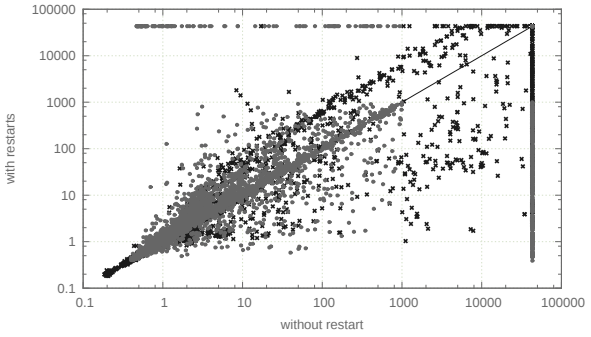

FIGURE 3 – Importance des redémarrages.

croix (resp. un cercle) indique que le solveur utilisé est Choco2 (resp. AbsCon). Les redémarrages améliorent globalement la résolution en permettant de résoudre un certain nombre d'instances supplémentaires dans le temps imparti. De plus, la dégradation des temps de résolution, dûe aux redémarrages, reste raisonnable, alors que les gains sont souvent importants. Dans le pire cas, les redémarrages détériorent plus la résolution avec AbsCon qu'avec Choco2 (ligne horizontale en haut), malgré les nogoods.

Le tableau 1 donne le nombre d'instances par classe. Plus de 600 instances sont quasiment triviales et seulement 226 instances sont réellement difficiles, dont 46 sont résolues par le portfolio.

5 Énumération de solutions

Notre but est maintenant de sélectionner des instances dont il est difficile d'énumérer les solutions (mais pour lesquelles c'est encore possible). Nous utiliserons uniquement Choco2 sans redémarrage sur 569 instances faciles, i.e., dont la première solution est trouvée en moins de deux minutes par le VWS Choco2. Tout d'abord, la tâche s'est révélée impossible pour 387 instances qui avait souvent des milliards de solutions ($> 2^{32}$). Au contraire, la tâche s'est avérée facile pour 153 instances résolues à nouveau en moins de deux minutes. Au final, seules subsistent 23 instances, dont certaines avec plus de 10^{12} solutions. Finalement, le tableau 2 indique les 15 instances difficiles pour le

	train.	easy	med.	chall.	hard	**total**
ACAD	144	48	233	23	128	576
REAL	318	122	751	23	52	1266
UNSAT	135	85	471	24	–	715
SAT	327	85	513	22	–	947
total	462	170	984	46	180	1842

TABLE 1 – Classification quantitative

Instance	#sol.	VBS
crossword-m1c-words-vg5-7_ext	14	562.3
crossword-m1-words-05-02	1025730	586.4
crossword-m1c-words-vg5-6_ext	2308	596.4
crossword-m1c-words-vg7-7_ext	6	908.3
crossword-m1-words-05-06	3298018	1094.4
crossword-m1c-words-vg6-6_ext	5845	1125.7
squares-9-9	9456120	1377.6
costasArray-13	12828	1447.7
series-14	9912	3386.1
queenAttacking-6	2678	3781.7
queens-15	2279184	3888.1
ortholatin-5	6220800	4881.2
costasArray-14	17252	7351.8
langford-3-17	26880	11715.5
costasArray-15	19612	37774.2

TABLE 2 – 15 instances pour l'énumération.

VBS avec moins de 10 millions de solutions (temps pour le VBS donné en secondes). On notera que l'influence de l'heuristique de sélection de variables est réduite. En effet, la moyenne géométrique du ratio des temps de résolution entre `impact` et `wdeg` est 0.996 avec un écart type de 0.105.

6 Décompte des points

Nous profitons des données fournies par cette campagne expérimentale intensive pour donner à titre indicatif un classement des solveurs les plus robustes par rapport à la bibliothèque XCSP. Pour classer les solveurs en fonction de leur performance, nous utilisons un principe de décompte des points basé sur la méthode Borda (système de vote pondéré). Chaque instance est un électeur qui classe les différents solveurs. Un solveur marque un nombre de points dépendant du nombre de solveurs qu'il bat. Un solveur s marque des points sur une instance i en comparant ses performances avec celles de chaque autre solveur s' : a) 1 point si s donne une meilleure réponse que s', b) si les réponses sont équivalentes, les points sont partagés, c) 0 point si s' donne une meilleure réponse. Quand les réponses de s et s' sont équivalentes, s marque $f(t,t') = t' \div (t+t')$ point ($f(0,0) = 0$). Nous proposons une nouvelle fonction $g(t,t') = g(t) + (1 - g(t) - g(t')) \times f(t,t')$ dans laquelle une partie des points est accordée à un solveur par contrat $g(t) = 1 \div (2 \times (\log_a(t+1)+1))$ ($a = 10$). Les points restants sont partagés entre les deux solveurs grâce à la fonction f.

Le tableau 3 donne les classements selon la fonction utilisée. Les deux premières colonnes (s) sont le nombre d'instances résolues dans la limite de temps par chaque solveur, une mesure de performance essentielle. Les changements pertinents induits par g sont indiqués en gras. Ainsi, le biais de la fonction f en faveur des solveurs rapides sur les instances faciles est réduit par la fonction g. Au sens des moindre carrés sur les classements, g est plus proche de s que f ($\|g-s\|_2 = 12$ et $\|f-s\|_2 = 30$. A noter que le suffixe `-re` signale une version avec redémarrages.

	s		Class. s		Class. f		Class. g	
	*	*-re	*	*-re	*	*-re	*	*-re
impact	1119	1196	14	12	14	12	14	12
wdeg	1156	1259	13	10	13	11	13	11
Choco2	**1299**		7		**10**		**8**	
activity	1337	1405	6	3	5	4	5	4
ddeg	**1275**	**1246**	9	11	**6**	**9**	**7**	**10**
impact	**1288**	**1372**	8	5	**8**	**7**	**9**	**6**
wdeg	1376	1465	4	2	3	2	3	2
AbsCon	1511		1		1		1	

TABLE 3 – Classements des solveurs.

7 Conclusion

Dans cet article, nous nous sommes intéressés à la question délicate de la classification quantitative des modèles (i.e., selon leur difficulté). Sur la base d'une approche portfolio, nous avons étudié les instances de deux catégories importantes de la bibliothèque XCSP. Ce travail sera utilisé et approfondi lors de la mise en place du nouveau site web, développé conjointement au nouveau format étendu de représentation XCSP 3.0, programmée courant 2014.

Remerciements

Ces travaux bénéficient du soutien du CNRS et d'OSEO dans le cadre du projet ISI "Pajero" et d'un accès aux moyens de calculs et de visualisation du Centre de Calcul Interactif hébergé à l'Université Nice Sophia Antipolis.

Références

[1] F. Boussemart, F. Hemery, C. Lecoutre, and L. Sais. Boosting systematic search by weighting constraints. *Actes de ECAI'04*, pages 146–150, 2004.

[2] C. Lecoutre, L. Sais, S. Tabary, and V. Vidal. Nogood recording from restarts. *Actes de IJCAI'07*, pages 131–136, 2007.

[3] L. Michel and P. Van Hentenryck. Activity-based search for black-box constraint programming solvers. *Actes de CPAIOR'12*, pages 228–243, 2012.

[4] P. Refalo. Impact-based search strategies for constraint programming. *Actes de CP'04*, pages 557–571, 2004.

[5] O. Roussel and C. Lecoutre. XML representation of constraint networks : Format XCSP 2.1. Technical Report arXiv :0902.2362, CoRR, 2009.