Backtracking distribué multi-variables avec sessions

Julien Vion **René Mandiau** **Sylvain Piechowiak**
LAMIH CNRS UMR 8201, UVHC, 59313 Valenciennes
{julien.vion, rene.mandiau, sylvain.piechowiak}@univ-valenciennes.fr

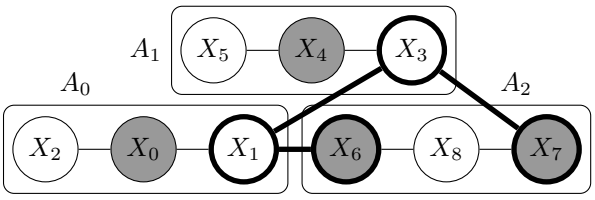
Résumé

Cet article est un résumé français de l'article *Multi-variable Distributed Backtracking with Sessions* [3]. Le formalisme des CSP Distribués (DisCSP) étend le modèle CSP classique pour représenter et résoudre des problèmes de décision ne pouvant être résolus de manière centralisée sur une seule machine, pour des raisons diverses (*e.g.*, taille excessive ou privacité). Dans cet article, nous proposons un algorithme complet de résolution de DisCSP, nommé *Backtracking distribué avec sessions* (DBS), qui a la particularité de gérer les DisCSP où chaque agent encapsule un sous-problème composé de plusieurs variables et contraintes. Nous prouvons que l'algorithme est correct et complet, et donnons des résultats expérimentaux prometteurs.

La majorité des travaux sur les DisCSP considèrent que chaque agent encapsule une seule variable. Cependant, la plupart des problèmes distribués (*e.g.*, emplois du temps, trafic routier, exploration multi-robots…) sont plus naturellement modélisés en affectant plusieurs variables et contraintes à chaque agent. Il est possible de décomposer le problème jusqu'à ce qu'il n'y ait plus qu'une seule variable par agent, ou encore de transformer le problème afin que le domaine de chaque variable représente l'ensemble des solutions au problème local, mais ces solutions ne sont guère satisfaisantes : elles ont entre autres pour effet d'augmenter exponentiellement le nombre de messages échangés ou la taille des domaines considérés.

Nous proposons l'algorithme *DBS*, qui se place dans la famille des algorithmes *ABT* [4, 1], avec la particularité de proposer une gestion spécifique des agents à plusieurs variables. D'autre part, pour établir le contexte des messages de backtrack, *ABT* nécessite de calculer un *nogood* lors de chaque conflit au cours de la recherche. Ces *nogoods* sont très coûteux à calculer [2]. Nous montrons qu'il est possible d'utiliser la notion de *session* pour définir le contexte des messages de backtrack, sans affecter les propriétés de correction et de complétude de l'algorithme. Ainsi, la prise en compte des messages est bien plus efficace. Cependant, il s'agit d'un compromis : en effet, les sessions sont moins porteuses d'information que les *nogoods* : il faut donc généralement échanger plus de messages pour résoudre un problème avec *DBS* qu'avec *(Multi-)ABT*. Nous réduisons l'impact de ces messages supplémentaires en proposant un ensemble de *filtres* permettant d'éliminer de nombreux messages inutiles sans affecter les propriétés de l'algorithme.

1 Fonctionnement de DBS



L'exemple ci-dessus est un problème de 2-coloration de graphe distribué entre trois agents A_0, A_1 et A_2, cités par ordre de priorité : *ABT* et *DBS* nécessitent de fixer un ordre de priorité statique entre les agents. Chaque agent choisit une solution locale (telle que représentée sur la figure), puis tente de l'étendre à ses voisins de priorité inférieure par l'envoi d'un message « OK ? ». Comme la solution de l'agent A_1 est incompatible avec la solution de A_0 (la contrainte $X_1 \neq X_3$ n'est pas respectée), A_1 va changer de solution pour avoir $X_3 =$ ●. L'agent A_2 reçoit trois messages : la solution locale de A_0, et les deux solutions locales successives de A_1. À chaque solution locale correspond un numéro de session. La première solution était consis-

tante et n'entrainait pas d'action de la part de A_2. La deuxième proposition va nécessiter pour A_2 de changer de solution, or il n'en existe pas qui soit compatible à la fois avec A_0 et A_1 : un message de backtrack est généré et est envoyé à A_1, accompagné du numéro de session correspondant.

Quand A_1 reçoit le message, il peut utiliser le numéro de session pour confirmer que la demande de backtrack correspond bien à sa solution courante (en effet, la première solution aurait pu être insatisfaisante pour A_2). Ici, *ABT* aurait nécessité des calculs complexes basés sur les nogoods attachés aux messages de backtrack. Comme A_1 n'a pas d'autre solution compatible, il transmet la demande de backtrack à A_0. Un processus similaire est alors executé pour la deuxième solution locale (symétrique) de A_0, qui détecte alors l'insatisfaisabilité du problème.

DBS utilise diverses structures de données pour mémoriser les solutions courantes des agents voisins de priorité supérieure ainsi que les solutions déjà proposées ou reçues : en effet, l'asynchronisme de l'algorithme peut amener un agent de priorité supérieure à proposer plusieurs fois une solution identique du point de vue de l'agent courant. Nous prouvons que *DBS* est correct et complet [3].

2 Gestion du multi-variable

Nous proposons une première optimisation simple pour gérer le multi-variable. Contrairement à *Multi-ABT* qui générait pour chaque agent une variable dont le domaine représentait l'ensemble des solutions locales, *DBS* ne considère que les solutions qui sont différentes du point de vue des autres agents (c'est-à-dire les affectations différentes des variables d'interface). Nous exploitons la notion de « voisinage partiellement interchangeable » (*PIN*).

D'autre part, *DBS* ne calcule pas l'ensemble des solutions au préalable, mais les génère au fur et à mesure de la recherche de manière « paresseuse ».

3 Filtrage des messages

Comme *DBS* tend à envoyer plus de messages que les autres algorithmes de la littérature, nous proposons un système de *filtres* (ou heuristiques) permettant d'éliminer certains messages obsolètes sans avoir à les traiter :

1. si un agent reçoit successivement deux propositions « OK ? » d'un même agent, seul le plus récent doit être considéré ;
2. à la réception d'un message « OK ? », tous les messages de backtrack en attente sont obsolètes ;
3. après l'envoi d'un message de backtrack, tous les messages de backtrack reçus des voisins de priorité inférieure à propos de la solution inconsistante deviennent obsolètes ;
4. si plusieurs messages « OK ? » sont en attente, il faut traiter en priorité les messages d'agents de priorité maximale pour réduire au mieux l'espace de recherche

4 Expérimentations & perspectives

Nous avons testé l'impact de l'utilisation des *PIN* et des différents filtres sur les performances de *DBS*, et l'avons comparé à d'autres algorithmes gérant le multi-variables : *Multi-ABT*, *Multi-AWC* et *AFC*.

Si le nombre de messages échangés par *DBS* peut rester important comparativement à ces algorithmes (jusqu'à deux ordres de magnitude avec une seule variable par agent), ceux-ci sont filtrés ou traités extrêmement rapidement, au point qu'aucune accumulation de messages n'est constatée dans la file d'attente. Dans le cas mono-variables, où *ABT* reste l'algorithme le plus performant, on constate que l'essentiel du temps de calcul de *DBS* est consacré à la gestion des messages, alors que pour ABT il est au niveau du calcul des nogoods. Quand le nombre de variables par agent augmente, *DBS* devient particulièrement performant (plus de deux ordres de magnitude en termes de *Wallclock CPU* de *NCCC*) par rapport aux autres algorithmes, dans la mesure où le nombre de messages n'explose plus.

Outre une amélioration possible des structures de données de *DBS*, nous prévoyons dans des travaux futurs d'étudier la gestion de la privacité et de la robustesse parmi cette famille d'algorithmes.

Références

[1] K. Hirayama, M. Yokoo et K. Sycara. "The Phase Transition in Distributed Constraint Satisfaction Problems : First results". In : *Proc. CP*. 2000, p. 515–519.

[2] D. L. Mammen et V. R. Lesser. "Problem Structure and Subproblem Sharing in Multi-Agent Systems". In : *Proc. ICMAS*. 1998, p. 174–181.

[3] R. Mandiau, J. Vion, S. Piechowiak et P. Monier. "Multi-variable Distributed Backtracking with Sessions". In : *Applied Intelligence* 41.3 (oct. 2014), p. 736–758.

[4] M. Yokoo. *Distributed Constraint Satisfaction : Foundations of Cooperation in Multi-agent Systems*. Springer Verlag, 2001.