VNS itératif guidé par la décomposition arborescente pour la minimisation d'énergie dans les modèles graphiques

Résumé de l'article paru à la conférence Uncertainty in Artificial Intelligence (UAI-17), pages 550-559, Sydney, Australia, 2017

A. Ouali[1] **D. Allouche**[2] **S. de Givry**[2] **S. Loudni**[1] **Y. Lebbah**[3] **F. Eckhardt**[2] **L. Loukil**[3]

1 University of Caen Normandy, CNRS, UMR 6072 GREYC, 14032 Caen, France.
2 INRA, MIA Toulouse, UR-875, 31320 Castanet-Tolosan, France.
3 University of Oran 1 Ahmed Ben Bella, Lab. LITIO, 31000 Oran, Algeria.

Les modèles graphiques probabilistes (MGP) [4] unifient la théorie des probabilités et les modèles graphiques via des variables aléatoires reliées par une distribution de loi jointe donnant l'aspect probabiliste.

Formellement, un MGP [4] est un triplet $(\mathcal{X}, \mathcal{D}, \mathcal{F})$ avec $\mathcal{X} = \{X_1, \ldots, X_n\}$ un ensemble de n variables aléatoires discrètes, $\mathcal{D} = \{D_1, \ldots, D_n\}$ un ensemble de domaines finis de valeurs, et $\mathcal{F}$, un ensemble de *fonctions de probabilités conditionnelles* à valeurs réelles positives. Une affectation de l'ensemble $\mathcal{X}$ est le tuple $x = (x_1, \ldots, x_n)$, avec $x_i \in D_i$. L'ensemble de toutes les affectation possibles de $\mathcal{X}$ est noté $\Delta = \prod_{i=1}^{n} D_i$. Soit S un sous-ensemble de $V = \{1, \ldots, n\}$, X_S, x_S et Δ_S dénotent respectivement un sous-ensemble de variables aléatoires $\{X_i, i \in S\}$, l'affectation $(x_i, i \in S)$ obtenue à partir de x, et l'ensemble des affectations possibles de X_S. Soit $\mathcal{S}$ un sous-ensemble des parties de V, l'ensemble $\mathcal{F} = \{f_S\}_{S \in \mathcal{S}}$ définit une factorisation de la distribution de probabilité jointe de $\mathbb{P}$ ssi :

$$\mathbb{P}(x) = \frac{1}{Z} \prod_{f_S \in \mathcal{F}} f_S(x_S) \quad (1)$$

où $Z = \sum_{x \in \Delta} \prod_{f_S \in \mathcal{F}} f_S(x_S)$ est la constante de normalisation. Le problème *Most Probable Explanation* (MPE) consiste à trouver une affectation $x \in \Delta$ de toutes les variables de $\mathcal{X}$ de telle sorte que la probabilité jointe $\mathbb{P}(x)$ soit maximale. Il s'agit d'un problème NP-difficile [9]. Le problème MPE se traduit directement en un *réseau de fonctions de coût* [7] de minimisation de la somme de fonctions de coût, appelées aussi *fonctions d'énergie* [3].

Les méthodes de résolution sont qualifiées respectivement de méthodes complètes ou incomplètes selon leur capacité à prouver ou non l'optimalité de la solution trouvée. Elles font généralement appel soit à la recherche arborescente soit à la recherche locale. La combinaison de ces deux aspects a été étudiée dans peu de travaux. Parmi eux, les approches qui consistent à explorer les voisinages dans la recherche locale par une recherche arborescente de manière systématique ou non systématique. VNS/LDS+CP [5] combine une métaheuristique VNS [8] avec une recherche partielle de type LDS [2]. Récemment, Fontaine et al [1] ont proposé le premier cadre générique, appelé DGVNS, qui exploite une décomposition arborescente au sein de VNS. Dans ce papier, nous proposons `UDGVNS`, une variante itérative de DGVNS capable de prouver l'optimalité des solutions trouvées.

DGVNS itératif. L'algorithme 1 décrit le pseudo-code de `UDGVNS`. Il restaure l'exhaustivité de DGVNS en appliquant des appels successifs avec des valeurs croissantes de discrepancy (notée ℓ) pour LDS [1], en contrôlant si la recherche arborescente est partielle grâce au drapeau *opt* et le fait que le voisinage actuel garde certaines variables assignées dans l'affectation partielle A (test à la ligne 6). Dans `UDGVNS`, l'optimalité peut être prouvée dans deux cas : (i) lorsque le voisinage actuel correspond à l'ensemble des variables du problème et que la valeur de discrepancy est supérieure ou égale au nombre maximal de branches droites ou (ii) en examinant les bornes au nœud racine ($ub = \texttt{lb}(\mathcal{D})$, cf. lignes 2 et 5). Dans ce cas, l'optimalité est prouvée implicitement, l'espace de recherche n'est

1. Nous supposons un arbre de recherche binaire où à chaque nœud soit la variable sélectionnée est affectée à sa valeur préférée (branche gauche) soit la valeur est supprimée du domaine (branche droite). Chaque suppression correspond à une mauvaise décision prise par la recherche.

Algorithm 1: Unified DGVNS algorithm.

```
   Function UDGVNS(ℓ_min, ℓ_max, +_ℓ, k_min, k_max, +_k, ub :
     In/Out, x : In/Out) : boolean
       let (C_T, T) be a tree decomposition of (X, D, F) ;
       opt ← true ;
1      LDS^r(∞, D, ub, x, opt) ;        // initial solution
2      if (ub = lb(D)) then opt ← true;
       c ← 1 ;                          // current cluster index
       r ← 0 ;                          // number of iterations
       ℓ ← ℓ_min ;                      // initial discrepancy limit
       while (¬opt ∧ ℓ ≤ ℓ_max) do
           i ← 0 ;                      // nb. of failed neighbor.
3          k ← k_min ;                  // init. neighbo. size
           while (¬opt ∧ k ≤ k_max) do
               A ← getNeighborhood(x, C_c, k) ;
               ub' ← ub, opt ← true ;
4              LDS^r(ℓ, A, ub', x', opt) ;   // nei. search
5              if (ub' = lb(D)) then opt ← true;
6              else if (A ≠ D) then opt ← false;
7              if (ub' < ub) then
                   x ← x', ub ← ub' ;   // new best sol
8                  i ← 0, k ← k_min ;
9                  r ← 0, ℓ ← ℓ_min ;
               else
                   i ← i + 1 ;
                   if (k < k_max) then
                       k ← min(k_max, k_min +_k i) ;
                   else k ← ∞;
10             c ← 1 + c mod |C_T| ;   // get next clus.
           r ← r + 1 ;
           if (ℓ < ℓ_max) then
               ℓ ← min(ℓ_max, ℓ_min +_ℓ r) ;
           else ℓ ← ∞;
11     return opt ;
```

pas exploré. La solution initiale est obtenue à la ligne 1 par une version modifiée de LDS, notée $\texttt{LDS}^r$, qui stoppe après l'obtention de la première solution.

UDGVNS ajuste le compromis entre preuve d'optimalité et comportement anytime via deux paramètres : la limite de la discrepancy (ℓ) et la taille du voisinage (k). Dès qu'une meilleure solution est trouvée par $\texttt{LDS}^r$ dans le voisinage courant (ligne 4), nous arrêtons la recherche afin de réinitialiser les deux paramètres à leur valeur minimale, car il est plus rapide d'explorer de petites voisinages (lignes 8-9). Nous avons évalué trois stratégies itératives de mise à jour de ℓ et k pour l'opérateur $+_{\ell/k}$: augmente d'une unité ($+$), multiplie par deux (mult2), enfin selon une suite Luby [6][2] qui renvoie la valeur $a +_{\ell/k} b = \texttt{Luby}(a, b) = a \times luby(1 + b)$ pour $\forall a, b \in N^*$.

L'opérateur $+_k$ contrôle le compromis entre intensification et diversification. Le but de la stratégie de Luby est d'intensifier l'effort de recherche sur les petits voisinages en augmentant exponentiellement le nombre de voisinages de petite taille versus ceux de grande taille. Permettant ainsi de passer plus de temps sur les petits voisinages pour améliorer localement la solution actuelle, favorisant ainsi l'intensification. La stratégie mult2 réduit le nombre d'explorations de voisinage pour une valeur de discrepancy donnée, afin d'essayer plus rapidement des valeurs de ℓ plus grandes. Si le problème peut être résolu par une recherche complète dans les délais impartis, la preuve d'optimalité sera également accélérée.

L'opérateur $+_\ell$ contrôle l'équilibre entre recherche incomplète et complète. L'utilisation d'une stratégie de croissance rapide accentue l'exhaustivité alors qu'une croissance lente devrait favoriser les comportements anytime. Nous avons noté qu'il est plus efficace de couvrir toutes les variables par l'union des voisinages explorés afin de ne pas manquer certaines variables importantes. Nous avons testé une quatrième stratégie pour k qui consiste en un incrément lent (de $+1$) au début jusqu'à $k = \max_{i \in I}(|C_i|) + |C_T| - 1$ puis *saute* directement à $k = k_{max}$. Cela garantit que k augmente lentement jusqu'à ce que le plus grand groupe ait été totalement exploré par au moins une recherche de voisinage. Quand $k = k_{max} = |\mathcal{X}|$, UDGVNS effectue une nouvelle recherche de solution via $\texttt{LDS}^r$ sur l'ensemble du problème. S'il ne parvient pas à trouver une meilleure solution, une discrepancy plus grande est envisagée et UDGVNS poursuit son processus d'intensification en commençant par une petite taille de voisinage (ligne 3).

Expérimentations. UDGVNS a été évalué sur 3016 instances issues de compétitions sur les modèles graphiques. Elles recouvrent divers domaines d'applications (Probabilistic Inference Challenge, Computer Vision and Pattern Recognition, OpenGM2 benchmark, Cost Function Library). À notre connaissance, ce travail est la première tentative de restauration de la complétude sur une méthode de recherche locale (i.e. VNS). Les résultats montrent que l'approche offre un bon compromis entre comportement anytime et preuve d'optimalité. L'article à UAI-17 présente également une version parallèle d'UDGVNS.

2. Rappelons que $luby(i) = \{1, 1, 2, 1, 1, 2, 4, 1, 1, 2, 1, 1, 2, 4, 8, \ldots\}$.

Références

[1] M Fontaine, S Loudni, and P Boizumault. Exploiting tree decomposition for guiding neighborhoods exploration for VNS. *RAIRO OR*, 47(2) :91–123, 2013.

[2] W Harvey and M Ginsberg. Limited discrepancy search. In *Proc. of IJCAI*, pages 607–615, 1995.

[3] B Hurley, B O'Sullivan, D Allouche, G Katsirelos, T Schiex, M Zytnicki, and S de Givry. Multi-Language Evaluation of Exact Solvers in Graphical Model Discrete Optimization. *Constraints*, 21(3) :413–434, 2016.

[4] D Koller and N Friedman. *Probabilistic graphical models : principles and techniques*. The MIT Press, 2009.

[5] S Loudni and P Boizumault. Solving constraint optimization problems in anytime contexts. In *Proc. of IJCAI*, pages 251–256, 2003.

[6] M Luby, A Sinclair, and D Zuckerman. Optimal speedup of Las Vegas algorithms. In *Proc. of TCS*, pages 128–133, 1993.

[7] P. Meseguer, F. Rossi, and T. Schiex. Soft constraints processing. In *Handbook of Constraint Programming*, chapter 9. Elsevier, 2006.

[8] N Mladenović and P Hansen. Variable Neighborhood Search. *Comput. Oper. Res.*, 24(11) :1097–1100, November 1997.

[9] S Shimony. Finding MAPs for belief networks is NP-hard. *Artificial Intelligence*, 68 :399–410, 1994.