Voisinage consistant sur des configurations partielles pour la résolution de problèmes réels de grande taille

Djamal Habet, Audrey Dupont, Michel Vasquez

Voisinage consistant sur des configurations partielles pour la résolution de problèmes réels de grande taille

Djamal Habet [1] **Audrey Dupont** [2] **Michel Vasquez** [2]

[1] LERIA, Université d'Angers, 2 bd Lavoisier, 49045, Angers Cedex 01
[2] Ecole des Mines d'Alès – EERIE – LGI2P, Parc S. G. Besse, 30035, Nîmes Cedex 01
Djamal.Habet@univ-angers.fr Audrey.Dupont@ema.fr Michel.Vasquez@ema.fr

Résumé

Nous présentons une approche générale pour la résolution des problèmes d'optimisation combinatoire. Dans ce but, nous concevons un voisinage, que nous appelons consistant, tel que à chaque nouvelle instanciation d'une variable, certaines autres variables, préalablement affectées, sont désinstanciées afin de maintenir la consistance des contraintes. Ainsi, au lieu d'autoriser des mouvements infaisables sur des configurations complètes, nous parcourons uniquement des configurations partielles et consistantes jusqu'à atteindre une solution complète.

Abstract

We present a general approach for solving constraint optimization problems. In this order, we design a consistent neighborhood which, after each variable assignment, deletes conflicting variables to maintain the constraint consistency. Hence, instead of allowing infeasible moves on complete configurations, we work only on partial consistent ones until a solution is found.

1 Introduction

Nous présentons une métode de résolution des problèmes d'optimisation combinatoire de grande taille, formalisés par des réseaux de contraintes de faible arité; le plus souvent binaire, et dont le temps de résolution est généralement limité. Cette méthode est basée sur un algorithme de recherche tabou [1] visitant un voisinage constitué de configurations partielles et consistantes. La consistance est maintenue par la propagation active des contraintes du problème. En effet, à chaque nouvelle instanciation d'une variable, on supprime les variables qui violeraient certaines contraintes. Ainsi, au lieu d'autoriser des mouvements infaisables sur les configurations complètes, on considère seulement des configurations partielles et consistantes jusqu'à la construction d'une configuration complète et consistante.

Dans cet article, nous définissons d'abord, dans la section 2, le formalisme CSP (Constraint Satisfaction Problem). Dans la section 3, nous présentons notre méthode de résolution, dite de *voisinage consistant*. Enfin, la section 4 conclut le papier.

2 Le formalisme CSP

Un problème de satisfaction de contraintes (CSP) [5, 4] est défini par un triplet $\mathcal{R} = (\mathcal{X},\mathcal{D},\mathcal{C})$ appelé réseau de contraintes, avec:

1. un ensemble de n variables $\mathcal{X} = \{x_1, \cdots, x_n\}$;
2. un ensemble de n domaines finis $\mathcal{D} = \{D_1, \cdots, D_n\}$, où D_i est l'ensemble des valeurs possibles de la variable x_i;
3. un ensemble de e contraintes $\mathcal{C} = \{C_1, \cdots, C_e\}$. Chaque contrainte C_i d'arité q est définie par:
   - le sous-ensemble ordonné $(x_{i_1}, \cdots, x_{i_q})$ des q variables sur lesquelles elle porte;
   - le sous-ensemble du produit Cartésien $D_{i_1} \times \cdots \times D_{i_q}$ qui spécifie quelles sont les valeurs des variables de $(x_{i_1}, \cdots, x_{i_q})$ qui sont compatibles entre elles.

Le problème consiste à trouver les valeurs à assigner à toutes les variables de l'ensemble $\mathcal{X}$ telles qu'elles satisfassent toutes les contraintes de l'ensemble $\mathcal{C}$, ou d'établir qu'une telle affectation n'existe pas. L'ensemble $\mathcal{S}$ de toutes les configurations possibles (*espace de recherche*) est le produit cartésien $\mathcal{S} = D_1 \times \cdots \times D_n$ de taille $|\mathcal{S}| = |D_1| \times \cdots \times |D_n|$.

3 Méthodologie

Lors de la résolution de CSP exprimés par un réseau de contraintes $\mathcal{R} = (\mathcal{X},\mathcal{D},\mathcal{C})$, les algorithmes de recherche locale travaillent souvent sur une *instanciation complète* des variables du problème et effectuent une série de réparations sur celle-ci jusqu'à l'aboutissement à une solution satisfaisant toutes les contraintes $\mathcal{C}$. Ainsi, les configurations visitées lors d'un processus de recherche locale sont complètes et souvent incohérentes avec les contraintes.

Notre idée est de concevoir une recherche locale qui parcourt uniquement des configurations qui maintiendraient la satisfaction des contraintes. De cela, nous considérons durant la recherche des instanciations partielles où certaines variables ne sont pas encore affectées mais les variables instanciées vérifient bien les contraintes du problème. Pour ce faire, à chaque étape de la recherche, on affecte une nouvelle valeur à une variable non instanciée tout en réparant la configuration ainsi constituée. Toutes les configurations visitées sont alors consistantes et c'est pour cette raison qu'on parle d'un *voisinage consistant*.

Nous décrivons ci-dessous les différentes étapes qui nous mènerons à la construction d'un tel voisinage et à son exploitation par un algorithme de recherche tabou. Cette description est établie sur des réseaux de contraintes binaires.

3.1 Configuration partielle

Habituellement, on définit une configuration par une affectation à toutes les variables du problème, on parle alors d'une configuration (instanciation) complète. L'espace de recherche S est composé de l'ensemble de toutes [1]
les configurations possibles. Cette configuration est représentée par le vecteur s décrit comme suit:

$$s = (v_1,\ v_2, \cdots, v_n) \text{ où } v_i \in D_i,\ i = 1 \cdots n. \quad [2]$$

Dans notre cas, l'espace de recherche est constitué de *configurations partielles* exprimées par $s = (v_{i_1},\ v_{i_2},\ \cdots, v_{i_{n_s}})$, telles que n_s variables sont instanciées, $n_s = |s| \leq n$. Toutefois, afin de revenir à la représentation classique d'une instanciation, on complète une configuration partielle par l'ajout d'une nouvelle valeur u (u pour *u*nconstrained: non-contrainte) à chaque domaine D_k pour indiquer que la variable x_k est libre (non instanciée). Dans la suite, les notations suivantes sont utilisées:

- v_i est la valeur courante de la $i^{ème}$ variable x_i, $i = 1 \cdots n$;
- $D_i[k]$ est la $k^{ème}$ valeur de D_i utilisée pour parcourir ce domaine.

3.2 Voisinage consistant

Comme on l'a indiqué précédemment, le voisinage exploré par une recherche locale est consistant. Le passage d'une configuration à une autre se fait par l'instanciation d'une variable non instanciée, dont *la valeur est fixée à u*, et la désinstanciation des variables qui causeront la violation de certaines contraintes. Ces désinstanciations sont opérées par la *propagation des contraintes* sur les variables voisines à celle récemment instanciée.

Un algorithme de recherche tabou sur un voisinage consistant entame donc la recherche par une configuration initiale partielle et consistante et tente d'aboutir à une solution complète. Guidé par ce but, le critère d'évaluation d'une configuration s, ainsi que de son voisinage $\mathcal{N}(s)$, est le nombre de variables instanciées dans s: la fonction *objectif* à maximiser est donc $f(s) = |s|$, sous l'ensemble des contraintes $\mathcal{C}$.

Toutefois, au lieu de calculer $f(s)$, on comptabilise, grâce à une table δ, et pour toute variable libre x_i, le nombre de variables à supprimer (à désinstancier), $\delta(i,\ k)$, si x_i est fixée à la valeur $D_i[k]$. Ce calcul se fait par la propagation implicite des contraintes portant sur x_i. La table δ est calculée dynamiquement (comme nous allons le développer plus loin) à chaque nouvelle instanciation et constitue de ce fait un historique de la recherche. À partir de ces éléments, l'évaluation des configurations voisines à s, par la table δ est comme décrite dans l'algorithme 1.

Algorithme 1 : ÉVALUATION_$\mathcal{N}(s)$
début
  $\delta_{min} \leftarrow n$; $\delta_{min}^{\neg tabou} \leftarrow n$; $\mathcal{L}_{cand} \leftarrow \emptyset$;
  pour tout $x_i \in s$, tel que $v_i = u$ **faire**
    pour $k = 1 \cdots |D_i|$ **faire**
      si $\delta(i,k) < \delta_{min}$ **alors**
        $\delta_{min} \leftarrow \delta(i,k)$;
      si $(x_i,D_i[k])$ *n'est pas tabou* **alors**
        si $\delta(i,k) < \delta_{min}^{\neg tabou}$ **alors**
          $\delta_{min}^{\neg tabou} \leftarrow \delta(i,k)$;
          $\mathcal{L}_{cand} \leftarrow (x_i,D_i[k])$;
        sinon
          si $\delta(i,k) = \delta_{min}^{\neg tabu}$ **alors**
            $\mathcal{L}_{cand} \leftarrow \mathcal{L}_{cand} \cup \{(x_i,D_i[k])\}$;
  retourner $(\mathcal{L}_{cand},\ \delta_{min})$
fin

δ_{min} est le nombre minimal de variables à supprimer parmi tous les voisins de la configuration s, par contre, $\delta_{min}^{\neg tabou}$ est biaisé par le caractère tabou de la recherche. Cette distinction est expliquée ultérieurement. Enfin, $\mathcal{L}_{cand}$ est la liste des candidats au mouvement $(x_i,D_i[k])$, tel que $\delta(i,k) = \delta_{min}^{\neg tabu}$.

Lors de cette évaluation, le parcours des domaines des variables libres dans un graphe de contraintes susceptiblement dense nécessiterait un temps de calcul élevé. Afin d'assurer une implémentation efficace de cette étape d'évaluation ainsi que l'étape de propagation de mouvement, nous avons eu recours aux principes du calcul incrémental. Nous consacrons la section 3.5 à la description des mécanismes mis en place à cette fin.

3.3 Heuristique de mouvement

Un mouvement, noté $mv(x_i, v_i)$, à partir d'une configuration s, consiste à affecter la valeur v_i à la variable x_i, préalablement instanciée à u, puis à désinstancier r variables $x_{j_1},\cdots,x_{j_r}$, préalablement affectées de $v_{j_1},\cdots,v_{j_r}$, respectivement. Ces désinstanciations sont nécessaires car les valeurs $v_i,\cdots,v_{j_r}$ des variables $x_{j_1},\cdots,x_{j_r}$ ne sont pas des supports pour la valeur v_i de x_i par les contraintes $C_{ij_1},\cdots,C_{ij_r}$. Elles sont accomplies par les mouvements $mv(v_{j_1}, u)$, $\cdots$, $mv(v_{j_r}, u)$. La configuration résultante de ces opérations est s' et on note $s' = s + mv(x_i, v_i)$. Cependant, afin de mieux s'approcher d'une solution complète, le mouvement minimisant le nombre de variables à désinstancier est sélectionné. Ces mouvements candidats sont contenus dans la liste $\mathcal{L}_{cand}$ retournée par l'algorithme 1 tel que $r = \delta^{\neg tabu}_{min}$.

Contrairement au mouvement classique, où la réparation s'effectue uniquement sur une seule variable, ce mouvement "instanciation + désinstanciation(s)" qu'on vient de définir est multi-attributs (un attribut correspond à une variable participant au mouvement). Il porte, par sa sémantique, sur plusieurs variables et permet ainsi une exploration plus large de l'espace de recherche.

3.4 Gestion de la liste tabou

Étant donné qu'un mouvement est multi-attributs, les risques d'occurrence de cycles dans l'espace de recherche sont plutôt importants. Afin de parer à la production de tels cas, on fait intervenir une liste tabou qui va contenir, tout au long de la recherche, des mouvements interdits. Plus précisément, pour éviter de défaire l'instanciation (x_i, v_i) prématurément, on classe tabou les mouvements $mv(x_{j_1}, v_{j_1}), \cdots, mv(x_{j_r}, v_{j_r})$ durant une certaine durée $\mathcal{DT}$ donnée par:

$$\mathcal{DT} = freq(x_k, v_k) \text{ et } tabou(x_k, v_k) = iter + \mathcal{DT}.$$

Où $freq(x_k, v_k)$ est le nombre d'affectations de v_k à x_k depuis le début de la recherche tabou. La prise en compte de ce statut tabou lors de l'évaluation de $\mathcal{N}(s)$ (voir algorithme 1) est indiquée par $\delta^{\neg tabou}_{min}$ calculée uniquement sur les mouvement acceptés.

Cependant, on autorise un mouvement tabou $mv(x_k, v_k)$ si la configuration s' produite à partir de s par un tel mouvement (*i.e.* $s' = s + mv(x_k, v_k)$) améliore strictement la configuration la plus complète atteinte depuis le début de la recherche, s^*. Autrement dit, si $|s'| > |s^*|$, et par l'application de ce critère d'aspiration, le statut tabou de $mv(x_k, v_k)$ est levé. De ce fait, la condition à la ligne [2] de l'algorithme 1 correspond au test $tabu(x_i, D_i[k]) \leq iter$ ou $|s'| > |s^*|$.

3.5 Propagation de contraintes

Cette étape de propagation s'effectue en deux phases. Toutes les deux visent à mettre à jour, de façon cohérente, la table δ.

– PROPAGER_VALEUR(x_i, v_i)

Cette fonction énumère les variables x_j voisines à x_i par la contrainte C_{ij} et parcourt leurs domaines pour supprimer les valeurs qui sont pas des supports de v_i (ligne [1] de l'algorithme 2). Cette suppression est effectuée en décrémentant la valeur $Dlibre_j$ (initialisée à $|D_j|$) qui indique le nombre de valeurs valides de D_j).

Algorithme 2 : PROPAGER_VALEUR(x_i,v_i)
début
  pour *tout* $x_j \in \Gamma(x_i)$ **faire**
    pour $k = 1$ *à* $|D_j|$ **faire**
      si $(v_i, D_j[k]) \notin C_{ij}$ **alors**
        si $\delta(j,k) = 0$ **alors**
          $Dlibre_j$−−;
        $\delta(j,k)$++ ;
    si $((v_j \neq u)$ *et* $((v_i, v_j) \notin C_{ij}))$ **alors**
      $\mathcal{L}_{del} \leftarrow \mathcal{L}_{del} \cup \{x_j\}$;
  retourner $\mathcal{L}_{del}$;
fin

[1] [2]

Enfin, si la variable x_j est préalablement fixée à une valeur $v_j \neq u$ qui n'est pas un support de v_i alors x_j est rajoutée à liste $\mathcal{L}_{del}$ des variables à désinstancier.

– PROPAGER_$u(x_j)$

Par sa définition, la valeur u ($u$nconstrained) est cohérente avec toute autre valeur des variables de $\mathcal{X}$. Par conséquent, on parcourt les domaines des variables $x_l \in \Gamma(x_j)$ afin de décrémenter la valeur de $\delta(l,k)$ si l'ancienne valeur v_j de x_j n'était pas un support de $D_l[k]$. Ainsi, si l'affectation de $D_l[k]$ à x_l ne supprimera aucune autre instanciation (*i.e.* $\delta(l,k) = 0$) alors le nombre de valeurs valides de D_l est incrémenté (ligne [1] de l'algorithme 3).

Algorithme 3 : PROPAGER_$u(x_j)$
début
  pour *tout* $x_l \in \Gamma(x_j)$ **faire**
    pour $k = 1$ *à* $|D_l|$ **faire**
      si $(v_j, D_l[k]) \notin C_{jl}$ **alors** $\delta(l,k)--$;
      si $\delta(l,k) = 0$ **alors**
[1] $Dlibre_l$++;
fin

Conjointement, ces deux fonctions achèvent la propagation de tout mouvement $mv(x_i,v_i)$ sur le réseau de contraintes $\mathcal{R}$ comme indiqué dans l'algorithme 4.

Algorithme 4 : PROPAGER_MOUVEMENT(x_i,v_i)
début
  $\mathcal{L}_{del} \leftarrow \emptyset$; $v_i \leftarrow D_i[k]$;
  $\mathcal{L}_{del} \leftarrow$ PROPAGER_VALEUR(x_i,v_i);
  tant que $\mathcal{L}_{del} \neq \emptyset$ **faire**
    $x_j \leftarrow choix(\mathcal{L}_{del})$;
    PROPAGER_$u(x_j)$;
    $v_j \leftarrow u$;
fin

3.6 Algorithme général

La combinaison des éléments décrits ci-dessus nous mène au schéma de résolution tabou sur un voisinage consistant présenté par l'algorithme suivant, et que nous nommons $C\mathcal{N}$-*Tabu* (*Consistent Neighborhood in Tabu Search*).

Algorithme 5 : $C\mathcal{N}$-*Tabu*
début
  $s \leftarrow glouton(S)$; $s^* \leftarrow s$;
  répéter
    $(\mathcal{L}_{cand}, \delta_{min}) \leftarrow$ ÉVALUATION_$\mathcal{N}(s)$;
    si $\mathcal{L}_{cand} \neq \emptyset$ **alors**
      $iter$++;
      $(x_i, D_i[k]) \leftarrow Selection\_Aléatoire(\mathcal{L}_{cand})$;
      PROPAGER_MOUVEMENT(x_i,v_i);
      $freq(x_i,v_i)$++ ;
      si $|s| > |s^*|$ **alors**
        $s^* \leftarrow s$;
        si $|s^*| = n$ **alors**
          retourner s^*;
  jusqu'à $\mathcal{L}_{cand} = \emptyset$;
  retourner s^*;
fin

L'algorithme $glouton(S)$ tente d'instancier successivement les variables libres tant qu'aucune contrainte n'est violée. S'il renvoie une configuration partielle alors la recherche tabou tente de l'améliorer pour construire une solution complète.

4 Conclusion

Nous avons proposé dans cet article une approche hybride entre la recherche locale et la propagation de contraintes pour traiter des problèmes d'optimisation combinatoire. Par le maintien continu de la consistance des configurations visitées, cette méthode se sert des contraintes pour définir la direction de l'espace de recherche à explorer. Nous avons appliqué cette méthode de voisinage consistant à la résolution de plusieurs problèmes (voir [6, 8, 7, 2]).

On retrouve dans la littérature des travaux utilisant des configurations partielles dans un algorithme de recherche locale, par exemple *decision-repair* [3]. Cependant, par restriction du nombre de pages, il est impossible d'effectuer des comparaisons avec ces approches proposées. Pour la même raison, quelques détails de la méthodologie ont été omis.

En perspective de ces travaux, nous nous intéresserons à l'application du voisinage consistant à d'autres problèmes académiques et industriels, notamment, les problèmes de satisfiabilité (SAT) et de coloration de graphes.

Références

[1] F. Glover and M. Laguna. *Tabu Search*. John Wiley & Sons, 1989.

[2] D. Habet and M. Vasquez. Saturated and Consistent Neighbourhood for Selecting and Scheduling Photographs of Agile Earth Observing Satellite. *In Proc. of the 5th Metaheuristics International Conference (MIC'2003)*, pages 28–1 – 28–6, 2003.

[3] N. Jussien and O. Lhomme. Local Search With Constraint Propagation and Conflict-Based Heuristics. *Artificial Intelligence*, 139(1):21–45, 2002.

[4] A. K. Mackworth. Constraint Satisfaction. In S. C. Shapiro, editor, *Encyclopedia of Artificial Intelligence*, John Wiley & Sons, New York, pages 285–293, 1992.

[5] E. Tsang. *Foundations od Constraint Satisfaction*. Academic Press, 1993.

[6] M. Vasquez. Arc–Consistency and Tabu Search for the Frequency Assignement Problem with Polarisation. In *Proc. of the 4th International Workshop on Integration of AI and OR techniques in Constraint Programming for Combinatorial Optimization Problems (CP-AI-OR'02)*, pages 359–372, Le Croisic, France, march 2002.

[7] M. Vasquez and J.K. Hao. A Heuristic Approach for Antenna Positioning in Cellular Network. *Journal of Heuristics*, 7:443–472, 2001.

[8] M. Vasquez and J.K. Hao. A "Logic-Constrained" Knapsack Formulation and a Tabu Algorithm for the Daily Photograph Scheduling of an Earth Observation Satellite. *Computational Optimization and Applications*, 20:137–157, 2001.