Autour du problème de la couverture exacte

Problème de la couverture exacte

Le problème de la couverture exacte est un problème NP-complet de référence en optimisation combinatoire. D'un point de vue théorique, il est utile pour prouver, par réduction, la NP-complétude d'autres problèmes classiques, comme celui du sac à dos [1]. D'un point de vue pratique, il correspond à une classe assez large de problèmes, dont les applications proposées ci-après sont des illustrations caractéristiques.

Énoncé Étant donné un ensemble fini E, et une famille P de parties de E recouvrant E, le problème de la couverture exacte de E consiste à déterminer s'il existe une partie de P qui constitue une partition de E[1]. Une solution constructive pour ce problème consiste à exhiber une partie de P qui est une partition de E, voire à donner l'ensemble des parties de P qui sont des partitions de E.

Par exemple, pour E = {a,b,c,d,e} et P = { {a,b}, {a,c,d}, {d,e}, {a,e}, {c,d}, {a,c,e}, {b,d}, {c} }, il existe plusieurs solutions : { {a,b}, {d,e}, {c} }, { {a,e},{b,d}, {c} } et { {a,c,e}, {b,d} }

Par contre, il n'existe pas de solution pour P' = { {a,b}, {a,c,d}, {d,e}, {a,e}, {c,d}, {a,c,e}, {b,c} }

Représentation matricielle On peut représenter P par une matrice dont chaque ligne représente un élément de P, sous la forme de sa fonction caractéristique, donnée en extension (chaque élément de P est un sous-ensemble de E).

On définit ainsi une matrice $M = (m_{i,j})_{1 \leq i \leq |P|, 1 \leq j \leq |E|}$ dont les colonnes correspondent aux éléments de E et dont les lignes correspondent aux éléments de P. Le coefficient $m_{p,e}$ de M vaut 1 si l'élément de E correspondant à la colonne e appartient à l'élément de P correspondant à la ligne p, et 0 sinon. Par exemple, pour l'exemple précédent, la matrice M est :

$$\begin{array}{c} \\ \{a,b\} \\ \{a,c,d\} \\ \{d,e\} \\ \{a,e\} \\ \{c,d\} \\ \{a,c,e\} \\ \{b,d\} \\ \{c\} \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array}$$

Le problème de la couverture exacte revient à rechercher une partie de P telle que tout élément de E appartient à un élément et un seul de cette partie de P. Avec la représentation matricielle, le problème de la couverture exacte revient à rechercher un ensemble de lignes tel que chaque colonne de cet ensemble de lignes contient un 1 et un seul. C'est par exemple le cas pour les lignes de la matrice précédente correspondant à la solution { {a,c,e}, {b,d} } :

$$\begin{pmatrix} 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{pmatrix}$$

Énoncé alternatif[2] Le problème de la couverture exacte peut aussi considéré du point de vue des éléments de P (des lignes de la matrice). Chaque élément e de E correspond alors à un sous-ensemble de P, à savoir les éléments de P qui contiennent e. Le problème de la couverture exacte consiste alors à déterminer une partie de P dont l'intersection avec chacune des parties de P correspondant aux éléments de E est réduite à un singleton.

1. Un ensemble de parties de de E est une partition de E si aucune de ces parties n'est vide, si elles sont 2 à 2 disjointes, et si leur union est égale à E

2. Cette formulation, pas vraiment immédiate, facilite cependant la modélisation de quelques problèmes.

Algorithme X (Donald Knuth)

L'algorithme X réalise une recherche exhaustive des solutions du problème de la couverture exacte. Sa structure est simple : elle consiste à explorer en profondeur d'abord l'ensemble des choix possibles. On va ainsi considérer successivement chacun des éléments de P comme premier élément d'une solution ; lorsqu'un élément p de P est choisi, les éléments de P restants ayant une intersection non vide avec p sont éliminés pour le choix suivant ; les éléments restants vont être successivement considérés comme élément suivant de la solution, etc... Lorsqu'une solution (ou une impasse) est trouvée, on effectue un retour arrière pour examiner le choix suivant. Cette exploration simple est assortie d'une heuristique tout aussi simple, mais qui s'avère très efficace, qui consiste à toujours choisir en priorité un élément de P contenant un élément de E commun à un minimum d'éléments de P, ce qui minimise le nombre de choix à explorer à un niveau donné.

En termes de représentation matricielle l'exploration d'une branche par l'algorithme X peut s'écrire ainsi :

```
algo_X(M : matrice)
  si M est vide alors
    ajouter la solution courante à l'ensemble des solutions
       (et terminer si la recherche n'est pas exhaustive)
    retour
  si une colonne est nulle, alors
    retour /* la solution courante ne peut être solution */
  choisir la première colonne c contenant un minimum de 1 /* heuristique */
  /* considérer une solution pour chaque élément de P contenant c */
  pour chaque ligne l  telle que m_{l,c} = 1 faire
    ajouter à la solution courante l'élément de P correspondant à l
    /* éliminer tous les éléments de P ayant une intersection non vide
     * avec la ligne considérée */
    pour tous les j tels que m_{l,j} = 1 faire
      pour toutes les lignes i telles que  m_{i,j} = 1 faire supprimer la ligne i de M
      /* retirer j de l'ensemble des éléments de E pour lesquels chercher une couverture */
      supprimer la colonne j de M
    /* poursuivre la recherche sur la matrice réduite */
    algo_X(M)
```

Par exemple, pour la matrice M précédente, et en s'arrêtant à la première solution trouvée :

État initial

$$\begin{array}{c} \\ \{a,b\} \\ \{a,c,d\} \\ \{d,e\} \\ \{a,e\} \\ \{c,d\} \\ \{a,c,e\} \\ \{b,d\} \\ \{c\} \end{array} \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{pmatrix} \end{array} = M_0, Solution = \varnothing$$

Niveau 1 choix de la colonne b (minimum de 1 : 2 lignes {a,b} et {b,d}).

1.1 Exploration de la branche {a,b} : éliminer toutes les lignes contenant a ou b, et les colonnes a et b

$$\begin{array}{c} \\ \{d,e\} \\ \{c,d\} \\ \{c\} \end{array} \begin{array}{c} \begin{array}{ccc} c & d & e \end{array} \\ \begin{pmatrix} 0 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \end{pmatrix} \end{array} = M_{n1.1}, Solution = \{\{a,b\}\}$$

n1.1, niveau 2 choix de la colonne e (minimum de 1 : 1 ligne {d,e}). Une seule branche : éliminer toutes les lignes contenant d ou e, et les colonnes d et e

$$\overset{c}{\{c\}}\ \binom{1}{} = M_{n1.1;n2.1}, Solution = \{\{a,b\},\{d,e\}\}$$

n1.1, n2.1, niveau 3 choix de la colonne c (minimum de 1 : 1 ligne {c}). Une seule branche : éliminer toutes les lignes contenant c, et la colonne c

$$() = M_{n1.1;n2.1;n3.1}, Solution = \{\{a,b\},\{d,e\},\{c\}\}$$

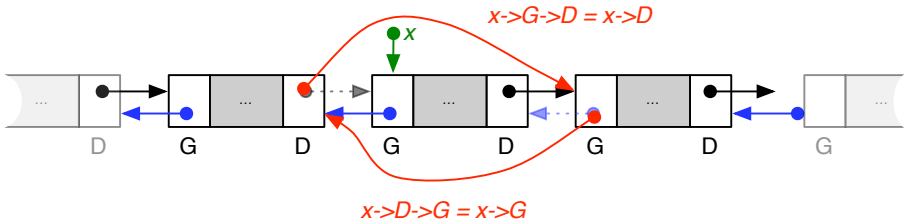
n1.1, n2.1, n3.1 niveau 4 la matrice est vide. La solution est valide. Arrêt (pour une recherche exhaustive, il faudrait explorer la branche 1.2 (ligne {b,d})).

Dancing Links

La technique des *dancing links* repose sur une utilisation ingénieuse de la structure de liste chaînée pour implémenter de manière efficace le retour sur trace à la base de l'algorithme X.

L'idée de base, très simple, est la suivante : On considère une liste doublement chaînée, dont les éléments sont des structures comportant 2 champs : G, qui contient la référence vers le successeur gauche de l'élément et D, qui contient la référence vers le successeur droit de l'élément.

Pour supprimer un élément dont la référence est x, il suffit, classiquement, de mettre à jour les champs G et D de ses successeurs droit et gauche, comme indiqué par la figure suivante (où l'on utilise la notation du langage C pour les champs de structures référencées)



L'observation intéressante est la suivante (*dancing links*) : si l'on considère l'élément qui vient d'être supprimé (de référence x), ses champs G et D contiennent les références des éléments de la liste qu'il conviendrait de mettre à jour si l'on voulait restaurer l'état de la liste avant la suppression. En outre cette restauration est simple : il suffit d'exécuter les affectations `x->D->G = x` et `x->G->D = x`.

Autrement dit, si l'on suppose que les éléments de la liste initiale sont toujours conservés en mémoire, même s'ils sont supprimés, alors on dispose d'un mécanisme de récupération nettement plus efficace que la classique sauvegarde/restauration dans un espace de manœuvre.

Cette observation est mise à profit dans l'article de Donald Knuth [2] pour implanter l'algorithme X. En effet, la matrice M spécifiant un problème de couverture exacte est représentée comme une matrice creuse : ses lignes et ses colonnes sont implantées par des listes doublement chaînées reliant les éléments non nuls de la matrice. La technique des *dancing links* permet alors d'effectuer une exploration séquentielle exhaustive en profondeur d'abord de l'ensemble des choix possibles : les *dancing links* permettent de réaliser de manière simple et efficace la sauvegarde/restauration associée au retour arrière, et il suffit de garder une unique copie de M en mémoire (il est inutile de gérer une pile ou des copies partielles de M par branche de l'arbre d'exploration). L'implémentation est étudiée et détaillée en [2], et de nombreuses implémentations sont disponibles en ligne.

Cette technique pourra être utilisée de manière tout aussi pertinente pour une version parallèle MapReduce de l'algorithme X : en effet, la parallélisation pourra se faire en largeur d'abord, mais l'exploration

séquentielle restera nécessaire lorsqu'un certain niveau de profondeur aura été atteint, car il n'est tout de même pas envisageable de dupliquer la matrice M en autant d'exemplaires que de branches de l'arbre d'exploration.

Pentominos

Les pentominos sont des casse-têtes, qui peuvent être vus comme des instances du problème de la couverture exacte. L'article de Donald Knuth [2] utilise les pentominos pour illustrer et évaluer l'algorithme X et la technique des *dancing links*.

Les grands traits du problème sont les suivants. On considère un damier comportant C colonnes et L lignes. On dispose d'un ensemble de pentominos [3] dont l'aire totale est égale à celle du damier (soit $C \times L$ cases). La question est de déterminer toutes les manières possibles de recouvrir exactement le damier avec les pentominos.

La formalisation en termes de couverture exacte consiste à définir une matrice avec une colonne pour chacune des cases, ainsi que pour chacun des pentominos. Pour chaque pentomino, on définit autant de lignes que de positions possibles pour le pentomino. La ligne associée à une position donnée pour un pentomino donné aura un 1 dans les colonnes associées aux cases correspondant à la position, ainsi que dans la colonne correspondant au pentomino, et des 0 dans les autres colonnes.

Sudoku

Les sudoku peuvent aussi être vus comme des instances du problème de la couverture exacte, et être résolus selon le principe de l'algorithme X. Il faut noter que, pour ce cas, l'algorithme est tellement efficace qu'une version séquentielle permet d'aboutir très rapidement à une solution, ainsi que cela est indiqué dans les exemples (fournis) accompagnant la distribution standard d'Hadoop. L'intérêt d'une version *MapReduce* ne devrait apparaître que pour des sudokus de grande taille (a priori supérieure à 100 x 100)

La formalisation en termes de couverture exacte considère le problème comme un ensemble de contraintes que le contenu de la grille doit satisfaire, chaque contrainte devant être satisfaite une fois et une seule. Par exemple : chaque ligne de la grille doit comporter un 9 et un seul.

Le problème est alors représenté par une matrice, avec

— une colonne pour chacune des contraintes,

— une ligne pour chaque contenu possible de chaque cellule. Lorsque le contenu de la cellule est fixé par la grille, la matrice ne comporte qu'une ligne ; lorsqu'il n'est pas connu, la matrice comporte autant de lignes que de valeurs possibles.
[Un sudoku 9×9 comporte 81 cases, pouvant avoir 9 valeurs possibles, soit 729 lignes au plus].

Plus précisément, la matrice comportera :

— une colonne pour chaque cellule (chaque cellule ne peut contenir contenir qu'une valeur)
[Pour un sudoku 9×9, 81 cellules, soit 81 colonnes pour la matrice]

— pour chaque ligne de la grille, une colonne pour chacune des valeurs devant figurer dans la ligne
[Pour un sudoku 9×9, 9 lignes, devant comporter 9 valeurs distinctes, soit 81 colonnes]

— pour chaque bloc de la grille, une colonne pour chacune des valeurs devant figurer dans le bloc
[Pour un sudoku 9×9, 9 blocs, devant comporter 9 valeurs distinctes, soit 81 colonnes]

— pour chaque ligne de la grille, une colonne pour chacune des valeurs devant figurer dans la ligne
[Pour un sudoku 9×9, 9 colonnes, devant comporter 9 valeurs distinctes, soit 81 colonnes]

Pour un sudoku 9×9, la matrice comportera donc $9 \times 9 \times 4 = 324$ colonnes. [3] fournit cette matrice sous la forme d'une table.

Chaque ligne de la matrice, représentant une valeur v donnée pour une cellule c donnée, aura un 1 dans la colonne correspondant à c, ainsi que dans les colonnes correspondant à v pour les bloc, colonne et ligne auxquels c appartient, et des 0 partout ailleurs.

3. Un pentomino est une forme constituée de 5 cases adjacentes (2 cases sont adjacentes si elles ont un bord en commun).

Références

[1] Richard M. Karp, « Reducibility Among Combinatorial Problems », dans Raymond E. Miller et James W. Thatcher, Complexity of Computer Computations, Plenum, 1972. Accessible en ligne : https ://people.eecs.berkeley.edu/ luca/cs172/karp.pdf

[2] Donald E. Knuth, « Dancing Links », dans Jim Davies ; Bill Roscoe et Jim Woodcock, Millennial Perspectives in Computer Science : Proceedings of the 1999 Oxford-Microsoft Symposium in Honour of Sir Tony Hoare, Palgrave, coll. « Cornerstones of Computing », 30 novembre 2000, 432 p. Voir page Moodle de l'UE.

[3] Robert M. Hanson, « Exact Cover Matrix », page d'un site web consacré à la résolution de sudokus. Accessible en ligne (le 8 octobre 2017) : http ://www.stolaf.edu/people/hansonr/sudoku/exactcovermatrix.htm.