Utilisation de codes canoniques pour résoudre efficacement le problème de génération de benzénoïdes en programmation par contraintes

Xiao Peng[1*] **Christine Solnon**[1]

[1] CITI, INSA Lyon / Inria, France
{xiao.peng,christine.solnon}@insa-lyon.fr

Résumé

Cet article est un résumé de [8].

Les benzénoïdes sont des molécules d'hydrocarbures dont les atomes de carbone forment des cycles hexagonaux. Un important problème de chimie concerne la génération de tous les benzénoïdes qui satisfont à certaines propriétés données [3]. Ce problème est appelé *Benzenoid Generation Problem* (BGP) dans [2]. Comme les benzénoïdes sont des grilles régulières à faces hexagonales, ils peuvent être représentés par des graphes hexagonaux de telle sorte qu'un sommet est associé à chaque face hexagonale et une arête à chaque paire de sommets correspondant à des faces adjacentes [1, 2]. Par exemple, nous présentons dans la Figure 1 un benzénoïde composé de cinq faces hexagonales et son graphe hexagonal associé.

Différents benzénoïdes peuvent être représentés par des graphes hexagonaux isomorphes. Par exemple, considérons le graphe hexagonal représenté dans la Figure 1. Le sous-graphe induit par c, d et e est isomorphe au sous-graphe induit par a, c et d alors que leurs molécules associées sont différentes car les hexagones c, d et e ne sont pas alignés alors que les hexagones a, c et d le sont. Pour surmonter ce problème, nous prenons en compte les directions des arêtes. Dans ce cas, le sous-graphe induit par c, d et e n'est plus isomorphe au sous-graphe induit par a, c et d car les arêtes (c, d) et (d, e) ont des directions différentes, alors que les arêtes (a, c) et (c, d) ont la même direction.

Un point clé pour une énumération efficace des graphes hexagonaux est d'être invariant aux rotations et aux symétries. Pour cela, nous proposons dans [8] de représenter les benzénoïdes par des codes canoniques.

*Papier doctorant : Xiao Peng[1] est auteur principal.

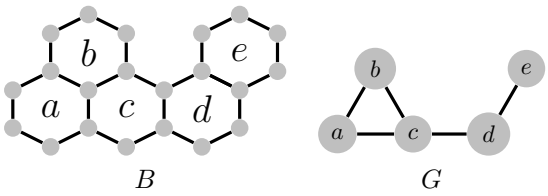

FIGURE 1 – Exemple d'un benzénoïde B et de son graphe hexagonal associé G.

Dans le cas général, la construction du code canonique d'un graphe est un problème qui n'est pas connu pour être dans $\mathcal{P}$ ni pour être $\mathcal{NP}$-complet (il est isomorphique-complet). Il existe des algorithmes efficaces tels que Nauty [6], mais ces algorithmes ont une complexité en temps exponentielle dans le pire des cas. Les codes canoniques sont largement utilisés dans les algorithmes de recherche de motifs fréquents dans des graphes tels que gSpan [9] ou Gaston [7]. Lorsque les graphes sont plongés dans un espace 2D, les codes canoniques peuvent être calculés en temps polynomial [5]. Ces codes canoniques peuvent être simplifiés si l'on considère des grilles dont toutes les faces sont des carrés [4]. Les codes canoniques définis dans [9, 7, 4] sont calculés en effectuant des parcours en profondeur d'abord (DFS) et en attribuant des numéros aux sommets en fonction de l'ordre de découverte.

Nous reprenons un principe similaire pour représenter les graphes hexagonaux, mais nous utilisons des parcours en largeur d'abord (BFS) au lieu de DFS. Cela nous permet d'une part d'élaguer l'espace de recherche, et d'autre part de définir nos codes en terme de contraintes, et donc d'utiliser la programmation par contraintes pour les énumérer. Nous pouvons alors facilement ajouter des contraintes afin de limiter l'énumé-

TABLE 1 – Résultats de BENZAI et CCODE pour BGP3 et BGP4. n est le nombre n de sommets, #i le nombre d'instances associées à n, et avg (resp. max et min) le temps moyen (resp. maximal et minimal) pour résoudre ces #i instances. Quand le temps excède 3600 secondes pour toutes les instances, nous reportons '-'.

	BGP3							BGP4						
n	#i	BENZAI			CCODE			#i	BENZAI			CCODE		
		avg	max	min	avg	max	min		avg	max	min	avg	max	min
2	2	0.02	0.02	0.02	0.01	0.01	0.00	0	—	—	—	—	—	—
3	6	0.02	0.03	0.01	0.00	0.01	0.00	0	—	—	—	—	—	—
4	7	0.03	0.04	0.02	0.00	0.02	0.00	3	0.07	0.16	0.02	0.01	0.03	0.00
5	8	0.05	0.07	0.01	0.01	0.07	0.00	11	0.06	0.20	0.04	0.01	0.06	0.00
6	8	0.38	0.43	0.23	0.03	0.15	0.00	23	1.41	2.51	0.93	0.02	0.16	0.01
7	8	0.95	1.06	0.79	0.13	0.35	0.01	29	3.78	5.42	2.30	0.10	0.50	0.01
8	8	20.33	23.84	13.94	0.73	1.61	0.04	34	65.06	83.21	50.28	0.49	1.69	0.03
9	8	306.30	399.25	75.05	3.94	10.52	0.23	35	400.51	574.15	226.87	2.94	10.85	0.20
10	8	-	-	-	34.60	69.75	2.20	36	-	-	-	24.19	71.03	1.61

ration à des graphes hexagonaux satisfaisant certaines propriétés (par exemple, l'absence de cliques d'ordre trois, ou l'inclusion de certains motifs). Nous introduisons une contrainte globale, permettant d'assurer que les codes générés sont bien canoniques, et donc d'éviter d'énumérer des graphes isomorphes. Nous introduisons également des modèles CP permettant de contraindre les graphes énumérés à contenir des motifs donnés.

Nous avons comparé les temps de notre approche (notée CCODE) avec ceux de l'approche de [2] (notée BENZAI), basée également sur la programmation par contraintes, mais n'utilisant pas de codes canoniques pour représenter les graphes. Nous donnons dans la table 1 les résultats pour deux BGP : BGP3 (resp. BGP4) vise à énumérer tous les graphes hexagonaux contenant un motif donné (resp. deux motifs donnés). Les motifs donnés sont les mêmes que ceux de [2]. Nous constatons que notre approche est toujours plus rapide. Quand $n = 9$, elle est 78 (resp. 136) fois plus rapide que BENZAI pour BGP3 (resp. BGP4). Quand $n = 10$, BENZAI ne résout aucune des 8 (resp. 36) instances de BGP3 (resp. BGP4) dans la limite de temps d'une heure tandis que CCODE ne met jamais plus de 71s.

Références

[1] Gunnar BRINKMANN, Gilles CAPOROSSI et Pierre HANSEN : A constructive enumeration of fusenes and benzenoids. *J. Algorithms*, 45(2):155–166, 2002.

[2] Yannick CARISSAN, Denis HAGEBAUM-REIGNIER, Nicolas PRCOVIC, Cyril TERRIOUX et Adrien VARET : Using constraint programming to generate benzenoid structures in theoretical chemistry. *In* Helmut SIMONIS, éditeur : *Principles and Practice of Constraint Programming - 26th International Conference, CP*, volume 12333 de *Lecture Notes in Computer Science*, pages 690–706. Springer, 2020.

[3] S. J. CYVIN, J. BRUNVOLL et B. N. CYVIN : Search for concealed non-kekulean benzenoids and coronoids. *Journal of Chemical Information and Computer Sciences*, 29(4):236–244, 1989.

[4] Romain DEVILLE, Élisa FROMONT, Baptiste JEUDY et Christine SOLNON : Grima : A grid mining algorithm for bag-of-grid-based classification. *In* Antonio ROBLES-KELLY, Marco LOOG, Battista BIGGIO, Francisco ESCOLANO et Richard C. WILSON, éditeurs : *Structural, Syntactic, and Statistical Pattern Recognition - Joint IAPR International Workshop, S+SSPR, Proceedings*, volume 10029 de *Lecture Notes in Computer Science*, pages 132–142, 2016.

[5] Stéphane GOSSELIN, Guillaume DAMIAND et Christine SOLNON : Efficient search of combinatorial maps using signatures. *Theor. Comput. Sci.*, 412(15):1392–1405, 2011.

[6] Brendan D. MCKAY et Adolfo PIPERNO : Practical graph isomorphism, II. *J. Symb. Comput.*, 60:94–112, 2014.

[7] Siegfried NIJSSEN et Joost N. KOK : The gaston tool for frequent subgraph mining. *In Proceedings of the 2nd International Workshop on Graph-Based Tools, GraBaTs 2004, Rome, Italy, October 2, 2004*, volume 127 de *Electronic Notes in Theoretical Computer Science*, pages 77–87. Elsevier, 2004.

[8] X. PENG et C. SOLNON : Using Canonical Codes to Efficiently Solve the Benzenoid Generation Problem with Constraint Programming. *In CP 2023 - 29th International Conference on Principles and Practice of Constraint Programming*, pages 28 :1–28 :17. LIPIcs, 2023.

[9] Xifeng YAN et Jiawei HAN : gspan : graph-based substructure pattern mining. *In 2002 IEEE International Conference on Data Mining, 2002. Proceedings.*, pages 721–724, 2002.