Cyclic-bandwith problem
Programmation par contraintes

Nicolas Compère, Tobias Sanvoisin

Master 1 ORO - 2023

Nantes Université

Résumé

Ce papier explore 3 modèles de programmation par contraintes pour modéliser la **Cyclic-bandwidth problem** ainsi que les cassages de symétries, la recherche dichotomique de paramètre et l'implémentation en `Python` à l'aide des bibliothèques `PyCSP3` et `PySAT`.

Table des matières

1 Introduction

Dans le cadre du module de Programmation par Contraintes du Master 1 ORO, nous avions à modéliser de 3 manières différentes la "Cyclic-bandwidth problem". Dans ce rapport, nous allons revenir sur ce qu'est le problème, les 3 différents modèles, les solutions apportées au problème de solutions symétriques ainsi que l'implémentation de ces modèles en `Python` et les résultats de leurs résolution.

Certaines parties de ce travail ont étés réalisés en collaboration avec les autres groupes de la promotion et s'inspirent de travaux réalisés par des promotions antérieurs.

2 Cyclic-bandwidth problem

Le "Cyclic-bandwidth problem" possède de nombreuses applications diverses, incluant le calcul de matrices creuses, la théorie des codes, les structures de données et l'intégration à très grande échelle (*VLSI* pour *Very-Large-Scale Integration* en anglais) des circuits intégrés [1].

Le problème est définie comme suit :
Soit un graphe $G = (V, E)$ avec V l'ensemble des sommets et E l'ensemble des arrêtes. Un étiquetage f d'un graphe G est une fonction bijective de V dans $\{1, ..., n\}$.
Soit $(u, v) \in E$ une arrête de G. La distance entre u et v est définie par $|f(v) - f(u)|$. La cyclic-bandwidth du graphe G par rapport à l'étiquetage de f est donnée par

$$CB(G, f) = \max_{(u,v)\in E}\{\min\{|f(v) - f(u)|, n - |f(v) - f(u)|\}\}$$

La cyclic-bandwidth problem consiste à trouver l'étiquetage g tel que

$$CB(G, g) = \min_{f\in\mathcal{E}}\{CB(G, f)\}$$

avec $\mathcal{E}$ l'ensemble des étiquetages.

3 Modèles

Nous allons voir 3 modèles différents dans cette section.

3.1 Modèle 1

Le modèle 1 est en variables de domaine fini entières et comporte une fonction objectif à minimiser.

Données : Un graphe $G = (V, E)$ avec $n = |V|$

Variables : x_i avec $i \in \{1, ..., n\}$

Domaine des variables : $x_i \in [1, ...n]^n$

Contraintes : `AllDifferent`(x_i), $\forall i \in \{1, ..., n\}$

Fonction objectif : $\min\left\{\max_{(u,v)\in E}\{\min\{|x_i - x_j|, n - |x_i - x_j|\}\}\right\}$

Son implémentation se trouve dans le fichier `model1.py` et a été réalisé avec le package `PyCSP3`.

3.2 Modèle 2

Le modèle 2 est en variables de domaine fini entières également, sans contraintes arithmétiques et axé sur la satisfaction (avec une valeur k donnée).

Données : — Un graphe $G = (V, E)$,

— $n = |V|$,

— $k \in \{1, ...n\}$,

— $P = \{(i, j), \min(|j - i|, n - |j - i|) \leq k \text{ et } i \neq j$

Variables : x_i avec $i \in \{1, ..., n\}$

Domaine des variables : $x_i \in [1, ...n]^n$

Contraintes : — `AllDifferent`(x_i), $\forall i \in \{1, ..., n\}$,

— $\forall (i, j) \in E$, $(x_i, x_j) \in P$

Son implémentation se trouve dans le fichier `model2.py` et a également été réalisé avec le package `PyCSP3`.

3.3 Modèle 3

Le troisième modèle est équivalent au modèle 2 mais il est écrit avec des variables booléennes et des clauses CNF (*Conjunctive Normal Form*, ou *Forme Normale Conjonctive* en français).

Données : — Un graphe $G = (V, E)$,

— $n = |V|$,

— $k \in \{1, ..., n\}$,

— P un ensemble de booléens tel que :

$$b_{ij} = \begin{cases} \textit{Vrai} \text{ si } \min(|j - i|, n - |j - i|) \leq k, \forall (i, j) \in \{1, ..., n\}^2 \\ \textit{Faux} \text{ sinon} \end{cases}$$

Variables : X une matrice de booléens tel que $x_{ij} =$ Vrai si le sommet i est étiqueté par j

Domaine des variables : $x_{ij} \in \{\text{Vrai, Faux}\}$, $\forall (i, j) \in \{1, ..., n\}^2$

Contraintes : — $\bigwedge_{i=1}^{n} \bigvee_{j=1}^{n} (x_{ij})$

— $\bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} \bigwedge_{k=1, k \neq i}^{n} (\neg x_{ij} \vee \neg x_{kj})$

— $\bigwedge_{(u,v) \in E} \bigwedge_{i=1}^{n} \bigwedge_{j=1}^{n} (\neg x_{ui} \vee \neg x_{vj} \vee b_{ij})$

Son implémentation se trouve dans le fichier `model3.py` et a été réalisé avec le package `PySAT`.

4 Symétrie

Pour chaque modèle, nous avons des possibilités de symétries qui viennent affecter les performances des solveurs.
Soit un étiquetage $\mathcal{E}$, un n-uplet d'entier de 1 à n avec $n = |V|$. Une symétrie $s(\mathcal{E})$ donne les mêmes résultats pour la cyclic-bandwidth que $\mathcal{E}$. Nous avons observé deux type de symétries dans le cadre de ce projet qui sont la symétrie centrale et la symétrie axiale.

L'implémentation des modèles avec gestion des symétries se trouvent dans les fichiers `model1+.py`, `model2+.py` et `model3+.py`.

4.1 Symétrie centrale

La première symétrie que nous allons voir est la symétrie centrale. Deux étiquettes sont symétriques par rotation lorsque l'on peut obtenir l'un en faisant tourner l'autre dans un sens donné.

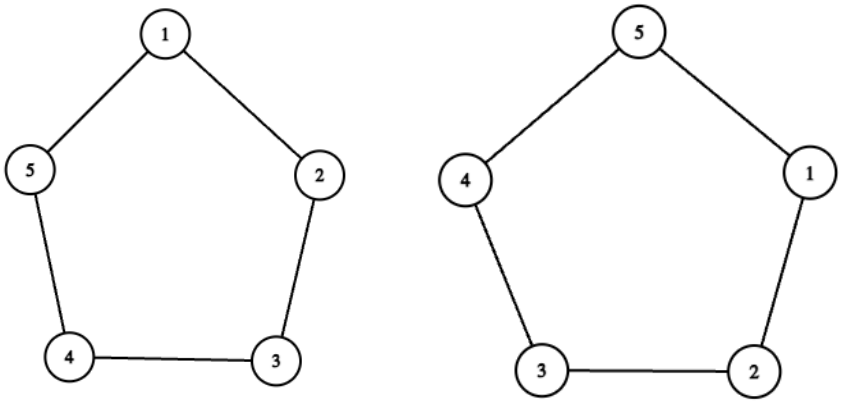

FIGURE 1 – Exemple de symétrie centrale

Pour remédier à ce problème, il nous suffit de fixer l'étiquetage d'un des sommets. On peut par exemple étiqueter le sommet 1 par l'étiquetage 1 avec $\mathcal{E}(1) = 1$.

4.2 Symétrie axiale

La deuxième symétrie que nous avons observé est la symétrie axiale. Deux étiquetages sont symétriques selon un axe si on peut obtenir l'un en faisant tourner l'autre par rapport à un axe qui passe par un sommet du graphe. Dans l'exemple ci-dessous, les deux graphes sont symétrique par rapport à un axe passant par le sommet 1.

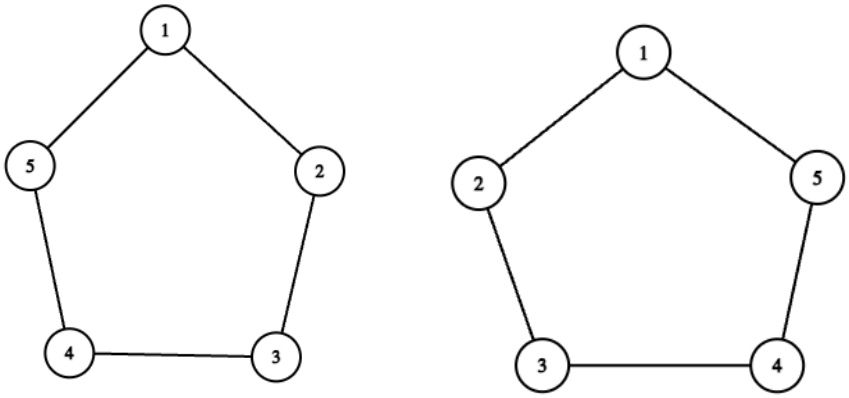

FIGURE 2 – Exemple de symétrie axiale

Afin de casser cette symétrie, on peut donner un ordre à l'étiquetage de deux sommets de l'arbre. Par exemple, on peut fixer l'étiquetage du sommet 2 comme inférieur à tout les autres étiquetages avec $\mathcal{E}(2) < \mathcal{E}(n)$.

5 Recherche dichotomique

Dans le cas des modèles 2 et 3, nous vérifions s'il existe un étiquetage qui fournit une valeur de la cyclic-bandwidth égale à k, qui est un paramètre en entrée. Afin de trouver le paramètre k le plus petit possible, nous avons mis en place un algorithme de recherche dichotomique.

Nous avons choisi l'encadrement

$$\left\lceil \frac{\Delta(G)}{2} \right\rceil \leq k \leq \left\lfloor \frac{n}{2} \right\rfloor$$

où $\Delta(G)$ est le degré et n le nombre de sommets du graphe G.

Son implémentation se trouve dans le fichier `dichotomie.py`.

6 Résultats

Les tests ont été réalisés sur une machine qui possède ces caractéristiques :

Système d'exploitation macOS Ventura 13.3.1

Processeur Apple M2 8 coeurs (4 performance et 4 efficacité)

Carte Graphique Apple M2 intégré 10 coeurs

RAM 16 Go LPDDR5

Les données utilisés sont des instances issues du benchmark Harwell-Boeing 113 (*HB113*).

Instances	Nombre de sommets	Nombres d'arêtes
curtis54	54	124
gent113	104	549
lund_b	147	1147
lns__131	123	275
bcsstk01	48	176
bcspwr01	39	46
can__144	144	576
steam3	80	424
will57	57	127
ibm32	32	90
bcspwr03	118	179
pores_1	30	103
dwt__234	117	162
west0156	156	371
ash85	85	219
lund_a	147	1151
can__161	161	608
bcsstk22	110	254
west0167	167	489

TABLE 1 – Instances sélectionnées pour les tests

Le solveur utilisé pour les modèles 1 et 2 est `ACE` et le solveur pour le modèle 3 est `Glucose4`.

Le code fournit rencontre des problèmes lors de l'exécution sur toutes les instances avec la recherche dichotomique, les temps suivants sont donc issus de l'exécution d'un programme similaire fonctionnel contenant la même implémentation pour les modèles.

Instance	`model1.py` CPU (s)	`model2.py` CPU (s)	k	`model3.py` CPU (s)	k
curtis54	1.17	1.33	6	8.93	6
gent113	1.57	1.73	13	100.56	13
lund_b	1.97	2.12	6	339.78	6
lns__131	1.96	2.11	4	93.14	4
bcsstk01	1.18	1.19	5	5.42	5
bcspwr01	1.18	1.20	2	2.06	2
can__144	1.98	1.98	7	209.26	7
steam3	1.58	1.57	6	46.23	6
will57	1.59	1.57	4	8.41	4
ibm32	1.18	1.20	4	2.05	4
bcspwr03	1.99	1.97	1	54.59	1
pores_1	1.17	1.18	2	2.03	2
dwt__234	1.98	1.98	2	41.24	2
west0156	1.99	1.99	5	120.19	5
ash85	1.53	1.53	3	28.18	3
lund_a	1.76	1.75	6	330.77	6
can__161	1.74	1.72	4	331.90	4
bcsstk22	1.54	1.53	2	63.57	2
west0167	1.54	1.54	10	344.78	10

TABLE 2 – Résulatts pour les modèles sans gestion des symétries

Instance	`model1+.py` CPU (s)	`model2+.py` CPU (s)	k	`model3+.py` CPU (s)	k
curtis54	1.19	1.19	6	5.40	6
gent113	1.61	1.60	13	95.47	13
lund_b	2.3	1.98	6	334.01	6
lns__131	2.1	1.99	4	86.92	4
bcsstk01	1.19	1.19	5	5.33	5
bcspwr01	1.17	1.16	2	2.5	2
can__144	1.98	1.98	7	247.72	7
steam3	1.59	1.60	6	45.73	6
will57	1.56	1.58	4	8.7	4
ibm32	1.19	1.18	4	2.2	4
bcspwr03	1.95	1.97	1	53.25	1
pores_1	1.18	1.17	2	2.1	2
dwt__234	1.99	1.98	2	46.83	2
west0156	1.98	1.96	5	160.41	5
ash85	1.53	1.53	3	27.26	3
lund_a	1.74	1.75	6	410.26	6
can__161	1.74	1.76	4	328.40	4
bcsstk22	1.51	1.54	2	60.49	2
west0167	1.51	1.51	10	341.56	10

TABLE 3 – Résultats pour les modèles avec gestion des symétries

Nous constatons que les résultats des modèles 1 et 2 sont très proches, que ce soit dans leurs versions avec et sans symétries. Les valeurs de k obtenus par la recherche dichotomique sont identiques entre les modèles 2, 2+, 3 et 3+.

La gestion des symétrie ne semble pas influencer les résultats sur les temps de calculs.

Les résultats obtenus ne semble pas corrects du point de vue des temps d'exécutions. Le modèle 1, qui utilise une fonction objectif, est supposé en théorie être moins efficace que les modèle 2 et 3 qui sont des modèles axés sur la satisfaction. Les temps du modèle 3 semble plus proche de ce qui est attendu, mais diffère grandement des temps du modèle 2 alors qu'ils devraient êtres similaires. La source de ces anomalies provient très probablement de la fonction de recherche dichotomique mais elle n'as pas été identifiée.

7 Conclusion

Malgré les essais, des problèmes persistent avec l'implémentation de la Cyclic-bandwidth problem. La plus grande difficulté réside dans le fait de lancer correctement les modèles avec la recherche dichotomique sur un lot d'instances. En effet, les modules `PyCSP3` et `PySAT` permettent de coder facilement les modèles définis en langage `Python`, mais limitent les manières de lancer les résolutions sur les instances, ce qui rend complexe sont intégration dans un programme et le manque de documentation ne facilite pas le processus.

Malgré cela, le projet fut intéressant car il permet de voir différentes façon de modéliser un même problème ainsi que la façon d'éviter d'avoir des solutions symétriques redondantes. Il est également intéressant de constater ce qui se fait dans la façon de coder ces modèles langage informatique.

Références

[1] Yixun LIN. "The cyclic bandwidth problem". In : *Systems Science and Mathematical Sciences* 7 (jan. 1994).