ASP : Answer Set Programming

Carito Guziolowski
Département MATH-INFO, LS2N, École Centrale de Nantes

10 mars 2022

TP2 : ASP

1 Circuit Hamiltonien

Considérer le problème de trouver un circuit Hamiltonien dans un graphe G, c'est-à-dire un chemin fermé qui visite tous les sommets du graphe exactement une fois. Un circuit Hamiltonien d'un graphe G peut être vu comme un sous-graphe C qui a le même ensemble de sommets V que G et qui en plus satisfait deux contraintes :

1. Chaque sommet en C est adjacent à exactement deux autres sommets.
2. Chaque sommet en C est accesible par un sommet donné u_0.

1.1 Question 3.1

Écrire un PL `ham.lp` pour trouver le(s) circuit(s) Hamiltonien. Justifier les règles utilisées et écrire le résultat généré avec le graphe représenté par les faits suivants (`graph.lp`), disponible sur Hippocampus.

```
% Sommets
node(1..6).
% Arcs (orientés)
edge(1,2; 1,3; 1,4).  edge(2,4; 2,5; 2,6).  edge(3,1; 3,4; 3,5).
edge(4,1; 4,2).    edge(5,3; 5,4; 5,6).  edge(6,2; 6,3; 6,5).
```

Optionnellement, vous pouvez utiliser le guide suivant pour construire le PL :

- **Génerer.** Générer un prédicat (e.g. `in(X,Y)`) qui représente un élement candidat du circuit Hamiltonien à partir des faits du graphe. Écrire une règle qui permet de vérifier la contrainte 1 pour le prédicat `in/2`.
- **Définir.** Écrire une règle pour introduire la notion *être accesible* pour un sommet du graphe. Vous pouvez utiliser un prédicat `accesible(X)`.
- **Tester.** Écrire une règle pour vérifier la contrainte 2. Le sommet u_0 est arbitraire, choisir 3 par exemple.
- **Affichage de la sortie.** Montrez seulement les prédicats `in/2` dans la sortie de votre PL à l'aide des instructions `#show.`

1.2 Question 3.2

Le problème du voyageur de commerce consiste, étant donné un ensemble de villes séparées par des distances données, à trouver le plus court chemin qui relie toutes les villes. On peut le voir comme une variante du problème de circuit Hamiltonien dans laquelle nous rajoutons des coûts aux arcs du graphe. Étant données les coûts suivants associés (`costs.lp`) au graphe de la Question 3.1 :

```
1 cost(1,2,2).  cost(1,3,3).  cost(1,4,1).
2 cost(2,4,2).  cost(2,5,2).  cost(2,6,4).
3 cost(3,1,3).  cost(3,4,2).  cost(3,5,2).
4 cost(4,1,1).  cost(4,2,2).
5 cost(5,3,2).  cost(5,4,2).  cost(5,6,1).
6 cost(6,2,4).  cost(6,3,3).  cost(6,5,1).
```

Écrire le PL (`min.lp`) pour trouver le circuit Hamiltonien à coût minimal, et trouver le coût minimal k associé en éxécutant :

(1) `clingo ham.lp min.lp costs.lp graph.lp`

Vous pouvez comparer vos réponses à la solution donnée pour ce probrème dans le serveur pédagogique.

La réponse à cette instruction permet de garantir une optimisation globale, par contre ils peuvent avoir d'autres circuits Hamiltoniens optimaux qui ne sont pas explorés. Pour les trouver il faudra exécuter l'instruction suivante:

(2)`clingo ham.lp min.lp costs.lp graph.lp --opt-bound=k --opt-mode=enum 0`

Où l'entier k a été obtenu après l'éxécution de (1). Existe-t-il plusieurs solutions optimales pour le graphe considéré ?

2 Affectation de rapporteurs

Vous êtes chargés d'affecter les rapporteurs aux n papiers soumis dans une conférence. Chaque papier soumis doit être affecté à des rapporteurs parmi les m membres du Comité du Programme (CP). Les auteurs de chaque papier ont signalé leurs conflits d'intérêt par rapport au fait que leur papier soit révisé par un rapporteur donnée. Les membres du CP ont tous lu les résumés des papiers et chacun a donné les numéros des papiers qu'il ou elle peut réviser. Notons A_i le sous-ensemble des papiers $\{1, \ldots, n\}$ qui peut être révisé par le rapporteur i $(1 \leq i \leq m)$. Vous devez sélectionner, pour chaque i, un sous-ensemble X_i des papiers qui doit être affecté au rapporteur i. Pour cela il faut satisfaire les 4 conditions suivantes :

1. X_i doit être un sous-ensemble de A_i.
2. La cardinalité de X_i doit satisfaire : $u \leq |X_i| \leq l$.
3. Chaque papier doit être affecté a exactement k rapporteurs.

4. Aucun papier ne doit être affecté à un rapporteur qui a été signalé par l'auteur du papier dans sa liste de conflits d'intérêt.

Écrivez le PL `rapporteurs.lp` en ASP pour résoudre ce problème ou pour déterminer s'il est impossible de le résoudre. Testez son fonctionnement pour l'instance `instance_rapporteurs.lp` avec $k = 1, u = 2, l = 2$, et comparez vos résultats avec la solution `solution_rapporteurs.txt`; les 2 fichiers sont disponibles sur le serveur pédagogique. Voici un exemple des données d'entrée pour ce problème :

```
1 rapporteur(r1). rapporteur(r2). ... rapporteur(rm).
2 papier(p1). paier(p2)... papier(pn).
3 affinite(r1,p1). ...
4 % Conflits d'intérêt
5 coi(r1,p3). ...
```

3 Nombres Schur

Un ensemble A d'entiers est dit *sans somme* s'il n'existe pas deux élements $x, y \in A$ tels que leur somme $x + y$ appartient à A. Par exemple, l'ensemble $\{5, \ldots, 9\}$ est *sans somme*; l'ensemble $\{4, \ldots, 9\}$ ne l'est pas : $(4 + 4 = 8, 4 + 5 = 9)$. Le plus grand nombre n tel que l'ensemble $\{1, 2, \ldots, n\}$ peut être divisé en k sous-ensembles *sans somme* est le nombre de Schur $S(k)$. Écrire un programme logique en ASP `schur.lp` pour estimer $S(k)$: étant donnés n et k, trouver s'il est possible de diviser l'ensemble $\{1, 2, \ldots, n\}$ en k sous ensembles *sans somme*.

Exercice 3.1. Combien de modèles stables existent pour $k = 2$ et $n = 4$?

Exercice 3.2. Nous appelons un ensemble d'entiers X *presque sans somme* si la somme de deux éléments différents de X n'appartient jamais à X. Par exemple, l'ensemble $\{1, 2, 4\}$ est *presque sans somme*. Si nous voulons décrire les partitions de $\{1, \ldots, n\}$ dans k ensembles *presque sans somme*, comment modifier le programme `schur.lp` pour satisfaire cette contrainte ?

♢

Si nous exécutons le programme `schur.lp` pour $k = 3$ et diverses valeurs de n alors nous trouverons que le n le plus grand pour lequel existent des modèles stables est 13. En d'autres termes, $S(3) = 13$. D'une manière similaire, nous pouvons déterminer que $S(4) = 44$. Le temps d'exécution de `clingo` pour $k = 4$ et $n = 45$ peut être grande, et il y a une astuce utile qui peut accélérer le processus. Pour cela il faudra installer clingo v4. A partir de cette version clingo supporte le *multi-threading* qui permet d'utiliser des différentes stratégies de recherche (pour parcourir l'arbre de solutions) sur différents *threads* en parallèle. Par exemple, si nous exécutons la commande avec `clingo v4`

```
clingo --const k=4 --const n=45 schur.lp -t4
```

la sortie rassemble à

```
clingo version 4.5.4
reading from schur.lp
solving...
USATISFIABLE
Models :  0
Calls :  1
Time :  61.631s (Solving:  61.61s 1st Model:  0.00s Unsat:
61.61s)
CPU Time :  238.250s
Threads :  4 (Winner:  1)
```

L'information affichée après *Winner* montre quel thread $0, 1, 2, 3$ est arrivé à l'objectif le premier.

Exercise 3.2. Utilisez `clingo v4` pour montrer que $S(5) \geq 130$

♢

La valeur exacte de $S(5)$ n'est pas connue.

4 Contraintes pour la coloration d'un graphe

Pour un graphe orienté et signé, nous disposons d'un coloriage partiel de ses sommets (voir Figure 1). Les signes admisibles des arcs du graphe sont "$+$" et "$-$". Les couleurs admisibles pour les noeuds du graphe sont *rouge* (down, $-$), et *vert* (up, $+$).

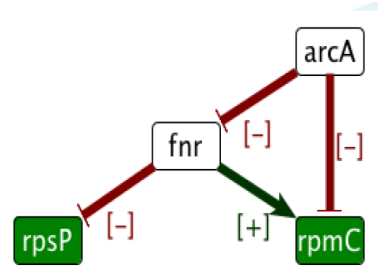

Figure 1: Graphe orienté et signé avec un coloriage partiel de ses noeuds.

Nous voudrions raisonner sur ce graphe, avec la règle de consistance suivante :

Contrainte 1 : coloriage consistant. *Le signe (couleur verte ou rouge respectivement) d'un noeud ayant prédecesseurs doit être égale au signe de l'influence d'au moins un de ses prédecesseurs directs dans le graphe.*

L'influence d'un noeud i du graphe sur son successeur j dans le graphe est égal au produit de son signe, *signe(i)*, et de l'arc (i, j), i.e. `signe(i) * signe(`$arc(i,j)$`)`

Dans la Figure 2 nous montrons des exemples de graphes avec un coloriage inconsistante par rapport à la contrainte 1.

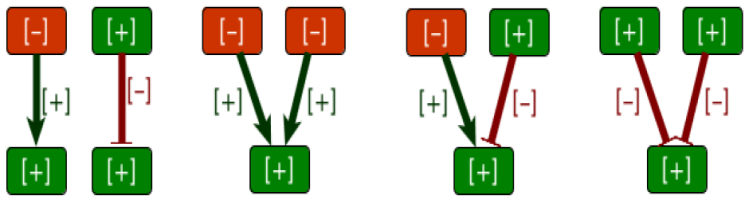

Figure 2: 5 graphes orientés et signés avec des coloriages inconsistantes.

4.1 Question 1.1

Écrivez les faits du graphe de la Figure 1 plus une coloration donnée sur `rpsP` (ci-dessous) dans un fichier `graph_col.lp`

```
1 %Donnees : un graphe avec une coloration donne
2 signe(down;up).
3 vertex(rpsP;fnr;arcA;rpmC).
4 observedV(rpsP,up).
5 observedE(fnr,rpsP,down). observedE(fnr,rpmC,up).
6 observedE(arcA,fnr,down). observedE(arcA,rpmC,down).
```

Reprendre la solution du TD2 (ci-dessous) qui permet de vérifier la contrainte 1 pour tous les noeuds d'un graphe par rapport à une coloration de noeuds donnée. Écrivez le PL dans `signs.lp`.

```
1 %Generer les solutions candidates
2  {labelV(I,S):signe(S)}=1 :- vertex(I).
3 labelV(I,S) :- observedV(I,S).
4 %Definir le predicat receive pour tester la contrainte 1
5 {receive(I,S) : signe(S)} = 1 :- vertex(I).
6 %Tester les candidats pour que la contrainte 1 soit satisfaite
7 receive(I,up) :- observedE(J,I,S), labelV(J,S).
8 receive(I,down) :- observedE(J,I,S), labelV(J,T), S!=T.
9 :- labelV(I,S), not receive(I,S).
```

Éxécuter le PL `signs.lp` avec l'instance `graph_col.lp` avec la commande `clingo graph_col.lp signs.lp` et répondre aux questions suivantes :

1. Quels sont les coloriages proposés pour les sommets du graphe initialement non-colorés et combien sont ils ?

2. Générer un 2ème fichier `graph_col2.lp` qui est une copie de `graph_col.lp` plus le fait `observedV(rpmC,up)`. Comment change la sortie du programme `signs.lp` avec

l'instance `graph_col2.lp` ? Justifiez cette réponse du programme et préciser si possible la ligne du code `signs.lp` qui génere cette réponse.

3. Pourquoi l'instance `graph_col.lp` devient `UNSATISFIABLE` par rapport au `signs.lp` lorsqu'on commente la ligne 5 de `signs.lp` ?

4. Une façon de permettre qu'un noeud n du graphe échappe à la *Contrainte 1* est d'ajouter un prédicat `echape(n)` dans l'instance `graph_col.lp` et ensuite modifier la contrainte (integrity constraint) du programme `signs.lp`. Écrivez ce PL dans le fichier `signs_mod.pl`.

5. Enlevez la contrainte 5 (`1{receive(I,S):signe(S)}1 :- vertex(I)`) du `signs_mod.pl`. Comment on modifie l'instance `graph_col2.lp` avec le prédicat `echape/1` pour qu'on obtient des réponses cohérentes ?

4.2 Question 1.2

Récuperer du serveur pédagogique le fichier correspondant au réseau biologique de l'opéron lactose : `graphe2.lp` (Figure 3) et les 4 jeux de coloriages des noeuds dans ce graphe : fichiers `obs1.lp` à `obs4.lp`, correspondant à des résultats expérimentales sur l'expréssion de certains molécules du graphe.

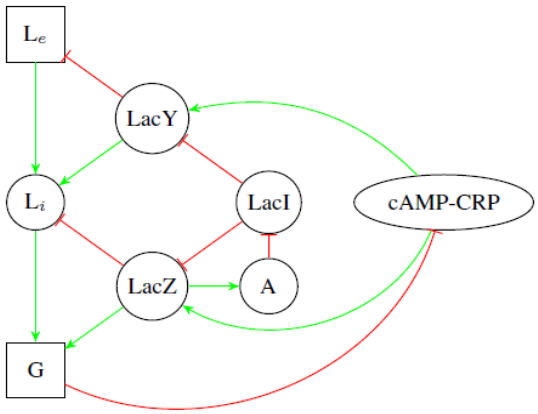

Figure 3: Graphe orienté et signé de l'opéron lactose de la bactérie Escherichia coli. Les arcs verts (resp. rouges) correspondent a des arcs observés à `up` (resp. `down`).

- Rajouter dans le `graphe2.lp` le fait que les noeuds le et g ne doivent pas être expliqués par la contrainte 1.

- Tester la consistance du `graphe2.lp` modifié par rapport aux 4 coloriages avec le PL `signs_mod.lp` . Quels sont les cas inconsistantes ? Quelle est la raison de l'inconsistance ? Quelle règle du PL n'est pas satisfaite ? Comparez vos réponses par rapport à la sortie souhaitée donnée dans le serveur pédagogique.

- **Bonus** Proposer une implémentation pour récupérer les zones (noeuds et arcs) inconsistantes. L'idée peut être décrite en mots.