Algorithmes du Monde Réel
Projet

Partie 1 - Réductions

Professeurs : M. Zeitoun, A. Muscholl, C. Gavoille

Master 2 Informatique

Ludovic Brochard, Fabien Kuntz, Benoît Védrenne

Table des matières

1 Introduction

1.1 Projet

L'objectif du projet est de réaliser un programme résolvant des problèmes *NP-complets* sur des graphes non orientés.

Pour ce faire, étant donné que le programme *"Minisat"*[1] sait résoudre "efficacement" des instances de *SAT*, nous utilisons les réductions des problèmes qui nous intéressent au problème *SAT*.

1.2 Organisation du rapport

Étant donné que le rapport doit être court, nous n'aurons malheureusement pas la possibilité d'expliquer la réalisation de notre projet dans les détails. Aussi, nous laisserons une partie d'annexes où se trouveront plusieurs exemples intéressants que nous n'avons pas mis dans le rapport.

Nous allons donc commencer par expliquer nos choix de programmation, puis allons donner les propriétés des problèmes qui nous ont permis de les réduire à *SAT*. Nous vous présenterons ensuite quelques tests et résultats de notre programme, pour finir par un bilan de notre projet.

2 Choix

Dans cette partie, nous allons présenter quelques uns des choix que nous avons faits concernant notre programme.

2.1 Langage

Nous avions le choix de faire ce projet en *C* ou en *C++*. Nous avons choisi *C++* pour la simple raison que des structures de données intéressantes (notamment *vector*) sont déjà présentes dans ce langage alors qu'elles ne le sont pas dans le langage *C*.

2.2 Structure de données

Dans ce projet, nous avions à faire le choix d'une structure de données pour représenter les graphes. Nous avons suivi les conseils en décidant d'utiliser des listes d'adjacence.

Plusieurs raisons nous ont poussés à utiliser les listes d'adjacence plutôt que d'autres structures[2] :

- La liste d'adjacence est une des structures qui prend le moins de place en mémoire car elle ne stocke que les sommets voisins (contrairement à une matrice d'adjacence par exemple).
- La liste d'adjacence est très efficace lorsqu'on a à considérer la liste des voisins des sommets (c'est son essence même), ce qui est le cas pour nos réductions.
- La liste d'adjacence est évidemment plus efficace que n'importe quelle structure d'incidence lorsqu'il s'agit de considérer les sommets plutôt que les arêtes.

2.3 Dépendances et Relations

Le programme est simple d'utilisation puisqu'il suffit de le lancer avec un fichier d'entrée de type graphe issu du programme *gengraph* de *C. Gavoille*[3] ainsi qu'avec le numéro du problème et éventuellement des paramètres propres au problème.

Le programme se déroule de la manière suivante :

- Le fichier de graphe est lu à partir du fichier de graphe fourni.
- Selon le problème sélectionné, on lance la réduction correspondante.
- Une formulation en clauses de type *CNF* est créée et envoyée au *SATSolver Minisat*.
- *Minisat* crée une solution (si cela est possible).
- La solution fournie est analysée afin de pouvoir être traitée. De cette manière, nous connaissons la solution au problème voulu sur le graphe passé en paramètre.

[1] http://minisat.se/

[2] Il n'y a ici, pas de bon choix à proprement parler, le mieux serait de faire au cas par cas (nous en parlerons dans le bilan du rapport).

[3] http://dept-info.labri.fr/~gavoille/gengraph.c

En décrivant l'exécution du programme de cette façon, il est aisé de voir apparaître les relations entre les modules de notre programme.

La fonction principale *main*, contenue dans *Solve.cpp*, joue le rôle du tri des arguments et de construction du graphe. Ensuite, est appelée la réduction voulue, contenue dans les fichiers portant le nom d'un problème (ou leur abréviation). Une fois la réduction effectuée par ce fichier, celui-ci appelle un "parser" pour *Minisat* (dans *MinisatBuilder.cpp*) qui gère le fait d'écrire un fichier d'entrée à *Minisat*, de lancer le *SAT-Solver* et d'analyser sa sortie. Une fois la sortie analysée, une assignation des variables est donnée. Celle-ci nous permet alors de fournir l'ensemble des arêtes ou sommets nécessaire définissant une solution du problème donné sur le graphe donné.

Voici un graphe représentant simplement les relations entre les différentes parties du programme (fig.1 page 2).

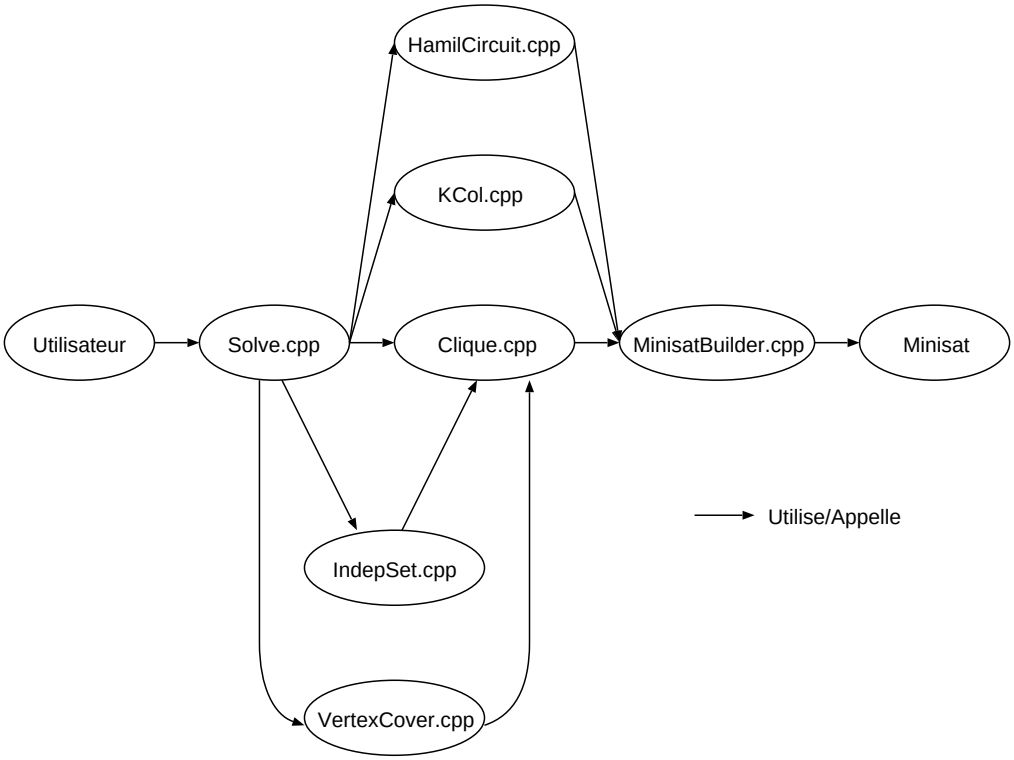

FIG. 1 – Relations entre les différentes parties du programme

2.4 Extension des fichiers

Étant donné que nous générons des fichiers avec notre programme, nous avons choisi des extensions différentes selon le but du fichier.

En effet nous avons introduit trois extensions différentes :

- *graph.gin* ⇝ Fichier graphe de type *gengraph* passé en paramètre par l'utilisateur au programme.
- *graph.sat* ⇝ Fichier qui contient la formule de type *CNF*[4] générée par notre programme à partir du choix du problème et du graphe.
- *graph.sol* ⇝ Fichier qui contient la solution retournée par *Minisat*.

Nous avons fait le choix de laisser ces fichiers sur le disque après exécution du programme, dans le cas où l'utilisateur souhaiterait y jeter un œil[5].

[4] *Conjunctive Normal Form*

[5] Les utilisateurs que nous sommes ont beaucoup apprécié cette possibilité.

3 Idées des réductions

Nous allons dans cette partie vous présenter très brièvement les idées de réductions que nous avons utilisées. Nous allons seulement citer les quelques propriétés caractérisant nos problèmes qui nous ont permis de générer les formules *CNF*.

Nous citerons aussi ce que nous avons appelé les "cas faciles", c'est à dire les cas où nous n'avons pas besoin de *Minisat* pour répondre à nos problèmes (problème de décision).

3.1 *K-Col* $\leq_P$ *SAT*

Propriétés caractérisant *K-Col* :

- Chaque sommet est colorié.
- Il n'y a qu'une seule couleur par sommet.
- Deux sommets qui se suivent n'ont pas la même couleur.

Cas faciles :

- Le nombre de couleurs est négatif.
- Le nombre de couleurs est supérieur au nombre de sommets.

3.2 *Circuit Hamiltonien* $\leq_P$ *SAT*

Propriétés caractérisant *Circuit Hamiltonien* :

- Chaque noeud est dans le circuit.
- Un noeud a exactement une position dans le circuit sauf pour le premier/dernier.
- Deux sommets non voisins ne peuvent pas se suivre dans le circuit.
- Deux sommets n'ont pas la même position dans le circuit.
- Le dernier sommet du circuit doit aussi être le premier.

Cas faciles :

- Le nombre d'arêtes du graphe est strictement inférieur au nombre de sommets $\Rightarrow$ Impossible.
- Le graphe est un graphe complet $\Rightarrow$ Toujours vrai.

3.3 *Clique* $\leq_P$ *SAT*

Propriétés caractérisant *Clique* :

- Chaque noeud est dans la clique.
- Un noeud a exactement une position dans la clique.
- Deux sommets non voisins ne peuvent pas être ensemble dans la clique.
- Deux sommets n'ont pas la même position dans la clique.

Cas faciles :

- *Clique* de taille négative $\Rightarrow$ Erreur en entrée.
- *Clique* de taille 0 $\Rightarrow$ Toujours vrai.
- *Clique* de taille 1 $\Rightarrow$ Nécessite 1 seul sommet.
- *Clique* de taille 2 $\Rightarrow$ Nécessite 1 seule arête.
- *Clique* de taille strictement supérieure au nombre de sommets $\Rightarrow$ Impossible.
- A partir d'un certain nombre d'arêtes par rapport à un nombre de sommets, il est obligatoire d'obtenir une clique d'une certaine taille.
- Il faut un nombre d'arêtes minimum pour former une clique d'une certaine taille.

3.4 *Ensemble indépendant* $\leq_P$ *SAT*

Propriétés caractérisant *Ensemble Indépendant* :

- Inutiles car il y a une équivalence pour Ensemble indépendant de taille k sur $G = (V, E)$ et *Clique* de taille k sur $\overline{G}$ (Réduction de *Ensemble Indépendant* à *Clique*).

Cas faciles :

- *IS*[6] de taille négative ⇒ Erreur.
- *IS* de taille 0 ⇒ Toujours vrai.
- *IS* de taille 1 ⇒ Nécessite 1 seul sommet.
- *IS* de taille strictement supérieure au nombre de sommets ⇒ Impossible.
- On a forcément un *IS* d'une certaine taille si le nombre d'arêtes est trop faible.
- Pour que k sommets puissent former un *IS* dans un graphe de taille n, il ne faut pas que le nombre d'arêtes dépasse un certain nombre.

3.5 *Couverture par sommets* $\leq_P$ *SAT*

Propriétés caractérisant *Couverture par sommets* :

- Inutiles car il y a une équivalence pour *Couverture par sommets* de taille k sur $G = (V, E)$ et *Clique* de taille $|V| - k$ sur $\overline{G}$ (Réduction de *Couverture par sommets* à *Clique*).

Cas faciles :

- *VC*[7] de taille négative ⇒ Erreur.
- *VC* de taille 0 ⇒ Vrai s'il n'y a aucune arête.
- *VC* de taille strictement supérieure au nombre de sommets ⇒ Impossible.
- Pour que k sommets puissent représenter une *VC* dans un graphe de taille n, il ne faut pas que le nombre d'arêtes dépasse un certain nombre.

4 Tests et résultats

Pour vérifier le bon fonctionnement de notre programme, nous avons effectué plusieurs tests qui nous ont permis de nous rendre compte de différents problèmes.

4.1 Tests

Nous avons fait moults tests sur différents types de fichiers d'entrée. Ces fichiers tests sont disponibles dans le répertoire *jeux/*.

Nous avons testé notre programme sur des fichiers vides, et différents types de graphes. Des graphes à une seule arête, et des graphes générés par le programme *gengraph* : une clique à 100 sommets (graphe complet), un chemin à 100 sommets, un graphe aléatoire de 100 sommets, un arbre à 100 sommets...

4.2 Résultats des tests

Ces tests nous ont permis de mettre en avant quelques erreurs de programmation que nous avons corrigées mais surtout de trouver des cas simples, dont nous avons parlés précédemment, ainsi que de voir les limites des réductions et de *Minisat*.

- *Circuit Hamiltonien :* Si on lançait la recherche d'un Circuit Hamiltonien sur un graphe complet, à cause du grand nombre de clauses, le temps d'exécution de *Minisat* explosait alors qu'il n'y a rien de plus simple à trouver. Cas que nous avons résolu (en ajoutant un cas facile), mais si on enlève une arête à un graphe complet le problème persiste.
- *Couverture par sommets :* Si on lance cette réduction sur un graphe de type chemin de taille 100 et que l'on cherche une couverture de sommet de taille 50 ou inférieure (qui n'est pas satisfaisable dans ce denier cas) le temps d'exécution explose littéralement alors qu'en TD, nous avons vu que, pour ce genre de graphe et pour un nombre de sommets pair n, la plus petite couverture par sommets est de taille $n/2$ (voir 6.1 page 7 en annexe).

Sur des graphes de taille correcte à l'échelle papier (i.e. 4 ou 5 sommets), les résulats ont tous été concluants. Le résultat de l'exécution du programme pour un *Circuit Hamiltonien* (3 page 5) sur le graphe *jeurapport.gin* (fig.2 page 5) est bien un *Circuit Hamiltonien* (voir aussi 6.2 page 8 en annexe pour K-Col).

[6] *Independant Set*
[7] *Vertex Cover*

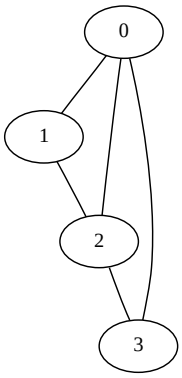

FIG. 2 – Graphe de test *jeurapport.gin*

```
killerpapy@killerpapy-port:~/COURS/AMR/projet/reduction$ ./solve jeux/jeurapport.gin 2
This is MiniSat 2.0 beta
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:  20                                                   |
|  Number of clauses:    75                                                   |
|  Parsing time:         0.00         s                                       |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |      15       67      170 |       22        0    nan |  0.000 % |
===============================================================================
restarts              : 1
conflicts             : 3              (inf /sec)
decisions             : 8              (0.00 % random) (inf /sec)
propagations          : 34             (inf /sec)
conflict literals     : 16             (5.88 % deleted)
Memory used           : 1.81 MB
CPU time              : 0 s

SATISFIABLE
Le graphe admet un circuit Hamiltonien.
Il suffit de considérer les arêtes suivantes :
{ 2-3, 3-0, 0-1, 1-2 }
```

FIG. 3 – *Circuit Hamiltonien* sur le graphe *jeurapport.gin* (fig.2 page 5)

5 Bilan

5.1 Difficultés rencontrées

Tout au long de notre projet, nous nous sommes heurtés à des difficultés.

Nous avons eu quelques soucis de démarrage avec ce que l'on appelle communément la barrière du langage. En effet, n'ayant jamais créé un programme en *C++*, les évidences, pour nous, n'en étaient pas. Les pointeurs que nous avions gentiment mis de côté dans notre esprit se sont rappelés à nous via des erreurs de compilation[8]. Cela nous a suivi tout au long de notre développement et continuera, à l'évidence, à nous suivre dans la suite du projet.

Au niveau des réductions, nous avons eu quelques problèmes, mais rien d'anormal. Nous avons de temps en temps oublié quelques propriétés mais nous nous en rendions assez rapidement compte en faisant tourner le programme sur des exemples.

Néanmoins, *Circuit Hamiltonien* nous a pris à revers dans la dernière ligne droite. En effet, nous n'avions pas considéré le cas du graphe formé de deux losanges rejoints par une arête (voir 6.3 page 9 en annexe). Une refonte totale de notre réduction a dû être opérée car nous avions utilisé des variables sur les arêtes, ce qui n'éliminait pas le problème. C'est en nous inspirant de la réduction de *Clique* à *SAT* que nous avons refait celle pour *Circuit Hamiltonien*.

Nous avons eu aussi des problèmes au niveau de la concordance des différents emplois du temps qui fut

[8] Ce qui, vous en conviendrez, n'est pas la plus agréable façon de se retrouver.

pour nous *NP-difficile*. Néanmoins ce problème n'était pas à l'insu de notre plein gré, car la composition du groupe ne nous était pas imposée.

Dernier problème et non le moindre, la contrainte de temps et le chevauchement des projets. En effet, nous n'avons pas pu faire tout ce que nous aurions aimé réaliser. Nous allons donc vous donner quelques idées d'améliorations que nous avons eues mais que nous n'avons pas pu inclure dans notre programme.

5.2 Améliorations

Nous allons rapidement vous donner une courte liste des choses que nous aurions aimé mettre en place dans le programme mais n'ont pas pu l'être :

- Vérifier les formats des fichiers en entrée[9].
- Trouver plus de "cas faciles".
- Donner le choix d'afficher ou pas l'exécution de *Minisat*. Il faudrait gérer l'option *-v* qui, si elle présente, permettrait d'afficher l'exécution de Minisat, et sinon, écrirait cette exécution dans *graph.min* (il faut utiliser une redirection de la sortie d'erreur de *Minisat* $(2>)$).
- Gérer les envois de signaux. Typiquement, durant l'exécution de *Minisat*, si on fait un *C-c*, le résultat affiché est quelque peu effrayant.
- Utiliser des structures de graphe différentes selon les réductions.

5.3 Conclusion

Tout d'abord, avant de conclure sur le travail effectué, nous tenons à exprimer le fait que nous sommes déçus de ne pas avoir pu expliquer plus en détail les raisonnements qui nous ont faits avancer dans le projet, notamment en ce qui concerne les réductions. Nous trouvons dommage que notre rapport doive se réduire à à peu près cinq pages[10].

Finalement, grâce au travail que nous avons effectué, nous avons compris l'utilité des réductions dans la pratique. Nous avons pu toucher du doigt la *NP-complétude* et entrevoir les limites à la fois de nos réductions et du *SAT-Solver Minisat*.

[9] Nous avons remarqué que si *Solve.cpp* était un graphe, alors il admettrait une *Couverture par sommets* de taille 1.
[10] Rapport $\leq_P$ Cinq-pages

6 Annexes

6.1 Test de couverture par sommets qui n'aboutit pas...

Couverture par sommets arrêtée après plus de 15 minutes : image 4 page 7. Même après plus de 30 minutes le résultat est le même mais nous n'avons pas d'image pour celui-là car nous avons dû redemarrer le pc... En effet on voit la memoire se remplir jusqu'à être pleine.

```
killerpapy@killerpapy-port:~/COURS/AMR/projet/reduction$ ./solve jeux/path100.gin 3 50
This is MiniSat 2.0 beta
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:  5000                                                 |
|  Number of clauses:    612600                                               |
|  Parsing time:         0.26        s                                        |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |    4900   596575  1430800 |   198858        0    nan |  0.000 % |
|       101 |    4900   596575  1430800 |   218744      101    745 |  0.000 % |
|       252 |    4900   596575  1430800 |   240618      252    736 |  0.000 % |
|       477 |    4900   596575  1430800 |   264680      477    724 |  0.000 % |
|       818 |    4900   596575  1430800 |   291148      818    713 |  0.000 % |
|      1324 |    4900   596575  1430800 |   320263     1324    693 |  0.000 % |
|      2085 |    4900   596575  1430800 |   352289     2085    703 |  0.000 % |
|      3225 |    4900   596575  1430800 |   387518     3225    679 |  0.000 % |
|      4934 |    4900   596575  1430800 |   426270     4934    718 |  0.000 % |
|      7496 |    4900   596575  1430800 |   468897     7496    712 |  0.000 % |
|     11341 |    4900   596575  1430800 |   515787    11341    738 |  0.000 % |
|     17115 |    4900   596575  1430800 |   567366    17115    780 |  0.000 % |
|     25768 |    4900   596575  1430800 |   624102    25768    823 |  0.000 % |
|     38744 |    4900   596575  1430800 |   686512    38744    835 |  0.000 % |
|     58206 |    4900   596575  1430800 |   755164    58206    914 |  0.000 % |
|     87398 |    4900   596575  1430800 |   830680    87398   1029 |  0.000 % |
|    131188 |    4900   596575  1430800 |   913748   131188   1144 |  0.000 % |
|    196873 |    4900   596575  1430800 |  1005123   196873   1171 |  0.000 % |

*** INTERRUPTED ***
restarts              : 18
conflicts             : 226613         (320 /sec)
decisions             : 333021         (1.11 % random) (470 /sec)
propagations          : 8905535        (12560 /sec)
conflict literals     : 277018187      (0.77 % deleted)
Memory used           : 1108.44 MB
CPU time              : 709.024 s

*** INTERRUPTED ***
Le graphe admet une couverture par sommets de taille 50.
Il suffit de considérer les sommets suivants :
 }
```

FIG. 4 – *Couverture par sommets* de taille 50 dans un chemin de taille 100.

6.2 Test de K-Col sur un petit graphe

3-Col sur graphe 5 page 8 de petite taille.

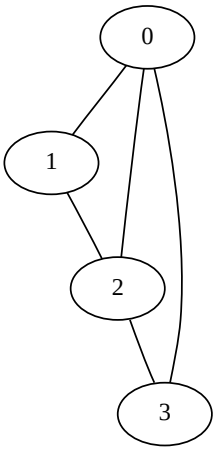

FIG. 5 – Exemple de graphe à 4 sommets.

```
killerpapy@killerpapy-port:~/COURS/AMR/projet/reduction$ ./solve jeux/jeurapport.gin 1 3
This is MiniSat 2.0 beta
WARNING: for repeatability, setting FPU to use double precision
============================[ Problem Statistics ]=============================
|                                                                             |
|  Number of variables:  12                                                   |
|  Number of clauses:    31                                                   |
|  Parsing time:         0.00         s                                       |
============================[ Search Statistics ]==============================
| Conflicts |          ORIGINAL         |          LEARNT          | Progress |
|           |    Vars  Clauses Literals |    Limit  Clauses Lit/Cl |          |
===============================================================================
|         0 |       0        0        0 |        0        0    nan |  0.000 % |
===============================================================================
restarts              : 1
conflicts             : 0              (nan /sec)
decisions             : 1              (0.00 % random) (inf /sec)
propagations          : 0              (nan /sec)
conflict literals     : 0              ( nan % deleted)
Memory used           : 1.82 MB
CPU time              : 0 s

SATISFIABLE
Le graphe admet une 3-Coloration.
Il suffit de considérer la coloration des sommets suivante :
{ 0 -> C2, 1 -> C1, 2 -> C3, 3 -> C1 }
```

FIG. 6 – Test de 3-Col sur le graphe 5

Ce qui donne comme résultat pour *3-Col* : fig.7 page 9

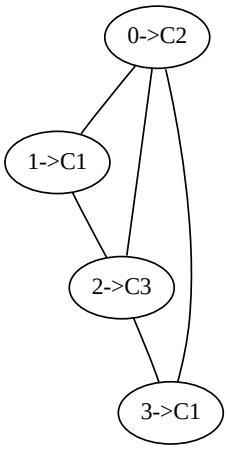

FIG. 7 – Test de 3-Col sur le graphe 5

6.3 Circuit Hamiltonien - Cas dangereux

Cas dangereux de graphe qui n'admet pas de *Circuit Hamiltonien* (fig.8 page 9).

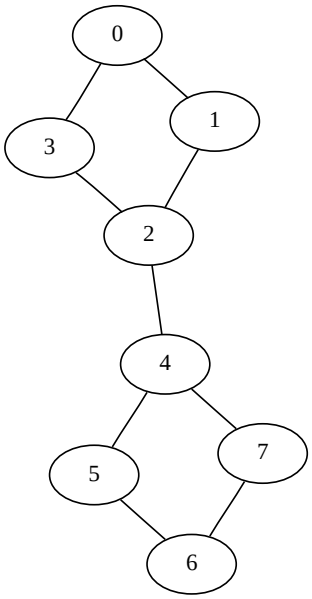

FIG. 8 – Graphe dangereux pour *Circuit Hamiltonien*