Université Paris Nanterre

Licence MIASHS deuxième année

Rapport de projet informatique

Génération et réfutation automatique de conjectures en théorie des graphes

Projet réalisé 06/01/2026

Membres du groupe

DAROUECHE Narmine – N° étudiant : 45002225
BACHIR Hichem – N° étudiant : 4500678

Dépôt GitHub : `https://github.com/gfk98gkvyr-lab/graph-conjecture-llm`

Table des matières

1 Introduction

La théorie des graphes fournit un langage simple pour décrire des systèmes composés d'objets et de relations : réseaux de communication, graphes de dépendances, réseaux sociaux, etc. Une grande partie du travail en théorie des graphes consiste à relier des propriétés globales (invariants) à la structure du graphe. Dans ce projet, nous nous intéressons notamment à la densité, au degré moyen, au degré maximal et au diamètre.

Dans la pratique, l'exploration exhaustive est rapidement impossible : le nombre de graphes différents croît extrêmement vite avec le nombre de sommets. Plutôt que de chercher à *prouver* des conjectures, l'objectif est de construire un outil expérimental capable d'explorer automatiquement un espace de graphes trop vaste pour une analyse complète, d'y observer des tendances et de produire des hypothèses testables.

Notre programme met en place une chaîne complète : génération de graphes, calcul d'invariants, génération de conjectures du type *IF/THEN*, puis recherche automatique de contre-exemples. Cette démarche s'inscrit dans l'esprit d'approches heuristiques comme GraphiTy : orienter la recherche vers des graphes intéressants et documenter ce que l'on observe.

2 Rappels théoriques

Un graphe $G = (V, E)$ est défini par un ensemble de sommets V et un ensemble d'arêtes E. Dans ce projet, nous manipulons des graphes non orientés.

Nous utilisons plusieurs **invariants** (propriétés globales) pour décrire un graphe :

- **Densité** : proportion d'arêtes présentes par rapport au nombre maximal possible.
- **Degré moyen** : moyenne des degrés des sommets.
- **Degré maximal** : plus grand degré parmi les sommets.
- **Diamètre** : plus grande distance (plus court chemin) entre deux sommets du graphe.

Ces invariants sont particulièrement adaptés à une approche expérimentale : ils résument une structure complexe via quelques valeurs numériques.

3 Architecture logicielle du projet

Le code est organisé en modules afin de séparer les responsabilités et faciliter la maintenance :

- `graph_utils.py` : fonctions utilitaires et calculs d'invariants.
- `graphity.py` : composantes heuristiques inspirées de GraphiTy (exploration locale / recherche de graphes extrêmes).
- `conjecture_schema.py` : structure des conjectures (prémisse, conclusion).
- `evaluator.py` : évaluation d'une conjecture sur un graphe (vrai/faux, vérification des conditions).
- `conjecture_generator.py` : génération de conjectures candidates (format IF/THEN).
- `counterexample_search.py` : stratégie de recherche de contre-exemples sous budget de calcul.
- `main.py` : point d'entrée, gestion des paramètres, sauvegarde, visualisation.

Cette structuration permet de modifier ou remplacer une partie sans réécrire l'ensemble.

4 Génération automatique des conjectures

Dans ce projet, une conjecture est représentée sous la forme d'une règle conditionnelle reliant deux invariants de graphes :

```
IF (condition sur un invariant) THEN (condition sur un autre invariant).
```

Concrètement, le système sélectionne deux invariants (par exemple le diamètre et le degré moyen), puis génère automatiquement des seuils numériques ainsi que des opérateurs de comparaison ($>$, $\leq$, etc.). Chaque combinaison définit une conjecture candidate pouvant être testée sur un ensemble de graphes.

L'intérêt de ce schéma est double :

— il permet de produire rapidement un grand nombre d'hypothèses testables sans intervention humaine directe ;

— il génère des règles simples et interprétables, facilitant l'analyse et la compréhension des relations entre invariants.

Il est important de souligner que cette étape ne vise pas à établir une vérité mathématique, mais à proposer des hypothèses candidates. Celles-ci sont ensuite soumises à une phase de validation expérimentale, reposant sur la recherche automatique de contre-exemples.

4.1 Phase 1 – FunSearch : le cœur algorithmique du projet

Cette phase constitue le cœur du projet. Elle s'inspire de l'approche FunSearch et vise à explorer automatiquement des graphes afin de générer et de tester des conjectures en théorie des graphes.

Le fonctionnement repose sur une boucle itérative. À chaque étape, le programme génère un graphe à partir de paramètres simples comme le nombre de sommets ou la probabilité de connexion. Sur ce graphe, plusieurs invariants sont calculés, par exemple le degré maximal, le diamètre ou la densité.

Ces invariants servent ensuite à tester des conjectures du type : si certaines propriétés sont vérifiées, alors une autre propriété doit l'être également. Lorsqu'un graphe contredit une conjecture, il est conservé comme contre-exemple.

Afin d'améliorer l'efficacité de la recherche, un système de score est utilisé. Les conjectures qui semblent intéressantes sont explorées davantage, tandis que celles qui échouent régulièrement sont progressivement écartées. Cela permet de limiter l'exploration exhaustive de tous les cas possibles.

Enfin, certains résultats non pertinents ou redondants sont filtrés. Cette approche permet d'orienter la recherche et de se concentrer sur les graphes et conjectures les plus informatifs. Cette phase constitue ainsi le moteur principal de l'exploration menée dans le projet.

4.2 Phase 2 – Intégration d'une intelligence artificielle (LLM)

Une intelligence artificielle de type modèle de langage (LLM) est utilisée dans le projet comme un outil d'assistance à la génération de conjectures. Son rôle n'est pas de remplacer les calculs algorithmiques, mais d'aider à formuler des hypothèses pertinentes à partir du contexte étudié.

L'IA intervient à partir de descriptions structurées du problème, incluant les invariants considérés et les objectifs de recherche. À partir de ces informations, elle propose des conjectures exprimées en langage naturel, qui servent ensuite de base à l'exploration automatique.

Les propositions générées sont ensuite interprétées par le système. Un mécanisme simple de filtrage permet de ne conserver que les conjectures respectant un format précis, afin d'éviter toute ambiguïté. Ces conjectures peuvent alors être testées automatiquement sur les graphes générés.

Lorsque l'IA produit des conjectures triviales ou incorrectes, ces résultats sont pris en compte pour ajuster les formulations utilisées lors des itérations suivantes. Ce retour progressif permet d'améliorer la pertinence des propositions sans intervention manuelle constante.

Enfin, il est important de souligner que l'IA ne fournit aucune preuve mathématique. Elle agit uniquement comme un outil d'aide à l'exploration, tandis que la validation ou la réfutation des conjectures repose entièrement sur les calculs algorithmiques et la recherche de contre-exemples.

5 Recherche de contre-exemples

Pour une conjecture donnée, l'outil cherche à identifier un graphe G tel que :

— la prémisse de la conjecture est satisfaite sur G,

— la conclusion de la conjecture est violée sur G.

La recherche de contre-exemples repose sur une approche heuristique. Le programme explore un ensemble de graphes générés automatiquement, principalement à l'aide de méthodes aléatoires, mais également par des modifications progressives de graphes existants afin d'explorer des configurations proches.

Cette exploration est encadrée par des paramètres contrôlant les limites imposés, tels que le nombre d'essais (`tries`) et le nombre d'itérations (`steps`). Ces paramètres permettent de limiter le temps d'exécution tout en conservant une diversité suffisante de graphes testés.

Lorsqu'un graphe contredisant une conjecture est identifié, celui-ci est conservé comme contre-exemple. Il est alors sauvegardé afin de pouvoir être analysé ultérieurement et éventuellement visualisé. Cette étape permet de mettre en évidence les limites ou les faiblesses de certaines conjectures générées automatiquement.

Critère d'arrêt. Si aucun contre-exemple n'est trouvé dans les limites fixées, l'outil retourne le message ■ aucun contre-exemple trouvé ■ (dans les limites testées). Il est important de souligner que ce résultat ne constitue pas une preuve de validité de la conjecture, mais indique uniquement qu'aucune violation n'a été observée sur l'ensemble des graphes explorés.

6 Expérimentations et résultats

Nous avons réalisé plusieurs exécutions avec les mêmes paramètres ($n = 20$, $p = 0.2$, `tries`=8, `steps`=1200) en faisant varier la graine aléatoire.

Chaque exécution produit :

— un fichier `run_*.json` contenant les paramètres, la conjecture générée et le résultat (contre-exemple trouvé ou non),

— une visualisation `example_graph_*.png` représentant un graphe exemple manipulé par l'outil,

— éventuellement une image `counterexample_*.png` si un contre-exemple est identifié et récupérable sous forme de graphe.

Observation principale. Selon la conjecture générée et la graine, l'outil peut soit ne trouver aucun contre-exemple dans les limites de recherche, soit identifier une violation. Cela illustre l'aspect exploratoire du projet : la difficulté n'est pas de ■ prouver ■, mais de parcourir efficacement un espace de tests et de documenter les résultats.

6.1 Phase 5 – Tests, résultats et stabilisation

De nombreuses exécutions du programme ont été réalisées afin d'évaluer le comportement global du système pour différents choix de paramètres. Ces tests ont permis d'observer la diversité des graphes générés, ainsi que la capacité de l'outil à identifier des contre-exemples pour certaines conjectures.

Les résultats obtenus sont automatiquement sauvegardés afin de conserver une trace des expérimentations réalisées. Ils prennent notamment la forme de fichiers au format JSON, contenant les informations relatives aux conjectures testées, ainsi que d'images représentant les graphes étudiés. Cette organisation facilite l'analyse a posteriori des résultats et permet de comparer différentes exécutions.

Une phase de nettoyage et de stabilisation a également été nécessaire. Elle a consisté à éliminer les résultats redondants, à filtrer certains cas peu informatifs et à corriger des comportements liés à l'aléatoire. Cette étape a permis d'améliorer la lisibilité des résultats et d'assurer une reproductibilité minimale des expériences menées, malgré le caractère aléatoire de certaines générations.

6.2 Analyse critique et limites

Malgré les résultats encourageants obtenus, ce projet présente plusieurs limites qu'il est important de souligner. Tout d'abord, l'exploration de l'espace des graphes reste partielle et dépend fortement des paramètres choisis lors des générations. Certaines configurations de graphes peuvent ne jamais être explorées dans les limites fixées.

Par ailleurs, l'intégration de l'intelligence artificielle reste volontairement limitée à un rôle d'assistance. Les conjectures proposées peuvent être imprécises, triviales ou redondantes, ce qui nécessite un filtrage rigoureux avant leur exploitation. De plus, l'absence de démonstration mathématique formelle constitue une limite importante : les résultats obtenus sont uniquement de nature expérimentale.

Enfin, le coût computationnel augmente rapidement lorsque la taille des graphes croît. Cette contrainte limite les expérimentations à des graphes de taille modérée et empêche une exploration exhaustive. Ces limites ouvrent toutefois des perspectives

d'amélioration pour de futurs travaux, notamment en termes d'optimisation et de stratégies d'exploration plus ciblées.

En termes de limites, ce projet repose sur une exploration partielle de l'espace des graphes. En raison du caractère combinatoire du problème, il n'est pas possible de tester l'ensemble des graphes possibles, ce qui implique que certaines configurations pertinentes peuvent ne pas être explorées.

Les résultats obtenus dépendent également fortement des paramètres choisis lors des expérimentations, tels que la taille des graphes, la probabilité de connexion ou le nombre d'itérations. Des choix différents peuvent conduire à des observations différentes, ce qui limite la généralisation des résultats.

Par ailleurs, la recherche de contre-exemples repose sur des heuristiques. Bien que ces méthodes permettent de réduire le temps de calcul, elles peuvent introduire un biais dans l'exploration et ne garantissent pas la découverte systématique de contre-exemples lorsqu'ils existent.

Enfin, l'utilisation de l'intelligence artificielle reste limitée à un rôle d'assistance. Les conjectures proposées peuvent être imprécises ou redondantes et nécessitent une validation algorithmique rigoureuse. De plus, aucune démonstration mathématique formelle n'est fournie, les résultats obtenus étant exclusivement de nature expérimentale.

7 Apport de l'intelligence artificielle dans la réalisation du projet

Une intelligence artificielle conversationnelle a été utilisée tout au long du projet comme un **outil d'assistance**. Son rôle principal a été d'accompagner les différentes étapes du travail, sans se substituer aux choix techniques ou conceptuels effectués par les membres du groupe.

Plus précisément, l'IA a été mobilisée pour :

— clarifier certains points conceptuels, notamment concernant les invariants de graphes et l'interprétation des résultats obtenus ;
— proposer des pistes pour la structuration du code, en particulier l'organisation modulaire et la définition d'un point d'entrée clair ;
— assister la mise en place de la sauvegarde des résultats, notamment sous forme de fichiers JSON, ainsi que la visualisation des graphes générés au format PNG ;

Il est important de souligner que l'IA n'a pas remplacé le travail de conception ni les prises de décision. Les choix finaux concernant les paramètres, l'organisation du projet, les exécutions et la validation des résultats ont été réalisés, analysés et vérifiés par les membres du groupe. L'IA a ainsi joué un rôle de soutien méthodologique en complément du travail humain.

8 Fonctionnement de l'intelligence artificielle utilisée

Dans le cadre de ce projet, l'IA a été utilisée comme un outil d'assistance. Elle a notamment permis de proposer des pistes de structuration du code ou du rapport, et d'aider à identifier d'éventuelles incohérences ou erreurs. Son utilisation s'est limitée à un rôle de soutien méthodologique et rédactionnelle.

Il est important de préciser que l'IA ne dispose d'aucun accès direct à l'exécution du programme. Toutes les expérimentations ont été réalisées localement par les membres du groupe, via le terminal. Les résultats obtenus ont ensuite été vérifiés à partir des

sorties générées et des fichiers sauvegardés, garantissant ainsi un contrôle humain sur l'ensemble du processus.

9 Environnement de travail

Le projet a été réalisé en Python, un langage largement utilisé pour le prototypage et l'expérimentation en informatique. Ce choix s'explique par la richesse de son écosystème de bibliothèques dédiées à la manipulation et à l'analyse de graphes.

La bibliothèque `NetworkX` a été utilisée pour représenter les graphes, générer différents types de graphes aléatoires et calculer plusieurs invariants tels que les degrés, les distances ou encore le diamètre. Elle constitue un outil adapté à l'expérimentation rapide en théorie des graphes.

Pour la visualisation des graphes générés, la bibliothèque `Matplotlib` a été employée afin d'exporter des figures au format image. Ces visualisations permettent notamment d'illustrer certains graphes significatifs, en particulier les contre-exemples identifiés lors des expérimentations.

Les résultats des différentes exécutions sont sauvegardés au format JSON. Ce format permet de conserver de manière structurée les paramètres utilisés (taille des graphes, probabilités, conjectures testées) ainsi que les résultats obtenus, facilitant ainsi leur analyse a posteriori.

10 Description du projet et objectifs

Le projet vise à mettre en place une chaîne complète d'exploration automatique en théorie des graphes. Cette chaîne repose sur plusieurs étapes successives permettant de générer, tester et analyser des conjectures de manière expérimentale :

1. générer des graphes, principalement de manière aléatoire ;
2. calculer des invariants simples et interprétables ;
3. générer automatiquement des conjectures sous la forme de règles conditionnelles de type *IF/THEN* ;
4. tester ces conjectures sur un ensemble de graphes et rechercher d'éventuels contre-exemples ;
5. sauvegarder les résultats obtenus et produire des figures afin de faciliter leur analyse.

L'outil développé permet de faire varier différents paramètres, tels que la taille des graphes, la probabilité de présence d'arêtes, le nombre d'essais ou les limites de l'exploration. Ces variations ont pour objectif d'observer des tendances empiriques et de mettre en évidence des exemples permettant de réfuter certaines hypothèses générées automatiquement.

11 Conclusion

Ce projet avait pour objectif d'explorer la génération et la réfutation automatique de conjectures en théorie des graphes. En combinant une approche algorithmique inspirée de FunSearch et l'assistance d'une intelligence artificielle, nous avons développé un

système capable de produire automatiquement des conjectures et de les tester sur un ensemble de graphes générés.

Les résultats obtenus montrent qu'il est possible d'identifier efficacement des contre-exemples à certaines conjectures, tout en structurant une exploration guidée de l'espace des graphes. L'intégration de l'intelligence artificielle apporte une aide pertinente à la formulation d'hypothèses et à l'orientation de la recherche, même si la validation et la réfutation reposent entièrement sur des calculs algorithmiques.

Ce travail met en évidence le potentiel des approches hybrides combinant expérimentation algorithmique et assistance par l'IA, tout en soulignant leurs limites actuelles. Parmi les perspectives d'amélioration, on peut citer l'étude de graphes de taille ou de complexité plus élevées, l'enrichissement des invariants considérés, ainsi qu'une interaction plus fine entre l'IA et le moteur algorithmique afin d'orienter plus efficacement l'exploration. Ces limites constituent des pistes d'amélioration pour de futurs travaux, notamment en termes d'optimisation de l'exploration et d'enrichissement des méthodes utilisées.

Remerciements

Nous remercions également notre encadrant pour l'accompagnement sur les choix techniques et la méthodologie de travail.

12 Annexe : exemple d'exécution et captures

13 Exemple d'exécution du projet

Cette annexe présente un exemple concret d'exécution du programme développé dans le cadre du projet. Elle illustre les différentes étapes du processus, depuis la génération des graphes jusqu'à l'identification de contre-exemples à certaines conjectures.

13.1 Capture du terminal

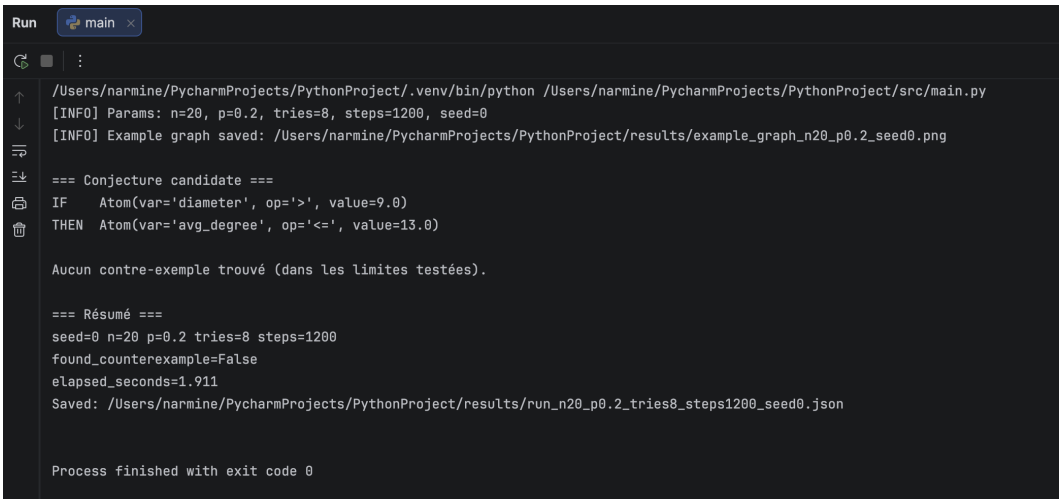

FIGURE 1 – Sortie du terminal lors d'une exécution du programme (seed = 0).

13.2 Graphe généré

14 Fichiers générés

Lors des différentes exécutions du programme, plusieurs fichiers sont générés automatiquement afin de conserver une trace des expérimentations réalisées. Ces fichiers permettent d'analyser les résultats obtenus et de reproduire certaines expériences.

15 Manuel utilisateur

Pour exécuter le programme, l'utilisateur doit disposer d'un environnement Python fonctionnel avec les bibliothèques nécessaires installées. Les instructions détaillées pour l'installation et l'exécution du projet sont fournies dans le fichier `README.md` du dépôt GitHub.

De manière générale, l'exécution du programme se fait via une commande dans le terminal, permettant de lancer la génération de graphes et l'évaluation automatique des conjectures.

— `results/run_n20_p0.2_tries8_steps1200_seed0.json`
— `results/example_graph_n20_p0.2_seed0.png`

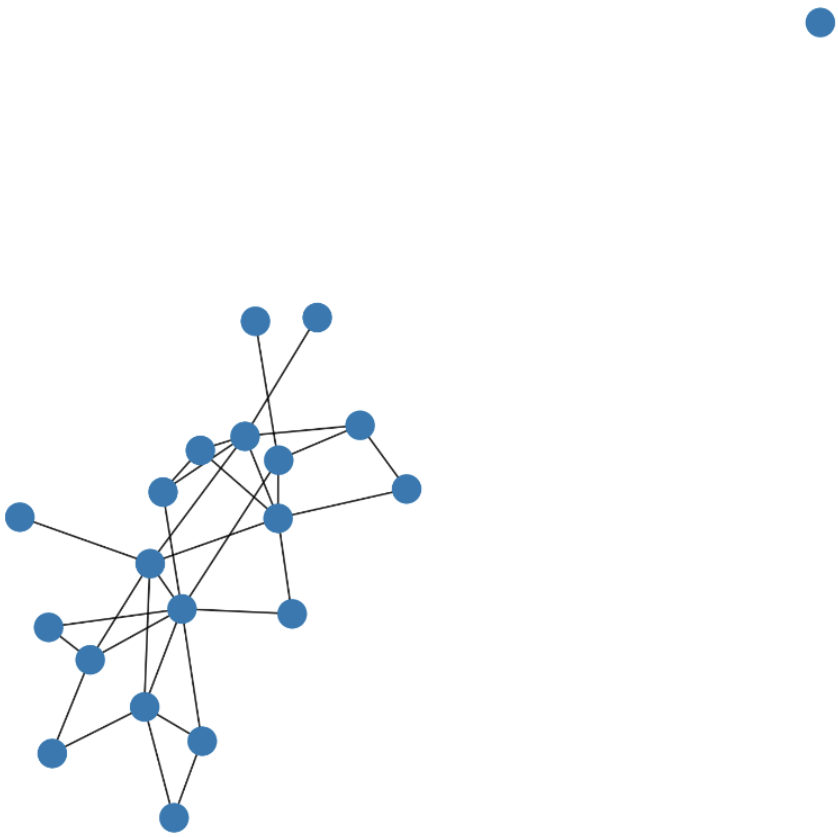

FIGURE 2 – Exemple de graphe généré servant de contre-exemple à une conjecture.