université
PARIS-SACLAY

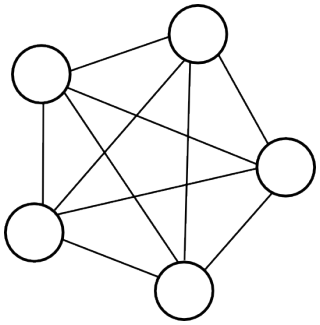
**Licence Double Diplôme**
**Informatique Mathématiques**

*Rapport de projet*

# Circuits booléens et graphes

*Auteurs :*
Alexis VO
Raphaël [redacted]
Théo [redacted]

*Encadrant :*
M. J. Marrez



Version du 07 mai 2025

# Table des matières

# 1 Introduction

Ce projet, réalisé dans le cadre de l'UE Projet Informatique (OLIN243B), vise à renforcer notre maîtrise de la programmation Python via la conception d'une bibliothèque de graphes et son application à la modélisation de circuits booléens. Nous avons combiné des approches théoriques (structures de graphes, logique propositionnelle) à des contraintes pratiques (visualisation, tests unitaires, travail collaboratif).

L'organisation du rapport suit le fil du développement :

- conception et architecture logicielle ;
- fonctionnalités de base et affichage des graphes ;
- extensions avancées : circuits booléens et algorithmes de graphe ;
- études de cas : half-adder, dijkstra, synthèse de formule ;
- bilan fonctionnel et perspectives.

# 2 Architecture du projet

Le projet repose sur une classe centrale `OpenDigraph`, complétée par des *mixins* pour la modularité. La synthèse des composants est la suivante :

```
/modules
   |-- open_digraph.py
   |-- digraph_matrix.py
   |-- bool_circ.py
   |-- mixins/
        |-- compositions_mixin.py
        |-- displate_mixin.py
        |-- shortest_path_mixin.py
        |-- longest_path_mixin.py
        |-- matrix_mixin.py
        |-- properties_mixin.py
/tests
   |-- open_digraph_test.py
```

Chaque mixin apporte une responsabilité claire (affichage, matrices, algorithmes, etc.). Les tests unitaires garantissent la robustesse des méthodes.

# 3 Implémentation et affichage de graphes

## 3.1 Manipulation de base

La classe `OpenDigraph` supporte :
— ajout/suppression de nœuds et d'arêtes ;
— contrôle de cohérence avec `is_well_formed` ;
— conversion graphe/matrice et génération aléatoire.

## 3.2 Affichage

La méthode `display` propose deux modes :
— **Local** : export `.dot` → `.png` via Graphviz, ouverture automatique ;
— **Web** : encodage URL pour Graphviz Online.

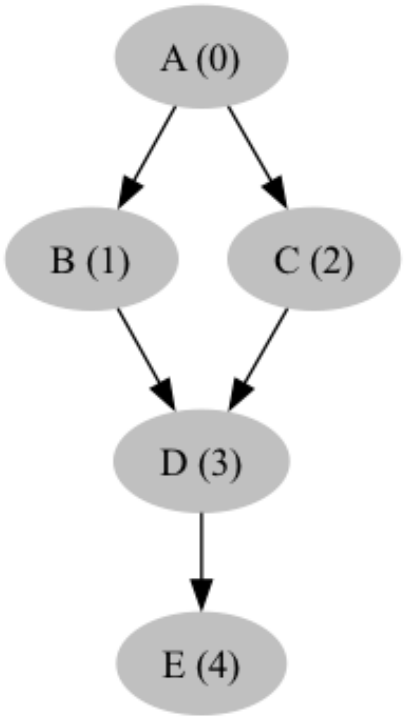



FIGURE 1 – Affichage local d'un graphe

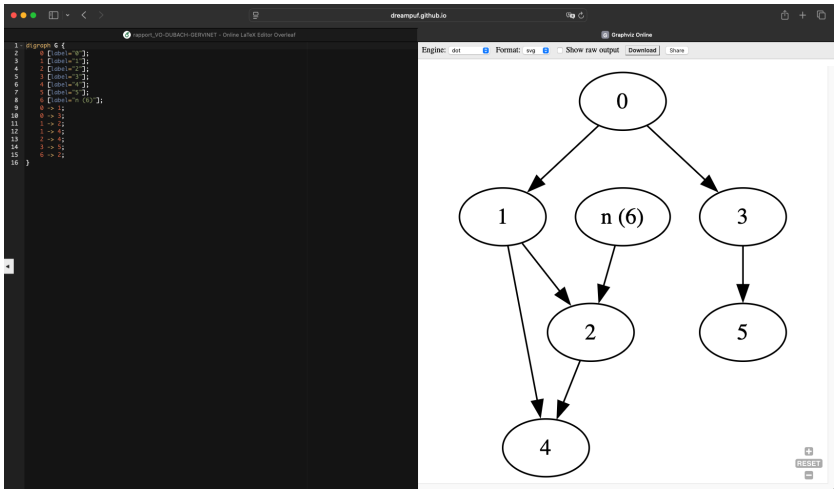



FIGURE 2 – Affichage web d'un graphe

# 4 Circuits booléens et algorithmes avancés

## 4.1 Classe `bool_circ`

Encapsulation d'un `OpenDigraph` pour représenter un circuit logique acyclique. Méthodes clés :

- `is_cyclic`, `is_well_formed` ;
- `shift_indices` ;
- synthèse depuis une formule propositionnelle.

## 4.2 Composition et connexité

Opérations `iparallel`, `icompose`, `identity`, `connected_components`. Exemples :

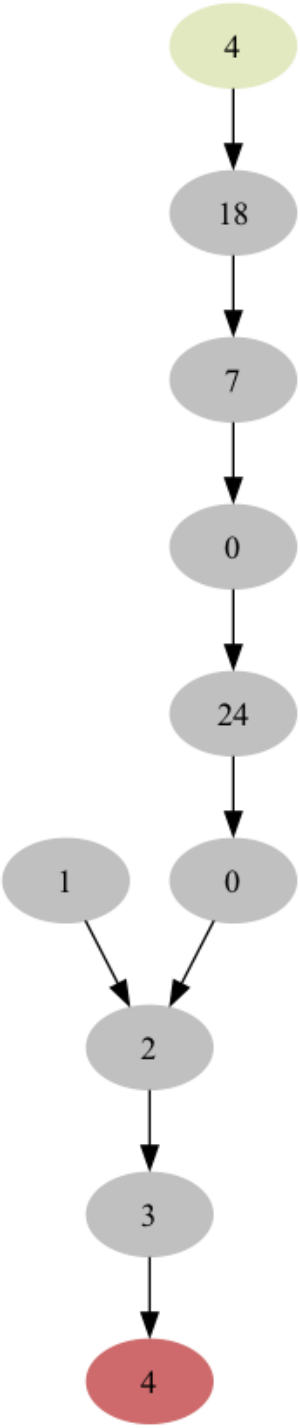



FIGURE 3 – Composition parallèle de deux graphes

## 4.3 Algorithmes de chemin

— Dijkstra et `shortest_path` ;
— ancêtres communs (distances) ;
— tri topologique et `longest_path`.

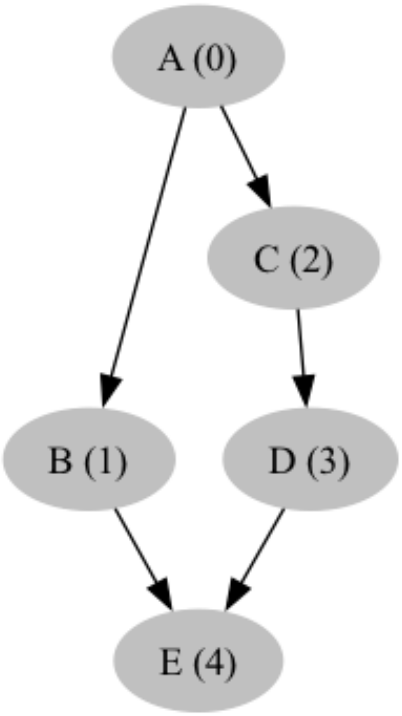



FIGURE 4 – Plus court (resp. plus long) chemin via Dijkstra.
Ici le plus court est $ABE$ (resp. le plus long est $ACDE$)

## 4.4 Synthèse de formule

Transformation d'expressions logiques (ex. `((x0)&(x1))|(~(x2))`) en graphe :

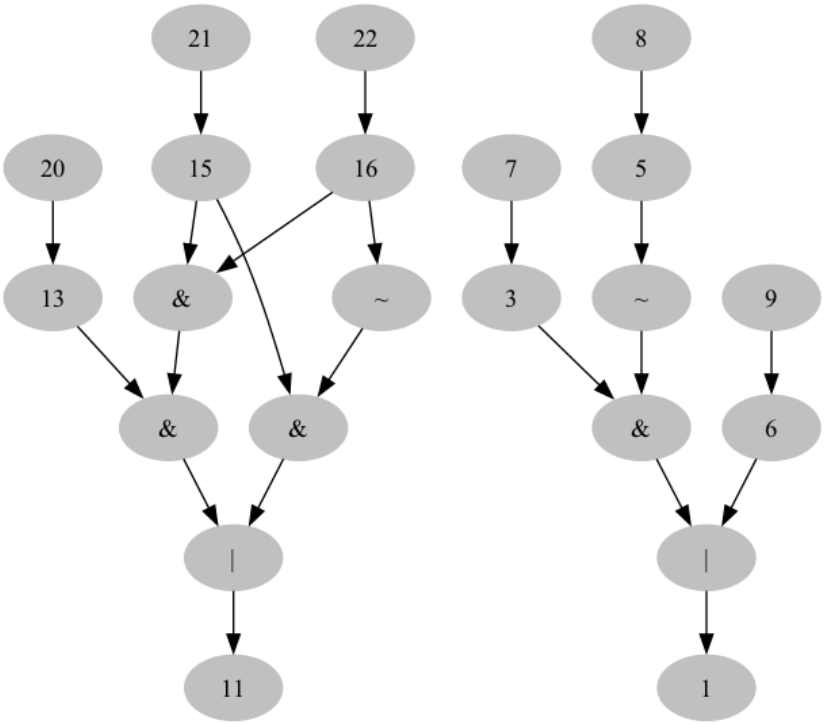



FIGURE 5 – Circuit synthétisé depuis une formule

# 5 Études de cas

## 5.1 Half-adder

Exemple de simulation et évaluation :

1. entrée $a_0, b_0$ ;
2. application de `evaluate()` ;
3. sortie $s_0, c_0$.

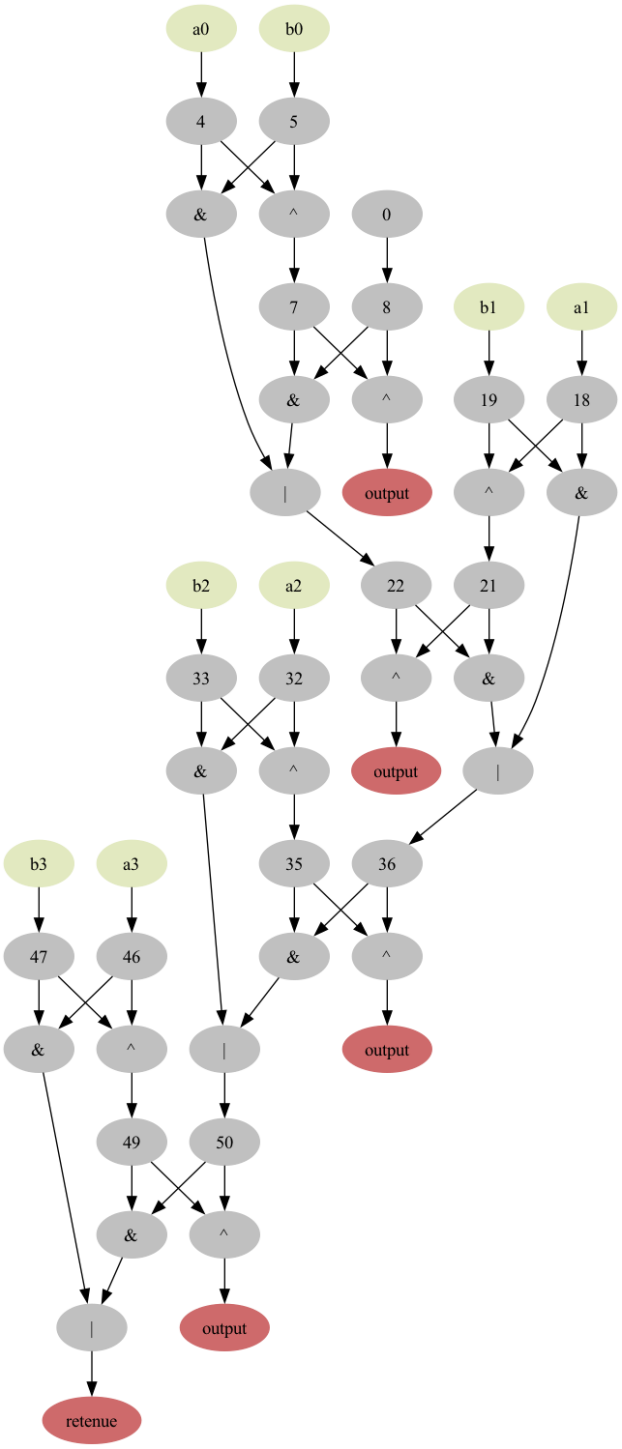



FIGURE 6 – Évaluation d'un half-adder

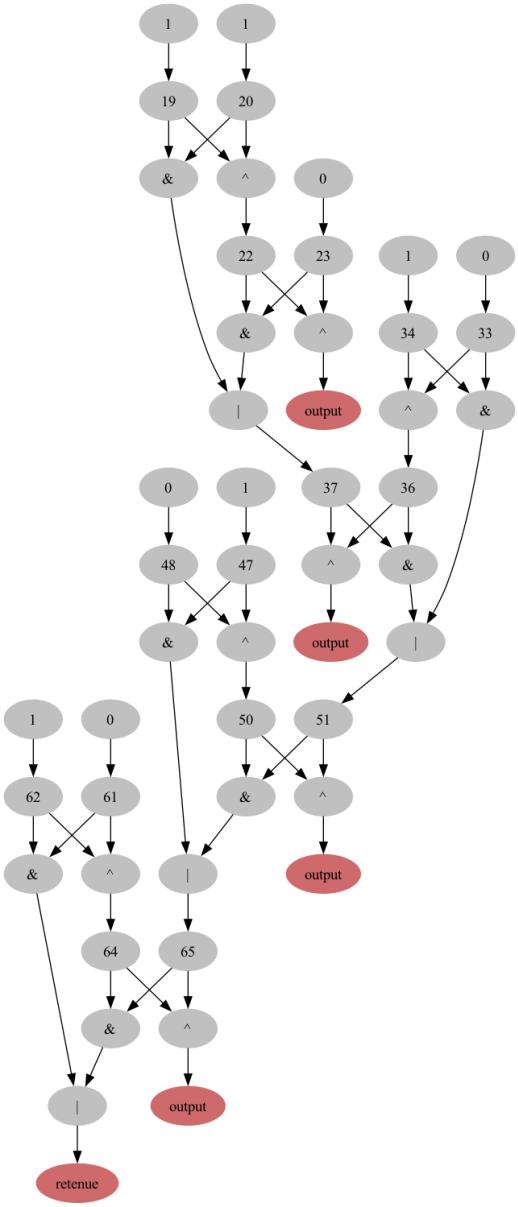



FIGURE 7 – Half-adder avant

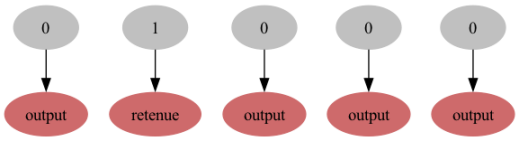



FIGURE 8 – Half-adder après

### 5.2 Dijkstra

Illustration des chemins calculés pour un petit graphe de démonstration.

### 5.3 Synthèse de formule

Exécution du test `PARSE_PARENTHESES` pour générer un graphe de circuit.

## 6 Bilan fonctionnel et perspectives

### 6.1 Bilan des réalisations

Nous avons implémenté et testé :
— génération de circuits booléens aléatoires et additionneurs ;
— évaluation et simplification via `evaluate()` ;
— vérification de code de Hamming ;
— algorithmes de graphe (Dijkstra, tri topologique, plus long chemin).

### 6.2 Outils

— VS Code + LiveShare pour la collaboration ;
— Git/GitHub pour le versioning ;
— Graphviz pour la visualisation ;
— tests unitaires & flake8 pour la qualité du code.

### 6.3 Perspectives

— extension aux circuits séquentiels ;
— simplification avancée (BDD, Quine-McCluskey) ;
— interface graphique de conception et simulation ;
— optimisation des performances et réduction de profondeur.