Rapport de Présentation de la Bibliothèque de Gestion des Digraphes et Circuits Booléens

Yehor KOROTENKO, Ivan KHARKOV, Sipan BAREYAN

March 14, 2025

1 Tâches Accomplies

- **Structure de Base du Graphe:**
  - Implémentation de la classe `node` : Représente des nœuds individuels avec des identifiants uniques, des étiquettes et des relations parent/enfant.
  - Développement de la classe `open_digraph` : Gère les nœuds, les entrées/sorties désignées et fournit des méthodes pour ajouter/supprimer des arêtes et des nœuds.
- **Opérations Avancées:**
  - Ajout de la capacité de copie profonde, décalage des indices des nœuds et génération de correspondances d'identifiants séquentiels.
- **Génération de Graphes Aléatoires:**
  - Création de fonctions pour générer des matrices d'adjacence aléatoires (libres, symétriques, orientées, triangulaires) et conversion de ces matrices en digraphes via `open_digraph.random()`.
- **Extension aux Circuits Booléens:**
  - Extension du graphe avec la classe `bool_circ`, imposant des contraintes logiques telles que l'acyclicité et des degrés de nœuds appropriés.
- **Visualisation:**
  - Développement d'unu méthode de visualisation utilisant graphviz pour générer des graphiques en PDF, organisant les nœuds en sous-graphes d'entrée/sortie.
- **Entrée/Sortie de Fichiers:**
  - Implémentation de l'exportation vers des fichiers DOT et de l'importation depuis ces fichiers, permettant un stockage et une récupération persistants des données du graphe.
- **La documentation**
  - La documentation qui est stockée dans le répértoire `docs` qui décrit le fonctionnement des modules, méthodes, fonctions et provient les exemples d'utilisation.
- **Collaboration et le répartition du travaille**
  - Un dépot sur GitHub avait été crée dans lequel le code est souvegardé et partagé. Chaque membre de l'équipe a l'accès à ce dépot où chacun contribut du code.

2 Principales Fonctions et Descriptions

- **Fonctions de la Classe Node:**
  - `get_id()`, `get_label()`, `set_parents()`, etc.
  - Gestion des arêtes : `add_child_id()`, `add_parent_id()`, `remove_child_once()`, `remove_parent_once()`.
  - Calculs de degré : `indegree()`, `outdegree()`, `degree()`.
- **Fonctions de `open_digraph`:**
  - Modifications du graphe : `add_edge()`, `add_node()`, `remove_edge()`, `remove_node_by_id()`.
  - Gestion de l'état du graphe : `new_id()`, `copy()`, `shift_indices()`.
  - Validation de la cohérence : `assert_is_well_formed()`.
  - Génération de matrices d'adjacence : `adjacancy_matrix()`.
- **Génération de Graphes Aléatoires:**
  - Fonctions : `random_int_matrix()`, `random_symmetric_int_matrix()`, `random_oriented_int_matrix()`, `random_triangular_int_matrix()`.
  - Intégration via `open_digraph.random()` permettant des paramètres tels que le nombre de nœuds, les bornes et les entrées/sorties désignées.
- **Spécificités des Circuits Booléens (`bool_circ`):**
  - Hérite de `open_digraph` et impose des contraintes logiques supplémentaires.
  - Utilise `is_well_formed()` pour garantir l'acyclicité et la validité des degrés des nœuds.
- **Module de Visualisation:**
  - La méthode `display()` utilise Graphviz pour créer une représentation PDF du graphe.

3 Tests et Validation

Tous les tests ont été exécutés avec succès à l'aide de la suite de tests fournie :

- **Opérations sur les Nœuds et Graphes:** Test d'initialisation, ajout et suppression d'arêtes/nœuds, copie et calculs de degré.
- **Vérification de la Cohérence des Graphes:** Validation de la cohérence de `open_digraph` et `bool_circ` avec `assert_is_well_formed()`.
- **Génération de Graphes Aléatoires:** Vérification que les fonctions génèrent les types de matrices corrects et des graphes conformes.
- **Entrée/Sortie des Fichiers DOT:** Test de l'exportation et de l'importation pour garantir la préservation des propriétés structurelles du graphe.

Les tests sont intégrés dans le framework `unittest` de Python et exécutés via le script shell `tests.sh`.

4 Exemples d'Utilisation

Voici quelques exemples démontrant les principales fonctionnalités de la bibliothèque :

Création d'un Graphe Aléatoire

```
from modules.open_digraph import open_digraph

# Création d'un digraphe aléatoire avec 6 nœuds, une borne de 9, 1 entrée et 2 sorties
random_graph = open_digraph.random(n=6, bound=9, inputs=1, outputs=2)
```

Création d'un Graphe concrète

```
n0 = node(0, '0i', {3:1}, {1:1, 2:2})
n1 = node(1, '1i', {0:1}, {3:3})
n2 = node(2, '2o', {0:2}, {})
n3 = node(3, '3a', {1:3, 4:1}, {0:1})
n4 = node(4, '4i', {}, {3:1})
graph = open_digraph([4], [2],
                  [n0, n1, n2, n3, n4])
```

Sauvegarde dans un Fichier DOT

```
# Sauvegarde du graphe généré dans un fichier DOT pour visualisation
graph.save_as_dot_file("./dot.dot")

# Le résultat obtenu dans le fichier dot.dot
digraph G{
subgraph inputs{
rank=same;
v4 [label="4i" shape=diamond]
}
subgraph outputs{
rank=same;
v2 [label="2o" shape=box]
}
v0 [label="0i" ]
v1 [label="1i" ]
v3 [label="3a" ]
v0 -> v1;
v0 -> v2;
v0 -> v2;
v1 -> v3;
v1 -> v3;
v1 -> v3;
v3 -> v0;
v4 -> v3;
}
```

Importation et Affichage du Graphe

```
from modules.open_digraph import open_digraph

# Importation du graphe depuis le fichier DOT
imported_graph = open_digraph.from_dot_file("./dot.dot")

# Affichage du graphe (Graphviz ouvrira un PDF)
imported_graph.display("imported_graph")
```

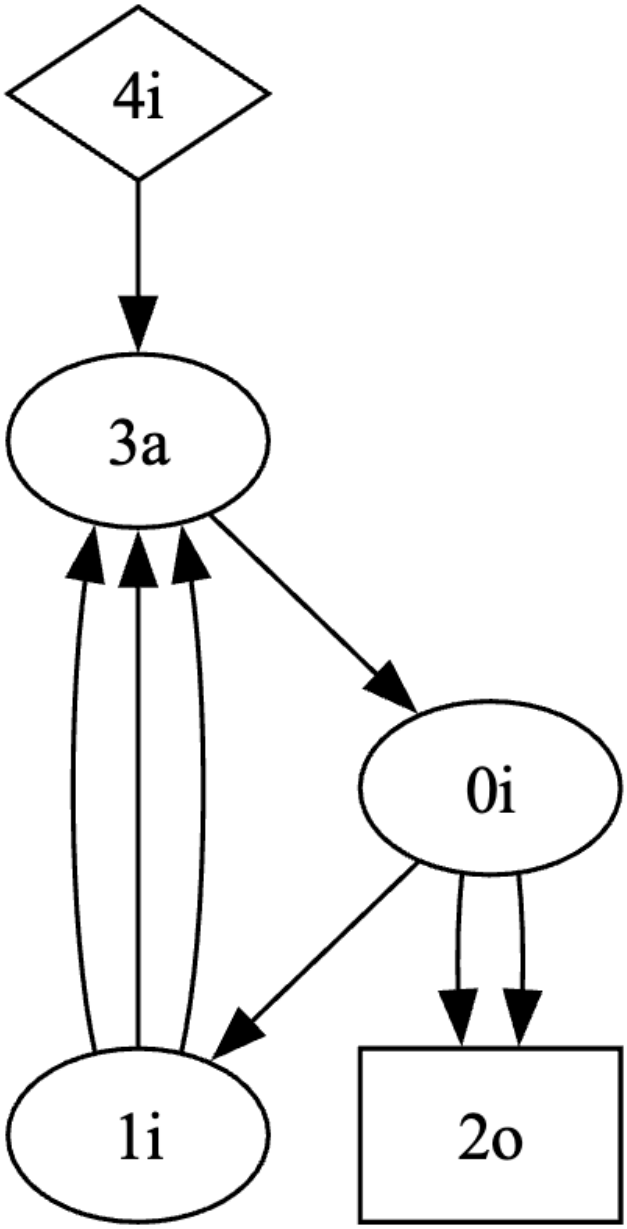

Figure 1: Affichage du graphe par `graph.display()`