Documentation sur l'utilisation du MétaQPN

Auteur : Romain RINCÉ

1 Présentation

La structure MétaQPN a pour objectif de pouvoir définir rapidement de nouveaux types de QPN sans avoir à ré-implémenter tous les algorithmes de parcours et de gestion des nœuds et arêtes.

2 Création d'un nouveau type de QPN

La définition de QPN que nous utilisons est légèrement modifiée dans le code puisque un QPN est défini par un type de relation qui existe entre les nœuds de votre graphe. Si vous souhaitez créé un QPN qui possède plus d'un type de relation, il vous faudra donc créer autant de QPNs dans le code. Ces QPNs sont ensuite fourni au MétaQPN qui se charge de leur gestion.

Ainsi, si vous souhaitez créer une modèle de QPN simple qui gère les influences et la synergie produit, il vous sera nécessaire de créer un modèle de QPN pour les influences et un pour les synergies. Cela peut sembler compliqué, mais vous n'aurez besoin que de très peu de code pour chaque QPN implémenté.

2.1 Création du type d'arc

Avant de définir votre QPN, vous devez créer un type d'arc qui définira *in fine* votre graphe.

Pour ce faire créer une classe héritant de `qpn_edge` et redéfinissez les méthodes `Sign getSign();` et `ostream& writeGraphVizFormat(ostream& os) const;`.

La première méthode doit pouvoir renvoyer le signe d'un arc au moment ou il est demandé (celui-ci pouvant dépendre du contexte). Lors de la construction de l'objet, penser donc à donner toutes les informations ou accès nécessaire à votre arc pour qu'il puisse répondre. cf. 2.2

La seconde méthode doit fournir les options qui seront nécessaire pour l'affichage des arcs avec GraphViz. Par exemple le signe pour les influences [1].

1. label="-" par exemple

2.2 Implémenter le nouveau QPN

Une fois votre structure d'arc définie, il vous vaudra créer la structure QPN qui va les utiliser.

Pour cela implémenter une classe qui hérite de `qpn_descriptor`. `qpn_descriptor` est défini avec deux templates. La première fixe le type de données dans le graphe qu'il est préférable de laisser générique afin de rester cohérent par la suite, mais cela vous regarde. Le second template doit être spécifié puisqu'il défini si votre graphe gère des arcs orientés ou non. Il peut donc être soit de type `boost::bidirectionalS` ou `boost::undirected`.

Votre QPN doit implémenter deux méthodes virtuelles de `qpn_descriptor` :
- `void addEdge(qpn_edge& new_edge, vector<string>& args )` qui définie comment créer et ajouter un nouvel arc au graphe. Pour créer l'arc dans le graphe, il vous sera nécessaire de l'ajouter dans la structure de Boost Graph grâce à `add_edge_by_label` par exemple qui vous renvoie un objet de type `Edge`. Cet objet doit être ajouté à `edgeMap` en tant que clef et votre structure d'arbre en tant que valeur.
- `void writeGraphVizEdges(ostream& os)` qui définie comment afficher toutes les arêtes de votre graphe.

Le qpn (nœud et arc) est accessible à partir de l'attribut `qpn` défini dans le `qpn_descriptor`.
Votre QPN est prêt à être utilisé

3 Utiliser votre nouveau QPN

Pour utiliser votre graphe, vous devez avoir une instance de `meta_qpn`, vous pouvez alors créer votre QPN et lui ajouter toutes les relations que vous souhaitez avec la méthode `addEdge` que vous avez normalement définie.

Une fois les arcs créés, ajoutez votre QPN au meta_qpn avec la méthode `addQpn(new_qpn)`. Faites de même avec tous les autres QPNs que vous avez pu créer.

À partir du méta_qpn vous pouvez définir des nœuds observés (`observeNodeValue`) ou une variation de signe sur un nœud (`observeNodeSign`). Cette dernière méthode ayant pour effet de propager le signe, il est préférable de la lancer à la fin. Enfin `writeGraphViz(ostream& out)` vous permet d'afficher le graphe au format GraphViz sur le flux de votre choix.