Développement d'un outil d'édition et de manipulation de réseaux bayésiens

Valentin Wegmann-Serin
Télécom Nancy
Mail: valentin.wegmann-serin@telecomnancy.eu

Christophe Simon
Laboratoire du CRAN
Mail: christophe.simon@univ-lorraine.fr

Raphael Agathon
Telecom Nancy
Mail : raphael.agathon@telecomnancy.eu

Java, Matlab, inférence, probabilité, graphe probabiliste

***Abstract*—The abstract goes here.**
[?]

I. Introduction

A. Le contexte

Notre projet s'inscrit dans le cadre de la recherche. Les chercheurs utilisent régulièrement des outils basés sur des graphes probabilistes ou quasi-probabilistes. Par exemple, l'étude de la fiabilité d'un système passe par l'analyse d'un graphe comprenant tous les différents composants du système avec leurs états et probabilités associés -la fiabilité étant l'étude des défaillances des systèmes d'un point de vue statistique-. Les graphes utilisés dans cette étude sont des réseaux bayésiens. Ce sont des modèles probabilistes que l'on trouve sous forme de graphe acyclique orientés. Ces modèles utilisent la méthode de l'inférence qui permet de déduire la probabilité d'un événement à partir de celles d'autres événements déjà évalués. Elle repose sur le théorème de Bayes.

B. Problèmatique

Les chercheurs utilisant principalement l'outil Matlab pour leur simulation, le but de ce projet est de développer une application opensource communiquant avec Matlab et permettant l'usage des réseaux bayésiens.

C. Contribution

D. Plan

Subsection text here.

II. État de l'art

Il existe de nombreuses applications permettant l'édition de réseaux bayésiens. Celles-ci ont toutes leurs spécificités que l'on peut organiser sous forme de catégories.

- Langage de programmation : Il existe des toolbox pour les réseaux bayésiens dans presque tous les langages de programmation. Les plus connus sont Java Bayes qui est en Java et Bayes Net Toolbox sous Matlab. Il existe d'autres application dont on ne connait pas le langage utilisé. Celle-ci sont la plupart du temps payante ou sont des applications web. La plus connue et utilisées est BayesiaLab. Les contraintes imposent que l'outil à développer soit sous Java et Matlab en même temps. Dimple est une API qui possède une version Java et une Matlab, mais il n'y a aucune communication entre les deux ce qui ne correspond pas à la problématique et aux contraintes. DynGraph, l'application utilisée est la seul à répondre à cette contrainte. Cette contrainte d'intercommunication est la plus restrictive car l'outil doit permettre l'utilisation de Matlab et de Java en parallèle. Toutes les modifications faites dans l'un des langage doit directement impacter l'autre.
- Interface graphique : Parmi toutes les applications connues de réseaux bayésiens, seules 30% possèdent une interface graphique. La plupart de celles-ci sont en Java comme Java Bayes ou payante comme Bayesialab. Bayes Net Toolbox, CRFtoolbox, DBNbox, OpenGM2 et Probabilistic Modeling Toolkit sont toutes les applications codées sous Matlab. Aucune d'entre elles ne possèdent d'interface graphique. La contrainte de l'interface graphique impose que 'outil en possède déja une pou que celle-ci soit codée par dessus. DynGraph possède une interface graphique en Java Swing, qui offre une grande liberté (présentation, affichage,...) contrairement à ce qui peut être fait sous Matlab.
- Le prix : l'outil doit être open source et gratuit. La plupart des applications existantes le sont, mais certaines sont payantes comme BayesiaLab qui à été utilisé dans la processus de construction de l'outil de développement de réseaux bayésiens. DynGraph à été codée par des étudiants de notre école,dans le même environnement que le nôtre. Cette application est donc open source et gratuite et nous pouvons facilement communiquer avec les développeurs.
- Réseaux bayésiens et Inférence :Les différentes applications permettent de générer des réseaux bayésiens et de faire de l'inférence sur ceux-ci. Il existe différents algorithme pour le calcul de l'inférence comme JTree et VarElim. DynGraph, l'outil utilisé ne possède aucun outil d'édition de réseaux bayésiens ou de calcul d'inférence. Cet outil permet de construite des graphes pouvant être

à nœuds colorés et arcs évalués orienté ou non.
Pour pallier à la non présence de réseaux bayésiens dans l'outil DynGraph il été choisit de combiner cet outil avec une application Java d'édition de réseaux bayésiens. L'application sélectionnée est Jayes. Jayes est un éditeur de réseaux bayésiens, open source et gratuit qui ne dispose pas d'interface graphique. Parmi les autres applications Java, Jayes possède un code facile de compréhension et permettant une réécriture du code plus facile.

III. Méthodologie

Pour arriver à l'objectif, le travail à été divisé en deux. Une partie de compréhension et de modification de l'outil DynGraph réaliser par M. Wegmann-Serin et une seconde partie sur la compréhension et l'implémentation des réseaux bayésiens ainsi que des calculs d'inférences. Cette seconde partie à été réalisée par M. Agathon.

A. Analyse de l'outil DynGraph

DynGraph est un outil permettant la création et l'édition de graphe. Celui-ci est implémenté en java et se lance depuis Matlab. L'interface (cf fig 1) est composé d'un menu basique et d'une barre composé de boutons permettant d'éditer le graphe. L'éditeur ne permet actuellement que de créer des graphes qui sont orientés ou non, colorés ou non. L'éditeur ne permet pas d'imposer des contraintes lors de la création du graphe telle que l'acyclicité ou la non-évaluation des arcs. Chaque graphe crée est représenté par une liste de nœuds et d'arcs. Un arc étant défini par un noeud-début et un noeud-fin et il n'est décrit que par cela. De plus, ce logiciel n'est qu'une visualisation de graphe et ne permet pas de faire des calculs. En effet, toute la partie utile aux calculs n'est pas présente : la liste des successeurs et prédécesseurs ou une matrice d'adjacence ou d'incidence. Seul le côté front-end est très implémenté. Pour ce qui est du back-end, il s'agit de la communication entre Java et Matlab qui est présente puisque chaque action effectué sur l'interface - en java - doit être fait en Matlab et inversement.

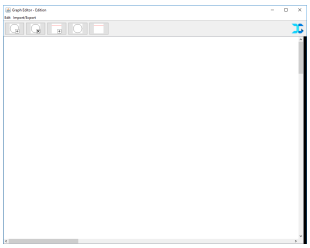

Fig. 1. interface

B. Implémentation des réseaux bayésiens

Un réseau bayésien est un graphe acyclique orienté non coloré. Les arcs ne possèdent pas de valeurs. Chaque nœud possède une table de probabilité qui dépend de ses états (vrai ou faux mais d'autres peuvent être ajouté) mais aussi des états de ses nœuds pères.

a) Algorithme d'acyclicité: Ainsi, pour transformer un graphe en réseau bayésien il faut vérifier la contrainte d'acyclicité. Les autres propriétés des réseaux bayésiens peuvent être remplies par de simples de modifications (décoloration et suppression des poids sur les arcs) du graphe qui n'impactent pas les données présente dans celui-ci.
Il existe différents algorithmes de détection de cycle dans un graphe. Des algorithmes comme celui du tri topologique, ou celui de Floyd's Tortoise and Hare sont performants mais nécessites des outils qui nécessite une totale refonte de la gestion des graphes dans l'outil DynGraph.
Nous avons donc décidé de coder notre propre algorithme de détection de cycle. Cet algorithme a été spécifiquement construit pour notre cas d'étude : permettre à un utilisateur de savoir si il y a un cycle dans le graphe et si oui afficher la ou les zones à problèmes.
L'algorithme repose sur le fait que tous nœud uniquement père ou uniquement fils ne fait pas partie d'un cycle. Par suppression successive de ces nœuds et arcs associes on peut déterminer qu'il y a un ou des cycles dans la ou les zones restantes.

```
Data: Graphe G contenant une liste de nœuds et une
      liste d'arc
Result: liste des nœuds à problème : N
N liste de nœuds initialiser avec les noeuds de G;
A liste d'arcs initialiser avec les arcs de G;
test = True;
while test = True do
    Ra liste d'arcs initialement vide;
    Rp liste des nœuds père, initialement vide;
    Rf liste des nœuds fils initialement vide;
    Rnpf liste des nœuds non père et non fils initialement
     vide;
    test = False;
    forall Arc a dans A do
        Ajout du noeud head à la liste Rp;
        Ajout du noeud tail à la liste Rf;
    end
    forall Node n dans N do
        Ajout des nœuds non présent dans l'intersection
         entre Rp et Rf dans Rnpf;
    end
    if Rnpf de taille non nulle then
        test = True;
        forall Arc a dans A do
            if Si un des nœuds de a appartient à Rnpf
             then
                Ajout de l'arc a dans la liste Ra;
            end
        end
        Suppression des nœuds de Ra dans la liste d'arcs
         A;
        Suppression des nœuds de Rnpf dans la liste de
         nœuds N;
    end
end
```

Algorithm 1: Algorithme de détection de cycle

Cet algorithme permet de détecter la présence de cycle dans un graphe. Cet algorithme ne permet pas d'avoir précisément les nœuds inscrits dans un cycle. Il est possible que certains nœuds, non présent dans un cycle soient inscrits dans les nœuds à problème. Cela car ils sont à l'intérieur d'un ou plusieurs cycle et sont donc mis en nœuds à problème.
Cet algorithme est suffisant pour la contrainte d'acyclicité. Il ne surcharge pas le graphe.

b) Transformation du graphe en réseau bayésien:

c) implémentation des probabilités:

C. Modification de l'interface graphique

Parler de la division en couche/Étape du code. Une étape pour la création de graphe, une pour la mise en place des probabilité, une pour le calcul d'inférence. l'objectif est de réduire le nombre de contrôle à effectuer a chaque modification du programme. Parler aussi des habitudes des gens qui utilisent ce genre de programme : Une création de graphe et 0 modifications, quelques modifications des probabilités et beaucoup de calculs d'inférences

D. Calcul de l'inférence

E. Communication Matlab

Subsection text here.

IV. Résultats

V. Conclusion

VI. Remerciements

Nous souhaitons adresser nos remerciements aux personnes nous ayant aider dans la rédaction de cet article, en particulier à M.Simon, notre encadrant de PIDR qui a su nous éclairer dans l'avancée de notre projet.

VII. annexe

[**?**] [**?**]