Librairie aGrUM : *a Graphical Universal Model*

Christophe Gonzales, Lionel Torti, Pierre-Henri Wuillemin

Sorbonne Universités, UPMC Univ Paris 06
UMR 7606
Paris, France
prenom.nom@lip6.fr

RÉSUMÉ. *Cet article présente aGrUM, une bibliothèque C++ qui implémente des modèles graphiques pour l'aide à la décision. Les modèles implémentés incluent les Réseaux Bayésiens, les Diagrammes d'Influences, les Réseaux Crédaux et les Modèles Probabilistes Relationnels. Cette bibliothèque est le résultat d'un effort continu de chercheurs et d'industriels pour fournir une solution open source multi plateforme (Linux, MacOS X et Windows 10), performante et maintenue. La bibliothèque fournit également des extensions pour utiliser aGrUM avec d'autres langages de programmation comme Python.*

ABSTRACT. *This paper presents the aGrUM framework, a C++ library providing state-of-the-art implementations of graphical models for decision making, including Bayesian Networks, Influence Diagrams, Credal Networks, Probabilistic Relational Models. This is the result of an ongoing effort of researchers and industrials to build an efficient and well maintained open source cross-platform software (running on Linux, MacOS X and Windows 10) for dealing with graphical models. The framework also contains wrappers for exploiting aGrUM within other programming languages like Python.*

MOTS-CLÉS : *PGM, BN, PRM, Framework, Open Source*

KEYWORDS: *PGM, BN, PRM, Framework, Open Source*

1. Introduction

Le projet **aGrUM** existe depuis huit ans au sein du département d'intelligence artificielle de l'université Pierre et Marie Curie (http://lip6.fr). Développé par plusieurs contributeurs, en particulier les auteurs de cet article, le projet est devenu une bibliothèque *open-source* complète, traitant d'un grand nombre de modèles graphiques. L'écosystème d'**aGrUM** comprend la bibliothèque C++ éponyme, plusieurs extensions et quelques applications, toutes fonctionnelles sous Linux, MacOS X et Windows 10 (les compilateurs supportés incluent g++, clang, mvsc et mingw). La bibliothèque est disponible gratuitement sur le site du projet http://agrum.lip6.fr [1]. Le site http://pyagrum.lip6.fr est dédié à l'extension Python: **pyAgrum**.

La création d'**aGrUM** a été motivée par le besoin d'une bibliothèque permettant la modélisation à l'aide de modèles graphiques pour l'aide à la décision (i.e. Réseaux Bayésiens, Diagrammes d'Influences, *etc.*) mais aussi l'étude expérimentale de nouveaux algorithmes sur ces mêmes modèles. Cette bibliothèque devait donc être à la fois performante, facile d'utilisation, aisément extensible et correctement maintenue. Pour son développement, l'accent est mis sur la performance, la qualité de code et l'utilisabilité. Aujourd'hui, la bibliothèque **aGrUM** est utilisée autant par des universitaires que par des industriels à travers le monde, tant à la fois comme utilisateurs que comme contributeurs. Les projets européens DREAM, MIDAS et SCISSOR ainsi que les projets ANR SKOOB, INCALIN, LARDONS et DESCRIBE l'ont utilisée. Elle constitue également l'infrastructure de base pour des travaux de recherche et elle est utilisée dans plus de cinquante articles de conférences internationales et revues scientifiques.

2. Fonctionalités d'aGrUM

La bibliothèque C++ **aGrUM** est divisée en sept modules, dont la majorité font références à différents modèles graphiques:

1. **BN** : Réseaux Bayésiens.
2. **Learning** : algorithmes d'apprentissage de Réseaux Bayésiens.
3. **CN** : Réseaux Crédaux.
4. **FMDP** : Processus Décisionnels de Markov Factorisés.
5. **ID** : Diagrammes d'Influence.
6. **PRM** : Modèles Probabilistes Relationnel.
7. **Core** : structure de données communes et utilitaires.

1. Le site contient également des guides pour l'installation, la documentation et le support.

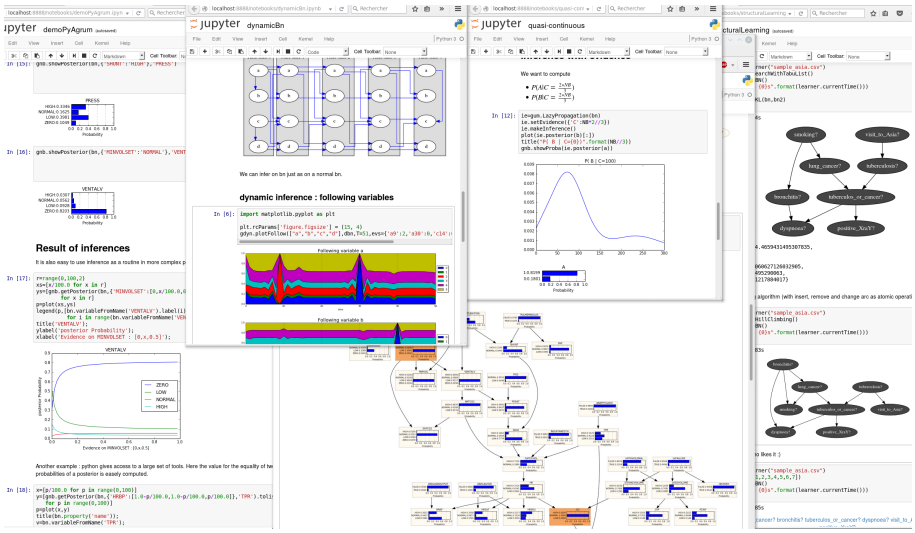

FIGURE *1. Notebooks IPython utilisant* ***pyAgrum***.

2.1. Réseaux Bayésiens (BN)

Le module BN fournit une implémentation flexible et efficace des Réseaux Bayésiens. Ceux-ci peuvent être lus (et écrits) dans un grand nombre de formats (BIF, DSL, net, cnf, BIFXML, UAI). Ils peuvent également être générés (aléatoirement) depuis plusieurs générateurs ou appris à partir de base de données en utilisant le module Learning.

La bibliothèque **aGrUM** permet aux utilisateurs de définir les paramètres des BNs via des Tables de Probabilités Conditionnelles (CPT) classiques, ou via des Noiy OR, Noisy AND, des modèles Logit, des agrégateurs (*and*, *or*, *min*, *max*, *forall*, *exists*, etc.). Afin d'améliorer les performances, les CPTs peuvent être encodées en utilisant différentes structures de données (tableaux, matrices creuses, diagrammes de décision algébrique, *etc*). Différents algorithmes d'inférence tels que Lazy Propagation, Shafer-Shenoy, Variable Elimination, Gibbs sampling, *etc.*, incluant différentes méthodes de *relevant reasoning* permettent de mener efficacement des calculs sur ces modèles.

Un module spécifique s'occupe de l'apprentissage de structures et/ou de paramètres de BNs depuis un jeu de données. Actuellement, ces données peuvent être des fichiers CSV ou peuvent être stockées dans une base de données SQL. La bibliothèque est ici aussi conçue pour être aussi flexible que possible et suit une approche par composant : les algorithmes d'apprentissage de structures sont composés d'un module pour lire la base de données, d'un autre pour le score (BD, BDeu, K2, AIC, BIC/MDL) avec la possibilité d'ajouter des *a priori* (lissage ou Dirichlet) (Heckerman

et al., 1995), (Cooper, Herskovits, 1992), (Buntine, 1991). Un composant pour monitorer les changement locaux de la structure permet de définir des contraintes telles qu'interdire/obliger la présence d'arcs, limiter les degrés entrant et sortant, ou encore imposer un ordre partiel sur les nœuds. Les algorithmes d'apprentissages actuellement implémentés comprennent *greedy hill climbing*, recherche locale avec tabou et K2. Les paramètres de BNs peuvent également être appris avec des algorithmes de maximum de vraisemblance ou de maximum a posteriori. Tous ces algorithmes sont fortement parallélisés.

2.2. Autres modules

D'autres modèles graphiques sont implémentés dans la bibliothèque : les Réseaux Crédaux (module CN), les Processus Décisionnels de Markov Factorisés (module FMDP), les Diagrammes d'Influence (module ID) et les Modèles Probabilistes Relationnels (module PRM). Tous les modules suivent la même philosophie que le module BN : forte flexibilité, inférences efficaces, formats de fichiers exhaustifs. De plus, tous ces modèles sont fournis avec des algorithmes d'inférences sur-mesures, e.g., *loopy propagation* et *Monte Carlo* pour CN (Hourbracq *et al.*, 2013), SPUFF pour FMDPs (Magnan, 2016), Shafer-Shenoy pour IDs, *etc*.

Tous les modules cités précédemment utilisent les structures de données et algorithmes implémentés dans le module "Core". Celui-ci inclut des structures de données classiques tel que les listes, les tables de hachages, les arbres de recherches AVL, les ensembles, les piles, *etc*, qui ont été implémentés dans la bibliothèques de sorte qu'elles soient à la fois sécurisées et particulièrement performantes. Des structures de données et des algorithmes plus complexes sont également disponibles, tel que les graphes avec, entres autres, toute une hiérarchie d'algorithmes de triangulations, notamment les versions incrémentales. Plusieurs implémentations des tables multidimensionnelles sont également proposées : représentations tabulaires, matrices creuses, diagramme de décision algébrique, Noisy OR, Noisy AND, logit, agrégateurs. Le module "Core" de la bibliothèque **aGrUM** fournit également des outils permettant d'assurer des critères de qualité élevé et la détection de fuites de mémoire.

3. Exemples d'utilisation des Réseaux Bayésiens avec aGrUM

L'exemple 1 suivant illustre comment définir un BN à l'aide d'aGrUM en C++. Il reprend le problème de l'arrosoir [2] illustré par la figure 2. Ce problème met en relation la pluie (variable Rain) qui influence l'activation de l'arrosoir (variable Sprinkler) et qui toutes les deux influencent l'humidité de la pelouse (variable Grass).

Dans l'exemple 1, les types suivants sont utilisés :

– `BayesNet`: représente un Réseau Bayésien.

2. Source https://en.wikipedia.org/wiki/Bayesian_network

– `NodeId`: identifie un nœud dans un graphe.
– `LabelizedVariable`: représente une variable aléatoire discrète.
– `Potential`: représente une table de probabilités conditionnelles (CPT).

```
auto bn = BayesNet<double>("Sprinkler");

// nodes
NodeId rain      = bn.add(LabelizedVariable("R", "Rain"        , 2));
NodeId sprinkler = bn.add(LabelizedVariable("S", "Sprinkler on", 2));
NodeId grass     = bn.add(LabelizedVariable("G", "Grass wet"   , 2));

//arcs
bn.addArc(rain     , sprinkler);
bn.addArc(rain     , grass);
bn.addArc(sprinkler, grass);

//cpts
bn.cpt(rain).fillWith({0.2, 0.8});
bn.cpt(sprinkler).fillWith({0.4 , 0.6 ,
                            0.01, 0.99});
bn.cpt(grass).fillWith({0.0  ,1.0 ,
                        0.8  ,0.2 ,
                        0.9  ,0.1 ,
                        0.99 ,0.01});
```

Exemple 1 – Création d'un BN avec aGrUM en C++

Les tables de probabilités conditionnelles sont définies par des instances de la classe `Potential<double>`, une implémentation de tables multidimensionnelles, qui peuvent être remplies à l'aide de `std::vector<double>`. Les CPT dans aGrUM somment à un *par ligne*. Par exemple, la variable `G` de notre exemple a deux parents `R` et `S`, la première valeur de CPT correspond donc à $P(G = False|R = False, S = False)$. La valeur suivante est $P(G = True|R = False, S = False)$, la valeur suivante $P(G = False|R = True, S = False)$ et ainsi de suite. Cet ordre revient à définir les CPT où chaque ligne représente $P(X|X_1 = x_1, \ldots, X_n = x_n)$ où $(X_1, \ldots, X_n)$ sont les parents de X dans le BN.

```
LazyPropagation<double> lazy(bn);
lazy.makeInference();

Potential<double> p = lazy.posterior(grass);
std::cout << p << std::endl;

// Observing rain
lazy.addHardEvidence(rain, 0);
lazy.makeInference();
p = lazy.posterior(grass);
std::cout << p << std::endl;
```

Exemple 2 – Inférence probabiliste avec aGrUM en C++

L'exemple 2 illustre comment invoquer un des algorithmes d'inférence d'aGrUM, par exemple ici, l'algorithme *Lazy Propagation* (Madsen, Jensen, 1999).

Tous les algorithms d'inférences respectent une même interface, définie par la classe `BayesNetInference`. Le résultat de l'exemple 2 est :

```
<G:0> :: 0.89848 /<G:1> :: 0.10152
<G:0> :: 0.54 /<G:1> :: 0.46
```

Exemple 3 – Résultat de l'exemple 2

L'apprentissage de BN avec aGrUM est conçu pour être le plus configurable possible, comme le montre l'exemple 4 où le BN Asia (Lauritzen, Spiegelhalter, 1988) est appris. L'exemple utilise l'algorithme de recherche local avec tabou et un score de log-vraisemblance. Il montre une partie des paramètres possibles : borne sur le nombre d'arcs entrant ou encore arcs obligatoires. Le score de log-vraisemblance n'est pas le seul disponible : sont disponibles les scores AIC, BD, BDeu, BIC ou encore K2. Il est également possible d'utiliser des *a priori* (lissage ou de Dirichlet). Finalement, l'algorithme d'apprentissage peut être chois parmi K2, *greedy hill climbing* ou encore recherche locale avec tabou (celui utilisé dans l'exemple).

```
std::string file = "asia.csv";
learning::BNLearner<double> learner( file );

learner.useLocalSearchWithTabuList( 100, 1 );
learner.setMaxIndegree( 10 );
learner.useScoreLog2Likelihood();

learner.addMandatoryArc( "bronchitis?", "lung_cancer?" );

learner.useAprioriSmoothing();
learner.setAprioriWeight( 1 );

BayesNet<double> bn = learner.learnBN();
```

Exemple 4 – Apprentissage d'un BN avec aGrUM en C++

4. Extensions

La bibliothèque **aGrUM** propose également des extensions pour d'autres langages de programmation. Ainsi **pyAgrum** est une extension Python d'**aGrUM**. **aGrUM** implémente également des langages dédiés de modélisations spécifiques tels que le langage **O3PRM** [3].

4.1. pyAgrum

pyAgrum est un *wrapper* Python d'une partie de la bibliothèque C++ **aGrUM**. Elle fournit une interface facile d'accès pour manipuler les modèles graphiques présents dans **aGrUM** tout en préservant les performances élevées de la bibliothèque

3. http://O3PRM.lip6.fr.

C++. Ceci lui permet d'être plus rapide de plusieurs ordres de grandeurs que d'autres paquets Pythons tels que **pgmpy**.

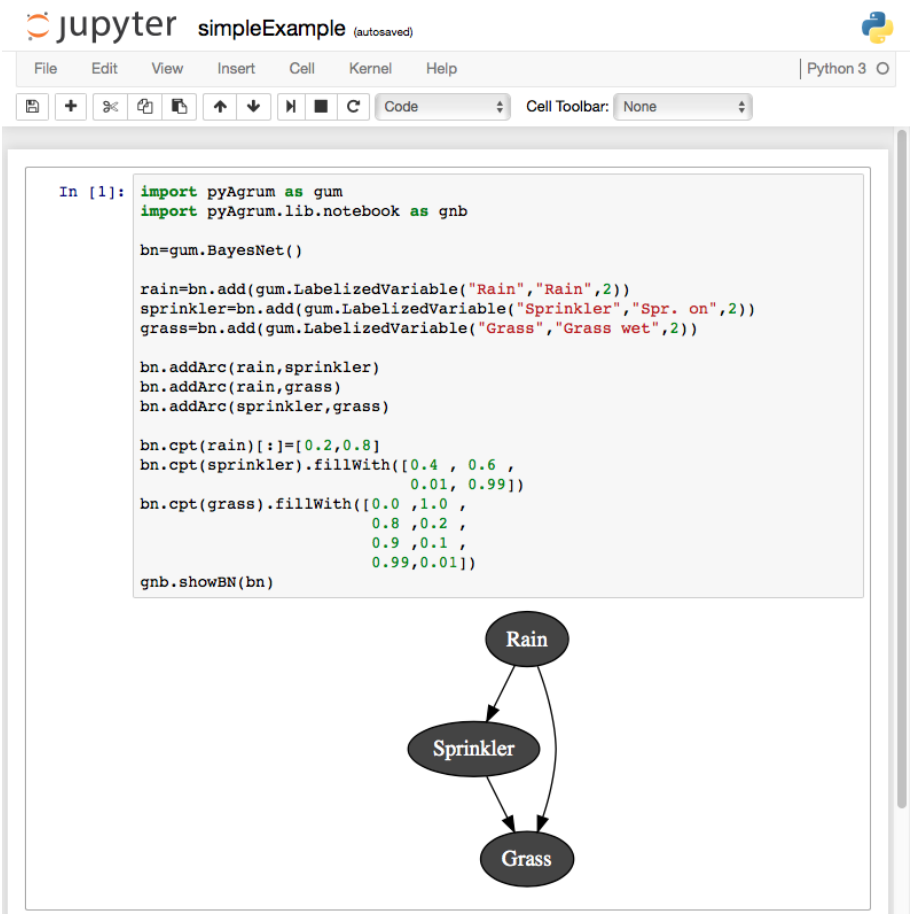

FIGURE 2. *Le BN de l'exemple 1 en python.*

Grâce aux outils de python, particulièrement les *notebooks* IPython (**Jupyter**), **pyAgrum** peut facilement être utilisé comme interface graphique. La figure 1 montre ces notebooks, illustrant comment lla modélisation d'un BN et les inférences peuvent être réalisées. Un intérêt non négligeable d'un *wrapper* python est l'accès aux nombreux autres *package* python comme **numpy** ou **pandas** (http://pandas.pydata.org).

Actuellement, pyAgrum propose une interface pour les Réseaux Bayésiens, les Réseaux Crédaux et les Diagrammes d'Influences. Toutes ces fonctionnalités rendent **pyAgrum** particulièrement polyvalent et en font un paquet de modèle graphiques pro-

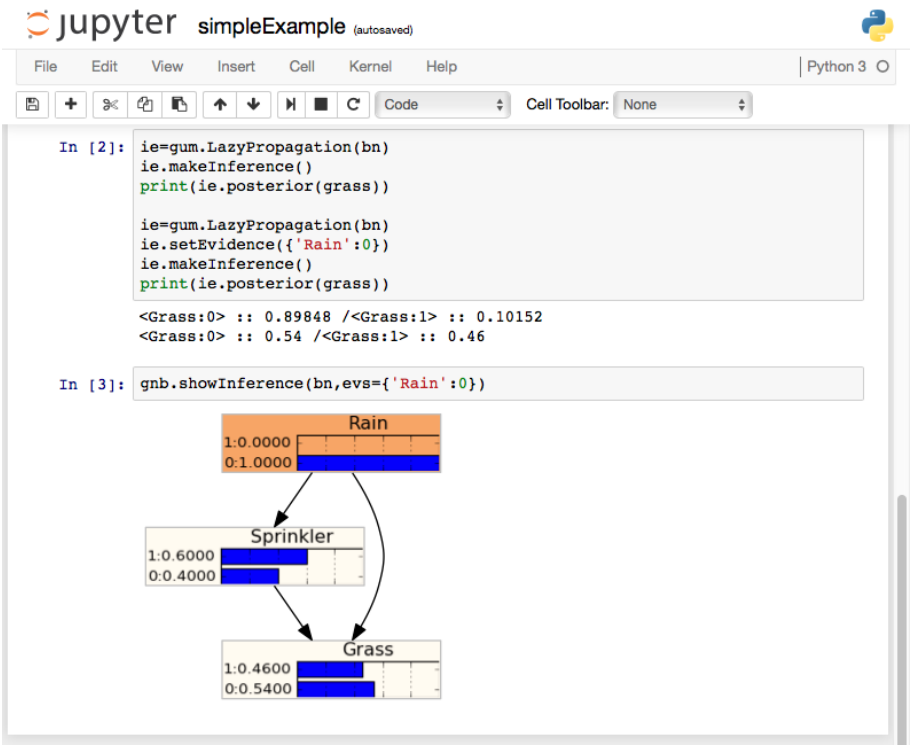

FIGURE *3. pyAgrum permet également les visualisations graphiques des résultats.*

babilistes efficace. Des tutoriaux, démonstrations, téléchargement et instruction d'installation peuvent être trouvés sur le site du projet http://pyagrum.lip6.fr.

4.2. O3PRM

La bibliothèques **aGrUM** contient un module spécifique appelé PRM pour les Modèles Probabilistes Relationnel (*Probabilistic Relational Models*). Ceux-ci sont une extension orienté-objet complète des Réseaux Bayésiens défini dans (Torti *et al.*, 2010) : le module implémente les notions de classes, interfaces, instances, attributs, références, chaînes de références, systèmes, *etc*. L'aspect orienté-objet améliore grandement l'expressivité des PRMs par rapport aux BNs et réduit les coûts de création et de maintenance pour la modélisation de systèmes complexes avec une forte répétition de sous-composants.

Un algorithme particulièrement efficace tel que *Structured Variable Elimination* (SVE) ou SVE avec *relevant reasoning* (Torti *et al.*, 2013) sont fournis avec ce module. Un lien avec le module BN permet de réduire un PRM à un BN et, de ce fait,

rend accessible l'ensemble des algorithmes d'**aGrUM** disponibles pour les BNs. Finalement, un langage dédié pour la modélisation de PRM, **O3PRM**[3] a été développé et permet aux utilisateurs de facilement créer des PRMs.

```
class Weather {
  boolean rain {[ 0.2, 0.8 ]};
}

class Garden {
  Weather weather;

  boolean sprinkler
  dependson weather.rain
  {[ 0.4, 0.01,
     0.6, 0.99 ]};

  boolean grass_is_wet
  dependson weather.rain, sprinkler
  {[ 0.0, 0.8, 0.9, 0.99,
     1.0, 0.2, 0.1, 0.01 ]};
```

```
system aGarden {
  Weather w;
  Garden g;
  g.weather = w;
}

system manyGardens {
  Weather w;
  Garden g[5];
  // Assigne w a toute les
  // references weather
  // des instances dans g
  g.weather = w;
}
```

Exemple 5 – L'exemple de l'arrosoir en O3PRM.

L'exemple 5 est une représentation du BN de l'arrosoir exprimée à l'aide du langage O3PRM. Cet exemple montre le principal intérêt du langage O3PRM puisqu'en décomposant ce problème en deux classes `Weather` et `Garden`, il est possible de représenter le BN de l'exemple 1 (système `aGarden`) mais également de le généraliser (sans rajouter une seule définition de CPT) au modèle où il y aurait k jardins différents mais naturellement un seul `Weather` (système `manyGardens`). Il serait également possible de rendre le problème pluscomplexe à moindre coût en autorisant, par exemple, plusieurs arrosoirs par jardin, etc.

5. Conclusion

Cet article présente **aGrUM**, une puissante bibliothèque C++ dédiée à la manipulation et l'exploration des modèles graphiques pour l'aide à la décision. Elle est conçue pour être flexible, maintenue à jour et particulièrement performante. Le cœur du projet est la bibliothèque C++ **aGrUM** mais des extensions tels que **pyAgrum** permetttent aux utilisateurs d'exploiter **aGrUM** avec des langages de haut niveaux.

Le développement d'**aGrUM** n'a pas uniquement été motivé par la recherche académique, elle a aussi été le résultat de différentes collaborations industrielles. Ainsi, le langage **O3PRM** est exploité dans une collaboration continue avec EDF pour la gestion des risques dans les centrales nucléaires et avec IBM pour l'exploitation de probabilité dans des systèmes experts à base de règles. Le module d'apprentissage de Réseau Bayésien est utilisé dans des projets avec l'IRSN pour la reconstruction de scénario d'incident nucléaire. D'autres projets avec Airbus et *Open Turns* utilisent **aGrUM** pour l'apprentissage structurel de copules avec des variables continues.

Actuellement, les auteurs d'**aGrUM** projettent de créer un consortium rassemblant les utilisateurs et les contributeurs du projet. Ceci devrait permettre d'augmenter les

ressources allouées au projet. Bien qu'ajouter de nouvelles fonctionnalités soit prioritaire, beaucoup d'efforts sont consacrés à améliorer la qualité de la documentation, du code (exploitant des outils tel que SonarQubehttp://www.sonarqube.org). Des outils de déploiement automatique de *nightly builds* (i.e. Jenkinshttps://jenkins.io) ont également été mis en place.

Bibliographie

Buntine W. (1991). Theory refinement on bayesian networks. In *Proceedings of the seventh conference on uncertainty in artificial intelligence*, p. 52–60.

Cooper G. F., Herskovits E. (1992). A bayesian method for the induction of probabilistic networks from data. *Machine Learning*, vol. 9, n° 4, p. 309–347. Consulté sur http://dx.doi.org/10.1007/BF00994110

Heckerman D., Geiger D., Chickering D. (1995, March). Learning bayesian networks: The combination of knowledge and statistical data. *Machine Learning*, vol. 20, n° MSR-TR-94-09, p. 197-243. Consulté sur http://research.microsoft.com/apps/pubs/default.aspx?id=65088

Hourbracq M., Baudrit C., Wuillemin P.-H., Destercke S. (2013, juillet). Dynamic Credal Networks: introduction and use in robustness analysis. In F. Cozman, T. Denoeux, S. Destercke, T. Seidenfeld (Eds.), *Eighth International Symposium on Imprecise Probability: Theories and Applications (ISIPTA 2013)*, p. 159-169. Compiegne, France. Consulté sur https://hal.archives-ouvertes.fr/hal-00861994

Lauritzen S. L., Spiegelhalter D. J. (1988). Local computations with probabilities on graphical structures and their application to expert systems. *Journal of the Royal Statistical Society. Series B (Methodological)*, vol. 50, n° 2, p. 157–224.

Madsen A. L., Jensen F. V. (1999). Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, vol. 113, n° 1, p. 203–245.

Magnan J.-C. (2016). *Représentations graphiques de fonctions et processus décisionnels markoviens factorisés*. Thèse de doctorat non publiée, Université Pierre et Marie Curie (UPMC). (Type : Thèse de Doctorat – Soutenue le : 2016-02-02 – Dirigée par : Gonzales, Christophe – Encadrée par : Wuillemin Pierre-Henri)

Torti L., Gonzales C., Wuillemin P.-H. (2013). Speeding-up structured probabilistic inference using pattern mining. *International Journal of Approximate Reasoning*, vol. 54, n° 7, p. 900–918.

Torti L., Wuillemin P.-H., Gonzales C. (2010). Reinforcing the object-oriented aspect of probabilistic relational models. In *Proceedings of the 5th workshop on probabilistic graphical models (pgm)*, p. 273–280.