PROCOP : probabilistic rules compilation and optimisation

Gaspard Ducamp

PROCOP :

Probabilistic Rules Compilation and Optimisation

par Gaspard Ducamp

Thèse de doctorat en informatique

encadrée par Philippe Bonnard et Pierre-Henri Wuillemin

présentée et soutenue publiquement le 13 avril 2021

Jury :

Philippe Leray	Prof., Polytech'Nantes, LS2N	Rapporteur
François Fages	Prof., INRIA Saclay, LIFEWARE	Rapporteur
Véronique Delcroix	MdC, UPHF, LAMIH	Examinatrice
Anthony Nouy	Prof., Centrale Nantes, LMJL	Examinateur
Christophe Gonzales	Prof., Université Aix Marseille, LIS	Directeur
Philippe Bonnard	Ing., IBM France	Co-encadrant
Pierre-Henri Wuillemin	MdC, Sorbonne Université, LIP6	Co-encadrant

Remerciements

Cette thèse, rendue possible grâce au soutien de l'ANRT et à la confiance qui m'a été accordée par le LIP6 et IBM France, a été rendue agréable et enrichissante par les personnes que j'ai eu l'occasion de fréquenter pendant ces trois années.

À cette occasion, j'aimerais remercier chaleureusement Philippe Bonnard et Pierre-Henri Wuillemin pour leur encadrement sans faille, depuis le début de mon stage de Master jusqu'à la soutenance de cette thèse. Au-delà de leur pédagogie et de la qualité de leur investissement dans la supervision de mes travaux, c'est aussi leurs bons mots et leur bienveillance qui ont rendu cette expérience si positive.

La présence de Christian de Sainte-Marie a su rendre l'exercice de rédaction en confinement plus vivable. Nos discussions hebdomadaires, parfois techniques, souvent légères, toujours joviales ont été stimulantes -donc essentielles- et je l'en remercie.

J'exprime également toute ma gratitude à Philippe Leray et François Fages pour avoir accepté de rapporter cette thèse et pour leurs retours encourageants. Je profite de cette occasion pour remercier profondément les autres membres du jury, Véronique Delcroix dont la relecture attentive du manuscrit a été d'une grande aide, Christophe Gonzales qui a accepté de diriger cette thèse et Anthony Nouy sans qui une partie importante de ces travaux n'auraient pas été possible.

Si mon encadrement a été remarquable, mes collègues, camarades et amis l'ont été tout autant. Aussi bien au sein du France Lab, où j'ai toujours été très bien accueilli et où j'ai facilement trouvé de l'aide pour toutes mes questions qu'au sein du LIP6, où la bonne humeur et l'entrain régnaient. Les discussions techniques et matchs de ping-pong de l'un, les débats interminables et les parties de jeux de société de l'autre marqueront ma mémoire et me feront regarder cette aventure avec beaucoup de bonheur.

J'aimerais particulièrement remercier mes camarades doctorants, notre vie de groupe m'a aidé à passer avec bonne humeur des moments pourtant éprouvants, de la dégustation des plats du CROUS jusqu'aux descentes acrobatiques du mont Buet.

Plus simplement : Adèle, Alexandre, Anne-Elisabeth, Bastien, Bruno, Cassandre, Clarouche, David S., David W., Fanny, Franco, Hugo, Ismaïl, Kostas, Lionel, Nadjet, Nawal, Nicolas, Olivier, Parham, Patrice, Pierre, Pierre-André, Santiago, Stéphane, Thibaut, Yoann et tous les autres, merci. Si tout s'est si bien passé, c'est grâce à vous.

Borg avait McEnroe, Hunt avait Lauda, j'ai eu Marvin. Vif, il répondait à mes questions comme le premier renvoyait les balles. Drôle, il dérapait comme le second... dérapait. Si j'ai réussi à terminer ce marathon, c'est aussi parce qu'il a su le rendre intéressant, sans aucun temps mort. Travailler avec lui était un plaisir quotidien, son accent, son mauvais esprit et nos échanges me manqueront. À l'heure où j'écris ces lignes, il termine sa propre course et j'ai hâte de le retrouver de l'autre côté de la ligne d'arrivée.

Pour terminer, j'aimerais remercier ceux qui m'ont encouragé et soutenu tout du long : ma famille, mes amis, Nellson, Groucho et Fifi et bien évidemment Lise, dont la patience a été remarquable.

Résumé

Adoptées depuis plus de 20 ans par le monde de l'industrie, les règles métiers (business rules) offrent la possibilité à des utilisateurs non-informaticiens de définir des politiques de prise de décision de manière simple et intuitive. Pour faciliter leurs utilisations, des systèmes à base de règles, dits « systèmes de gestion des règles métier », ont été développés, séparant la logique métier de l'application informatique. S'ils sont adaptés pour traiter des données structurées et complètes, ils ne permettent pas aisément de travailler sur des données probabilistes.

PROCOP (Probabilistic Rules Optimized COmPilation) est une thèse proposant une nouvelle approche pour l'intégration de raisonnement probabiliste dans IBM Operational Decision Manager (ODM)[1], le système de gestion des règles métier développé par IBM, notamment via l'introduction d'une notion de risque global sur l'évaluation des conditions d'exécution d'une action, complexifiant la phase de compilation du système mais augmentant l'expressivité des règles métiers.

Diverses méthodes sont explorées, implémentées et comparées afin de permettre l'utilisation d'une telle capacité de raisonnement à large échelle, notamment afin de répondre aux problématiques liées à l'utilisation de modèles graphiques probabilistes dans des réseaux complexes.

[1] `https://www.ibm.com/fr-fr/products/operational-decision-manager`

Abstract

Widely adopted for more than 20 years in industrial fields, business rules offer the opportunity to non-IT users to define decision-making policies in a simple and intuitive way. To facilitate their use, rule-based systems, known as *business rule management systems*, have been developed, separating the business logic from the computer application. While they are suitable for processing structured and complete data, they do not easily allow working with probabilistic data.

PROCOP (Probabilistic Rules Optimized COmPilation) is a thesis proposing a new approach for the integration of probabilistic reasoning in IBM Operational Decision Manager (ODM)[2], IBM's business rules management system, in particular through the introduction of a concept of global risk on the evaluation of the execution conditions of an action, complicating the compilation phase of the system but increasing the expressiveness of the business rules.

Various methods are explored, implemented and compared in order to allow the use of such a powerful reasoning capacity on a large scale, in particular in order to answer the problems linked to the use of probabilistic graphical models in complex networks.

[2] `https://www.ibm.com/products/operational-decision-manager`

Contents

Notations

PGMs

P	a probability distribution
X	a random variable, BN's node or class attribute
$\mathbf{X}$	a set of random variables
$\mathcal{C}$	a PRM class
$\boldsymbol{\mathcal{C}}$	a set of PRM classes
Val(X)	the set of values of X, also called its domain
Val($\mathbf{X}$)	a set obtained by the Cartesian product $\text{Val}(\mathbf{X}) = \otimes_{\text{X}\in\mathbf{X}} \text{Val}(\text{X})$
$\vec{\mathcal{G}} = (\mathcal{V}, \mathcal{A})$	a directed graph over nodes $\mathcal{V}$ and arcs $\mathcal{A}$
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	an undirected graph over nodes $\mathcal{V}$ and edges $\mathcal{E}$
$\text{Pa}_{\text{X}}^{\vec{\mathcal{G}}}$	the parents of X in $\vec{\mathcal{G}}$

Tensors

$\mathbf{T}$	a tensor
$\widetilde{\mathbf{T}}_{\epsilon}$	the ϵ-approximation of a tensor
$\|\mathbf{A}\|_F$	Frobenius norm
$\mathbf{A} \circledast \mathbf{B}$	Hadamard product
$\mathbf{A} \otimes \mathbf{B}$	Kronecker product
$\mathbf{A} \boxtimes \mathbf{B}$	Partial Kronecker product
$\mathbf{A} \times^1 \mathbf{B}$	Mode-(M,1) contracted product

Common acronyms

AI	Artificial Intelligence
BAL	Business Action Language
BIS	Bayesian Insight System
BN	Bayesian Network
BR	Business Rule
BRMS	Business Rules Management System
CPT	Conditional probability Table
DAG	Direct Acyclic Graph
DBN	Dynamic Bayesian Network
IRL	ILOG Rule Language
IT	Information Technology
ODM	IBM Operational Decision Manager
OM	Object Model
PGM	Probabilistic Graphical Models
PPR	Probabilistic Production Rule
PRIME	Probabilistic Reasoning Insight ModulE
PRM	Probabilistic Relational Models
RBS	Rule-Based System
SS	Shafer-Shenoy inference
SSTT	TT-based Shafer-Shenoy inference
TT	Tensor Train
WM	Working Memory

Chapter 1

Introduction

Contents

1.1 Context

Emulating the decision-making processes of experts is at the heart of artificial intelligence (AI) and the emergence of modern computers and high-level programming languages at the end of the first half of the 20th century, made this objective possible. Since 1965 and the introduction of Dendral [Buchanan et al. 1968], a system helping organic chemists to identify unknown organic molecules, by a pluridisciplinary research team of Stanford University, many systems proposed to simulate, support or even improve upon the decision-making process of experts. Since models are increasingly precise and complex, this field of study is still relevant today.

1.1.1 Artificial Intelligence and Expert System

A rule-based expert system (RBS) is an AI tool designed to simulate the reasoning of a specialist in a precise and well-delimited field, by exploiting a certain amount of knowledge explicitly provided by those specialists in the form of *facts* and *rules* into a *knowledge base* (KB). An inference engine allows us to obtain deductions, conclusions but also to produce explanations on how the results are obtained by repeating the following pattern :

1. `Detection of relevant rules according to facts`
2. `Choosing of the rule to be instantiated`
3. `triggering the rule thus possibly modifying facts`

Various algorithms can be used according to the defined rules. One can search, given facts, for consequences, such approach is known as *forward chaining* or *data-driven* as we reach to the goal using available data.

Conversely, with *backward chaining*, one can start from conclusions to determine its causes. This approach is also called *goal-driven*, as a list of goals decides which rules are selected and used.

Some component usually allows RBS to communicate with end-users to help them develop or maintain knowledge base or to enter additional data. An interface may display the conclusions made by the system, as shown in Figure 1.1.

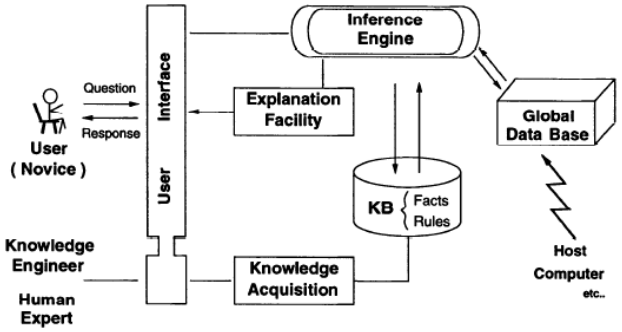


Figure 1.1: Schematic representation of the functioning of a RBS

1.1.2 Applications of RBS

Without making an exhaustive list (as can be found in [Kokkinaki et al. 1993]), expert systems have been used in very different fields [Tzafestas 1993]. From medical diagnostics (Mycin [Buchanan and Shortliffe 1984]) where users described their symptoms to a computer to obtain a diagnosis, incident monitoring in nuclear reactors (REACTOR [Nelson 1982]) to exploring the use of advanced automation in the mission operations arena during Shuttle mission control (INCO [Rasmussen et al. 1990]).

One of the strengths of an expert system comes from the explicability of its results and the traceability of its reasoning, which is decisive in this type of processes. More generally, the explanability and interpretability of a result are determining notions in AI, if impressive progress has been made using so-called *black box* models (such as those used in deep learning), especially in image processing and classification, their practical and theoretical flaws [Biggio and Roli 2018] should be an obstacle to an industrial and generalized use in an increasingly automated world where transparency and accountability in decision making processes are looked upon factors, as stated by multiple reports and guidelines of European commissions [Scientific Foresight Unit (STOA) 2019a; Madiega 2019; Scientific Foresight Unit (STOA) 2019b].

1.1.3 Limitations

While such tools are still used in some advanced industries, such as Mistral [Masera et al. 2015][1], an expert system monitoring the activity of hydroelectric dams in Italy or Brazil, they are rigid and difficult to maintain. This lack of flexibility, of a tool by and for expert domains, was a particularly strong constraint for a business use of those systems.

1.1.4 The BRMS (r)evolution

Business Rules Management Systems (BRMS), such as *IBM Operational Decision Manager* (ODM), are developed since the 90's to facilitate authoring, testing, deploying and executing business policies by domain users, in the form of conditions/actions rules. Syntactically close to the business language, these ease the translation of decision-making and business strategies, making them accessible to users with no programming experience. Along with a more understandable syntax, these frameworks also provide a set of tools that help define and monitor a system, making them suitable for business uses where many institution policies/strategies can be described as set of rules (we can think of the rules guiding whether or not a loan can be granted or the different procedures that monitor the conformity of a good on a production line).

1.2 Motivations

When developing intelligent systems, it may be inevitable to deal with uncertainty. Having complete or certain information about a domain is, usually, unrealistic. Reasoning with uncertainty is, in fact, at the core of many active area of research in AI. This issue can have multiple origins such as measurement errors, noisy automatic process or even the modelling process itself. Handling such uncertainty in a BRMS could allow business user to represent and reason with complex and real-world data.

[1] `https://www.cesi.it/technical-papers/mistral-software-for-online-monitoring-system/`

Numerous methods have been used in the rule-based system community to deal with uncertainty, using *certainty factors* [Buchanan and Shortliffe 1984], *likelihood ratio* [Hart et al. 1978] or even *fuzzy logic* [Zadeh 1965]. However, there was some limitations using such approaches, mainly due to interpretation being incoherent with probability theory [Heckerman and Shortliffe 1992] or inconsistency in the conclusions when performing chains of inference [Ng and Abramson 1990]. Another solution could be to use models that combine first-order logic and probabilistic reasoning, such as Markov logic networks [Richardson and Domingos 2006]. Bayesian techniques, mostly based of Bayesian networks [Pearl 1988; Weber et al. 2012; Arru 2011], have been used to model domains with uncertainty but are not suited for complex systems involving high design and maintenance costs [Koller and Pfeffer 1997]. To address this issue we will consider the use of Probabilistic Relational Models, PRM [Koller and Pfeffer 1998; Pfeffer 2000], an object-oriented extension of BN, with business rules.

1.3 Running Example

To illustrate the proposals made in thesis we will look at the (fictional) case of a state willing to monitor and manage its hospitals resources during a pandemic due to a SARS-CoV. Using both rules and probabilistic reasoning could help deciding whether or not a patient should be taken care of and, if so, which institution it should be sent to according to multiple criteria to avoid, as much as possible, for institutions to find themselves in situations of critical sanitary tension (e.g. no emergency beds available or overworked caregivers). The Figure 1.2 describes such a system.

A state is defined by a set of hospitals that contain a triage (reception area that manages the flow and allocation of patients) and different units (oncology, ICU, ...). Patients, assigned to physicians, may be in either of these locations. The use of rules will regulate this influx according to the intake capacities of the institutions while the underlying probabilistic graphical models will allow characterizing the state of a particular patient. Given a number of facts about the patient's condition, symptoms or environment

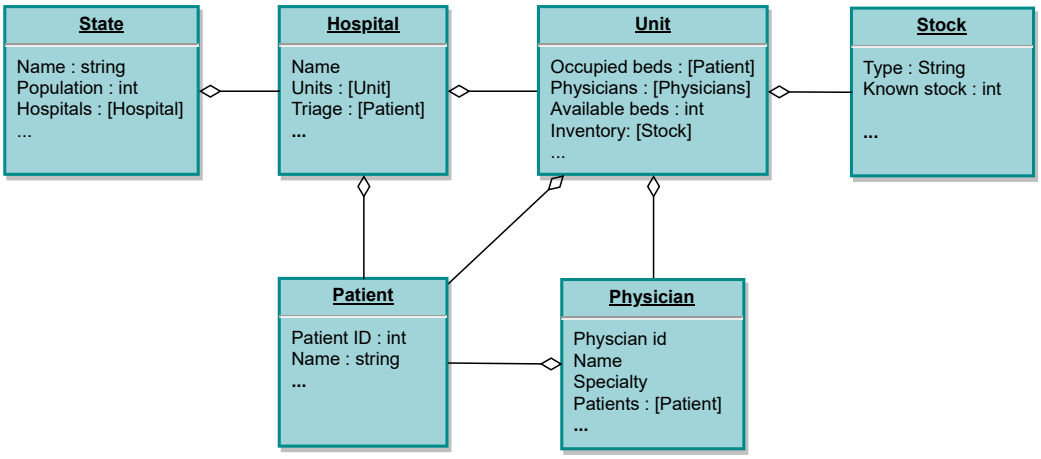


Figure 1.2: Schematic representation of our running example

they will allow to assess the potential severity of the consequences of his or her infection caused by the virus. Inventories are present in each unit in order to monitor their stocks of essential products and medicines.

1.4 Contribution and Outline

This manuscript is composed of three distinct parts. The first is devoted to the state of the art. Chapter 2 deals with BRMS, their tools and the description of the process of compiling business rules. Chapter 3 presents the theory behind probabilistic graphical models (PGMs) and their advantages. We conclude, in Chapter 4, with a review of the proposals made on the joint use of BRMS and PGMs.

The second part is dedicated to the presentation of our practical and theoretical proposals. In the chapter 5 we introduce a new syntax for probabilistic business rules. Easier to use, it required an extension of the compilation process. This work was presented in [Ducamp et al. 2020a]. The use of PGMs in a BRMS being limited by the complexity of the models, we propose, in the section 5.4 a first approach to simplify it and make the inference scalable on a large scale as presented in [Ducamp et al. 2020c]. Finally, in the chapter 6 we introduce a new method to represent the data necessary for probabilistic calculations, allowing to limit the spatial complexity during an inference

based on the use of a particular form of low rank tensors, the Tensor Train format [Ducamp et al. 2020b].

The manuscript concludes with a critical analysis of our results and perspectives for future developments. In order to facilitate the reading of the document and given the diversity of the topics covered, we have chosen to include a bibliography at the end of each chapter.

Bibliography

Arru, M. (2011). Introduction of probabilistic reasoning in business rules managmenet systems. Master's thesis, Polytech Nantes.

Biggio, B. and Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331.

Buchanan, B., Feigenbaum, E., and Lederberg, J. (1968). Heuristic dendral: A program for generating explanatory hypotheses in organic chemistry. *Machine Intelligence*, 4.

Buchanan, B. and Shortliffe, E. (1984). *Rule-based Expert System – The MYCIN Experiments of the Stanford Heuristic Programming Project*.

Ducamp, G., Bonnard, P., De Sainte Marie, C., and Wuillemin, P.-H. (2020a). Advanced Syntax and Compilation for Probabilistic Production Rules with PRM. In *RuleML+RR Doctoral Consortium 2020 (4th International Joint Conference on Rules and Reasoning)*, volume 2644 of *CEUR Workshop Proceedings*, pages 103–110, Oslo, Norway. CEUR-WS.org. Virtual conference.

Ducamp, G., Bonnard, P., Nouy, A., and Wuillemin, P.-H. (2020b). An efficient low-rank tensors representation for algorithms in complex probabilistic graphical models. In Jaeger, M. and Nielsen, T. D., editors, *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, pages 173–184. PMLR.

Ducamp, G., Bonnard, P., and Wuillemin, P.-H. (2020c). Uncertain reasoning in rule-based systems using prm. In *Florida Artificial Intelligence Research Society Conference*.

Hart, P. E., Duda, R. O., and Einaudi, M. T. (1978). Prospector-a computer-based consultation system for mineral exploration. *Journal of the International Association for Mathematical Geology*.

Heckerman, D. and Shortliffe, E. (1992). From certainty factors to belief networks. *Artificial Intelligence In Medicine*.

Kokkinaki, A. I., Valavanis, K. P., and Tzafestas, S. G. (1993). *A Survey of Expert System Tools and Engineering-Based Expert Systems*, pages 366–378. Springer Berlin Heidelberg, Berlin, Heidelberg.

Koller, D. and Pfeffer, A. (1997). Object-oriented bayesian networks. *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 302–313.

Koller, D. and Pfeffer, A. (1998). Probabilistic frame-based systems. In *AAAI/IAAI*, pages 580–587.

Madiega, T. (2019). Eu guidelines on ethics in artificial intelligence: Context and implementation. Technical Report PE 640.163, European Parliamentary Research Service (ERPS).

Masera, A., Zatonni, A., Mariano Olivera, M., Cortezzi, F., Saccarello dos Santos, M., and Patrone, J. (2015). Integrated approach to dam safety. Technical report, CESI-ISMES, CESI do Brasil Consultoria Ltda, Romagna Acque, Itapu Binacional, UTE.

Nelson, W. (1982). Reactor: An expert system for diagnosis and treatment of nuclear reactor accidents. pages 296–301.

Ng, K. C. and Abramson, B. (1990). Uncertainty management in expert systems. *IEEE Expert-Intelligent Systems and their Applications*.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference (Morgan kaufmann series in representation and reasoning)*. Morgan Kaufmann Publishers, San Mateo, Calif.

Pfeffer, A. (2000). *Probabilistic reasoning for complex systems*. PhD thesis, Stanford University.

Rasmussen, A., Muratore, J., and Heindel, T. (1990). The inco expert system project: Clips in shuttle mission control.

Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine learning*, 62(1-2):107–136.

Scientific Foresight Unit (STOA) (2019a). A governance framework for algorithmic accountability and transparency. Technical Report PE 642.262, European Parliamentary Research Service (ERPS).

Scientific Foresight Unit (STOA) (2019b). Understanding algorithmic decision-making: Opportunities and challenges. Technical Report PE 642.261, European Parliamentary Research Service (ERPS).

Tzafestas, S. G. (1993). *An Overview of Expert Systems*. Springer Berlin Heidelberg.

Weber, P., Medina-Oliva, G., Simon, C., and Iung, B. (2012). Overview on bayesian networks applications for dependability, risk analysis and maintenance areas. *Engineering Applications of Artificial Intelligence*.

Zadeh, L. (1965). Fuzzy sets. *Information and Control*, 8(3):338–353.

PART A

State Of The Art

Chapter 2

BRMS and compilation theory

Contents

2.1 Business Rule Management Systems

As we have seen, the lack of flexibility would make it tedious, if not impossible, for a company in the 1980s to use expert systems. To address this issue the principles of expert systems have been implemented in more flexible and comprehensive environments, *business rule management system* (BRMS). It is in a competitive context, where a company's economic health or even survival may depend on its ability to adapt to changing markets, that business rules show their advantages [Tony 2002; Ross 2003b]. The flexibility and agility of a BRMS allow to centralize, manage and automate business decision within a dedicated system, helping companies to reduce costs and improve both their performance and productivity. Consequently the global Business Rules Management System market size is expected to grow from USD 0.84 Billion in 2018 to USD 1.4 Billion USD by 2023[1]. But what are business rules ? What mechanisms and practical tools allow users to automate the application of their business strategies ? Let us recall some definitions, as given by [Agli 2017; Hay and Healy 2000].

Definition 2.1

Business policy *A business policy is a statement of directions or guidelines that governs the decisions of an organization and controls its actions scope.*

In our hospital context, the drug inventory management processes of pharmacies can be guided by a business policy. Instead of expecting for someone to identify and order missing products, with the endogenous risks (e.g., a shortage of critical materials) that it includes due to an error or negligence in the performance of the task (misread values or lack of tracking of expiry dates), such procedure could be automated with the help of rules.

[1] `https://www.marketsandmarkets.com/Market-Reports/business-rules-management-system-market-210469074.html`

Definition 2.2

Business rule *A business rule (BR) is a compact, atomic, well-formed, declarative statement that defines or constrains an aspect of the business and its collaborators. It must be expressed against a domain ontology (business policies vocabulary) in a natural-language that is understandable by whom it may concern, such as business and IT professional and customer.*

The business rules are the constituent units of policies, they capture the different constraints applied to procedures that are dictated by such policies. When translated in a set of action-conditions the latter can be automatically orchestrated with the help of an inference engine, as we will show in Section 2.3.

Example 2.3

Check stocks
If *the stock of Medecine A is below 600*
Then *order 2000 units*

Example 2.4

Check expiry date
If *some units of Medecine A have an outdated expiry date*
Then *destroy these units*

For detailed views about business rules and BRMS, one can refer to [Ross 2003b;a; Graham 2007; Tony 2002] and especially the introduction made by [Agli 2017].

2.1.1 Common Tools

To facilitate their adoption by non-IT users, BRMSs offer users different interoperable modules to monitor a system, from the design of the first rules and the deployment of the complete system, throughout its lifetime during which it can be modified, fine-tuned, to be adjusted to changing business policies.

2.1.1.1 Comprehensive Environment

The main component of a BRMS with which its users interact is a development environment, to edit rules and the underlying models, usually with a comprehensive graphical interface. Nowadays BRMS provide such environments directly through browsers. Lower level tools, such as plugins for Eclipse (cf Figure 2.1), may also be available.

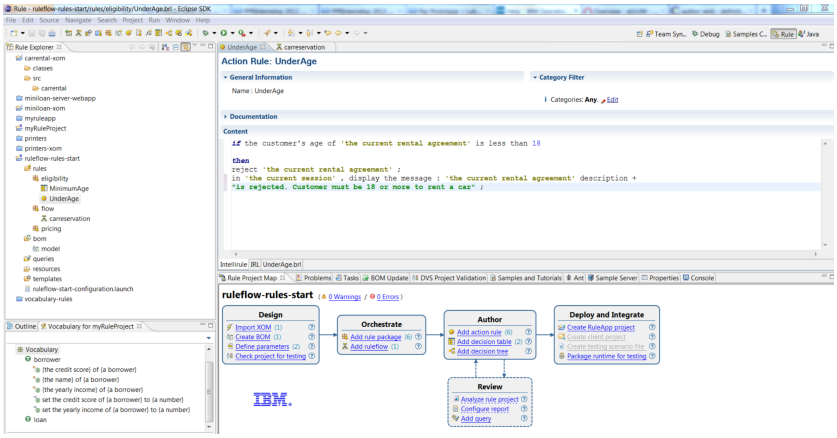


Figure 2.1: View of the Eclipse Rule Designer plugin

These environments can take place in more global management contexts, where simulation and testing tools allow the models to be evaluated and fine tuned, as shown in Figure 2.2.

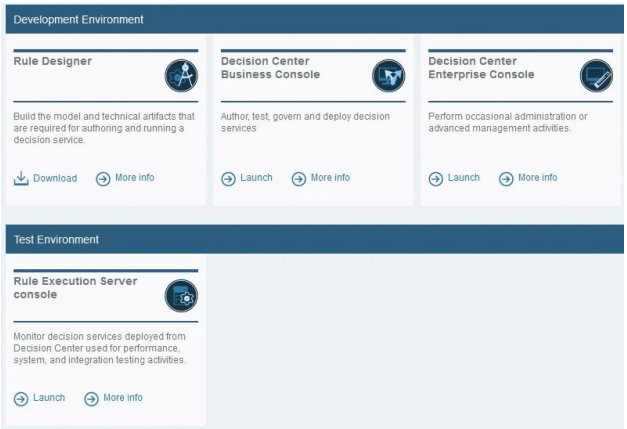


Figure 2.2: ODM on Cloud programming interface

2.1.1.2 Flexible Rule Definition and Monitoring

Business rules can be authored in multiple formats. If we are mainly interested in their textual forms, as we will see in 2.2, they can also be defined using decision trees or tables, as shown in the Figure 2.3. A decision table could, for example, contain minimum stock thresholds for each drug which, in addition to being easily readable, makes it convenient to add new drugs or manage changes in storage policies. The rules are centralized in a searchable collaborative directory called *rule base*.

	Repayment		Corporate Score		Risk of Payment Default low than	Grade	
	min	max	min	max		grade	message
1	0	10,000	≥ 900		0.5	A	Very low risk loan
2	0	10,000	600	900	0.4	A	Very low risk loan
3	0	10,000	300	600	0.3	B	Low risk loan
4	10,000	30,000	≥ 900		0.45	A	Very low risk loan
5	10,000	30,000	600	900	0.4	B	Low risk loan
				600	0.35	C	Average risk loan
			900		0.3	B	Low risk loan
				900	0.25	C	Average risk loan
				600	0.2	D	Risky loan
			900		0.25	C	Average risk loan
				900	0.23	D	Risky loan
12	≥ 60,000		300	600	0.22	E	Very risky loan

Figure 2.3: Example of decision table and rule preview

Most BRMS propose to specify, using a structure called ruleflow, how the rules are related to each other. This form allows us to orchestrate a complex problem in the form of tasks and sub-tasks and therefore ensure that a sub-rule is activated. In our previous example, we could want the rule ***Check expiry date*** (Example 2.4) to automatically trigger the rule ***Check stocks*** (Example 2.3). A graphical editor usually allows the creation of these flows in an intuitive way using drag and drop functions. Other more advanced functionalities, such as event processing, allow to define more precisely the interactions between rules, for example by defining a chronological order between business processes.

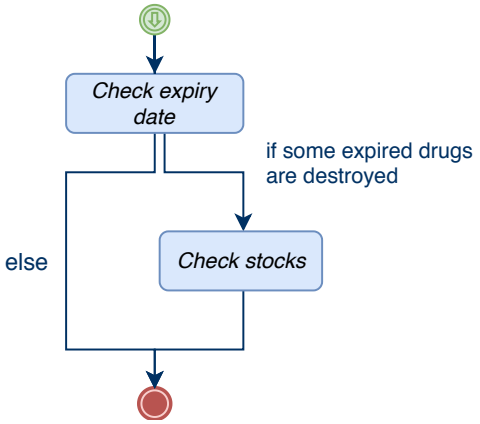


Figure 2.4: Ruleflow example

Another essential concept to rules execution is the *Working Memory* (WM) which contains the facts of the system. It can be considered as the knowledge of the system state, as a duality with the rule base. As we will see in 2.3 such facts are used to select the rules that will be applied. In our running example, all the doctors, hospitals, patients as well as their features will be in our WM. Tools allow to visualize and inspect the elements contained within the WM.

2.1.2 Operational Benefits

The diversity and complementarity of tools present in a BRMS allow its users to have a high degree of agility resulting in operational benefits that can be characterized in a few points :

- **Autonomy from IT :** BRMS help business users with a code-free approach by separating rules from their technical implementation. The various resources for defining and monitoring the ruleset help to reduce or remove reliance on IT departments for changes in live systems.

- **Control over decision :** Increased control over implemented decision logic for compliance and better business management with the ability to express decision logic with increased precision, using a business vocabulary syntax and graphical rule representations (decision tables,

trees, scorecards and flows). Being able to easily update rules criteria is important in a context where markets evolve rapidly, new regulations are put in place or to align with the competition and the centralization of decision-making processes ensures coordination between the different agents working on the system.

- **Endogenous risk minimisation :** By automating such processes in a precise, unambiguous and controlled way, endogenous issues due to errors in judgement or misalignment between the business requirements specification and their IT implementations making can be avoided (in particular because tools allow a precise monitoring of the modifications —of the evolution— of the models as well as a complex coordination between the agents working on it).

2.2 ODM Rule Syntax

Each BRMS offers its own programming and execution environment and every product has its own language to write rules technically, but most of them provide a mechanism to write in a natural language-like format. ODM offers business users to define their rules and models with two languages, namely BAL and IRL.

2.2.1 Business Action Language

BAL, for *Business Action Language*, is close to natural language and is essentially designed to help business professionals to enter BRs in a human readable format. The rules that are defined using BAL are called *action rules* and they are based on the well known IF/THEN-ELSE constructs.

```
if
    the stock of 'medecine A' in ICU is less than 600
then
    place an order of 2000 units.
```

Figure 2.5: Example of rule in BAL

2.2.2 ILOG Rule Language

In this thesis we will consider rules under their technical formalism, the ILOG Rule Language (IRL). Rules defined with this Java-like language are also made of a condition part and an action part, as we can see in the Figure 2.6. The condition part, which begins with the keyword *when*, binds variables to objects and attribute values, and specifies tests on attribute values. The action part, which begins with the keyword *then*, specifies the actions to be carried out if the rule is executed. An optional second part which begins with the keyword *else*, that applies only if the last evaluated statement in the condition part is false.

```
rule stock.minimum medecine stock {
    when {
        hp : hospital();
        u : Unit(u.name=="ICU") in hp.units;
        s : Stock(s.type=="medicine A") in u.inventory;
        evaluate ( s.known stock < 600);
    } then {
        p.addToMessages("Stock of medicine A below 600");
        p.placeOrder(MedicineA,2000);
};
```

Figure 2.6: Example of the same rule in IRL

The main elements and keywords constituting the condition part of a rule written in the IRL format are described below. An exhaustive presentation of the grammar is available in ODM's documentation [2].

While the syntax of the BAL is more suitable for the business user community, it is less refined than that of the IRL, its modelling capabilities are more limited. As we will see in Section 2.4.2 the rules in BAL format are actually rewritten in IRL format before being compiled.

[2] https://www.ibm.com/support/knowledgecenter/SS7J8H/com.ibm.odm.dserver.rules.ref.designer/lang_irl_ref_topics/tpc_irl_grammar_intro.html

2.2.2.1 Class Condition

Class conditions are the simplest form of conditions. They bind a variable (in the the following condition, *phy*) to an object from the working memory, which consists of a type name (*Physician*) and, potentially, inside of parentheses, some constraints on the type attributes (*specialty ==”anaesthetist”*).

```
phy: Physician(specialty =="anaesthetist")
```

2.2.2.2 Existential Conditions

The keyword *exists* checks for the existence, in the WM, of an element satisfying a specific condition. The condition below is satisfied whenever an anaesthetist is found in the WM.

```
exists Physician(specialty =="anaesthetist")
```

Conversely, the *not* statement returns true if there is no WME matching the condition. For example, the following condition is satisfied when there is no physician in the WM that is anaesthetist.

```
not Physician(specialty =="anaesthetist")
```

2.2.2.3 From

The keyword *from* is used inside the rule to access ruleset parameters. These are a special type of parameters that are used to exchange data between the application and the ruleset. If we assume that *physician* is a ruleset input parameter, which is previously defined, then the following condition binds the variable phy to that parameter.

```
phy :Physician() from physician;
```

2.2.2.4 In

The keyword *in* restricts the scope of a variable to a collection of values. All the physicians in the hospital *hp* will be mapped to the variable *phys* in the following condition.

```
hp : Hospital();
phys : Physician() in hp.physicians;
```

2.2.2.5 Aggregate Condition

One of the most complex condition format available in the language is the aggregator one, which computes a value from a collection of values. Examples of aggregations are the average, sum, or maximum of a numeric collection. Here is an example of a condition that computes the number of patients under the care of a physician who are over 80 years old. When we want to condition the result of the aggregate, the key word where is used.

```
phy : Physcian();
agg:aggregate{
    p:Patient() in phy.patients;
    } do {
        sum(p.age > 80)
    } where { agg > 4}
}
```

2.2.2.6 Evaluate

The *evaluate* operator simply checks for the truth value of a statement.

```
phy : Physician()
evaluate ( phy.numberOfPatients < 5 )
```

We will see, in the chapters 4 and 5, how these different bricks of language influence the rewriting of probabilistic rules.

2.2.3 Object Model

As previously mentioned, business rules are applied to instances of objects present in the working memory. The description of such objects are defined by an object model (OM) whose syntax is very close to that of Java, as showed in Figure 2.7. OM contains classes, type and attributes definitions as well as function declaration (such as attribute getters and setters or constructors).

```
public class Physician {
    public String name;
    public String specialty;
    public int age;
    public int numberOfPatients;
    ...
}
```

Figure 2.7: Example of OM declaration

2.3 Inference Engine

As with expert systems, the orchestration of rule execution is done through an inference engine. When based on the Rete algorithm [Forgy 1982] or RetePlus, the ODM rule execution mode based on this algorithm [3], the engine will seek to find rules whose conditions are compatible with the objects present in the WM. A chosen rule will be applied on these objects with possible edge effects (this algorithm is designed to incremently manage every data modification that are performed in the action part), updating the memory and/or informing the user, as shown in the Figure 2.8. Specifically, it permits to take into account rules chaining during the inference process. Other execution modes can be used within ODM, depending on the business logic (and the expected performances) [4].

[3] https://www.ibm.com/support/knowledgecenter/SSQP76_8.10.x/com.ibm.odm.dserver.rules.designer.run/optimizing_topics/con_opt_execmodes_reteplus.html

[4] https://www.ibm.com/support/knowledgecenter/SS7J8H/com.ibm.odm.dserver.rules.designer.run/optimizing_topics/tpc_opt_choose_execmode_criteria.html

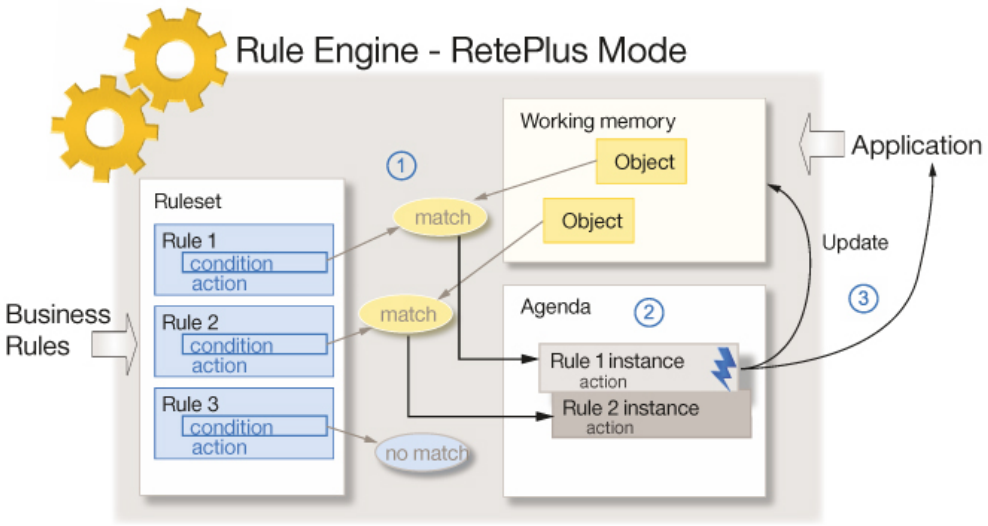


Figure 2.8: Schematic representation of the RetePlus mode

It is interesting to see the ruleset as a function whose parameters (the WM) will define a service implemented no longer using classical procedural code but with rules whose execution will be orchestrated by an inference engine.

Because the rules are written in a high level language and since we are going to propose to extend its syntax it is appropriate to present the mechanisms allowing a computer to interpret them.

The next section is devoted to a brief presentation of the different steps of a classical compilation chain.

2.4 Compilation

In order to execute some high-level code on a various range of computers, one need to make it readable by the targeted machine. This is the main role of a compiler which is, simply stated, a program that reads code written in one language, usually with a high level of abstraction, and translate into an equivalent program in another language, usually close to level-machine.

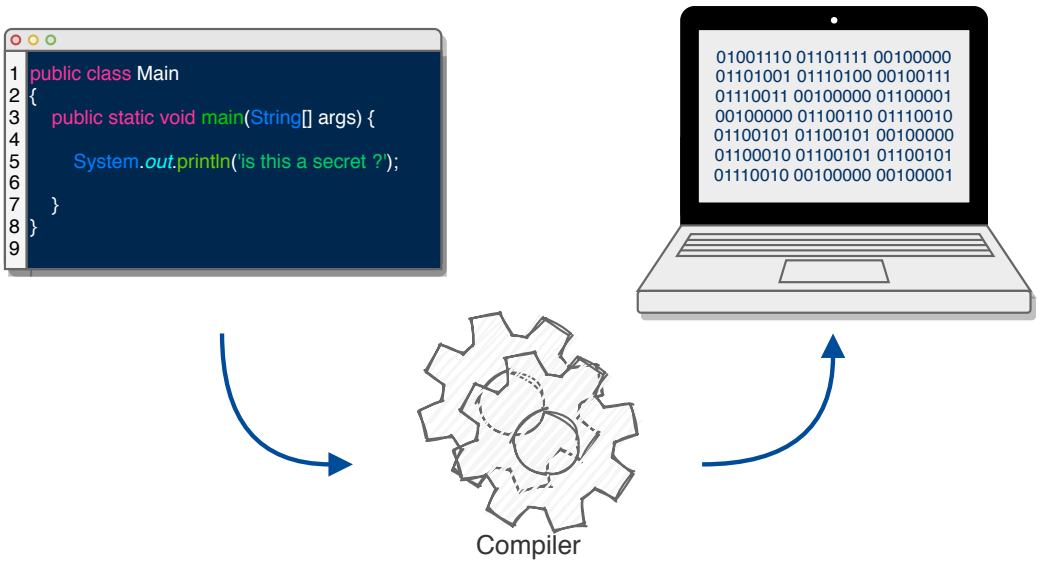


Figure 2.9: Example of the compilation of a high level language

If, from this definition, we could consider a lot of different compilers, their structure and principles stay essentially the same [Wirth 1996]. In this section we will briefly outline them. The adventurous reader might look upon [Aho et al. 1986], a reference book about compilation technology, to have a more in-depth view, both theoretical and practical, about this subject.

2.4.1 Toolchain

The compilation process, called toolchain, is divided into two phases, namely *front end* and *back end*. Each phase is composed of several steps. The main role of the front end is to translate the source code, written in a certain language, into intermediate code. The back end will transform this intermediate representation into code readable by the targeted machine, this phase will generally do not depend on the source language and can therefore be shared for different source code. If some languages, like C, require some specific phases (like pre-processing, to support macro substitution and conditional compilation) we will focus here on the phases shared by toolchains during the front end phase.

2.4.1.1 Lexical Analysis

The front end process starts with a lexical analysis. The purpose of this step is to read the stream of characters making up the source program and split it into a set of atomic units of the language called tokens. They correspond to the keywords, identifiers or symbols that are the atomic unit of the languages. This phase is also called scanning or lexing; the software, such as Lex [Lesk and Schmidt 1990], that performs a lexical analysis is called a lexical analyzer or scanner. The following example comes from [Aho et al. 1986].

Example 2.5

The statement `position := initial + rate * 60` *will be divided into :*

position	:=	*initial*	+	*rate*	*	*60*

2.4.1.2 Syntactic Analysis

During the syntactic analysis phase, tokens will be grouped so as to have a collective meaning given a grammar specifying the syntax of the language (called context-free [Chomsky 1956; Knuth 1968]), it's generally based on the construction of a tree structure, such as the one presented in Figure 2.10.

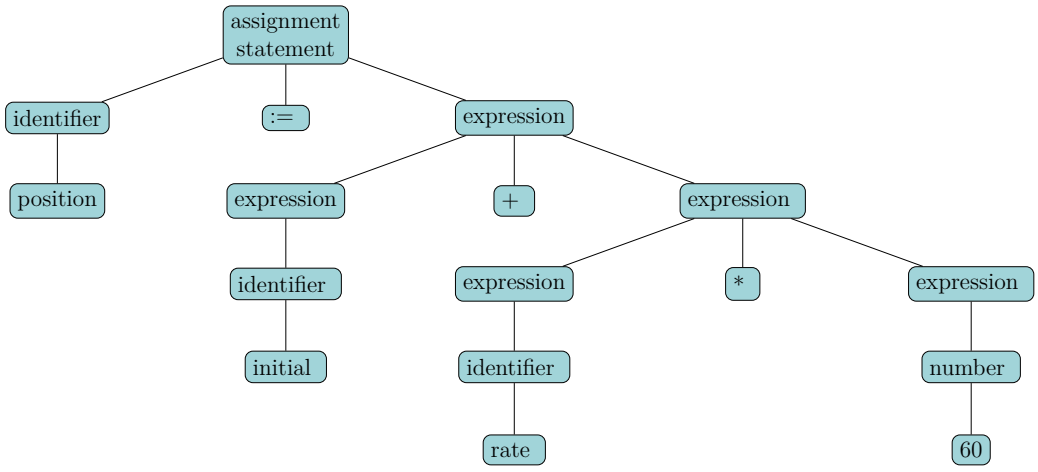


Figure 2.10: Syntactic generated tree from Example 2.5

This grammar can be described in several forms, such as the Backus-Naur Form [Backus et al. 1963; Knuth 1964]. The analysis tree is often modified and improved during the compilation process in order to optimize the process.

2.4.1.3 Semantic Analysis

Semantic analysis is the phase during which the compiler adds semantic information to the analysis tree. This phase checks the type (type error checking), or the binding object (associating variables and function references with their definitions), or a defined task (all local variables must be initialized before use), can issue warnings, or reject incorrect programs. Many errors can be the cause of an incorrect program like a misspelled keyword (lexical error), an arithmetic operation with misplaced parentheses (syntactic error), an operator applied to an incompatible operand (semantic error). In the example in Figure 2.11 and since the '*' operator is applied to real number the type checker will convert integer to real using an **inttoreal** operation.

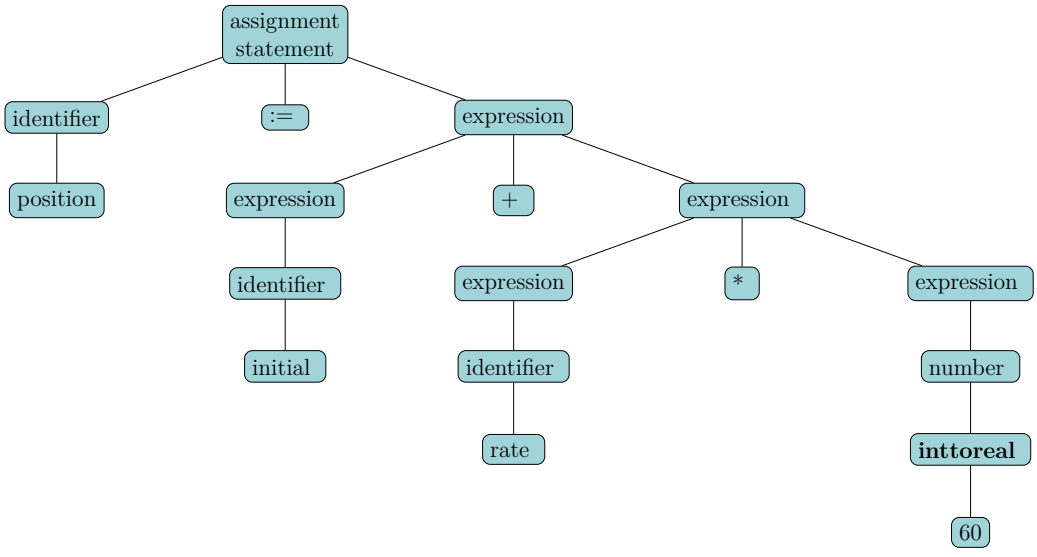


Figure 2.11: Our syntactic tree after type checking

2.4.1.4 Intermediate Code

Source codes are usually not directly compiled into a target language. Using abstract intermediate code, independent of the target language, allows the same front-end to be reused for different machines. Intermediate language-specific optimizations can also be applied, as shown in Figure 2.12, where the three-adresses intermediate code is simplified in order to eliminate unnecessary operations.

```
temp1 := inttoreal(60)
temp2 := id3 * temp1
temp3 := id2 + temp2
id1 := temp3
```

(a) Intermediate code

```
temp1 := id3 * 60.0
id1 := id2 + temp1
```

(b) Optimized code

Figure 2.12: Intermediate code generation

2.4.1.5 Code Generation

The final phase in a compiler consist of generating target-specific code, again using optimization techniques. Figure 2.13 shows how the previous intermediate code is rewritten in a Assembly-like language where values are moved from one register to another with operations applying on them.

```
MOVF id3, R2
MULF #60.0, R2
MOVF id2, R1
ADDF R2, R1
MOVF R1, id1
```

Figure 2.13: Generated low-level code

In the case of ODM, the compilation chain transforms BAL/IRL code into Java bytecode.

2.4.2 ODM Toolchain

Figure 2.14 shows a simplified view of ODM's toolchain. When business rules are entered using the BAL, they are first translated into IRL rules. The ruleset (and the OM) is then parsed and checked. Different rewriting operations are performed on their semantic representation (SemRuleset). The rewriting of the ruleset includes —among other things— a step of renormalization of the rules and another of optimization of the predictive terms. The tree is compiled while taking into account the chosen inference algorithm before being transformed into an intermediate Java-like language (IROS), where debug code is added and exception handled. The code is rewritten one last time into bytecode in an archive (JAR) using JavaCC [Kodaganallur 2004], facilitating deployment and execution on the target machines.

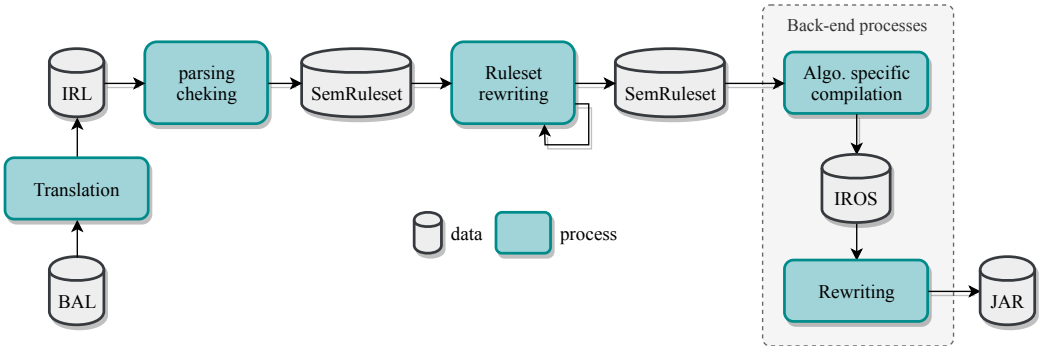


Figure 2.14: Simplified view of the ruleset toolchain

2.5 Conclusion

In this chapter we briefly present what BRMS are, how their understandable syntax and adapted operating mechanisms help business users translate business policies and automate decision making. While they address the accessibility issues faced by early expert systems in order to be used by non-computer specialists, their modelling capabilities are limited by the new syntax.

Bibliography

Agli, H. (2017). *Uncertain reasoning for business rules*. PhD thesis, Université Pierre et Marie Curie - Paris VI.

Aho, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., USA.

Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Naur, P., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijngaarden, A., and Woodger, M. (1963). Revised report on the algorithmic language ALGOL 60. *The Computer Journal*, 5(4):349–367.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17 – 37.

Graham, I. (2007). *Business Rules Management and Service Oriented Architecture: A Pattern Language*. Wiley.

Hay, D. and Healy, K, a. (2000). Defining business rules - what are they really ? Technical report, Business Rules Group.

Knuth, D. E. (1964). Backus normal form vs. backus naur form. *Commun. ACM*, 7(12):735–736.

Knuth, D. E. (1968). Semantics of context-free languages. In *In Mathematical Systems Theory*, pages 127–145.

Kodaganallur, V. (2004). Incorporating language processing into java applications: A javacc tutorial. *IEEE Software*, 21(4):70–77.

Lesk, M. and Schmidt, E. (1990). Lex—a lexical analyzer generator.

Ross, R. (2003a). Business rules manifesto. Technical report, The Business Rules Group.

Ross, R. (2003b). *Principles of the Business Rule Approach*. Addison-Wesley.

Tony, M. (2002). *Business Rules and Information Systems: Aligning IT with Business Goals*. Addison-Wesley Professionel.

Wirth, N. (1996). *Compiler construction*. International computer science series. Addison-Wesley.

Chapter 3

Probabilistic Graphical Models

Contents

3.1 Introduction

As noted in the introduction, it may be difficult to properly model a real application without taking into account the existence of random factors. Measurement errors, structural uncertainty, or even the stochastic nature of the models themselves make it necessary to take this uncertainty into account. Several publications deal with the different approaches regarding this subject, such as [Halpern 2017] or [Shafer and Pearl 1990], a collection of key papers about probabilistic expert systems, probability theory and the Bayesian approach.

Probabilistic graphical models [Koller and Friedman 2009; Pearl 1988] propose to combine both graph theory and probability theory into a framework helping to represent and reason on complex probabilistic models. As we will see, their graphical properties have two major advantages : computations in these models are simplified (not to say feasible) and such visualization makes them easier to be understood and therefore accessible. In this chapter, we present the basics of probability theory that allow us to appreciate these models. After having introduced Bayesian networks, the keystone of PGMs, and the different types of calculations they allow, we will discuss PRMs, an object-oriented extension of the latter allowing the modelling of large scale complex systems. Learning those models, which is just as crucial as inferring within them, is not addressed in this thesis, the curious reader may refer to [Koller and Friedman 2009; Neapolitan 2003] for more information on this topic.

To illustrate the next sections, we will consider the case of patients possibly suffering from SARS-CoV infections. The disease caused by such a virus, called SARS (for *Severe Acute Respiratory Syndrome*), presents multiple symptoms including muscle pain, headache, fever, dyspnoea or pneumonia, the use of PGMs could help to determine or not the severity of their condition according those symptoms and observations about the patient (age, sex, comorbidities, contact with others people, ...).

In the simpler example that we will follow in this manuscript, taken from the models described in [Fenton et al. 2020][1], the severity states of patients will be determined by their age, whether or not they have risk factors and, of course, whether or not they are infected with the virus. If this is the case, certain symptoms may be observable and the presence of the virus detected by one of the tests.

Inference in PGMs is similar to the concept of reasoning in ODM. The definition proposed in Merriam-Webster for inference is "the act of passing from one proposition, statement, or judgement considered as true to another whose truth is believed to follow from that of the former". In PGMs, inference allows us to propagate evidence, knowing probabilities on so-called random variables.

Many of the following definitions are given by [Koller and Friedman 2009], a must-read reference for PGM enthusiasts.

3.1.1 Random Variables and Joint Distribution

In this section we will introduce some of the fundamental notions of the probability theory.

3.1.1.1 Experiment Outcomes and Random Variables

We can define *events* by assuming that there is an accepted space of possible outcomes, denoted by Ω. If we consider, for example, the possible outcomes of a dice roll, we will set $\Omega = \{1, 2, 3, 4, 5, 6\}$. In the case of the evaluation of a patient's state, we will consider $\Omega = \{none, asymptomatic, mild, severe\}$.

In addition, we assume that there is a set of *measurable events* S to which we are willing to assign probabilities. Formally, each *event* $\alpha \in S$ is a subset of Ω. In the die example, the event $\{1\}$ represents the case where the die shows 1, and the event $\{2, 4, 6\}$ represents the case of an even outcome.

[1] raw data: `http://www.eecs.qmul.ac.uk/~norman/Models/covid19_for_contact_tracing_paper.cmpx`
processed data: `https://gitlab.com/agrumery/pgmrepository/-/blob/master/bif/covid19.bif`

The event space must satisfy three basic properties to be consistent w.r.t the probabilistic theory :

- It contains the *empty event* $\emptyset$, and the *trivial event* Ω.
- It is closed under union. That is, if $\alpha, \beta \in S$, then so is $\alpha \cup \beta$.
- It is closed under complementation. That is, if $\alpha \in S$, then so is $\Omega - \alpha$.

Given these properties, we can formally define probability distributions.

Definition 3.1

Probability Distribution *A probability distribution* $\mathbb{P}$ *over* (Ω, S) *is a mapping from events in* S *to real values that satisfies the following conditions:*

- $\mathbb{P}(\alpha) \geq 0$ *for all* $\alpha \in S$ *: all the probabilities are not negative.*
- $\mathbb{P}(\Omega) = 1$ *: the "trivial event", which allows all possible outcomes, has the maximal probability.*
- *If* $\alpha, \beta \in S$ *and* $\alpha \cap \beta = \emptyset$*, then* $\mathbb{P}(\alpha \cup \beta) = \mathbb{P}(\alpha) + \mathbb{P}(\beta)$ *: the probability that one of two mutually disjoint events will occur is the sum of probabilities of each event.*

Suppose that we want to reason about the state of a patient, to compute its probability of being asymptomatic. We can use an event such as *StateAsymptomatic* to denote the subset of infected patients that show no symptoms and use it in our formulation. However, this discussion becomes rather cumbersome if we also want to consider the patients with a mild or severe state. The notion of random variables allows us to directly reports the state of a patient in a clean, mathematical way. Suppose that we have a random variable *State* that reports the state of a patient, then the statement $\mathbb{P}(State = Asymptomatic)$ is another notation for $\mathbb{P}(StateAsymptomatic)$.

Formally, a random variable, such as *State*, is defined by a function that associates with each outcome in Ω a value. For example, *State* is defined by a function f_{State} that maps each person in Ω to his or her state. The event $State = mild$ is a shorthand for the event $\{\omega \in \Omega : f_{State}(\omega) = mild\}$.

We can distinguish two types of random variables based on the set of values that they can take. Discrete random variables, which may take on only a countable number of distinct values and those that can take an infinite number of them, whether enumerable or continuous. The result of a patient's test is discrete, it will be either positive or negative. His height is continuous (it can however be discretized in several ranges). From now on we will denote random variables by capitalized letters, Val(X) represents the set of possible values for a random variable X and x refers to a generic value of X. We use the notation $x_1, \ldots, x_n$, for $n = |\,\mathrm{Val(X)}|$ (the number of elements in Val(X)), when we need to enumerate the specific values of X. The Table 3.1 describes the random variables for our running example -which are all discrete- and their possible values (also called domains).

X	**Val(X)**
Cough (C)	{yes,no}
Age (A)	{<15, 15-49, 50-64, 65-80, >80}
Risk factors (F)	{low, medium, high}
Infected with SARS-CoV (I)	{yes, no}
Loss of taste or smell (L)	{yes, no}
Test type (T)	{CT-scan, PCR nasal, antibody, no test}
Tested result (R)	{negative, positive, NA}
Current SARS status (S)	{severe, mild, asymptomatic, none}

Table 3.1: Discrete random variables for within our SARS-CoV example

Once a random variable X is defined, we can consider the distribution over events that can occur using X. Denoted by $\mathbb{P}(\mathrm{X})$, this distribution is referred as the *probability distribution* of X. It is defined such that :

$$\begin{array}{rccl} \mathbb{P}(\mathrm{X}): & \mathrm{Val(X)} & \rightarrow & [0,1] \\ & x & \mapsto & \mathbb{P}(\mathrm{X}^{-1}(x)) \end{array} \tag{3.1}$$

Consequently to the properties given by the Definition 3.1 we have $\forall x \in \text{Val}(\text{X}), \mathbb{P}(\text{X} = x) \geq 0$ and $\sum_{i=1}^{n} \mathbb{P}(\text{X} = x_i) = 1$.

The Figure 3.1 shows the probability distribution of our random variable S, which characterize the severity of a patient's condition. We will see, in section 3.3.3 how to compute such values.

Curent SARS status			
severe	**mild**	**asymptomatic**	**none**
0,0008	0,0053	0,0438	0,9500

Figure 3.1: Probability of having a certain severity of disease, $\mathbb{P}(\text{S})$

3.1.1.2 Joint Distribution

We are usually interested in questions that involve the value of several random variables. For example, we might consider the event "A =< 15, I = yes and S = serious", (e.g. being young, infected with the virus and having severe form of disease). To discuss such event, we need to introduce the *joint distribution* over these two random variables. More generally, the joint distribution over a set $\mathbf{X} = \text{X}_1, \ldots, \text{X}_n$ of random variables is denoted $\mathbb{P}(\text{X}_1, \ldots, \text{X}_n)$ and is the distribution that assigns probabilities to events that are specified in terms of these random variables. Formally, it can be defined as :

Definition 3.2

Joint Distribution *Let $\mathbb{P}(\text{X}_1, \ldots, \text{X}_n)$ denote the joint distribution over a set of random variable $\{\text{X}_1, \ldots, \text{X}_n\}$, it verifies that :*

$$\begin{aligned} \mathbb{P}(\text{X}_1, ..., \text{X}_n) : \quad & \text{Val}(\text{X}_1) \times ... \times \text{Val}(\text{X}_n) \to [0, 1] \\ & (x_1, ..., x_n) \mapsto \mathbb{P}(\textstyle\bigcap_{i=1}^{n} \text{X}_i^{-1}(x_i)) \end{aligned} \tag{3.2}$$

The Figure 3.2 states, for example, that the probability of having a cough and an asymptomatic form of the virus is equal to 0.04%.

	Curent SARS status			
Cough	**severe**	**mild**	**asymptomatic**	**none**
no	0,0002	0,0018	0,0434	0,7600
yes	0,0006	0,0035	0,0004	0,1900

Figure 3.2: Probability of having a certain status and a cough: $\mathbb{P}(\mathrm{S}, \mathrm{C})$

As we will see in more detail in the section 3.20, we can always find the marginal of a variable within a joint distribution by summing over the other ones.

3.1.2 Conditional probability

A key concept in probability theory is that of conditional probabilities. Such notion allows us to take into account knowledge about our model, for example to compute the probability for a patient to have a particular form of disease knowing that he lost his smell and have a cough, as in the Figure 3.3.

Curent SARS status			
severe	**mild**	**asymptomatic**	**none**
0,0215	0,1122	0,0002	0,8861

Figure 3.3: $\mathbb{P}(\mathrm{S}|\mathrm{C} = yes, \mathrm{L} = yes)$

Definition 3.3

Conditional probability *The conditional probability of a a random variable* X *given knowledge about the value of another* Y*, denoted* $\mathbb{P}(\mathrm{X}|\mathrm{Y})$ *is defined such as :*

$$\mathbb{P}(\mathrm{X}|\mathrm{Y}) = \frac{\mathbb{P}(\mathrm{X}, \mathrm{Y})}{\mathbb{P}(\mathrm{Y})},$$

with $\mathbb{P}(\mathrm{Y}) \neq 0$.

Interestingly, with this formulation, we see a logic similar to that of inferences made with an expert system. Trying to characterize a diagnostic variable according to a set of knowledge about a patient's symptoms is similar to what is done with the forward chaining of a rule engine.

By writing such a formula as $\mathbb{P}(\mathrm{X},\mathrm{Y}) = \mathbb{P}(\mathrm{Y})\,\mathbb{P}(\mathrm{X}|\mathrm{Y})$, we can, under no further assumption, extend it to multiple variables using the so-called *chain rule*.

Definition 3.4

Chain Rule *Let P represents a joint distribution over a set of variable $\{\mathrm{X}_1,\ldots,\mathrm{X}_k\}$, P can be expressed as :*

$$\mathbb{P}(\mathrm{X}_1,\ldots,\mathrm{X}_k) = \mathbb{P}(\mathrm{X}_1)\,\mathbb{P}(\mathrm{X}_2|\mathrm{X}_1)\ldots\mathbb{P}(\mathrm{X}_k|\mathrm{X}_1,\ldots,\mathrm{X}_{k-1}) \tag{3.3}$$

Such a formulation holds for any joint distribution but since the conditional probabilities in the factorization on the right-hand side are neither natural nor compact, it is, at that point, not very helpful. Moreover, the factorization is not unique, it depends on the chosen order of the variables. However, we will show in section 3.2.3 how using conditional independencies and some particular order, called topological order, can help such distribution to be overly simplified.

3.1.3 Conditional Independencies

When throwing two dice, the value of the first throw doesn't inform us about the value of the second one, they are independent events. However, if a random variable is, for example, associated with the sum of those dice, is known, this assertion doesn't hold, the value of the dice are not conditionally independent knowing their sum.

Definition 3.5

Probabilistic Conditional Independence *Let* $\mathbf{X}, \mathbf{Y}, \mathbf{Z} \subset V$*, we say that* $\mathbf{X}$ *is conditionally independent of* $\mathbf{Y}$ *given* $\mathbf{Z}$*, w.r.t P, and we write* $(\mathbf{X} \perp_{\mathbb{P}} \mathbf{Y}|\mathbf{Z})$*, if and only if* $\mathbb{P}(\mathbf{X},\mathbf{Y}|\mathbf{Z}) = \mathbb{P}(\mathbf{X}|\mathbf{Z})$*. The variables in* $\mathbf{Z}$ *are often said to be observed. When* $\mathbf{Z} = \emptyset$*, we say that* $\mathbf{X}$ *and* $\mathbf{Y}$ *are marginally independent and we write* $(\mathbf{X} \perp_{\mathbb{P}} \mathbf{Y})$*.*

The intuition behind this definition is that two sets $\mathbf{X}$ and $\mathbf{Y}$ are conditionally independent given a set $\mathbf{Z}$ when, given our knowledge about this latter, our belief on $\mathbf{X}$ is not influenced by any knowledge we learn about $\mathbf{Y}$.

3.1.4 Evaluation of probabilities in complex models

Computing these different metrics directly in the entire joint probability distribution of a model would be disaster in terms of performance, if not impossible. We would need, for example, to store 5760 values to encode the joint law of our running example. In the more complex model showcased in Appendix 2, we would need more than 10^{15} (one quadrillion !). Even if we could store such data, one would need to marginalize over each values in order to compute a variable's posterior. Circumventing the problem of complexity in terms of memory will not prevent the issue of time-related cost. Furthermore, from a statistical point of view, if we want to find out the distribution from data, it would take ridiculously large amounts of data to estimate reliably these many parameters. Fortunately, properties such as conditional independencies help to mitigate such complexity.

3.2 Bayesian Networks

The examples above convince us that unless we are in an extremely simple case, the exhaustive representation of the joint distribution is unmanageable from every perspective. Storing - and manipulating - such models in memory would be too expensive, if not impossible. Perhaps even worse, it is impossible to obtain so many numbers from a human expert; additionally, the numbers are extremely small and do not correspond to events that people can reasonably consider.

By representing the independence properties of the distributions and by using an alternative parametrization exploiting these independences, Bayesian networks, whose essential principles are presented here, make it possible to tackle the aforementioned challenges.

3.2.1 Model

A Bayesian network provides a compact representation of the joint probability distribution of a set of random variables. These appear in the form of nodes in a directed acyclic graph (DAG), denoted $\vec{\mathcal{G}}$, where the absence of arcs represents some conditional independencies. Formally, we have :

Definition 3.6

Bayesian network *A Bayesian network is a pair* $\mathcal{B} = (\vec{\mathcal{G}}, \mathbb{P})$ *where* $\vec{\mathcal{G}} = (\mathcal{V}, \mathcal{A})$ *is a DAG and* $\mathbb{P}$ *characterize the factorized joint probability distribution that satisfies the chain rule for BN.*

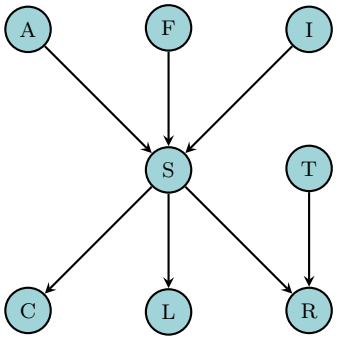
Definition 3.7

Chain Rule for BN *Let* $\mathcal{B}$ *be a BN over the variables* $\mathrm{X}_1, \ldots, \mathrm{X}_n$*. We say that a distribution* $\mathbb{P}$ *over the same space factorizes according to* $\vec{\mathcal{G}}$ *if* $\mathbb{P}$ *can be expressed as the following :*

$$\mathbb{P}(\mathrm{X}_1, \ldots, \mathrm{X}_n) = \prod_{i=1}^{n} \mathbb{P}(\mathrm{X}_i | \mathrm{Pa}_{\mathrm{X}_i}^{\vec{\mathcal{G}}}) \tag{3.4}$$

Practically, a Bayesian network $\mathcal{B}$ is associated with $(\vec{\mathcal{G}}, \mathbb{P})$ where $\mathbb{P}$ is given as the set of local conditional probabilities distributions of the variables given their parents in $\vec{\mathcal{G}}$: $\Theta = \{\mathbb{P}(\mathrm{X}_i | \mathrm{Pa}_{\mathrm{X}_i}^{\vec{\mathcal{G}}})\}$. When the variables are discrete, each node is associated with a CPT, such as the one in Figure 3.4b, that contains the conditional probabilities of the random variable with respect to its parents.

Figure 3.4a illustrates the underlying Bayesian network used in our running example. In this BN, the severity of a patient's status (S) influences its symptoms (C,L) as well as the result of his test (R). It is conditioned by its age (A), some risk factors (F) and the fact that he's been infected (I). We will denote such graph by $\vec{\mathcal{G}}_{\mathrm{SARS}}$.



(a)

	Cough	
Current SARS-CoV Status	no	yes
severe	0,2900	0,7100
mild	0,3430	0,6570
asymptomatic	0,9900	0,0100
none	0,8000	0,2000

(b)

Figure 3.4: (a) SARS-CoV related BN. (b) C's CPT

The resulting factorized representation substantially reduces the spatial complexity of the model, particularly for sparse structures. In a distribution over n binary random variables, the specification of the joint distribution requires $2^n - 1$ independents parameters. If the distribution factorizes according to a graph $\vec{\mathcal{G}}$ where each node has at most k parents, the total number of independents parameters will be less than $n \cdot 2^k$.

Example 3.8

Using the chain rule for BN w.r.t $\vec{\mathcal{G}}_{SARS}$ *allows us the redefine the joint distribution of our running example as follows :*

$$\begin{aligned} \mathbb{P}(\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{S},\mathrm{T},\mathrm{C},\mathrm{L},\mathrm{R}) &= \mathbb{P}(\mathrm{A})\,\mathbb{P}(\mathrm{F})\,\mathbb{P}(\mathrm{I}) \\ &\quad \mathbb{P}(\mathrm{S}|\mathrm{A},\mathrm{F},\mathrm{I})\,\mathbb{P}(\mathrm{T}) \\ &\quad \mathbb{P}(\mathrm{C}|\mathrm{S})\,\mathbb{P}(\mathrm{L}|\mathrm{S})\,\mathbb{P}(\mathrm{R}|\mathrm{S}) \end{aligned} \tag{3.5}$$

To represent such factorized distribution we would need 198 *parameters instead of* 5670*. The saving in terms of spatial complexity is even more convincing in the case of the largest version of the network, presented in Appendix 2. When more than* 10^{15} *values are necessary to directly describe the joint distribution, only* 777 *are sufficient using such property.*

In the following section we will introduce some graphical vocabulary regarding Bayesian network in order to express the semantics of such representation and explain why it is far from being insignificant in terms of modelling capabilities.

3.2.2 Graphical vocabulary

For any graph $\vec{\mathcal{G}}$, let $\mathcal{V}(\vec{\mathcal{G}})$ express the set of its nodes and $\mathcal{A}(\vec{\mathcal{G}})$ the set of its arcs (in order to lighten the notation and if there is no ambiguity about the DAG at hand, we will omit to specify $\vec{\mathcal{G}}$). $\mathrm{X} \in \mathcal{V}$ is said to be a parent of $\mathrm{Y} \in \mathcal{V}$ if the arc (X,Y) is in $\mathcal{A}$.

Definition 3.9

Trail *We will call trail in* $\vec{\mathcal{G}}$ *the set* $\{\mathrm{X}_1,...,\mathrm{X}_{n+1}\} \subset \mathcal{V}$ *if either* $(\mathrm{X}_i,\mathrm{X}_{i+1})$ *or* $(\mathrm{X}_{i+1},\mathrm{X}_i) \in \mathcal{A}, \forall i \in \{1,...,n\}$.

Definition 3.10

Direct path *When a trail from* X_1 *to* X_n *is a set such that* $(\mathrm{X}_i,\mathrm{X}_{i+1}) \in \mathcal{A}, \forall i \in \{1,...,n\}$ *(i.e. , all the arcs are oriented towards* X_n*) it is called a (directed) path.*

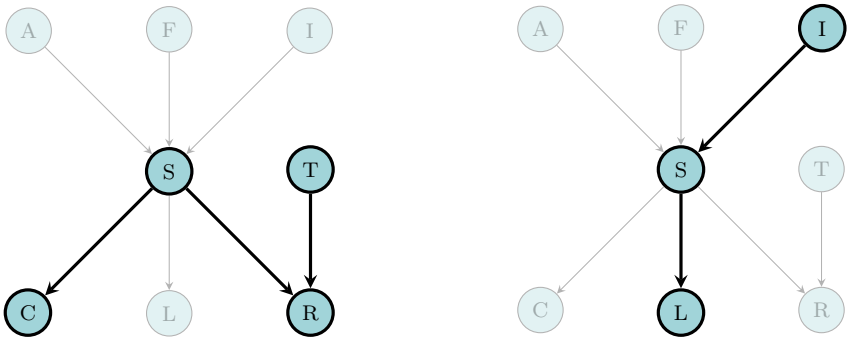


Figure 3.5: (a) The trail between C and T. (b) The direct path between I and L

Defining the concept of trail allows us to introduce the notion of ascendance (and descendance) between two variables of a BN. For any pair $X, Y \in \mathcal{V}$, X is said to be a ancestor of Y (and Y a descendant of X) if there is direct path from X to Y. As the Figure 3.5a shows, L is a descendant of I. If we denote by Descendants_X (resp. Ascendants_X) the set of its descendants (resp. ascendants) we can let NonDescendants_X denote the variables in the graph that are not descendants of X. As we have seen, an arbitrary choice is made on the order of enumeration of variables when factorizing a joint distribution with the equation 3.3. We will show in Example 3.14 how using a constrained order, called topological, helps reduce the complexity of such an equation.

Definition 3.11

Topological order *An order* $\{X_1, ..., X_n\}$ *upon* $\mathcal{V}$ *is said to be topological if, for all pair* $\{X_i, X_j\}$ *within this order, the existence of a direct path between* X_i *and* X_j *implies that* $i \leq j$.

3.2.3 Structural independences

The factorization of the joint distribution allowed by the equation 3.4 is explained by the existence of a structural independence property on $\vec{\mathcal{G}}$, called local Markov property, which can be defined using ancestry relationships between the variables of such graph.

Property 3.12

Local Markov Property *A DAG $\vec{\mathcal{G}}$ that verifies the local Markov property encodes a set of conditional independence assumptions, called the local independencies, denoted by $\mathcal{I}_l(\vec{\mathcal{G}})$, such that :*

For each variable X_i *:* $(\mathrm{X}_i \perp_{\mathbb{P}} \mathrm{NonDescendants}_{\mathrm{X}_i} \,|\, \mathrm{Pa}_{\mathrm{X}_i}^{\vec{\mathcal{G}}})$

In that regards, the structure of a Bayesian network can be seen as a set of independence assertions while $\mathbb{P}$ is defined through the set of CPTs associated with $\vec{\mathcal{G}}$. In this section we will show that these definitions are equivalent. Indeed, a distribution $\mathbb{P}$ satisfies the local independencies associated with a graph $\vec{\mathcal{G}}$ if and only if $\mathbb{P}$ is representable as a set of CPTs associated with such a graph.

Definition 3.13

Independencies in $\mathbb{P}$ *Let $\mathbb{P}$ be a distribution over χ. We define $\mathcal{I}(\mathbb{P})$ to be the set of independence assertions of the form $(\mathbf{X} \perp_{\mathbb{P}} \mathbf{Z}|\mathbf{Y})$ that hold in $\mathbb{P}$.*

Using such a notation, the statement "$\mathbb{P}$ satisfies the local independencies associated with $\vec{\mathcal{G}}$" can be simply stated as $\mathcal{I}_l(\vec{\mathcal{G}}) \subset \mathcal{I}(\mathbb{P})$. We say that $\vec{\mathcal{G}}$ is an *independency map* for $\mathbb{P}$, denoted $\mathcal{I}$-map.

Example 3.14

When applying the chain rule on $\vec{\mathcal{G}}_{SARS}$ using a topological order, the resulting factorization can be simplified assuming that $\vec{\mathcal{G}}_{SARS}$ is an $\mathcal{I}$-map for $\mathbb{P}$. Knowing, for example, that $(\mathrm{C} \perp_{\mathbb{P}} \{\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{T}\}|\mathrm{S})$, the following equality holds $\mathbb{P}(\mathrm{C}|\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{S},\mathrm{T}) = \mathbb{P}(\mathrm{C}|\mathrm{S})$. As a consequence, we can rewrite such an equation :

$$\begin{aligned}
\mathbb{P}(\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{S},\mathrm{T},\mathrm{C},\mathrm{L},\mathrm{R}) &= \mathbb{P}(\mathrm{A})\,\mathbb{P}(\mathrm{F}|\mathrm{A})\,\mathbb{P}(\mathrm{I}|\mathrm{A},\mathrm{F})\,\mathbb{P}(\mathrm{S}|\mathrm{A},\mathrm{F},\mathrm{I}) \\
&\quad \mathbb{P}(\mathrm{T}|\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{S})\,\mathbb{P}(\mathrm{C}|\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{S},\mathrm{T}) \\
&\quad \mathbb{P}(\mathrm{L}|\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{S},\mathrm{T},\mathrm{C}) \\
&\quad \mathbb{P}(\mathrm{R}|\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{S},\mathrm{T},\mathrm{C},\mathrm{L}) \\
&= \mathbb{P}(\mathrm{A})\,\mathbb{P}(\mathrm{F}|\mathrm{A})\,\mathbb{P}(\mathrm{I}|\mathrm{A},\mathrm{F})\,\mathbb{P}(\mathrm{S}|\mathrm{A},\mathrm{F},\mathrm{I}) \\
&\quad \mathbb{P}(\mathrm{T}|\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{S})\,\mathbb{P}(\mathrm{C}|\mathrm{S}) \\
&\quad \mathbb{P}(\mathrm{L}|\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{S},\mathrm{T},\mathrm{C}) \\
&\quad \mathbb{P}(\mathrm{R}|\mathrm{A},\mathrm{F},\mathrm{I},\mathrm{S},\mathrm{T},\mathrm{C},\mathrm{L})
\end{aligned}$$

By iteratively looking through the independences that holds in $\vec{\mathcal{G}}_{SARS}$ we get back to the equation 3.5 given by the chain rule for BN.

The set of independences verified by the local independence property is satisfied by $\mathbb{P}$ but the opposite is not necessarily true, some independences may not be expressed by this assumption. In the next section we will present an intuitive way to find other ones.

3.2.4 D-separation

The d-separation criterion (for directional separation)[Pearl 1986; Geiger et al. 1989] detects, given three disjoint set of nodes $\mathbf{X},\mathbf{Y},\mathbf{Z} \in \mathcal{V}$, if $\mathbf{X}$ is independent of $\mathbf{Z}$ given $\mathbf{Y}$ by testing if all trails from $\mathbf{X}$ to $\mathbf{Z}$ are blocked by $\mathbf{Y}$. This notion of blocked trail allows us to see the propagation of knowledge in a probabilistic graphical model as a flow of information.

Given a set of nodes $\mathbf{Y}$, a trail is called blocked if it contains a closed two-arc trail given $\mathbf{Y}$, the trail is said to be active otherwise. Before describing under which condition a two-arc trail is said to be closed, it is necessary to distinguish the different forms it can take. Consider a two-arc trail $\{\mathrm{X},\mathrm{Y},\mathrm{Z}\}$, it can have three different structure (illustrated in Figures 3.6, 3.7 and 3.8) :

- Sequential : $X \rightarrow Y \rightarrow Z$ or $X \leftarrow Y \leftarrow Z$;

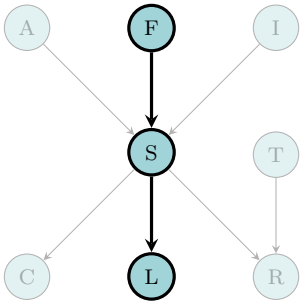


Figure 3.6: A sequential structure

- Divergent : $X \leftarrow Y \rightarrow Z$;

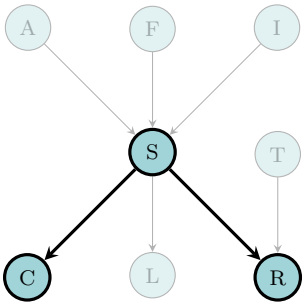


Figure 3.7: A divergent structure

- Convergent (also known as v-structure) : $X \rightarrow Y \leftarrow Z$;

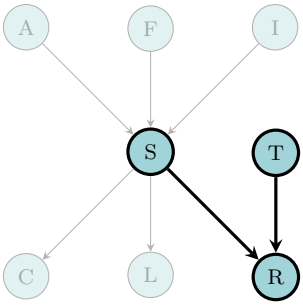


Figure 3.8: A v-structure

In the case of a sequential or divergent structure, the middle node blocks the flow of information once it is observed. This implies that any evidence over X will not change our beliefs about Z once Y is known. The third case behaves differently. Indeed, a v-structure $X \rightarrow Y \leftarrow Z$ block information between X and Z if Y is not observed : evidence on X (resp.Z) does not change our beliefs about Z (resp. X). But if we have evidence over Y or Descendants_Y, then evidence about X (resp.Z) does influence our beliefs about Z (resp.X) (recall the example of the sum of a dice throw in the section 3.1.3).

a trail $\{X_1, X_2, X_3\}$	closed given $\mathbf{Y}$
Sequential or divergent	$X_2 \in \mathbf{Y}$
V-structure	neither $\{X_2\}$ nor $\text{Descendants}_{X_2} \subset \mathbf{Y}$

Table 3.2: Conditions under which a two-arc trail is closed

Definition 3.15

d-separation *Let* $\mathbf{X}$, $\mathbf{Y}$ *and* $\mathbf{Z}$ *be disjoint sets of nodes in* $\vec{\mathcal{G}}$. *We say that* $\mathbf{X}$ *an* $\mathbf{Z}$ *are d-separated by* $\mathbf{Y}$, *written* $\mathbf{X} \perp_{\vec{\mathcal{G}}} \mathbf{Z}|\mathbf{Y}$, *iff every path between a node in* $\mathbf{X}$ *and a node in* $\mathbf{Z}$ *is blocked by* $\mathbf{Y}$. *We use* $\mathcal{I}(\vec{\mathcal{G}})$ *to denote the set of independencies that correspond to d-separation :*

$$\mathcal{I}(\vec{\mathcal{G}}) = \{(\mathbf{X} \perp_{\vec{\mathcal{G}}} \mathbf{Z}|\mathbf{Y})\}$$

Using such graphical criterion allows us to assert in a simple and yet effective way whether two variables are probabilistically dependent given our knowledge about another ones.

One may wonder whether for every two nodes X, Z d-separated by some set $\mathbf{Y}$, we have X is conditionally independent from Z given $\mathbf{Y}$, w.r.t $\mathbb{P}$. This is the purpose of the following proposition :

Property 3.16

d-separation soundness *Let* $\mathcal{B} = (\vec{\mathcal{G}}, \mathbb{P})$ *be a BN and* $\mathbf{X}$, $\mathbf{Y}$ *and* $\mathbf{Z}$ *be disjoint sets of nodes in* $\mathcal{V}$, *then* $(\mathbf{X} \perp_{\vec{\mathcal{G}}} \mathbf{Z}|\mathbf{Y}) \Rightarrow (\mathbf{X} \perp_{\mathbb{P}} \mathbf{Z}|\mathbf{Y})$, *i.e.* $\mathcal{I}(\vec{\mathcal{G}}) \subseteq \mathcal{I}(\mathbb{P})$.

This property is very useful from a modelling perspective. Indeed, to obtain a valid BN we can specify direct dependencies and fill the resultant CPTs to obtain a factorized probability distribution for which the BN graph is an I-map. Consequently, defining a BN is a more simple and intuitive task.

Theorem 3.17

d-separation weak completeness *Let* $\vec{\mathcal{G}}$ *be a directed graph whose nodes are a set of random variable* $\mathbf{X}$*. If* X *and* Z *are not d-separated given* $\mathbf{Y}$ *in* $\vec{\mathcal{G}}$*, then* X *and* Z *are dependent given* $\mathbf{Y}$ *in some distribution* $\mathbb{P}$ *that factorizes over* $\vec{\mathcal{G}}$*.*

This notion of weak completeness tells the d-separation cannot always detect all the conditional independences that hold in $\mathbb{P}$. Generally, the conditional independences of $\mathbb{P}$ verify a number of properties called *graphoid axioms* [Pearl and Paz 1986]. Several other methods allow to test conditional independences implied by these axioms, we can for example mention the u-separation [Butz et al. 2016] or the Markov separation [Lauritzen et al. 1990].

➢ Model Interpretation and Expert Knowledge

The absence/presence of an arc between two variables describes a dependency relationship. From a quality perspective and while it does not exclude the existence of a possible latent variable, unknown in the model, Bayesian networks allow us to easily take into account a priori expert knowledge about correlation between variables (e.g., a patient's age influences the severity of his or her infection) [de Campos and Castellano 2007]. From a quantitative point of view, asking an expert to estimate a probability is much simpler (and natural) with such a framework.

3.3 Queries and inferences in PGMs

3.3.1 Usual queries

Several types of queries are supported by PGMs, algorithms to answer some of them are presented in section 3.3.3.

3.3.1.1 Posterior Probability Distribution

The most common query in a PGM concerns the search for the probability of a set of variables $\mathbf{V}$ being given observations on others. Let $\mathbf{E}$ be the subset of variables of the model whose values are known, this subset represents *evidence* on the model. (e,g, 'the patient is 27 years old', 'the patient lost his smell or taste','the patient CT scan is positive). This measure, called *posterior probability distribution*, is noted $\mathbb{P}(\mathbf{V}|\mathbf{E} = e)$. In the examples in Figure 3.9 and 3.10 we show the differences in the posterior probabilities within our SARS-CoV example given knowledge about a patient. If we know that a young patient has had a positive CT-scan and has lost his taste or smell then the probability that he his mildly affected by the virus increases from 0.53% to almost 69%.

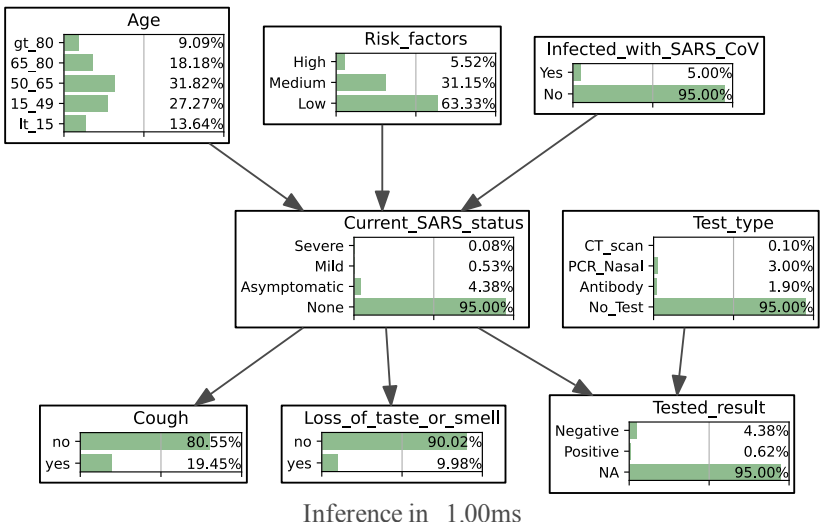


Figure 3.9: An inference without evidence

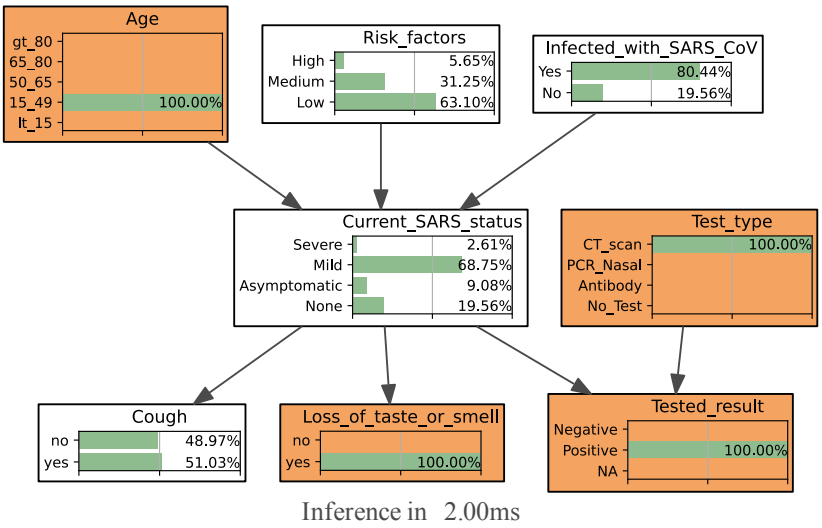


Figure 3.10: An inference with evidence (in orange)

3.3.1.2 Most Probable Explanation

This query tries to find the most probable instantiation of variables $\mathbf{Q} = \mathbf{V} \backslash \mathbf{E}$ given the evidence $\mathbf{E} = e$. To put it another way, this amounts to find the assignment of $\mathbf{Q}$ that maximizes $\mathbb{P}(\mathbf{Q}|e)$ i.e., computing the quantity :

$$\text{argmax}_{q \in \mathbf{Q}} \mathbb{P}(q|e)$$

3.3.1.3 Maximum A Posteriori

This is a more general case of MPE in the sense that while $\mathbf{Q}$ covers all non evidence variables, it is now just a subset of variables for which we therefore seek to find a high-probability joint assignment. It is also called the marginal MAP, because this involves both marginalizing out some variables and computing some argmax on others. More precisely, if $\mathbf{W} = \mathbf{V} \setminus \{\mathbf{Q} \cup \mathbf{E}\}$, MAP computes :

$$\text{argmax}_{q \in \mathbf{Q}} \sum_{w \in \mathbf{W}} \mathbb{P}(q, w|e)$$

3.3.2 Operations Between Potentials

The data used to manipulate PGMs (notably their CPTs) are stored in so-called potentials that can be viewed as multi-dimensional table where a value is assigned to each instantiation of a set of variables $\mathbf{X}$.

Definition 3.18

Potential *A potential over a set of discrete random variables* $\mathbf{X}$ *is a function* ϕ *such that* $\phi :$ $\mathrm{Val}(\mathbf{X}) \mapsto \mathbb{R}$, *where* $\mathrm{Val}(\mathbf{X}) = \otimes_{\mathrm{X}\in\mathbf{X}} \mathrm{Val}(\mathrm{X})$. $\mathbf{X}$ *is called the domain of* ϕ *and denoted* $dom(\phi)$.

Most of the algorithms that will be described later in this document involve two operations on potentials : a product and a marginalization operation.

Definition 3.19

Product *Let* ϕ_1 *and* ϕ_2 *be two potentials over* $\mathbf{X}_1, \mathbf{X}_2$ *and* $\mathbf{X}_2, \mathbf{X}_3$ *respectively, with* $\mathbf{X}_1, \mathbf{X}_2$ *and* $\mathbf{X}_3$ *three disjoint sets of random variables (possibly empty). The potential product of* ϕ_1 *and* ϕ_2, *denoted* $\phi_1 \times \phi_2$, *is the potential* ψ *such that :*

$$\begin{array}{rcl} \psi : \mathrm{Val}(\mathbf{X}_1, \mathbf{X}_2, \mathbf{X}_3) & \rightarrow & \mathbb{R} \\ (\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3) & \mapsto & \phi_1(\boldsymbol{x}_1, \boldsymbol{x}_2) \times \phi_2(\boldsymbol{x}_2, \boldsymbol{x}_3) \end{array}$$

Definition 3.20

Marginalization *This operation eliminates a set of variables from a potential. If we have a potential* $\phi(\mathbf{X}, \mathbf{Y})$ *over two sets of variables* $\mathbf{X}, \mathbf{Y}$, *marginalizing* $\mathbf{Y}$ *produces a new potential* $\tau(\mathbf{X})$*:*

$$\tau(\boldsymbol{x}) = \sum_{\boldsymbol{y}} \phi(\boldsymbol{x}, \boldsymbol{y}),$$

where the sum is over all joint assignments to the set of variables $\mathbf{Y}$.

Remark 3.21

τ refers to the marginalized potential. Even if ϕ was a CPT, the marginalized potential does not necessarily correspond to a probability. For instance :

$$\sum_{X} \mathbb{P}(X|Y) = \mathbb{1}_Y$$

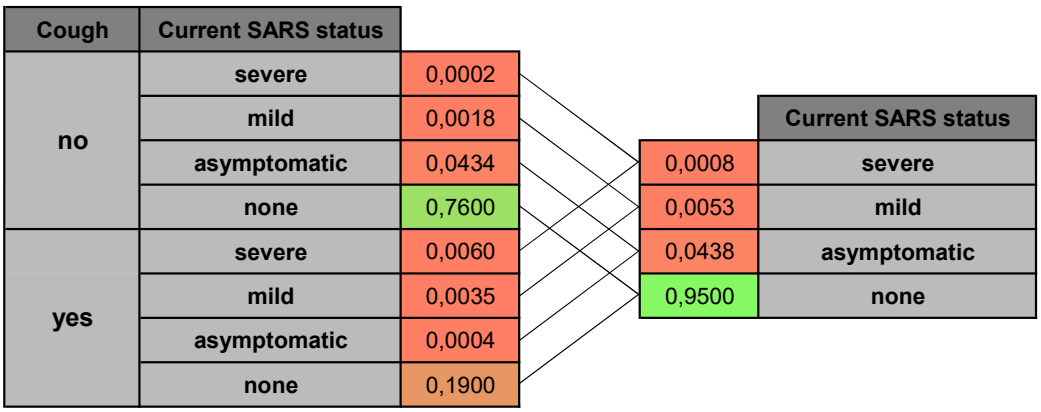


Figure 3.11: Example of marginalization $\mathbb{P}(S) = \sum_C \mathbb{P}(C,S)$

3.3.3 Inference algorithms

Numerous algorithms are used to compute the queries presented in Section 3.3. Two families of algorithms can be distinguished : exact algorithms, some of which manipulate secondary graphical structures in order to compute multiple queries at once, and approximated algorithms ones when the exact algorithms can no longer be used because of the spatial complexity of the models. Despite the various proposed optimizations and refined algorithms, the inference in a PGM is NP-Hard [Cooper 1990; Dagum and Luby 1993].

3.3.3.1 Exact Inferences

➢ Variable Elimination

Variable Elimination (VE) [Zhang and Poole 1994; Dechter 1999] is a simple algorithm for exact inference within PGMs. As its name indicates it consists in successively eliminating variables from a factorized joint distribution in order to compute MAPs or marginal distributions. The algorithm initializes a set of factors with the BN CPTs, then, each time a variable needs to be eliminated, compute the product of factors that contains such variable. The variable is then marginalized out from the combined factor. By repeating this elimination process until no more variables need to be removed, one can obtain the marginal distribution over any subset of variables. The performance of such an algorithm is related to the elimination order of the variables and finding such an optimal elimination order is, in itself, a NP-Hard problem [Dechter 1999; Arnborg 1985].

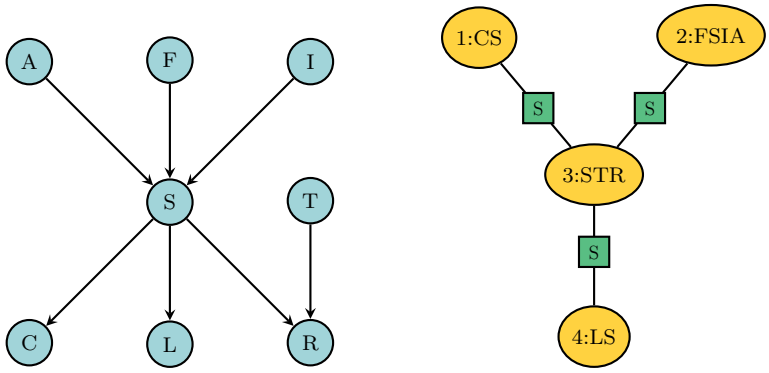
➢ Exact inferences with junction trees

One of the limitations of using VE is that it only allows one value to be calculated at a time. Algorithms based on the use of a secondary structure, called a junction tree (sometimes a clique tree), make it possible to overcome this problem.

Denoted $\mathcal{T}$, a junction tree is a structure generated from a DAG $\vec{\mathcal{G}}$ where all of $\vec{\mathcal{G}}$'s variables are grouped into clusters called cliques (yellow nodes in Figure 3.12) connected by separators (in green) such that $\mathcal{T}$ verifies the running intersection properties. Let $\mathcal{T}$ be a junction tree over a set of factors Φ. We denote by $\mathcal{V}(\mathcal{T})$ the vertices and by $\mathcal{E}(\mathcal{T})$ its edges.

Property 3.22

Running intersection property *$\mathcal{T}$ verifies the running intersection property if, whenever there is a variable* X *such that* $\mathrm{X} \in C_i$ *and* $X \in C_j$*, then* X *is also in every clique, and separator, in the path between* C_i *and* C_j *in* $\mathcal{T}$ *.*



(a) (b)

Figure 3.12: (a) a Bayesian network. (b) a possible junction tree

To have a detailed description of the procedure transforming a Bayesian network into a junction tree, one can refer to [Koller and Friedman 2009, chap. 10]. We can however note that the generation of a JT is not unique and even if heuristics are used to make it as simple as possible, the search for an optimal JT is NP-hard [Verner Jensen and Jensen 2013].

➢ Message passing protocol within JT

Algorithms, such as Shafer-Shenoy [Shenoy and Shafer 1990], Hugin [Jensen et al. 1990] or Lazy Propagation [Madsen and Jensen 2013], are based on a message passing protocol between adjacent nodes of the junction tree. For $(i,j) \in \mathcal{E}(\mathcal{T})$ we denote $\psi_{i\to j}$ the potential associated with the separator $\mathbf{S}_{ij}$ which represents the message from i to j over $\mathbf{S}_{ij}$. Since messages will be sent in both directions, we need to distinguish $\psi_{i\to j}$ from $\psi_{j\to i}$. In order to guarantee the correctness of computations, the message-passing protocol needs that the message $\psi_{i\to j}$ should be sent only when clique i has received messages from all of its neighbours except from j. Exchanging a message from i to j in the junction tree will propagate toward j information that has been gathered in i.

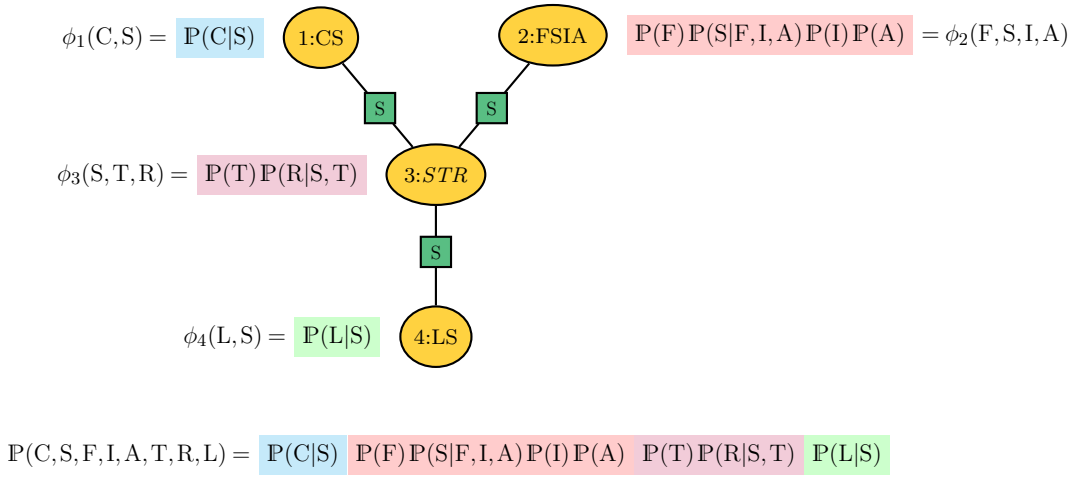


Figure 3.13: Initialized junction tree

One way to organize messages computations is to perform two phases, namely *Collect* and *Distribute* from a predetermined root $r \in \mathcal{V}(\mathcal{T})$. During the *Collect* phase, messages are sent along edges from leaves toward r (highlighted with a red node in figures 3.14 and 3.15).

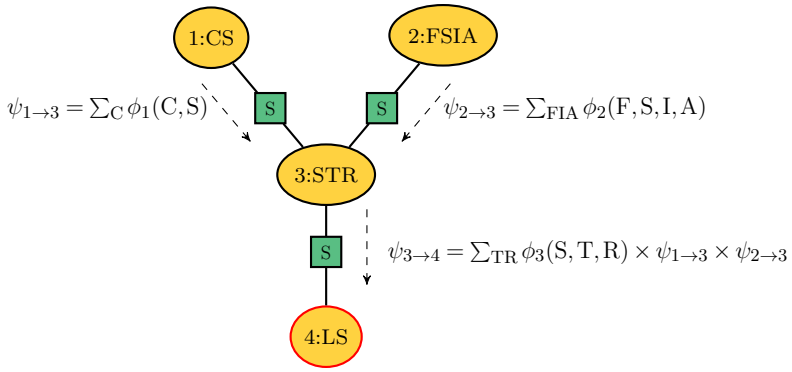


Figure 3.14: Message passing during the collection phase

During the *Distribute* phase, messages are sent from r towards the leaves. After propagating all the messages, in order to have the joint posterior distribution over the variables in $\mathbf{C}_i$ up to some normalization constant, all we need is to compute the product $\phi_i \times \prod_{(k,i)\in\mathcal{E}(\mathcal{T})} \psi_{k\to i}$, where ϕ_i denotes the potential associated with the clique $\mathbf{C}_i$.

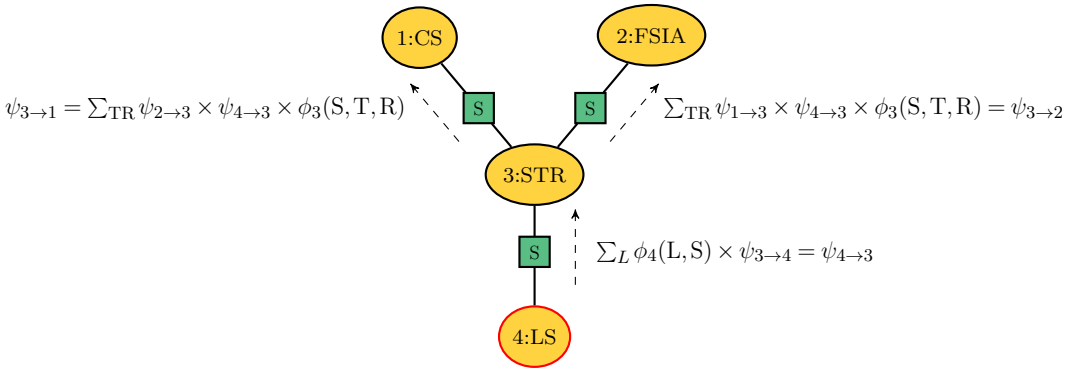


Figure 3.15: Message passing during the distribution phase

To sum up, an exact inference with junction trees manipulates potentials with two operations, a marginalization and a product (and a division in the case of Hugin), the limiting factor in its feasibility being the size of the potentials. Indeed, the complexity of an inference in a BN is NP-Hard [Cooper 1990], growing exponentially in the tree-width of the network. In the case of JT-based algorithm the tree-width is related to the size of its largest clique, determined by the products of the domains of its variables [Robertson and Seymour 1986].

As we will see in Chapter 6, if a more compact representation of the information contained in the potentials can be found and aforementioned operations redefined using such an embedding, we could make a more scalable inference, at the cost of a controlled approximation.

3.3.3.2 Approximate Inference

Exact methods make it possible to answer requests in *simple* cases but when models become more complex, it is quickly necessary to use approximate algorithms that offer to limit the spatial complexity of the models at the cost of an increase in computation time or a relaxation of the model constraints, which introduces an error in the results of these calculations, sometimes without offering any guarantee of convergence. We briefly outline here the principles of such approaches and again invite the interested reader to refer to [Koller and Friedman 2009, Chapters 11 and 12] for detailed explanation about these subjects.

➢ Loopy Belief Propagation

We saw that the junction tree algorithm has a running time that is potentially exponential in the size of the largest clique and that sometimes, the act of finding a tree minimizing this one is complicated, let alone inferring in it. However, it is not always necessary to look for an exact solution as provided by these algorithms; an approximation provided rapidly may be sufficient. The Loopy Belief Propagation algorithm (LBP) [Pearl 1988] is one of the traditional techniques that allows us to perform such an approximative calculation on complex graph.

The general idea behind LBP is to iteratively apply a message propagation algorithm (like the one presented before) but this time on a loopy graph in which the nodes will not wait to receive their messages before sending theirs. The algorithm will stop after a certain number of iterations of the message passing protocol or when the differences introduced by the updates are no longer significant.

This approach often works surprisingly well in practice [Murphy et al. 1999]. In general, however, convergence isn't guaranteed. Empirically we think that it probably converges on trees and on graphs with at most one cycle. If the method does converge, its beliefs may not necessarily equal the true marginals, although very often in practice they will be close. LBP can be viewed as a structurally relaxed version of the junction tree algorithms presented above.

➢ Sampling methods

Based on stochastic sampling, these methods (such as Gibbs Sampling or Metropolis-Hastings) can be used to perform both marginal, MAP inference queries as well as the computation of interesting quantities, such as expectations [Mackay 1998]. The general idea behind these approaches is no longer to infer from a distribution $\mathbb{P}$ that would be too complex to evaluate, but to draw samples from this distribution in order to, at some point, converge to the targeted value.

The main disadvantage of these methods is that the quality of the approximation is directly related to the number of samples performed and therefore to the time spent sampling. Furthermore, it is difficult to estimate *when* an approximation will be accurate.

➢ **Variational methods**

Contrary to the methods presented above, variational approaches (such as the Mean-Field approximation) will use optimization techniques in order to approach a distribution. Variational techniques will try to solve an optimization problem over a class of tractable distributions $\mathbb{Q}$ in order to find a $\mathbb{Q} \in \mathbb{Q}$ that is most similar to $\mathbb{P}$. We will then query $\mathbb{Q}$ (rather than $\mathbb{P}$) in order to get an approximate solution. The formulation of inference as a constrained optimization problem opens the door to the application techniques developed in the optimization literature (such as stochastic gradient optimization, parallelization over multiple processors, and GPU acceleration). If these methods have an increasing popularity [Blei et al. 2017; Zhang et al. 2018] they are nevertheless more complex to use, notably because several design choices have to be made between the objective function that we aim to optimize, the space of pseudo-distribution $\mathbb{Q}$ used as well as the algorithm to choose to perform the optimization.

3.3.4 Large Scale BN and Expressivity

When attempting to model large scale problems, on an industrial scale for example, with BN, we are confronted with a challenge : these are quickly tedious to handle and loose expressiveness. Let's consider, as showcased in the example in Figure 3.16, that some random variables count the number of patients with a severe form of disease for each physician (the green nodes). If the reading of such a network is already complex in such a simple setting, how could we easily model and monitor systems containing tens or hundreds of patients ? Several models, such as OOBN [Bangsø and Wuillemin 2000] and PRM [Koller and Pfeffer 1998; Pfeffer 2000; Torti et al. 2010], offer an answer to this problem.

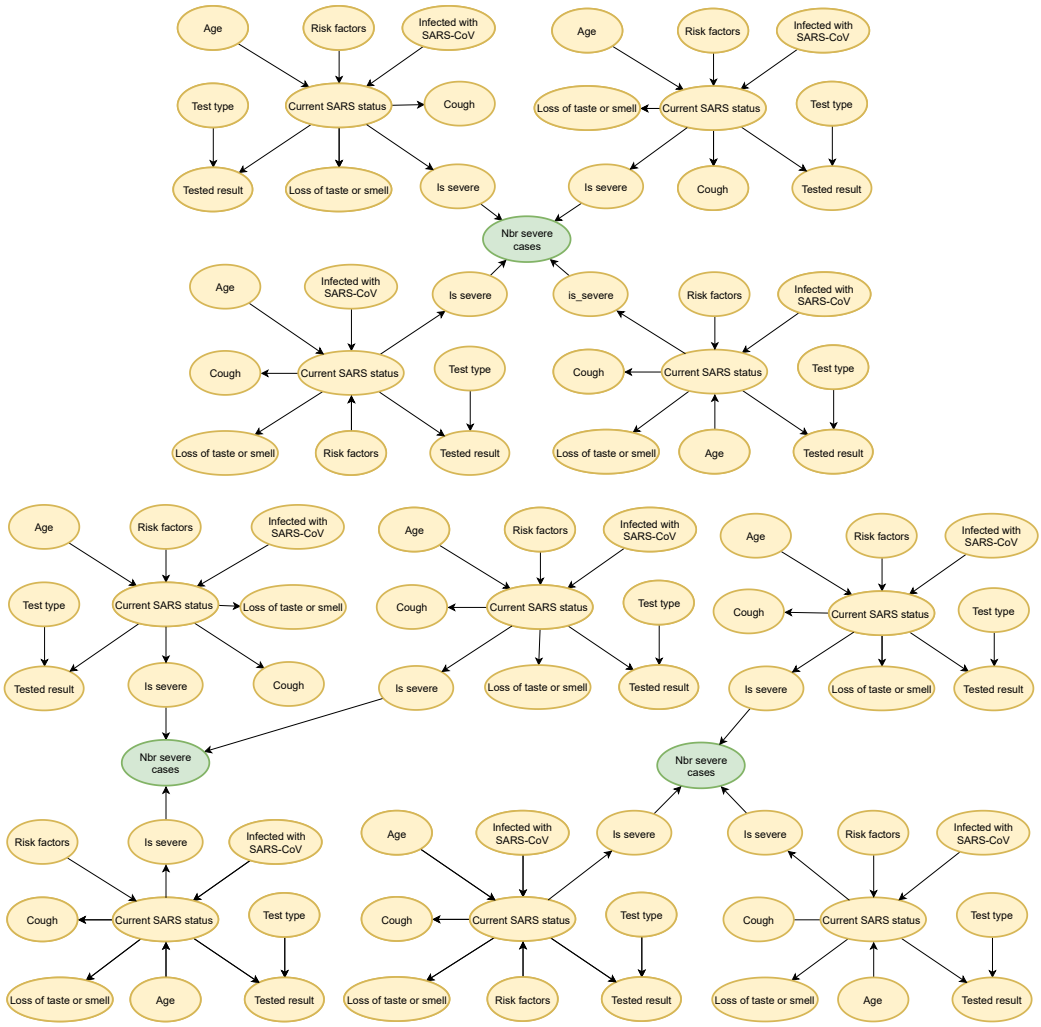


Figure 3.16: BN representing a system with 3 physicians and 10 patients

3.4 Probabilistic Relational Models

Probabilistic relational models (PRM) are combining notions from BNs and from the paradigm of object-oriented languages, where the focus is set on classes of objects and by defining relations among them. The expressiveness gained when adding notions of random variables and conditional probabilities to classes, attributes, relations, interface, inheritance and instantiations makes graphical models reusable and scalable [Medina Oliva et al. 2010].

3.4.1 Model

The random variables of a PRM are grouped within classes, it is a means of abstracting a certain complementarity between them. We will, for example, speak of the *Patient* class or the *Physician* one. Each class can be seen as an independent DAG, the variables characterizing the relationships within a class are called attributes. To allow communication between attributes of two different classes, it is possible to define references. We will show that this abstraction allows us to easily instantiate a complex model to a specific context. Now let us give formal definitions for PRM concepts. We will illustrate such concepts with the help of a simple model, presented in Figure 3.17, from [Torti 2012].

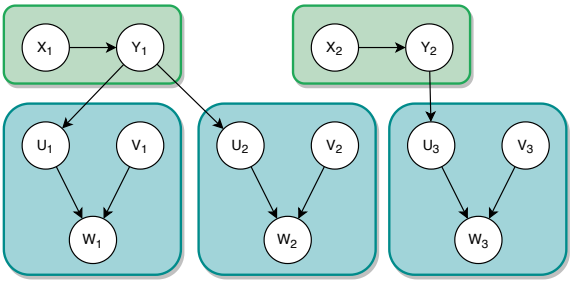


Figure 3.17: A BN with abstractable classes

Definition 3.23

Classes *In PRM, the world consists of base entities, partitioned into classes* $\mathcal{C}_1, \mathcal{C}_2, ..., \mathcal{C}_n$*. Each class is associated with a set of attributes* $A(\mathcal{C})$ *and a set of reference slots* $R(\mathcal{C})$*.*

Classes could be identified thanks to recurring patterns, as shown in Figures 3.17 and 3.18. In complex systems there are often many random variables sharing the same domain, thus by defining once an attribute's type we reduce the amount of redundant information that must be specified when modeling the system.

Definition 3.24

Attribute's type *An attribute's type τ describes a family of distinct discrete random variables sharing the same domain $\tau = l_1, \cdots, l_n$, where n is the domain size of τ.*

Random variables such as *Cough*, *Infected with SARS-CoV* and *Loss of taste or smell* described in the table 3.1, share the same type.

Definition 3.25

Attributes *Let $\mathcal{C}$ be a class. An attribute* X *of a class $\mathcal{C}$, denoted* $\mathcal{C}$.X *in case of ambiguity about its class, is a triplet $< \tau_{\mathrm{X}}, \mathrm{Pa(X)}, \phi_{\mathrm{X}} >$, where τ_{X} is* X*'s type,* Pa(X) *its set of parents and ϕ_{X} a potential encoding its probability distribution according to* Pa(X).

Definition 3.26

Reference slots *Let $\mathcal{C}$ and $\mathcal{D}$ be two classes. A reference slot $\mathcal{C}.\rho = \mathcal{D}$ is a pointer in $\mathcal{C}$ that refers to $\mathcal{D}$ and allows us to access one of its element. We say that $\mathcal{C}$ is the domain of $\mathcal{C}.\rho$ denoted $Domain(\mathcal{C}.\rho)$ and $\mathcal{D}$ is the range of $\mathcal{C}$, denoted $Range(\mathcal{C}.\rho)$. A reference slot is simple if it relates one class to one class and it is multiple if it relates one class to many. The inverse of $\mathcal{C}.\rho$ is called inverse slot and is denoted $\mathcal{D}.\rho^{-1}$. We have $Range(\mathcal{D}.\rho^{-1}) = Domain(\mathcal{C}.\rho)$ and $Domain(\mathcal{D}.\rho^{-1}) = Range(\mathcal{C}.\rho)$.*

A class thus abstracts the interactions between the random variables it contains and can be seen as a fragment of a BN on its attributes. The figure 3.18 shows the abstract classes from the figure 3.17 and the corresponding reference slot (dashed parts). The mechanism of reference slot can be applied recursively to allow one attribute to access attributes of different classes by navigating through other classes, we call this path of reference a slot chain.

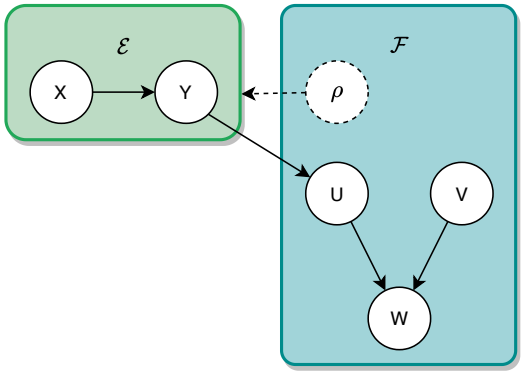


Figure 3.18: An abstraction of the BN in Figure 3.17

Definition 3.27

Slot chain *PRM also define the concept of slot chain* $\rho_1, ..., \rho_k$ *as a sequence of relation slots such that* $\forall i, Range[\rho_i] = Domain[\rho_{i+1}]$.

3.4.1.1 PRM Instantiation and Complex Networks

These notions make it possible to define the constituent elements of a PRM. From these descriptions, one can easily define more or less complex systems through a so called *PRM system*. An extended example of a PRM declaration in the o3prm format can be found online [2].

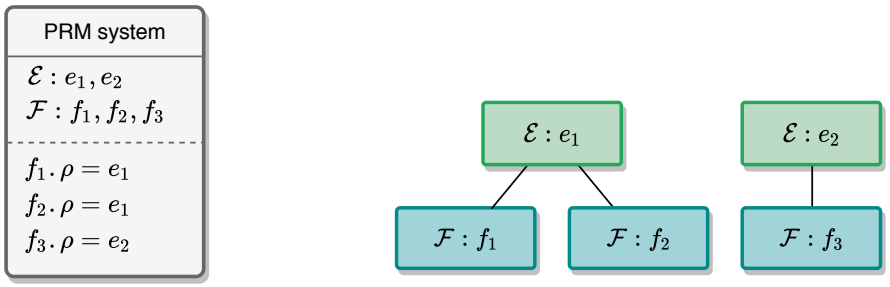
Definition 3.28

Instance *An instance* c *of a class* $\mathcal{C}$ *is an actual object of this class, i.e. , a BN fragment whose attributes are random variables generated from the class level template and where references slots refer to sets of their range's instances.*

Definition 3.29

Relational Skeleton *A relational skeleton* $\mathcal{S}$ *is a set of instances such that for any instance* c *of class* $\mathcal{C}$ *and any reference slot* $\mathcal{C}.\rho = \mathcal{D}$*, there exists at least one instance* $d \in \mathcal{S}$ *such that* d *is an instance of* $\mathcal{D}$ *and* $d \in Range(c.\rho)$.

[2] `https://gitlab.com/agrumery/pgmrepository/-/blob/master/o3prm/complexprinters_system.o3prm`



(a) (b)

Figure 3.19: (a) a PRM system. (b) the relational skeleton of such system

3.4.1.2 PRM and Grounded BN

Now that these elements have been specified, we can formally define what a PRM is and, interestingly, specify a procedure for generating a Bayesian network, known as a grounded BN, from that PRM.

Definition 3.30

PRM *A PRM* $\mathbf{M}$ *is a pair* $(\mathcal{C},\mathcal{S})$*, where* $\mathcal{C}$ *is a set of classes and* $\mathcal{S}$ *is a relational skeleton. It encodes the joint probability distribution over* $A(\mathcal{S})$*, the set of all instance attributes in* $\mathcal{S}$*, as the following product:*

$$\mathbb{P}(A(\mathcal{S})) = \prod_{C \in \mathcal{C}} \prod_{i \in I_{\mathcal{S}(C)}} \prod_{i.\mathrm{X} \in A(i)} \mathbb{P}(i.\mathrm{X} | \mathrm{Pa}(i.\mathrm{X}))$$

Given a PRM $\mathbf{M} = (\mathcal{C},\mathcal{S})$, its associated grounded Bayesian network is a BN constructed using the following steps :

1. There is a node for every attribute i.X of every instance $i \in S$, named i.X.

2. Each i.X depends probabilistically on parents of the form i.Y or j.Y such that there exists a slot chain K with $j \in Range(i.\mathrm{K})$.

3. i.X's conditional probability distribution is a CPT generated from the attribute's CPT of the corresponding class.

3.4.2 PRM Specific Inferences

All the inferences presented can be used on the grounded BN generated from the instantiation of a PRM but it would be not using one of the characteristics of PRMs: their structural redundancy. Some works have proposed, in this perspective, to adapt classical algorithms to PRMs. Structured Variable Elimination (SVE) [Pfeffer 2000] proposes an extension of VE exploits structural repetition in open worlds systems to reduce the number of computations. In [Torti and Wuillemin 2010; Wuillemin and Torti 2012], authors proposed an adaptation of the Bayes-Ball algorithm [Shachter 1998], to speed-up the computation.

3.5 Conclusion

In this chapter we have presented the tools that PGMs are, the theory behind their definition and their interest, especially on a practical level, thanks to their expressiveness or the richness of existing methods for their manipulation. In the introduction we mentioned some practical and theoretical limits related to the use of uncertain reasoning in RBS. We will now discuss the various approaches studied within the IBM France Lab regarding the use of PGM could allow users who are neither mathematicians nor computer scientists to manipulate probabilistic rules.

Bibliography

Arnborg, S. (1985). Efficient algorithms for combinatorial problems on graphs with bounded, decomposability—a survey. *BIT*, 25(1):2–23.

Bangsø, O. and Wuillemin, P.-H. (2000). Object oriented bayesian networks a framework for topdown specification of large bayesian networks and repetitive structures.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational in-

ference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Butz, C. J., dos Santos, A. E., Oliveira, J. S., and Gonzales, C. (2016). A simple method for testing independencies in bayesian networks. In Khoury, R. and Drummond, C., editors, *Advances in Artificial Intelligence*, pages 213–223, Cham. Springer International Publishing.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2):393 – 405.

Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in bayesian belief networks is np-hard. *Artificial Intelligence*, 60(1):141 – 153.

de Campos, L. M. and Castellano, J. G. (2007). Bayesian network learning algorithms using structural restrictions. *International Journal of Approximate Reasoning*, 45(2):233 – 254. Eighth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005).

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1):41 – 85.

Fenton, N., McLachlan, S., Lucas, P., Dube, K., Hitman, G., Osman, M., Kyrimi, E., and Neil, M. (2020). A privacy-preserving bayesian network model for personalised covid19 risk assessment and contact tracing. *medRxiv*.

Geiger, D., Verma, T., and Pearl, J. (1989). d-separation: From theorems to algorithms. In *UAI*, pages 139–148.

Halpern, J. Y. (2017). *Reasoning about Uncertainty, second edition*.

Jensen, F. V., Olesen, K. G., and Andersen, S. K. (1990). An algebra of bayesian belief universes for knowledge-based systems. *Networks*, 20(5):637–659.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Koller, D. and Pfeffer, A. (1998). Probabilistic frame-based systems. In *AAAI/IAAI*, pages 580–587.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N., and Leimer, H.-G. (1990). Independence properties of directed markov fields. *Networks*, 20(5):491–505.

Mackay, D. J. C. (1998). *Introduction to Monte Carlo Methods*, pages 175–204. Springer Netherlands, Dordrecht.

Madsen, A. L. and Jensen, F. V. (2013). Lazy propagation in junction trees.

Medina Oliva, G., Weber, P., Levrat, E., and Iung, B. (2010). Use of probabilistic relational model (PRM) for dependability analysis of complex systems. In *IFAC Proceedings Volumes (IFAC-PapersOnline)*.

Murphy, K. P., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, page 467–475, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Neapolitan, R. E. (2003). *Learning Bayesian Networks*. Prentice-Hall, Inc., USA.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241 – 288.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference (Morgan kaufmann series in representation and reasoning)*. Morgan Kaufmann Publishers, San Mateo, Calif.

Pearl, J. and Paz, A. (1986). Graphoids: Graph-based logic for reasoning about relevance relations or when would x tell you more about y if you already know z? In *Proceedings of the 7th European Conference on Artificial Intelligence - Volume 2*, ECAI'86, page 357–363. North-Holland.

Pfeffer, A. (2000). *Probabilistic reasoning for complex systems*. PhD thesis, Stanford University.

Robertson, N. and Seymour, P. (1986). Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322.

Shachter, R. D. (1998). Bayes-ball: Rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, page 480–487, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Shafer, G. and Pearl, J. (1990). *Readings in Uncertain Reasoning*. Morgan Kaufman Publishers.

Shenoy, P. P. and Shafer, G. (1990). Axioms for probability and belief-function propagation. In *Uncertainty in Artificial Intelligence*. Elsevier, Amsterdam.

Torti, L. (2012). *Structured probabilistic inference in object-oriented probabilistic graphical models*. Theses, UPMC.

Torti, L. and Wuillemin, P.-H. (2010). Structured Value Elimination with D-Separation Analysis. In *International Florida Artificial Intelligence Research Society Conference*, pages 122–127, Daytona Beach, United States.

Torti, L., Wuillemin, P. H., and Gonzales, C. (2010). Reinforcing the object-oriented aspect of probabilistic relational models. In *Proceedings of the 5th European Workshop on Probabilistic Graphical Models, PGM 2010*.

Verner Jensen, F. and Jensen, F. (2013). Optimal junction trees. *arXiv*, pages arXiv–1302.

Wuillemin, P.-H. and Torti, L. (2012). Structured Probabilistic Inference. *International Journal of Approximate Reasoning*, 53(7):946–968.

Zhang, C., Butepage, J., Kjellstrom, H., and Mandt, S. (2018). Advances in variational inference.

Zhang, N. and Poole, D. (1994). A simple approach to bayesian network computations. In *7th Canadian Conference on Artificial Intelligence*.

Chapter 4

Probabilistic Production Rules

Contents

An internal study [Ait-Kaci and Bonnard 2011], several internships [Arru 2011; Perez 2013] and a PhD thesis [Agli 2017] have shown that a weak coupling between ODM and probabilistic inference engines makes it possible to reason on probabilistic data by introducing the notion of probabilistic production rules.

4.1 PPR Studies

The main objective of [Arru 2011] was to introduce uncertain reasoning in a BRMS throughout the development of a prototype integrated in JRules' rule engine (JRules is the former name of ODM). One of the contributions of this internship was the introduction of a *probability* operator in the engine language, as shown in the following figure.

In the rest of the manuscript and for readability reasons, the random variables present in the rules will be highlighted with **teal**.

```
rule planExaminationRule {
    when {
        p : Patient(probability(p.LungCancer) > 0.4);
    } then {
        planExamination(patient);
    }
};
```

Figure 4.1: Example of original PPR rule

In order to map the random variables and class attributes of the problem, JRules object model was extended with the help of annotations. In the developed prototype, each class containing probabilistic attributes was linked to a Bayesian network with the annotation *@PprClass* and probabilistic attributes linked to Bayesian variables with the annotations *@PprVariable*. These annotations made it possible to query, at runtime, the probability in the Bayesian engine when the rule evaluate the predicate including the *probability* operator. A first light coupling between the two engines was therefore introduced.

In addition to a comparison between different probabilistic inference engines (SMILE [Druzdzel 1999], Hugin [Andersen et al. 1989], unBBayes [Matsumoto et al. 2011], Bayesia [Conrady and Jouffe 2015], ProBT [Mekhnacha et al. 2006]), the use of models other than BN, such as OOBN [Bangsø and Wuillemin 2000] or PRM [Koller and Pfeffer 1998; Pfeffer 2000] were mentioned, particularly since the former are inadequate to represent large and complex domains such as defined in JRules BRMS. This work concluded that "limitation of Bayesian networks is a direct consequence of the fact that they lack the concept of an object (or domain entity)" (*i.e.* , there is a mismatch between the object model of the rule engine and the flat model of the Bayesian network).

4.2 URBS

In this section we will focus on some of the contributions made in URBS [Agli 2017] insofar as they are the starting point of the current thesis.

4.2.1 ARL-PRM Mapping

The contribution that we are most interested in concerns the use of PRM instead of BN. As it has been shown, the structuring of the information in PGMs is very close to the one in the object data model defined in the rule engine. The conceptual similarities (classes, attributes, relationships) between the two models ease the interaction between the two engines. In this case it is no longer just a question of mapping the variables but of defining the CPTs directly in the object model, so a PRM model could be generated directly from the annotated version of the OM.

The example below shows an excerpt of what an OM declaration using these annotations looks like. The Patient class will be translated in the PRM using the @PrmClass annotation. It contains several random variables such as *status* or *is_severe* that describe the condition of a patient and indicate if it is severe. These annotations (@PrmAttribute and @PrmAggregator) are composed of the elements necessary for their creation (name of parents, cpt, aggregation type, ...). Dependencies between classes are declared thanks to @PrmReference and @PrmMultiReference annotations.

```
@PrmRestrictedTypeClass(modalities={"severe","mild","asymp","none"})
public StatusType restricts int#[0,3];

@PrmClass
public class Patient {
    public String name;

    @PrmAttribute(parents={"infected","age","risk_factors"},cpt=...)
    public StatusType status;

    @PrmAttribute(parents={"status"},cpt={{1,1,1,0},
                                          {0,0,0,1}})
    public int is_severe;

    ...
}

@PrmClass
public class Physician {
    @PrmMultiReference
    public Patient[] pats;

    public String name;

    public String specialty;

    @PrmAggregator(aggName="sum",attribute ="pats.is_severe",mod="")
    public int number_of_severe_patients;

    ...
}
```

Figure 4.2: Example of the use of annotations during the declaration of a system

It is important to note that this definition phase is agnostic of the chosen probabilistic inference engine. This generality contributes to the looseness of the coupling between the two engines and allows greater freedom of use but leaving the task of determining which variables are probabilistic in the model induces a selection bias that is not without consequence. This is one of the limitations of the approach.

4.2.2 IJTI

Since the working memory of a BRMS evolves incrementally, [Agli et al. 2016] proposed a new exact inference algorithm to take such property into account. The key idea was to take into account previous computations within a junction tree to optimize a new inference by computing only the parts of the tree modified by the increment. This algorithm, well adapted for a combined use of a BRMS and probabilistic graphical models, is no longer applicable when the initial query is not computable (let us recall that one of the main criteria of feasibility of an inference based on the use of a junction tree is the size of its cliques).

4.2.3 Toolchain

Figure 4.3 shows how BIS, the plugin developed in URBS, fits into the ODM compilation chain. Different semantic trees are generated there after analysing and checking the syntax of the user file describing the rules and objects of the model (*IRL*). In addition to producing the one describing the rules (*SemRuleset*), this study proposed to use the object model to generate a probabilistic relational model (*SemPRM*), then to rewrite the probabilized expressions of the rules in order to be able to make calls to an inference engine using the generated file (*PRM file*).

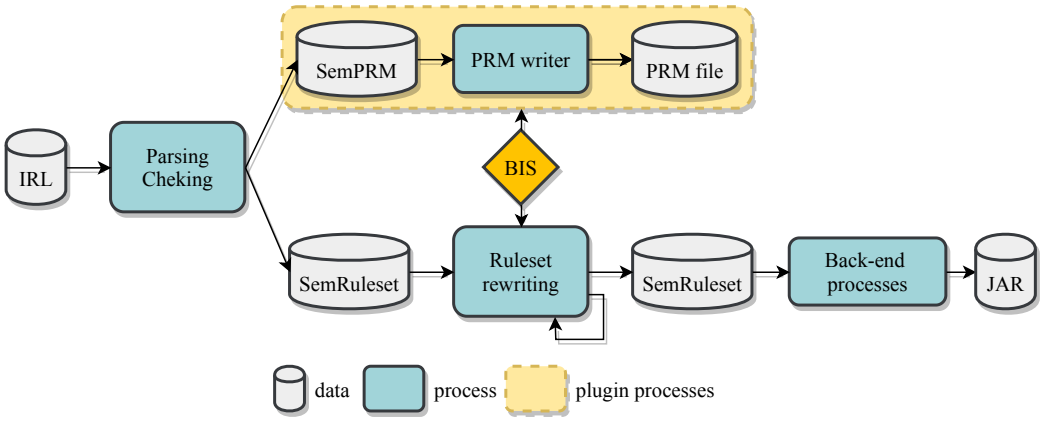


Figure 4.3: BIS compilation process

4.2.4 Runtime Engine Coupling

The evaluation of probabilistic queries and the monitoring of the PRM System is made possible by the use, at runtime, of a second engine, dedicated to this type of computation. This engine is parameterized prior to compilation and must support two types of interaction with the rule engine, as showed in Figure 4.4:

- The BRE can request posteriors of query to the probabilistic engine,
- The BRE can inform the probabilistic engine of changes or observations to keep the PRM system up-to-date. If a patient is attached to a new physician, references must be updated within the PRM, for example.

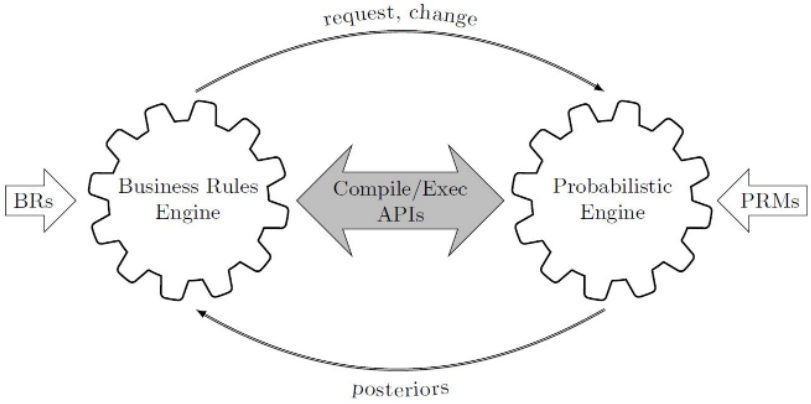


Figure 4.4: BIS coupling

aGrUM has been choosen to be coupled with ODM's rule engine for efficency and accessibility reasons. The aGrUM [Gonzales et al. 2017][1] framework, primarily developed in the LIP6, is a LGPL C++ library providing state-of-the-art implementations of graphical models for decision making, including Bayesian Networks, Markov Networks (Markov random fields), Influence Diagrams, Credal Networks and Probabilistic Relational Models thanks to the o3prm syntax[2]. A custom Java wrapper of aGrUM was implemented using SWIG [Beazley 1996].

[1] `httpl://agrum.gitab.io`
[2] `http://o3prm.gitlab.io/`

4.3 Discussion

By introducing new operators in the ODM syntax these studies allow the computation of probabilities of atomic events during rule evaluation. They have raised and answered some modelling problems related to such syntax through the use of PRMs and have proposed solutions to make inference scalable in such a system. However, two issues weren't addressed in the previous works :

- **Business user friendliness**: such rules can be difficult to define and to understand by a business user, expressing a probability on particular conditions requiring a deep level of knowledge of the probabilistic models used. This overly intricate mixture of the two models (rules and probabilistic) is a problem insofar as it is not the same individuals who would intervene on each of them.
- **Performance**: ODM provides the ability for users to define different types of conditions (for example, using filters, aggregators, and nested conditions). Neither these constructions, more complex, nor their impacts on the performances have been studied.

In the following chapters of this document, we will present some initial solutions to address these two challenges. First, we will introduce a more general and, hopefully, more accessible syntax, allowing business users to use probabilistic rules (Chapter 5). In a second time, we will examine the issue of scalability by proposing a change in the data representation within PGMs (Chapter 6).

Bibliography

Agli, H. (2017). *Uncertain reasoning for business rules*. PhD thesis, Université Pierre et Marie Curie - Paris VI.

Agli, H., Bonnard, P., Gonzales, C., and Wuillemin, P.-H. (2016). Incremental Junction Tree Inference. In *IPMU16*, Eindhoven, Netherlands.

Ait-Kaci, H. and Bonnard, P. (2011). Probabilistic production rules. Technical report, IBM France Lab.

Andersen, S. K., Olesen, K. G., Jensen, F. V., and Jensen, F. (1989). Hugin - a shell for building bayesian belief universes for expert systems. In Sridharan, N. S., editor, *IJCAI*, pages 1080–1085. Morgan Kaufmann.

Arru, M. (2011). Introduction of probabilistic reasoning in business rules managmenet systems. Master's thesis, Polytech Nantes.

Bangsø, O. and Wuillemin, P.-H. (2000). Object oriented bayesian networks a framework for topdown specification of large bayesian networks and repetitive structures.

Beazley, D. M. (1996). Swig: An easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, TCLTK'96, page 15, USA. USENIX Association.

Conrady, S. and Jouffe, L. (2015). *Bayesian Networks and BayesiaLab: A Practical Introduction for Researchers*. Bayesia USA.

Druzdzel, M. J. (1999). Smile: Structural modeling, inference, and learning engine and genie: A development environment for graphical decision-theoretic models. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, page 902–903, USA. American Association for Artificial Intelligence.

Gonzales, C., Torti, L., and Wuillemin, P.-H. (2017). aGrUM: a Graphical Universal Model framework. In *International Conference on Industrial Engineering, Other Applications of Applied Intelligent Systems*, Proceedings of the 30th International Conference on Industrial Engineering, Other Applications of Applied Intelligent Systems, Arras, France.

Koller, D. and Pfeffer, A. (1998). Probabilistic frame-based systems. In *AAAI/IAAI*, pages 580–587.

Matsumoto, S., Carvalho, R., Ladeira, M., Costa, P., Santos, L., Silva, D., Onishi, M., Machado, E., and Cai, K. (2011). *UnBBayes: a Java Framework for Probabilistic Models in AI*.

Mekhnacha, K., Smail, L., Ahuactzin, J.-M., Bessière, P., and Mazer, E. (2006). A unifying framework for exact and approximate Bayesian inference. Research Report RR-5797, INRIA.

Perez, K. (2013). Probabilistic production rules. Master's thesis, Ecole des Mines de Saint-Etienne.

Pfeffer, A. (2000). *Probabilistic reasoning for complex systems*. PhD thesis, Stanford University.

PART B

Towards Effective Probabilistic Rules

Chapter 5

A New Syntax For Probabilistic Production Rules

Contents

In this chapter we present a new syntax for a sub-part of the Ilog Rule Language (Section 5.1). We describe the different rewriting procedures used when compiling the rules alongside with the generation of a PRM (Section 5.2) as well as a first method to use this syntax in a context where execution time and memory management are fundamental (Section 5.4).

5.1 Business-User Friendliness and Risk Definition

Previous works [Ait-Kaci and Bonnard 2011; Agli 2017] have shown that a loose coupling between a rule engine and probabilistic graphical models (Bayesian networks initially, then PRM) allowed reasoning and making decision with uncertain data. The introduction, in the ODM syntax, of several operators as well as a mechanism allowing, at runtime, the communication between the rule engine and the probabilistic one allows the use of a new type of reasoning and thus enhances the reasoning capabilities of BR.

However, a number of problems have been raised with those approaches [Ducamp et al. 2018], mainly related to accessibility issues for a business user, the proposed syntaxes requiring a deep understanding of the probabilistic model used. This complexity can be a hindrance to the use of these reasoning capabilities by business users who are more interested in decision making than in formulating probabilistic queries, they may be confused if they have to deal directly with programming components or mathematical concepts.

To address this *business user friendliness* issue, we have redefined the treatment of uncertainty in the expression of rules by replacing the probability thresholds attached to single variables by an aggregated notion of acceptable risk on the evaluation of the conditions of the rule as a whole, as shown in the Figure 5.1.

In this example, if we think ($probability > 0.8$) that a patient in the waiting room of an hospital is a high-risk case ($p.is\_severe == true$) and a physician is not overloaded with severe patients ($p.nbr\_of\_severe\_patients < 2$), then the physician will be assigned to care for the patient.

```
rule PatientAssignement {
  when {
    hp : Hospital();
    p : Patient(p.is_severe == true) in hp.triage;
    phy : Physician(phy.nbr_of_severe_patients < 2) in hp.physicians;
  } [with probability > 0.8] then { // Assign him the patient }
};
```

Figure 5.1: A new syntax for probabilistic rule

➢ Threshold interpretation

This threshold, which conditions the eligibility of a rule, characterizes the probability that a set of conditions are verified given the state of the working memory (*i.e.* , the set of facts/evidence known about the monitored system). Given the following formulation over n conditions, we will check if $p(rule)$ is greater than the fixed threshold.

$$p(rule) = p(c_1, ..., c_n | WM)$$

The probability operator can be seen as a generalization of the Boolean case, for which rule actions are executed when the conjunction of the rule conditions is satisfied. In this case, the threshold is simply equal to 1 (or omitted) to express the fact that we are certain about the conditions.

➢ Threshold parametrization

The parametrization of the threshold depends on the business policies and risk aversion of users. In case of high health tension a hospital department might decide to take less risk on accepting people who could actually stay at home. Conversely, when an institute has many beds and staff available it might take the risk of accepting people whose condition is more uncertain. If we consider the cases where the rule is executed (not executed) correctly, denoted TP (resp. TN) and those where it is executed (resp. not executed) wrongly, FP (resp. FN) as well as the gains g_X or costs c_X associated with each of these cases, then we can see the choice of the threshold as an

optimization problem. One could, for example, seek to fix the threshold so as to maximize his gains (find x such as $max(p_x * g_{TP} + (1 - p_x) * g_{TN})$) or to minimize his losses ($min(p_x * c_{FP} + (1 - p_x) * c_{FN})$). Some might even use a combination of the latter, to establish a trade-off between gains and losses. These very interesting aspects, however, are not part of the scope of this thesis.

➢ Probabilistic query formulation

To be able to evaluate the threshold one have to formulate a probabilistic query over the condition part. To achieve this, it is necessary to generate the PRM not only from the defined object model, as in [Arru 2011; Agli 2017], but also from the set of rules. Since the rule engine does not know how to interpret a probability over a set of conditions it is necessary to change its toolchain (compilation phase) and intervene during the rewriting phase to make such rules usable.

As we will see in this chapter, we indirectly propose a link between PRMs and first-order logic. If, as we said in the introduction, other models are specifically made for this purpose (such as MLN [Richardson and Domingos 2006]), the ease of manipulation/modification of PRMs and their interpretability make them a good candidate for our business-oriented use.

5.2 Rule Rewriting and PRM Enhancement

5.2.1 A Structural Approach

As we have seen in the Section 2.2.2, the condition part of a rule consists of a set of unitary conditions which can take multiple forms, that of a simple condition, an existential condition or an aggregator, each of them with predicates to help filter the objects they concern (a physician with a certain specialty, the patients with a particular age within a waiting room, ...), as showcased in Figure 5.2. These objects can either be found in the WM or be declared as ruleset parameters but we only consider the first case in this document as it is the most general one. Let us recall that Rete works by looking incrementally for the elements of the WM verifying the conditions.

```
rule A {
    when {
        ...
        p : Patient(predicate);
        ...
    } [with probability > x] then { // test and alert }
};
```

Figure 5.2: A rule with our new syntax

Each condition corresponds to a mapping between a variable and elements present in the WM. These elements can be calculated from the WM for constituting an aggregation of them, as we will see in Section 5.2.4.1, or conditioned by a predicate which is composed of a set of atomic propositions connected to each other by logical operators. In this work we will consider two of them, the conjunction and the disjunction.

A first "naïve" approach to handle the probability could be to encode all predicates in a new class within the generated PRM, to use evidence within this model to characterize the values of the deterministic attributes of the found elements and query the posterior of the conjunction of those predicates at the end of the condition part, as showcased in Figure 5.3 and 5.4 (for the

sake of readability we will not display the PRM in its entirety, only the classes and random variables concerned by the rule, the other variables/dependencies being dashed). The exact formulation of such a request will be described in the Section 5.2.3.

```
rule A {
    when {
        ...
        p : Patient();
        ...
        // evaluation of probability(rule) > x;
    } then { // test and alert }
};
```

Figure 5.3: Naïve rewriting of rule A

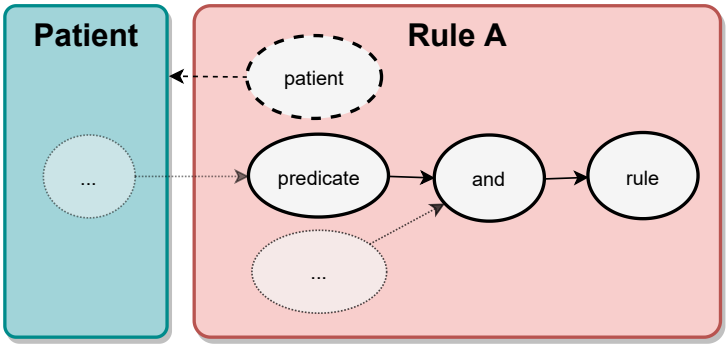


Figure 5.4: New class with the naïve approach

This method would not be flawless, not using predicates within rule would be equivalent to not filtering the elements prior to the inference and thus greatly increasing the number of probabilistic calculations. Probabilistic attributes lie in the Bayesian domain while deterministic ones relate to logic, for which rule engines are optimized. Even if a probabilistic engine could evaluate all types of conditions over probabilistic and deterministic attributes (which could be characterized by CPTs filled with zeros and ones), it would be at the price of highly degraded performances (the calculation of an inference is rarely insignificant in terms of memory/time costs, so we try to minimize their number as much as possible).

5.2.1.1 Rewriting Procedure and Compilation Process

A more effective rewriting may be considered. Since we know which attributes are probabilistic in predicates (they are annotated with @PRMattribute in the Data model, as described in the Figure 4.2), we can use rewriting rules that allow us to encode only what is strictly necessary in the PRM and keep as much filter as possible within the condition to limit the number of element on which we will query the final probability. A brief description of the rewriting process we have chosen to implement as a rule engine plugin is the following :

Rewriting procedure

foreach *probabilistic rules* **do**
1. Create a class in the PRM representing the rule;
2. **foreach** *probabilistic condition* **do**
   a. Enhance the PRM w.r.t. the condition;
   b. Rewrite the condition;
3. Add a call to the probabilistic engine;

This procedure allows us to redefine the toolchain of the ODM rule engine. Contrarily to previous proposals, where the PRM model was only generated using the annotated user model, we will subsequently enhance it with elements extracted from the probabilistic rules, reinforcing the coupling between the two engines. Our plugin, called PRIME (for *Probabilistic Reasoning Insight ModulE*), operates such rewriting procedure and supports a part of the technical language of the engine (extending the BAL-IRL conversion tool would be possible but would be out of the scope of the thesis). The action of the plugin on the toolchain is schematized on Figure 5.5.

The different possible cases supported by our prototype are described in the following sections of the document. We will first address the treatment of simple conditions that act as mapping between a variable and an object from the WM regarding some filters (Section 5.2.2), then describe how probabilistic queries are formulated (Section 5.2.3) to finally consider more complex cases involving aggregators or existential conditions (Section 5.2.4).

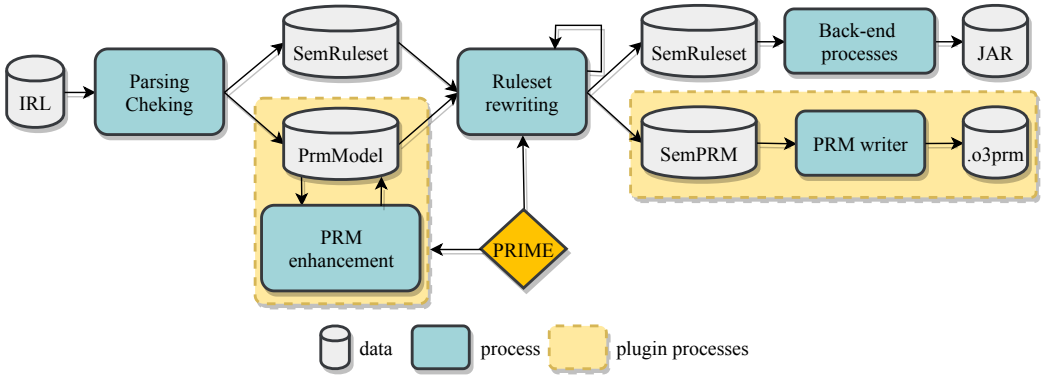


Figure 5.5: PRIME new compilation process

The first step during the rewriting of a probabilistic rule consists in adding a new class in the PRM model, where we will encode the different references to the objects as well as probabilistic conditions encountered. It is important to note that it is the plasticity of PRMs, inherited from the object-oriented paradigm, that allows us to easily adapt and enrich them without having to modify the base model. We can, for example, easily add new classes and references towards the base model and quickly instantiate many objects of such classes within the PRM system. Such manipulations would have been tedious —if not impossible— in a classical BN.

5.2.2 Condition Rewriting

Each type of condition can involve a predicate, simple or complex, which will be used to select (or not) an element of working memory according to its attributes. It is at this level that the notion of probability will propagate and therefore that our rewriting will intervene. During the process of rewriting probabilistic rules we will sequentially go through each condition. Our goal, by rewriting the conditions, is to keep, when possible, the deterministic elements within the logical part of the evaluation (managed by the rule engine), and to transfer the others to new variables of the PRM that will be taken into account by the Bayesian engine. The different cases that can be encountered and the corresponding actions required in these are outlined in the Table 5.1.

Predicate type	Example	Action
Deterministic	is unconscious	Don't rewrite
Probabilistic	status is severe	Add in PRM
Conjunction	is unconscious and in a severe condition	Decompose
Disjunction	is unconscious or in a severe condition	Add in PRM
Complex	tested positive or (severe and unconscious)	Decompose

Table 5.1: Types of predicates used within IRL conditions

In the next pages we will describe precisely the rewriting rules for each of the constructs presented in 2.2.2 and how PRIME modifies the PRM during the *PRM enhancement process*. If the treatment of the so-called simple types are only particular cases of the complex version, we thought it would be useful to describe the mechanisms in these cases for illustrative purposes but also because we believe that these cases would be the most commonly used. Unless stated otherwise, a predicate without any comparison operator implies that this predicate is equal to true (e.g., p.is_severe is to be interpreted as p.is_severe == True). In the case of a purely deterministic condition, no additional rewriting is necessary, the rule engine knows how to interpret it as it stands.

5.2.2.1 Probabilistic Predicate

When a predicate involves a random variable, as with *p.is_severe* in Figure 5.6, the rule engine cannot interpret it, it is up to the probabilistic engine to do so.

```
rule B {
    when {
        ...
        p : Patient(p.is_severe);
        ...
    } [with probability >= x] then {
        // test and alert
    }
};
```

Figure 5.6: Condition involving a probabilistic predicate

The predicate is, consequently, added to the PRM class concerning the rule, as shown in Figure 5.7, along with a reference *patient* linking to an object of the Patient class. This reference is used to link, at runtime, a patient instance filtered by the rule engine and its reference within the Rule B instance in the PRM system (as described in Section 5.2.3). The predicate is then directly linked to a logical variable, the *and* node, that acts as a conjunction between the different conditions encoded in the class (since this variable only takes binary variables as parents, its CPT can be easily generated).

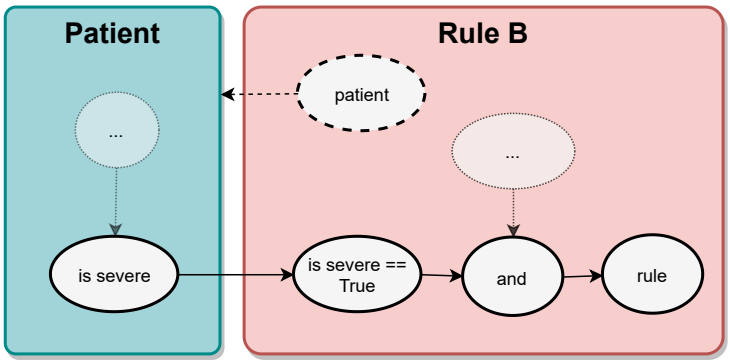


Figure 5.7: Enhanced PRM from the analysis of rule B

Once the PRM is completed, the predicate is removed from the original condition of the rule, *i.e.* it will not no longer be managed by the rule engine. In our previous example, the rule engine will, therefore, only search for patients in the working memory (according to the previous conditions). It will be up to the probabilistic engine to evaluate the probability of the random variable *rule* at the end of the condition part, as shown in Figure 5.8. We will describe in Section 5.2.3 the context of such an evaluation as well as the different parameters that it must take in order to properly compute the probability of the rule.

```
rule B {
  when {
    ...
    p : Patient();
    ...
    evaluate(PRMengine.getPosterior("Rule B",
                                    {patient:p;...},
                                    {...}) > x);
  } then {
    // test and alert
  }
};
```

Figure 5.8: Rewritten form of rule B

5.2.2.2 Conjunctive Predicate

When the condition is a conjunction of two predicates, one of which is probabilistic, it is possible to keep part of the predicate within the rule (in the example in Figure 5.9, the deterministic predicate *is_conscious == False*). Keeping as much deterministic information as possible limits the elements that will be concerned by the probabilistic inference and may improve the global performance.

```
rule C {
    when {
        ...
        p : Patient(p.is_severe and p.is_conscious == False);
        ...
    } [with probability >= x] then {
        // test and alert
    }
};
```

Figure 5.9: Condition involving a conjunctive predicate

The probabilistic component of the conjunction is added in the PRM in a similar fashion as before, as showcased in the Figure 5.10.

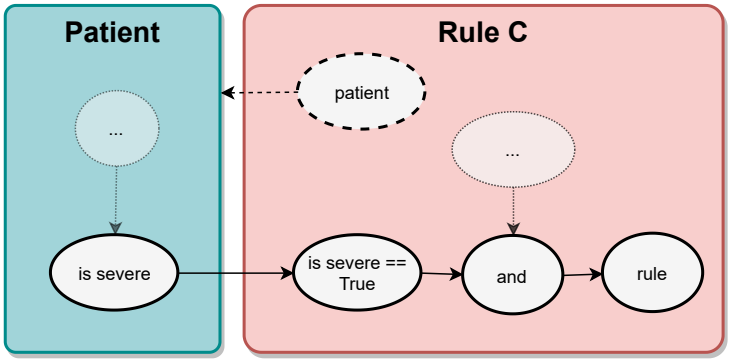


Figure 5.10: Enhanced PRM from the analysis of rule C

The interest of such a rewriting is noticeable in the example showcased in Figure 5.11: if there is no object in the working memory that checks the deterministic predicate (if no patient were unconscious) we would not need to verify the rest of the rule, which limits the use of calls to the probabilistic engine at runtime. Such savings wouldn't be possible with the naïve approach described above.

```
rule C {
  when {
    ...
    p : Patient(p.is_conscious);
    ...
    evaluate(PRMengine.getPosterior("Rule C",
                                    {patient:p,...},
                                    {...}) > x);
  } then {
    // test and alert
  }
};
```

Figure 5.11: Rewritten form of rule C

5.2.2.3 Disjunctive Predicate

In the case of a disjunction between a probabilistic element and a deterministic one (as in the Figure 5.12), however, the deterministic element cannot be kept within the condition, as shown in Figure 5.14.

```
rule D {
    when {
        ...
        p : Patient(p.is_severe or p.is_conscious == False);
        ...
    } [with probability >= x] then {
        // test and alert
    }
};
```

Figure 5.12: Condition involving a disjunctive predicate

It is not possible to filter successively the elements that verify the deterministic part of the predicate ($p.is\_conscious == False$) and then, in this subset, the elements that verify the probabilistic part ($p.is\_severe$). It is clear, in our example, that the set of patients severely affected is not necessarily a subset of the patients who are unconscious. The deterministic predicate is therefore also added in the PRM and linked to the probabilistic predicate by an automatically generated random variable *or* as showcased in the example in Figure 5.13.

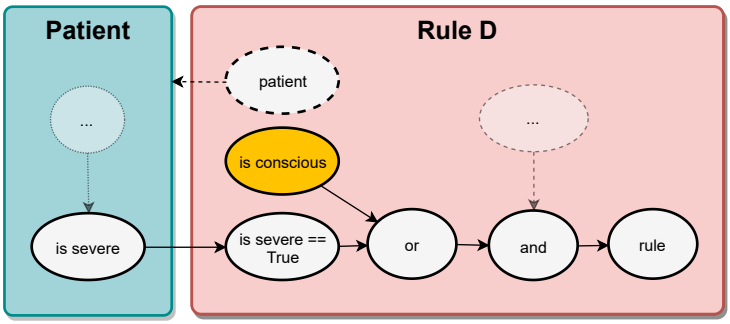


Figure 5.13: Enhanced PRM from the analysis of rule D

At runtime, an evidence on the truth value of the deterministic variable is added depending on the elements filtered by the rule engine, as shown in Figure 5.14.

```
rule D {
  when {
    ...
    p : Patient();
    ...
    evaluate(PRMengine.getPosterior("Rule D", {patient:p,...},
                                    {is_conscious:p.is_conscious,...})
            > x);
  } then { // test and alert }
};
```

Figure 5.14: Rewritten form of rule D

In some cases, we could have a more sophisticated mechanism that would start by checking the value deterministic predicate, since it is inherently independent from the probabilistic one if it is true we would not need to check the rest of the condition. In the case of our rule we would therefore avoid calculating the probability that a patient is in a severe state if we already know that he is unconscious. If other conditions require evaluation, the Bayesian network could be pruned to remove the disjunction in question from the *and* aggregator. However, the implementation of such an optimization is not trivial since it would require to be able, on the rule engine side, to switch between different types of probabilistic queries (with or without pruning) at runtime. If such an optimization should work in simple cases it is no guarantee that it will work in the general, more complex, case that we will now present.

5.2.2.4 Complex Predicate Rewriting

More generally, a predicate can involve numerous logical operators linking attributes in an arbitrarily complicated manner. Let us consider the condition in Figure 5.15, where the predicates d_1, d_2, d_3 are deterministic and p_1 is probabilistic. We recall that our plugin supports the rewriting of binary *and* and *or* that we assume to be always commutative. If other operations are possible (notably with n-ary operators), we have restricted ourselves to the most classical —not to say basic— ones, the idea being to iteratively enrich the capabilities of our plugin in the future.

```
rule E {
    when {
        ...
        c : ClassC(d1 and ((p1 or d2) and d3));
        ...
    } [with probability > x] then {
        // test and alert
    }
};
```

Figure 5.15: Complex condition

To ease their reading and their manipulation, such conditions can be shaped in the form of a tree that we call *predicate tree*.

Definition 5.1

Predicate Tree *A predicate tree is a structure encoding a complex condition as a binary tree where the root and internal nodes are logical operators (and/or) and leaves are predicates.*

The condition shown in the Figure 5.15 can be represented with the following tree:

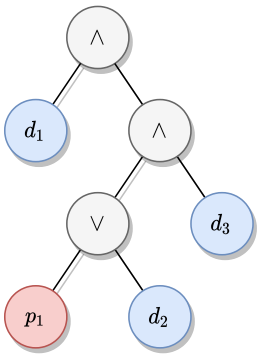


Figure 5.16: Predicate tree based on the condition in rule E

Proposition 1

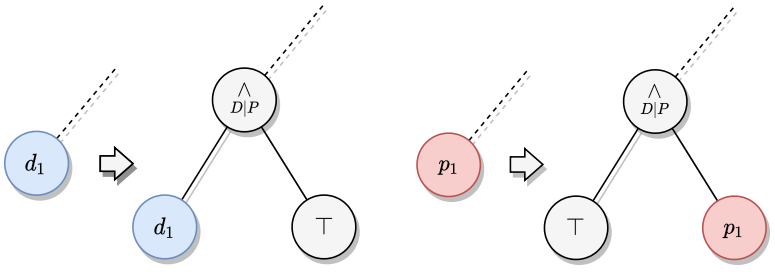
Any complex condition can be decomposed into a conjunction of two sub-conditions, a purely deterministic one that will be used to filter the elements concerned by the probabilistic query encoded in the second one.

To operate such a transformation we have defined a procedure T which, given a predicate tree τ will produce a new predicate tree τ' whose root will be a binary operator denoted $\underset{\text{D|P}}{\wedge}$, which acts like a classical logic $\wedge$ but constrains its left hand operand, ${\tau'}_l$, to contain only deterministic elements while the right hand one, ${\tau'}_r$, contains all the probabilistic elements. We recall that the main goal is to isolate deterministic terms in order for the rule engine to evaluate them first and, if necessary, request for the evaluation of probabilistic ones. Before describing the transformation process in more detail (Algorithm 2), we will illustrate the different possible rewriting scenarios in the sub-language that we support.

➢ Leaves transformation

When the rewriting procedure concerns a leaf f, it will be transformed so as to be placed on either side of a new predicate tree according to its nature (on the left side if deterministic, right if probabilistic), as shown in the Figure 5.17. The operator at the root of this tree, a $\underset{\text{D|P}}{\wedge}$, will be associated to its neutral element, denoted $\top$, to preserve the value associated with the evaluation of this tree.

$$\text{T}(f) = \begin{cases} \top \underset{\text{D|P}}{\wedge} f & \text{if } f \text{ is a probabilistic attribute} \\ f \underset{\text{D|P}}{\wedge} \top & \text{otherwise} \end{cases}$$



(a) (b)

Figure 5.17: (a) T applied on a deterministic attribute. (b) T applied on a probabilistic one.

➢ Conjunction rewriting

Our second case describes the rewriting process of a conjunction. Once the operands of the tree are themselves transformed (and thus described using $\underset{\text{D|P}}{\wedge}$), the procedure transfers the deterministic (resp. probabilistic) component of the new operands to the left (resp. right) of the conjunction, as depicted in the Figure 5.18.

$$\mathrm{T}(l \wedge r) = (\mathrm{T}(l)_l \wedge \mathrm{T}(r)_l) \underset{\text{D|P}}{\wedge} (\mathrm{T}(l)_r \wedge \mathrm{T}(r)_r)$$

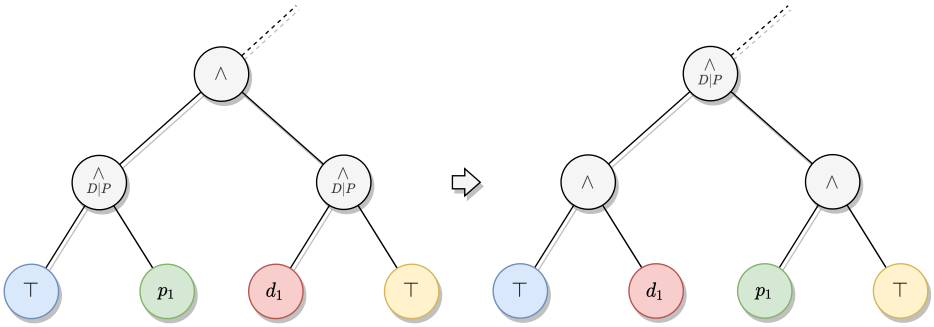
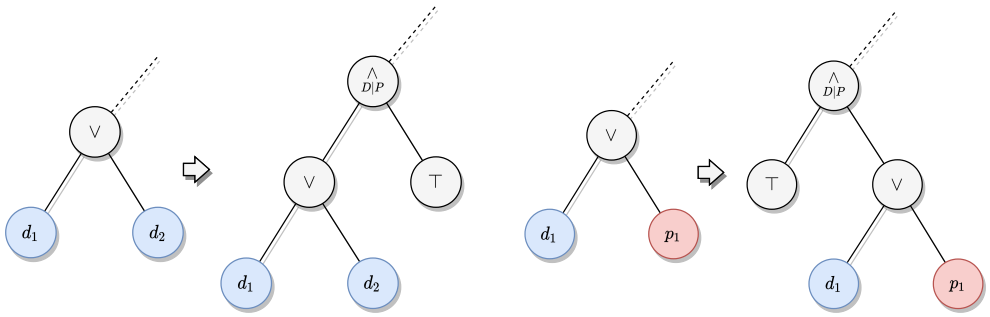


Figure 5.18: τ applied on a conjunction

➢ Disjunction rewriting

When we seek to transform the disjunction of two predicate trees, l and r, themselves already transformed, we must verify if their probabilistic parts are both equal to $\top$ (*i.e.* , if $\mathrm{T}(l)_r = \top$ and $\mathrm{T}(r)_r = \top$), if and only if this is the case then we will switch the whole disjunction into the left part of a new tree whose operator will be $\underset{\text{D|P}}{\wedge}$ and the right part $\top$, as shown in the Figure 5.19a. If not, the disjunction will be encoded to the right of the new tree, as shown in the Figure 5.19b.

$$\mathrm{T}(l \vee r) = \begin{cases} \top \underset{\text{D|P}}{\wedge} (\mathrm{T}(l) \vee \mathrm{T}(r)) & \text{if } \mathrm{T}(l)_r \text{ or } \mathrm{T}(r)_r \text{ is different from } \top \\ (\mathrm{T}(l) \vee \mathrm{T}(r)) \underset{\text{D|P}}{\wedge} \top & \text{otherwise} \end{cases}$$



(a) (b)

Figure 5.19: (a) T applied on a purely deterministic disjunction. (b) T applied on a probabilistic one.

Transform T(τ) - Predicate tree transformation procedure

Input : a predicate tree τ
Output : a predicate tree
if τ *is a leave* **then**
  if τ *is deterministic* **then**
    return $\tau \underset{\text{D|P}}{\wedge} \top$;
  else
    return $\top \underset{\text{D|P}}{\wedge} \tau$;
else
  Set l as τ's left operand;
  Set r as τ's right operand;
  $l' \leftarrow$ **Transform**(l);
  $r' \leftarrow$ **Transform**(r);
  if τ *is a conjunction* **then**
    return $(l'_l \wedge r'_l) \underset{\text{D|P}}{\wedge} (l'_r \wedge r'_r)$;
  if τ *is a disjunction* **then**
    if $l'_r == \top$ *and* $r'_r == \top$ **then**
      return $(l' \vee r') \underset{\text{D|P}}{\wedge} \top$;
    else
      return $\top \underset{\text{D|P}}{\wedge} (l' \vee r')$;

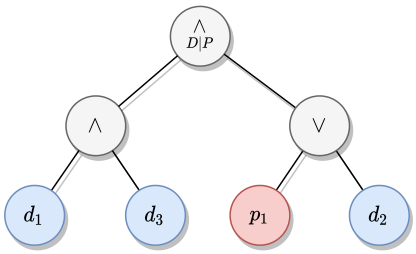


Figure 5.20: Rewritten predicate tree of condition in rule E

As illustrated in the Figure 5.20, using such procedure on the predicate tree allows us to move as much deterministic predicates as possible to the left side of the tree. Once the rewritten predicate tree has been simplified (in order to make the neutral elements disappear), the left branch will be used in the condition to filter in the matching elements ("d_1 and d_3" in our example), the right part will be encoded in the PRM ("p_1 or d_2"), as shown in Figure 5.21. If there are deterministic elements in the right hand part of the tree, their values will be used as evidence during inference.

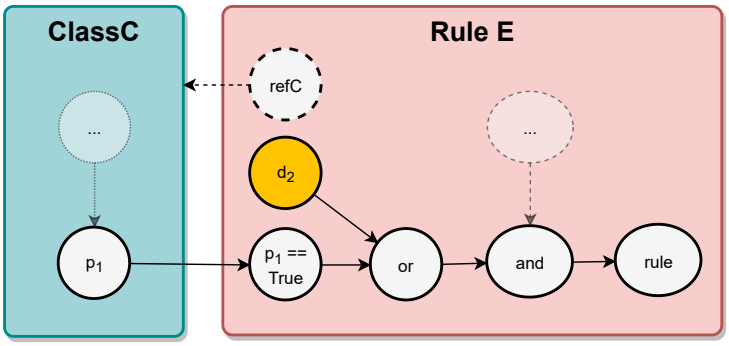


Figure 5.21: Enhanced PRM based on the analysis of rule E

We will now describe how the probabilistic query is formulated at the end of the condition part.

5.2.3 Probabilistic Query

Once the conditions of a probabilistic rule have been rewritten according to their nature, the mechanism allowing the probabilistic engine to be asked for the requested value must be formulated. In order to do so, we rely on the use

```
rule E {
  when {
    ...
    c : ClassC(d1 and d3);
    ...
    evaluate(PRMengine.getPosterior("Rule E", {...,refC:c,...},
                              {...,d2:c.d2,...}) > x);
  } then { // test and alert }
};
```

Figure 5.22: Rewritten version of rule E

of the evaluation function included in the language and presented in Section 2.2.2.6. This function will compare the result of a call to the probabilistic inference engine, using the method *PRMengine.getPosterior()*, and the value chosen to trigger the rule. Several parameters are used in this method, as shown in Figure 5.23, mainly to specify to the probabilistic engine the context in which the PRM should be instantiated:

- The name of the PRM class corresponding to the current rule to be instantiated (e.g., Rule D),
- The context of object variables to be used in the probabilistic engine as a mapping between the elements identified in the WM and the names of their references slots in the PRM class (e.g., *p* in the rule and *patient*),
- A mapping between the deterministic elements necessary to characterize random variable within the PRM class and their values according to the WM (e.g., the value of *p.is_conscious* for the patient *p*).

```
evaluate(  PRMengine.getPosterior(  Rule D,{patient : p},
                                    {is_conscious : p.is_conscious})
           >                        threshold );
```

Figure 5.23: Example of probabilistic query within a rewritten rule

5.2.4 Complex Condition Rewriting

As discussed in the Section 2.2.2.5, more complex forms of conditions exist in IRL. If these complex constructions are widely used, they have not been discussed in previous works. We distinguish existential ones (*exists* or *not*) that verifies conditions without binding variables and aggregation conditions that perform calculations on sets of values in order to return one (an average age, for example). We will begin this section by describing the process of rewriting the latter.

5.2.4.1 Aggregate Condition

Rewriting probabilistic aggregators is inherently more challenging than ordinary conditions. As mentioned in section 2.2.2, we can distinguish three parts within an aggregator :

1. A generative part, which will look for elements in the working memory given a set of conditions,

2. An aggregation function, which will be applied to these elements or their attributes,

3. And optionally a filter on the aggregation result

However we will only consider the cases of probabilistic generative part (1) as well as of probabilistic functions that are used with filters on their results (2 and 3). Unfiltered aggregators are generally used within the action part of rules to update parameters in the working memory or, more generally, to perform an action or treatment on it (*e.g.* , sending a notification to a list of customers). Without a filter acting as a predicate upon the result, a probabilistic function wouldn't be useful to evaluate the probability of the conditions.

We will now describe the different mechanisms that allow us to manage such interesting formulations, as these functions can be seen as "rules within rules" whose complexity of evaluation will represent the main challenge for a large scale use of probabilistic rules.

➢ Probabilistic generative function

When the generative function —which defines the nature of the elements matched in the working memory— involves probabilistic predicates, it is necessary, as illustrated in the rule in Figure 5.24, to add a new probabilistic threshold on it as we are going to filter elements locally. This probability is to be interpreted in the same way as the global probability defined on the condition part of the rule. In the rule below, we are going to apply the actions that if we think, among other things, to find patients whose severity is -almost- certain awaiting treatment in the triage of a hospital. If other conditions involve probabilistic elements, it is still necessary to define a threshold on the rule (*probability* > *.8* in the example below).

```
rule F {
    when {
        hp : Hospital();
        agg : aggregate {
            p: Patient(p.is_severe) in hp.triage;
        } [with probability > .9] do {
            agg = new ArrayList<Patient>{p};
        }
        ...
    } [with probability > .8] then { ... }
};
```

Figure 5.24: Rule with a probabilistic generative function

If this formulation of nested probabilities complicates the definition and interpretation of the rule, we have nevertheless deemed it necessary to propose a rewriting procedure. This rewriting is done in the same way as for simple conditions except that we will no longer encode the conditions in the class of the rule but in a new class specific to the aggregator, as shown in Figure 5.25 resulting from the enhancement of the PRM. Naturally, if other probabilistic conditions exist outside the aggregator, they will be rewritten according to their nature.

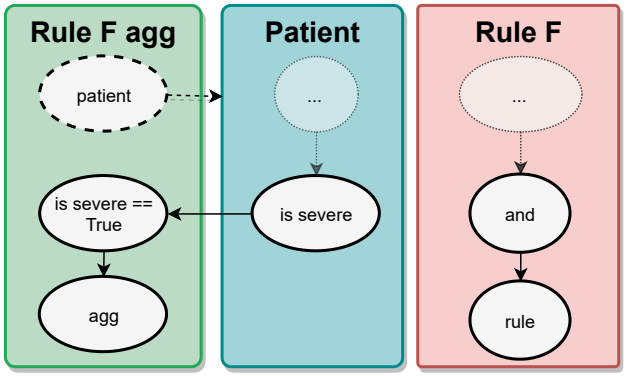


Figure 5.25: Enhanced PRM from the analysis of rule F

```
rule F {
    when {
        hp : Hospital();
        agg : aggregate {
            p: Patient() in hp.triage;
            evaluate(PRMengine.getPosterior("Rule F agg",
                                            {patient : p},
                                            {}) > .9);
        } do {
            agg = new ArrayList<Patient>{p};
        }
        ...
        evaluate(PRMengine.getPosterior("Rule F",{...},{...})
                > .8);
    } then { ... }
};
```

Figure 5.26: Rewritten version of rule F

At runtime, and for each set of elements verifying the deterministic conditions within the generative part, an intermediate call to the probabilistic engine will be made, as described in the Figure 5.27. It is important to note the multiplication of calls to the probabilistic inference engine introduces issues regarding time performance that could be a bottleneck preventing the use of such a syntax: in addition to being less obvious to interpret, these formulations multiplies the number of inferences and make the rule more complex to be evaluated.

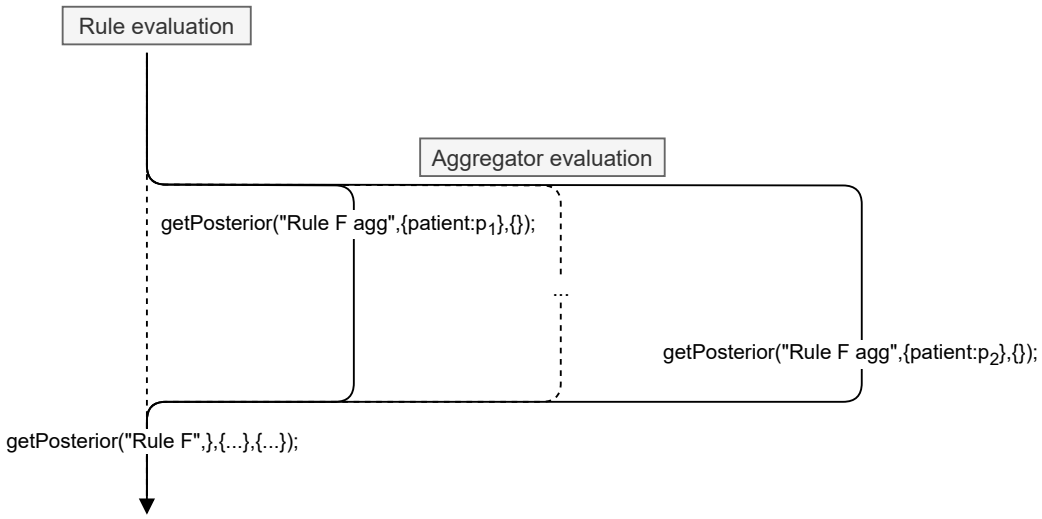


Figure 5.27: Runtime calls to the probabilistic engine

➢ Filtered probabilistic aggregation function

An aggregation function can be described as function that takes an unbounded number of values as parameters and returns only one (a more formal definition will be given in Section 5.4). ODM proposes to use some classical functions in its rules, they will be applied on the elements filtered by the generative part of an aggregator. The aggregation functions available in ODM are described in the table below which indicates which ones are also available within aGrUM.

Function	aGrUM
Exists	yes
Not	yes
Sum	yes
Count	yes
Median	yes
Mean	no
Min/Max	yes
List/Multiset	no

Table 5.2: Main aggregation functions supported by ODM and their existence in aGrUM

Some aggregators do not exist *as is* in aGrUM, the aggregation function *exists* requires to set its modality, when it is equal to *true* it is to be interpreted as ODM's *exists*, when it is equal to *false*, it is equivalent to ODM's *not*. If aggregators in PRM take an unbounded number of parents it is however necessary to set their output domains. This constraint of the PRM language comes in particular from the need for the direct descendants of an attribute to know its modalities in order to be able to define their CPTs, as described in the section 3.25. If this is naturally the case for exist/not (that are either true or false) or min/max (parametrized w.r.t input's domain), it is not the case for the *sum* which will have to be truncated. This limitation is, however, not a problem, since the results of the aggregators that we use are filtered, it is possible to use these filters during the generation of the PRM in order to characterize the domains of the aggregators.

When it comes to the functions that *aggregates* objects into lists or multisets, aGrUM doesn't support them. This kind of operation is not classical in probabilistic graphical models, firstly because their results are, by nature, unbounded, but also because the elements they would manipulate would not correspond to usual inputs for a random variable. Similarly, a *mean* aggregator isn't definable as it would require the output to be continuous.

If the result of an aggregation is not filtered but is used in other conditions our plugin raises an error. If this is not the case, we consider the aggregation to be neutral with respect to pattern matching, so it can be removed, as our plugin does not support the use of probability distribution in the action part of a rule (this could however be used as development work). It is interesting to note that business users do not always use the most efficient formulations for their conditions, some of them could, for example, generate lists and compute their cardinalities instead of using an aggregator *count*, therefore it would be appropriate to identify these formulations in order to allow our plugin to optimize them.

In the rule presented in Figure 5.28 we will try, among other things, to verify if more than 5 patients with severe complications are in the triage of a hospital. Since the aggregation function (*sum*) relates to a probabilistic attribute (*is_severe* of patient p), it will have to be rewritten by our plugin.

```
rule G {
    when {
        hp : Hospital();
        agg : aggregate {
            p: Patient() in hp.triage;
        } do {
            sum(p.is_severe)
        }  where { agg >= 5 }
        ...
    } [with probability > .9] then  { ... }
};
```

Figure 5.28: A rule using a probabilistic sum as an aggregation function and a filter on its results

If the aggregation function has no equivalent on the PRM side our plugin raises an error. Otherwise it is encoded into the rule class, as shown in Figure 5.29, along with a predicate over the value of *is_severe* for any p in the reference slot *patients* as well as the aggregation filter ($sum >= 5$).

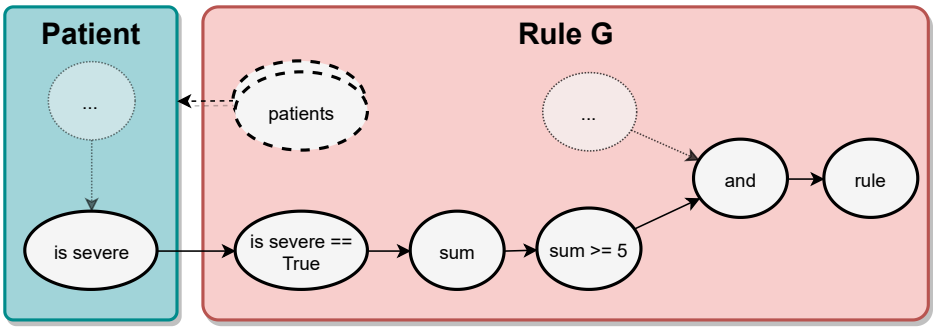


Figure 5.29: Enhanced PRM according to the compilation of rule G

In order to instantiate the reference slot *patients* in the PRM class of the rule above and be able to effectively calculate the return value of this aggregation function, it is necessary to inform the probabilistic engine of all the objects that will be concerned by the function (in our case, the list of patients in the triage). To this regard, the rule is rewritten in order to generate such a list using the instruction $agg = new\ ArrayList <Patient> \{p\_temp\}$, as shown in the Figure 5.30.

The rule engine is therefore no longer in charge of evaluating the aggregation function, but of constituting the list of elements on which the probabilistic engine will do so.

```
rule G {
    when {
        hp : Hospital();
        agg : aggregate {
            p: Patient() in hp.triage;
        } do {
            agg = new ArrayList<Patient>{p};
        }
        evaluate(PRMengine.getPosterior(Rule G,{patients:agg},{})
                 > .9);
    } then { ... }
};
```

Figure 5.30: Rewritten form of rule G

As we will see in the Section 5.4 the use of aggregators in out PRMs (and PGMs in general) is not without consequences on the performance of our algorithms. If the use of probability in the generative part of the rule multiplies the number of calls without necessarily making them more complex, the use of probabilistic aggregation functions can very quickly make inference - and thus its use within our rules - intractable.

5.2.4.2 Existential Condition

The keyword *exists* is used in the condition part of a rule to test whether any object in the working memory matches the condition. The usage of such keyword is similar to the one of a classical condition except that the condition does not discriminate which object was matched, you cannot bind an exists statement to an external variable. In the rule described in Figure 5.31, we are looking to determine if there is a conscious patient who cough and has lost his sense of smell or taste in the triage of a hospital with no SARS-infected cases. If this is the case, a testing and isolation procedure could be put in place to identify cases as early as possible and take the necessary measures in order to limit the spread of the virus.

```
rule H {
  when {
    hp : Hospital(hp.SARS_alert == False);
    exists Patient(is_conscious and loss_of_smell and cough)
                    in hp.triage;
  } [with probability >.9] then {
    // test and alert
  }
};
```

Figure 5.31: A rule with a probabilistic existential condition

To be able to rewrite a probabilistic existential condition it is necessary, as shown in Figure 5.33, to replace it by an aggregation function that will construct a list containing the elements of the working memory verifying the deterministic part of the condition. This process is needed in order to keep in memory the elements that will have to be tested in the PRM, as parents of the node *exists* as pictured in the Figure 5.32.

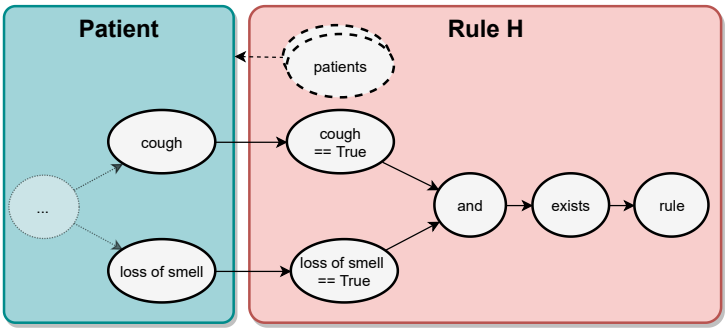


Figure 5.32: Enhanced PRM according to the compilation of rule H

In the new condition part of our rule, the set of patients found at a hospital triage (*p_temp: Patient(is_conscious) in hp.triage*) is added within a list (*tmp*). Since the predicate of the condition is complex (according to the definition given in section 5.2.2.4), its deterministic part is used to help limit the number of elements added in such a list. When adding the evaluation, the list will be used to instantiate the multiple references *patients* in the PRM class.

```
rule H {
    when {
        hp : Hospital(hp.SARS_alert == False);
        tmp : aggregate {
            p_temp = Patient(p_temp.is_conscious) in hp.triage;
        } do {
            tmp = new ArrayList<Patient>{p_temp};
        }
        evaluate(PRMengine.getPosterior("Rule H",
                                        {patients:tmp},
                                        {}) > .9);
    } then {
        // test and alert
    }
};
```

Figure 5.33: Rewritten form of rule H

An analogous reasoning is applied in the case of the *not* function except that the *exists* aggregator will be parametrized to *false*.

These different rewriting rules allow PRIME to extend the reasoning capabilities of the engine to a sub-language of instructions using a simple syntax that we believe to be easily understood by business users. The use of these rules is however not without impact on runtime performance.

5.3 Runtime Evaluation

When the condition part of one of our rules is evaluated and the deterministic conditions are all verified, then the probability encoded in the PRM must be calculated. In this section we will follow the different interactions between the two engines that allow such an evaluation. We recall that, in a general way and similarly to what was proposed in previous work (cf. Figure 4.4), the rule engine can require the calculation of a probability and indicate to the probabilistic engine the modifications of the WM that have an impact on the PRM system. If the interactions are the same, the mechanisms allowing the computation of the probabilities are different and we will present and illustrate the different steps here.

5.3.1 Probabilistic Query

In this section, we will illustrate the interaction mechanisms linking the two engines at runtime. We will follow the evaluation of the rule presented in Figure 5.34. In this rule, a doctor who doesn't have many patients will be assigned with a new one if we are almost certain that this new patient is thought to be in a serious state or unconscious.

```
rule I {
  when {
    hp : Hospital();
    p : Patient(p.is_severe || p.is_conscious == False) in hp.triage;
    phy : Physician(phy.nbr_of_patients < 3) in hp.physicians;
  } [with probability > 0.8] then {
      // Assign him the patient
  }
};
```

Figure 5.34: An example probabilistic rule

Figure 5.35 shows how this rule is rewritten by applying the procedures previously described in the manuscript. When the probabilistic engine is called and in addition to the name of the class to be instantiated, the rule engine communicates the different references to be instantiated as well as the value of deterministic attributes encoded within the class.

```
rule I {
  when {
    hp : Hospital();
    p : Patient() in hp.triage;
    phy : Physician(phy.nbr_of_patients < 3) in hp.physicians;
    evaluate(PRMengine.getPosterior("Rule I",
                                    {patient:p},
                                    {is_conscious:p.is_conscious})> .8)
    } then {// Assign him the patient }
};
```

Figure 5.35: Rewritten version of rule I

In the PRM system that we will track (cf. Figure 5.36), 2 doctors (phy_1 and phy_2) are taking care of 5 patients. 4 others, unassigned (p_6, p_7, p_8, p_9), correspond to patients waiting for triage at a hospital (if the class does not contain probabilistic elements it does not need to exist on the PRM representation of the system).

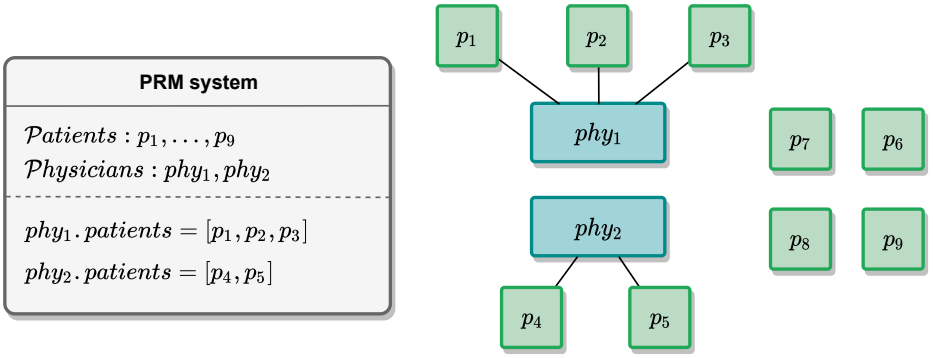
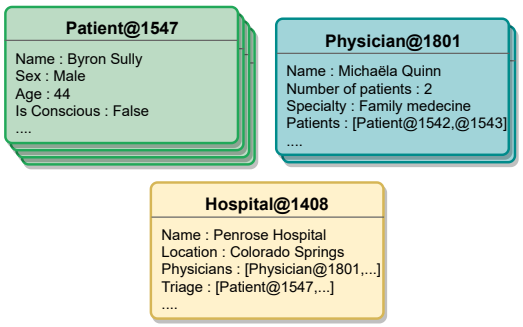


Figure 5.36: The PRM system of our example and its relational skeleton

In order to easily follow the evolution of both systems (WM and PRM) it is necessary to have a mapping between the elements they contain (the elements of the PRM system are a subset of the WM elements). The Table 5.37b illustrates such a mapping.



(a)

WM ids	PRM ids
Patient@1542	p_1
...	..
Patient@1547	p_6
...	..
Physician@1801	phy_1
Physician@1802	phy_2
Hospital@1408	doesn't exists here

(b)

Figure 5.37: (a) WM state. (b) Table of equivalence

➢ **Addition of rule instance and edition of references**

When the RETE looks through the rule and evaluates the conditions it iteratively forms tuples of objects which, collectively, verify them. The engine could for example contain, once arrived at the probability assessment of the rule 5.35, these elements from the WM :

- hp ← Hospital@1408
- p ← Patient@1547
- phy ← Physician@1802

When the probabilistic query is made using the method :

getPosterior("**Rule I**",{patient:p},{is_conscious:p.is_conscious}),

the PRM system is modified in order to instantiate an object of the class indicated in parameter, in our case an object of type **Rule I** (called r in the example in Figure 5.38). The second parameter indicates the correspondence between the variable in the rule and its reference in the PRM, the reference slot $r.patient$ must point to the object p matched by the RETE, *i.e.* , the Patient@1547 (identified by p_6 in the PRM system).

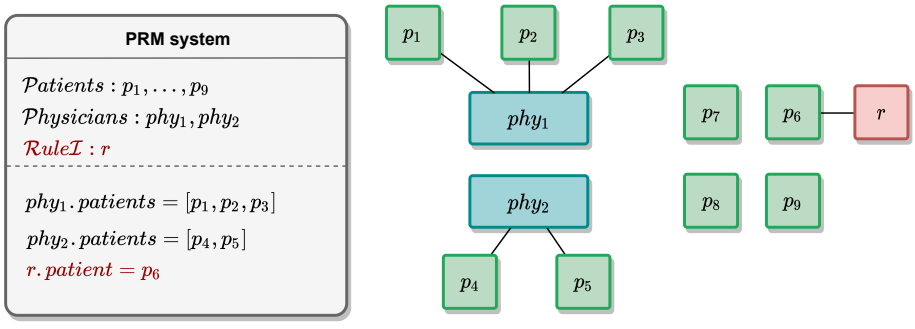


Figure 5.38: PRM system after adding the rule instance and the edition of references

➢ **Knowledge as evidence**

Once all the references of r have been edited, the question of the valuation of the deterministic predicates encoded in it arises. Since at this moment the patient p is known (Patient@1547) it is possible to add an evidence on the variable **is conscious**, in our case the rule engine will indicate to the probabilistic one that **is conscious == Patient@1547.is_conscious** (which is False, according to the figure 5.37a).

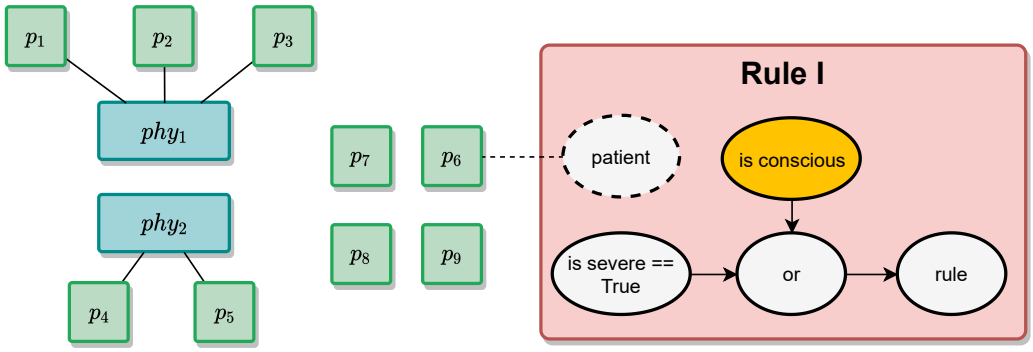


Figure 5.39: Addition of evidence in the PRM.

➢ **Grounded BN and inference**

When all the relevant information for the evaluation of r has been taken into account by the probabilistic engine, it is finally possible to ground the PRM (as described in the Section 3.4.1.2) in order to compute the probability of the variable *rule*, *i.e.* , $p(r.rule == True)$.

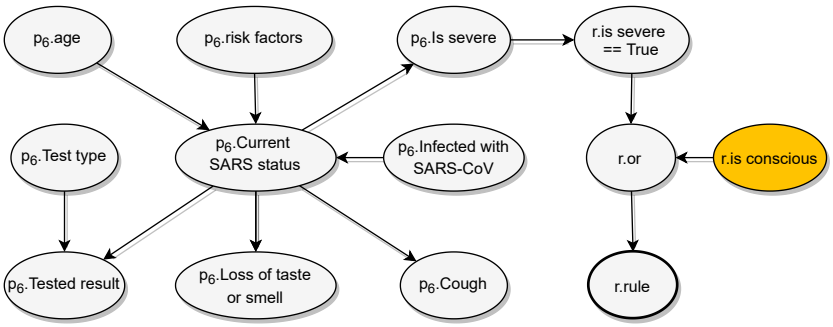


Figure 5.40: Grounded BN generated from 5.39

The result will then be compared to the threshold of the condition part (*e.g.* , 80% in the case of the rule 5.35). Once the inference is complete, any changes made to the PRM system are removed.

➢ **Side-effect and PRM update**

The action part of a rule can have repercussions on the state of the working memory, so it is necessary to give the rule engine mechanisms to indicate these changes to the inference engine, if needed. Lets recall that three kind of updates are possible :

- A structural change implying a modification of references in the PRM system (*e.g.* , the transfer of a patient from one physician to another).
- The addition/withdrawal of a new item in the WM (a patient arriving or leaving the hospital).
- The addition of evidence on a probabilistic attribute (the result of a PCR test or the age of a patient, for example)

In our example, if the probability associated with the patient Patient@1547 being in a severe state is greater than the threshold, he can be assigned to the physician Physician@1802, this information must also be taken into account within the PRM system (and therefore indicate that p_6 is in the list of the reference slot $phy_2.patients$, for example).

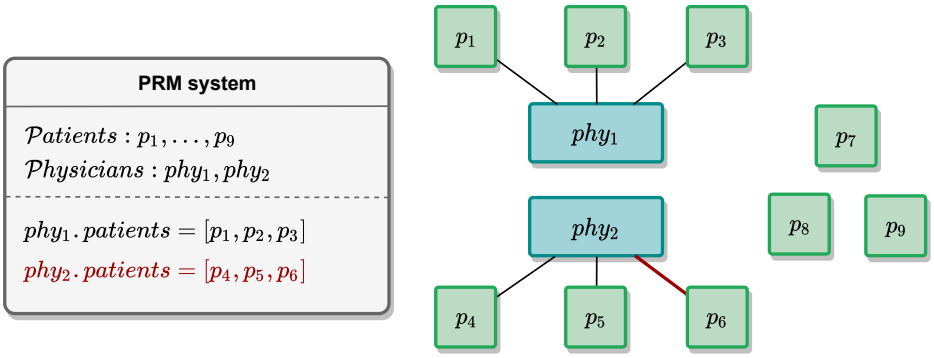


Figure 5.41: Evolution of the PRM system when p_6 is assigned to phy_2 in the WM

5.3.2 Inferences and Bottleneck

To summarize, when the rule engine reaches the end of rule evaluation, it asks the probabilistic engine to make an inference given a certain context. The result of such inference will be used to decide whether or not to activate the rule. If our rewriting procedure is done in a flexible and inexpensive way, the evaluation of probabilistic requests could, however, be a limiting factor for the use of such rules. These studies were presented during two doctoral consortiums of a conference focused on rules and logical reasoning : RuleML+RR [Ducamp et al. 2018; 2020a].

Beyond the significant increase in the number of calculations, it is mainly the use of aggregation functions in PGMs that could be problematic. In the rest of this chapter, we will present a first approach to overcome related to the use of aggregators when they verify certain mathematical properties.

5.4 Self Decomposable Aggregators : a Graphical Approach

We saw in Section 5.2.4.1.2 that our syntax offers users the possibility to instantiate probabilistic aggregators with many parents very easily. However, if the definition of aggregators over tens, hundreds of variables is made accessible thanks to their functional definition and the automatic mappings between WM and the PRM system, we will show that the evaluation of such models isn't possible. As it stands, the syntax is too powerful for our inference capabilities.

To illustrate such a use case -and the related combinatorial explosion- we will stay within the framework of hospital management but this time we will focus on the stock control of essential supplies. In the context of a SARS pandemic it is necessary, for example, for a hospital to regularly ensure that it has a sufficient stock of masks for medical workers. We will however assume that some stocks may be poorly identified (because of theft, loss or badly anticipated expiration dates), using a probabilistic graphical model

could help to better estimate stocks and coupled with rules, automatically avoid critical cases of shortages.

```
rule Stock Control {
    when {
        hp : Hospital();
        agg : aggregate {
            u: Unit() in hp.units;
            s: Stock(s.type=="N95 masks") in u.inventory;
        } do {
            sum(s.estimated_stocks)
        }  where { agg <= 50 }
    } [with probability > .7] then  { ... }
};
```

Figure 5.42: A rule using a probabilistic sum as an aggregation function and a filter on its results

In the example in Figure 5.42, we will go through all the stocks of N95 masks present in the inventories of the different units of a hospital in order to verify that the estimated stocks are above a certain critical threshold (the rewritten form of such rule is shown in Figure 5.44). We will simplify the problem by assuming that a stock is evaluated on a scale of 0 (none) to 10 (full stock).

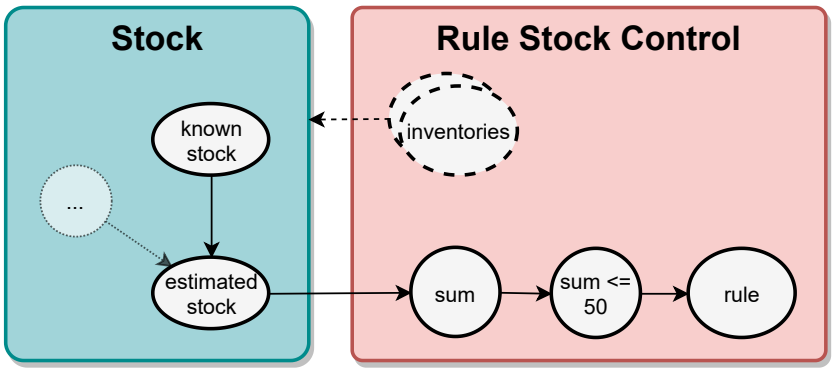


Figure 5.43: Enhanced PRM based on the analysis of rule Stock Control

```
rule Stock Control {
    when {
        hp : Hospital();
        agg : aggregate {
            u: Unit() in hp.units;
            s: Stock(s.type=="N95 masks") in u.inventory;
        } do {
            agg = new ArrayList<Patient>{p};
        }
        evaluate(PRMengine.getPosterior("Rule Stock Control",
                                        {stocks:agg},{})
                 > .7);
    } then { ... }
};
```

Figure 5.44: Rewritten form of rule Stock Control

In the case of the aggregator *sum* in the PRM class *Rule Stock Control* (showcased in the Figure 5.43), if we suppose that, during the evaluation, 7 units are in the reference slot *inventories* (with each variable *estimated_stock* characterized by 11 values), we will need to store $51 \cdot 11^7$ values to fully characterize *sum*'s CPT (its output value is in $[\![0,50]\!]$).

Table 5.3 shows the inference time using Lazy Propagation algorithm [Madsen and Jensen 1999] and generated file size (from the grounded BN), allowing us to have an idea of the space and time complexity of such cases, even for a small number of parents. Let's recall that we denote the set of x's parents as $\mathrm{Pa_x}$ and $|\mathrm{Pa_x}|$ the cardinality of such set.

$\lvert \mathrm{Pa}_{sum} \rvert$	**time**	**file size**
3	2.1e-3s	147Ko
4	8.7e-3s	1.5Mo
5	4.7e-2s	16.8Mo
6	1.35s	184.3Mo
7	17.0s	2.03Go
8	–	22.29Go

Table 5.3: Inference time to compute the probability of the rule Stock Control and size of the generated file, depending on $|\mathrm{Pa}_{sum}|$

These first results show that even for a limited number of parents, the exponential growth in the size of the CPT characterizing the aggregation function prevents our inference from being made. In our example, 8 parents are enough to exceed the memory capacity of our test machine (32go of RAM), if the model weighs "only" 22Go, the majority of this space is only used to characterize the potential associated with the aggregator. Such potential being multiplied with others during the initialization of an exact inference based on the use of a junction tree (as described in Section 3.3.3.1.3) to form cliques' own potentials, it is easy to identify the limits of such an approach not to mention the fact that these potentials can, for certain functions, be extremely sparse.

This section adapts and extends a part of the work presented in [Ducamp et al. 2020b] in which we were looking for a first solution to this application problem which may make it impossible for ODM users to define probabilistic rules with our syntax. In particular, the examples have been modified in order to make them more consistent this manuscript.

5.4.1 Aggregators and Combinatorial Explosion

As we suggested earlier, aggregation functions can be considered as a way to summarize information over a set of data. If the relevant literature lingers on their use in distributed data algorithms [Jesus et al. 2015] none are specific to the particular case of aggregation in PGMs. For probabilistic aggregation we need to provide, for each possible instantiation of the aggregated objects, a value in its CPT. Here, we provide more precise definitions, some (5.2 and 5.5) given or adapted from [Jesus et al. 2015], and consider that the process consists in the computation of an *aggregation function* defined by:

Definition 5.2

Aggregation function *An aggregation function f takes a multiset of elements from a domain I and produces an output of a domain O:*

$$f : \mathbb{N}^I \mapsto O$$

The use of multiset as input implies, among other things, that the order in which the elements are aggregated does not matter. A multiset M can be formally defined as a 2-tuple (A, m) where A is the set of distinct elements, also called *support* of M, and m is a function indicating, for each $x \in A$, the number of occurrences of x in M.

Definition 5.3

Probabilistic aggregation function *An aggregate function g is said to be a probabilistic aggregate function if :*

$$g : \mathbb{N}^I \times O \mapsto [0,1], \forall x \in \mathbb{N}^I, \sum_{y \in O} g(x,y) = 1$$

The aggregators that can be used in ODM have the particularity of having deterministic output values, so we can restrict our previous definition to a certain case of probabilistic aggregators whose CPTs will consequently be composed only of 0 and 1.

Definition 5.4

Deterministic aggregators *A probabilistic aggregate function $\mathcal{A}_f : \mathbb{N}^I \times O \mapsto [0,1]$ is a deterministic aggregate if it exists a function $f : \mathbb{N}^I \to O$ such that :*

$$\mathcal{A}_f(y, x_1, \cdots, x_d) = \mathbb{1}_{y=f(x_1,\cdots,x_d)}$$

Having to specify the CPT of a probabilistic aggregator implies that even if the cardinality of the output domain usually takes much less space than the set to be aggregated (because we aim to summarize the information), the encoding of such table is very consuming in terms of memory (which will have a direct impact on inference time). For an aggregator with n parents, whose input domain size is $|I|$, the CPT is defined using $|O| * |I|^n$ parameters.

5.4.2 Self-Decomposable Aggregation Functions

Some deterministic aggregators can be, however, evaluated by computing the aggregate for subsets, and then aggregating these results, reducing the size of the concerned CPTs. Similarly to [Jesus et al. 2015], we call such aggregators *self-decomposable*.

Definition 5.5

Self-decomposable aggregation function *A deterministic aggregation function $f : \mathbb{N}^I \to O$ is said to be self-decomposable if, for some merge operator $\diamond$ and all non-empty multisets $\mathbf{X}$ and $\mathbf{Y}$:*

$$f(\mathbf{X} \uplus \mathbf{Y}) = f(\mathbf{X}) \diamond f(\mathbf{Y})$$

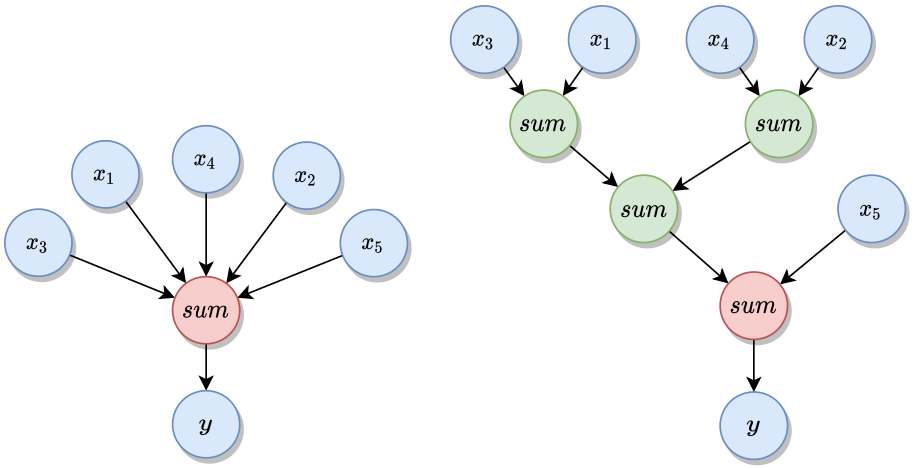
In the above, $\uplus$ denotes the multiset sum. For two multisets $M_1 = (A_1, m_1)$ and $M_2 = (A_2, m_2)$, their sum $M_3 = (A_3, m_3)$ is defined by $A_3 = A_1 \cup A_2$ and for all $x \in A_3, m_3(x) = m_1(x) + m_2(x)$ (with $m(x) = 0$ if $x \notin A$). Given that the aggregation result is the same for all possible partitions of a multiset, it follows that the operator $\diamond$ has to be both commutative and associative.

Many probabilistic aggregators such as MIN, MAX and SUM are self-decomposable, as shown below :

$$\begin{aligned} &\textsc{sum}(\{x\}) = x, \qquad \textsc{sum}(\mathbf{X} \uplus \mathbf{Y}) = \textsc{sum}(\mathbf{X}) + \textsc{sum}(\mathbf{Y}) \\ &\textsc{min}(\{x\}) = x, \quad \textsc{min}(\mathbf{X} \uplus \mathbf{Y}) = \textsc{min}(\textsc{min}(\mathbf{X}), \textsc{min}(\mathbf{Y})) \end{aligned}$$

To face our issue with aggregation being a bottleneck we propose to manipulate the grounded BN before the generation of its attributes' CPTs (aGrUM allows us to declare aggregators with their functional definition and to instantiate their CPTs only at inference time). When the BN contains a self-decomposable aggregator n with $|\mathrm{Pa}(n)| > 2$ we can create intermediate aggregators between its parents (green nodes in Figure 5.45b), grouping them by pairs, then linking these aggregators by pairs themselves (similarly with the divorcing method [von Waldow and Röhrbein 2015]).

Figure 5.45 shows how such manipulation is changing a simple BN with one sum aggregator having 5 parents. In this example, each attribute $x_i \in \{x_1, ..., x_n\}$ can take 10 values. In the decomposed counterpart the largest CPT contains 2500 times less parameters than in the original one.



(a) (b)

Figure 5.45: (a) is a BN before its decomposition, its largest CPT contains $50 \cdot 10^5$ values. (b) is the same BN, after decomposition. Its largest CPT contains $50 \cdot 40 \cdot 10$ values.

There is a trade-off between the number of parents of the intermediate aggregators and the depth of the tree. Intuitively, increasing the depth of the tree seems to complexify the model linearly in the number of parameters, while a "horizontal" expansion due to an additional parents would complexify the model exponentially, consequently we choose to limited the number of parents to 2.

Table 5.4 shows the time needed to decompose and compute marginals in our example. As expected reducing the size of the aggregator's node helped us reducing the complexity of the inference, we are now able to perform them in more acceptable times, even with an important number of parents.

| Pa |	decomposition	inference	file size
5	6.6e-4s	3.9e-3s	109Ko
10	1.9e-3s	4.9e-2s	1,5Mo
25	4.6e-3s	1.6e-1s	1,5Mo
50	1.0e-2s	4.1e-1s	3.3Mo
75	1.7e-2s	5.9e-1s	5.2Mo
100	2.0e-2s	7.5e-1s	6.8Mo

Table 5.4: Decomposition and inference time to compute the probability of the rule Stock Control and size of the generated file, depending on |Pa|

5.4.3 Discussion

If this simple method is effective it does not answer all the challenges related to the scalability of PGMs:

- First of all, not all aggregators are self-decomposable (like the median);
- We move from a functional representation, free in terms of space, to a digital representation which are by nature more expensive, even if we limit their complexity. Moreover, these structures are generally very sparse, a lot of memory is needed to contain only a little information;
- Non-observable variables (intermediate aggregators) are added in the model, affecting its readability;
- Finally, we must insist on the fact that beyond aggregators, any random variable can have an arbitrarily high number of parents and can therefore cause a modelling issue even before being one in inference.

The problem of scalability is therefore more general than simply related to the use of aggregators. As discussed in section 3.3, some approximate inference algorithms propose solutions to overcome such a challenge, but neither at no computational cost, nor totally reliably.

5.5 Conclusion

In this chapter we have proposed a simpler and more accessible syntax for business users to use probabilistic reasoning with their business rules. This syntax has been tested experimentally through a functional prototype on a sub-language of ODM technical language, IRL, and integrated in its compilation chain. While the results are encouraging, the syntax is too powerful for our reasoning capabilities, notably because of the very simplified use of aggregators. A first application solution to limit this complexity has been proposed.

In the rest of this document, we will present a new possible solution to address this issue, but this time based on a change in the way potentials are represented.

Bibliography

Agli, H. (2017). *Uncertain reasoning for business rules*. PhD thesis, Université Pierre et Marie Curie - Paris VI.

Ait-Kaci, H. and Bonnard, P. (2011). Probabilistic production rules. Technical report, IBM France Lab.

Arru, M. (2011). Introduction of probabilistic reasoning in business rules managmenet systems. Master's thesis, Polytech Nantes.

Ducamp, G., Bonnard, P., de Sainte Marie, C., Gonzales, C., and Wuillemin, P.-H. (2018). Improving probabilistic rules compilation using prm. In *RuleML+RR Doctoral Consortium 2018 (2nd International Joint Conference on Rules and Reasoning)*, Esch-sur-Alzette, Luxembourg.

Ducamp, G., Bonnard, P., De Sainte Marie, C., and Wuillemin, P.-H. (2020a). Advanced Syntax and Compilation for Probabilistic Production Rules with PRM. In *RuleML+RR Doctoral Consortium 2020 (4th International Joint Conference on Rules and Reasoning)*, volume 2644 of *CEUR Workshop*

Proceedings, pages 103–110, Oslo, Norway. CEUR-WS.org. Virtual conference.

Ducamp, G., Bonnard, P., and Wuillemin, P.-H. (2020b). Uncertain reasoning in rule-based systems using prm. In *Florida Artificial Intelligence Research Society Conference*.

Jesus, P., Baquero, C., and Almeida, P. (2015). A survey of distributed data aggregation algorithms. *IEEE Communications Surveys and Tutorials*, 17(1):381–404.

Madsen, A. L. and Jensen, F. V. (1999). Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*.

Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine learning*, 62(1-2):107–136.

von Waldow, U. and Röhrbein, F. (2015). Structure learning in bayesian networks with parent divorcing.

Chapter 6

A New Representation for Potentials : The TT Format

Contents

We have previously seen that one of the constraints to the algorithms in PGMs is the complexity of the studied models and that such complexity was directly related to the size of the CPTs associated with each of their random variables. Consequently, when exact algorithms suffer from space complexity in large-scale models, approximate ones can only offer a trade-off between time complexity and precision, sometimes without a guarantee of convergence toward a stationary distribution. The objective of this chapter is to show that the use of low-rank tensor methods might be a possible way to mitigate the *curse of dimensionality* for discrete high-dimensional models.

The use of tensor decomposition in PGMs is not unheard of. In previous works, [Savický and Vomlel 2007] proposed to manipulate the structure of a Bayesian network and use tensor rank-one decomposition to approximate *some* special forms of CPTs, [Vomlel and Tichavský 2014] use CP decomposition of tensors corresponding to CPTs of threshold functions, exactly l-out-of-k functions, and their noisy counterparts. In a more general case, tensor methods combined with exact algorithms could provide a new approach to deal with complex PGM in a controlled and tractable way. It is important to note that this approach is not limited to Bayesian networks nor to inference algorithms. Every representation of a high-dimensional multivariate function as a product of multivariate factors, every algorithm that operates on a commutative semiring of such multivariate factors are limited by the dimension of these very factors and then could benefit from this compact representation with controlled approximation, our approach is intended to be more general than those proposed in the work cited above.

As an example of the use of low-rank tensor representation for PGM, we propose in this chapter to focus on probabilistic inference in Bayesian networks using the Tensor Train format.

This chapter adapts, corrects and completes the work presented in [Ducamp et al. 2020].

6.1 Tensors

Tensor methods have become a prominent tool for solving high-dimensional problems arising in physics, financial mathematics, statistics, uncertainty quantification, data science, and many other fields involving the approximation of high-dimensional functions or multidimensional arrays. For an introduction to tensor methods and their applications in numerical analysis and machine learning, the reader is referred to the monograph [Hackbusch 2019] and the surveys [Kolda and Bader 2009; Nouy 2017; Bachmayr et al. 2016; Cichocki et al. 2016; 2017; Ji et al. 2019].

6.1.1 Definitions and Metrics

In this manuscript we define tensors as a generalization of the notions of scalars, vectors and matrices to a larger number of dimensions, *i.e.* , as multidimensional arrays. While vectors have entries v_i with one index and matrices have entries M_{ij} with two indices, tensors will carry d indices. Such number of indices is called the *order* of the tensor.

$$\lambda \qquad \begin{pmatrix} v_1 \\ v_2 \\ v_3 \end{pmatrix} \qquad \begin{pmatrix} M_{11} M_{12} M_{13} \\ M_{21} M_{22} M_{23} \\ M_{31} M_{32} M_{33} \end{pmatrix} \qquad \begin{pmatrix} T_{111} T_{121} T_{131} \\ T_{211} T_{221} T_{231} \\ T_{311} T_{321} T_{331} \end{pmatrix}$$

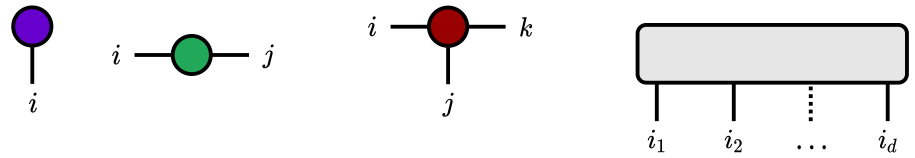
(a) (b) (c) (d)

Figure 6.1: Example of tensors with (a) $d = 0$ (b) $d = 1$ (c) $d = 2$ (d) $d = 3$

We denote by $\mathbb{R}^{n_1 \times \cdots \times n_d}$ the space of tensors of order d and size $n_1 \times \ldots \times n_d$. The entries of a tensor $\mathbf{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ are denoted by $\mathbf{T}(i_1, \ldots, i_d)$ (sometimes $\mathbf{T}_{i_1,\ldots,i_d}$), $1 \leq i_\nu \leq n_\nu$, $1 \leq \nu \leq d$ where the index i_ν is related to the ν-th mode (or dimension) of the tensor.

A tensor $\mathbf{T}$ can be identified with a vector $\mathsf{vec}(\mathbf{T})$ whose entries are $\mathsf{vec}(\mathbf{T})(\overline{i_1, \ldots, i_d}) = \mathbf{T}(i_1, \ldots, i_d)$, with $\overline{i_1, \ldots, i_d} = i_d + (i_{d-1} - 1)n_d + \ldots + (i_1 - 1)n_2 \ldots n_d$.

When the order is small it is easy to conceptualize the manipulated objects since they correspond to scalars, vectors, matrices or even 3D matrices, as shown in the Figure 6.1. But what about higher dimensions ? To illustrate the document and facilitate its reading we will sometimes use a graphic notation called Tensor Diagram Notation[1] [Penrose 1971; Bridgeman and Chubb 2017] where tensors are notated by solid shapes, and tensor indices are notated by lines emanating from these shapes.



(a) (b) (c) (d)

Figure 6.2: (a) a vector v_i. (b) a matrix M_{ij}. (c) a tensor $\mathbf{T}_{ijk}$ (d) a tensor $\mathbf{T}_{i_1,\ldots,i_d}$

6.1.2 Operations Between Tensors

In order to manipulate tensors, we need to introduce a number of elementary operations on them. For a more detailed and exhaustive list, the reader may refer to [Lee and Cichocki 2018; Hackbusch 2019]. We will take this opportunity to observe equivalences between tensor algebra and potential algebra. The first two operations on tensors that we need to introduce are the Kronecker product and its partial counterpart.

Definition 6.1

Kronecker product *The Kronecker product (denoted $\otimes$) of two tensors $\mathbf{A} \in \mathbb{R}^{I_1 \times \ldots \times I_N}$ and $\mathbf{B} \in \mathbb{R}^{J_1 \times \ldots \times J_N}$ yields a tensor $\mathbf{C} = \mathbf{A} \otimes \mathbf{B}$ of size $I_1 J_1 \times \cdots \times I_N J_N$ with entries*

$$\mathbf{C}(\overline{i_1 j_1}, \ldots, \overline{i_N j_N}) = \mathbf{A}(i_1, \ldots, i_N)\mathbf{B}(j_1, \ldots, j_N) \tag{6.1}$$

[1] https://tensornetwork.org/diagrams/

We will use Kronecker's product rather than the more general outer product because it allows a conservation of orders (while the outer product conserves the ranks), as showcased in the Figure 6.3. This characteristic is important for us since the representation of tensors that we will work with thereafter only uses order-3 tensors. One can nevertheless notice the equivalence between these two operations, as Remark 6.2 shows.

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \otimes \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{12}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \\ a_{21}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} & a_{22}\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} & a_{11}b_{12} & a_{12}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{11}b_{22} & a_{12}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{21}b_{12} & a_{22}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{21}b_{22} & a_{22}b_{21} & a_{22}b_{22} \end{bmatrix}$$

Figure 6.3: Example of Kronecker product between two order-2 tensors resulting in another order-2 tensor

Remark 6.2

Let $\mathbf{U} \in \mathbb{R}^3$ and $\mathbf{V} \in \mathbb{R}^2$ denotes two order-1 tensors. If we denote by $\otimes_{outer}$ the outer product, it holds that :

$$\mathbf{U} \otimes \mathbf{V} = \begin{bmatrix} u_1v_1 \\ u_1v_2 \\ u_2v_1 \\ u_2v_2 \\ u_3v_1 \\ u_3v_2 \end{bmatrix}, \mathbf{U} \otimes_{outer} \mathbf{V} = \begin{bmatrix} u_1v_1 & u_1v_2 \\ u_2v_1 & u_2v_2 \\ u_3v_1 & u_3v_2 \end{bmatrix}$$

As we can see, in the case of order-1 tensors, the Kronecker product can be viewed as a form of vectorization (or flattening) of their outer product.

Definition 6.3

Partial Kronecker product *The partial Kronecker product of two tensors along the modes α is denoted $\boxtimes_\alpha$. The product $\boxtimes_{\{1,\dots,M\}}$ along modes $\{1,\dots,M\}$ for two tensors $\mathbf{A} \in \mathbb{R}^{R_1 \times \cdots \times R_M \times I_1 \times \cdots \times I_N}$ and $\mathbf{B} \in \mathbb{R}^{S_1 \times \cdots \times S_M \times I_1 \times \cdots \times I_N}$ yields a tensor $\mathbf{C} = \mathbf{A} \boxtimes_{\{1,\dots,M\}} \mathbf{B}$ of size $R_1 S_1 \times \cdots \times R_M S_M \times I_1 \times \cdots \times I_N$ with sub-tensors*

$$\mathbf{C}(:,\dots,:,i_1,\dots,i_N) = \mathbf{A}(:,\dots,:,i_1,\dots,i_N) \otimes \mathbf{B}(:,\dots,:,i_1,\dots,i_N) \quad (6.2)$$

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \boxtimes_{\{1\}} \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{11}b_{21} & a_{12}b_{22} \\ a_{21}b_{11} & a_{22}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}$$

Figure 6.4: Example of partial Kronecker product along first mode (rows) between two order-2 tensors

Remark 6.4

Let's consider the case where we have $\mathbb{P}(\mathrm{A},\mathrm{B},\mathrm{C})$ and $(\mathrm{A} \perp_{\mathbb{P}} \mathrm{B}|\mathrm{C})$. Thus, we would have $\mathbb{P}(\mathrm{A},\mathrm{B},\mathrm{C}) = \mathbb{P}(\mathrm{A}|\mathrm{C})\mathbb{P}(\mathrm{B},\mathrm{C})$. If we denote by $\mathbf{T}_{\{\mathrm{A},\mathrm{B},\mathrm{C}\}}$ the tensor based on the potential $\mathbb{P}(\mathrm{A},\mathrm{B},\mathrm{C})$ and $\mathbf{T}_{\{\mathrm{A}|\mathrm{C}\}}$ (resp. $\mathbf{T}_{\{\mathrm{B},\mathrm{C}\}}$) the one associated with $\mathbb{P}(\mathrm{A}|\mathrm{C})$ (resp. $\mathbb{P}(\mathrm{B},\mathrm{C})$) it holds that :

$$\mathbf{T}_{\{\mathrm{A},\mathrm{B},\mathrm{C}\}} = \mathbf{T}_{\{\mathrm{A}|\mathrm{C}\}} \boxtimes_{\{2\}} \mathbf{T}_{\{\mathrm{B},\mathrm{C}\}}$$

The last operation we need is the contracted product which corresponds to a contraction between two tensor indices. In our case it will always be between the last (M^{th}) mode of the first tensor and the first mode of the second one tensor.

Definition 6.5

Mode-(M,1) contracted product *The mode-(M,1) contracted product (denoted $\times^1$) of tensors $\mathbf{A} \in \mathbb{R}^{I_1 \times \ldots \times I_M}$ and $\mathbf{B} \in \mathbb{R}^{J_1 \times \ldots \times J_N}$ with $I_M = J_1$ yields a tensor $\mathbf{C} = \mathbf{A} \times^1 \mathbf{B}$ of size $I_1 \times \cdots \times I_{M-1} \times J_2 \times \cdots \times J_N$ with entries*

$$\mathbf{C}(i_1, \ldots, i_{M-1}, j_2, \ldots, j_N) = \sum_{i_M=1}^{I_M} \mathbf{A}(i_1, \ldots, i_M)\mathbf{B}(i_M, j_2, \ldots, j_N) \quad (6.3)$$

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} \times^1 \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{bmatrix} = \begin{bmatrix} (a_{11}b_{11} + a_{12}b_{21} + a_{13}b_{31}) & (a_{11}b_{12} + a_{12}b_{22} + a_{13}b_{32}) \\ (a_{21}b_{21} + a_{22}b_{21} + a_{23}b_{31}) & (a_{21}b_{11} + a_{22}b_{22} + a_{23}b_{32}) \end{bmatrix}$$

Figure 6.5: Example of Mode-(M,1) contracted product between two order-2 tensors

The Figure 6.6 illustrates such operation using the Tensor Diagram Notation, where connected index lines implies a contraction, or summation over the connected indices.

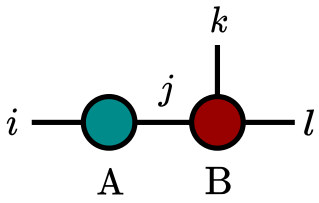


Figure 6.6: Mode contracted product over two tensors $\mathbf{A}_{ij}$ and $\mathbf{B}_{jkl}$.

Remark 6.6

It is interesting to note that this operation is the one used when one marginalizes a common variable in the result of a product of potentials! Let $\mathrm{A}, \mathrm{B}, \mathrm{C}$ *and* D *be random variables and* $\phi(\mathrm{A}, \mathrm{B}, \mathrm{C})$, $\phi(\mathrm{C}, \mathrm{D})$ *denote potentials over these variables. If we denote by* $\mathbf{T}_{\{\mathrm{A},\mathrm{B},\mathrm{C}\}}$ *(resp.* $\mathbf{T}_{\{\mathrm{C},\mathrm{D}\}}$*) the tensor associated with* $\phi(\mathrm{A}, \mathrm{B}, \mathrm{C})$ *(resp.* $\phi(C, D)$*), we have :*

$$\mathbf{T}_{\{\mathrm{A},\mathrm{B},\mathrm{C}\}} \times^1 \mathbf{T}_{\{\mathrm{C},\mathrm{D}\}} = \sum_{\mathrm{C}} \phi(\mathrm{A}, \mathrm{B}, \mathrm{C}) \times \phi(\mathrm{C}, \mathrm{D})$$

Now that we have defined the operations necessary for our calculations, we have to find a representation for our tensors that allows us to reduce their dimensionality.

6.2 Low-Rank Tensor Formats

In this section we will introduce some tools to circumvent the main limitation related to the use of *full* tensors. Since the number of entries of such a tensor grows exponentially with the order d —and so does the storage consumption and computational complexity of basic multilinear algebra operations between tensors—, it is essential to consider structured tensor formats, such as low-rank tensor formats, for handling high-order tensors.

6.2.1 Tensor Ranks and Related Tensor Formats

An elementary tensor $\mathbf{T} = \mathbf{T}^{(1)} \otimes \ldots \otimes \mathbf{T}^{(d)}$ is the tensor product of d order-1 tensors (vectors) $\mathbf{T}^{(\nu)} \in \mathbb{R}^{n_\nu}$, whose entries are $\mathbf{T}(i_1, \ldots, i_d) = \mathbf{T}^{(1)}(i_1) \ldots \mathbf{T}^{(d)}(i_d)$. It is noteworthy that if we consider a set of marginals of random variables all independent of each other, the potential associated with the product of these marginals is an elementary tensor. The canonical rank of a tensor $\mathbf{T}$, denoted $\mathrm{rank}(\mathbf{T})$, is the minimal integer r such that the tensor can be written as a sum of r elementary tensors.

Every tensor $\mathbf{T}$ can be represented as a linear combination of r order-1 tensors, such representation is called *Canonical Polyadic Decomposition* [Hitchcock 1927] (and sometimes referred as CANDECOMP [Carroll and Chang 1970] or PARAFAC [Harshman 1970]). This format was notably used in [Savický and Vomlel 2007; Vomlel and Tichavský 2014] in order to factorize some CPTs during probabilistic inferences. However, computing the canonical rank of such tensor is NP-Hard [Hillar and Lim 2013, Theorem 1.13]. We will see that other representations, more complex but also more structured, can be defined.

For any subset $\alpha \subset \{1,\dots,d\} := D$ and its complementary subset $\alpha^c = \{1,\dots,d\}\backslash\alpha$, a tensor $\mathbf{T}$ can be identified with a matrix $\mathcal{M}_\alpha(\mathbf{T})$ whose entries are —up to a permutation of indices—:

$$\mathcal{M}_\alpha(\mathbf{T})(\overline{(i_\nu)_{\nu\in\alpha}}, \overline{(i_\nu)_{\nu\in\alpha^c}}) = \mathbf{T}(i_1,\dots,i_d).$$

The map $\mathcal{M}_\alpha$, called the α-matricisation operator, is a bijection from $\mathbb{R}^{n_1\times\cdots\times n_d}$ to $\mathbb{R}^{N_\alpha\times N_{\alpha^c}}$, where $N_\beta = \prod_{\nu\in\beta} n_\nu$. The rank of the matrix $\mathcal{M}_\alpha(\mathbf{T})$ is called the α-rank of $\mathbf{T}$, and denoted $\mathrm{rank}_\alpha(\mathrm{T})$. By convention, the D-rank and $\emptyset$-rank of a tensor are equal to 1.

Letting $S \subset 2^D$ be a set of subsets of D, we define the S-rank of a tensor $\mathbf{T}$ as the tuple $(\mathrm{rank}_\alpha(\mathrm{T}))_{\alpha\in S} \in \mathbb{N}^{|S|}$. For a given set S and a tuple $r = (r_\alpha)_{\alpha\in S}$, a tensor format $\mathcal{T}_r^S$ is defined as the set of tensors with S-rank less than r,

$$\mathcal{T}_r^S = \{\mathbf{T} \in \mathbb{R}^{n_1\times\cdots\times n_d} : \mathrm{rank}_\alpha(\mathrm{T}) \leq r_\alpha, \alpha \in S\}$$

When S is a dimension partition tree over D (with root D and leaves $\{\nu\}, 1 \leq \nu \leq d$) or a subset of such a tree, $\mathcal{T}_r^S$ is called a tree-based tensor format [Falcó et al. 2018]. Such format includes the Tucker format for a trivial tree (where $S = \{\{1,\dots,d\},\{1\},\dots,\{d\}\}$), the Hierarchical Tucker (HT) format [Hackbusch and Kuhn 2009] for a binary tree, and the Tensor Train format described below.

The Figure 6.7 show the standard Tucker decomposition of a tensor $\mathbf{T}(i_1,\dots,i_8)$ into a core tensor (red circle) and eight factor matrices (blue circles). In this case, $\mathbf{T} = \mathcal{T} \times_1 \mathbf{T}^{(1)} \cdots \times_8 \mathbf{T}^{(8)}$, where $\mathcal{T}$ is an order-8 ten-

sors, $\mathbf{T}^{(i)}$ the i^{th} order-2 tensor and $\times_i$ denotes the i-mode (matrix) product. Its transformation into an equivalent Hierarchical Tucker model using interconnected smaller size 3rd-order core tensors and the same factor matrices is also presented.

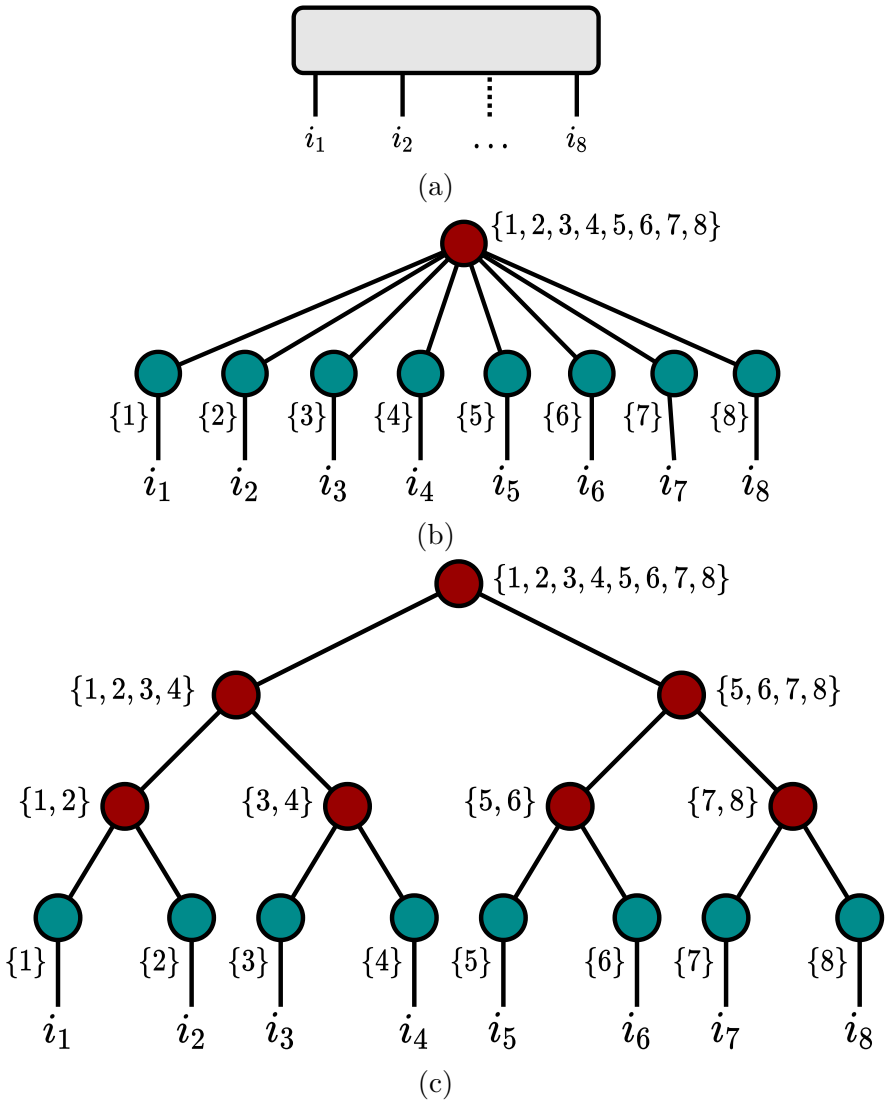


Figure 6.7: Example of tensor in the (a) *full* format (b) Tucker format (c) HT format

6.2.2 Tensor Train Format

The Tensor Train format has been introduced in [Oseledets 2009; Oseledets and Tyrtyshnikov 2009] in the context of numerical analysis. It was already known in quantum physics as Matrix Product State (see, e.g., [Schollwöck 2011]). This format corresponds to the tensor format $\mathcal{T}_r^S$ with $S = \{\emptyset, \{1\}, \{1,2\}, \ldots, \{1,\ldots,d-1\}, D\}$. Given a tuple of integers $r = (r_0, r_1, \ldots, r_d)$, with $r_0 = r_d = 1$, a tensor $\mathbf{T}$ in the tensor format $\mathcal{T}_r^S$ admits the representation :

$$\mathbf{T}(i_1,\ldots,i_d) = \sum_{k_1=1}^{r_1} \cdots \sum_{k_{d-1}=1}^{r_{d-1}} \mathbf{T}^{(1)}(1,i_1,k_1)\cdots\mathbf{T}^{(d)}(k_{d-1},i_d,1) \tag{6.4}$$

where the $\mathbf{T}^{(i)} \in \mathbb{R}^{r_{i-1}\times n_i \times r_i}$ are order-3 tensors called TT cores. The minimal integers $(r_0, r_1, \ldots, r_d)$ such that $\mathbf{T}$ has a representation (according to Equation 6.4) is called the TT-rank of $\mathbf{T}$. The Figure 6.8 illustrates such a format (r_0 and r_d are usually omitted since $r_0 = r_d = 1$).

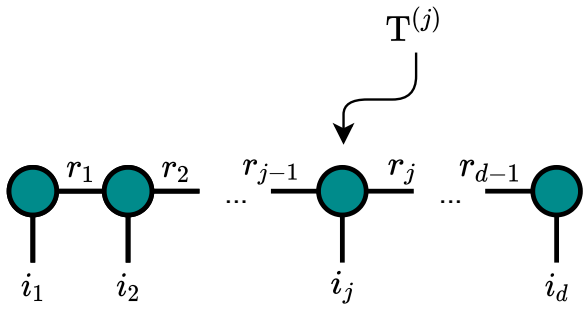


Figure 6.8: A tensor $\mathbf{T}(i_1,\ldots,i_j,\ldots,i_d)$ in the Tensor Train format

Property 6.7

Storage Complexity *The storage complexity of a tensor with TT ranks bounded by R and mode sizes bounded by N is in $O(dNR^2)$. This tensor format allows us to circumvent the curse of dimensionality for classes of tensors with TT-rank uniformly bounded or growing polynomially with d.*

6.2.3 Approximation in Tensor Train Format

Every tensor has an exact representation in the Tensor Train format [Oseledets 2011] although without any compression (possibly with high representation ranks). The Figure 6.9 show a tensor $\mathbf{T}(i_1, \ldots, i_6)$ converted within a Tensor Train format where $(r_1, \ldots, r_5)$ are its representation ranks.

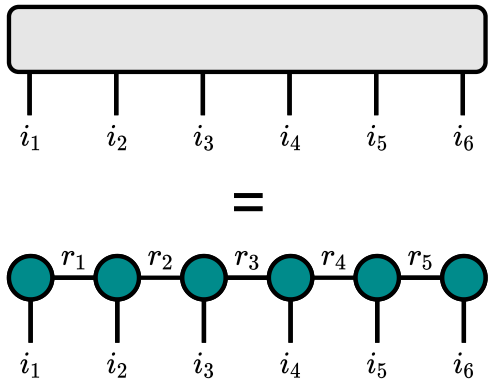


Figure 6.9: A *full* tensor and its TT representation

In order to control their sizes, many algorithms have been proposed to not only transform but also compress full tensors into tensor in the Tensor Train format. The algorithm described below, introduced in [Oseledets 2011][p. 2301], allows us to obtain an approximation $\widetilde{\mathbf{T}}_\epsilon$ in the Tensor Train format of a given tensor $\mathbf{T}$ with a prescribed relative precision ϵ, *i.e.* , $\left\|\mathbf{T} - \widetilde{\mathbf{T}}_\epsilon\right\|_F \leq \epsilon \|\mathbf{T}\|_F$, where $\|\cdot\|_F$ denotes the Frobenius (or canonical) tensor norm (if $\mathbf{T} \in \mathbb{R}^{\{n_1, \cdots, n_d\}}$, $\|\mathbf{T}\|_F^2 = \sum_{i_1, \cdots, i_d} \mathbf{T}_{i_1, \cdots, i_d}^2$). The algorithm relies on standard singular value decompositions of matrices. For more details on the Tensor Train format and its applications, see, *e.g.* , [Gelß 2017]. The algorithm is here described for the case where the input $\mathbf{T}$ is full tensor (a multidimensional array) but a version where the input is directly in the TT format also exists [Oseledets 2011][p. 2305].

Compress($\mathbf{T}, \epsilon$) - Higher-order truncated SVD for the approximation in TT format

Input : Tensor $\mathbf{T} \in \mathbb{R}^{n_1 \times \cdots \times n_d}$ and tolerance ϵ
Output Approximation $\widetilde{\mathbf{T}}_\epsilon$ of $\mathbf{T}$ in TT format with cores $\mathbf{U}^{(1)}, \ldots, \mathbf{U}^{(d)}$ and ranks $r_0, \ldots, r_d$
Set $r_0 = 1$ and $r_d = 1$. Set $\mathbf{A} \in \mathbf{R}^{r_0 \times n_1 \times \ldots \times n_d}$ such that $\mathbf{A}(1, i_1, \ldots, i_d) = \mathbf{T}(i_1, \ldots, i_d)$
foreach $\nu = 1, \ldots, d-1$ **do**
  $A \leftarrow \mathcal{M}_{\{1,2\}}(\mathbf{A}) \in \mathbf{R}^{(r_{\nu-1} n_\nu) \times (n_{\nu+1} \ldots n_d)}$;
  Compute SVD of A, i.e. $A = U \Sigma V^T$ with $\Sigma = \mathsf{diag}(\sigma_1, \ldots, \sigma_d) \in \mathbb{R}^{s \times s}$, $U \in \mathbb{R}^{(r_{\nu-1} n_\nu) \times s}$ and $V \in \mathbb{R}^{(n_{\nu+1} \ldots n_d) \times s}$;
  Set $r_\nu \leq s$ to the smallest index such that $\sigma^2_{r_{\nu+1}} + \ldots + \sigma^2_s \leq \frac{\epsilon^2}{(d-1)}$;
  Discard rows and columns of U, Σ, and V corresponding to singular values $\sigma_{r_{\nu+1}}, \ldots, \sigma_s$;
  Define the ν-th core $\mathbf{U}^{(\nu)} = \mathcal{M}^{-1}_{\{1,2\}}(U) \in \mathbb{R}^{r_{\nu-1} \times n_\nu \times r_\nu}$;
  Define $\mathbf{A} = \mathcal{M}^{-1}_{\{1\}}(\Sigma V^T) \in \mathbb{R}^{r_\nu \times n_{\nu+1} \times \ldots \times n_d}$
Define the d-th core $\mathbf{U}^{(d)} = \mathcal{M}^{-1}_{\{1\}}(\mathbf{A})$

It is possible to define a similar procedure where, instead of a tolerance ϵ, we compress a tensor $\mathbf{T} \in \mathbb{R}^{(r_1 \times \ldots \times r_d)}$ w.r.t a maximum value for r_{max} such that $\forall i \in (1, \ldots, d-1), r_i \leq r_{max}$. The Figure 6.10 illustrates such a procedure. The maximum number of parameter necessary to describe this new tensor will consequently change from $d \cdot N \cdot R^2$ to $d \cdot N \cdot r^2_{max}$ with N being the max mode size in $\mathbf{T}$.

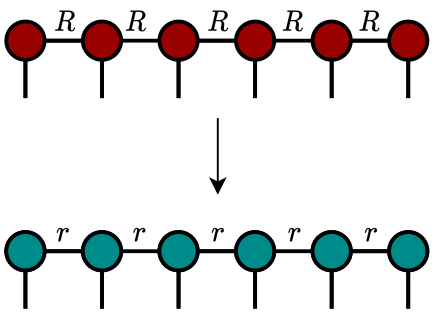


Figure 6.10: Rank based compression of a tensor in the TT format with $r_i = R$ for all $i \in (1, \ldots, 5)$ and $r_{max} = r$ (and $r \leq R$)

6.3 BN and TT Format

Using the TT format seems to be a promising approach to reduce the memory consumption of potentials in PGMs. Such format make it possible to go from having a representation of potentials that evolves no longer exponentially with the number of parents but linearly, with a control on the induced approximation when they are further compressed. Given the context of this thesis, we decided to focus on an experimentation around an inference that would manipulate potentials as tensors in the Tensor Train format instead of multidimensional arrays. Proposing a reinterpretation of the Shafer-Shenoy inference (presented in Section 3.3.3.1.3) as a first experiment was straightforward since it requires redefining only a few operations and that it will be simple to compare our new approach with this standard algorithm. Let's summarize the benefits of such a format :

1. This shift in data representation could be interesting not only for inferences but for all algorithms that manipulate large potentials.

2. An approximation with controlled precision can be obtained using the **Compress** procedure;

3. An approximation can be found using an upper limit for TT ranks, allowing to easily control memory usage;

Before redefining Shafer-Shenoy elementary operations for tensors in the TT format and, finally, the complete algorithm, we have to introduce another operation on tensor trains, the Hadamard product. For comparison purposes, we give the definition of such operation for full tensors.

Definition 6.8

Hadamard product *The Hadamard product (denoted $\circledast$) of two tensors* $\mathbf{A}$ *and* $\mathbf{B}$ *of same size* $n_1 \times \ldots \times n_d$ *yields a tensor* $\mathbf{C} = \mathbf{A} \circledast \mathbf{B}$ *with entries*

$$\mathbf{C}(i_1, \ldots, i_d) = \mathbf{A}(i_1, \ldots, i_d)\mathbf{B}(i_1, \ldots, i_d) \tag{6.5}$$

6.3.1 Hadamard Product in TT Format

Let's recall that a tensor $\mathbf{T}$ with the representation in TT format (Equation 6.4) can be written

$$\mathbf{T} = \mathbf{T}^{(1)} \times^1 \cdots \times^1 \mathbf{T}^{(d)}.$$

If $\mathbf{A}$ and $\mathbf{B}$ are two tensors with the same size and with representations in TT formats $\mathbf{A} = \mathbf{A}^{(1)} \times^1 \cdots \times^1 \mathbf{A}^{(d)}$ and $\mathbf{B} = \mathbf{B}^{(1)} \times^1 \cdots \times^1 \mathbf{B}^{(d)}$, then their Hadamard product $\mathbf{A} \circledast \mathbf{B}$ has a representation in TT format :

$$\mathbf{A} \circledast \mathbf{B} = (\mathbf{A}^{(1)} \boxtimes_{\{1,3\}} \mathbf{B}^{(1)}) \times^1 \cdots \times^1 (\mathbf{A}^{(d)} \boxtimes_{\{1,3\}} \mathbf{B}^{(d)}) \tag{6.6}$$

where $\boxtimes_{\{1,3\}}$ is the partial Kronecker product along modes 1 and 3, defined in Equation 6.1. The Figure 6.11 illustrates such operation.

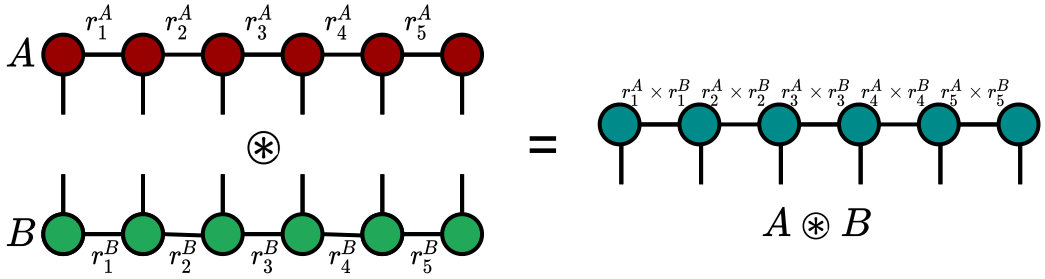


Figure 6.11: Hadamard product of two tensors in the TT format

Remark 6.9

Let $\mathbf{A}^{(i)} \in \mathbb{R}^{r^A_{i-1} \times n^A_i \times r^A_i}$ and $\mathbf{B}^{(i)} \in \mathbb{R}^{r^A_{i-1} \times n^B_i \times r^B_i}$ denotes the tensor cores of $\mathbf{A}$ and $\mathbf{B}$ for all $i \in (1, \ldots, d)$. To be able to compute the partial Kronecker product $\mathbf{A}^{(i)} \boxtimes_{\{1,3\}} \mathbf{B}^{(i)}$ we need to make sure that $n^A_i = n^B_i$. For this purpose and to avoid having to manipulate the position of the cores during inference we will force an arbitrary order on the variables within a potential (and for the sake of simplicity we will use a topological one).

6.3.2 Potentials Algebra with TT Format

Using the **Compress** procedure, we are now able to compress any tensor associated with a multivariate function (Conditional Probability Distribution, CPT, potentials, etc.) in a Tensor Train format controlled by a given relative error. In order to build algorithms for PGM using this compressed format, we now need to define the operations used by such algorithms. In our case, the clique-separator product and a marginalization.

6.3.2.1 Clique-Separator Product

The product of the potential ϕ of a clique with one of its separators ψ can be obtained using a Hadamard product between tensors. However, this requires ϕ and ψ to be tensors with the same order and size. Since separator's variables form a subset of the clique's variables, ψ has to be identified with a tensor ψ' with the order and size of ϕ. It is sufficient to consider the case where ϕ is a tensor of order d and size $n_1 \times \ldots \times n_d$ depending on variables $(i_1, \ldots, i_d)$ and ψ is a tensor of order $d-1$ depending on variables $(i_1, ..., i_{\nu-1}, i_{\nu+1}, ..., i_d)$. Then ψ' is the tensor of order d such that $\psi'(i_1, ..., i_d) = \psi(i_1, ..., i_{\nu-1}, i_{\nu+1}, ..., i_d)$, and the Hadamard product $\phi \circledast \psi$ has to be interpreted as $\phi \circledast \psi'$.

Property 6.10

If ψ has a representation in TT format with cores $\mathbf{T}^{(\mu)}$ and representation ranks r_ν, $\mu \in \{1, \ldots, \nu-1, \nu+1, \ldots, d\}$, then ψ' has a representation in TT format with cores $\mathbf{T'}^{(\mu)} = \mathbf{T}^{(\mu)}$ for $\mu \in \{1, \ldots, \nu-1, \nu+1, \ldots, d\}$ and $\mathbf{T'}^{(\nu)} \in \mathbb{R}^{r_{\nu-1} \times n_\nu \times r_\nu}$ such that $\mathbf{T'}^{(\nu)}(k_{\nu-1}, i_\nu, k_\nu) = \delta_{k_{\nu-1}, k_\nu}$, where δ represents the Kronecker delta.

Let ϕ and $\psi \in \mathbb{R}^{i_1 \times \ldots \times ..., i_d}$ be two tensors in the TT format with $\phi^{(i)} \in \mathbb{R}^{r^\phi_{i-1} \times n_i \times r^\phi_i}$ and $\psi'^{(i)} \in \mathbb{R}^{r^{\psi'}_{i-1} \times n_i \times r^{\psi'}_i}$ for all $i \in \{1, \ldots, d\}$ then if we denote ϕ' tensor in the TT format that equals to $\phi \circledast \psi$ and because of the internal partial Kronecker products used (cf Equation 6.6), we have $\phi'^{(i)} \in \mathbb{R}^{(r^\phi_{i-1} \times r^{\psi'}_{i-1}) \times n_i \times (r^\phi_i \times r^{\psi'}_i)}$. Given the (usually) large number of clique-separator products and in order to limit the growth of ϕ' internal ranks it is necessary to recompress each of these results using the **Compress** procedure.

Figure 6.12 illustrates such a process.

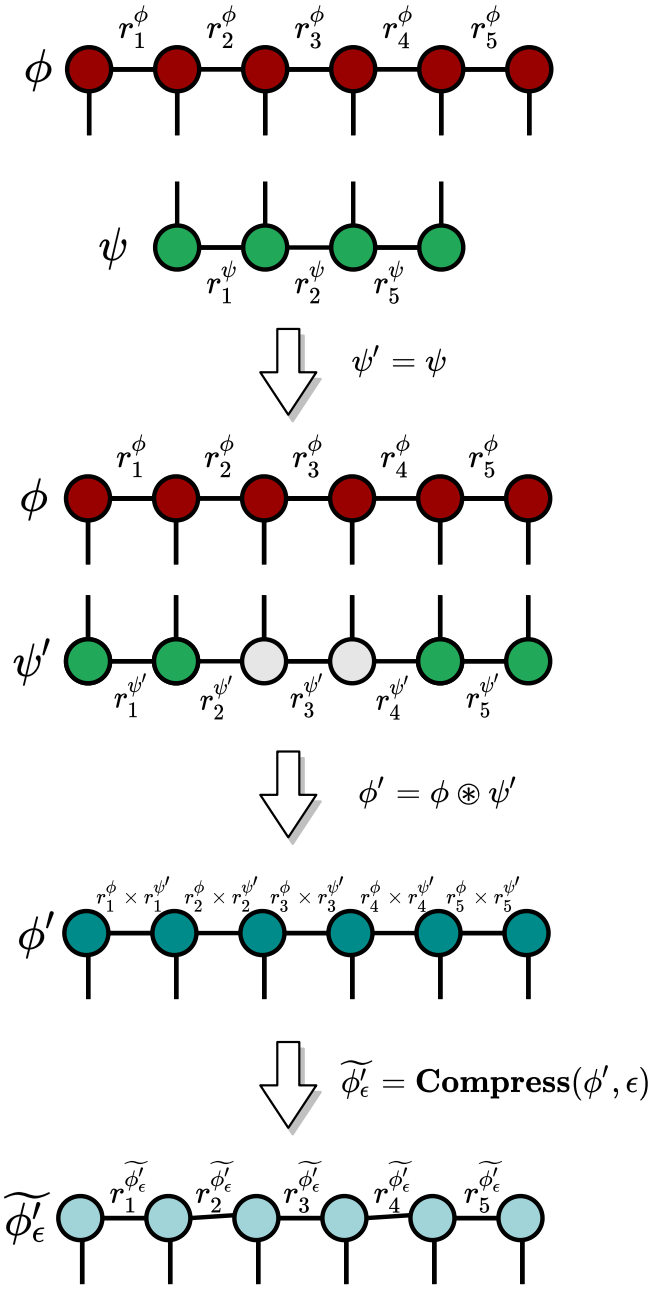


Figure 6.12: Sequence of operations during a clique-separator product of two tensors in the TT format, ϕ (the clique) and ψ (the separator). It holds that $r_1^\psi = r_1^{\psi'}, r_2^\psi = r_2^{\psi'} = r_3^{\psi'} = r_4^{\psi'}$ and $r_5^\psi = r_5^{\psi'}$. Furthermore, for all $i \in \{1, \ldots, 5\}$, $r_i^{\widetilde{\phi}_\epsilon} \leq r_i^\phi \times r_i^{\psi'}$.

6.3.2.2 Marginalization

Adapting the marginalization operation (3.20) to the case of potentials in the form of a tensor trains is more straightforward. Variables in a separator $\psi_{i\to j}$ between two cliques $\mathbf{C}_i$ and $\mathbf{C}_j$ being a subset of $\mathbf{C}_i$ and $\mathbf{C}_j$'s variables, we can marginalize the tensor ϕ associated with $\mathbf{C}_i$ in order to form $\psi_{i\to j}$. If ϕ has a representation in TT format with cores $\mathbf{T}^{(\nu)}$, $\nu \in \{1,\dots,d\}$ and $\psi_{i\to j}$ is a separator over $(i_1,...,i_{\nu-1},i_{\nu+1},...,i_d)$, then $\psi_{i\to j}$ has a representation in TT format with cores $\mathbf{T'}^{(\nu)} = \mathbf{T}^{(\nu)}$ for $\nu \in \{1,\dots,\nu-1,\nu+2,\dots,d\}$ and, for a right marginalization :

$$\mathbf{T'}^{(\nu+1)} = \sum_{i_\nu} \mathbf{T}^{(\nu)}(:,i_\nu,:) \times^1 \mathbf{T}^{(\nu+1)}$$

This operation is (usually) less expensive than its equivalent on full tensors. In the following algorithms we will denote by $\mathbf{Marginalize}(\phi,\psi_{i\to j})$ the operation that marginalize ϕ over the variables that are not in $\psi_{i\to j}$.

6.3.3 Shafer-Shenoy with Tensor Train Format

We can now redefine the Shafer-Shenoy algorithm, a classical message passing algorithm described in Section 3.3.3.1.3, using the TT format and our newly defined operations. Let $\mathcal{B} = (\vec{\mathcal{G}},\mathbb{P})$ be a Bayesian network with $\mathbb{P}$ characterized by $\Theta = \{\mathbb{P}(\mathrm{X}|\,\mathrm{Pa}_{\mathrm{X}}^{\vec{\mathcal{G}}}), \forall \mathrm{X} \in \mathcal{V}(\vec{\mathcal{G}})\}$. Before initializing a Junction Tree $\mathcal{T}$ according to $\mathcal{B}$, we can construct a new set of potentials Θ', that will be used to generate the potentials associated with each cliques (as picture in Figure 3.13), such that :

$$\Theta' = \{\theta'_{\mathrm{X}} | \theta'_{\mathrm{X}} = \mathbf{Compress}(\mathbb{P}(\mathrm{X}|\,\mathrm{Pa}_{\mathrm{X}}^{\vec{\mathcal{G}}}),\epsilon), \forall \mathrm{X} \in \mathcal{V}(\vec{\mathcal{G}})\}$$

This phase is not particularly expensive and could be done prior to inference (in order to store and reuse the tensorized model, for example). The following algorithms describe how our inferences work for a given Junction Tree $\mathcal{T}$, root r within such tree and tolerance ϵ.

CollectTT$(\mathcal{T}, i, j, \epsilon)$

Input : an initialized JT $\mathcal{T}$, $i, j \in \mathcal{V}(\mathcal{T})$ and a tolerance ϵ
Output: recursively computed $\psi_{i \to j}$
$\phi \leftarrow \phi_i$;
foreach $k \in \text{Adj}(i) \backslash \{j\}$ **do**
  CollectTT$(\mathcal{T}, k, i, \epsilon)$;
  $\phi \leftarrow$**Compress**$(\phi \circledast \psi_{k \to i}, \epsilon)$;
$\psi_{i \to j} \leftarrow$**Marginalize**$(\phi, \psi_{i \to j})$;

DistributeTT$(\mathcal{T}, i, j, \epsilon)$

Input : an initialized JT $\mathcal{T}$, $i, j \in \mathcal{V}(\mathcal{T})$ and a tolerance ϵ
Output : recursively computed $\psi_{i \to j}$
$\phi \leftarrow \phi_i$;
foreach $k \in \text{Adj}(i) \backslash \{j\}$ **do**
  $\phi \leftarrow$**Compress**$(\phi \circledast \psi_{k \to i}, \epsilon)$;
$\psi_{i \to j} \leftarrow$**Marginalize**$(\phi, \psi_{i \to j})$;
foreach $l \in \text{Adj}(j) \backslash \{i\}$ **do**
  DistributeTT$(\mathcal{T}, j, l)$;

Shafer-ShenoyTT$(\mathcal{T}, r, \epsilon)$

Input : a JT $\mathcal{T}$ of $\mathcal{B} = (\vec{\mathcal{G}}, \Theta')$, a root $r \in \mathcal{V}(\mathcal{T})$ and a tolerance ϵ
Output : a JT $\mathcal{T}$ with messages in both directions on all the separators
foreach $\mathrm{X} \in \mathcal{V}(\vec{\mathcal{G}})$ **do**
  Assign θ'_{X} to a clique $\mathbf{C}$ s.t $(\mathrm{X} \cup \mathrm{Pa}_{\mathrm{X}}^{\vec{\mathcal{G}}}) \subseteq \mathbf{C}$;
CollectTT$(\mathcal{T}, r, r, \epsilon)$;
DistributeTT$(\mathcal{T}, r, r, \epsilon)$;

This new version of the Shafer Shenoy algorithm is very close to the classical version based on the use of multidimensional arrays (see Appendix 7.2.2). If the various elementary operations are not as memory intensive as their full tensors counterpart, we will see that the systematic calls to the compression algorithm after each product is a limitation. However, we believe that this problem could be overcome, as suggested in [Kressner and Periša 2017].

6.4 Experimental Results

In order to obtain experimental results, we develop a first implementation using several packages : T3F [Novikov et al. 2018] [2] for the manipulation of tensors in TT format, TensorFlow Abadi et al. [2015] [3] for tensors related operations and pyAgrum for the manipulation of Bayesian networks, junction trees and potentials. But for stability and practicality reasons, we then switched to another package to manipulate tensors in the TT format, Tensap [Nouy and Grelier 2020][4]. Our code is available as a Python package called TenGeRine [5].

For the discussion, we compare 2 implementations : Shafer-Shenoy (called SS) and Shafer-Shenoy with Tensor Train format (called SSTT). The two implementations are identical as much as possible except that the first one manipulates potentials (model and operations implemented in C++) and the second one manipulates tensors in Tensor Train format (model and operations implemented in TensorFlow or Tensap with mixed python and C++). We compare both inference time (denoted T_{SS} for SS, T_{SSTT} for SSTT) as well as the number of parameters (in cliques and separators) at the end of each inferences ($\#_{SS}$ for SS, $\#_{SSTT}$ for SSTT) and the compression factor between them ($\frac{\#_{SS}}{\#_{SSTT}}$, denoted τ). All the tests have been performed on a dual E5-2630v2@2.60GHz with 32Go of RAM. We do not *yet* take advantage of the ease of calculation offered by tensor frameworks at the moment (GPGPU and parallelization).

6.4.1 Models from the Literature

We will first compare SSTT and SS on classical models from the literature [6]. Our objective here is to verify that the use of tensor trains does allow to reduce the memory space without introducing too many errors due to approximation in the calculated potentials.

[2] `https://t3f.readthedocs.io`
[3] `www.tensorflow.org`
[4] `https://anthony-nouy.github.io/tensap/`
[5] `https://gitlab.com/agrumery/tengerine`
[6] `https://gitlab.com/agrumery/pgmrepository`

6.4.1.1 Inference Time and Compression Factor

Table 6.1 showcases how using our approach on classical BNs can improve memory usage of inferences, especially within complex networks. Instead of looking at the number of nodes/arcs in the models, we considered it more relevant to look at the size of the cliques, especially the largest ones, as this is the discriminant criterion for inferences based on junction trees.

BN	#C	d_{Cmax}	$\overline{d_C}$	ds_{Cmax}	$\overline{ds_C}$	T_r	τ
Alarm	27	9	**5.0**	5.8e+03	**9.6e+02**	114.6	0.86
Asia	6	6	**4.2**	64	**26**	91.9	0.6
Barley	36	16	9.0	9.6e+14	3.52e+13	1.71	**35.8**
Carpo	48	10	**3.6**	1.0e+03	**8.7e+01**	365.51	0.78
Child	17	6	**4.0**	1.3e+03	**2.4e+02**	152.05	0.72
Diabetes	337	12	7.8	1.6e+13	3.8e+11	41.31	**2.03**
Hailfinder	43	16	6.7	2.1e+09	5.6e+07	170.15	0.94
Insurrance	18	14	8.6	1.1e+07	1.1e+06	89.5	**3.21**
Link	592	68	14.1	4.9e+27	8.4e+24	207.1s	-
Mildew	29	16	9.1	2.7e+15	9.3e+13	2.4	**19.9**
Munin1	159	20	8.1	1.2e+13	1.1e+11	0.05	**2148.21**
Pigs	367	19	6.7	1.2e+09	8.5e+06	65	**37.79**
Water	19	19	10.1	6.5e+10	3.8e+09	3.5	**187.48**

Table 6.1: Inference time ratio T_r ($\frac{T_{SSTT}}{T_{SS}}$) and compression factor τ ($\frac{\#_{SS}}{\#_{SSTT}}$) for classical BNs ($\epsilon = 0.001$). The number of cliques (#C), their max (resp. mean) number of dimensions d_{Cmax} (resp. $\overline{d_C}$) as well as the maximum (resp. mean) number of parameters ds_{Cmax} (resp $\overline{ds_C}$) are indicated, singular values are highlighted in bold.

The TT format does not seem to be helpful when the JT have small cliques, like in Asia or Alarm. Indeed, when the cliques and separators are small (regarding their domain sizes ds_C), the tensor train format can increase the number of parameters needed to describe them. In the case of Asia, for example, the largest clique, is an order-6 tensor with only 64 values. It is consequently not really surprising to see the number of parameters grows when converting such a tensor into 6 order-3 tensors. Such results indicates that there should be a lower limit in the dimensionality of potentials from which using TT format would be counterproductive.

However, in case of complex network such as Munin1, the tensor train format greatly helps reducing the number of parameters (by a factor of 2148 !). In the case of Link (592 cliques, including one with 20 dimensions), our Shafer-Shenoy implementation couldn't finish the inference, due to a lack of memory (SSTT took 207 seconds). Memory savings don't seem to imply a reduction of the inference time, as Barley shows, but this is not surprising since, as mentioned above, our prototype does not use all possible optimizations related to the use of tensor frameworks. Furthermore, operations on TT are, in their actual form, quite complex and computationally expensive. Fortunately, we think there is a lot of room for improvement from an algorithmic point of view, as suggested in [Kressner and Periša 2017], where the authors propose effective strategies to limit the computational costs associated with the use of Hadamard products between tensors in the Tucker format.

6.4.1.2 Approximation Error

Let's recall that when using the **Compress** algorithm, the tolerance parameter ϵ is used within the successive truncations of SVDs to reduce the size of the initial tensor by removing as much as possible *irrelevant* information. This deletion introduces, consequently, an error in the calculations, which should be discussed.

In that regards we compare in Table 6.2 the exact value of each probability p in each posterior and its approximated version with SSTT $\hat{p}$ associated with the inferences from the table 6.1. We observe the absolute error $|p - \hat{p}|$ as well as the relative one $\frac{|p-\hat{p}|}{p}$. If other evaluation criteria a Kullback–Leibler divergence between p and $\hat{p}$ have been considered, they seemed to us to be the easiest ones to interpret.

While absolute errors seem to be contained for a tolerance of $\epsilon = 1e-3$, within the range of one percent, relative errors can be high, suggesting that small probabilities are poorly approximated. In the case of Mildew, for example, the maximum relative error occurs when a probability of $1,35e-11$ is approximated with $4,60e-08$. Fortunately, as shown in Figure 6.13, the lower the tolerance, the smaller the errors.

	Absolute error		Relative error	
BN name	**Maximum**	**Mean**	**Maximum**	**Mean**
Alarm	3,78e-04	4,10e-05	9,73e-03	3,53e-04
Asia	3,96e-04	1,98e-04	3,96e-02	4,56e-03
Barley	4,09e-04	2,18e-05	9,99e-01	9,04e-03
Carpo	7,76e-04	7,68e-05	9,80e-01	8,20e-03
Child	7,69e-05	3,69e-06	1,53e-03	3,43e-05
Diabetes	8,25e-04	5,46e-05	1,00e+00	7,27e-03
Hailfinder	2,76e-04	5,68e-06	8,09e-04	2,11e-05
Insurance	4,97e-04	4,13e-05	8,40e-01	1,10e-02
Mildew	2,43e-04	1,32e-05	3,40e+03	4,60e+00
Munin1	3,19e-03	2,30e-04	6,84e+02	1,16e+00
Pigs	3,33e-15	6,29e-16	1,33e-14	2,25e-15
Water	8,85e-04	6,88e-05	1,00e+00	3,27e-03

Table 6.2: Absolute ($|p - \hat{p}|$) and relative ($\frac{|p-\hat{p}|}{p}$) errors for classical BN ($\epsilon = 0.001$)

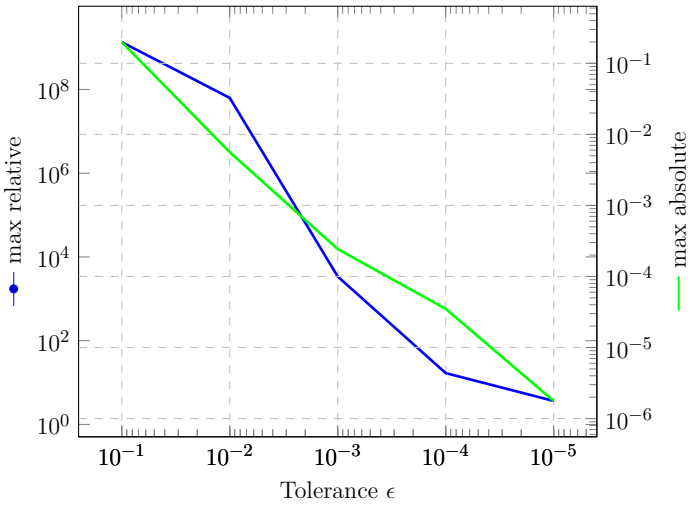


Figure 6.13: Evolution of maximum errors for Mildew according to multiple tolerance thresholds

The case of Pigs, showcased in the Figure 6.14, is very interesting. This model, complex by its size (and by the presence of a few large cliques) compresses very well (by a factor of 38) without introducing noticeable errors (in the order of machine epsilon).

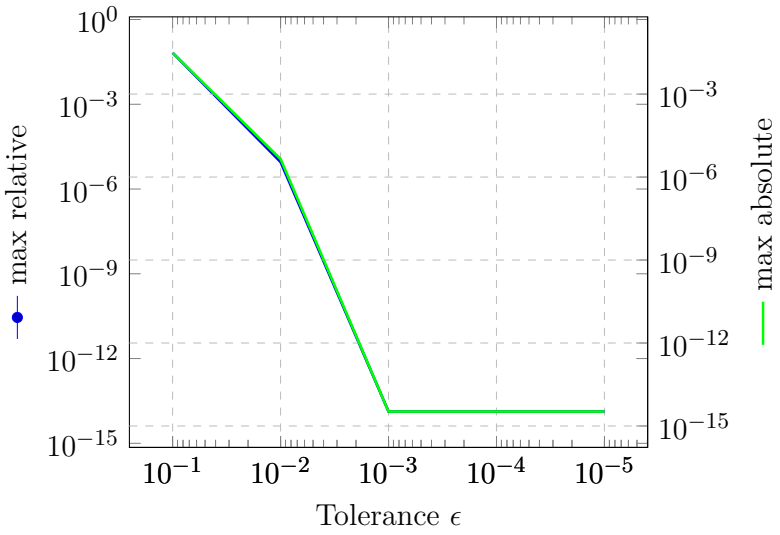


Figure 6.14: Evolution of maximum errors Pigs according to multiple tolerance thresholds

A first analysis suggests that this could be due to the fact that Pigs' CPTs are very deterministic: in this case, we expect the full tensors associated with each potentials to be sparse and therefore easily compressed with truncated SVDs.

If these first results are encouraging, we wanted to place ourselves in arbitrarily more complex cases to ensure the scalability of our approach.

6.4.2 SSTT Within Complex Networks

In order to generate BN with growing complexity, we relied on Dynamic Bayesian Network (dBN) [Dagum et al. 1992], dBNs are a generalisation of Markov Chain and can be seen as BNs that relate variables to each other over adjacent time steps (called time slices). Once the relation between two time slices is defined (in a graph called 2TBN), they can be easily deployed (unrolled) for a particular number of steps. When we create a junction tree from an unrolled dBN, the cliques tend to be very large, often making exact inference intractable [Murphy 2002].

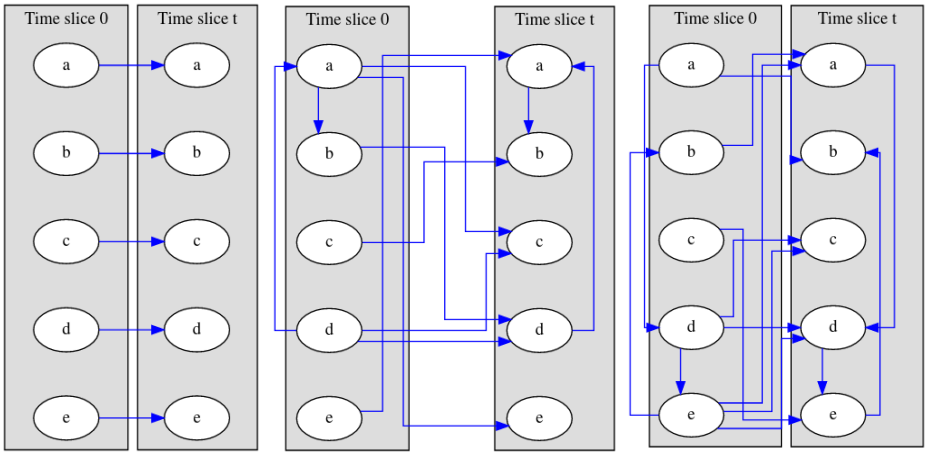


Figure 6.15: 2TBN of : (a) dBN_1, (b) dBN_2, (c) dBN_3

For our experiments, we have defined 3 dBNs, whose related 2TBN are presented in Figure 6.15. Each variable's domain size is 10 and CPT were randomly generated. The dBN_1 (Figure 6.15.a) is expected to be the worst case for our Tensor Train based algorithm : it is exactly five Markov chains and all the cliques have a size of 2 while dBN_2 and dBN_3 (Figure 6.15.b and 6.15.c) are growing in complexity, with more and more arcs intra/inter slices.

6.4.2.1 Inference Time and Memory Usage

Comparing inference time of Shafer-Shenoy and our algorithm confirms that the more complex and memory intensive an inference is, the more interesting is the usage of the Tensor Train format, as shown in the Table 6.3. In the case of dBN_1, a decomposition is unnecessary since the size of potentials won't change, only their number will linearly grow with the number of time slices. With dBN_2, on the other hand, the TT version of Shafer-Shenoy shows how much compressing potentials can help to scale inference. Finally, in the case of dBN_3, memory lacks are observed over 7 time slices for the standard algorithm when the TT version can easily scale linearly to 150 time slices. The limiting factor being the treewidth of the junction tree, using a low-rank representation for potentials, such as the TT format, greatly improves

memory usage, as shown in Table 6.4. For a tolerance factor ϵ of 0.05, the number of parameters in our model uses less than a hundredth of the space used by the exact one, with a maximum relative error of 0.12.

	$\mathbf{dBN}_1$	$\mathbf{dBN}_2$	$\mathbf{dBN}_3$	
Nb. Time slices	$\frac{T_{SSTT}}{T_{SS}}$	$\frac{T_{SSTT}}{T_{SS}}$	$\frac{T_{SSTT}}{T_{SS}}$	T_{SSTT}
4	179.89	61.07	7.79	2.95 s
5	69.14	38.83	1.46	4.31 s
7	69.47	1.47	-	**21.07 s**
15	64.24	**0.38**	-	**57.40 s**
50	55.40	**0.33**	-	**167.42 s**
75	45.97	**0.27**	-	**318.71 s**
100	34.59	**0.26**	-	**382.31 s**
150	32.68	**0.21**	-	**522.02 s**

Table 6.3: Ratio between inference times for SSTT and SS ($\epsilon = 0.05$)

Nb. Time slices	**SS**	**SSTT**	τ
4	3.33e+05	4.15e+04	8.0
5	1.04e+06	1.75e+05	5.9
7	1.71e+07	1.16e+05	147.8
15	1.09e+08	6.81e+05	160.5
50	5.05e+08	4.63e+06	109.1
75	8.01e+08	4.57e+06	175.4
100	1.07e+09	9.20e+06	115.9
150	1.66e+09	1.29e+07	128.8

Table 6.4: Number of parameters and compression factor (τ) in SS and SSTT (dBN_2, $\epsilon = 0.05$)

6.4.2.2 Error of Approximation using SSTT

To evaluate the errors introduced by the compression, we performed inferences ten times on dBN_2 with 50 time slices and randomized CPTs for each iteration. Figures 6.16 and 6.17 shows that, as expected, a lower tolerance ϵ in the **Compress** algorithm decreases the maximum relative and absolute errors but increases the inference time.

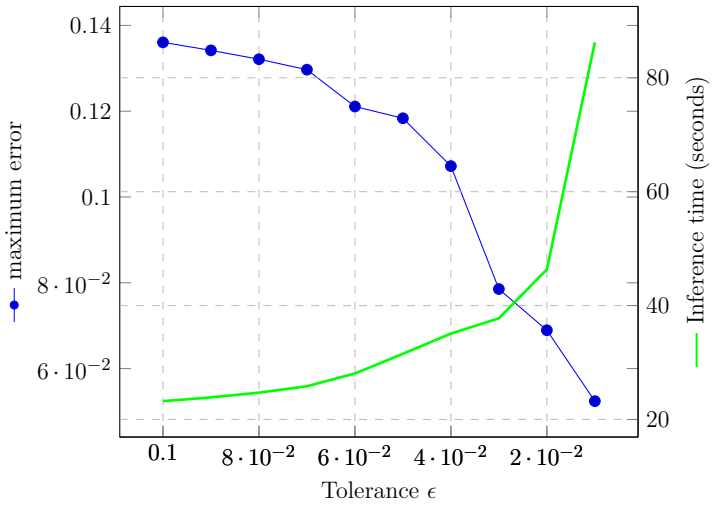


Figure 6.16: Evolution of the maximum relative error ($\frac{|p-\hat{p}|}{p}$) and inference time in dBN_2 with 50 timeslices

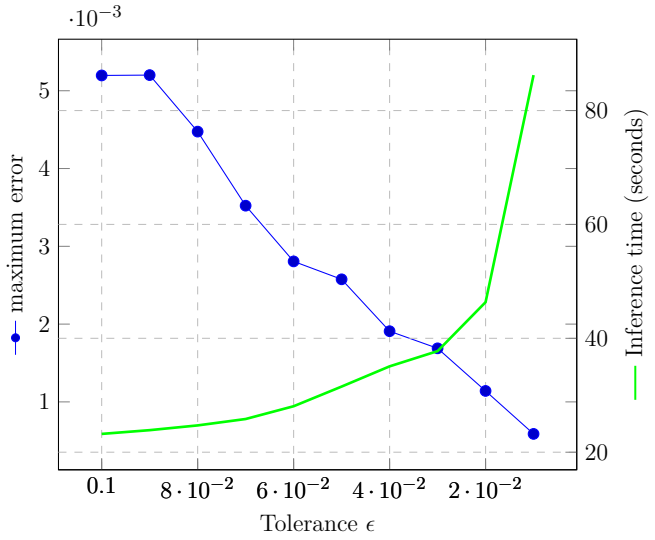


Figure 6.17: Evolution of the maximum absolute error ($|p-\hat{p}|$) and inference time in dBN_2 with 50 timeslices

Interestingly, Figure 6.18 shows that the maximum error is almost constant with the number of time slices (when one might have expected them to be amplified when the number of times slices increases). It seems to indicate that there is few error propagation along the JT branches.

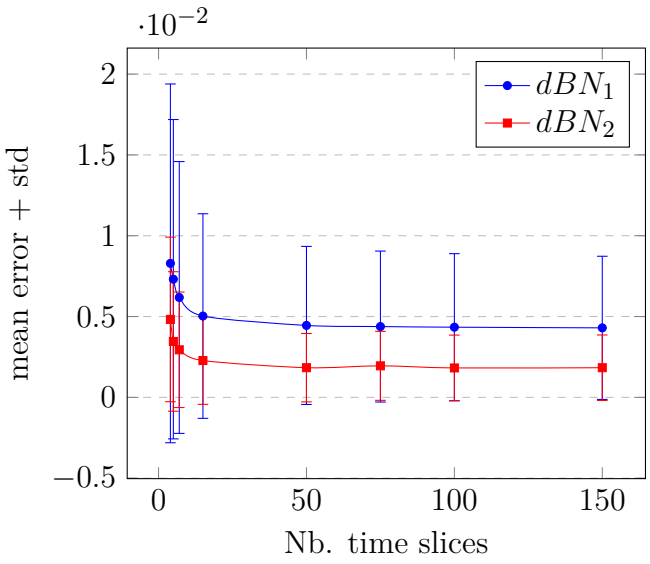


Figure 6.18: Evolution of the mean error ($\frac{|p-\hat{p}|}{p}$) in posteriors ($\epsilon = 0.05$)

These first experimental results on complex models are very encouraging; they tend to confirm that the use of low-rank tensor formats allow inferences to scale with a controllable error in cases where an exact inference would be impossible due to a lack of memory.

6.4.3 Comparison with Another Approximate Inference

In order to extend our comparison, we have compared our approach against another approximate inference, Loopy Belief Propagation (LBP) [Murphy et al. 1999], rather than sampling methods that offer no guarantee in terms of inferences times. The preliminary results presented in the table 6.5 show us two things :

- Our approach offers better results with respect to absolute error,
- A reduction of the tolerance used during truncation allows us to have overall much better relative errors than LBP on this set of models. When LBP's results better best it is usually only marginally, with the exception of Mildew whose very small marginals are noised by our approximation.

	Maximum absolute error			Maximum relative error		
BN	**1e-3**	**1e-5**	**LBP**	**1e-3**	**1e-5**	**LBP**
Alarm	3.78e-04	1.38e-06	2.85e-02	9.73e-03	4.29e-05	3.48e-01
Asia	3.96e-04	6.40e-09	1.10e-03	3.96e-02	1.42e-08	2.27e-03
Barley	4.09e-04	1.30e-06	1.58e-02	9.99e-01	1.11e-02	1.08e+01
Carpo	7.76e-04	7.28e-06	8.63e-03	9.80e-01	8.57e-03	1.24e-01
Child	7.69e-05	5.50e-09	5.69e-03	1.53e-03	1.55e-08	3.46e-02
Diabetes	8.25e-04	1.77e-05	2.28e-02	1.00e+00	9.99e-01	2.65e-01
Hailfinder	2.76e-04	5.45e-07	1.87e-03	8.09e-04	2.17e-06	6.91e-03
Insurrance	4.97e-04	3.31e-06	1.57e-02	8.40e-01	2.52e-01	1.03e-01
Mildew	2.43e-04	1.77e-06	4.41e-03	3.40e+03	3.64e+00	4.66e-01
Munin1	3.19e-03	1.28e-05	1.20e-02	6.84e+02	9.99e-01	4.44e-01
Pigs	3.33e-15	3.33e-15	4.45e-03	1.33e-14	1.33e-14	1.24e-02
Water	8.85e-04	5.20e-06	6.36e-04	1.00e+00	9.77e-01	1.25e-01

Table 6.5: Maximum errors between SSTT with multiple tolerances and LBP

Furthermore, and although LBP is faster than our algorithm, it does not guarantee to converge and, in fact, does not allow to estimate the quality of the approximation. These first results are encouraging, if the use of our algorithm is, in the current state, more time-consuming than this classical approximate inference algorithm, it seems nevertheless to propose a scalable and controlled approximation.

6.4.4 Aggregators in the TT Format

This section is the result of a prospective (unpublished) work with Anthony Nouy with whom we worked on [Ducamp et al. 2020]. So far we have considered the transformation of a full tensor into a tensor in the tensor train format. However, as we have seen in Section 5.4, aggregators can be described in a functional way in order to avoid exhaustive elicitation of their parameters which would quickly make their manipulation impossible. In this section we will propose, for a certain number of aggregators —or type of aggregators—, an explicit and exact representation in the TT format.

Let's recall that, for each random variable X, we denote with Val(X) the (finite, in our case) set of values that X can take. Moreover, if a node Y in a BN is an aggregator, the number of its parents $\mathrm{X}_1, \ldots, \mathrm{X}_d$ is indeterminate at the time of modelling and there is a function

$$\mathcal{A} : \mathrm{Val}(\mathrm{Y}) \times \bigtimes_{i=1}^{d} \mathrm{Val}(\mathrm{X}_i) \to [0,1]$$

such that :

$$\mathbb{P}(\mathrm{Y} = y | \mathrm{X}_1 = x_1, \cdots, \mathrm{X}_d = x_d) = \mathcal{A}(y, x_1, \cdots, x_d) \tag{6.7}$$

We can distinguish, in a general way, several types of aggregations.

➢ Deterministic aggregators

As mentioned earlier, deterministic aggregators are probabilistic aggregators whose output values are certain given their parents, *i.e.* , $\exists f : \bigtimes_{i=1}^{d} \mathrm{Val}(\mathrm{X}_i) \to \mathrm{Val}(\mathrm{Y})$ then

$$\mathcal{A}_f(y, x_1, \cdots, x_d) = \mathbb{1}_{y=f(x_1,\cdots,x_d)} \tag{6.8}$$

All logic gates are of this type. More generally, we distinguish self-decomposable aggregators, which we discussed in the section 5.4 (min, max, or, sum, etc) and deterministic aggregators that are not (l-out-of-k, median, etc).

➢ Non deterministic aggregators

Contrary to the former, the outputs of the so-called *non deterministic* aggregators can take any value in $[0,1]$. In order to avoiding the problem of specifying large CPTs, a common trick is to assume some model of interaction among causes (parent influences) that defines the effect's (child node's) CPT. The most popular class of model in this category is based on the concept known as causal independence or independence of causal influences (ICI) [Heckerman and Breese 1996; Zagorecki and Druzdzel 2006]. It includes gates such as the noisy-OR and the noisy-AND.

Certain other types of probabilistic aggregators should be noted, in particular, if Y is binary ($\mathrm{Val}(\mathrm{Y}) = \{0,1\}$), a type of probabilistic aggregator seems relevant:

$$\exists \gamma : \bigtimes_{i=1}^{d} \mathrm{Val}(\mathrm{X}_i) \to [0,1], \gamma(x_1, \cdots, x_d) = \mathbb{P}(\mathrm{Y} = 1 | \mathrm{X}_1 = x_1, \cdots, \mathrm{X}_d = x_d)$$

And thus,

$$\mathcal{A}_\gamma(y, x_1, \cdots, x_d) = y \cdot \gamma(x_1, \cdots, x_d) + (1-y) \cdot (1 - \gamma(x_1, \cdots, x_d)) \tag{6.9}$$

In this category, we can find :

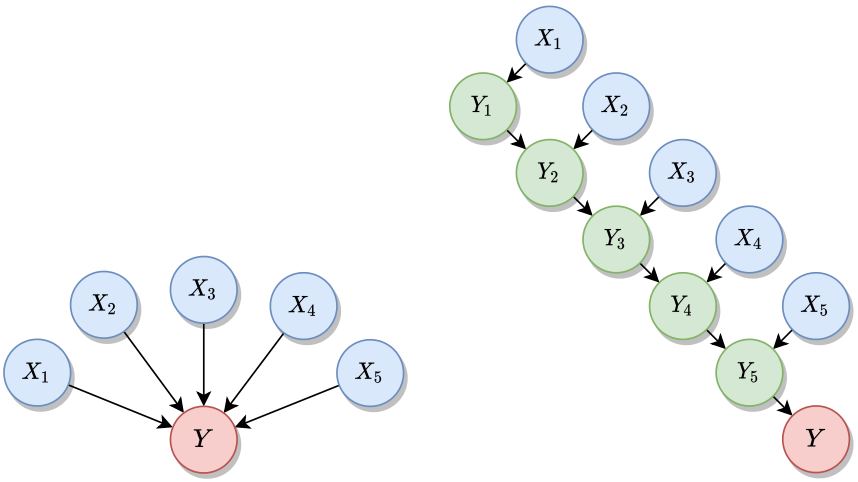
- Logistic regression : $\gamma(x_1, \cdots, x_d) = \frac{e^{w_0 + w \cdot x}}{1 + e^{w_0 + w \cdot x}}$
- Artificial neuron : $\gamma(x_1, \cdots, x_d) = \sigma(1 + w \cdot x)$ where $\sigma : \mathbb{R} \to [0,1]$ is a sigmoid-like function.

To show the interest of such an approach, going directly from a functional form to a tensor train based one, we decided to focus on decomposable aggregators and the memory gains from such a transformation insofar as they can be used in the context of probabilistic business rules that use the syntax defined in Chapter 5.

6.4.4.1 TT Factorization of a Self-Decomposable Aggregator

Let's recall that we defined an aggregation function f as a self-decomposable one if, for some merge operator $\diamond$ and all non-empty multisets $\mathbf{X}$ and $\mathbf{Y}$, $f(\mathbf{X} \uplus \mathbf{Y}) = f(\mathbf{X}) \diamond f(\mathbf{Y})$ where $\uplus$ denotes the multiset sum. The tensor train factorization is quite natural insofar as the structure of a comb-wised decomposed aggregator as shown in the Figure 6.19 recalls that of a tensor train —contrary to the tree-shaped one proposed in Section 5.4 that recalls tensors in the HT format (cf, Figure 6.7) —.

The purpose of this section is to show that it is possible to find an exact representation of a self-decomposable aggregators into tensor trains directly from their functional definition (*i.e.* , without their full CPTs).



(a) (b)

Figure 6.19: (a) a self-decomposable aggregator node with 5 parents. (b) the same aggregator after its decomposition

Proposition 2

A self-decomposable aggregator admits an exact representation in Tensor-Train format with ranks $(r_1, \ldots, r_d)$ *such that* $r_i \leq n_i$*, with* n_i *the number of possible values taken by* $\mathrm{Y}_i = f(\mathrm{X}_1, \ldots, \mathrm{X}_i)$*.*

Proof: Let $\mathrm{Y} = f(\mathrm{X}_1, \cdots, \mathrm{X}_d)$ and f being self-decomposable. We propose a train-like decomposition of this aggregator : let $\mathrm{Y}_1 = f(\mathrm{X}_1)$ and for all $0 \leq i < d, \mathrm{Y}_{i+1} = \mathrm{Y}_i \diamond f(\mathrm{X}_i) = f(\mathrm{Y}_i, \mathrm{X}_i)$. Finally $\mathrm{Y} = f(\mathrm{Y}_d)$.

$$\mathbb{P}(\mathrm{Y}|\mathrm{X}_1, \cdots, \mathrm{X}_d) = \sum_{\mathrm{Y}_1} \cdots \sum_{\mathrm{Y}_d} \mathbb{P}(\mathrm{Y}, \mathrm{Y}_1, \cdots, \mathrm{Y}_d|\mathrm{X}_1, \cdots, \mathrm{X}_d)$$

and

$$\begin{aligned} \mathbb{P}(\mathrm{Y}, \mathrm{Y}_1, \cdots, \mathrm{Y}_d|\mathrm{X}_1, \cdots, \mathrm{X}_d) \quad = \quad & \mathbb{P}(\mathrm{Y}_1|\mathrm{X}_1, \cdots, \mathrm{X}_d) \\ & \cdot \mathbb{P}(\mathrm{Y}_2|\mathrm{Y}_1, \mathrm{X}_1, \cdots, \mathrm{X}_d) \\ & \cdot \mathbb{P}(\mathrm{Y}_3|\mathrm{Y}_1, \mathrm{Y}_2, \mathrm{X}_1, \cdots, \mathrm{X}_d) \\ & \cdots \\ & \cdot \mathbb{P}(\mathrm{Y}|\mathrm{Y}_1, \cdots, \mathrm{Y}_d, \mathrm{X}_1, \cdots, \mathrm{X}_d) \end{aligned}$$

Since the Local Markov Property (Property 3.12) states that a variable

is independent from its non-descendants given its parents, it follows :

$$\mathbb{P}(\mathrm{Y}, \mathrm{Y}_1, \cdots, \mathrm{Y}_{d-1} | \mathrm{X}_1, \cdots, \mathrm{X}_d) = \mathbb{P}(\mathrm{Y}_1 | \mathrm{X}_1) \prod_{i=1}^{d-1} \mathbb{P}(\mathrm{Y}_{i+1} | \mathrm{Y}_i, \mathrm{X}_i) \cdot \mathbb{P}(\mathrm{Y} | \mathrm{Y}_d)$$

and consequently

$$\mathbb{P}(\mathrm{Y} | \mathrm{X}_1, \cdots, \mathrm{X}_d) = \sum_{\mathrm{Y}_1} \cdots \sum_{\mathrm{Y}_d} \mathbb{P}(\mathrm{Y}_1 | \mathrm{X}_1) \cdot \prod_{i=1}^{d-1} \mathbb{P}(\mathrm{Y}_{i+1} | \mathrm{Y}_i, \mathrm{X}_i) \cdot \mathbb{P}(\mathrm{Y} | \mathrm{Y}_d).$$

It ensues a TT-decomposition of any self-decomposable aggregator with tensors

$$\begin{cases} \mathbf{T}^{(1)}(1, \mathrm{X}_1, \mathrm{Y}_1) = \mathbb{P}(\mathrm{Y}_1 | \mathrm{X}_1) \\ \mathbf{T}^{(d+1)}(\mathrm{Y}_d, \mathrm{Y}, 1) = \mathbb{P}(\mathrm{Y} | \mathrm{Y}_d) \\ \forall 1 \leq i < d, \mathbf{T}^{(i)}(\mathrm{Y}_i, \mathrm{X}_i, \mathrm{Y}_{i+1}) = \mathbb{P}(\mathrm{Y}_{i+1} | \mathrm{Y}_i, \mathrm{X}_i) \end{cases} \tag{6.10}$$

■

Let us now look at the complexity of such a representation and more particularly for an exact one.

Let $N_{\mathcal{A}_f}$ denotes the complexity of an aggregator $\mathcal{A}_f$ in the TT format over $\{\mathrm{X}_1, \cdots, \mathrm{X}_d\}$. According to the Equation 6.4, it holds that :

$$N_{\mathcal{A}_f} = \sum_{i=1}^{d} |\operatorname{Val}(\mathrm{X}_i)| \cdot r_{i-1} \cdot r_i \tag{6.11}$$

with $r_0 = r_d = 1$ and $r_i = \operatorname{rank}_{\{1,\cdots,i\}}(\mathcal{A}_f)$.

Let $\mathrm{Y}_\alpha = f(\mathrm{X}_\alpha)$, denote by $\operatorname{Val}_\alpha$ the set of values taken by Y_α. We can find an upper bound for the the α-ranks of $\mathcal{A}_f$, with $\alpha = \{1, \ldots, i\}$, $1 \leq i \leq d$.

Proposition 3

For any $\alpha \subset \{1, \ldots, d\}$, we have

$$\operatorname{rank}_\alpha \mathcal{A}_f \leq |\operatorname{Val}_\alpha|$$

Proof: The result simply follows from the decomposition $A_f(x_1, \ldots, x_d, y) = \mathbb{1}_{f(x)=y} = \mathbb{1}_{f(x_\alpha) \diamond f(x_{\alpha^c}) = y} = \sum_{k \in \operatorname{Val}_\alpha} \mathbb{1}_{f(X_\alpha)=k} \mathbb{1}_{k \diamond f(x_{\alpha^c})=y}$.

■

Proposition 4

A self-decomposable aggregator $\mathcal{A}_f(x_1,\ldots,x_d,y) = \mathbb{1}_{y=f(x)}$ *admits an exact tensor train decomposition with a complexity* $N_{\mathcal{A}_f} = O(dNR^2)$ *where* $R = |\mathrm{Val(Y)}|$ *and* $N = max|\mathrm{Val(X}_i)|$ *for all* $i \in \{0,\cdots,d\}$.

Proof: From equation 6.11 and proposition 3 it holds that :

$N_{\mathcal{A}_f} = \sum_{i=1}^{d} |\mathrm{Val(X}_i)| \cdot r_{i-1} \cdot r_i$ and $\forall i, r_i = \mathrm{rank}_{\mathrm{Y}_i}(\mathcal{A}_f) \leq |\mathrm{Val}_i|$

and consequently that $N_{\mathcal{A}_f} = O(d \cdot max|\mathrm{Val(X}_i)| \cdot |\mathrm{Val(Y)}|^2)$ for all $i \in \{0,\cdots,d\}$. ■

The storage complexity of a self-decomposable aggregator in the TT format is therefore growing polynomially with its number of entries. Particularly, if the inputs and the aggregator are binary, the complexity of the aggregator is in $O(8 \cdot d)$ instead of in $O(2^{d+1})$!

It is therefore easy —and inexpensive— to switch from a functional representation to a representation in the form of a tensor train. If this transformation is still done through CPTs, as described in 6.10, they are of reasonable size, with at most $|\mathrm{Val(X)}| \cdot |\mathrm{Val(Y)}|^2$ parameters each. If not introducing auxiliary variables as we do in Section 5.4 is a good thing (the model does not change and does not lose readability) it is above all the consequent decrease of the memory complexity that makes this result so important and extending this logic to the noisy counterpart of such aggregators would be highly profitable.

6.5 Conclusion and Discussion

We have shown the interest of using such a compressed representation for potentials and although a lot of theoretical and practical work remains to be done (addressed in the next section), we truly believe that this approach is promising. As we have seen in Section 3.4, PRMs are particularly well suited to modelling systems on an industrial scale, complex networks can therefore be generated very quickly and the question of the choice of an

appropriate inference algorithm arises. Furthermore, we have seen that the syntax proposed in Chapter 5 allows business users to build very easily — perhaps even without knowing it— aggregators that are very complex to represent and evaluate. When used during the evaluation of probabilistic rules, our tensor train based approach could help to take over in the most complex cases. Moreover, in these circumstances, the modification of the user model adds variables whose CPTs are very deterministic (predicates) and our first experimental results lead us to believe that such sparse structures are well compressed using tensor trains.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Bachmayr, M., Schneider, R., and Uschmajew, A. (2016). Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations. *Foundations of Computational Mathematics*, pages 1–50.

Bridgeman, J. C. and Chubb, C. T. (2017). Hand-waving and interpretive dance: an introductory course on tensor networks. *Journal of Physics A: Mathematical and Theoretical*, 50(22):223001.

Carroll, J. D. and Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319.

Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., and Mandic, D. P. (2016). Tensor networks for dimensionality reduction and large-scale optimization: Part 1 Low-rank tensor decompositions. *Foundations and Trends® in Machine Learning*, 9(4–5):249–429.

Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., Sugiyama, M., and Mandic, D. P. (2017). Tensor networks for dimensionality reduction and large-scale optimization: Part 2 Applications and future perspectives. *Foundations and Trends® in Machine Learning*, 9(6):249–429.

Dagum, P., Galper, A., and Horvitz, E. (1992). Dynamic Network Models for Forecasting. In *Uncertainty in Artificial Intelligence*.

Ducamp, G., Bonnard, P., Nouy, A., and Wuillemin, P.-H. (2020). An efficient low-rank tensors representation for algorithms in complex probabilistic graphical models. In Jaeger, M. and Nielsen, T. D., editors, *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, pages 173–184. PMLR.

Falcó, A., Hackbusch, W., and Nouy, A. (2018). Tree-based tensor formats. *SeMA Journal*.

Gelß, P. (2017). *The Tensor-Train Format and Its Applications*. PhD thesis, Freien Universität Berlin.

Hackbusch, W. (2019). *Tensor Spaces and Numerical Tensor Calculus*. Springer Series in Computational Mathematics. Springer International Publishing.

Hackbusch, W. and Kuhn, S. (2009). A new scheme for the tensor representation. *Journal of Fourier analysis and applications*, 15(5):706–722.

Harshman, R. (1970). Foundations of the parafac procedure: Models and conditions for an "explanatory" multi-model factor analysis.

Heckerman, D. and Breese, J. (1996). Causal independence for probability assessment and inference using bayesian networks. 26(6):826–831.

Hillar, C. and Lim, L.-H. (2013). Most tensor problems are np-hard.

Hitchcock, F. L. (1927). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189.

Ji, Y., Wang, Q., Li, X., and Liu, J. (2019). A survey on tensor techniques and applications in machine learning. *IEEE Access*, PP:1–1.

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

Kressner, D. and Periša, L. (2017). Recompression of hadamard products of tensors in tucker format. *SIAM Journal on Scientific Computing*, 39:A1879–A1902.

Lee, N. and Cichocki, A. (2018). Fundamental tensor operations for large-scale data analysis using tensor network formats. *Multidimensional Systems and Signal Processing*.

Murphy, K. P. (2002). *Dynamic bayesian networks: Representation, inference and learning*. PhD thesis, University of California, Berkeley.

Murphy, K. P., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, page 467–475, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Nouy, A. (2017). Low-rank methods for high-dimensional approximation and model order reduction. In Benner, P., Cohen, A., Ohlberger, M., and Willcox, K., editors, *Model Reduction and Approximation: Theory and Algorithms*. SIAM, Philadelphia, PA.

Nouy, A. and Grelier, E. (2020). anthony-nouy/tensap v1.1.

Novikov, A., Izmailov, P., Khrulkov, V., Figurnov, M., and Oseledets, I. (2018). Tensor train decomposition on tensorflow (t3f). *arXiv preprint arXiv:1801.01928*.

Oseledets, I. V. (2009). A new tensor decomposition. *Doklady Mathematics*, 80(1):495–496.

Oseledets, I. V. (2011). Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317.

Oseledets, I. V. and Tyrtyshnikov, E. E. (2009). Breaking the curse of dimensionality, or how to use SVD in many dimensions. *SIAM Journal on Scientific Computing*, 31(5):3744–3759.

Penrose, R. (1971). Applications of negative dimensional tensors. *Combinatorial mathematics and its applications*, (1).

Savický, P. and Vomlel, J. (2007). Exploiting tensor rank-one decomposition in probabilistic inference. *Kybernetika*, 43(5):747–764.

Schollwöck, U. (2011). The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192.

Vomlel, J. and Tichavský, P. (2014). Probabilistic inference with noisy-threshold models based on a cp tensor decomposition. *International Journal of Approximate Reasoning*, 55(4):1072 – 1092. Special issue on the sixth European Workshop on Probabilistic Graphical Models.

Zagorecki, A. and Druzdzel, M. J. (2006). Probabilistic independence of causal influences. In Studený, M. and Vomlel, J., editors, *Third European Workshop on Probabilistic Graphical Models, 12-15 September 2006, Prague, Czech Republic. Electronic Proceedings*, pages 325–332.

Chapter 7

Conclusions and perspectives

7.1 Conclusions

During this thesis we investigated whether it was possible to define a more understandable and accessible syntax for probabilistic business rules (Chapter 5). Based on previous practical work and by modifying the compilation chain of an industrial BRMS, ODM, our prototype support this new syntax, on a sub-language of the system complete enough to be relevant to a business user. If such syntax is more business user friendly, its execution can be more challenging, especially because it supports the use of very memory-intensive forms of operation : aggregators. A first method (presented in Section 5.4), based on the graphical decomposition of the latter, allowed us to solve some instances of this problem in a fast and exact way, but it did not solve the problem in a general way.

We then explored the use of a new representation for the data used in probabilistic calculations based on low-rank decomposition of tensors. By compressing the information present in the potentials within so-called tensor trains we have shown that it is possible to calculate inferences in complex networks with a controlled approximation. We have shown that it is possible to convert the aggregators presented in the Section 5.4 from their functional form directly into tensors in TT format, which is a very encouraging result form the usability of such a method.

7.2 Perspectives

Many questions have been raised in this thesis, we hereby conclude this document with a non-exhaustive list of perspectives that could guide the developments of future works. We distinguish between those relating to the world of rules and those relating to PGMs, although these may be of common interest.

7.2.1 On the Rule Side

If our work has led to the development of a functional prototype consistent with the objective of making the use of probabilistic rules simpler for a business user, other extensions could go in this direction:

- **Set the threshold according to objectives**: ODM allows users to test their ruleset in order to evaluate the impact of their execution. As we said in the introduction of the Chapter 5, it might be possible to set the probabilistic threshold of our rule in relation to gain maximization, loss limit or other strategies. Given a ruleset and a test WM one could find the parameters that best fit these different objectives whose characterization could be more intuitive for a business user.

- **Learning the probabilistic models used**: For the moment, the user is asked to specify, using annotation, the probabilistic model used in his model. It could be interesting to propose a tool generating (learning) such a model given the elements present in the WM and their attributes. Such an approach could also help a modeller to identify (and correct) the selection biases discussed in the section 4.2.1 by identifying unannotated random variables.

Conversely, more complex execution modes exist for rules and proposing a probabilistic extension could enrich their modelling capabilities:

- **Probabilistic rules for complex event processing**: Some BRMS propose the execution of temporal rules [1], where one can follow the evolution of elements over time and parameterize the rules according to intervals of times or timestamp, as in the example in [2]. It would be interesting to try to adapt our approach to this problem known as Complex Event Processing, for example by proposing a temporal extension of PRMs.

[1] `https://www.ibm.com/support/knowledgecenter/en/SSQP76_8.10.x/com.ibm.odm.itoa.overview/topics/con_what_is_i2a.html`

[2] `https://www.ibm.com/support/knowledgecenter/en/SSQP76_8.10.x/com.ibm.odm.itoa.overview/topics/tpc_bike_hire_use_case.html`

7.2.2 On the PGM Side

Thanks to the implementation done in our proof of concept, we have already identified development axes linked to the use of techniques specific to tensor frameworks (GPU computing) and to the complexity of certain operations (recompression of elements after Hadamard products). Additional developments, just as interesting, are possible:

- **Hybrid inference**: We have seen that, in some cases, it was useless (even counterproductive) to transform multidimensional arrays into tensor trains. It could then be appropriate to "tensorize" only potentials with a dimension above a certain threshold. Such an *hybrid* inference would manipulate both multidimensional arrays and tensor trains and could help inferring when models become more complex when both the number and the sizes of cliques increases.

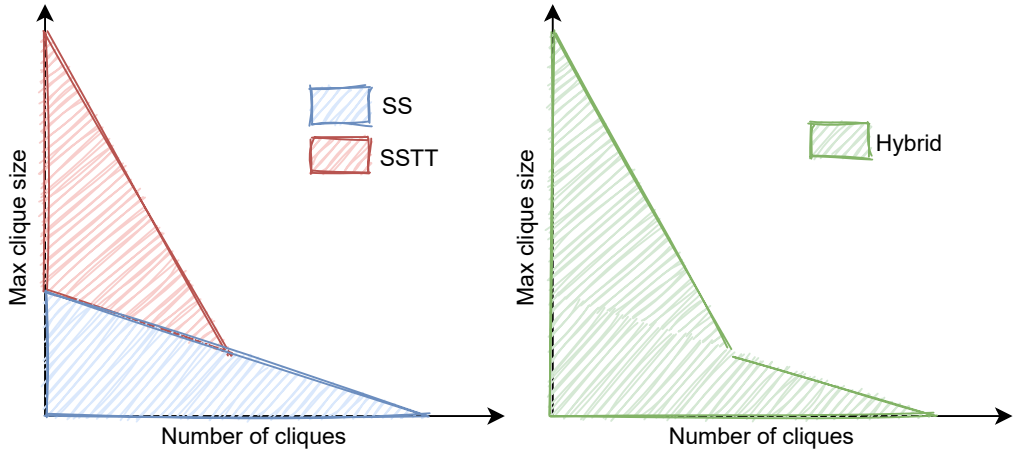


Figure 7.1: Theoretical cases where each inference would be appropriate

- ϵ **induced error bounding**: The use of the compression algorithm introducing an error, it might be possible to estimate, given the parameterized tolerance ϵ, an upper bound for it.
- **ICI models with TT**: Searching for an exact representation of such functions as tensor trains, similarly to what was done in section 6.4.4.1 for decomposable aggregators, would be relevant.

- **Memory constrained inference**: As we have seen it is possible to set parameters of the **Compress** algorithm to use a maximum size for the ranks of a tensor train instead of a tolerance ϵ. Given a memory limit dedicated to inference it might be possible to find the best representations of the tensor train. This approach could, for example, allow complex inferences to be made in embedded systems where resources are scarce or constrained.

- **Tensorized PRM specific inference**: For the moment we have experimented the use of TT for BNs specific inferences but, as we have seen at the end of part 3.4.2, it is possible to use the structural redundancy of PRM in order to limit the cost of such an operation. Consequently adapting such inferences to use a tensor trains framework could be useful in our context of use with BRMS.

- **Tree based tensor**: Tensor trains are only one particular form of a more general family of structures called tensor trees (which also include the Tucker and Hierarchical Tucker tensor formats). If the constrained form of the tensor train format helped us to redefine an inference by avoiding, once the potentials have been transformed, to manipulate their structure, others could help to further improve the compression and limit the induced errors.

Bibliography

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2015). TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Agli, H. (2017). *Uncertain reasoning for business rules*. PhD thesis, Université Pierre et Marie Curie - Paris VI.

Agli, H., Bonnard, P., Gonzales, C., and Wuillemin, P.-H. (2016). Incremental Junction Tree Inference. In *IPMU16*, Eindhoven, Netherlands.

Aho, A. V., Sethi, R., and Ullman, J. D. (1986). *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Longman Publishing Co., Inc., USA.

Ait-Kaci, H. and Bonnard, P. (2011). Probabilistic production rules. Technical report, IBM France Lab.

Andersen, S. K., Olesen, K. G., Jensen, F. V., and Jensen, F. (1989). Hugin - a shell for building bayesian belief universes for expert systems. In Sridharan, N. S., editor, *IJCAI*, pages 1080–1085. Morgan Kaufmann.

Arnborg, S. (1985). Efficient algorithms for combinatorial problems on graphs with bounded, decomposability—a survey. *BIT*, 25(1):2–23.

Arru, M. (2011). Introduction of probabilistic reasoning in business rules managmenet systems. Master's thesis, Polytech Nantes.

Bachmayr, M., Schneider, R., and Uschmajew, A. (2016). Tensor networks and hierarchical tensors for the solution of high-dimensional partial differential equations. *Foundations of Computational Mathematics*, pages 1–50.

Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Naur, P., Perlis, A. J., Rutishauser, H., Samelson, K., Vauquois, B., Wegstein, J. H., van Wijngaarden, A., and Woodger, M. (1963). Revised report on the algorithmic language ALGOL 60. *The Computer Journal*, 5(4):349–367.

Bangsø, O. and Wuillemin, P.-H. (2000). Object oriented bayesian networks a framework for topdown specification of large bayesian networks and repetitive structures.

Beazley, D. M. (1996). Swig: An easy to use tool for integrating scripting languages with c and c++. In *Proceedings of the 4th Conference on USENIX Tcl/Tk Workshop, 1996 - Volume 4*, TCLTK'96, page 15, USA. USENIX Association.

Biggio, B. and Roli, F. (2018). Wild patterns: Ten years after the rise of adversarial machine learning. *Pattern Recognition*, 84:317–331.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Bridgeman, J. C. and Chubb, C. T. (2017). Hand-waving and interpretive dance: an introductory course on tensor networks. *Journal of Physics A: Mathematical and Theoretical*, 50(22):223001.

Buchanan, B., Feigenbaum, E., and Lederberg, J. (1968). Heuristic dendral: A program for generating explanatory hypotheses in organic chemistry. *Machine Intelligence*, 4.

Buchanan, B. and Shortliffe, E. (1984). *Rule-based Expert System – The MYCIN Experiments of the Stanford Heuristic Programming Project.*

Butz, C. J., dos Santos, A. E., Oliveira, J. S., and Gonzales, C. (2016). A simple method for testing independencies in bayesian networks. In Khoury, R. and Drummond, C., editors, *Advances in Artificial Intelligence*, pages 213–223, Cham. Springer International Publishing.

Carroll, J. D. and Chang, J.-J. (1970). Analysis of individual differences in multidimensional scaling via an n-way generalization of "eckart-young" decomposition. *Psychometrika*, 35(3):283–319.

Chomsky, N. (1956). Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124.

Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., and Mandic, D. P. (2016). Tensor networks for dimensionality reduction and large-scale optimization: Part 1 Low-rank tensor decompositions. *Foundations and Trends® in Machine Learning*, 9(4–5):249–429.

Cichocki, A., Lee, N., Oseledets, I., Phan, A.-H., Zhao, Q., Sugiyama, M., and Mandic, D. P. (2017). Tensor networks for dimensionality reduction and large-scale optimization: Part 2 Applications and future perspectives. *Foundations and Trends® in Machine Learning*, 9(6):249–429.

Conrady, S. and Jouffe, L. (2015). *Bayesian Networks and BayesiaLab: A Practical Introduction for Researchers*. Bayesia USA.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using bayesian belief networks. *Artificial Intelligence*, 42(2):393 – 405.

Dagum, P., Galper, A., and Horvitz, E. (1992). Dynamic Network Models for Forecasting. In *Uncertainty in Artificial Intelligence*.

Dagum, P. and Luby, M. (1993). Approximating probabilistic inference in bayesian belief networks is np-hard. *Artificial Intelligence*, 60(1):141 – 153.

de Campos, L. M. and Castellano, J. G. (2007). Bayesian network learning algorithms using structural restrictions. *International Journal of Approximate Reasoning*, 45(2):233 – 254. Eighth European Conference on Symbolic and Quantitative Approaches to Reasoning with Uncertainty (ECSQARU 2005).

Dechter, R. (1999). Bucket elimination: A unifying framework for reasoning. *Artificial Intelligence*, 113(1):41 – 85.

Druzdzel, M. J. (1999). Smile: Structural modeling, inference, and learning engine and genie: A development environment for graphical decision-theoretic models. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, page 902–903, USA. American Association for Artificial Intelligence.

Ducamp, G., Bonnard, P., de Sainte Marie, C., Gonzales, C., and Wuillemin, P.-H. (2018). Improving probabilistic rules compilation using prm. In *RuleML+RR Doctoral Consortium 2018 (2nd International Joint Conference on Rules and Reasoning)*, Esch-sur-Alzette, Luxembourg.

Ducamp, G., Bonnard, P., De Sainte Marie, C., and Wuillemin, P.-H. (2020a). Advanced Syntax and Compilation for Probabilistic Production Rules with PRM. In *RuleML+RR Doctoral Consortium 2020 (4th International Joint Conference on Rules and Reasoning)*, volume 2644 of *CEUR Workshop Proceedings*, pages 103–110, Oslo, Norway. CEUR-WS.org. Virtual conference.

Ducamp, G., Bonnard, P., Nouy, A., and Wuillemin, P.-H. (2020b). An efficient low-rank tensors representation for algorithms in complex probabilistic graphical models. In Jaeger, M. and Nielsen, T. D., editors, *Proceedings of the 10th International Conference on Probabilistic Graphical Models*, volume 138 of *Proceedings of Machine Learning Research*, pages 173–184. PMLR.

Ducamp, G., Bonnard, P., and Wuillemin, P.-H. (2020c). Uncertain reasoning in rule-based systems using prm. In *Florida Artificial Intelligence Research Society Conference*.

Falcó, A., Hackbusch, W., and Nouy, A. (2018). Tree-based tensor formats. *SeMA Journal*.

Fenton, N., McLachlan, S., Lucas, P., Dube, K., Hitman, G., Osman, M., Kyrimi, E., and Neil, M. (2020). A privacy-preserving bayesian network model for personalised covid19 risk assessment and contact tracing. *medRxiv*.

Forgy, C. L. (1982). Rete: A fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17 – 37.

Geiger, D., Verma, T., and Pearl, J. (1989). d-separation: From theorems to algorithms. In *UAI*, pages 139–148.

Gelß, P. (2017). *The Tensor-Train Format and Its Applications*. PhD thesis, Freien Universität Berlin.

Gonzales, C., Torti, L., and Wuillemin, P.-H. (2017). aGrUM: a Graphical Universal Model framework. In *International Conference on Industrial Engineering, Other Applications of Applied Intelligent Systems*, Proceedings of the 30th International Conference on Industrial Engineering, Other Applications of Applied Intelligent Systems, Arras, France.

Graham, I. (2007). *Business Rules Management and Service Oriented Architecture: A Pattern Language*. Wiley.

Hackbusch, W. (2019). *Tensor Spaces and Numerical Tensor Calculus*. Springer Series in Computational Mathematics. Springer International Publishing.

Hackbusch, W. and Kuhn, S. (2009). A new scheme for the tensor representation. *Journal of Fourier analysis and applications*, 15(5):706–722.

Halpern, J. Y. (2017). *Reasoning about Uncertainty, second edition*.

Harshman, R. (1970). Foundations of the parafac procedure: Models and conditions for an "explanatory" multi-model factor analysis.

Hart, P. E., Duda, R. O., and Einaudi, M. T. (1978). Prospector-a computer-based consultation system for mineral exploration. *Journal of the International Association for Mathematical Geology*.

Hay, D. and Healy, K, a. (2000). Defining business rules - what are they really ? Technical report, Business Rules Group.

Heckerman, D. and Breese, J. (1996). Causal independence for probability assessment and inference using bayesian networks. 26(6):826–831.

Heckerman, D. and Shortliffe, E. (1992). From certainty factors to belief networks. *Artificial Intelligence In Medicine*.

Hillar, C. and Lim, L.-H. (2013). Most tensor problems are np-hard.

Hitchcock, F. L. (1927). The expression of a tensor or a polyadic as a sum of products. *Journal of Mathematics and Physics*, 6(1-4):164–189.

Jensen, F. V., Olesen, K. G., and Andersen, S. K. (1990). An algebra of bayesian belief universes for knowledge-based systems. *Networks*, 20(5):637–659.

Jesus, P., Baquero, C., and Almeida, P. (2015). A survey of distributed data aggregation algorithms. *IEEE Communications Surveys and Tutorials*, 17(1):381–404.

Ji, Y., Wang, Q., Li, X., and Liu, J. (2019). A survey on tensor techniques and applications in machine learning. *IEEE Access*, PP:1–1.

Knuth, D. E. (1964). Backus normal form vs. backus naur form. *Commun. ACM*, 7(12):735–736.

Knuth, D. E. (1968). Semantics of context-free languages. In *In Mathematical Systems Theory*, pages 127–145.

Kodaganallur, V. (2004). Incorporating language processing into java applications: A javacc tutorial. *IEEE Software*, 21(4):70–77.

Kokkinaki, A. I., Valavanis, K. P., and Tzafestas, S. G. (1993). *A Survey of Expert System Tools and Engineering-Based Expert Systems*, pages 366–378. Springer Berlin Heidelberg, Berlin, Heidelberg.

Kolda, T. G. and Bader, B. W. (2009). Tensor decompositions and applications. *SIAM Review*, 51(3):455–500.

Koller, D. and Friedman, N. (2009). *Probabilistic graphical models: principles and techniques*. MIT press.

Koller, D. and Pfeffer, A. (1997). Object-oriented bayesian networks. *Proceedings of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (UAI-97)*, pages 302–313.

Koller, D. and Pfeffer, A. (1998). Probabilistic frame-based systems. In *AAAI/IAAI*, pages 580–587.

Kressner, D. and Periša, L. (2017). Recompression of hadamard products of tensors in tucker format. *SIAM Journal on Scientific Computing*, 39:A1879–A1902.

Lauritzen, S. L., Dawid, A. P., Larsen, B. N., and Leimer, H.-G. (1990). Independence properties of directed markov fields. *Networks*, 20(5):491–505.

Lee, N. and Cichocki, A. (2018). Fundamental tensor operations for large-scale data analysis using tensor network formats. *Multidimensional Systems and Signal Processing*.

Lesk, M. and Schmidt, E. (1990). Lex—a lexical analyzer generator.

Mackay, D. J. C. (1998). *Introduction to Monte Carlo Methods*, pages 175–204. Springer Netherlands, Dordrecht.

Madiega, T. (2019). Eu guidelines on ethics in artificial intelligence: Context and implementation. Technical Report PE 640.163, European Parliamentary Research Service (ERPS).

Madsen, A. L. and Jensen, F. V. (1999). Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*.

Madsen, A. L. and Jensen, F. V. (2013). Lazy propagation in junction trees.

Masera, A., Zatonni, A., Mariano Olivera, M., Cortezzi, F., Saccarello dos Santos, M., and Patrone, J. (2015). Integrated approach to dam safety. Technical report, CESI-ISMES, CESI do Brasil Consultoria Ltda, Romagna Acque, Itapu Binacional, UTE.

Matsumoto, S., Carvalho, R., Ladeira, M., Costa, P., Santos, L., Silva, D., Onishi, M., Machado, E., and Cai, K. (2011). *UnBBayes: a Java Framework for Probabilistic Models in AI*.

Medina Oliva, G., Weber, P., Levrat, E., and Iung, B. (2010). Use of probabilistic relational model (PRM) for dependability analysis of complex systems. In *IFAC Proceedings Volumes (IFAC-PapersOnline)*.

Mekhnacha, K., Smail, L., Ahuactzin, J.-M., Bessière, P., and Mazer, E. (2006). A unifying framework for exact and approximate Bayesian inference. Research Report RR-5797, INRIA.

Murphy, K. P. (2002). *Dynamic bayesian networks: Representation, inference and learning*. PhD thesis, University of California, Berkeley.

Murphy, K. P., Weiss, Y., and Jordan, M. I. (1999). Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the Fifteenth Conference on Uncertainty in Artificial Intelligence*, UAI'99, page 467–475, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Neapolitan, R. E. (2003). *Learning Bayesian Networks*. Prentice-Hall, Inc., USA.

Nelson, W. (1982). Reactor: An expert system for diagnosis and treatment of nuclear reactor accidents. pages 296–301.

Ng, K. C. and Abramson, B. (1990). Uncertainty management in expert systems. *IEEE Expert-Intelligent Systems and their Applications*.

Nouy, A. (2017). Low-rank methods for high-dimensional approximation and model order reduction. In Benner, P., Cohen, A., Ohlberger, M., and Willcox, K., editors, *Model Reduction and Approximation: Theory and Algorithms*. SIAM, Philadelphia, PA.

Nouy, A. and Grelier, E. (2020). anthony-nouy/tensap v1.1.

Novikov, A., Izmailov, P., Khrulkov, V., Figurnov, M., and Oseledets, I. (2018). Tensor train decomposition on tensorflow (t3f). *arXiv preprint arXiv:1801.01928*.

Oseledets, I. V. (2009). A new tensor decomposition. *Doklady Mathematics*, 80(1):495–496.

Oseledets, I. V. (2011). Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317.

Oseledets, I. V. and Tyrtyshnikov, E. E. (2009). Breaking the curse of dimensionality, or how to use SVD in many dimensions. *SIAM Journal on Scientific Computing*, 31(5):3744–3759.

Pearl, J. (1986). Fusion, propagation, and structuring in belief networks. *Artificial Intelligence*, 29(3):241 – 288.

Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference (Morgan kaufmann series in representation and reasoning)*. Morgan Kaufmann Publishers, San Mateo, Calif.

Pearl, J. and Paz, A. (1986). Graphoids: Graph-based logic for reasoning about relevance relations or when would x tell you more about y if you already know z? In *Proceedings of the 7th European Conference on Artificial Intelligence - Volume 2*, ECAI'86, page 357–363. North-Holland.

Penrose, R. (1971). Applications of negative dimensional tensors. *Combinatorial mathematics and its applications*, (1).

Perez, K. (2013). Probabilistic production rules. Master's thesis, Ecole des Mines de Saint-Etienne.

Pfeffer, A. (2000). *Probabilistic reasoning for complex systems*. PhD thesis, Stanford University.

Rasmussen, A., Muratore, J., and Heindel, T. (1990). The inco expert system project: Clips in shuttle mission control.

Richardson, M. and Domingos, P. (2006). Markov logic networks. *Machine learning*, 62(1-2):107–136.

Robertson, N. and Seymour, P. (1986). Graph minors. ii. algorithmic aspects of tree-width. *Journal of Algorithms*, 7(3):309 – 322.

Ross, R. (2003a). Business rules manifesto. Technical report, The Business Rules Group.

Ross, R. (2003b). *Principles of the Business Rule Approach*. Addison-Wesley.

Savický, P. and Vomlel, J. (2007). Exploiting tensor rank-one decomposition in probabilistic inference. *Kybernetika*, 43(5):747–764.

Schollwöck, U. (2011). The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192.

Scientific Foresight Unit (STOA) (2019a). A governance framework for algorithmic accountability and transparency. Technical Report PE 642.262, European Parliamentary Research Service (ERPS).

Scientific Foresight Unit (STOA) (2019b). Understanding algorithmic decision-making: Opportunities and challenges. Technical Report PE 642.261, European Parliamentary Research Service (ERPS).

Shachter, R. D. (1998). Bayes-ball: Rational pastime (for determining irrelevance and requisite information in belief networks and influence diagrams). In *Proceedings of the Fourteenth Conference on Uncertainty in Artificial Intelligence*, UAI'98, page 480–487, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Shafer, G. and Pearl, J. (1990). *Readings in Uncertain Reasoning*. Morgan Kaufman Publishers.

Shenoy, P. P. and Shafer, G. (1990). Axioms for probability and belief-function propagation. In *Uncertainty in Artificial Intelligence*. Elsevier, Amsterdam.

Tony, M. (2002). *Business Rules and Information Systems: Aligning IT with Business Goals*. Addison-Wesley Professionel.

Torti, L. (2012). *Structured probabilistic inference in object-oriented probabilistic graphical models*. Theses, UPMC.

Torti, L. and Wuillemin, P.-H. (2010). Structured Value Elimination with D-Separation Analysis. In *International Florida Artificial Intelligence Research Society Conference*, pages 122–127, Daytona Beach, United States.

Torti, L., Wuillemin, P. H., and Gonzales, C. (2010). Reinforcing the object-oriented aspect of probabilistic relational models. In *Proceedings of the 5th European Workshop on Probabilistic Graphical Models, PGM 2010*.

Tzafestas, S. G. (1993). *An Overview of Expert Systems*. Springer Berlin Heidelberg.

Verner Jensen, F. and Jensen, F. (2013). Optimal junction trees. *arXiv*, pages arXiv–1302.

Vomlel, J. and Tichavský, P. (2014). Probabilistic inference with noisy-threshold models based on a cp tensor decomposition. *International Journal of Approximate Reasoning*, 55(4):1072 – 1092. Special issue on the sixth European Workshop on Probabilistic Graphical Models.

von Waldow, U. and Röhrbein, F. (2015). Structure learning in bayesian networks with parent divorcing.

Weber, P., Medina-Oliva, G., Simon, C., and Iung, B. (2012). Overview on bayesian networks applications for dependability, risk analysis and maintenance areas. *Engineering Applications of Artificial Intelligence*.

Wirth, N. (1996). *Compiler construction*. International computer science series. Addison-Wesley.

Wuillemin, P.-H. and Torti, L. (2012). Structured Probabilistic Inference. *International Journal of Approximate Reasoning*, 53(7):946–968.

Zadeh, L. (1965). Fuzzy sets. *Information and Control*, 8(3):338–353.

Zagorecki, A. and Druzdzel, M. J. (2006). Probabilistic independence of causal influences. In Studený, M. and Vomlel, J., editors, *Third European Workshop on Probabilistic Graphical Models, 12-15 September 2006, Prague, Czech Republic. Electronic Proceedings*, pages 325–332.

Zhang, C., Butepage, J., Kjellstrom, H., and Mandt, S. (2018). Advances in variational inference.

Zhang, N. and Poole, D. (1994). A simple approach to bayesian network computations. In *7th Canadian Conference on Artificial Intelligence*.

Appendices

Complex BN Example

Figure 2 illustrates a more extensive Bayesian network to better identify individuals suspected of being infected by a SARS-CoV. Although the number of symptoms taken into account is greater, it is mainly the history and the living environment of the person that allows the model to be more accurate.

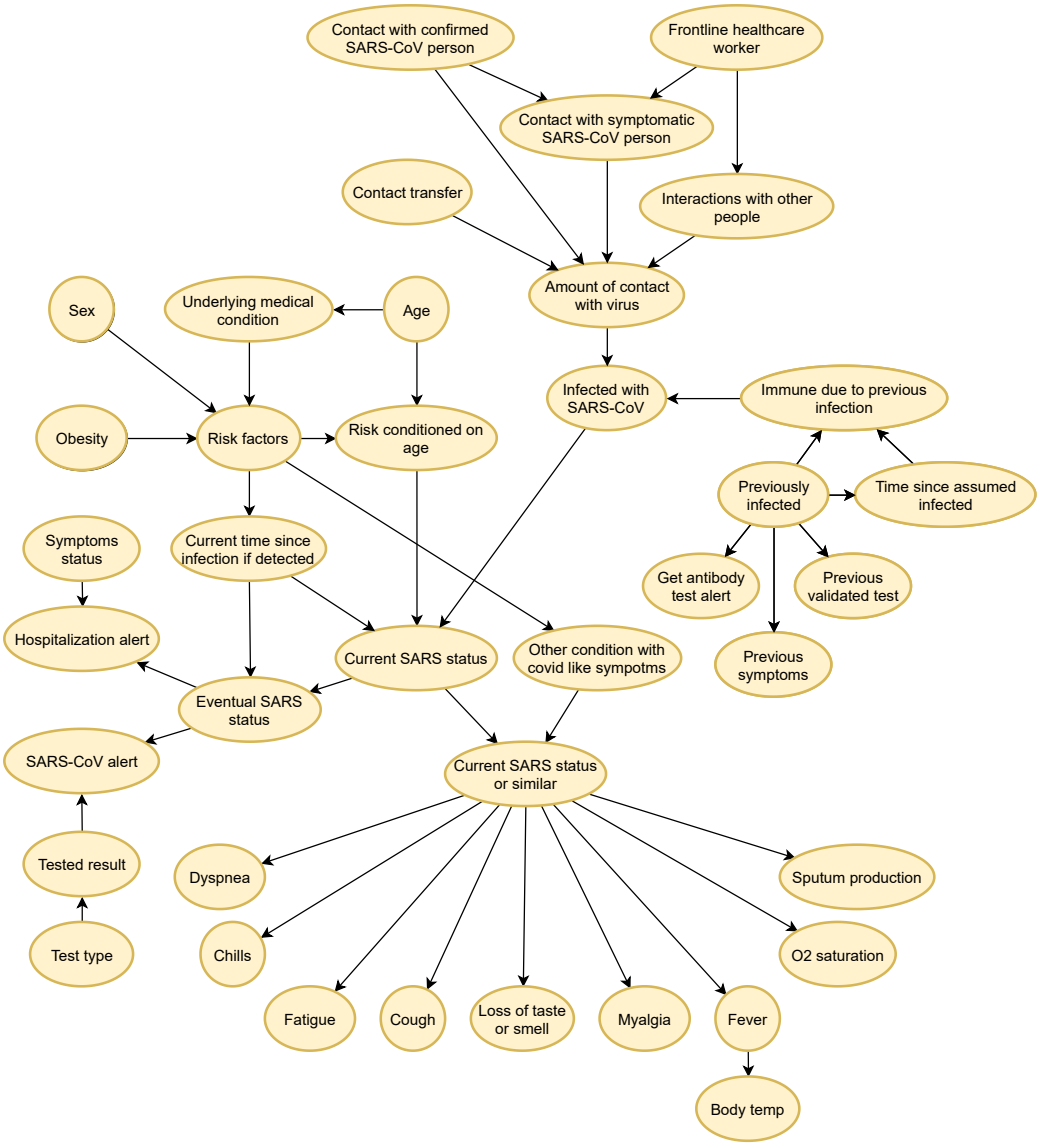


Figure 2: A more complex version of our SARS-CoV model

OM Annotations

These different annotations describe the constituent elements of the PRM that will be generated in the first part of the compilation phase (cf, Section 4.2.1). The different parameters are displayed.

➢ @PrmClass

Specify that the class exists in the PRM Model

```
@PrmClass
public class Physician {
    ...
}
```

➢ @PrmRestrictedTypeClass<*modalities*>

Define an attribute type using *modalities* as modalities.

```
@PrmRestrictedTypeClass(modalities={"severe","mild","asymp","none"})
public StatusType restricts int#[0,3];
```

➢ @PrmAttribute<*parents*, *cpt*>

Specify that the attribute exists as a PRM class in the PRM with the attributes in *parents* as parents and *cpt* as cpt.

```
@PrmAttribute(parents={"status"},cpt={{1,1,1,0},
                                      {0,0,0,1}})
public int is_severe;
```

➢ @PrmSingleReference

Specify that the reference is as a reference slot in the PRM

```
@PrmSimpleReference
public Physician physician;
```

➢ **@PrmMultiReference**

Specify that the reference is as a multiple reference slot in the PRM

```
@PrmMultiReference
public Patient[] pats;
```

➢ **@PrmAggregator<*aggName*, *attribute*, *modality*>**

Specify that the attribute is an aggregator in the PRM with *aggName* as aggregation function and uses the element in the reference slot *attribute* as parents. If a modality is required, *modality* can be used (to specify that an existential aggregation function is expected to be true, for example).

```
@PrmAggregator(aggName="sum",attribute ="pats.is_severe",mod="")
public int number_of_severe_patients;
```

Multidimensional array based Shafer-Shenoy pseudocode

Collect$(\mathcal{T}, i, j)$

Input : an initialized JT $\mathcal{T}$ and $i, j \in \mathcal{V}(\mathcal{T})$
Output : recursively computed $\psi_{i \to j}$
$\phi \leftarrow \phi_i$;
foreach $k \in \text{Adj}(i) \backslash \{j\}$ **do**
    Collect$(\mathcal{T}, k, i)$;
    $\phi \leftarrow \phi \times \psi_{k \to i}$;
$\psi_{i \to j} \leftarrow \sum_{X \in \mathbf{C}_i \backslash \mathbf{S}_{ij}} \phi$;

Distribute$(\mathcal{T}, i, j)$

Input : an initialized JT $\mathcal{T}$ and $i, j \in \mathcal{V}(\mathcal{T})$
Output : recursively computed $\psi_{i \to j}$
$\phi \leftarrow \phi_i \times \prod_{k \in \text{Adj}(i) \backslash \{j\}} \psi_{k \to i}$;
$\psi_{i \to j} \leftarrow \sum_{X \in \mathbf{C}_i \backslash \mathbf{S}_{ij}} \phi$;
foreach $l \in \text{Adj}(j) \backslash \{i\}$ **do**
    Distribute$(\mathcal{T}, j, l)$;

Shafer-Shenoy$(\mathcal{T}, r)$

Input : a JT $\mathcal{T}$ of $\mathcal{B} = (\vec{\mathcal{G}}, \Theta)$, a root $r \in \mathcal{V}(\mathcal{T})$
Output : a JT $\mathcal{T}$ with messages in both directions on all the separators
foreach $\text{X} \in \mathcal{V}(\vec{\mathcal{G}})$ **do**
    Assign $\mathbb{P}(\text{X}|\text{Pa}_\text{X})$ to a clique $\mathbf{C}$ s.t $(\text{X} \cup \text{Pa}_\text{X}^{\vec{\mathcal{G}}}) \subseteq \mathbf{C}$;
Collect$(\mathcal{T}, r, r)$;
Distribute$(\mathcal{T}, r, r)$;

List of Figures

List of Tables