University of São Paulo
Institute of Mathematics and Statistics
Bachelor of Computer Science

Tractable Probabilistic Description Logic

Algorithms and Implementation

Andrew Ijano Lopes

Final Essay

mac 499 — Capstone Project

Program: Computer Science

Advisor: Prof. Dr. Marcelo Finger

São Paulo
January 20th, 2021

Tractable Probabilistic Description Logic

Algorithms and Implementation

Andrew Ijano Lopes

Resumo

Andrew Ijano Lopes. **Lógica de Descrição Probabilística Tratável:** ***Algoritmos e Implementação***. Monografia (Bacharelado). Instituto de Matemática e Estatística, Universidade de São Paulo, São Paulo, 2020.

Elemento obrigatório, constituído de uma sequência de frases concisas e objetivas, em forma de texto. Deve apresentar os objetivos, métodos empregados, resultados e conclusões. O resumo deve ser redigido em parágrafo único, conter no máximo 500 palavras e ser seguido dos termos representativos do conteúdo do trabalho (palavras-chave). Deve ser precedido da referência do documento.

Palavras-chave: Lógicas de descrição. Palavra-chave2. Palavra-chave3.

Abstract

Andrew Ijano Lopes. **Tractable Probabilistic Description Logic: *Algorithms and Implementation***. Capstone Project Report (Bachelor). Institute of Mathematics and Statistics, University of São Paulo, São Paulo, 2020.

Elemento obrigatório, elaborado com as mesmas características do resumo em língua portuguesa. De acordo com o Regimento da Pós-Graduação da USP (Artigo 99), deve ser redigido em inglês para fins de divulgação. É uma boa ideia usar o sítio www.grammarly.com na preparação de textos em inglês.

Keywords: Description logics. Keyword2. Keyword3.

Lista de Abreviaturas

DL	Description Logic

Lista de Símbolos

List of Figures

List of Tables

List of Programs

Contents

Todo list

Chapter 1

Introduction

Need to rewrite, but the example is OK

mfinger: meus comentários em verde

Description logics are a family of formal knowledge representation languages, being of particular importance in providing a logical formalism for ontologies and the Semantic Web. Also, they are notable in biomedical informatics for assisting the codification of biomedical knowledge. Due to these uses, there is a great demand to find tractable (i.e., polynomial-time decidable) description logics.

One of them, the logic $\mathcal{EL}^{++}$, is one of the most expressive description logics in which the complexity of inferential reasoning is tractable (Baader *et al.*, 2005). Even though it is expressive enough to deal with several practical applications, there was also a need to model situations in which a General Concept Inclusion Axiom is not always true, which has already been proposed in the literature (Boole, 1854).

Example 1.1

> Consider a medical situation, in which a patient may have non-specific symptoms, such as high fever, cough, and headache. Also, COVID-19, a severe acute respiratory syndrome caused by the SARS-CoV-2 virus, is a disease that can account for those symptoms, but not all patients present all symptoms. Such an uncertain situation is suitable for probabilistic modeling.
>
> In a certain hospital, a patient with a high fever has some probability of having COVID-19, but that probability is 20% larger if the patient has a cough too. On the other hand, COVID-19 is not very prevalent and is not observed in the hospital 90% of the time. If those probabilistic constraints are satisfiable, one can also ask the minimum and maximum probability that a hospital patient Mary, with fever and cough, is a suspect of suffering from COVID-19.

For classical propositional formulas, this problem, called *probabilistic satisfiability* (PSAT), has already been presented with tractable fragments (Andersen and Pretolani, 2001). On the other hand, in description logics, most studies result in intractable reasoning;

moreover, by adding probabilistic reasoning capabilities to $\mathcal{EL}^{++}$, in order to model such situation, the complexity reaches NP-completeness (Finger, 2019).

To solve this problem, probabilistic constrains can be applied to axioms and its probabilistic satisfaction can be seen in a linear algebraic view. Furthermore, it can be reduced to an optimization problem, which can be solved by an adaptation of the simplex method with column generation. (Finger, 2019) Thus, it is possible to reduce the column generation problem to the *weighted partial maximum satisfatibility*.

Then, recent studies show that it is necessary to focus on a fragment of $\mathcal{EL}^{++}$ for obtain a tractable probabilistic reasoning. This fragment is called *Graphic* $\mathcal{EL}^{++}$ ($\mathcal{GEL}$) and it is defined as an $\mathcal{EL}^{++}$-fragment in which its set of axioms and *role inclusions* contains formulas in *normal form* and does not allow explicit conjunction axioms. Therefore, axioms can be seen as edges in a graph, as opposed to hyperedges in a hypergraph, which is the case of $\mathcal{EL}^{++}$. This allows the use of graph-based machinery to develop tractable algorithm for the *weighted partial Maximum SATisfatibility* for $\mathcal{GEL}$ (Max $\mathcal{GEL}$-SAT) and, as a result, a tractable probabilistic description logic.

1.1 Objectives

- Investigate a potentially tractable fragment of $\mathcal{EL}^{++}$;
- Study and implement tractable algorithms for the problem of *weighted partial* Max $\mathcal{GEL}$-SAT;
- Study and implement algorithms for the problem of *probabilistic satisfatibility for* $\mathcal{GEL}$ (PGEL-SAT), using the Max $\mathcal{GEL}$-SAT solver as a subroutine. Thus, it is excepted to achieve a tractable algorithm for a probabilistic description logic.

1.2 Structure of this work

In this paper, we describe the implementation of these algorithms [1] and is organized as follows: Section **??** highlights related results in the literature. The basic definition of $\mathcal{GEL}$ with its algorithms for MaxSAT and PSAT are described in Section **??** and followed by Section **??**, which presents details about the implementation and its experimental evaluation.

[1]Available at https://github.com/AndrewIjano/pgel-sat

Chapter 2

Background

In this chapter, we present the theoretical background of description logics,

Add initial description of the chapter

2.1 Description logics

Description logics (DLs) are used to represent knowledge, such as the semantic of words, people and their relations, and medical terms. These scenarios require precise specification and meaning so that different systems behave the same way. The first DL modeling languages appeared in the mid-1980s and have an important role in the context of the Semantic Web, an initiative to represent web content in a form that is more machine friendly (Krötzsch *et al.*, 2012).

As their name suggests, DLs are logics; indeed, most of them are fragments of first-order logic. This relation with logics is what provides their precise specification, called *formal semantics*. Also, it equips their languages with a formal deduction to *infer* additional information, and the computation of these inferences is called *reasoning*. The performance of algorithms for reasoning strongly relies on the expressiveness of the DL: fast algorithms usually exist for lightweight logics. Then, there is not just a single DL because the balance between expressiveness and performance depends on the application. (Krötzsch *et al.*, 2012)

2.1.1 Building blocks of description logic ontologies

A DL is composed of concepts, roles, and individual names. Concepts are sets of individuals, roles are binary relations between individuals and individual names are single individuals in the domain.

For example, an ontology modeling the situation in Example 1.1 can use the concepts Patient, to represent the set of all patients in the hospital, and Symptom, to represent the set of all symptoms; roles such hasSymptom, to represent the binary relation between

Maybe explain what is an ontology

patients and symptoms; and individual names such as mary and s1, to represent the individuals Mary and Mary's symptoms.

Additionally, DLs allows us to describe more complex situations, creating new concepts and roles from the previously defined ones.

Some concept constructors provide boolean operations similar to that found in set theory and logic expressions. For example, if we want to describe the set of individuals that are both fever and cough, we could use the *conjunction* operator, as follows

$$\mathsf{Fever} \sqcap \mathsf{Cough}.$$

We can link concepts and roles using role restrictions. For example, to describe all individuals that are suspect of some disease that is COVID-19, we use the *existential restriction*

$$\exists \mathsf{suspectOf}.\mathsf{COVID\text{-}19}.$$

Also, to define concepts with only one individual we use *nominals* like $\{\mathsf{mary}\}$.

More expressive logics can have other operations such as *disjunction* ($C \sqcup D$), *negation* ($\neg C$), *universal restriction* ($\forall r.C$) and *number restrictions* ($\leq n\, r.C$).

To capture knowledge about the world, DL ontologies also allow us to describe relations between concepts, roles, and individual names. For example, the fact that all fevers are symptoms is represented by the *concept inclusion*

$$\mathsf{Fever} \sqsubseteq \mathsf{Symptom};$$

the knowledge that someone that has a symptom which is caused by some disease is suspect of that disease can be expressed by the *role inclusion* with a *role composition*

$$\mathsf{hasSymptom} \circ \mathsf{hasCause} \sqsubseteq \mathsf{suspectOf};$$

and the fact that Mary is a patient of the hospital and has symptoms is represented by the *assertions* $\mathsf{Patient(mary)}$ and $\mathsf{hasSymptom(mary, s1)}$.

After that, if we have a set of these relations, one could ask if there is a set of individuals, or instances, that satisfies these relations, which is called an *interpretation.* Interpretations can be understood as the assignment of meaning to logical terms in an ontology. Because a DL usually considers all the possibles situations, property that is sometimes referred to as *open world assumption*, an ontology can have multiple satisfiable interpretations. The fewer restrictions it has, the more interpretations satisfy this ontology. The computational complexity to find the existence of these interpretations is one of the key aspects to choose different DL fragments.

These terms will be formally defined in the section 2.2, in the case of the DL $\mathcal{EL}^{++}$.

2.1.2 Description logic fragments and OWL

There are many DL fragments. Each subset of features, like those described previously, can lead to different fragments of first-order logic. For example, the logic $\mathcal{ALC}$ does not allow role inclusions and admits only $\sqcap, \sqcup, \neg, \exists$ and $\forall$ as concept constructors; their best reasoning algorithms, however, are worst-case exponential time. On the other hand, the $\mathcal{EL}$ logic allows only $\sqcap$ and $\exists$ as concept constructors, and its reasoning algorithms are polynomial time.

To express DL ontologies, the World Wide Web Consortium (W3C) designed the OWL 2 Web Ontology Language (OWL 2) (Hitzler *et al.*, 2009). This declarative language is part of the W3C's Semantic Web technology stack and comes with various syntaxes, such as RDF/XML. Because of this use on the web, names in OWL are *international resource identifiers* (IRIs).

2.2 The description logic $\mathcal{EL}^{++}$

$\mathcal{EL}^{++}$ is an extension of the DL $\mathcal{EL}$ (Baader *et al.*, 2005). It was created with large bio-health ontologies in mind, such as SNOMED-CT, the NCI thesaurus, and Galen, and became an official OWL 2 profile (Hitzler *et al.*, 2009). We concentrate on presenting $\mathcal{EL}^{++}$ without concrete domains.

2.2.1 Syntax

In $\mathcal{EL}^{++}$, *concept descriptions* are defined inductively from a set $\mathsf{N_C}$ of *concept names*, a set $\mathsf{N_R}$ of *role names* and set $\mathsf{N_I}$ of *individual names* as follows:

- $\top$, $\bot$ and concept names in $\mathsf{N_C}$ are concept descriptions;
- if C and D are concept descriptions, $C \sqcap D$ is a concept description;
- if C is a concept description and $r \in \mathsf{N_R}$, $\exists r.C$ is a concept description;
- if $a \in \mathsf{N_I}$, $\{a\}$ is a concept description.

To represent knowledge using concept descriptions, we need to define facts (axioms and role inclusions) and assertions.

An *axiom*, or a *general concept inclusion* (GCI), is an expression of the form $C \sqsubseteq D$, where C and D are concept inclusions. Also, we write $C \equiv D$ to represent the axioms $C \sqsubseteq D$ and $D \sqsubseteq C$. A *role inclusion* (RI) is an expression of the form $r_1 \circ \cdots \circ r_k \sqsubseteq r$, where $r_1, \ldots r_k, r \in \mathsf{N_R}$. The symbol "$\circ$" denotes composition of binary relations. A *constraint box* (CBox) is a finite set of GCIs and a finite set of RIs.

Similarly, a *concept assertion* is an expression of the form $C(a)$ and a *role assertion*, $r(a, b)$, where C is a concept description, $a, b \in \mathsf{N_I}$ and $r \in \mathsf{N_R}$. A finite set of concept assertions and role assertions is an *assertional box* (ABox).

Then, an $\mathcal{EL}^{++}$ *knowledge base* $\mathcal{K}$ (KB) is a pair $(\mathcal{C}, \mathcal{A})$, where $\mathcal{C}$ is a CBox and $\mathcal{A}$ is an ABox.

2.2.2 Semantics

The semantics of $\mathcal{EL}^{++}$ are given by *interpretations* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$. The *domain* $\Delta^{\mathcal{I}}$ is a non-empty set of individuals, and the *interpretation function* $\cdot^{\mathcal{I}}$ maps each concept name $A \in \mathsf{N_C}$ to a subset $A^{\mathcal{I}}$ of $\Delta^{\mathcal{I}}$, each role name r to a binary relation $r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$, and each individual name a to an element $a^{\mathcal{I}} \in \Delta^{\mathcal{I}}$. The extension of $\cdot^{\mathcal{I}}$ for an arbitrary concept description is inductively defined by the third column of Table 2.1.

Name	Syntax	Semantics
top	$\top$	$\Delta^{\mathcal{I}}$
bottom	$\bot$	$\emptyset$
nominal	$\{a\}$	$\{a^{\mathcal{I}}\}$
conjunction	$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$
existential restriction	$\exists r.C$	$\{x \in \Delta^{\mathcal{I}} \mid \exists y \in \Delta^{\mathcal{I}} : (x, y) \in r^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
GCI	$C \sqsubseteq D$	$C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$
RI	$r_1 \circ \cdots \circ r_k \sqsubseteq r$	$r_1^{\mathcal{I}} \circ \cdots \circ r_k^{\mathcal{I}} \subseteq r^{\mathcal{I}}$
concept assertion	$C(a)$	$a^{\mathcal{I}} \in C^{\mathcal{I}}$
role assertion	$r(a, b)$	$(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$

Table 2.1: *Syntax and semantics of $\mathcal{EL}^{++}$ without concrete domains*

The interpretation $\mathcal{I}$ *satisfies*:

- an axiom $C \sqsubseteq D$ if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (represented as $I \models C \sqsubseteq D$);
- a RI $r_1 \circ \cdots \circ r_k \sqsubseteq r$ if $r_1^{\mathcal{I}} \circ \cdots \circ r_k^{\mathcal{I}} \subseteq r^{\mathcal{I}}$ (represented as $I \models r_1 \circ \cdots \circ r_k \sqsubseteq r$);
- an assertion $C(a)$ if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (represented as $I \models C(a)$);
- an assertion $r(a, b)$ if $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in r^{\mathcal{I}}$ (represented as $I \models r(a, b)$).

Also, we say that $\mathcal{I}$ is a *model* of:

- a CBox $\mathcal{C}$ if it satisfies every axiom and RI in $\mathcal{C}$ (represented as $\mathcal{I} \models \mathcal{C}$);
- an ABox $\mathcal{A}$ if it satisfies every assertion in $\mathcal{A}$ (represented as $\mathcal{I} \models \mathcal{A}$);

Then, an important problem in $\mathcal{EL}^{++}$ is to determine its *consistency*, that is if $\mathcal{A}$ and $\mathcal{C}$ have a common model, which is in PTime (Baader *et al.*, 2005).

2.2.3 Normal form

We can convert an $\mathcal{EL}^{++}$ knowledge base into a normal form, in polynomial time, by introducing new concept and role names (Baader *et al.*, 2005).

First, there is no need of explicit ABox, because $\mathcal{I} \models C(a) \iff \mathcal{I} \models \{a\} \sqsubseteq C$ and $\mathcal{I} \models r(a, b) \iff \{a\} \sqsubseteq \exists r.\{b\}$. In other words, a knowledge base can be represented by just a CBox, by transforming assertions in axioms.

In addition, given a CBox $\mathcal{C}$, consider the set $\mathsf{BC}_{\mathcal{C}}$ of *basic concept descriptions*, which is the smallest set of concept descriptions that contains the top concept $\top$, all concept names used in $\mathcal{C}$ and all concepts of the form $\{a\}$ used in $\mathcal{C}$.

Then, every axiom can be represented in the following normal form, where $C_1, C_2 \in \mathsf{BC}_{\mathcal{C}}$, $D \in \mathsf{BC}_{\mathcal{C}} \cup \{\bot\}$:

$$C_1 \sqsubseteq D \qquad \text{(simple)}$$
$$C_1 \sqsubseteq \exists r.C_2 \qquad \text{(existential-head)}$$
$$\exists r.C_1 \sqsubseteq D \qquad \text{(existential-body)}$$
$$C_1 \sqcap C_2 \sqsubseteq D \qquad \text{(conjunctive-body)}$$

And every RI are of the form $r \sqsubseteq s$ or $r_1 \circ r_2 \sqsubseteq s$.

Example 2.1

Consider the following CBox $\mathcal{C}_{exa}$ representing the situation in Example 1.1. On the left, we have basic knowledge of diseases and, on the right, the specific knowledge about Mary. Note that, for simplicity, it is not in normal form.

$\mathsf{Fever} \sqsubseteq \mathsf{Symptom}$	$\{\mathsf{mary}\} \sqsubseteq \mathsf{Patient}$
$\mathsf{Cough} \sqsubseteq \mathsf{Symptom}$	$\{\mathsf{s1}\} \sqsubseteq \mathsf{Fever} \sqcap \mathsf{Cough}$
$\mathsf{COVID\text{-}19} \sqsubseteq \mathsf{Disease}$	$\{\mathsf{mary}\} \sqsubseteq \exists\mathsf{hasSymptom}.\{\mathsf{s1}\}$
$\mathsf{Symptom} \sqsubseteq \exists\mathsf{hasCause}.\mathsf{Disease}$	
$\mathsf{Patient} \sqsubseteq \exists\mathsf{hasSymptom}.\mathsf{Symptom}$	
$\mathsf{hasSymptom} \circ \mathsf{hasCause} \sqsubseteq \mathsf{suspectOf}$	

Because CBoxes can only represent facts, there is no way to describe uncertain knowledge. Even though, in cases when which of them are true, we could define three axioms

Ax_1 := $\mathsf{Fever} \sqsubseteq \exists\mathsf{hasCause}.\mathsf{COVID\text{-}19}$, when fever is actually caused by COVID-19;

Ax_2 := $\mathsf{Fever} \sqcap \mathsf{Cough} \sqsubseteq \exists\mathsf{hasCause}.\mathsf{COVID\text{-}19}$, when both fever and cough are caused by COVID-19;

Ax_3 := $\mathsf{COVID\text{-}19} \sqsubseteq \bot$, when there are no presence of COVID-19 in the hospital.

In the following sections, it will be presented how to add these axioms in a KB with probabilistic properties.

We want to model uncertain information using DLs. However, it has been proved that, by adding probabilistic reasoning capabilities to $\mathcal{EL}^{++}$, the complexity reaches NP-completeness (Finger, 2019). Then, it is necessary to reduce the expressiveness of this language.

2.3 Graphic $\mathcal{EL}$ ($\mathcal{GEL}$)

Graphic $\mathcal{EL}$ ($\mathcal{GEL}$) is a fragment of $\mathcal{EL}^{++}$ in which every axiom and RI are in normal form and only simple and existential-head axioms are allowed (Finger and Lopes, n.d.).

The semantics are the same as that of $\mathcal{EL}^{++}$.

Example 2.2

> Since there are conjunctive-body axioms in the CBox in Example 2.1, we need to modify this knowledge in order to represent it in $\mathcal{GEL}$. First, we substitute every concept description Fever ⊓ Cough by a new basic concept FeverAndCough. After that, we add axioms FeverAndCough ⊑ Fever and FeverAndCough ⊑ Cough to CBox.

The name of this fragment comes from the fact that each GCI can be represented as arrows in a graph where nodes are basic concepts. This representation is useful for the development of algorithms and it is used to define its SAT decision.

2.3.1 Graphical representation

Consider a $\mathcal{GEL}$-CBox $\mathcal{C}$ with n_R roles, its graphical representation is a edge-labeled graph $G(\mathcal{C}) := (N, E, \ell)$, where N is a set of nodes, $E \subseteq N^2$ is a set of directed edges and $\ell : E \longrightarrow \{0, 1, \dots, n_R\}$ is a labeling function.

In addition, Finger and Lopes (n.d.) defines the following notation. The set $E_i \subseteq E$ is the set of all edges e such that $\ell(e) = i$. We write $X_1 \longrightarrow_i X_2$ if there is an edge $(X_1, X_2) \in E_i$, and $X_1 \not\longrightarrow_i X_2$ if $(X_1, X_2) \notin E_i$. The expression $X \longrightarrow_i^* Y$ represents the reflexive transitive closure of $\longrightarrow_i$, which is the existence of a path in the graph of size ≥ 0, starting in X, ending in Y, and only going through edges in E_i, for $0 \leq i \leq n_R$. Finally, $X \rightsquigarrow Y$ represents a path from X to Y using any type of edge. Furthermore, we write $(u \longrightarrow_i v)$ to refer to an edge $e := (u, v)$ such that $\ell(e) = i$; and we write $(x \cdots u \longrightarrow_i v \longrightarrow_j w \cdots y)$ to refer to a path $(x, \dots, u, (u \longrightarrow_i v), v, (v \longrightarrow_j w), w, \dots, y)$.

Then, the graph $G(\mathcal{C}) = (N, E, \ell)$ can be constructed from a CBox $\mathcal{C}$ with the following steps:

1. the set N of nodes is obtained from the basic concepts of $\mathcal{C}$, an initial-node symbol $Init$ and the bottom concept $\perp$ as follows:

$$N := \{Init, \perp\} \cup \mathrm{BC}_{\mathcal{C}};$$

2. if $C \sqsubseteq D \in \mathcal{C}$ then $C \longrightarrow_0 D$;
3. if $C \sqsubseteq \exists r_i.D$ then $C \longrightarrow_i D$;
4. $Init \longrightarrow_0 \top$;[1]
5. for every node of the form $\{a_i\} \in N$, $Init \longrightarrow_0 \{a_i\}$.

Example 2.3

> Consider the CBox in Example 2.2 and the uncertain information in Example 2.1. Its graphical representation is displayed in the Figure 2.1. The $\longrightarrow_0$-edges are represented by continuous black arrows, the $\longrightarrow_i$-edges, $i \geq 1$, are represented by blue labeled arrows and red dotted arrows indicate that source is uncertain information.

[1]Talk about sat

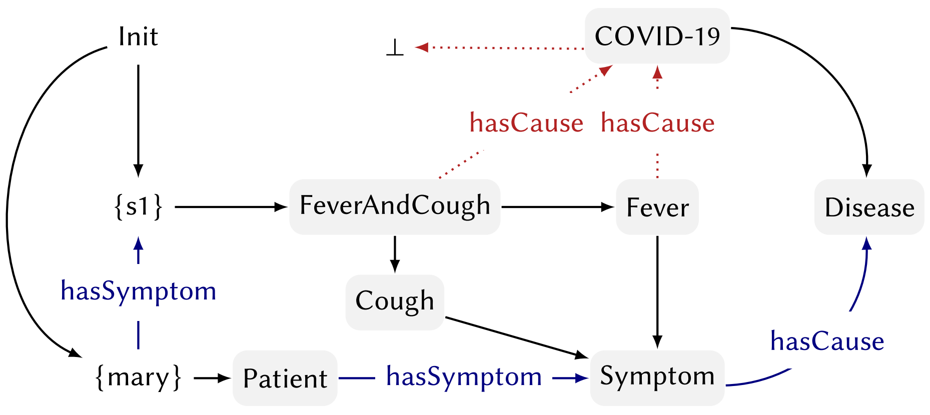

Figure 2.1: *Graphical representation of the ontology in Example 2.2.*

2.3.2 SAT decision

To calculate the satisfiability of a CBox $\mathcal{C}$, Finger and Lopes (n.d.) defined a completed graph $G^{\bullet}(\mathcal{C})$, obtained by applying the *graph completion rules* in Figure 2.2 until no more rules apply. Also, edges from the original graph $G(\mathcal{C})$ are called *basic edges* and the edges inserted from the rules in $G^{\bullet}(\mathcal{C})$ are called *derived edges.*

GC1 If $C \longrightarrow_0 C'$, $C' \longrightarrow_i D$ but $C \not\longrightarrow_i D$; insert $(C \longrightarrow_i D)$ into E_i;

GC2 If $C \longrightarrow_i C'$, $C' \longrightarrow_0 D$ but $C \not\longrightarrow_i D$; insert $(C \longrightarrow_i D)$ into E_i;

GC3 If $D \longrightarrow_0^* \bot$, $C \longrightarrow_i D$ but $C \not\longrightarrow^* D$; insert $(C \longrightarrow_0 D)$ into E_0;

GC4 If $C \longrightarrow_0^* \{a\}, D \longrightarrow_0^* \{a\}, Init \rightsquigarrow D$, $Init \rightsquigarrow C$ but $C \not\longrightarrow_0^* D$; insert $(C \longrightarrow_0 D)$ into E_0;

GC5 If $r_i \sqsubseteq r_j \in \mathcal{C}$, $C \longrightarrow_i$ but $C \not\longrightarrow_j D$; insert $(C \longrightarrow_j D)$ into E_j;

GC6 If $r_i \circ r_j \sqsubseteq r_k \in \mathcal{C}, C \longrightarrow_i D'$ and $D' \longrightarrow_j D$ but $C \not\longrightarrow_k D$; insert $(C \longrightarrow_k D)$ into E_k.

Figure 2.2: *Graph completion rules*

From Finger and Lopes (n.d.), we have the following lemma.

Lemma 1. *Given a $\mathcal{GEL}$-CBox $\mathcal{C}$, it is unsatisfiable iff $Init \longrightarrow_0^* \bot$ in the completed graph $G^{\bullet}(\mathcal{C})$. Furthermore, this decision can be made in polynomial time in $|\mathcal{C}|$, the number of symbols in $\mathcal{C}$.*

Lemma 2. *Given a $\mathcal{GEL}$-CBox $\mathcal{C}$, the following two statements are equivalent*

(a) $Init \rightsquigarrow \bot$ in $G(\mathcal{C})$;

(b) $Init \longrightarrow_0^* \bot$ in $G^{\bullet}(\mathcal{C})$, which is $G(\mathcal{C})$ after completion.

Proof. We break the proof in two parts.

(*a*) $\Longrightarrow$ (*b*). Suppose that we have $Init \rightsquigarrow \bot$ in $G(\mathcal{C})$. In particular, consider one path $P := (Init, e_1, \dots, e_m, \bot)$ in $G(\mathcal{C})$. Every edge e_j, $1 \le j \le m$, in this path or has a zero label ($\ell(e_j) = 0$) or a non-zero label ($\ell(e_j) \neq 0$).

Because the graph completion only add edges to the completed graph, there is also a path $P^{\bullet} := P$ in $G^{\bullet}(\mathcal{C})$.

By induction in the number k of non-zero labeled edges in $P^{\bullet}$, we are going to prove that for every $k \geq 0$ there is a path $P' := (Init, e'_1, \dots, e'_{m'}, \bot)$ in $G^{\bullet}(C)$ such that $\ell(e_j) = 0\ \forall j, 1 \leq j \leq m'$.

Base case: If $k = 0$, every edge in $P^{\bullet}$ is labeled with 0. Thus, there is nothing to prove.

Inductive case: Assume the induction hypothesis that the proposition is true for $k = n$.

Consider a path P with $n + 1$ non-zero labeled edges. So, there are vertices u, v such that $(Init, e_1, \dots, u, e, v, \dots, \bot)$, $\ell(e) \neq 0$ and $v \longrightarrow_0^* \bot$. That is, (u, v) is the last non-zero labeled edge in P.

By the *graph completion rule 3*, the completed graph $G^{\bullet}(C)$ will also have $u \longrightarrow_0 v$ by inserting an artificial edge e'. Then, we have a path $P' := (Init, \dots, u, e', v, \dots, \bot)$ in $G^{\bullet}(C)$, which has n non-zero labeled edges.

By the induction hypothesis and P', there is also a path P'' in $G^{\bullet}(C)$ such that every edge is labeled with 0, which completes the inductive step.

Then, we have that there is a path $P' := (Init, e'_1, \dots, e'_{m'}, \bot)$ in $G^{\bullet}(C)$ with every edge labeled with 0. That is, we have $Init \longrightarrow_0^* \bot$ in $G^{\bullet}(C)$.

$(a) \Longleftarrow (b)$. Suppose that we have $Init \longrightarrow_0^* \bot$ in $G^{\bullet}(C)$. In particular, consider one path $P^{\bullet} := (Init, e_1, \dots, e_m, \bot)$ such that $\ell(e_j) = 0\ \forall j, 1 \leq j \leq m$. This path may have derived edges, which are not in $G(C)$.

We can create a path P' in $G^{\bullet}(C)$ without derived edges using the following algorithm: If there is an derived edge $e := (u \longrightarrow_i v)$ in $P^{\bullet}$, it was derived by some graph completion rule N. Then, modify $P^{\bullet}$ following the *path recovering rule N* (PRN). At the end, we obtain a path P' without derived edges. Because there are a finite number of edges, this algorithm terminates.

Need to improve this because of PR5.

PR1 If e was inserted by rule GC1, there is a vertex u' such that $u \longrightarrow_0 u'$ and $u' \longrightarrow_i v$, for some $i \neq 0$.

Then, we substitute $(u \longrightarrow_i v)$ by $(u \longrightarrow_0 u' \longrightarrow_i v)$ in $P^{\bullet}$.

PR2 If e was inserted by rule GC2, there is a vertex u' such that $u \longrightarrow_i u'$ and $u' \longrightarrow_0 v$, for some $i \neq 0$.

Then, we substitute $(u \longrightarrow_i v)$ by $(u \longrightarrow_i u' \longrightarrow_0 v)$ in $P^{\bullet}$.

PR3 If e was inserted by rule GC4, we also have $u \longrightarrow_i v$, for some $i \neq 0$.

Then, we substitute $(u \longrightarrow_0 v)$ by $(u \longrightarrow_i v)$ in $P^{\bullet}$.

PR4 If e was inserted by rule GC5, we have a path P_v from $Init$ to v, because $Init \rightsquigarrow v$.

Then, we substitute $(Init, \cdots, u)$ by P_v in $P^{\bullet}$.

PR5 If e was inserted by rule GC6, e is of the form $(u \longrightarrow_j v)$, we have $r_i \sqsubseteq r_j \in C$ and $u \longrightarrow_i v$, for some $i, j \neq 0$.

Then, we substitute $(u \longrightarrow_j v)$ by $(u \longrightarrow_i v)$ in $P^{\bullet}$.

PR6 If e was inserted by rule GC7, e is of the form $(u \longrightarrow_k v)$, we have $r_i \circ r_j \sqsubseteq r_k \in \mathcal{C}$, and a vertex u' such that $u \longrightarrow_i u'$ and $u' \longrightarrow_j v$, for some $i, j, k \neq 0$.

Then, we substitute $(u \longrightarrow_k v)$ by $(u \longrightarrow_i u' \longrightarrow_j v)$ in $P^{\bullet}$.

Because P' does not have any derived edge, there is a path $P := P'$ in $G(\mathcal{C})$. That is, we have $Init \rightsquigarrow \bot$ in $G(\mathcal{C})$.

□

Theorem 1. *Given a $\mathcal{GEL}$-CBox $\mathcal{C}$, it is unsatisfiable iff $Init \rightsquigarrow \bot$ in $G(\mathcal{C})$. Furthermore, this decision can be made in polynomial time in $|\mathcal{C}|$, the number of symbols in $\mathcal{C}$.*

Proof. By Lemma 1 and Lemma 2, we have that a $\mathcal{GEL}$-CBox $\mathcal{C}$ is unsatisfiable iff $Init \rightsquigarrow \bot$ in $G(\mathcal{C})$. Also, by Lemma 1, the graph construction can be made in polynomial time, and finding a path in a graph is also computed in polynomial time, and so is the satisfiability decision. □

Then, we have that the unsatisfiability decision in $\mathcal{GEL}$ can be reduced to finding a path $Init - \bot$ in the graph $G(\mathcal{C})$.

2.4 MaxSAT for $\mathcal{GEL}$

Before focusing on the probabilistic extension for $\mathcal{GEL}$, we need to define its MaxSAT problem, which will compose further the probabilistic reasoner.

The *weighted partial maximum satisfiability problem for $\mathcal{GEL}$* ($\mathcal{GEL}$-MaxSAT) can be defined as follows: given a potentially inconsistent weighted CBox $\mathcal{C}$, we want to find the maximal satisfiable subset of axioms; since it is partial, some axioms must be present in this subset. Usually, partiality can be modeled assigning infinite weights to the axioms that must not be excluded.

A *weighted* CBox is a pair $\langle \mathcal{C}, w \rangle$ where $\mathcal{C}$ is a CBox and $w : \mathcal{C} \longrightarrow \mathbb{Q} \cup \{\infty\}$ is a weight function, which maps axioms in $\mathcal{C}$ to weights. The infinite weight is used to represent axioms that must not be excluded in the maximal satisfiable subset. Also, it is defined that RIs of the form $r \sqsubseteq s$ and $r_1 \circ r_2 \sqsubseteq s$ always have infinite weight.

Then, given a weighted CBox $\langle \mathcal{C}, w \rangle$, a solution for the weighted partial $\mathcal{GEL}$-MaxSAT problem is a set $\mathcal{C}_{max} \subseteq \mathcal{C}$ such that:

- $\mathcal{C}_{max}$ is satisfiable; and
- $\mathcal{C}_{max} \models C \sqsubseteq D$ if $w(C \sqsubseteq D) = \infty$; and
- the sum of finite weights in $\mathcal{C}_{max}$ is maximal.

To handle evenly finite and infinite weights, and given that $\mathcal{C}$ is finite, we could use an alternative definition. The solution for the weighted partial $\mathcal{GEL}$-MaxSAT problem is a satisfiable set C_{max} such that the sum of weights in the deleted set $C_{del} := C \setminus C_{max}$ is minimal and finite.

2.5 Probabilistic $\mathcal{GEL}$

Probability in $\mathcal{GEL}$ is constructed from a *probability function* P (Finger and Lopes, n.d.). Consider a finite number of interpretation, $\mathcal{I}_1, \dots, \mathcal{I}_m$, we define the probability function $P : \{\mathcal{I}_1, \dots, \mathcal{I}_m\} \longrightarrow \mathbb{Q}$, such that $P(\mathcal{I}_i) \geq 0$ and $\sum_{i=1}^{m} P(\mathcal{I}_i) = 1$. We can also define the probability of an axiom $C \sqsubseteq D$ as follows

$$P(C \sqsubseteq D) = \sum_{\mathcal{I}_i \models C \sqsubseteq D} P(\mathcal{I}_i).$$

A *probabilistic knowledge base* is a pair $\langle \mathcal{C}, \mathcal{P} \rangle$, where $\mathcal{C}$ is a CBox and $\mathcal{P}$ is a PBox. A PBox is a set of k linear constraints over n axioms, of the form

$$\sum_{j=1}^{n} a_{ij} \cdot P(C_j \sqsubseteq D_j) \leq b_i; \quad 1 \leq i \leq k. \tag{2.1}$$

We can define the *satisfiability problem* for this probabilistic KB (PGEL-SAT) as deciding if it is consistent or not. If it is consistent, the solution is a set of interpretations $\{\mathcal{I}_1, \dots, \mathcal{I}_m\}$ and a probability function $P : \{\mathcal{I}_1, \dots, \mathcal{I}_m\} \longrightarrow \mathbb{Q}^+$ such that $\sum_{i=1}^{m} P(\mathcal{I}_i) = 1$, $P(C \sqsubseteq D) = 1$ for $C \sqsubseteq D \in \mathcal{C}$ (axioms in CBox are certain) and P verifies all linear constraints in $\mathcal{P}$.

Example 2.4

Now we can model the uncertain situation stated in Example 1.1 using the probability knowledge base $\langle \mathcal{C}_{exa}, \mathcal{P}_{exa} \rangle$, where $\mathcal{C}_{exa}$ is the CBox from Example 2.2 and $\mathcal{P}_{exa}$ is given by

$$\begin{aligned} \mathcal{P}_{exa} := \{ \quad & P(Ax_2) - P(Ax_1) = 0.2, \\ & \qquad\qquad P(Ax_3) = 0.9 \quad \}. \end{aligned}$$

Then, we need a polynomial algorithm to find if this probabilistic KB is consistent.

2.5.1 Linear algebraic view

The PGEL-SAT problem was also defined by Finger and Lopes (n.d.) in a linear algebraic view, which is useful to develop its polynomial reasoning algorithm. It was shown that a probabilistic KB $\langle \mathcal{C}, \mathcal{P} \rangle$ is satisfiable iff the linear equation $C \cdot x = d$ has a solution $x \geq 0$, where

$$C := \begin{bmatrix} -I_n & M_{n\times m} \\ A_{k\times n} & 0_{k\times m} \\ 0'_n & 1'_m \end{bmatrix} \quad x := \begin{bmatrix} p_n \\ \pi_m \end{bmatrix} \quad d := \begin{bmatrix} 0_n \\ b_k \\ 1 \end{bmatrix} \tag{2.2}$$

and

- $A_{k \times n}$ is a $k \times n$ matrix whose elements a_{ij} are given by Equation 2.1;
- b_k is a k vector whose elements b_i are also given by Equation 2.1;
- $M_{n \times m}$ is a $n \times m$ matrix given by the following steps:

  Consider an interpretation $\mathcal{I}$ model of $\mathcal{C}$, also called $\mathcal{C}$-*satisfiable interpretation*, its corresponding vector in $\mathcal{P}$ is a $\{0, 1\}$-vector y such that $y_i = 1$ iff $\mathcal{I} \models C_i \sqsubseteq D_i$, for $1 \leq i \leq n$.

  Then, given a set of interpretations $\mathcal{I}_1, \ldots, \mathcal{I}_m$, we define $M_{n \times m}$ a matrix whose column M^j is $\mathcal{I}_j$'s corresponding vector in $\mathcal{P}$;
- I_n is the n-dimensional identity matrix;
- 0_n is a column 0-vector of size n (similarly for 1_n);
- $0'_n$ is the previous vector's transpose;
- $0_{k \times m}$ is a 0-matrix of shape $k \times m$;
- p_n is a vector of size n which corresponds to the probability of axioms occurring in Equation 2.1;
- π_m is a vector of size m which corresponds to the probability distribution over interpretations $\mathcal{I}_1, \ldots, \mathcal{I}_m$.

Intuition about this matrices

Thus, we can use techniques for solving linear equations to find a tractable algorithm for PGEL-SAT.

Chapter 3

Development

In this section, we describe the development of a tractable algorithm for PGEL-SAT[1], described by FINGER and LOPES (n.d.). It was implemented using Python programming language (VAN ROSSUM and DRAKE, 2009).

Numpy

3.1 Input and output format

The algorithm accepts as input a P$\mathcal{GEL}$ KB encoded in an OWL 2 ontology. Both certain and uncertain knowledge must be a $\mathcal{GEL}$-CBox in the normal form, with the additional support of equivalence axioms. Due to limitations in the OWL parser used, which will be detailed further, RDF/XML, OWL/XML and NTriples are the only file formats supported.

In addition, uncertain axioms must have an annotation (`rdfs:comment`) with its unique numerical index. That is, given an uncertain axiom Ax_i, its annotation must be of the following form in Figure 3.1.

```
#!pbox-id i
```

Figure 3.1: *Annotation format for an uncertain axiom*

PBox restrictions are represented with annotations in the $\top$ concept (`owl:Thing`). That is, given a restriction of the form $a_0P(Ax_0) + a_1P(Ax_1) + \cdots + a_{n-1}P(Ax_{n-1}) = b$, its annotation must be of the form in Figure 3.2. Also, inequalities $\leq$ and $\geq$ can be represented, respectively, with the symbols <= and >=.

The output of the algorithm is `True` if the given KB has a solution, and `False` if not.

Maybe explain how to execute the algorithm

[1]Available at https://github.com/AndrewIjano/pgel-sat.

```
#!pbox-restriction
0 $a_0$
1 $a_1$
...
$n-1$ $a_{n-1}$
==
$b$
```

Figure 3.2: *Annotation format for a PBox restriction*

3.2 Knowledge base representation

To read the ontology in OWL 2, it was used the Python module *Owlready2* (Lamy, 2017).

The probabilistic KB is represented as the edge-labeled graph in subsection 2.3.1 with three matrices for the inequalities in Equation 2.1.

Need to improve this

The probabilistic KB representation allows us to develop a PGEL-SAT solver, using this data structure as input.

3.3 PGEL-SAT solver

In order to understand the PGEL-SAT solver implemented, we need to find a solution for the linear system stated in subsection 2.5.1.

First, consider the matrix C in Equation 2.1, it has p-columns and π-columns, referring to which part of x they multiply. We say that a p-column pc_i is *well-formed* if it is of the form in Equation 2.1, starting with the i-th column of $-I_n$, followed by the i-th column of A and with one 0 at the end. Also, we say that a π-column is *well-formed* if it starts with a corresponding vector of an interpretation model of C, followed by k zeros and with one 1 at the end. A column that is not well-formed is *ill-formed*.

Now, consider that the matrix C may have ill-formed columns. We define a binary *cost vector* c such that each element $c_i = 1$ iff column C^i is ill-formed; otherwise $c_i = 0$. Then, the linear system stated in subsection 2.5.1 has a solution iff the following minimization problem has minimum 0.

$$\begin{aligned} \text{minimize} \quad & c' \cdot x \\ \text{subject to} \quad & C \cdot x = d \\ & x \geq 0 \end{aligned} \tag{3.1}$$

This problem can be solved by a linear algebraic solver but we need to find beforehand C and d that reaches the minimum 0. Since matrix A and vector b are generated from

PBox, we need to choose a set of interpretations that provides the solution. However, we potentially have an exponential collection of possible interpretations, and looking over all of them would make the algorithm untractable.

Need to say about M being large not the collection of possible columns

Even though, from Carathéodory's Theorem (Eckhoff, 1993), Finger and Lopes (n.d.) states that if constrains in Equation 2.1 are solvable then there exists a solution where x has at most $n + k + 1$ values such that $x_j > 0$. This allows us to avoid generating an exponential number of columns of C by using a linear algebraic technique called *column generation* (Gilmore and Gomory, 1961; Gilmore and Gomory, 1963). With this technique, we can create an algorithm PGEL-SAT-Solver which generates well-formed π-columns of C on the fly by solving an auxiliary problem and uses a linear solver in each iteration to verify if the solution for Equation 3.1 has reached minimum 0. This solution leads to the algorithm implemented, which is shown in Algorithm 1.

Algorithm 1 The PGEL-SAT solver algorithm

```
1:  function PGEL-SAT-Solver($\langle\mathcal{C}, \mathcal{P}\rangle$)
2:      $c \longleftarrow$ Initialize-c($\mathcal{P}$)
3:      $C \longleftarrow$ Initialize-C($\mathcal{P}$)
4:      $d \longleftarrow$ Initialize-d($\mathcal{P}$)

5:      $lp \longleftarrow$ LinProg($c, C, d$)

6:      while $lp.minCost \neq 0$ do
7:          $result \longleftarrow$ GenerateColumn($\langle\mathcal{C}, \mathcal{P}\rangle, lp$)
8:          if $result.isSuccess$ = False then
9:              return False                          ▷ $\langle\mathcal{C}, \mathcal{P}\rangle$ is unsatisfiable
10:         end if

11:         $y \longleftarrow result.column$              ▷ well-formed $\pi$-column
12:         $C \longleftarrow (C\,|\,y)$
13:         $c \longleftarrow (c\,|\,0)$
14:         $lp \longleftarrow$ LinProg($c, C, d$)
15:     end while

16:     return (True, $lp$)                           ▷ $\langle\mathcal{C}, \mathcal{P}\rangle$ is satisfiable
17: end function
```

The algorithm PGEL-SAT-Solver in Algorithm 1 receives a potentially unsatisfiable probabilistic knowledge base and returns *True* if it is satisfiable, and *False*, otherwise.

It starts with matrices c, C and d. Because the algorithm needs to start with a cost vector with non-zero elements, the C matrix is initialized with $n + k + 1$ ill-formed columns, represented by the identity matrix I_{n+k+1}, and well-formed p-columns as shown in Equation 2.2; therefore, the cost vector c is initialized with $n + k + 1$ 1-elements plus n 0-elements. The d vector is unaltered during the execution and is given by Equation 2.2. These initializations are shown in Algorithm 2.

In addition, the algorithm uses a program LinProg. It receives matrices c, C and d and

Algorithm 2 Matrices initializations

```
 1: function INITIALIZE-c($\mathcal{P}$)
 2:     return $( 1'_{n+k+1} \mid 0'_n )'$
 3: end function

 4: function INITIALIZE-C($\mathcal{P}$)
 5:     Get $A$ from PBox $\mathcal{P}$ restrictions
 6:     $PC \leftarrow \begin{pmatrix} -I_n \\ A \\ 1'_n \end{pmatrix}$        ▷ $p$-columns
 7:     return $( I_{n+k+1} \mid PC )$
 8: end function

 9: function INITIALIZE-d($\mathcal{P}$)
10:     Get $b$ from PBox $\mathcal{P}$ restrictions
11:     return $( 0'_n \mid b' \mid 1 )'$
12: end function
```

returns the solution for the optimization problem in Equation 3.1. Its output lp needs to contain the minimal cost ($lp.minCost$), the primal solution ($lp.primal$) and dual solution ($lp.dual$).

Then, the algorithm executes the column generation process while the minimal cost is not zero. In each iteration, a well-formed π-column y is generated by an auxiliary program GENERATECOLUMN. After that, the column is appended to C, with its respective 0 cost in c, and a new minimal cost is calculated. If no more columns can be generated with a minimal cost higher than 0, represented by $result.isSuccess = False$, the KB is unsatisfiable.

3.4 Linear solver

The program LINPROG was implemented using the open-source GNU linear program kit (MAKHORIN, 2001) and the Python wrapper for this library, the *swiglpk* package (SONNENSCHEIN, n.d.). The library was chosen because it is open-source, has tractable algorithms for large-scale problems, and provides a solution that contains the required elements described in section 3.3.

There are several methods for solving a linear optimization problem. The *simplex method* is known to be very efficient in practice (BORGWARDT, 2012) although its worst time complexity is untractable (KLEE and MINTY, 1972). On the other hand, the modern *interior point method* has a proven polynomial-time complexity and is especially ideal for very sparse problems (BOYD *et al.*, 2004) but may not be as efficient as the simplex method in all cases.

Because of the theoretical tractability, the linear solver LINPROG implemented uses the interior point method.

3.5 Column generation

The program GenerateColumn is responsible for generating a well-formed π-column for C that reduces $c'x$ until it is minimal; in particular, when the KB is satisfiable, this value is equal to 0. To find such a column, we need to analyze the optimization problem we are solving from the theory of linear programming (Bertsimas and Tsitsiklis, 1997).

3.5.1 Cost reduction

Suppose that the matrix C is composed of I_{n+k+1} and all well-formed p-columns PC, which are result of Initialize-C, concatenated with all possible (ill and well formed) π-columns $y^{(j)}$, as illustrated below.

$$C := \left(I_{n+k+1} \,|\, PC \,|\, y^{(1)} \,|\, y^{(2)} \cdots \right)$$

This matrix can be rearranged in the form $C = (B\,|\,N)$, where B is a linear independent square matrix called *basis matrix*. This allows obtaining a *basic feasible solution* $x = (\, x'_B \,|\, x'_N \,)'$ where $x_N = 0$ and $x_B = B^{-1}d$. In particular, if $B = I_{n+k+1}$ we have $x_B = d$.

However, this solution may, and probably will, not be optimal. Then, we need to find some column C^j in N and increase x_j by some positive value θ, moving x to $x + \theta s$, for some s. We create this vector s such that $s_j = 1$, $s_i = 0$ for every non basic index $i \neq j$ and its basic part $s_B = (s_{B(1)}, s_{B(2)}, \ldots, s_{B(m)})$.

Since we want feasible solution, we need $C(x + \theta s) = d$; given that x is feasible, we also have $Cx = d$. Thus, we need $C \cdot s = C \cdot (x + \theta s - x) = d - d = 0$. Also, because $s_j = 1$ and $s_i = 0$ for every non basic index $i \neq j$, we have that

$$0 = Cs = \sum_{i=1}^{n+k+1} C^i s_i = \sum_{i=i}^{m} C^{B(i)} s_{B(i)} + C^j = Bs_B + C^j.$$

This follows that $s_B = -B^{-1}C^j$.

Then, we could calculate the variation of the objective function by:

$$c'(x + \theta s) - c'x = \theta c' s = \theta(c_j + c'_B s_B) = \theta(c_j - c'_B B^{-1} C^j).$$

This value $r_j := c_j - c'_B B^{-1} C^j$ is called *reduced cost* of the variable x_j. The quantity is useful for finding columns in C that lead to an optimal solution. In case of nondegeneracy, if $r_j < 0$ for some j, we could minimize the goal increasing x_j.

Explain the derivation of reduced cost

From that, we define that the *reduced cost* r_y of a column y to be added to the solution is

$$r_y = c_y - w \cdot y.$$

Because y is well-formed, we have $c_y = 0$. Then, to reduce the cost of the objective function, we need $r_y \leq 0$ and, therefore, the new column must satisfy

$$w \cdot y \geq 0. \tag{3.2}$$

3.5.2 Intermediate problem

In this algorithm, only the n first elements, which correspond to a column of M, must be generated, since the last elements are composed of k 0-elements and one 1-element. This new column M^j is the corresponding vector in $\mathcal{P}$ of a $\mathcal{C}$-satisfiable interpretation. Then, to find such column, we can solve an intermediate problem.

Define $\mathcal{C} \cup \mathcal{P}$ as a potentially unsatisfiable CBox with all axioms of $\mathcal{C}$ and axioms from restrictions of $\mathcal{P}$. We want to find a new CBox $\mathcal{C}^{\bullet} \subseteq \mathcal{C} \cup \mathcal{P}$ such that $\mathcal{C}^{\bullet}$ is satisfiable and $\mathcal{C}^{\bullet} \models C \sqsubseteq D$ if $C \sqsubseteq D \in \mathcal{C}$. An interpretation model of $\mathcal{C}^{\bullet}$ will be $\mathcal{C}$-satisfiable and can generate a corresponding vector M^j in $\mathcal{P}$. Also, the new column y created from M^j must obey the restriction in Equation 3.2.

Instead of generating a column y that just satisfies the conditions above, we could generate one that maximizes the cost reduction in Equation 3.2. This can be done maximizing $\sum_{i=1}^{n} w_i \cdot y_i$ which is choosing axioms Ax_i for $\mathcal{C}^{\bullet}$ based on weights w_i, $1 \leq i \leq n$.

Then, this problem can be modeled as a $\mathcal{GEL}$-MaxSAT for $\mathcal{C} \cup \mathcal{P}$ where axioms of $\mathcal{C}$ have infinite weight and every axiom Ax_i from $\mathcal{P}$ has weight w_i.

3.5.3 Generating column

This leads to the solution in Algorithm 3. We generate a new column solving the intermediate problem with $\mathcal{GEL}$-Max-SAT-Solver and an auxiliary function ExtractColumn. If the $\mathcal{GEL}$-MaxSAT problem does not have a solution or Equation 3.2 cannot be satisfied, it returns False; otherwise, it returns True and the generated column y.

Algorithm 3 The algorithm of column generation

```
1:  function GenerateColumn($\langle \mathcal{C}, \mathcal{P} \rangle$, $lp$)
2:      $w \longleftarrow lp.dual$
3:      $result \longleftarrow \mathcal{GEL}$-Max-SAT-Solver($\mathcal{C} \cup \mathcal{P}, (w_1, \dots, w_n)$)
4:      if $result.isSuccess$ = False then
5:          return False
6:      end if

7:      $y \longleftarrow$ ExtractColumn($result$)
8:      if $w \cdot y < 0$ then
9:          return False
10:     end if

11:     return (True, $y$)
12: end function

13: function ExtractColumn($result$)
14:     $\mathcal{C}_{max} \longleftarrow result.solution$
15:     Define $n$-vector $m_n$ such that $m_i = 1$ iff we have $Ax_i$ in $\mathcal{C}_{max}$
16:     $y \longleftarrow (\, m_n' \,|\, 0_k' \,|\, 1\,)'$
17:     return $y$
18: end function
```

3.6 $\mathcal{GEL}$-MaxSAT solver

The function $\mathcal{GEL}$-Max-SAT-Solver is responsible for generating a solution to the $\mathcal{GEL}$-MaxSAT problem. To find such solution, we need to look at the graph modeling used for its SAT decision.

A weighted $\mathcal{GEL}$-CBox $\langle \mathcal{C} \cup \mathcal{P}, w \rangle$ can be represented as a weighted directed graph $G_w(\mathcal{C} \cup \mathcal{P}) = (N, E, \ell, w)$, following the representation in section 2.3, where (N, E, ℓ) are the same as before and $w : E \longrightarrow \mathbb{Q} \cup \{\infty\}$ is an edge weight function with $w((C \longrightarrow_0 D)) = \infty$ iff $C \sqsubseteq D \in \mathcal{C}$. We also write $w(u, v)$ to represent $w((u, v))$, with $u, v \in N$.

In this problem, we want to find a satisfiable subset of $\mathcal{C} \cup \mathcal{P}$. From Theorem 1, we know that a $\mathcal{GEL}$-CBox $\mathcal{C}$ is satisfiable iff there is a $Init$-$\bot$ path in $G(\mathcal{C})$. Thus, to find a satisfiable $\mathcal{C}^{\bullet} \subseteq \mathcal{C} \cup \mathcal{P}$, we need to remove every $Init$-$\bot$ path in $G_w(\mathcal{C} \cup \mathcal{P})$ which can be done by removing some edges in these paths. For maximality, we need to remove edges that minimize the sum of weights of the deleted set.

Also, given a weighted directed graph $G_w(\mathcal{C})$ and nodes $s, t \in N$, a *s-t cut* in $G_w(\mathcal{C})$ is a partition (S, T) such that $s \in S$ and $t \in T$. A *cut set* $Cut(S, T)$ is the set of edges of $G_w(\mathcal{C})$ with one end in S and other end in T; we use $Cut(s, t)$ to refer to a cut set from a s-t cut. To remove all paths from s to t, we need to remove all edges of a cut set $Cut(s, t)$. The weight of a cut set is given by the sum of weights of its edges:

$$w(Cut(s,t)) = \sum_{(u,v)\in Cut(s,t)} w(u,v).$$

A graph usually has many s-t cuts; a s-t cut is *minimal* (*min cut*) if it has minimal weight; it is finite if its cut set has only edges of finite weight.

Then, the $\mathcal{GEL}$-MaxSAT problem can be modeled as finding a min cut in a weighted directed graph, which leads to the solution in Algorithm 4. It receives a CBox $\mathcal{C} \cup \mathcal{P}$ and an n-vector w. First, we compute $G_w(\mathcal{C} \cup \mathcal{P})$ where the weights are defined as $w(C_i, D_i) = w_i$ if $C_i \sqsubseteq D_i$ appears in $\mathcal{P}$; otherwise, $w(C, D) = \infty$. After that, we use the program MinCut to calculate the cut set of a $Init$-$\bot$ min cut. If the weight of the minimal cut set is infinite, meaning that we need to remove a certain axiom to keep satisfiability, then there is no solution; otherwise, return E without the edges in the cut set.

Algorithm 4 The $\mathcal{GEL}$-MaxSAT solver algorithm

```
1: function $\mathcal{GEL}$-MAX-SAT-SOLVER($\mathcal{C} \cup \mathcal{P}$, $w$)
2:     Compute $G_w(\mathcal{C} \cup \mathcal{P}) \longleftarrow (N, E, \ell, w)$
3:     $Cut(Init, \bot) \longleftarrow$ MINCUT($G_w(\mathcal{C} \cup \mathcal{P})$, $Init$, $\bot$)
4:     if $w(Cut(Init, \bot)) = \infty$ then
5:         return False
6:     end if

7:     return (True, $E \setminus Cut(Init, \bot)$)
8: end function
```

The solution in Algorithm 5 calculates the $Init$-$\bot$ min cut in $G_w(\mathcal{C} \cup \mathcal{P})$, based on Edmonds-Karp algorithm for maximum-flow (Cormen *et al.*, 2009). We create a *residual graph* G_R where its weights w_R represent its *residual capacity*. In each iteration we search for the shortest $Init$-$\bot$ path P with positive weights (augment path), calculated with a *breadth first search*. We get a augment flow $\alpha := \{ w_R(u,v) \ : \ (u,v) \in P \}$ and propagate the flow through the path, which is decrementing by α the residual capacity forward, and incrementing backwards.

After executing the iterations in lines 5-10, we have a partition (S, T), where $Init \in S$, $\bot \in T$ and every edge (u, v) such that $u \in S$ and $v \in T$ have residual capacity 0. Also, the sum of residual capacities from edges from T to S is equal to the flow f passing from S to T. Then, because every edge from S to T have residual capacity 0, f is the sum of capacities (or weights) of these edges in $G_w(\mathcal{C} \cup \mathcal{P})$ which means that $Cut(S, T)$ is minimal (Ford and Fulkerson, 1962).

Also, note that edges with negative weight are removed beforehand from $G_w(\mathcal{C} \cup \mathcal{P})$ for maximality.

Algorithm 5 The minimal cut algorithm

```
1:  function MinCut($G_w(\mathcal{C} \cup \mathcal{P})$, $Init$, $\bot$)
2:      remove edges with negative weight in $G_w(\mathcal{C} \cup \mathcal{P})$
3:      define the residual graph $G_R := (N_R, E_R, \ell_R, w_R)$ as a copy of $G_w(\mathcal{C} \cup \mathcal{P})$
4:      while there is an augment shortest path $P$ in $G_R$ do
5:          $\alpha \longleftarrow \min \{ w_R(u, v) : (u, v) \in P \}$      ▷ $\alpha$ is the augment flow in $P$
6:          for each $(u, v) \in P$ do
7:              decrement $\alpha$ of $w_R(u, v)$
8:              increment $\alpha$ of $w_R(v, u)$
9:          end for
10:     end while
11:     get the set $S$ of nodes reachable from $Init$ by edges with positive weight
12:     return edges $(u, v) \in E$ such that $u \in S$ and $v \notin S$
13: end function
```

3.7 Time complexity analysis

In this section, we concentrate on presenting a polynomial time complexity bound of PGEL-SAT in Algorithm 1. Consider a probabilistic $\mathcal{GEL}$-knowledge base with n concepts, m certain axioms, p uncertain axioms and k probabilistic restrictions. Also, define $N = p + k + 1$.

Algorithm 1 starts with $O(N^2)$ arithmetic operations plus a call to LinProg. After that, it executes several iterations with calls to GenerateColumn and LinProg. From Carathéodory's Theorem (Eckhoff, 1993), the algorithm stops after at most N iterations.

The linear solver LinProg in this project implements an easy version of the primal-dual interior-point method based on Mehrotra's technique (Makhorin, 2001; Mehrotra, 1992). Although there are polynomial bounds for several variations of this method (Y. Zhang and D. Zhang, 1995; Salahi *et al.*, 2008; Teixeira and Almeida, 2012), we did not investigate the exact implementation to justify the use of a specific bound. Thus, for simplicity, we choose the known $O(N^3L)$ time complexity for interior-point methods (Goldfarb and Todd, 1989), where L is the bit-length of input data.

The program GenerateColumn in Algorithm 3 executes $O(N)$ operations, to handle the column, plus a call to $\mathcal{GEL}$-Max-SAT-Solver. This function, in Algorithm 4, builds the weighted graph, whose complexity is $O(n + (m + p))$, and calculates the minimal cut with Algorithm 5. The program MinCut is based on Edmonds-Karp algorithm (Cormen *et al.*, 2009), which is $O(n(m + p)^2)$, with an additional search in the graph for the cut, which is $O(n+m+p)$. Then, we have that Algorithm 3 has an upper bound of $O(N+n(m+p)^2)$.

Finally, Algorithm 1 has an upper bound of $O(N^4L + Nn(m + p)^2)$ operations. Therefore, the algorithm proposed to solve the PGEL-SAT problem is polynomial.

Chapter 4

Experiments and results

- tests to mesure time complexity
- experiments for different algorithms

Chapter 5

Related work

The problem of probabilistic reasoning and extensions in logics to deal with uncertainty have been studied for several decades. The first known proposal of PSAT, for propositional formulas, is attributed to Boole (1854) and it has already been shown to be NP-Complete (Georgakopoulos *et al.*, 1988).

In the relational domain, the literature contain several logics with probabilistic reasoning capabilities although they have led to intractable decision problems. Some of them extend the already intractable $\mathcal{ALC}$, with probabilistic constrains over concepts (Heinsohn, 1994; Lukasiewicz, 2008; Gutiérrez-Basulto *et al.*, 2011). For the expressive and lightweight $\mathcal{EL}$-family, some extensions such as Gutiérrez-Basulto *et al.* (2017) and Ceylan and Peñaloza (2017) have led to ExpTime-hard or PP-complete probabilistic reasoning; futhermore, NP-completeness can be achieved with probability capabilities over axioms (Finger, 2019).

On the other hand, many results implies that the research on Max-SAT has a impact on the solutions of PSAT problems (Andersen and Pretolani, 2001). Also, there was already proposed a MaxSAT-solver for a propositional fragment of horn logic by a max-flow/min-cut formulation (Jaumard and Simeone, 1987). Thus, it is expected to ask if one could also take such results to a relational domain.

Chapter 6

Conclusion and future work

References

[ANDERSEN and PRETOLANI 2001] Kim Allan ANDERSEN and Daniele PRETOLANI. “Easy cases of probabilistic satisfiability”. In: *Annals of Mathematics and Artificial Intelligence* 33.1 (2001), pp. 69–91 (cit. on pp. 1, 27).

[BAADER *et al.* 2005] Franz BAADER, Sebastian BRANDT, and Carsten LUTZ. “Pushing the EL Envelope”. In: *Proceedings of the 19th International Joint Conference on Artificial Intelligence.* IJCAI’05. 2005, pp. 364–369 (cit. on pp. 1, 5, 6).

[BERTSIMAS and TSITSIKLIS 1997] Dimitris BERTSIMAS and John N TSITSIKLIS. *Introduction to linear optimization.* Vol. 6. Athena Scientific Belmont, MA, 1997 (cit. on p. 19).

[BOOLE 1854] G. BOOLE. *An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities.* Collected logical works. Walton and Maberly, 1854. URL: https://books.google.com.br/books?id=DqwAAAAAcAAJ (cit. on pp. 1, 27).

[BORGWARDT 2012] Karl Heinz BORGWARDT. *The Simplex Method: a probabilistic analysis.* Vol. 1. Springer Science & Business Media, 2012 (cit. on p. 18).

[BOYD *et al.* 2004] Stephen BOYD, Stephen P BOYD, and Lieven VANDENBERGHE. *Convex optimization.* Cambridge university press, 2004 (cit. on p. 18).

[CEYLAN and PEÑALOZA 2017] Ismail Ilkan CEYLAN and Rafael PEÑALOZA. “The Bayesian Ontology Language BEL”. In: *Journal of Automated Reasoning* 58.1 (2017), pp. 67–95 (cit. on p. 27).

[CORMEN *et al.* 2009] Thomas H CORMEN, Charles E LEISERSON, Ronald L RIVEST, and Clifford STEIN. *Introduction to algorithms.* MIT press, 2009 (cit. on pp. 22, 23).

[ECKHOFF 1993] Jürgen ECKHOFF. “Helly, Radon, and Carathéodory type theorems”. In: *Handbook of convex geometry.* Elsevier, 1993, pp. 389–448 (cit. on pp. 17, 23).

[Finger 2019] Marcelo Finger. “Extending EL++ with Linear Constraints on the Probability of Axioms”. In: *Description Logic, Theory Combination, and All That. Essays Dedicated to Franz Baader on the Occasion of His 60th Birthday.* Ed. by Carsten Lutz, Uli Sattler, Cesare Tinelli, Anni-Yasmin Turhan, and Frank Wolter. Vol. LLNCS 11560. Theoretical Computer Science and General Issues. Springer International Publishing, 2019. isbn: 978-3-030-22101-0. doi: 10.1007/978-3-030-22102-7 (cit. on pp. 2, 7, 27).

[Finger and Lopes n.d.] Marcelo Finger and Andrew Ijano Lopes. “Tractable Probabalistic SAT in Graphic Description Logics”. In preparation (cit. on pp. 7–9, 12, 15, 17).

[Ford and Fulkerson 1962] LR Ford and DR Fulkerson. “Flows in networks”. In: (1962) (cit. on p. 22).

[Georgakopoulos *et al.* 1988] George Georgakopoulos, Dimitris Kavvadias, and Christos H Papadimitriou. “Probabilistic satisfiability”. In: *Journal of complexity* 4.1 (1988), pp. 1–11 (cit. on p. 27).

[Gilmore and Gomory 1961] Paul C Gilmore and Ralph E Gomory. “A linear programming approach to the cutting-stock problem”. In: *Operations research* 9.6 (1961), pp. 849–859 (cit. on p. 17).

[Gilmore and Gomory 1963] Paul C Gilmore and Ralph E Gomory. “A linear programming approach to the cutting stock problem—Part II”. In: *Operations research* 11.6 (1963), pp. 863–888 (cit. on p. 17).

[Goldfarb and Todd 1989] Donald Goldfarb and Michael J Todd. “Chapter ii linear programming”. In: *Handbooks in operations research and management science* 1 (1989), pp. 73–170 (cit. on p. 23).

[Gutiérrez-Basulto *et al.* 2011] Víctor Gutiérrez-Basulto, Jean Christoph Jung, Carsten Lutz, and Lutz Schröder. “A Closer Look at the Probabilistic Description Logic Prob-EL”. In: *Proc. 25th Conference on Artificial Intelligence (AAAI-11).* Ed. by Wolfram Burgard and Dan Roth. AAAI Press, 2011, pp. 197–202 (cit. on p. 27).

[Gutiérrez-Basulto *et al.* 2017] Víctor Gutiérrez-Basulto, Jean Christoph Jung, Carsten Lutz, and Lutz Schröder. “Probabilistic description logics for subjective uncertainty”. In: *Journal of Artificial Intelligence Research* 58 (2017), pp. 1–66 (cit. on p. 27).

[Heinsohn 1994] Jochen Heinsohn. “Probabilistic description logics”. In: *Uncertainty Proceedings 1994.* Elsevier, 1994, pp. 311–318 (cit. on p. 27).

[Hitzler *et al.* 2009] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F Patel-Schneider, Sebastian Rudolph, *et al.* “OWL 2 web ontology language primer”. In: *W3C recommendation* 27.1 (2009), p. 123 (cit. on p. 5).

[Jaumard and Simeone 1987] Brigitte Jaumard and Bruno Simeone. “On the complexity of the maximum satisfiability problem for Horn formulas”. In: *Information Processing Letters* 26.1 (1987), pp. 1–4 (cit. on p. 27).

[Klee and Minty 1972] Victor Klee and George J Minty. “How good is the simplex algorithm”. In: *Inequalities* 3.3 (1972), pp. 159–175 (cit. on p. 18).

[Krötzsch *et al.* 2012] Markus Krötzsch, Frantisek Simancik, and Ian Horrocks. “A description logic primer”. In: *arXiv preprint arXiv:1201.4089* (2012) (cit. on p. 3).

[Lamy 2017] Jean-Baptiste Lamy. “Owlready: Ontology-oriented programming in Python with automatic classification and high level constructs for biomedical ontologies”. In: *Artificial intelligence in medicine* 80 (2017), pp. 11–28 (cit. on p. 16).

[Lukasiewicz 2008] Thomas Lukasiewicz. “Expressive probabilistic description logics”. In: *Artificial Intelligence* 172.6-7 (2008), pp. 852–883 (cit. on p. 27).

[Makhorin 2001] Andrew Makhorin. “GNU linear programming kit”. In: *Moscow Aviation Institute, Moscow, Russia* 38 (2001) (cit. on pp. 18, 23).

[Mehrotra 1992] Sanjay Mehrotra. “On the implementation of a primal-dual interior point method”. In: *SIAM Journal on optimization* 2.4 (1992), pp. 575–601 (cit. on p. 23).

[Salahi *et al.* 2008] Maziar Salahi, Jiming Peng, and Tamás Terlaky. “On Mehrotra-type predictor-corrector algorithms”. In: *SIAM Journal on Optimization* 18.4 (2008), pp. 1377–1397 (cit. on p. 23).

[Sonnenschein n.d.] Nikolaus Sonnenschein. *swiglpk - Simple swig bindings for the GNU Linear Programming Kit.* url: https://pypi.org/project/swiglpk/ (cit. on p. 18).

[Teixeira and Almeida 2012] Ana Paula Teixeira and Regina Almeida. “On the complexity of a mehrotra-type predictor-corrector algorithm”. In: *International Conference on Computational Science and Its Applications.* Springer. 2012, pp. 17–29 (cit. on p. 23).

[Van Rossum and Drake 2009] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual.* Scotts Valley, CA: CreateSpace, 2009. isbn: 1441412697 (cit. on p. 15).

[Y. Zhang and D. Zhang 1995] Yin Zhang and Detong Zhang. “On polynomiality of the Mehrotra-type predictor—corrector interior-point algorithms”. In: *Mathematical Programming* 68.1-3 (1995), pp. 303–318 (cit. on p. 23).