An Exact Algorithm for First Order Probabilistic Logic

Razia Sultana

A Thesis

in

The Department

of

Computer Science and Software Engineering

Presented in Partial Fulfilment of the Requirements for
the Degree of Master of Computer Science at
Concordia University
Montréal, Québec, Canada

December 2007

Your file Votre référence
ISBN: 978-0-494-40954-1
Our file Notre référence
ISBN: 978-0-494-40954-1

ABSTRACT

An Exact Algorithm for First Order Probabilistic Logic

Razia Sultana

Human beings often have to reason and make decisions based on uncertain knowledge of real world problems. Therefore, many artificial intelligence (AI) applications, such as expert systems, must have the ability to understand the way human beings reason from uncertain data or knowledge in order to reach a conclusion. Several approaches have been proposed in this respect to deal with various kinds of uncertainty in AI. Among these approaches, probabilistic theory is used in many research areas such as knowledge-based systems, data mining, etc. Nilsson revisited in 1986 the early work of Boole (1854) and of Hailperin (1976) on logic and probability. He proposed a generalization of logic in which the truth values of sentences are probability values. The main problem addressed by Nilsson is the probabilistic satisfiability (PSAT) for both propositional and first-order logic: determine, given a set of sentences (i.e., clauses) and probabilities that these sentences are true, whether these probabilities are consistent. Since first-order logic is used in many AI applications due to its expressiveness to represent knowledge over propositional logic, our thesis proposes an extension of the mathematical modeling of PSAT to first-order logic, FOPSAT for short. We next propose an exact algorithm based on delayed column generation technique, to check consistency and, if consistency holds, to entail new probability values for an additional logical sentence to be true and such that the augmented set of sentences remains consistent. We illustrate the proposed algorithm on an example, and discuss its potential to solve medium size FOPSAT instances.

Acknowledgement

This research work could not be accomplished without the guidance and sincere support from a lot of individuals. I would like to express my heart felt gratitude to all of them and mention the following people especially for their contributions.

First of all, I would like to thank my supervisor Dr. Brigitte Jaumard for her invaluable guidance, advice, support and criticism since from the beginning of my work. Undoubtedly she is one of the best supervisors as well as individuals that I have ever worked with. She was always encouraging, patient and gave me enough time for the better understanding of the problems and guided me through the way of research. It is because of her, my graduate studies became meaningful and I ended up with a successful completion of my thesis in time.

I would also like to express the deepest appreciation to my committee chair Dr. Clement Lam, the examiners Dr. Nematollaah Shiri and Dr. Peter Grogono for their valuable suggestions and corrections for maintaining the quality of the thesis.

My sincere gratitude goes to Professor Monty Newborn, School of Computer Science, and McGill University for helping me with 'Theo 2006' the automated theorem-proving program and his kind response to the e-mails. I also would like to thank Dr. Geoff Sutcliffe, department of Computer Science, University of Miami for providing help and information regarding Thousands of Problems for Theorem Provers (TPTP) and a few tips for formulation of our instances.

Sincere appreciation goes to all of my lab members for their effective help and suggestions. Special thanks go to my friends who have helped me with their expert advice when I was stuck. Last but not the least, I do convey my heart felt thanks to my parents and siblings for their love, affection and support throughout the journey.

Dedicated

To My Friends and Family

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Human being has an excellent capability of reasoning and drawing conclusions from uncertain information and environment. Therefore, any method which is designed to mimic human reasoning must possess such capability to reason with uncertain data. Due to incompleteness of our real world knowledge, models may not capture the uncertainty whereas collected data may be partial, ambiguous, erroneous, or imprecise. Reasoning under uncertainty plays an essential role in many artificial intelligence applications, e.g., in expert systems, where many of the rules and data are obtained from experts or users with some degree of uncertainty. It is also applicable in automated theorem proving, testing of combinational circuits and reliability. Therefore, various generalizations of logic dealing with uncertainties were studied by many AI researchers. Pearl (see [61], p.2-4) classified them into three categories *logicist, neo-calculist* and *neo-probabilist.* Researchers dealing with uncertainty using non-numerical techniques are categorized as *logicist* by Pearl [61]. *Neo-calculist*

uses numerical representations with an entirely new calculi, such as Dempster-Shafer calculus, fuzzy logic or certainty factors, whereas, *neo-probabilist* handles uncertainty with traditional probability theory associated with computational facilities needed to perform AI tasks. Pearl [61] also mentioned another category of researchers who handle uncertainty in an ad-hoc way. For instance, an uncertainty factor has been incorporated in a heuristic fashion in most early systems such as MYCIN [6] or PROSPECTOR [44]. According to semantical point of view in various approaches, Pearl [61] also compares *extensional* vs. *intensional* approaches. In an *extensional* approach, used, e.g., in *production* systems or *rule-based* systems or *procedure based* systems, uncertainty is handled by assigning truth values to formulas. Moreover, the uncertainty of a new formula is deduced as a function of the uncertainties of its sub-formulas which is "computationally convenient but semantically sloppy" [61]. Whereas, uncertainty is assigned to "state of affair" or subset of *possible worlds* in the *intensional* approach which is "semantically clear but computationally clumsy" [61]. This *extensional* and *intensional* approaches are related to the *proof theory* vs. *model theory.*

In this thesis, our focus is on reasoning under uncertainty models where uncertainty is captured using probability values. The earliest works are due to Boole (1854) [5] and later on, to Hailperin (1976). Hailperin [26] revisited the works of Boole with a first attempt to use the linear programming tools in order to solve the mathematical models. In 1986, Nilsson [56] published a seminal article on the so-called probabilistic logic where he discussed probabilistic logic not only for propositional formulas but also for first-order formulas.

Nilsson addressed two forms of the probabilistic logic problem [56] namely, the *decision* form and the *optimization* form. The *decision* form corresponds to the so called *probabilistic satisfiability* or P^{PSAT} problem, i.e., it deals with the consistency of logical sentences together

with their probabilities of being true. For instance, assume the sentences, *Bangladesh wins today's match, Bangladesh wins today's match* $\Rightarrow$ *Bangladesh is champion*, and *Bangladesh is champion* have probability 0.8, 0.6 and 0.3 respectively. Are these probabilities consistent? Now let us assume that the above set of sentences together with the set of probabilities is consistent; let *Bangladesh wins today's match* $\Rightarrow$ *Bangladesh celebrates* be a new sentence with an unknown probability. The *optimization* form leads to the so called *probabilistic entailment* or P^{PENTAIL} problem that deals with what is the tightest range of probability values for this additional sentence such that the overall augmented set of sentences remains consistent.

The P^{PSAT} problem can be formally expressed as follows. Let $S_1, S_2, \ldots, S_m$ be propositional sentences obtained from the logical variables $x_1, x_2, \ldots, x_n$ with the usual boolean operators $\vee$, $\wedge$, $\neg$. Let π_1, π_2, ..., π_m be the probabilities that these sentences are true. Are these probabilities consistent? Assume that the system with $S_1, S_2, \ldots, S_m$ of m sentences to be consistent; let S_{m+1} be an additional propositional sentence with an unknown probability π_{m+1}. The P^{PENTAIL} problem considers finding the best possible lower and upper bounds on the probability π_{m+1} such that the overall augmented system remains consistent.

Before answering how to solve Nilsson's probabilistic logic problem, let us recall two well known problems in AI inference, namely *satisfiability* and *logical consequence* problems. Recall first that, an *interpretation* I of a set of propositions is the assignment of a truth value, either T (true) or F (false), to each propositional symbol ([49], p. 50). If a given set of propositional sentences $S_1, S_2, \ldots, S_m$ is T under an interpretation I, then we say that I satisfies the set, or the set is satisfied by I. If a sentence S_i is T in an interpretation I, we say that I is a *model* of S_i or I satisfies S_i ([7], p. 11). A set of propositional

sentences $S_1, S_2, \ldots, S_m$ is said to be *satisfiable* (or *consistent*) if and only if there exists an *interpretation* I, such that the set of sentences is evaluated to T in I. Now let us consider a set of sentences $S_1, S_2, \ldots, S_m$ and an additional sentence S_{m+1}. S_{m+1} is said to be a *logical consequence* of $S_1, S_2, \ldots, S_m$ (or S_{m+1} *logically follows* from $S_1, S_2, \ldots, S_m$) if and only if for every interpretation I in which $S_1 \wedge S_2 \wedge \cdots \wedge S_m$ is T, S_{m+1} is also T [7]. For example, if $(P \Rightarrow Q)$ and $(Q \Rightarrow R)$ are given, then $(P \Rightarrow R)$ is a logical consequence of $(P \Rightarrow Q) \wedge (Q \Rightarrow R)$.

In order to solve the probabilistic logic problem, one needs to develop a model for both the decision and optimization forms. If we exclude the probability issues, Nilsson's decision or P^{PSAT} problems can be reduced to the *satisfiability* problem whereas, the optimization or P^{PENTAIL} can be reduced to the *logical consequence* problem. However, it is to be noted that, not all the models which have been proposed in the literature for reasoning under uncertainty deal with both problems. For instance, in *fuzzy logic*, only logical consequence is addressed but not the satisfiability.

Nilsson's two forms of the probabilistic logic problem can be formulated as a mathematical model (see section 2.1.1). A straightforward way to solve this model is by linear program tools using the simplex or the revised simplex algorithm (see, e.g., Chvatal [10], p.97). Unfortunately, such a solution becomes intractable even for a moderate size problem. However, there are some tools in operation research which can efficiently deal with large scale problems, namely the column generation techniques (see [10], p.198).

First-order logic has more expressive power than other representations, like propositional calculus. Moreover, its primitive semantic features are very closely related to the manner in which we perceive our environment [3]. Human beings identify individual objects and

classify them according to their properties. First-order logic is built up similarly from a collection of individual objects, and from grouping together individuals into sets who share some common properties. The primitive semantic functions which serve to build up complex assertions are equally transparent [3]. First-order logic is used in many AI applications due to its expressive power and transparent semantics to represent knowledge. Therefore, we are interested in generalization to first-order from propositional logic. The goal of this thesis is to develop and implement a well defined scalable exact algorithm to solve medium size *first-order probabilistic logic* or FOPL instances. The *column generation methods* proved to be efficient for *propositional probabilistic logic* or PPL sentences. Therefore, we are interested in investigating column generation methods for FOPL. The models for reasoning under uncertainty as well as algorithms for entailing new information in first-order logic have been discussed in several papers (see, e.g., Lukasiewicz [50, 51], Page and Srinivasan [59], Halpern [28], Ng and Subrahmanian [55]) but the proposed algorithms are heuristic and not always scalable when there is a large number of predicates and formulas.

1.2 Thesis Contribution

In this thesis, the first contribution is a mathematical modeling for the first-order probabilistic logic or FOPL. We started from the mathematical model for propositional probabilistic logic [39] or PPL, based on a column generation formulation. While in propositional calculus, the underlying considerations are only with propositional variables and symbols, in first-order logic, it is mandatory to handle quantifiers, predicates and functions. Therefore, it is more difficult to provide a mathematical model for first-order probabilistic logic than for propositional probabilistic logic. Still, we managed to generalize the PPL mathematical

programming formulation for the FOPL problem. It is also based on a column generation formulation. This has already been published in [38].

As the second contribution, we have developed a solution scheme for the FOPL mathematical model. While it can be solved by linear programming using the simplex or revised simplex algorithms [10], the approach is limited to small instances as the number of variables grows exponentially in the number of predicates. However, as for PPL, by using column generation techniques (see Chvatal [10], p. 195), these limitations can be overcome. Moreover, it is much faster than the classical simplex algorithm for solving linear programs since it is not required to consider explicitly all the possible worlds to guarantee an optimal solution. A *possible world* is a truth assignment on the set of sentences such that the set of sentences is satisfiable for that truth assignment. A possible world corresponds to a column in the column generation model. Column generation begins with taking a small, manageable part of a problem (few of the possible worlds), solves that part, then analyses its partial solution to determine an additional small subset of worlds (one or more possible worlds) to be added to the model, and then solves the enlarged model. This column-wise modeling repeats this process until it satisfies an optimality condition for the problem.

More formally, the column generation method relies on a decomposition of the initial linear program into a *master* and a *pricing* problem. At each iteration, we consider only a *restricted master problem*, i.e., the master problem with only a small number of variables, i.e., a small subset of possible worlds being considered. The *pricing problem* corresponds to the mechanism of generating possible worlds which improve the value of the current solution and which are added in the restricted master problem. Therefore, in order to solve the FOPL, we need an algorithm for both the restricted master problem as well as for the

pricing problem.

Usually solving the *restricted master problem* is easy but solving the *pricing problem* is more difficult. The pricing problem is an optimization problem. Therefore, it is defined by an *objective* and a *set of constraints.* The *objective* of the pricing problem is to find a world with an imposed (negative or positive) sign for the reduced cost such that it will improve the current solution. The *reduced cost* is a metric that is used to check the optimality of a solution of an LP [10]. The *set of constraints* corresponds to the rules associated with the definition of a possible world. In PPL, a world is a possible or valid truth assignment over the sentences, i.e., it is possible if there exists a truth assignment on the variables that leads to that truth assignment on the sentences. Finding a possible world is easy in PPL, indeed, the truth assignment or number of interpretations of a propositional sentence is finite. Whereas, in FOPL, it is more difficult to obtain a possible world. Let us recall from Chang and Lee ([7], p. 31), an *interpretation* for a first-order formula F consists of two things, namely, a nonempty domain D and an assignment of *values* to each constant, function symbol and predicate symbol occurring in F. In order to emphasize the domain D, it is often considered as an interpretation of the first-order formula over D. For instance, evaluating the truth value of a first-order formula in an interpretation over domain D, $(\forall x)$ is interpreted as *"for all elements x in D"* and $(\exists x)$ is interpreted as *"there is an element in D"* ([7], p. 31). Therefore, the number of interpretations of a first-order formula is infinite ([7], p. 35). The definition of *satisfiability* problem for first-order logic and propositional calculus is same. However, the definition of *interpretation* is different as we just discussed. Finding a feasible solution of the pricing problem can be reduced to the well known *satisfiability* problem of first-order logic which has been studied for a long time. In order to solve this satisfiability

problem we have used a package named *Theo-2006*. This package [54] reaches a decision by searching for contradiction or inconsistency rather than satisfiability using the *resolution refutation* principle (see Section 2.2.1.3). However, solving the satisfiability problem is only one piece of our pricing problem, corresponding to the search of a feasible solution. We still have to meet the objective of finding a negative reduced cost in order to reach the optimal solution. This problem can be solved by using a *branch and bound* method.

As a third contribution of this thesis, we have devised an algorithm for solving the *pricing problem*. Our proposed algorithm is an *exact* algorithm. An exact algorithm guarantees to find an optimal solution if there exists one. Our algorithm is a scalable one as it can solve medium size first-order instances. It is to be noted that, to the best of our knowledge, so far, it leads to the first exact algorithm for solving the first-order probabilistic logic problem.

The final contribution is, we have implemented this algorithm successfully. The experiments and the analysis of the results are discussed in Chapter 5.

1.3 Thesis Organization

An introduction to probabilistic logic problem and its mathematical programming formulations are given in Chapter 2. In Chapter 3, review a variety of the models for reasoning under uncertainty in the propositional case and first-order models with probabilities. In addition, a review on analytical and numerical solutions of probabilistic logic problems and some practical and scalable first-order theorem provers are also discussed in Chapter 3. The mathematical model for FOPL and our proposed algorithms for solving the model are explained in Chapter 4. An example is also provided in the same chapter to illustrate

the SOLFOPL algorithm. Moreover, we discuss its efficiency for solving medium size FOPL instances in this chapter. The numerical results are presented in Chapter 6 on different medium size FOPL instances with up to 26 formulas, 11 predicates and 3 variables. We conclude the thesis in Chapter 6, outlining suggestions and scopes for future work.

Chapter 2

Introduction to Probabilistic Logic

2.1 Propositional Probabilistic Logic

Uncertainty is a common factor that is dealt in many artificial intelligence applications. Probabilistic inference is one of the perspectives that has been studied in order to reason under uncertainty for, e.g., in expert systems. To deal with probability of logical sentences in either propositional or first-order logic, Nilsson [56] proposed a stringent framework in 1986. Moreover, he considered two related forms of probabilistic logic problem, namely the *decision* and the *optimization* forms.

Recall from Chapter 1, given a set of sentences with their associated probabilities, the decision form of the probabilistic logic problem which corresponds to the *probabilistic satisfiability* or P^{PSAT} problem answers the question: Are these probabilities consistent? Now, assume that, the given set of sentences with the associated probabilities is consistent. The optimization form of probabilistic logic problem which corresponds to the *probabilistic entailment* or P^{PENTAIL} problem finds the best possible probability value or interval for an

additional logical sentence to be true such that the augmented set of sentences remains consistent together with its probabilities. In other words, the probabilistic entailment problem determines the range of values of the probability associated with a new logical sentence such that the overall augmented set of sentences remain consistent.

Jaumard *et al.* [39] extended Nilsson's model [56] and the approach of Georgakopoulos *et al.* [20] by considering intervals of probability values and conditional probabilities for the sentences to be true. Moreover, their extensions can also determine minimum changes in the probability values or intervals in order to restore consistency in the case where the initial probabilities are inconsistent.

2.1.1 Decision form

The probabilistic logic problem in *decision* form P^{PSAT} is defined in [39] as follows: consider a set of m logical sentences $\mathcal{S} = \{S_1, S_2, \ldots, S_m\}$ defined on n propositional (boolean) variables $X = \{x_1, x_2, \ldots, x_n\}$ with the usual Boolean operators. These variables correspond to elementary propositions which are either *true* or *false*. Assume probabilities $\pi_1, \pi_2, \ldots, \pi_m$ for these sentences to be true are given. Are these probabilities consistent?

In propositional calculus, a *literal* is either a propositional variable or its negation, e.g., y_j and $\overline{y}_j$ are positive and negative literal respectively. Let us consider sentences from the propositional calculus and let us also assume, as in most expert systems, that sentences S_i's are logical implications of the form:

$$S_i : \varphi_i \Rightarrow \psi_i \tag{1}$$

$$\text{where} \quad \begin{cases} \varphi_i & \text{the antecedent or premise of the implication} \\ \psi_i & \text{the consequent} \end{cases}$$

are boolean functions. In propositional calculus, a *clause* is a disjunction of literals, e.g., $y_j \vee \overline{y}_{j'}$. An example of conjunction of literals is $y_j \wedge \overline{y}_{j'}$. Let us introduce two normal forms of clause representation, the *Disjunctive Normal Form* or DNF and the *Conjunctive Normal Form* or CNF. A sentence is in DNF if it is a set of disjunctions of conjunctions of literals, whereas, a sentence is in CNF if it is expressed using a set of conjunctions of disjunctions of literals. For instance, $((y_j \wedge \overline{y}_{j'}) \vee (\overline{y}_j \wedge y_{j''}))$ and $((y_j \vee \overline{y}_{j'}) \wedge (\overline{y}_{j''} \vee y_{j'}))$ are examples of DNF and CNF respectively. If the φ_i are written in CNF and the ψ_i are written in DNF, then S_i in (1), is a DNF expression:

$$\bigwedge_{j \in A} y_j \Rightarrow \bigvee_{j \in C} y_j \tag{2}$$

$$\equiv (\bigvee_{j \in A} \overline{y}_j) \vee (\bigvee_{j \in C} y_j). \tag{3}$$

Usually, sentences are represented in clausal form as shown above in (3).

Nilsson [56] defines a *world* as any truth assignment w over $\mathcal{S}$. A world w is *possible* if there exists a truth assignment over the set X of variables which leads to w over $\mathcal{S}$, and the world is *impossible* otherwise. From now on, we will only deal with possible worlds.

Let $p = (p_1, p_2, \dots)$ be a *probability distribution* on W. Assume we are also given, probabilities $\pi_1, \pi_2, \dots, \pi_m$, one for each logical sentence. The probability distribution *satisfies* the set of logical sentences together with the probabilities if: *for each sentence* S_i

($i = 1, 2, \ldots, m$), the sum of all p_j's over all truth assignments w_j which satisfy S_i equals π_i.

Let A be an $m \times \mid W \mid$ matrix such that:

$$a_{ij} = \begin{cases} 1 & \text{if } w_j \text{ satisfies } S_i \\ 0 & \text{otherwise.} \end{cases} \tag{4}$$

The *Probabilistic satisfiability* P^{PSAT} can be mathematically defined as follows: Is there a probability distribution p such that the following system admits at least one solution?

$$(P^{\text{PSAT}}) \quad \begin{cases} \mathbb{1} \cdot p = 1 \\ Ap = \pi \\ p \geq 0, \end{cases} \tag{5}$$

where $\mathbb{1}$ is a k unit row vector and $k \leq 2^m$. The value of k is 2^m in the worst case, but in practice it is smaller due to the fact that (i) not all worlds are feasible and (ii) two different value assignments on the variables may lead to the same world. Let p and π denote the column vectors $(p_1, p_2, \ldots, p_n)^T$ and $(\pi_1, \pi_2, \ldots, \pi_m)^T$ respectively.

Example 1: Consider the set of sentences from the introductory example, with $x_1 \equiv$ *Bangladesh wins today's match*, and $x_2 \equiv$ *Bangladesh is champion.*

$$S_1 \equiv x_1 \qquad \pi_1 = 0.8$$

$$S_2 \equiv \overline{x}_1 \vee x_2 \qquad \pi_2 = 0.6$$

$$S_3 \equiv x_2 \qquad \pi_3 = 0.3.$$

Denote by $\mathbb{1}$ the 2^2 unit row vector [1 1 1 1], and let

$$p = \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix}, \quad \pi = \begin{bmatrix} \pi_1 \\ \pi_2 \\ \pi_3 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.6 \\ 0.3 \end{bmatrix}.$$

Now let us show how to find the set of possible worlds W from the possible assignments on variables x_1 and x_2. For instance, let us assign the values (1, 1) to the variables (x_1, x_2). Compute the values of sentences (S_1, S_2, S_3), we find a first possible world, w_1= (1, 1, 1). Similarly, by considering the assignments (1, 0), (0, 1) and (0, 0) for the variables and then computing the resulting values for the sentences, we find three more possible worlds (1, 0, 0), (0, 1, 1), (0, 1, 0) respectively. The matrix A is then

$$\begin{array}{c} \\ S_1 \\ S_2 \\ S_3 \end{array} \begin{array}{c} \begin{array}{cccc} w_1 & w_2 & w_3 & w_4 \end{array} \\ \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \end{array}. \tag{6}$$

The P^{PSAT} mathematical program associated with this example can be written as follows:

$$\mathbb{1} \cdot p \equiv [1111] \times \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = p_1 + p_2 + p_3 + p_4,$$

$$Ap = \pi \equiv \begin{bmatrix} 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 \end{bmatrix} \times \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} 0.8 \\ 0.6 \\ 0.3 \end{bmatrix},$$

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} \geq 0;$$

i.e., it reduces to find p such that:

$$\begin{aligned} p_1 + p_2 \quad + p_3 + p_4 &= 1 \\ p_1 + p_2 &= 0.8 \\ p_1 \quad + p_3 + p_4 &= 0.6 \\ p_2 + p_3 &= 0.3 \\ p_j \geq 0, \qquad & j = 1, 2, 3, 4. \end{aligned}$$

or show that there is no such p.

Building a possible world is easy as it amounts to computing the values of a set of sentences, for given values assigned to the variables. Checking whether a world is possible

is however NP-complete as it reduces to the SAT problem.

2.1.2 Optimization form

Consider an instance of the P^{PSAT} problem $(\mathcal{S},\pi)$ that is consistent. Let S_{m+1} denote an additional logical sentence which is deduced possibly from $\mathcal{S}$ with an unknown probability π_{m+1}. The *probabilistic entailment* or P^{PENTAIL} problem of Nilsson [56] determines the lower and upper bound $[\ \underline{\pi}_{m+1},\ \overline{\pi}_{m+1}\ ]$ of the probability π_{m+1} which is associated with S_{m+1}, such that $(\ \mathcal{S} \bigcup S_{m+1},\ (\pi,\ \pi_{m+1}))$ remains consistent. In order to solve the P^{PENTAIL} problem, let us consider the objective function $A_{m+1}P$, where $A_{m+1} = (a_{m+1,j}), j = i, 2, \ldots$ and

$$a_{m+1,j} = \begin{cases} 1 & \text{if } S_{m+1} \text{ is true for the possible world } w_j \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

We next determine $\underline{\pi}_{m+1} = \min A_{m+1}P$ and $\overline{\pi}_{m+1} = \max A_{m+1}P$ subject to constraints (5). Note that it is possible that S_{m+1} may contain some variables which do not appear in the logical sentences of $\mathcal{S}$. No matter what, due to the addition of a new sentence, the set of possible worlds might remain the same or double in the worst case. The mathematical formulation of the *probabilistic entailment* problem can be written as follows:

$$(P^{\text{PENTAIL}}) \qquad \begin{array}{l} \min\ (\max)\quad A_{m+1}p \\ \text{subject to:} \end{array} \begin{cases} \mathbb{1} \cdot p = 1 \\ Ap = \pi \\ p \geq 0. \end{cases} \tag{8}$$

Example 2: Let us again consider Example 1, setting π_3 to 0.5 to obtain consistency. Add a new logical sentence $S_4 \equiv$ *Bangladesh wins today's match* $\Rightarrow$ *Bangladesh celebrates*, i.e., $S_4 \equiv x_1 \rightarrow x_3$ with $x_3 \equiv$ *Bangladesh celebrates*. As there is a new logical variable x_3 which was not present before in the sentences of $\mathcal{S}$, there are now eight truth assignments on X, and six possible worlds, i.e., (1, 1, 1, 1), (1, 0, 0, 1), (0, 1, 1, 1), (0, 1, 0, 1), (1, 1, 1, 0) and (1, 0, 0, 0). The third one corresponds to the truth assignments (0, 1, 1) and (0, 1, 0) over X and the fourth one to the truth assignments (0, 0, 1) and (0, 0, 0). The matrix A is then

$$\begin{array}{c} \\ S_1 \\ S_2 \\ S_3 \\ S_4 \end{array} \begin{array}{c} \begin{array}{cccccc} w_1 & w_2 & w_3 & w_4 & w_5 & w_6 \end{array} \\ \left[\begin{array}{cccccc} 1 & 1 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 \end{array}\right] \end{array} . \tag{9}$$

We want to find the probability range for π_4 associated with S_4 so that the system $\mathcal{S} \bigcup \{S_4\}$ remains consistent. The mathematical formulation of the corresponding probabilistic entailment or P^{PENTAIL} problem is:

$$\min \text{ (max) } \pi_4 = \quad p_1 + p_2 + p_3 + p_4$$

subject to:

$$
\begin{aligned}
& p_1 + p_2 + p_3 + p_4 + p_5 + p_6 = 1 \\
& p_1 + p_2 \qquad\qquad + p_5 + p_6 = 0.8 \\
& p_1 \qquad + p_3 + p_4 + p_5 \qquad = 0.6 \\
& p_1 \qquad + p_3 \qquad + p_5 \qquad = 0.5 \\
& p_j \geq 0, \qquad\qquad\qquad\qquad j = 1, 2, 3, 4, 5, 6.
\end{aligned}
$$

Its optimal solution yields $[\underline{\pi}_4, \overline{\pi}_4] = [0.2,\ 1]$.

2.1.3 Extensions

It is often more realistic to use probability intervals rather than single point probability values, as already observed by Hailperin [25], in order to take into account uncertainty on the sentences. The P^{PSAT} problem, with given probability intervals for the truth of sentences, is then defined as *Is there a probability distribution p such that the system $(\mathcal{S}, \pi)$* admits at least one solution? In terms of a mathematical program, it leads to:

$$
(P^{\text{PSAT}}) \qquad \begin{cases} \mathbb{1} \cdot p = 1 \\ \underline{\pi} \leq Ap \leq \overline{\pi} \\ p \geq 0. \end{cases} \tag{10}
$$

The optimization form, P^{PENTAIL} problem can be written as follows:

$$(P^{\text{PENTAIL}}) \qquad \text{subject to:} \quad \begin{cases} \min(\max) \quad A_{m+1}p \\ \mathbb{1} \cdot p = 1 \\ \underline{\pi} \leq Ap \leq \overline{\pi} \\ p \geq 0. \end{cases} \tag{11}$$

2.2 First-Order Probabilistic Logic

So far we described the probabilistic logic model in the context of propositional logic. In this section, we will introduce the probabilistic logic model in the context of first order logic which is the main focus of this thesis work.

2.2.1 Decision form

Consider a set of q logical predicates $\mathcal{P} = \{P_1, P_2, \ldots, P_q\}$ defined on a set of n variables $X = \{x_1, x_2, \ldots, x_n\}$. Let $\mathcal{F}$ be a set of m formulas in prenex normal form defined as follows:

$$\begin{aligned} \mathcal{F} = \{F_i = (Q_1^i x_1)(Q_2^i x_2) \ldots (Q_n^i x_n)(M_i) : \\ M_i = S_i(P_1, P_2, \ldots, P_q),\ i = 1, 2, \ldots, m\} \end{aligned} \tag{12}$$

where $(Q_j^i x_j), j = 1, 2, \ldots, n$ is either $(\exists x_j)$ or $(\forall x_j)$, and S_i is a logical sentence that contains no quantifier and is built on the set of predicates using the logical connectives $\neg, \wedge$ and $\vee$. $(Q_1^i x_1)(Q_2^i x_2) \ldots (Q_n^i x_n)$ is called the prefix and M_i is called the matrix of the

formula F_i, $i = 1, 2, \ldots, m$. For instance, $\forall x \exists y [P(x, y) \vee Q(y)]$ is a first-order formula, where $P(x, y)$ and $Q(y)$ are predicates, x is universally quantified and y is existentially quantified.

The *decision* form of first-order probabilistic logic problem, or P^{FOSAT} for short, is defined as follows: given a set $\mathcal{F}$ of m formulas in prenex normal form, and a probability vector $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ associated with these formulas, is the set $(\mathcal{F}, \pi)$ consistent?

We define a *world* as any truth assignment w over $\mathcal{F}$, as in Nilsson [56]. A *world* is *possible* if there exists a truth assignment over the set of predicates and a value assignments over X which leads to w over $\mathcal{F}$, and the world is impossible otherwise.

Building a possible world amounts to considering possibly several truth assignments on the set of predicates, i.e., one value assignment for an existentially quantified variable, but two value assignments for a universally quantified variable. If we assume the number of variables and predicates occurring in a formula to be bounded, it is polynomial. However, in the general case, it is NP-complete. On the other hand, checking whether a world is possible amounts to checking whether a set of first-order formulas is satisfiable: it is therefore NP-complete.

2.2.1.1 Decidability

Before we discuss the solution for the P^{FOSAT} problem, let us have a closer look at the specific difficulties of first-order logic. Unlike the propositional calculus, first-order logic is undecidable, provided that the language has at least one predicate with at least 2 variables. Let us recall the following two fundamental results.

Theorem 1. *(Church [9], Turing [70]). The satisfiability problem for first-order logic is undecidable.*

Theorem 2. *(Trakhtenbrot [69], Craig [11]). The satisfiability problem for first-order logic on finite structures is undecidable.*

In spite of these negative results, i.e., no sound and complete proof system for validity even on finite structures, several studies were conducted in order to explore decidable first-order fragments. A survey by Hustadt *et al.* [37] provides references of the known decidable first-order fragments. Indeed, there are several well identified decidable first-order fragments, e.g., the AEA class consists of all the relational (i.e., without function symbols) first-order sentences with quantifier prefix of the form $\forall\exists\forall$ [23], (see, e.g., Dreben and Goldfarb [15] and Aspvall *et al.* [2]), or the two-variable fragment (see, e.g., Grädel *et al.* [23]). From now on, we will only consider set of formulas belonging to some decidable first-order fragments.

2.2.1.2 First-order concepts

In this section, we describe some first-order logic concepts for better understanding.

- **Atomic formula ([49], p. 56):** An *atomic formula* or *atom* is a predicate constant of arity n, followed by n terms, $t_1, t_2, \ldots t_n$, enclosed in parenthesis and separated by commas. For example, friends(nazma, istiaque) is a *atomic formula.* Truth values are also atoms.

- **Literal ([49], p. 517):** A *literal* is an atomic expression or the negation of an atomic expression.

- **Clause ([7], p. 48):** A *Clause* is a finite disjunctions (or $\vee$) of zero or more literals. For example, $(\neg lily(X) \vee flower(X))$ is a clause with two literals $\neg lily(X)$ and

$flower(X)$.

- **Empty clause ([7], p. 48):** An *empty clause* ($\square$) can be attained by creating contradiction between two sets of literals. For instance, $flower(X)$ and $\neg flower(X)$ results a *null* or *empty clause* ($\square$).

- **Function ([49], p. 53):** A *function* denote a mapping of one or more elements in a set (called the *domain* of the function) into a unique element of another set (the range of the function) where elements of the domain and range are objects in the world of discourse. Every function has an associated *arity*, indicating the number of elements in the domain mapped onto each element of the range. For instance, *father(rusmi)* denotes a function of arity 1 that maps people onto their (unique) male parent and *plus(2, 3)* is a function of arity 2 that maps two numbers onto their arithmetic sum 5.

- **Term ([49], p. 54):** A *term* is either a constant, variable or function expression which is used to denote objects and properties in a problem domain. For example, *cat, mother(sadaf)*, etc.

- **Predicate ([49], p. 55):** A *predicate* names a relationship between zero or more objects in the world where the number of objects so related is the arguments of the predicate. An example of predicate with 2 arguments is *likes(X, jane)* where, X is a variable and jane is a constant. The arguments of a predicate may include terms which are constants, variables or function expressions. For example, *friends(fatherof(rusmi), fatherof(raika))* is a predicate describing a relationship between two objects in a domain of discourse. If the function expressions (*fatherof(rusmi)* is *istiaque* and *fatherof(raika)*

is *rejaul*) are evaluated, the expressions become *friends(istiaque, rejaul)*.

- **Interpretation ([7], p. 31):** An *interpretation* for a first-order formula F consists of a nonempty domain D, and an assignment of *values* to each constant, function symbol and predicate symbol occurring in F.

- **Satisfiable ([7], p. 34):** A formula F is *satisfiable* (*consistent*) if and only if there exists an interpretation I such that F is evaluated to T in I. If a formula F is T in an interpretation I, we say that I is a model of F and I satisfies F.

- **Logical consequence ([7], p. 34):** A first-order formula F_{m+1} is a *logical consequence* (*logically follows*) of (from) set of formulas $F_1, F_2, \ldots, F_n$ if and only if for every interpretation I, if $F_1 \wedge F_2 \wedge \cdots \wedge F_n$ is true, then F_{m+1} is also true in I.

- **Inference rules ([49], p. 65):** An *inference rule* for both the propositional and first-order logic can be defined as a formal way of generating a new formula which is a logical consequence of a given set of existing formulas.

- **Sound ([49], p. 66):** An inference rule is *sound* if every formula produced by the inference rule from a set $\mathcal{F}$ of formulas also logically follows from $\mathcal{F}$.

- **Complete ([49], p. 66):** An inference rule is *complete* if, given a set $\mathcal{F}$ of formulas, the inference rule can infer every formulas that logically follows from $\mathcal{F}$.

- **Refutation complete ([49], p. 529):** An inference rule is *refutation complete* if, given an unsatisfiable set of clauses, the unsatisfiability can be established by use of this inference rule alone. Every resolution based automated theorem prover uses refutation proof procedure for first-order formulas ([49], p. 517).

- **Proof procedure [49], p. 66):** The *proof procedure* is a combination of an *inference rule* and an algorithm for applying the inference rule to a set of formulas. In first-order logic, a formula that logically follows from a given set of formulas can be produced by using *proof procedures.* The *modus ponens* is a *sound* (but not *complete*) inference rule for first-order logic. There are two kinds of proof procedures namely, *direct proof* and *refutation proof.* Consider a set of formulas $F_1, F_2, \ldots, F_n$. *Direct proof* showed that a new formula F_{m+1} is a logical consequence. That means *direct proof* shows that $(F_1 \wedge F_2 \wedge \cdots \wedge F_n) \rightarrow F_{m+1}$ leads to consistency. Whereas, *refutation proof* adds the negation of the new formula $\neg F_{m+1}$ and proves that $(F_1 \wedge F_2 \wedge \cdots \wedge F_m \wedge \neg F_{m+1})$ leads to an inconsistency. *Refutation proof* is *sound* and *refutation complete.*

- **Substitution ([7], p. 75):** A *substitution* is a function of the form $\{t_1/v_1, \ldots, t_n/v_n\}$, where every v_i is a variable, every t_i is a term different from v_i, and $v_i \neq v_j$, for $i \neq j$. The following two sets are substitutions: $\{f(z)/x, y/z\}$, $\{a/x, g(y)/y, f(g(b))/z\}$, i.e., a substitutes x.

- **Unification ([7], p. 76):** A substitution $\theta = \{t_1/v_1, \ldots, t_n/v_n\}$ is called a *unifier* for a set of expressions $\{E_1, \ldots, E_k\}$ if and only if $E_1\theta = E_2\theta = \cdots = E_k\theta$. The set $\{E_1, \ldots, E_k\}$ is said to be *unifiable* if there is a unifier for it. For example, the set $\{P(a, y), P(x, f(b))\}$ is unifiable since the substitution $\theta = \{a/x, f(b)/y\}$ is a *unifier* for the set.

2.2.1.3 Resolution refutation

In first-order formulas, we have to deal with quantifiers, predicates and functions, consequently, possible worlds for first-order logic cannot be obtained as in propositional logic.

Determining the set of possible worlds from a set of decidable first-order formulas amounts to checking whether a set of first-order formulas is consistent. In practice, it is done using the *resolution refutation method.* This section briefly explains the rules followed by this method for better understanding.

The *resolution refutation* ([49], p. 517) proof procedure answers a query or deduces a contradiction out of the set of clauses where contradiction is represented by the null clause ($\square$). The contradiction is produced by using the modus ponen rule for resolving pairs of clauses from the database. If resolution fails to produce a contradiction directly, then the *resolvent* clause produced by the resolution is added to the database of clauses and the process continues. More precisely, in a refutation proof procedure, the new formula is negated and added to the set of formulas (axioms) that are known to be true. Then, it uses the resolution rule of inference to show that this leads to a contradiction. If the theorem prover shows that the negated goal is inconsistent with the given set of formulas, it follows that the original goal is consistent. A strategy is *complete* if it guarantees to find contradiction using *refutation-complete* inference rule whenever a set of formulas is unsatisfiable ([49], p. 529). Resolution refutation proofs involve the following steps ([49], p. 517):

1. The formulas or axioms are transformed into *clause form.*

2. Goal is negated in clause form and added to the set of axioms.

3. Clauses are *resolved* together, producing new clauses that logically follow from them.

4. Contradiction is produced by generating the empty clause.

5. The aim of the substitutions is to produce a clause whose truth value is opposite to

the negated goal which results in an empty clause.

Binary resolution is the most common form of resolution which is applied between two clauses when one contains a literal and the other one its negation ([49], p.517). Unification is needed if these literals contain variable to make them identical. Eventually, the resulting new clause comprises of the disjuncts of all the predicates in the two clauses minus the literal and its negative instance, having been *resolved away.* The resulting clause undergoes the necessary unification and substitution which make sure the predicate and its negation are *equivalent.* For example, the knowledge base for the *dead dog* problem ([49], p.518) may be represented in clause form as:

$$(\neg dog(X) \vee animal(X)) \wedge (\neg animal(Y)) \vee die(Y)) \wedge (dog(fido)).$$

To this expression we add (by conjunction) the negation of our goal which is $\neg die(fido)$. The resolution proof for this is shown in Figure 1.

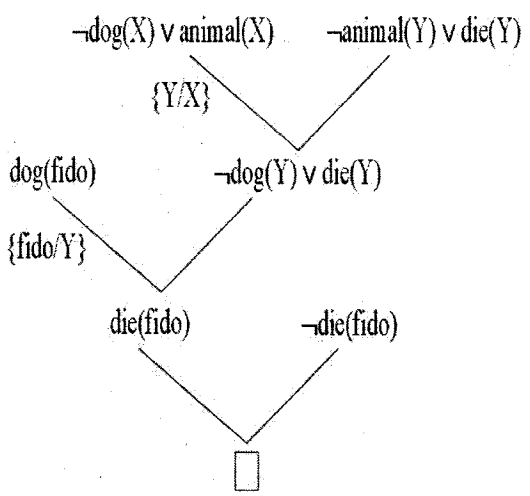

Figure 1: Resolution proof for the *dead dog* problem

Transforming formulas into clause form requires 8 steps which are given below ([7], p. 37):

1. Use the following laws to eliminate the logical connectives $\leftrightarrow$ and $\rightarrow$.

$$F \leftrightarrow G \equiv (F \rightarrow G) \wedge (G \rightarrow F). \tag{13}$$

$$F \rightarrow G \equiv \neg F \vee G. \tag{14}$$

2. Repeatedly use the following laws to bring the negation signs immediately before atoms.

$$\neg(\neg F) \equiv F. \tag{15}$$

De Morgan's laws

$$\neg(F \vee G) \equiv \neg F \wedge \neg G. \tag{16a}$$

$$\neg(F \wedge G) \equiv \neg F \vee \neg G. \tag{16b}$$

and the laws

$$\neg((\forall X)F(X)) \equiv (\exists X)(\neg F(X)). \tag{17a}$$

$$\neg((\exists X)F(X)) \equiv (\forall X)(\neg F(X)). \tag{17b}$$

3. Rename variables so that no two quantifiers bind the same variable.

4. Use the following laws to move the quantifiers to the left of the entire formula to

obtain a prenex normal form.

$$(QX)F(X) \vee G \equiv (QX)(F(X) \vee G). \tag{18a}$$

$$(QX)F(X) \wedge G \equiv (QX)(F(X) \wedge G). \tag{18b}$$

$$(\forall X)F(X) \wedge (\forall X)H(X) \equiv (\forall X)(F(X) \wedge H(X)). \tag{19a}$$

$$(\exists X)F(X) \vee (\exists X)F(X) \equiv (\exists X)(F(X) \vee H(X)). \tag{19b}$$

$$(Q_1X)F(X) \vee (Q_2X)H(X) \equiv (Q_1X)(Q_2Z)(F(X) \vee H(Z)) \tag{20a}$$

$$(Q_3X)F(X) \wedge (Q_4X)H(X) \equiv (Q_3X)(Q_4Z)(F(X) \wedge H(Z)) \tag{20b}$$

For instance let us obtain a *prenex normal form* for the following formula:

$$(\forall X)(\forall Y)((\exists Z)(p(X,Z) \wedge p(Y,Z)) \rightarrow (\exists U)q(X,Y,U)).$$

Using (14), we get: $(\forall X)(\forall Y)(\neg((\exists Z)(p(X,Z) \wedge p(Y,Z))) \vee (\exists U)q(X,Y,U))$.

Using (17b) and (16a), we get: $(\forall X)(\forall Y)((\forall Z)(\neg(p(X,Z) \vee \neg p(Y,Z)) \vee (\exists U)q(X,Y,U))$.

Using (18a), we get: $(\forall X)(\forall Y)(\forall Z)(\exists U)(\neg(p(X,Z) \vee \neg p(Y,Z) \vee q(X,Y,U))$.

So, we obtain the last formula as a prenex normal form of the first formula.

5. Replace each existentially quantified variable by a skolem constant or skolem function

and remove the quantifier $\exists$.

Let us assume variable x is existentially quantified.

case 5a. If x is not universally quantified, replace x by unique, fresh skolem constant.

For example: $\exists X p(X)$ is skolemized to p(a). Similarly,

$\exists X \forall Y p(X, Y, X) \Rightarrow \forall Y p(b, Y, b)$ and

$\exists X \exists Y \forall Z p(X, Y, Z) \Rightarrow \forall Z p(a, b, Z)$.

case 5b. If the predicate has more than one argument and the existentially quantified variable is within the scope of universally quantified variables, the existential variable must be a function of those other variables. For example:

$\exists X \forall Y \forall Z \exists U \forall V \exists W p(X, Y, Z, U, V, W) \Rightarrow$

$\forall Y \forall Z \forall V p(a, Y, Z, f(Y, Z), V, g(Y, Z, V))$.

6. Remove all universal quantifiers as there is no conflicts between the variables.

7. Convert the expression to the CNF form. This requires using the associative and distributive properties of $\vee$ and $\wedge$.

8. Rewrite as a set of clauses, where "$\wedge$" is considered to be default between clauses. For instance, $(p(X, Y) \vee r(X)) \wedge q(Y)$ is written as $(p(X, Y) \vee r(X))$ and $q(Y)$.

2.2.1.4 Strategies and simplification methods for resolution

Resolution is a search problem whose complexity is exponential. Therefore, we have to apply heuristic strategy for large problems. A *strategy is complete* if by using it with a refutation-complete inference rule, it is guaranteed to find the refutation whenever a set of clauses is unsatisfiable ([49], p. 529). Let us discuss some of the strategies used in resolution.

- **Breadth-first strategy ([49], p. 529):** In *breadth-first search* an exhaustive clause comparison is done. Therefore, it guarantees to find the shortest solution path. It is also a *complete* strategy in the sense that if it continues to search long enough, it is guaranteed to find a refutation if one exists.

- **Set of support strategy ([49], p. 530):** *Set of support* of Wos and Robinson [72] is a good strategy for large clause space. In this strategy, a subset T is specified from a set of input clauses S for resolution. It can be proved that, if S is unsatisfiable and $S - T$ is satisfiable, then this strategy is *refutation complete.* It is complete only if the right subset is chosen.

- **Unit preference strategy ([49], p. 531):** The *unit preference strategy* usually resolves with shorter clauses specially, clauses with one literal, called unit clauses whenever they are available. It guarantees that the resolvent is smaller than the largest parent clause. *Unit resolution* is a related strategy that requires that one of the resolvent always be a unit clause. However, it is an incomplete strategy. The combination of unit preference and set of support strategy can produce a more efficient complete strategy.

- **Linear input from strategy ([49], p. 531):** *Linear input from strategy* directly use negated goal and the original set of formulas. It takes the negated goal and resolves it with one of the formula at a time. This process repeats until the empty clause is produced. This is an incomplete strategy.

We can use some simplification methods in order to reduce the search space and speed up the resolution-based problem solver for finding a solution. Let us describe few of them

below:

- **Elimination of tautological clauses ([49], p. 533):** Any tautological clause need not be considered as it will never be falsified. Therefore, it is not useful in a solution attempt.

- **Subsumption ([49], p. 533):** *Subsumption* is a complete strategy in which more specific clauses are removed, e.g., if S contains predicates father(X, Y) then we can remove father(istiaque, rushmi).

2.2.2 Optimization form

In his paper on probabilistic logic, Nilsson [56] also discussed the optimization form of FOPL that can be stated as follows: Given a set $\mathcal{F}$ of m formulas in prenex normal form, and a probability vector π associated with these formulas, such that the set $(\mathcal{F}, \pi)$ is consistent. Let us consider an additional formula F_{m+1}. How to compute bounds on the probability of F_{m+1} so that the system $(\mathcal{F} \cup F_{m+1}, \pi)$ remains consistent. We next discuss in detail the FOPL problem, how to express this problem in mathematical program and its possible solution schemes in Chapter 4.

Chapter 3

Literature Review

According to Smithson [68], uncertainty is a subjective measure of certainty of something (e.g., occurrence of some events) therefore, it can be modelled in terms of a quantitative measure, i.e., a numerical value between 0 and 1 where 0 denotes falsity and 1 denotes truth. Klir and Yuan ([43], p.267) showed reasoning under uncertainty is a well recognized issue in several theories, for instance, probability theory [18, 17], fuzzy set theory [73], possibility theory [74, 16], etc. In this chapter, we will briefly discuss about few uncertainty based models in AI for propositional, as well as for first-order logic. Focus will be on the *probabilistic logic* model (equivalent to Nilsson's [56] *probabilistic satisfiability* and *probabilistic entailment problems*) and its extensions.

This chapter is organized as follows. In Section 3.1, we briefly describe some models which treat uncertainty by different means other than probability. Analytical and numerical solutions to probabilistic logic problem are discussed in Section 3.2 and Section 3.3 respectively. In Section 3.4, we focus on first-order logic. There are two parts in this section. In the first part of Section 3.4, we provide a review on reasoning under uncertainty models

with probabilities in first-order logic. In the second part of Section 3.4, we focus on some key features of some recently used practical and scalable theorem provers, along with some of their adaptive strategies.

3.1 Reasoning under Uncertainty Models

In artificial intelligence, reasoning under uncertainty has long been studied with different perspectives. It has been argued that probability theory is not sufficient to deal with uncertainty in AI [75]. Fuzzy sets [73, 14], possibility theory [74, 16] were few alternative proposed models. Besides these alternative formalisms, for combining uncertainty measures or estimates, many specific rules were also elaborated, e.g., the certainty factor of MYCIN [6]. Few papers have been published in defense of such disbelief and alternate solutions [8, 60]. Successful model by Nilsson's [56] paper on "Probabilistic Logic" and Lauritzen and Spiegelhalter's [47] paper on local computation on Bayesian networks stimulated a return in favor of probability logic. We will discuss fuzzy logic and fuzzy sets, Bayesian logic briefly in this section.

3.1.1 Fuzzy logic and fuzzy sets

Theory of Fuzzy sets was proposed in 1965 by Lotfi Zadeh at the University of California, Berkeley [73] and fuzzy logic is a consequence [43]. Fuzzy sets theory provides the basis for fuzzy logic where each proposition (or formula) is assigned a value between 0 and 1 which is called the *degree of fuzziness or membership degree* (1 means full membership, 0 means no membership and anything in between, e.g., 0.5 is called intermediate membership). In

classical logic, a truth value is either 0 or 1 assigned (0 represents *false* and 1 represents *true*) to a formula.

Uncertainty is expressed differently in fuzzy logic and in probability. In probability theory, probability is usually defined in a likelihood of some events or conditions to occur in a set (i.e., how probable do I think that a variable is in a set). Whereas, fuzzy truth represents membership in vaguely defined sets (i.e., how much a variable is in a set). For example, if a climber climbs 600 ft of a 1000 ft hill, then, two fuzzy sets, *Bottom* and *Top* can be defined. One might define the height climbed by the climber as being 0.4 Bottom and 0.6 Top. As the concept of *height* is linguistic, the definition of height (Top and Bottom) will be depend on the observer or the designer. Another observer might equally consider a set membership function where the hill would be considered Top for all values above 500 ft. On the other hand, in probabilities, at first, a variable for the height of the hill would be defined, and second, distributions describing the probability that someone would call the hill as Top given a specific height level.

Fuzzy logic also allows set of membership values in its linguistic term, imprecise concepts like *slightly*, *quite*, *very*, etc. Specifically, it allows partial membership in a set. In other words, linguistic terms can be modified by using some special linguistic terms which are called *linguistic hedges* (or simply *hedges*) in fuzzy logic (see [43], p.229). For example, if a fuzzy proposition x *is* F and a *hedge* H are given we can conclude x *is* HF, where HF is the fuzzy set obtained by applying H to the given set F. Any hedge H is usually interpreted as a unary operation h on the interval [0, 1] (see [43], p.230). For example, hedge *very* is often interpreted as $h(a) = a^2$ whereas, *fairly* interpreted as h(a) = $\sqrt{a}$ ($a \in [0, 1]$). These hedges are called *modifier* (see [43], p.229). Assume, John is 26 and fuzzy set YOUNG(26) = 0.8.

Then we can say VERY YOUNG(26) = $(0.8)^2 = 0.64$.

For fuzzy inference, fuzzy logic use their own fuzzy rules, usually IF-THEN rules. Based on the fuzziness of a set of given formulas, the *degree of fuzziness* of a new formula is computed by using fuzzy set operators, fuzzy logical functions and fuzzy rules. For example, it is possible to approximately generalize the classical inference rule *modus ponens* to fuzzy inference. *If the banana is ripe, then the color of banana is yellow* and *the banana is very ripe* are given, by fuzzy inference we can conclude *the color of banana is very yellow.*

Fuzzy inference properties depend on the choice of fuzzy logical functions [14] such as *minimum*, different *triangular norms*, involved in composition of formulas. This choice is made more or less arbitrarily in many applications of fuzzy logic. These arbitrariness of fuzzy choices, caused by the lack of clear semantics, is identified as the major drawback of fuzzy logic by Diez *et al.* [14]. According to their observation, there is no clear way of determining which membership degree to use in building a specific real world application even though there are several definitions of *membership degrees.* All fuzzy systems assign numbers between 0 and 1 but the semantics and the way of assigning those numbers vary significantly in different applications. Moreover, it is not clear how fuzzy logic justifies whether a given set of formulas with their degree of membership values are consistent or not. Besides these drawbacks, due to the simplicity of modeling and fast decision making ability, fuzzy logic has become extensively useful in automatic control and widely used in industry from small appliances to large systems. Fuzzy logics are also extended by adding universal and existential quantifiers in a manner similar to the way that first-order logic is created from propositional calculus [48].

3.1.2 Bayesian network

A set of variables and their probabilistic dependencies are represented by a graphical model named *Bayesian network* (or a *belief network*). For example, the probabilistic relationships between diseases and symptoms can be represented by a Bayesian network. Given some symptoms associated with probabilities, the network can be used to compute the probabilities of the presence of various diseases. More formally, in a Bayesian network [61], $G = (V, U)$ where vertices or nodes $v_j \in V$ are associated with simple events (or logical variable $x_j \in X$ having only two possible outcomes, either true or false), whereas the directed arcs (v_i, v_j) are used to represent probabilistic dependency among the nodes or variables. In other words, Bayesian networks are usually directed acyclic graphs [61] whose nodes represent random variables, and whose arcs represent conditional dependencies among the variables. Two directly connected nodes have parent-child relationship [13] where parent has influence on the child node. i.e., the probability that a variable is true depends on its immediate predecessor variables. Initially, each node is specified with an unconditional probability distribution which is often called the *belief function* or *evidence* of that node. When a new piece of information or a new finding arrives, the belief functions attached to all relevant nodes in the network are updated.

In Bayesian networks, the general probabilistic inference problem of finding the probability of an event, given a set of prior evidence, can be solved by sequential applications of Bayes Theorem. Unless the immediate predecessors are outnumbered, this leads to an easy way to compute the probabilities of events. Models and examples of performing inference in Bayesian networks are provided by Pearl [61], Lauritzen and Spiegelhalter [47]. However, to define a unique point probability distribution, they assumed that sufficient information

for all the predecessors of a node is provided.

Anderson and Hooker [1] examined how to relax some of these assumptions by combining Bayesian networks with probabilistic satisfiability and proposed a more complicated model. It has been shown by Hansen *et al.* [35] that, the usual computation in Bayesian networks can also be mapped into probabilistic entailment problem with probability intervals (11).

3.2 Analytical Solution of PPL

In this section, we present studies on analytical solution for *Propositional Probabilistic Logic* problem or PPL in short. At first, we will discuss Boole's analytical solution [5] in algebraic form to PPL problem. Later on, we will discuss about Hailperin's extensions of Boole's method. Hailperin was one of the first who investigated Boole's analytical solution. He pointed out that there is a way to find an analytical expression for the solution using the enumeration of the extreme points of a particular polyhedron. Let us start with Boole's algebraic method.

3.2.1 Boole's algebraic method

To solve the decision and optimization forms of probabilistic logic problem analytically, Boole outlined some algebraic manipulations in his book in 1854 [5], and in several of his contemporary and subsequent papers which are similar for both cases. The simplest and most efficient method carries on as follows [32] :

Step 1. All m logical sentences are expressed as sum of complete products, i.e., products of all variables in direct or complemented form.

Step 2. Associate each of these products with an unknown probability p_j, write linear equations such that for each associated logical sentence $\sum_j p_j = \pi_i$ for S_i to be true. Noted that $j \leq 2^m$, because not all the products are valid and some of the products may be duplicated. Add constraints stating that $\sum_j p_j = 1$ and each p_j is non-negative.

Step 3. Eliminate as many p_j as possible from the so obtained equalities and inequalities (we will see in an example later).

Step 4. From the inequalities obtained in the previous steps eliminate the remaining probabilities p_j as well as π_{m+1} by considering all upper and lower bounds on one of them, stating that each lower bound $<$ upper bound, removing redundant constraints and iterating.

Thus we obtain relations involving $\pi_1, \pi_2, \ldots, \pi_m$ which are called *conditions for feasible experience* by Boole. Moreover, these relations involve π_{m+1} which gives the best possible bounds on the probability of the additional logical sentence. This is known as the solution to Boole's *general problem.* Let us see an example to solve a problem by using the above steps 1 to 4.

Example 1 (Boole [4]) Find the best possible bounds on the probability of $S_6 = x_3$ considering the following:

$$prob(S_1 \equiv x_1) = \quad \pi_1$$

$$prob(S_2 \equiv x_2) = \quad \pi_2$$

$$prob(S_3 \equiv x_1x_3) = \quad \pi_3$$

$$prob(S_4 \equiv x_2x_3) = \quad \pi_4$$

$$prob(S_5 \equiv \overline{x}_1\overline{x}_2x_3) = 0.$$

Step 1, gives

$$x_1 = \quad x_1x_2x_3+x_1x_2\overline{x}_3+x_1\overline{x}_2x_3+x_1\overline{x}_2\overline{x}_3$$

$$x_2 = \quad x_1x_2x_3+x_1x_2\overline{x}_3+\overline{x}_1x_2x_3+\overline{x}_1x_2\overline{x}_3$$

$$x_1x_3 = x_1x_2x_3+x_1\overline{x}_2x_3$$

$$x_2x_3 = x_1x_2x_3+\overline{x}_1x_2x_3.$$

Set $p_1 = prob(x_1x_2x_3)$, $p_2 = prob(x_1x_2\overline{x}_3)$, $p_3 = prob(x_1\overline{x}_2x_3)$, $p_4 = prob(x_1\overline{x}_2\overline{x}_3)$, $p_5 = prob(\overline{x}_1x_2x_3)$, $p_6 = prob(\overline{x}_1x_2\overline{x}_3)$, $p_7 = prob(\overline{x}_1\overline{x}_2x_3)$, $p_8 = prob(\overline{x}_1\overline{x}_2\overline{x}_3)$. Step 2 yields the following equalities and inequalities:

$$
\begin{array}{llll}
p_1 + p_2 & +p_3 + p_4 & & = \pi_1 \\
p_1 + p_2 & & +p_5 + p_6 & = \pi_2 \\
p_1 & +p_3 & & = \pi_3 \\
p_1 & & +p_5 & = \pi_4 \\
 & & & +p_7 \quad = 0 \\
p_1 + p_2 & +p_3 + p_4 + p_5 + p_6 + p_7 + p_8 & & = 1 \\
p_1, p_2, \ldots, p_8 & \geq 1 & &
\end{array}
$$

Eliminating successively the variables $p_7, p_4, p_3, p_6, p_5, p_1$ and p_2 yields, at the end of Step 3, the bounds

$$\max\{\pi_3, \pi_4\} \leq \pi_6 \leq \min\{1-\pi_1+\pi_3, \pi_3+\pi_4, 1-\pi_2+\pi_4\}$$

and the consistency conditions

$$\pi_1 \leq \pi_3 \quad \text{and} \quad \pi_2 \geq \pi_4.$$

Eliminating π_6 yields the additional condition

$$\pi_1 - \pi_3 + \pi_4 \leq 1.$$

Hailperin [27] extended and provided a systematic treatment for Boole's algebraic method to deal with conditional probabilities. Hailperin showed [27] that, the extensions can be

done in two ways, either using conditional probability in the objective function or using conditional probability in the constraints.

3.2.2 Polyhedral methods

This method has been contrived for obtaining an analytical solution of probabilistic logic problem which is different from Fourier-Motzkin elimination. It is based on the study of dual polyhedra for (8). Let us consider the following two linear programs (LP^1, LP^2):

$$LP^1 = \begin{cases} \min\ y_0 + \pi y \\ \text{subject to: } \mathbb{1}y_0 + A^t y \geq A^t_{m+1}. \end{cases}$$

$$LP^2 = \begin{cases} \max\ y_0 + \pi y \\ \text{subject to: } \mathbb{1}y_0 + A^t y \leq A^t_{m+1}. \end{cases} \tag{21}$$

It can be observed that, the polyhedron defined by (21) includes the vector $(1, 0)\big((0,0)\big)$ respectively, therefore, the corresponding polyhedra are non empty. In order to obtain the condition under which the probabilities are consistent, consider the dual of the probabilistic satisfiability problem in decision form (5), with a dummy objective function, $0p$, to be maximized:

$$\begin{aligned} &\min \quad y_0 + \pi y \\ \text{subject to:} \quad &\mathbb{1}y_0 + A^t y \leq 0. \end{aligned} \tag{22}$$

Hansen, Jaumard and Poggi de Arago, [33] showed that:

Theorem 1. *The probabilistic satisfiability problem (5) is consistent if and only if*

$$(1, \pi)^t r \leq 0 \tag{23}$$

for all extreme rays r of (22).

Using the duality theorem of linear programming, Hailperin [25] showed that:

Theorem 2. *The best lower bound for π_{m+1} is given by the following convex piecewise linear function of the probability assignment:*

$$\underline{\pi}_{m+1}(\pi) \quad = \quad \max_{j=1,2,3,\ldots,k_{max}} \quad (1, \pi)^t y^j_{max} \tag{24}$$

where $y^j_{\max}$ for all j represent the k_{max} extreme points of (21).

The best upper bound for π_{m+1} is given by the following concave piecewise linear function of the probability assignment:

$$\overline{\pi}_{m+1}(\pi) \quad = \quad \min_{j=1,2,3,\ldots,k_{min}} \quad (1, \pi)^t y^j_{min} \tag{25}$$

where $y^j_{\min}$ for all j represent the k_{min} extreme points of (21).

This result provides best possible bounds on $\underline{\pi}_{m+1}$ and $\overline{\pi}_{m+1}$. Both theorems 1 and 2 readily extend to the case of probability intervals (11) but not to the case of conditional probabilities [32].

3.3 Numerical Solution of PPL

Numerical methods are needed to assess the solution of the *Propositional Probabilistic Logic* problem (or PPL in short) in practice as analytical solution can be used only for very small instances. Numerical solution methods for P^{PSAT} and P^{PENTAIL} can be categorized into two types namely, *exact* and *heuristic* methods. Let us briefly describe what are these exact and heuristic methods.

An *exact* algorithm guarantees to find the best optimal solution if there exists one whereas, a *heuristic* algorithm usually only finds a good or near optimal solution without any information on how far it is from the optimal solution. However, it may take too much time to solve large sized PPL problems using exact method. Therefore, one might need to use heuristic method in order to find a solution in a reasonable amount of time.

In the subsequent sections, we will discuss about the *linear programming* solution with simplex or revised simplex algorithm ([10], p.97) and the linear programming solution with *column generation* technique ([10], p.195), both of them correspond to exact method. Next, ADPSAT or AD-SOLFOPL (based on anytime deduction) and *variable neighborhood search* which fit into the category of heuristic solution methods are also discussed. Linear programming and column generation methods can solve PPL problem comprised of instances of conditional and unconditional probabilities.

3.3.1 Linear programming modeling

It has been already shown in Chapter 2 that, the probabilistic satisfiability or the probabilistic entailment problem can be reformulated as a linear program. The simple method

for solving (5), (8), (11), (30), (31), (32) and (33) problems include firstly searching for the overall set of possible worlds (see chapter 2), and secondly, solving the resulting linear program. We can solve the linear program by using simplex or revised simplex algorithm [10]. Although simple, this method can be used only for small instances since the size of the input grows exponentially with the increase of the number of logical variables $x_1, x_2, \ldots, x_n$. In view of the enormous size of these programs (about 10^{18} columns for $\min\{m, n\} = 60$, where m is the number of sentences and n is the number of logical variables in these sentences), it is impossible to solve large sized probabilistic satisfiability or probabilistic entailment problem using this method. This led Nilsson [57] to suggest looking for heuristic solution methods only. Such a view is overly pessimistic as we will see in the next section.

3.3.2 Column generation solution

When linear programs cannot be solved using the simplex or revised simplex algorithm, tools from large scale optimization come to rescue. In operations research, there exist some tools, namely the column generation which deals with solving linear programs with a huge number of variables. In this section, we will see how column generation technique works using a description from [40].

Column generation starts by solving a small, manageable part of a problem (few of the possible worlds), by analyzing this partial solution it determines an additional small subset of the worlds (one or more possible worlds) to be added in the model, and then solves the enlarged model. This column-wise modeling repeats this process until it satisfies an optimality condition for the problem. More formally, column generation technique extends the revised simplex algorithm (see e.g., Chvatal [10], p.97), in which only a small number

of columns are kept explicit, by determining the entering column through the solution of an auxiliary subproblem.

To describe the general principle of column generation technique, let us assume, without the loss of generality, a linear program (26) for a minimization problem in order to get lower bound in the optimal solution:

$$\begin{cases} \min\ cx \\ \\ \text{subject to:} \quad Ax \geq b \\ \\ \qquad\qquad\quad x \geq 0. \end{cases} \tag{26}$$

The variable x of the equations of (26) form the columns of the matrix A. Therefore, variables and columns are used with the same meaning. By using either the simplex algorithm or the column generation technique, the optimal solution (which provides a lower bound) of equation (26) can be obtained. However, when the number of variables is very large, it is better to use the column generation methods.

In column generation methods, the initial linear program is decomposed into a restricted linear program and a pricing problem. The restricted linear program is called the *restricted master* problem which corresponds to a linear program associated with a restricted matrix A' such that, A' is a sub-matrix of A. In principle, if there exists an optimal solution for the restricted linear program, it may also be an optimal solution for the initial linear program. However, it depends on the signs of the reduced costs of the missing columns, i.e., columns which are not explicitly considered in equation (26).

Column generation methods proceed as follows: the linear program corresponding to a sub-matrix $A^1(= A')$ of dimension $m \times n_1$ of the original matrix is solved by using CPLEX,

and an feasible solution x^1_{LP} is obtained. Note that, the columns of A^1 is denoted by c^1 which is a sub-vector of c.

$$\begin{cases} \min\ c^1 x \\ \\ A^1 x \geq b \\ \text{subject to:}\ 0 \leq x \leq 1 \\ \\ x \in \mathbb{R}^{n_1}. \end{cases} \tag{27}$$

Does there exist a column $a^j \in A \backslash A^1$ such that $\overline{c}_j < 0$? i.e., can we find a column of the matrix $A \backslash A^1$ for which the reduced cost is negative? If the answer is *no*, then the feasible solution x^1_{LP} is optimal for equation (26). If the answer is *yes*, we must add one or more columns to the matrix A^1 in order to find the optimal solution. Therefore, we need to solve the following system x^2_{LP}:

$$\begin{cases} \min\ c^2 x \\ \\ A^2 x \geq b \\ \text{subject to:}\ 0 \leq x \leq 1 \\ \\ x \in \mathbb{R}^{n_2}. \end{cases} \tag{28}$$

with $A^2 = A^1 \cup \{a^j\}$ such that $\overline{c}_j < 0$.

This process is repeated until a column j with a negative reduced cost is found, i.e., $\overline{c}_j \leq 0$. However, if no more iteration is possible, we conclude that we have obtained an optimal solution for equation (26).

Now in order to find if there exists a column with a negative reduced cost one has to solve the so-called *pricing problem*. Considering the master problem, e.g., (27), let us assume that we want to find if there exists a column a^j with a negative reduced cost. So,

we must solve the pricing problem according to:

$$\begin{cases} \min\ \overline{c}(a^j) \\ \text{with constraints on the component of } (a^j) \\ \text{in order to guarantee that} A^2 \subseteq A, \end{cases} \tag{29}$$

where A^2 is the concatenation of A^1 and $a^j : A^2 = (A^1 \mid a^j)$.

For the master problem, the reduced cost is defined in the matrix form as follows:

$$\overline{c} = c - vA$$

where v is the vector of the dual variables associated with equation (26). For the master problem, we obtain:

$$\overline{c}_j = c_j - v^1 a^j = c_j(a^j) - v^1 \cdot a^j$$

for the column j where $\overline{c}_j$ is the reduced cost of the column j, v^1 is the optimal dual vector obtained when solving equation (27).

Solving the pricing problem is equivalent to solving the following problem:

$$\overline{c}^* = \min\{\overline{c}(a) = c(a) - va : a \in \mathcal{A}\},$$

where $\mathcal{A} = \{a \in \mathbb{R}^m : (A^1 \mid a)$ is a sub-matrix of $A\}$ and $\overline{c}^*$ is the best known reduced cost.

The column generation process is summarized in Table 1.

The pricing problem (29) is often a NP-complete problem which is very difficult to solve. However, it is not mandatory to solve the pricing problem exactly at each iteration. In order

Table 1: Column generation process

Create an artificial solution
Solve the restricted master problem
WHILE (there exists a column a_j with a negative reduced cost)
Include a^j to the restricted master problem
Solve the restricted master problem

to ensure an iteration of the revised simplex algorithm to take place, it is enough to find a column with a negative reduced cost. Therefore, a heuristic algorithm can be designed for finding such a column. If a feasible solution is obtained heuristically, the decision form of the probabilistic logic problem is solved. However, finding no feasible solution by choosing the entering column in a heuristic way cannot guarantee that none exists. Therefore, when no more column with negative reduced cost is obtained heuristically, it is necessary to turn to an exact algorithm to prove that, there is no feasible solution for the decision form of the probabilistic logic problem. In addition, also to prove that, there exists no feasible solution which gives a better bound than the incumbent one for the optimization form of probabilistic logic problem. The detailed expression for the solution of the pricing problem in optimization form of the first-order probabilistic logic problem is described in Section 4.4.

To find an approximate optimal solution, *Steepest-Ascent-Mildest-Descent* (SAMD) or Tabu search heuristic by (Hansen and Jaumard [31], Glover [21, 22]) may be used which is a proven meta-heuristic optimization method. Whereas to find an exact optimal solution, a variant of the *Basic algorithm* of Hammer *et al.* [29], Hammer and Rudeanu [30], or the *Basic algorithm revisited* of Crama *et al.* [12] may be used.

In order to start the column generation technique either a feasible solution or an artificial

solution is needed. Usually it starts with an artificial solution which can be generated in an insignificant time. The artificial solution corresponds to a set of columns (tiny compare to the number of variables), which constitute a square matrix as large as the number of constraints. It is to be noted that, in order to minimize the objective function, the algorithm needs a negative reduced cost, but if the aim is to maximize the objective function, then a positive reduced cost is required.

3.3.3 ADPSAT and AD-SOLFOPL

Alternative solution methods have been also proposed in order to solve the two basic problems of probabilistic reasoning namely, the probabilistic satisfiability and the probabilistic entailment problems. For instance, *anytime deduction* by Frisch and Haddawy [19] is one of such proposed solution method which proceeds by computing increasingly narrow probability intervals that contain the tightest probability interval. This approach is *deductive* as it can derive final bounds sequentially. In addition, it is called *anytime* because we can get a partial solution (not necessarily the optimal one) any time we stop (before the final step). However, Frisch and Haddawy [19] neither provide an explicit and well defined procedure to perform this deduction nor discuss the consistency issue. Hansen *et al.* [34] proved that, their procedure could not detect inconsistency. Moreover, another drawback of anytime deduction [19] is that, it often does not provide the tightest probability interval bounds when solving the probabilistic entailment problem.

Jaumard and Parreira [41] proposed an explicit deductive procedure, called ADPSAT, using their previous work [33] on the analytical solution of Nilsson's [56] probabilistic logic

problem. In ADPSAT, only a small subset of logical sentences (typically one or two) together with the interval probability values of a small subset of variables (typically one to four) are considered at each step to determine the probability interval values of either a selected variable or a selected sentence. Afterwards, another subset of sentences is considered to compute the probability interval values and the newly computed probability value is compared with the previously computed probability values. By repeating this deductive mechanism, the final tight probability interval value is determined.

ADPSAT is capable of checking inconsistency of a given set of sentences very quickly but it does not always guarantees to do so. However, in practice, ADPSAT is able to find very often the tightest probability bounds, for instance, usually when the sentences contain two variables. The entailment is achieved using a sequential deductive approach. Finally, ADPSAT has solved, by far, some of the largest instances with 1000 variables and 2500 sentences for a reasoning under uncertainty model based on probability theory.

In their approach [41], the authors have used an ordered set of well thought combinations of small number of sentences and variables which they call *primitives*. When this approach is used to solve PPL instances with at most three variables, only a small number of primitives are needed here.

It is to be noted that, the ADPSAT was aimed to solve propositional instances of large size which has now also been extended to the case of first-order logic, called AD-SOLFOPL [67]. ADPSAT and AD-SOLFOPL are both *heuristic* solution methods whereas, our proposed approach to solve probabilistic reasoning problems is an *exact* solution method (which is an extension of PSATCOL for propositional to first-order logic) where we use *column generation techniques*. Therefore, computational experiments have been conducted to compare the

proposed algorithm of AD-SOLFOPL with our proposed algorithm, both with respect to the range of probability interval values and the computing times.

3.3.4 Variable neighborhood search

Variable Neighborhood Search, or VNS in short, was introduced by Mladenović and Hansen [53]. VNS is a meta-heuristic which helps escaping when the search process is trapped in a local optima by changing the neighborhood structures systematically. Initially, a set of neighborhood structures is preselected, a stopping condition is determined and an initial local solution is found. In the main loop of the method, it search for a better solution by changing the neighborhood structures using three process (i) shaking, (ii) local search and (iii) move or not. It moves to a new solution from the current best known solution if the new one is better or it may accept a worse solution with certain probability in order to escape the local optima and move towards the global optima.

For solving the probabilistic logic problem, Jovanović *et al.* [42] suggested using VNS based heuristic. Hansen and Perron [36] use VNS to generate multiple columns (or possible worlds [56]) with negative reduced cost simultaneously at each iteration of the column generation method. Adding multiple columns at each iteration instead of only one column reduces the number of iteration and speeds up the algorithm [36].

3.4 First-Order Logic

In this section, we will focus on first-order logic. We divide our review for first-order logic into two parts. In the first part, a review is given based on reasoning under uncertainty

models with probabilities. In the second part, we briefly describe some procedures and strategies that are adapted in several present-day theorem provers for efficient satisfiability checking in first-order logic.

3.4.1 Reasoning under uncertainty models with probabilities

In this section, we will review some first-order models with probabilities. Several papers have been published on the first-order probabilistic logic [28, 55, 50, 59]. We will discuss a few of those models which are related to our work on models for reasoning uncertainty with probabilities. In the subsequent sections, we will cover Halpern's [28] *degrees of belief* and *chance setup*, Lukasiewicz's [50] *probabilistic logic programming* and Milch and Russell's [52] *Bayesian logic* or BLOG.

3.4.1.1 Halpern's semantics for probabilistic logic

Halpern [28] provided semantics to first-order logics for two different approaches of probabilistic reasoning. One of them deals with the probability in the domain or at the level of statistical information which is illustrated with the statement "The probability that a randomly chosen bird flies is greater than 0.9". The other one deals with uncertainty at the level of possible outcomes or worlds (Nilsson [56]). Halpern illustrated the second one with the statement "The probability that Tweety (a particular bird) flies is greater than 0.9". The first approach of Halpern is also called "*chance setup*" [24] while the latter one is called "*degrees of belief*" [3]. A similar type of semantics was also studied by Bacchus [3].

For the probability on domain, Halpern [28] assumed that there is a given first-order language for reasoning about some domain. If a formula φ in logic is given, formulas of

the form $w_x(\varphi) \geq 1/2$ are also allowed which can be interpreted as "the probability that a randomly chosen x in the domain satisfies φ is greater than or equal to $1/2$". For instance [28], $w_x(Son(x,y))$ describes the probability that a randomly chosen x is the son of y, whereas $w_y(Son(x,y))$ describes the x is the son of randomly chosen y. Halpern extended this to allow arbitrary sequences of distinct variables in the subscript by providing the syntax and semantics of a two classified languages. The function and predicate symbols, and a limited class of object variables $x^0, y^0, \ldots$, in the φ are considered in the first class which describe the elements of the domain of reasoning. The second class defines syntax for binary function symbols $+$ and $\times$, constant symbols 0 and 1, and limited class of field variables $x^f, y^f, \ldots$, with the intention of ranging over the real numbers. Halpern [28] allowed only two field functions $+$ and $\times$ in his syntax and did not consider Bacchus's [3] *measuring functions* which map object terms into field terms.

In the semantics of chance setup, Halpern [28] gave a probability structure called *type 1*, where probability functions are defined over the domain. These probability functions are standard real-valued and countably additive which make them significantly different from the semantics of Bacchus [3] where non-standard probability functions are considered which take values in arbitrary ordered fields and are only finitely additive.

The syntax and semantics for *degrees of belief* is essentially the same as in the former approach except for few cases. Among them, probability is defined over the set of states or possible worlds [56] instead of taking over domain. The functions or predicates might have different meaning for different states. The probability structure called *type 2* is defined for reasoning about possible worlds. Some simplifying assumptions were also made for the representation of probability on possible worlds. For example, all functions and predicates

can take fixed or flexible structure. Moreover they also assume that there is only one domain and only one predicate measure on the set of states.

Halpern [28] used a deductive mechanism to infer new information for *chance setup* and *degrees of belief*. Deductive mechanism (e.g., modus ponens) usually concludes a new information from a given certain premises. Halpern [28] used an axiom system which is sound, but is complete only for bounded sized domains. For instance, in order to provide complete axiomatization, problems are restricted to unary predicates for the probability on domain. However, there is no straightforward way to capture statistical information from degrees of belief or vice-versa. Therefore, in order to simultaneously reason about statistical information and degrees of belief, these two approaches are combined by Halpern [28] in one framework.

3.4.1.2 Probabilistic logic programming

Similar to Halpern's [28] semantics to formulas that describe *degrees of belief*, Ng and Subrahmanian [55] proposed a new approach called *probabilistic logic programming* where uncertainty is handled by defining probability distribution over the set of possible worlds [56]. It was further investigated by several authors, see e.g., [45, 46], and more recently by Lukasiewicz [50]. In order to deduce a tight probability bounds for an additional clause, Lukasiewicz proposed two solutions. As a first solution, he [50] showed that it is possible to compute the tight bounds by using straightforward linear program. However, he also proved that when the number of variables (or possible worlds) increases, the solution grows exponential. Therefore, Lukasiewicz [50] proposed another solution technique to generate linear programs that generally have a much lower number of variables. For this purpose, he

partitioned the probabilistic logic program ($\mathcal{P}$) into a set of *logical program* clauses ($\mathcal{L}$) and a set of purely *probabilistic program* clauses ($\mathcal{P}\backslash\mathcal{L}$) and finally solve them using two linear programs. This linear programming approach for probabilistic deduction shows efficient results only in restricted cases. Whereas, there is no clause level division in our case, i.e., we kept the clauses intact. Moreover, to increase scalability, we use column generation method which relies on a decomposition of the initial linear program into a master and pricing problem (see Section 3.3.2).

3.4.1.3 Bayesian logic or BLOG

Milch and Russell [52] pointed out that in order to express the real world problem, propositional probabilistic languages such as *Bayesian network* or BN are inadequate.

This inadequacy of BN results from the fact that a fixed set of random variables as well as dependencies and probability distributions for each of the variables must be specified individually. Real world problems often involve many objects and their dependent objects which are unknown or uncertain in nature. So, to define uncertainty using a fixed set of random variables and a fixed dependency structure is not sufficient. For example tracking multiple people from a video sequence, is very difficult to define using a fixed set of variables and structures as the number and the types of people are unknown in advance.

Milch and Russell [52] proposed a new probabilistic modeling language, called *Bayesian logic* or BLOG which can model large families of random variables compactly by abstracting over objects and mapping between objects and observations. The authors [52] define probability distribution over relational structures with varying sets of objects. For a particular scenario, BLOG specifies certain non-random aspects which is handled by a typed

first-order language and the remaining aspects are specified with a probability model. The probability model describes a generative process for constructing a possible world in two steps. At the first step, boolean functions are assigned values to evaluate some objects. In the second step, new objects are added to their world.

In order to infer, Milch and Russell [52] use a sampling-based inference algorithm which proved to be too slow for many tasks. Therefore, they are still working on improving their inference mechanism.

3.4.2 Practical and scalable procedures for satisfiability

In first-order logic, automated theorem provers are used to prove the satisfiability of a set of formulas. In order to serve our purpose of checking satisfiability of a set of given first-order formulas, we were searching for a suitable and efficient theorem-prover. We decide to focus on the theorem provers which participated in the CADE ATP System Competition. Among them, Vampire, SPASS, Theo are the most well reputed theorem provers in the international competitions. Next, we go through their underlying assumptions and strategies. For example, vampire uses saturation with resolution and paramodulation; SPASS combines saturation with superposition (a variant of demodulation), conventional splitting with branching and backtracking; and Theo uses resolution refutation. More detail of Theo is presented in Section 4.6.

We have already discussed *refutation* proving, some strategies and simplification methods for *resolution* proving in Section 2.2.1.4. In order to make resolution more efficient, several sophisticated *inference rules* have also been developed and applied in the efficient theorem provers. In this section, we will discuss two of such techniques: *paramodulation*

and *saturation.* Moreover, some recent adaptive strategies that are used to improve the performance of resolution based first-order theorem provers are also discussed in the last section.

3.4.2.1 Paramodulation

Control of equality is one of the important and difficult issues in designing an automated theorem prover. Mathematical facts and relationships are often represented in multiple forms which can be obtained by applying associative and communicative laws. For example, (3 + 6 + 9) can also be represented as (3 + (6 + 9)) or (3 + (6 + 0) + 9). *Demodulation* (see [49], p. 548) is the process which rephrases or rewrites expressions such that they automatically take on a chosen canonical form. The *canonical* form means reducing to the simplest and most significant form possible without the loss of generality. The unit clauses which are used to reduce the set of expressions to its canonical form is called *demodulators.* The task of these demodulators is to specify equality relations of different expressions, so that, an expression can be transformed into its canonical form. Therefore, demodulators can be thought of as a pre-processor which turn the set of clauses into a simple and reduced format before placing them into the clause space. For example (see [49], p. 548), let us consider a demodulator *equal(father(father(x)), grandfather(x))* and the new clause *age(father(father(nazma)), 70).* By applying the demodulator before adding this new clause to the clause space we can add *age(grandfather(nazma)), 70)* instead. Therefore, the equality problem here was in the naming. Similarly, other family relationships such as *brother(father(y))* can be represented by *uncle(y)* (see [49], p. 548).

Paramodulation (see [49], p. 548) is a generalization of equality substitution at the term

level. For example (see [49], p. 548), for a given expression *older(mother(x), x))* and the equality relationship *equal(mother(sadaf), papia)* we can conclude *older(papia, sadaf)*. Note that this is a term level matching and replacement of $\{sadaf/x\}$ and *mother(sadaf)* for *papia*. There are few restrictions on demodulation. First, the equality clause of demodulation must be a unit clause. Therefore, we cannot do demodulation if the clause was $\{f(f(x)) = g(x), r(x)\}$. Second, when we unify the clauses, we can only perform the substitution on the unit equality clause. Paramodulation relaxes these restrictions. Paramodulation allows a nontrivial replacement of variables in both the arguments of the equality predicate and the predicate into which the substitution is made. Note that, by using paramodulation once, one can get an expression into its final form whereas to achieve the same final form multiple demodulators may be needed (see [49], p. 548). Obviously, demodulation is not complete because if we have any non-unit clauses with equality, we would not be able to prove some facts that are in fact entailed.

3.4.2.2 Saturation

Saturation procedure, also known as *given-clause algorithm*, was introduced by Overbeck [58] in 1974. In this procedure, a participating clause is selected to enable the inferences between this clause and the other selected clauses so far. In a saturation state, all the set of clauses are maintained. These clauses are divided into two sets namely, *active* and *passive* sets. Clauses from the active set are available for deduction inferences. Clauses in the passive set are only available for simplifications. Passive clauses are waiting their turn to become active. At each step of the saturation process, a deduction inference is selected, which can be made from some clauses in the active sets and a participating clause

from the passive set. As a result, a new set is generated. Saturation [62] can be viewed as a way of searching in the space of all derivations from the initial clause set. There are two ways we can reduce the search space. In the first one, by deriving an empty clause using simplification rule, we can reduce the size of the search space to be explored. By abandoning some search directions in this way we can conclude that a solution lies in the remaining search directions. Redundant search directions can be removed by identifying some redundant clauses from the active set and excluding them from consideration. This clause redundancy [62] can be checked by using *subsumption* (see Section 2.2.1.4). Another way of reducing search space depends on identifying *redundant inferences* [62]. This is done by imposing some restrictions on how resolution and paramodulation are applied.

3.4.2.3 Adaptive strategies

Several strategies are used to enhance the performance of first-order theorem provers. Among them **Limited Resource Strategy** (LRS) of Riazanov and Voronkov [62, 64] greatly improves the effectiveness of saturation algorithm. This strategy addresses the problem of reasoning in limited time. According to the authors, LRS is an adaptive strategy as it can dynamically adjusts the limit on some weight of clauses. This dynamic adjustment is based on the collected statistics on the earlier stages of proof searching.

The cost of backtracking is usually very high as it may contain several hundreds or thousands of clauses, literals. In order to prevent this problem, Riazanov and Voronkov suggested another strategy **Splitting Without Backtracking** (SWB) in [63]. The SWB is an intelligent backtracking that contains the splitting history. Therefore, by analysing the history, it is easy to identify the path of splitting to reach a contradiction. For instance,

consider, a set S of clauses where C_1 and C_2 are two new clauses. Now to refute the set $S \cup \{C_1 \vee C_2\}$ we can split it as $S \cup \{C_1\}$ and $S \cup \{C_2\}$ and check refutation.

Riazanov and Voronkov also shown in [65], the use of demodulation in two different modes, *forward* demodulation and *backward* demodulation. In forward demodulation, according to the unit equalities that already exists in the current clause set, newly derived clauses are re-written. While in the backward demodulation, old set of clauses are rewritten based on the positive unit equalities of the newly derived clauses. One need a technique to retrieve instances in order to re-write the subset of clauses with unit equalities. The authors [65] have introduced **Path Indexing** technique for smooth instance retrieval. Consider every instance t of a term q contains all paths from q. From the index of the set of terms, the candidate clause set which contains all the terms, i.e., contains maximal path can be extracted.

Chapter 4

First-Order Probabilistic Logic Problem

4.1 Statement of the FOPL Problem

The first-order probabilistic logic or FOPL problem has been formally defined in Section 2.2.1. The *decision* form of FOPL (P^{FOSAT} for short), is defined as follows: given a set $\mathcal{F}$ of m formulas in prenex normal form, and a probability vector $\pi = (\pi_1, \pi_2, \ldots, \pi_m)$ associated with these formulas, is the set $(\mathcal{F}, \pi)$ consistent? The *optimization* form of FOPL (P^{FOENTAIL} for short) also formally defined in Section 2.2.2, can be stated as follows: given a set $\mathcal{F}$ of m formulas in prenex normal form, and a probability vector π associated with these formulas, such that the set $(\mathcal{F}, \pi)$ is consistent, considering an additional formula F_{m+1}, what are the probability values for F_{m+1} to be true so that the system $(\mathcal{F} \cup F_{m+1}, \pi)$ remains consistent.

4.2 Mathematical Modeling

Let $(\mathcal{F}, \pi)$ be a system of formulas in prenex normal form with probabilities. Recall from Section 2.2.1 that, a *world* is defined as any truth assignment w over $\mathcal{F}$, as in Nilsson [56]. A *world* is *possible* if there exists a truth assignment over the set of predicates and a value assignments over X which leads to w over $\mathcal{F}$, and the world is impossible otherwise. Indeed, a possible world can be equivalently defined as the set of truth assignments over the set of formulas which lead to the same truth assignment w over $\mathcal{F}$, see Section 2.2.1 where we discuss the definition of a possible world. Let W be the set of possible worlds and let $p = (p_1, p_2, \dots)$ be a probability distribution on W. The probability distribution satisfies the set of logical formulas together with their probabilities if: for each formula F_i $(i = 1, 2, \dots, m)$, the sum of all p_j's over all truth assignments w_j which satisfy F_i equals π_i.

Let A be an $m \times |W|$ matrix such that a_{ij} is equal to 1 if w_j satisfies F_i, and equal to 0 otherwise. The P^{FOSAT} problem is defined as follows. Is there a probability distribution p such that

$$(P^{\text{FOSAT}}) \quad \begin{cases} \mathbb{1} \cdot p = 1 \\ Ap = \pi \\ p \geq 0. \end{cases} \tag{30}$$

has at least one solution?

Assume now that the answer to the P^{FOSAT} problem defined by $(\mathcal{F}, \pi)$ is positive. Let F_{m+1} denote an additional logical formula, possibly deduced from $\mathcal{F}$. The P^{FOENTAIL} problem as defined by Nilsson [56] consists in determining the range $[\underline{\pi}, \overline{\pi}]$ of values of the probability

π_{m+1} associated with F_{m+1} such that $(\mathcal{F} \cup \{F_{m+1}\}, (\pi.\pi_{m+1}))$ remains consistent. This last problem can be solved by considering an objective function $A_{m+1}P$ where $A_{m+1} = (a_{m+1,j}), j = 1, 2, \ldots$ and $a_{m+1,j}$ is equal to 1 if F_{m+1} is true for the possible world w_j and equal to 0 otherwise, and determining

$$(P^{\text{FOENTAIL}}_{\text{min}}) \qquad \underline{\pi}_{m+1} = \max\{A_{m+1}p : \mathbb{1} \cdot p = 1, Ap = \pi, p \geq 0\} \tag{31}$$

$$(P^{\text{FOENTAIL}}_{\text{max}}) \qquad \overline{\pi}_{m+1} = \min\{A_{m+1}p : \mathbb{1} \cdot p = 1, Ap = \pi, p \geq 0\}. \tag{32}$$

Similar to PPL, the two forms of FOPL problem, namely the *decision* and the *optimization* forms correspond to the *consistency* and *entailment* problems respectively.

Again, it is easy to extend the models with single point probabilities to models with probability intervals and also to introduce some conditional probabilities for situations where knowledge is known only when some conditions hold. As shown in [39] for the PPL mathematical models, the above models for FOPL can readily be extended to include both the probability intervals and the conditional probabilities. For instance, for probability intervals, the probabilistic entailment model become:

$$(P^{\text{FOENTAIL}}_{\text{min(max)}}) \qquad \text{subject to:} \quad \begin{cases} \text{min(max)} \quad A_{m+1}p \\ \mathbb{1} \cdot p = 1 \\ \underline{\pi} \leq Ap \leq \overline{\pi} \\ p \geq 0. \end{cases} \tag{33}$$

Recall from [32, 39], the probabilistic entailment model for probability intervals (33) can be reduced to a single inequality in which the number of constraints remains equal to $m + 1$

and the model become :

$$(P^{\text{FOENTAIL}}_{\min(\max)}) \qquad \text{subject to:} \quad \begin{cases} \min(\max) \quad A_{m+1}p \\ \mathbb{1} \cdot p = 1 \\ \underline{\pi} \leq Ap + s \leq \overline{\pi} \\ p \geq 0 \\ o \leq s \leq \overline{\pi} - \underline{\pi}. \end{cases} \tag{34}$$

where, s denotes *slack* variables of a revised simplex method. A slack variable is usually added to a constraint of a simplex method to turn the inequality into an equation.

4.3 Outline of the Solution

A straightforward method for solving the FOPL problem consists in searching firstly for the overall set of possible worlds, and secondly, solving the resulting linear program. However, the search for possible worlds is very tedious in first-order logic. Indeed, one needs to use, e.g., the resolution principle to check whether a world is possible or impossible. The reader can also refer to the book of Chang and Lee [7] for a thorough review of the various resolution methods for first-order logic. Although simple, this method is not scalable as checking whether a world is possible is NP-complete. Moreover, the number of possible worlds very quickly becomes very large as the number of predicates and number of formulas increases. As for the solution of the probabilistic logic problem, it would be better to use column generation techniques, i.e., to consider explicitly only a subset of possible worlds while considering implicitly the overall set of possible worlds when solving the FOPL problem.

The linear program being solved is decomposed into two problems: the *master* problem and the *pricing* problem. At each iteration of the column generation method, we only consider a *restricted master problem*, i.e., the master problem with only a subset of possible worlds, i.e., a small number of variables explicitly. The pricing problem is a new one created to generate a possible world, i.e., a column or a new variable which will be added in the restricted master problem at each iteration of the solution process. The objective function of the pricing problem is the reduced cost of the new variable (or possible world) and the constraints correspond to the definition of a possible world.

Assuming we are minimizing, the column generation process works as follows. The restricted master problem is solved. From its optimal solution, we are able to obtain dual values for each of the constraints in the restricted master problem. This information is then utilized in the objective function of the pricing problem. The pricing problem is solved. If the objective value of the pricing problem is negative (assuming we are minimizing), a new possible world with negative reduced cost has been identified. This world is then added to the restricted master problem, and the augmented restricted master problem is re-solved. Re-solving the restricted master problem will again generate a new set of dual values, and the process is repeated until either there exists no possible world or there exists a possible world with a negative reduced cost is found. In such a case we can conclude that the optimal solution of the restricted master problem is the optimal solution of the master problem.

4.4 Solution of the Pricing Problem

We discuss how to solve the pricing problem associated with the FOPL column generation model. Without loss of generality for the ($P_{\min}^{\text{FOENTAIL}}$) problem, it can be written as follows:

$$(P_{\text{PRICING}}^{\min}) \qquad \min_{w \in W} \overline{c}(w) \tag{35}$$

$$\text{where} \qquad \overline{c}(w) = c(w) - u_0 - w \cdot u \tag{36}$$

where $\overline{c}(w)$ denotes the reduced cost of w, $c(w)$ is the entailed cost of w associated with the truth value assignment of the additional formula F_{m+1}, i.e., $c(w) = a_{m+1}$ of (33), $u_0 \in R$ is the dual variable associated with the first constraint of ($P_{\min}^{\text{FOENTAIL}}$) of (33) and $u \in R^m$ is the optimal dual vector associated with the second set of constraints of ($P_{\min}^{\text{FOENTAIL}}$) of (33) and W is the set of possible worlds. Note that, as each world component takes only 0-1 values, we have

$$0 \leq c(w) \leq 1. \tag{37}$$

We propose to use a branch-and-bound algorithm to solve the pricing problem because it is NP-complete. We therefore need bounds on $\overline{c}(w)$. Let $\overline{c}_{\text{LO}}$ and $\overline{c}_{\text{UP}}$ denote the lower and upper bounds on $\overline{c}(w)$ respectively:

$$\overline{c}_{\text{LO}} \leq \overline{c}(w) \leq \overline{c}_{\text{UP}}. \tag{38}$$

First straightforward bounds are:

$$\overline{c}_{\text{LO}} = -u_0 - \sum_{i=1}^{m} w_i \times \max\{0, u_i\}, \tag{39}$$

$$\overline{c}_{\text{UP}} = 1 - u_0 - \sum_{i=1}^{m} w_i \times \min\{0, u_i\}. \tag{40}$$

As we are only interested in possible worlds with a negative reduced cost, we therefore search for a possible world w such that $\overline{c}(w) < 0$.

In order to solve (P_{PRICING}) problem, we propose to use the following branch-and-bound algorithm named SOLPRICING. The algorithm is designed with the aim to find a possible world with the most negative reduced cost if one exists. However, it includes the option to stop as soon as a possible world w with a negative reduced cost has been found, even if it is not the most negative one. Note that, in order for the column generation algorithm to iterate it is enough to have at hand a column with a negative reduced cost, even if it is not the most negative one. At each iteration of the SOLPRICING algorithm, we calculate the $\overline{c}_{\text{LO}}$ and $\overline{c}_{\text{UP}}$ bounds on the reduced cost, i.e., at each node of the search tree. Then, we select and branch on the node with the smallest $\overline{c}_{\text{UP}}$, i.e., we use the classical node selection in a branch-and-bound in order to reduce as quickly as possible the uncertainty interval (i.e., the width of the interval where the optimal value lies). Branching is done by selecting a formula F_i and defining two branches, one associated with the value assignment F_i = true, the other one with F_i = false. Moreover, we also want the world to be possible. Therefore, at each node of the search tree we check whether the partial world, made of a subset of formulas with an assigned true/false value, is possible. The outline of the algorithm SOLFOPL for

solving the FOPL problem is given in Table 2, it is following be a detailed description of the SOLPRICING procedure.

Table 2: Algorithm SOLFOPL.

Step 1:	Solve the restricted master problem;
Step 2:	Solve the pricing problem;
	If a column with a negative reduced cost is found then go to step 1;
	Else STOP.

Without any loss of generality, we describe the SOLPRICING procedure for the solution of the ($P_{\min}^{\text{FOENTAIL}}$) problem below. Here, F_{m+1} denotes the additional formula for which the objective function is defined. Moreover, when assignments of truth value to all the formulas is not yet accomplished, we denote a world as a *partial world*, otherwise, it is denoted as a *world* which might be a possible world. However, we will keep this possible world based on the sign of the reduced cost .

Procedure SOLPRICING

Initialization

$\mathcal{L} \leftarrow \{\mathcal{L}^0\}$ where $\mathcal{L}$ is the set of m formulas at current node which are not assigned with truth values yet. Initially, $\mathcal{L}^0 \leftarrow \{F\}$. But the size of $\mathcal{L}$ decreases, as at each iteration, one formula is assigned a truth value. We have as many $\mathcal{L}^k$ as the number of nodes and the number of nodes increases as we develop the search tree.

$\mathcal{D} \leftarrow \{u_0, u\}$ where $\mathcal{D}$ is the ordered set of dual values (decreasing order) u_0 is the dual variable associated with the first set of constraint of (36) and u is the m-vector of dual variables associated with the second set of constraints of (36).

$\mathcal{N} \leftarrow \{N^0\}$ where $\mathcal{N}$ is the set of nodes of the search tree. It is initialized with N^0, i.e., the root of the search tree.

$\widehat{w} \leftarrow \{\widehat{w}_0\}$ where $\widehat{w}$ denotes, at a current iteration, the *partial* world made of the set of formulas with an assigned **true (false)** value. At the beginning, none of the formulas has been assigned a truth value, (hence $\widehat{w}_0 = \emptyset$). However, as we iterate with the separation and branching steps, we add to $\widehat{w}$, formulas which are assigned truth values. We have as many $\widehat{w}_k$ as the number of nodes and the number of nodes increases as we develop the search tree. $\widehat{w}$ corresponds to a possible (but not possibly only partial) world only if all the formulas with the associated truth values belongs to $\widehat{w}$ (i.e., have been assigned a truth value) and the associated reduced cost is negative.

Initial bounds on reduced cost

$$\overline{c}^0_{\text{LO}} \leftarrow -u_0 - \sum_{i=1}^{m} w_i \times \max\{0, u_i\}; \tag{41}$$

$$\overline{c}^0_{\text{UP}} \leftarrow 1 - u_0 - \sum_{i=1}^{m} w_i \times \min\{0, u_i\}; \tag{42}$$

$\overline{c}^\star \leftarrow +\infty$ where $\overline{c}^\star$ is the best known value for the reduced cost associated with a possible world (not partial world).

$N^{current} = N^0$ where $N^{current}$ is the current node.

$\widehat{w} \leftarrow \widehat{w}_0$, $\widehat{c}_{\text{LO}} \leftarrow \overline{c}^0_{\text{LO}}$, $\widehat{c}_{\text{UP}} \leftarrow \overline{c}^0_{\text{UP}}$.

FSB or Formula Selection and Branching (at node N^k)

Select a formula F^k_i from $\mathcal{L}$ with the largest dual value. In case of ties select the

formula with smallest index. Delete F_i^k from $\mathcal{L}$.

Define and add to $\mathcal{N}$, the following two nodes or subproblems:

$\mathcal{N} \leftarrow \mathcal{N} \cup \{N^{k'}\}$ with $N^{k'}$ derived from N^k with F_i^k = True

and $\mathcal{N} \leftarrow \mathcal{N} \cup \{N^{k''}\}$ with $N^{k''}$ derived from N^k with F_i^k = False

Define the following two partial worlds:

$\widehat{w}_{k'} \leftarrow \widehat{w}_k \cup \{\mathrm{F}_i^k = True\}$

and $\widehat{w}_{k''} \leftarrow \widehat{w}_k \cup \{\mathrm{F}_i^k = False\}$

Calculate $\overline{c}_{\mathrm{LO}}^{k'}$, $\overline{c}_{\mathrm{UP}}^{k'}$, $\overline{c}_{\mathrm{LO}}^{k''}$, $\overline{c}_{\mathrm{UP}}^{k''}$.

PSR or Problem Selection and Relaxation

Select and delete a node $N^k \in \mathcal{N}$ based on the smallest $\overline{c}_{\mathrm{UP}}^k$ on the reduced cost associated with all the nodes. Break the ties with the smallest $\overline{c}_{\mathrm{LO}}^k$ and if there are still some ties, select the node with the largest index.

Set $N^{current} = N^k$, set $\widehat{w} \leftarrow \widehat{w}_k$ and $\mathcal{L} \leftarrow \mathcal{L}^k$.

If ($\mathcal{L} = \emptyset$), i.e., If all formulas of $\widehat{w}$ have been assigned a truth value, go to *Stopping Condition.*

PC or Pruning and Contradiction (at node N^k)

If ($\overline{c}_{\mathrm{LO}}^k \geq \min\{\overline{c}^\star, 0\}$), prune the branch and go to Step *PSR.*

If (contradiction) is found, prune the branch and go to Step *PSR.*

Otherwise, go to Step *FSB.*

Stopping Condition

For each truth value (v) of F_{m+1}, do the following:

Compute reduced cost $\widehat{c}$ and check if a contradiction can be found (i.e., whether $\widehat{w}$ is a possible world).

If $(\widehat{c} \geq \min\{\overline{c}^{\star}, 0\})$ or contradiction is found, discard the node.

Otherwise, if $(\widehat{c} < \min\{\overline{c}^{\star}, 0\})$, update $\overline{c}^{\star} \leftarrow \min\{\overline{c}^{\star}, \widehat{c}\}$ and $\widehat{w}^{\star} \leftarrow \widehat{w} \cup \{\mathrm{F}_{m+1} = v\}$.

End For

If $(\mathcal{N} \neq \emptyset)$, go to step *PSR*.

If $(\overline{c}^{\star} < 0)$, $\widehat{w}^{\star} \leftarrow \widehat{w}$ is the possible world with the most negative reduced cost. STOP.

Otherwise, there is no possible world with a negative reduced cost. STOP.

The SOLPRICING algorithm is given assuming we are looking for the possible world with the most negative reduced cost. However, we have the option to stop as soon as we obtain a possible world with the first negative reduced cost (which may not be the most negative one). In order to stop with the first negative reduced cost, in *stopping* condition, instead of checking $(\widehat{c} \geq \min\{\overline{c}^{\star}, 0\})$, we compare the reduced cost with 0, i.e., $(\widehat{c} \geq 0)$ and check if there exists a contradiction. The algorithm stops as soon as we find a possible world with a negative reduced cost.

The SOLPRICING algorithm is similar for $(P_{\max}^{\text{FOENTAIL}})$ problem except in the steps *FSB*, *PSR* and *PC*. In step *FSB*, we select a formula F_i^k from $\mathcal{L}$ with the smallest dual value. Break the ties using smallest formula index. In step *PSR*, we select a node N^k based on the largest $\overline{c}_{\text{UP}}^k$ on the reduced cost associated with all nodes. Break ties using first the largest $\overline{c}_{\text{LO}}^k$, and then the largest node index. In step *PC*, if $(\overline{c}_{\text{UP}}^k \leq \max\{\overline{c}^{\star}, 0\})$ and/or contradiction

is found we prune the branch.

4.5 Another Solution for the Pricing Problem

In this section, we propose another algorithm named, SOLPRICING$^+$ for solving the pricing problem efficiently. This is an attempt to get more quickly an accurate estimation of the reduced cost.

As before, the ultimate goal is to reduce the search space of the tree in order to get a possible world with negative reduced cost as quickly as possible. If we can prune branches at an early stage, the tree size will be smaller. In SOLPRICING, when a partial world is in consideration, we prune by computing the bounds (with associated sign) on the reduced cost of a node or checking the contradiction. Therefore, it is sometimes possible to prune nodes early and reduce the tree size. Most often, we do not have the option to compute the exact value of the reduced cost before we reach the leaf nodes of the tree. However, it is known that very often when column generation is used, the number of zero dual values increases as we iterate. In other words, if several dual variables are equal to zero, it offers the opportunity to compute exactly the value of the reduced cost before we reach a leaf node. This is the opportunity that we want to explore in the SOLPRICING$^+$ algorithm.

In the SOLPRICING$^+$ algorithm, the branching process is similar to the SOLPRICING except that we first perform separation and branching on formulas such that the associated zero dual variable values is non zero. Before zero dual values come across, we can compute the reduced cost (36) before further branching, even if the current world is a partial one. We observed from the equation (36) that, if a formula is associated with a null dual variable, its

value has no impact on the value of the reduced cost. Therefore, by computing the reduced cost as soon as we encounter formulas with zero dual values, we may be able to prune some branch at an earlier stage due to positive reduced cost if there is any.

We propose below another exact algorithm named, SOLPRICING$^+$ for solving the pricing problem.

Procedure SOLPRICING$^+$

Initialization

Same as SOLPRICING except the ordering of the dual values. Dual values are organized as positive first, then negative and zeros at the end.

FSB or Formula Selection and Branching (at node N^k)

If $(u_i \neq 0)$,

Select a formula F_i^k from $\mathcal{L}$ with the largest non zero dual value. In case of ties select the formula with smallest index. Delete F_i^k from $\mathcal{L}$.

Else select and delete a formula F_i^k from $\mathcal{L}$ in lexicographic order.

Define and add to $\mathcal{N}$, the following two nodes or subproblems:

$\mathcal{N} \leftarrow \mathcal{N} \cup \{N^{k'}\}$ with $N^{k'}$ derived from N^k with $\mathrm{F}_i^k = \text{True}$

and $\mathcal{N} \leftarrow \mathcal{N} \cup \{N^{k''}\}$ with $N^{k''}$ derived from N^k with $\mathrm{F}_i^k = \text{False}$

Define the following two partial worlds:

$\widehat{w}_{k'} \leftarrow \widehat{w}_k \cup \{\mathrm{F}_i^k = True\}$

and $\widehat{w}_{k''} \leftarrow \widehat{w}_k \cup \{\mathrm{F}_i^k = False\}$

If $(u_i = 0)$, compute reduced cost and set $\overline{c}_{\mathrm{LO}}^{k'} = \overline{c}_{\mathrm{UP}}^{k'} = \overline{c}_{\mathrm{LO}}^{k''} = \overline{c}_{\mathrm{UP}}^{k''} = \overline{c}^k$, go to *PSR*.

Else calculate $\overline{c}^{k'}_{\text{LO}}$, $\overline{c}^{k'}_{\text{UP}}$, $\overline{c}^{k''}_{\text{LO}}$, $\overline{c}^{k''}_{\text{UP}}$.

PSR or Problem Selection and Relaxation

Same as SOLPRICING.

PC or Pruning and Contradiction (at node N^k)

If $\exists$ a formula F_i such that $u_i \neq 0$

If $(\overline{c}^k_{\text{LO}} \geq \min\{\overline{c}^\star, 0\})$, prune the branch and go to Step *PSR*.

If (contradiction) is found, prune the branch and go to Step *PSR*.

Else, go to Step *FSB*

Stopping Condition

Same as SOLPRICING

SOLPRICING$^+$ algorithm is also given assuming we are looking for the possible world with the most negative reduced cost. Therefore, similar to the SOLPRICING algorithm, we stop as soon as we obtain a possible world with the first negative reduced cost (which may not be the most negative one).

Let us describe how we select a formula at each iteration. We ordered the formulas in $\mathcal{L}$ according to the associated dual values. The dual values are ordered as positive first, then negative and zeros at the end. Let us illustrate with an example how a formula is selected at each iteration. Consider, we have a set of formulas $(F_1, F_2, F_3, F_4, F_5, F_6)$ with the associated dual values (10, 15 ,-30, 0, -1, 0). After ordering the dual values in $\mathcal{D}$ we have (15, 10, -1, -30, 0, 0). Therefore, in $(\mathcal{L})$ we have the formulas ordered as $(F_2, F_1, F_5, F_3, F_4, F_6)$.

Note that, In case of same dual values, formula with the smallest index is considered first. Now when formula with zero dual value comes in consideration, formula selection is done in lexicographic order.

4.6 Satisfiability Checking

In order to check the satisfiability of a set of formulas in the SOLPRICING algorithm we use the package [54] Theo-2006. It is an open source code, implemented in C. It runs under FREEBSD and LINUX. It is a resolution refutation theorem prover for first-order logic where a theorem consists of a set of axioms and a conjecture (goal clause). In first-order logic, the theorem might be presented as a satisfiable set of first-order formulas in clausal form and an additional formula also in clausal form. According to the resolution refutation principle (see Section 2.2.1.3), the goal (conjecture) should be negated before being submitted to the theorem prover. If a contradiction is found using the set of inference rules and reduction-simplification techniques, one can conclude of the satisfiability of the original set of formulas.

Theo works more or less in a similar fashion than what we mentioned earlier in Section 3.4.2. Formulas are fed into Theo in conjunctive normal form (CNF) and inside Theo they are first converted into clausal form and the conjecture is negated. Basically the inference operations [54] in Theo are binary resolution, binary factoring, substitution by constant or variable, instantiation of one or two variables in a clause (see Section 3.4.2). Theo also performs subsumption (see Section 2.2.1.4) and redundant term elimination for further simplification.

Originally it was called The *Great Theorem Prover*. Theo uses a large hash table (16 million entries) to store clauses which are derived from the initial ones using different mechanisms. This permits complex proofs to be found, some as long as 500 inferences. Theo uses what might be called a iteratively deepening depth-first search looking for a contradiction while storing information about clauses in its hash table. Similar to other first-order theorem provers, e.g., SPASS, Theo-2006 also works only on decidable first-order instances [23], i.e., Theo-2006 can indirectly prove the satisfiability of a set of decidable first-order instances in finite time, whereas it will run for an indefinite time for undecidable first-order instances.

Theo was not the best theorem prover among those we have studied but we choose this one for its fairly simple structure, ease of use, and above all it serves our purpose for medium size formulas. Theo has a limitation of dealing with a maximum of 16 predicates in a single formula. The operations performed by Theo use three parameters, maximum number of predicates in a single formula, maximum number of different variables in a single formula, maximum number of different predicates in a single formula. One of the largest instance with 100 formulas, 6 predicates and 174 variables solved by Theo in the order of 376 seconds [54]. The main aim of Theo-2006 is to search for contradiction in finite time. It can not directly prove the satisfiability. In order to conclude on the satisfiability of a set of given decidable formulas with a conjecture we proceed as follows:

- We check the contradiction of a given set of decidable formulas with negating the conjecture. If any contradiction found on a finite time limit we decide on the given set of formulas with a conjecture to be satisfiable. If no contradiction is found on a finite time limit then we are forced to STOP the SOLPRICING as we cannot conclude

due to excessive CPU time.

4.7 An Illustrative Example

Let us consider the following example with 5 formulas defined as follows. The set made of the first four formulas along with their probability values is consistent and we are interested to find the probability values that can be assigned to F_5 such that the augmented set of formulas remains consistent.

$$\begin{cases} F_1 \equiv \forall x\, \exists y & [P(x,y) \vee Q(y)] & 0.9 \\ F_2 \equiv \exists x & [\neg R(x)] & 0.6 \\ F_3 \equiv \exists y & [\neg Q(y)] & 0.6 \\ F_4 \equiv \forall x\, \forall y & [\neg P(x,y) \vee R(x) \vee Q(y)] & 0.7 \\ F_5 \equiv \exists x\, \exists y & [\neg P(x,y)] & [\underline{\pi}, \overline{\pi}] \end{cases} \tag{43}$$

Identifying the set of possible worlds might be a tedious task. Let us, for instance, check whether the world 11110 is possible or not given 1111 is a possible world. Using the resolution principle for first order logic, it is equivalent to check whether the system $\{F_1, F_2, F_3, F_4, F_5\}$ is inconsistent if it is a possible world, and whether the system $\{F_1, F_2, F_3, F_4, \overline{F}_5\}$ is inconsistent if it is an impossible world. Let us show that the world 11110 is impossible. We therefore need to show that we can generate a contradiction out of

$\{F_1, F_2, F_3, F_4, \overline{F}_5\}$. Let us apply the Skolemization (see Section 2.2.1.3), we obtain:

$$\begin{cases} S_1 \equiv P(x, (f(x)) \vee Q(f(x)) & \\ S_2 \equiv \neg R(A) & A \text{ is a skolem constant} \\ S_3 \equiv \neg Q(B) & B \text{ is a skolem constant} \\ S_4 \equiv \neg P(x, y) \vee R(x) \vee Q(y) & \\ S_5 \equiv P(x, y). & \end{cases} \tag{44}$$

Combining S_4 with S_3 leads to $S_6 \equiv \neg P(x, B) \vee R(x)$. Combining S_6 with S_2 leads to $S_7 \equiv \neg P(A, B)$. We then conclude that there is a contradiction out of S_5 and S_7.

We therefore deduce that 11110 is an impossible world.

Reiterating the same process on all potential worlds leads to the results described in Table 3, i.e., the list of possible (P) and impossible (I) worlds. Possible worlds are indexed in order to simplify the subsequent exposure.

4.8 Straightforward LP Solution

Assuming we do search first for the whole set of possible worlds, the FOPL problem leads to solve the following linear program.

$$\min \text{ (max)} \quad \pi_5 = p_1 + p_2 + p_3 + p_5 + p_7 + p_9 + p_{11} + p_{12}$$

subject to:

$$
\begin{aligned}
p_1 + p_2{+}p_3 + p_4{+}p_5 + p_6{+}p_7 + p_8{+}p_9 + p_{10}{+}p_{11} + p_{12} &= 1 \\
+ p_4{+}p_5 + p_6{+}p_7 + p_8{+}p_9 + p_{10}{+}p_{11} + p_{12} &= 0.9 \\
+ p_2{+}p_3 \qquad\qquad + p_8{+}p_9 + p_{10}{+}p_{11} + p_{12} &= 0.6 \\
p_1 + p_2{+}p_3 \qquad + p_6{+}p_7 \qquad + p_{10}{+}p_{11} + p_{12} &= 0.6 \\
p_1 \quad + p_3 + p_4{+}p_5 + p_6{+}p_7 + p_8{+}p_9 \qquad + p_{12} &= 0.7 \\
p_j \geq 0, \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad j &= 1, 2, \ldots, 12.
\end{aligned}
$$

where each column is associated with the truth value assignments of first 4 formulas. In addition, the truth value assignments of the additional formula is the cost of the objective function, together which leads to a possible world, see Table 3. Using the simplex algorithm, one obtains: $\underline{\pi}_5 = 0.1, \overline{\pi}_5 = 1$. Solving the linear program for a sample small test problem is easy, however, as the number of worlds grows exponentially with the number of predicates, it very rapidly become intractable to solve FOPL problems with the simplex algorithm.

4.9 A Scalable LP Solution

Let us now solve the same example using column generation. Let us first consider the ($P_{\min}^{\text{FOENTAIL}}$) problem. Assume worlds to be indexed as follows: $w_j = (w_{1j}, w_{2j}, w_{3j}, w_{4j}, w_{5j})$.

We first introduce 5 artificial worlds $\alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5$ in order to start with a feasible basis without the need to search for an initial set of possible worlds. Note that in this example, worlds associated with the artificial worlds are (10000), (01000), (00100), (00010),

(00001) and are all non possible worlds. The first linear program is therefore:

$$(P^1\text{R-MASTER}) \qquad \min\ 1000 \sum_{i=1}^{5} \alpha_i \tag{45}$$

$$\text{subject to:} \begin{cases} \alpha_1 = 0.9 \\ \alpha_2 = 0.6 \\ \alpha_3 = 0.6 \\ \alpha_4 = 0.7 \\ \alpha_5 = 1 \\ \alpha_i \geq 0 \qquad i = 1, 2, 3, 4, 5. \end{cases} \tag{46}$$

Solving (P^1R-MASTER) leads to $\alpha = (0.9, 0.6, 0.6, 0.7, 1)$ and $u = (1000, 1000, 1000, 1000, 1000)$. Using algorithm SOLPRICING, we identify $w_{12} = (11111)$ as a first world with a negative reduced cost: $\bar{c}_{12} = -4999$. We next solve problem (P^2R-MASTER) that can be stated as follows:

$$(P^2\text{R-MASTER}) \qquad \min\ 1000 \sum_{i=1}^{5} \alpha_i + p_{12} \tag{47}$$

$$\text{subject to:} \begin{cases} \alpha_1 + p_{12} & = 0.9 \\ \alpha_2 + p_{12} & = 0.6 \\ \alpha_3 + p_{12} & = 0.6 \\ \alpha_4 + p_{12} & = 0.7 \\ \alpha_5 + p_{12} & = 1 \\ \alpha_i \geq 0 & \quad i = 1,2,3,4,5 \\ p_{12} \geq 0. & \end{cases} \tag{48}$$

The optimal solution is $\alpha = (0.3, 0, 0, 0.1, 0.4)$, $p_{12} = 0.6$,

$u = (1000, -3900, 1000, 1000, 1000)$.

As α_2 is now a non basis column, we can eliminate it. Using algorithm SOLPRICING, we identify $w_6 = (10110)$ as a new world with a negative reduced cost: $\bar{c}_6 = -4000$. Let us solve problem $(P^3_{\text{R-MASTER}})$.

$$(P^3_{\text{R-MASTER}}) \qquad \min 1000(\alpha_1 + \alpha_3 + \alpha_4 + \alpha_5) + p_{12} + p_6 \tag{49}$$

$$\text{subject to:} \begin{cases} \alpha_1 + p_{12} + p_6 & = 0.9 \\ p_{12} & = 0.6 \\ \alpha_3 + p_{12} + p_6 & = 0.6 \\ \alpha_4 + p_{12} + p_6 & = 0.7 \\ \alpha_5 + p_{12} + p_6 & = 1 \\ \alpha_i \geq 0 & i = 1, 3, 4, 5 \\ p_6, p_{12} \geq 0. \end{cases} \tag{50}$$

The optimal solution is $\alpha = (\alpha_1, \alpha_3, \alpha_4, \alpha_5) = (0.3, 0, 0.1, 0.4)$, $p_{12} = 0.6$, $p_6 = 0$ and α_2 is now a non basis column. In the subsequent iterations, we have:

- Iteration 3. w_8 is added, and α_3 leaves the basis
- Iteration 4. w_{10} is added, and α_4 leaves the basis
- Iteration 5. w_3 is added, and α_1 leaves the basis
- Iteration 6. w_1 is added.
- Iteration 7. w_4 is added.

We then conclude to the optimality of the solution of $(P_{\min}^{\text{FOENTAIL}})$ after the addition of 7 possible worlds (i.e., computation of 7 possible worlds), leading to $\underline{\pi} = 0.1$.

In order to solve the $(P_{\max}^{\text{FOENTAIL}})$ problem, we can either start again with a set of artificial worlds, or from the optimal basis of the $(P_{\min}^{\text{FOENTAIL}})$ problem. Note that, before start solving $(P_{\max}^{\text{FOENTAIL}})$ problem using the optimal basis of the $(P_{\min}^{\text{FOENTAIL}})$ problem we check

the satisfiability of the remaining artificial variables (if there exists any). Using this last initialization, we reach the optimal solution after 2 iterations and the addition of two more possible worlds, leading to $\overline{\pi} = 1$.

To reflect probability intervals instead of fixed point probability the above example can be modified as follows:

$$\begin{cases} F_1 \equiv \forall x\, \exists y & [P(x,y) \vee Q(y)] & [0.85, 0.9] \\ F_2 \equiv \exists x & [\neg R(x)] & [0.55, 0.6] \\ F_3 \equiv \exists y & [\neg Q(y)] & [0.55, 0.6] \\ F_4 \equiv \forall x\, \forall y & [\neg P(x,y) \vee R(x) \vee Q(y)] & [0.65, 0.7] \\ F_5 \equiv \exists x\, \exists y & [\neg P(x,y)] & [\underline{\pi}, \overline{\pi}] \end{cases} \tag{51}$$

Using SOLPRICING algorithm we find the optimal solution of ($P_{\min}^{\text{FOENTAIL}}$) after the addition of 6 possible worlds in 6 iterations, leads to $\underline{\pi} = 0.1$. In the subsequent 6 iterations, we have:

- Iteration 1. w_{12} is added.
- Iteration 2. w_4 is added, and α_2 leaves the basis
- Iteration 3. w_8 is added, and α_4 leaves the basis
- Iteration 4. w_6 is added, and α_1 leaves the basis
- Iteration 5. w_{10} is added, and α_3 leaves the basis
- Iteration 6. w_1 is added, and α_5 leaves the basis

The solution of ($P_{\max}^{\text{FOENTAIL}}$) problem is found after 5 iterations and the addition of four more possible worlds, leading to $\overline{\pi} = 1$.

- Iteration 1. w_{12} is added.
- Iteration 2. w_9 is added.
- Iteration 3. w_7 is added.
- Iteration 4. w_5 is added.
- Iteration 5. w_{11} is added.

Although it is a small example, we can already observe that only a fraction of the possible worlds need to be identify in order to solve the ($P_{\min}^{\text{FOENTAIL}}$) and ($P_{\max}^{\text{FOENTAIL}}$) problems. In practice, only a small fraction (e.g., some hundreds) of the overall set of possible worlds needs to be computed even when the overall number of worlds is over several hundreds of millions.

Table 3: List of possible/impossible worlds

S_1	S_2	S_3	S_4	S_5	
0	0	0	0	0	I
0	0	0	0	1	I
0	0	0	1	0	I
0	0	0	1	1	I
0	0	1	0	0	I
0	0	1	0	1	I
0	0	1	1	0	I
0	0	1	1	1	P : world w_1
0	1	0	0	0	I
0	1	0	0	1	I
0	1	0	1	0	I
0	1	0	1	1	I
0	1	1	0	0	I
0	1	1	0	1	P : world w_2
0	1	1	1	0	I
0	1	1	1	1	P : world w_3
1	0	0	0	0	I
1	0	0	0	1	I
1	0	0	1	0	P : world w_4
1	0	0	1	1	P : world w_5
1	0	1	0	0	I
1	0	1	0	1	I
1	0	1	1	0	P : world w_6
1	0	1	1	1	P : world w_7
1	1	0	0	0	I
1	1	0	0	1	I
1	1	0	1	0	P : world w_8
1	1	0	1	1	P : world w_9
1	1	1	0	0	P : world w_{10}
1	1	1	0	1	P : world w_{11}
1	1	1	1	0	I
1	1	1	1	1	P : world w_{12}

Chapter 5

Numerical Results

In this chapter, we first analyse the two proposed algorithms SOLPRICING and SOLPRICING$^+$. Next we discuss the key data structures that have been used in order to implement them efficiently. We then present the plan and the results of the experiments for the two algorithms on a series of first-order instances. Finally, we discuss the numerical results.

5.1 Algorithm Analysis

Let us analyze the differences between the SOLPRICING and SOLPRICING$^+$ algorithms. Both of them use a branch-and-bound method with a best-first search strategy in order to find a possible world with a negative reduced cost. Both of the SOLPRICING and the SOLPRICING$^+$ are exact algorithms which finds a optimal solution if there exists one. However, SOLPRICING$^+$ algorithm is more efficient than SOLPRICING. The major difference between these two algorithms lies in their branching scheme. So, in this section, we will discuss about their respective branching scheme.

5.1.1 SOLPRICING branching scheme

In the SOLPRICING algorithm, several selection criteria are possible in Step *Formula Selection and Branching* for selecting a new formula with a truth assignment. The choice of selection criteria depends on whether the goal is to prune quickly the branch and therefore deduces a contradiction, or whether the first goal is to exhibit a possible world with a negative cost. It should be based on some hints whether the current pricing problem is likely to be infeasible or to obtain a feasible solution (i.e., a possible world) with negative reduced cost. However, how to get an intuition on which situation is most likely to occur is difficult.

For selecting a new formula, we use the set of dual values. Formulas are serialized in $\mathcal{L}$ according to their associated dual values. The dual values are ordered in decreasing order, assuming we are minimizing the objective function, with an attempt to obtain a negative reduced cost quickly. By keeping the formulas serialized based on the associated dual values saves us from searching for a formula with the largest dual value each time we are selecting a new formula, i.e., we can select a formula serially from $\mathcal{L}$ as it is ordered based on the dual values. In case of maximization, dual values are organised in ascending order with an attempt to obtain a positive reduced cost quickly.

Once a formula is selected, two new nodes or subproblems are added to $\mathcal{N}$ for each truth value assignment of the selected formula. Additionally, two partial worlds are also added to $\widehat{w}$ for each truth value assignment of the selected formula. Then bounds are computed on the reduced cost for both of these nodes. Assuming the goal is to minimize the objective, the next node to be investigated is the node in $\mathcal{N}$ with the smallest upper bound. If there are ties, we go for the node with the smallest lower bound and if there are still some ties, we select the node with the largest node index. The selected node is set as the current node.

Similar to branch-and-bound, in SOLPRICING algorithm, we use two schemes to prune the node in order to keep the tree size smaller. In the first scheme, when a partial world is in consideration, we prune by computing the bounds (with associated sign) on the reduced cost of a node or checking the contradiction. Therefore, it is sometimes possible to prune nodes early and reduce the tree size. In the second scheme, when a world (not partial) is in consideration, i.e., all the formulas are assigned with true or false values, we prune by computing the exact value of the reduced cost (with associated sign) or checking the contradiction at the leaf node. Most often, we do not have the option to compute the exact value of the reduced cost before we reach a leaf node of a tree. However, it is known that, when column generation is used, very often the number of zero dual values increases as we iterate. Therefore, if several dual variables are equal to zero, it offers the opportunity to compute the reduced cost before we reach a leaf node. So, we are interested to investigate this opportunity in the SOLPRICING$^+$ algorithm. Moreover, average traversing at each iteration is usually high in SOLPRICING algorithm which can be solved by using SOLPRICING$^+$.

In order to select the formula with a true/false value assignment, several other possible criteria could have been used. Examples are as follows.

- If there is a formula with a single predicate, select it.
- If there exists more than one formula with a single predicate, select the formula associated with the predicate, say P, that has the largest number of occurrences in the other formulas, with the most balanced number of occurrences in terms of P vs $\neg P$.

Other parameters that would be useful to be taken into account are the fact that, in the selected formula, at least one of the predicate appears both negated and non negated in the previously or forthcoming selected formulas. This helps to ensure that several consensus will or can be generated. While selecting a unit formula is useful for the first selected formula, the priority for the subsequent choices should be more toward the identification of a possible world with a negative reduced cost, and therefore, subsequent selections have to be done only with respect to the values of the dual variables and the "sharing" for the new formulas of many predicates under different forms (negated vs non negated) with the previously selected formula. Similarly, several selection criteria are possible in Step *Problem selection and Relaxation* in order to select the next problem (P^k_{PRICING}) to be examined. We use the classical selection criterion that is used in branch-and-bound methods in Operations Research, i.e., the sub-problem with the smallest upper bound when we minimize the objective and largest upper bound when we maximize the objective.

5.1.2 SOLPRICING$^+$ branching scheme

In order to reduce the average traversing at each iteration and speed up SOLPRICING, we proposed SOLPRICING$^+$ algorithm as a solution for the pricing problem. As shown in Figure 2, we divide the branching tree for SOLPRICING$^+$ into two parts. In the upper part of the tree, we branch in the usual way as long as dual values are non-zero. For this part, we prune and branch by checking both the reduced cost and satisfiability criteria. However, once we encounter zero dual values, we go on pruning and branching based on satisfiability checking only, as reflected in the lower part of the tree. Note that, in the lower part of the tree, formula selection for branching is done in lexicographic order. It would have

been possible to use other strategies, for instance, branching on weighted formulas (see the previous section). The lexicographic order may not be the best selection strategy but it is a simple one. As a result, traversing is reduced on formulas which have their corresponding dual values as zero. So, by using SOLPRICING$^+$ we are able to find the exact solution with improved efficiency and less traversing.

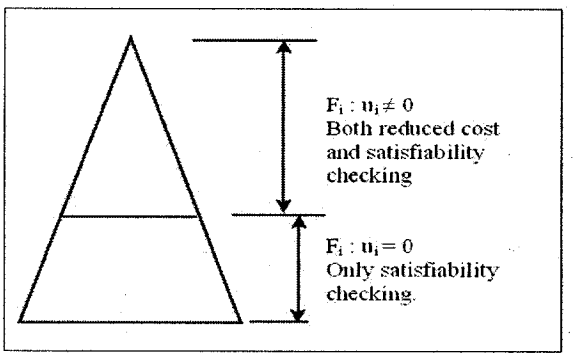

Figure 2: An abstract view of the branching strategy used in SOLPRICING$^+$

The steps used by the procedure SOLPRICING$^+$ are almost same as that of SOLPRICING. The formula is selected same way as SOLPRICING based on the ordered dual values. However, the dual values are ordered differently, for instance, they are ordered in decreasing order as most positive, then most negative and zeros at the end. Break the ties by selecting the formula with smallest index. However, in *formula selection and branching*, the SOLPRICING$^+$ algorithm follows the same process as long as formulas with non-zero dual values are encountered. Once a formula with a zero dual value comes in consideration, SOLPRICING$^+$ algorithm use lexicographic order to select a formula. Next, SOLPRICING$^+$ will compute the reduced cost using equation (36) on the node even though the world might be a partial one. Therefore, no need to calculate the bounds on the reduced cost at the node as the

computed reduced cost corresponds to the bounds in the subsequent iterations.

Assuming we are solving $P_{\min}^{\text{FOENTAIL}}$, if the current reduced cost is positive, the branch is pruned, otherwise satisfiability is checked on the partial world. If a contradiction is found, the branch is pruned, else a new formula (with zero dual values) is selected in a lexicographic order. This time, further branching is decided only by checking the satisfiability or contradiction on the partial world. When all the formulas are assigned with true/false values and there remains no node in $\mathcal{N}$, SOLPRICING$^+$ algorithm checks if there is a contradiction. SOLPRICING$^+$ stops as soon as it finds a first possible world with a negative reduced cost. Otherwise, it searches for a world with a negative reduced cost as long as $\mathcal{N}$ is not empty.

5.2 Programming Environment

We have implemented the two proposed algorithms, SOLPRICING and SOLPRICING$^+$ in C++. The supported compilers are gcc 3.4.4 and higher versions. It contains about 5000 lines of code, runs under Linux Red Hat 3.4.4-2. We use ILOG CPLEX 10.1.1 tool to solve the linear programs. We run our test instances in computers with AMD dual processors, cpu speed 2392.132 Mhz, RAM 15.6 GBs.

5.3 Efficient Implementation

For efficient implementation we choose a priority queue data structure, the *binary heap* (see Sahni [66] and Weiss [71]). Let us now explain how we map our data into a heap. The structure of each node is shown in Figure 3. It stores the truth values of a set of formulas, the lower and upper bounds on the reduced cost at each node.

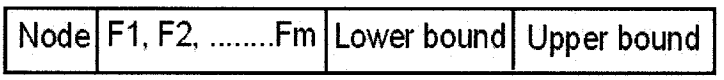

Figure 3: Node structure

In the node structure of Figure 3, the formula fields F_i is set with three different values.

$$F_i = \begin{cases} 0 & \text{if False} \\ 1 & \text{if True} \\ 2 & \text{if Not yet assigned with a truth value} \end{cases} \tag{52}$$

The $\overline{c}^k_{\text{LO}}$ and $\overline{c}^k_{\text{UP}}$ are computed using the set of equations in (39) and (40) respectively. Two issues related to node status that need to be considered, (i) if a node is *pruned* then we delete it, (ii) we store the current node in a special dummy structure.

Considering the objective is to minimize, the data structure is initialized at root node N_0 as illustrated in Table 4. As the value of all the formulas is *not yet assigned with a truth value*, 2 is assigned for all the formulas. The $\overline{c}^k_{\text{LO}}$ and $\overline{c}^k_{\text{UP}}$ are computed as $\overline{c}^0_{\text{LO}}$ and $\overline{c}^0_{\text{UP}}$ respectively.

Table 4: Initial node structure

$\mathcal{N}$	F_1	F_2	F_3	..	F_m	$\overline{c}^k_{\text{LO}}$	$\overline{c}^k_{\text{UP}}$
N_0	2	2	2	..	2	$\overline{c}^0_{\text{LO}}$	$\overline{c}^0_{\text{UP}}$

Now, we have to compute the bounds and check the contradiction at each node based on the truth values assigned to the formulas belonging to their associated partial world $\widehat{w}$. We continue on branching by selecting a formula based on the associated dual value. For each truth value assignment of F_i, two nodes are added to the node list $\mathcal{N}$, N_1 with $F_i =$

true and N_2 with F_i = false. Moreover, two partial worlds $\widehat{w}_1$ with F_i = true and $\widehat{w}_2$ F_i = false are also defined at the current node. Then we compute the upper and lower bounds on the reduced cost of each of these two nodes and the node which has the smallest upper bound on the reduced cost is selected. Say, N_2 has the smallest upper bound on its reduced cost, therefore, we will modify the node status of N_2 as current. Next, contradiction will be checked at node N_2 based on the truth value assignments of the formulas defining partial world $\widehat{w}_2$. On the other hand, if there is no contradiction found at N_2, we continue on branching by selecting next formula based on the associated dual value. For this particular case, we still do not check the contradiction as both of the nodes have only one formula. Therefore, we will continue branching.

Considering the illustrative example in Section 4.7, the corresponding data structure and the tree structure of first three nodes while obtaining the possible world [00111] for minimization is shown in Table 5 and Figure 4 respectively. This process is repeated until all the formulas with their truth value assignments are considered. The associated dual values are given in Table 6.

Table 5: The data structure considering the first three nodes

$\mathcal{N}$	F_1	F_2	F_3	F_4	F_5	$\overline{c}^k_{\text{LO}}$	$\overline{c}^k_{\text{UP}}$
N_0	2	2	2	2	2	-1000	1
N_1	2	1	2	2	2	-1000	1
N_2	2	0	2	2	2	-1000	1

Moreover, considering the same example the heap data structures of all the nodes and their corresponding values to find the possible world [00111] by using the SOLPRICING algorithm is shown in Table 7. Figure 5 shows the associated tree structure for possible world 00111 considering $P^{\text{FOENTAIL}}_{\text{min}}$ and Figure 6 shows the corresponding tree structure for possible world 11011 considering the $P^{\text{FOENTAIL}}_{\text{max}}$. In Table 6, the dual values associated with possible

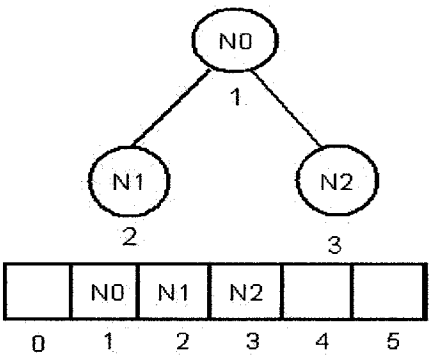

Figure 4: Heap and corresponding array representation for first three nodes

worlds 00111 and 11011 are also provided.

Table 6: Snapshot of the dual values

Possible world	u_1	u_2	u_3	u_4	u_5
00111	−1	0	0	0	1
11011	−1	1	0	1	0

It then leads to the possible expression for the reduced cost: for each truth value assignment on the additional formula, by computing the reduced cost and checking if there is a contradiction we can obtain whether there exists a possible world with a negative reduced cost. Note that in Table 7, the lower and upper bounds on reduced cost of the three nodes N_8, N_{10} and N_6 are set to $\times$ to show that they are pruned.

5.4 Experimental Results

In this section, we evaluate the performance of SOLFOPL with the two algorithms SOLPRICING and SOLPRICING$^+$. For simplicity, we name the combined solution of the restricted master and SOLPRICING as SOLFOPL, and the restricted master and SOLPRICING$^+$ as SOLFOPL$^+$.

Table 7: Data structure associated with possible world [00111].

$\mathcal{N}$	F_1	F_2	F_3	F_4	F_5	$\overline{c}^k_{\text{LO}}$	$\overline{c}^k_{\text{UP}}$
N_0	2	2	2	2	2	-1000	1
N_1	2	1	2	2	2	-1000	1
N_2	2	0	2	2	2	-1000	1
N_3	2	0	1	2	2	-1000	1
N_4	2	0	0	2	2	-1000	1
N_5	2	0	0	1	2	-1000	1
N_6	2	0	0	0	2	$\times$	$\times$
N_7	1	0	0	1	2	-1000	1
N_8	0	0	0	1	2	$\times$	$\times$
N_9	2	0	1	1	2	-1000	1
N_{10}	2	0	1	0	2	$\times$	$\times$
N_{11}	1	0	1	1	2	-1000	1
N_{12}	0	0	1	1	2	-1000	-999

5.4.1 Building the test instances

The test instances are generated randomly similarly to that was proposed in Jaumard *et al.* [39]. The logical formulas correspond to clauses (disjunction of predicates) with at most 3 predicates. Formulas with 1, 2, 3 predicates are distributed uniformly as well as positive or negative predicates, i.e., each predicate have a number of positive occurances equal to the number of negative occurances. We keep at most two variables in each predicate in order to build decidable instances [23]. Moreover, there are at most 3 variables in each formula.

In order to associate consistent probability values for first-order formulas we generate randomly 0 or 1 values which leads to a world. Then we check with Theo if it is possible world or not. If the world is possible we keep it otherwise, we reject it. By following this we generate p possible worlds for m first-order formulas. Next, equal probability distribution (1/p) is assigned over each possible world. Then by summing up the probability distributions on each formula a fixed probability value is generated. By subtracting 0.02 and

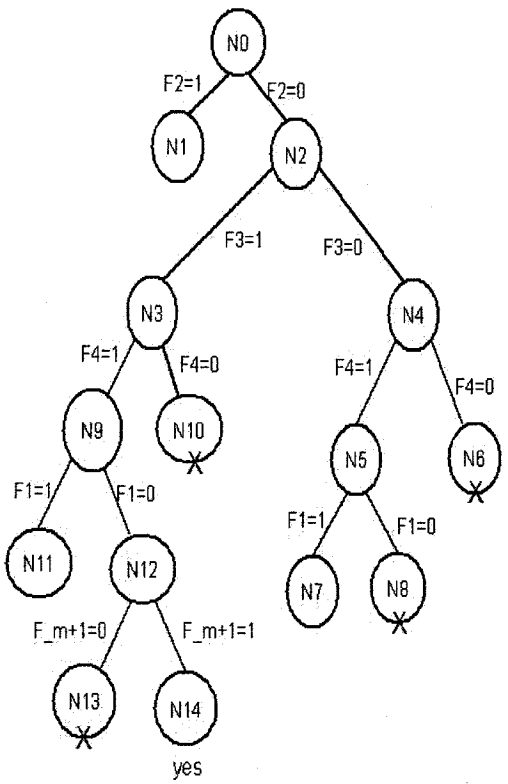

Figure 5: Tree structure: possible world [00111] considering $P_{\min}^{\text{FOENTAIL}}$.

adding 0.08 to the fixed probability value, we finally generate the lower and upper probability values respectively (i.e., probability interval values) for each formula. Then we check the consistency of these first-order formulas with the associated interval probability values by using CPLEX. Instances which are formed inconsistent are discarded. An illustration of generating consistent probability values for a test instance with 7 first-order formulas (for 12 randomly created possible world) is shown in the Appendix.

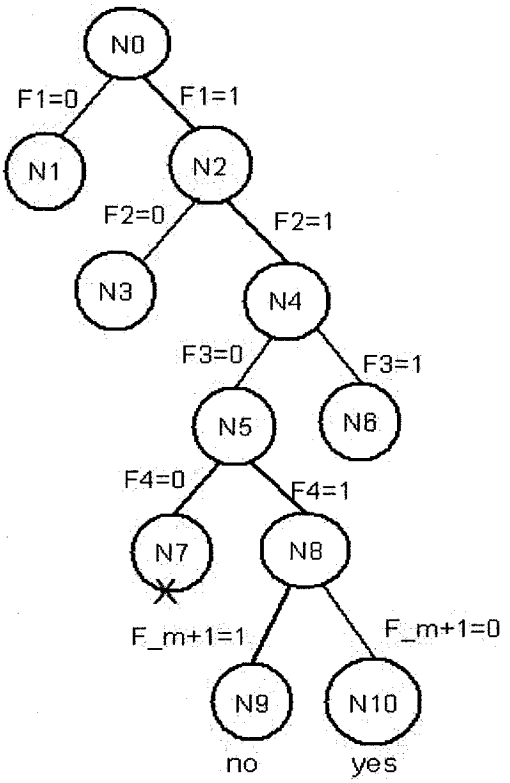

Figure 6: Tree structure: possible world [11011] considering $P_{\max}^{\text{FOENTAIL}}$.

5.4.2 Comparison of the SOLFOPL and SOLFOPL$^+$

The programs were implemented in C++ and tested on a computer with characteristics described earlier (see P. 91). We evaluate the efficiency of the two proposed exact algorithms, SOLFOPL and SOLFOPL$^+$ with a heuristic algorithm AD-SOLFOPL [67] (see Section 3.3.3).

Results of SOLFOPL and SOLFOPL$^+$ are given in Table 8 and in Table 9 respectively on a set of consistent FOPL instances. The structure of instance labeling for the first instance is shown in Figure 7. The first instance consists of 4 formulas (except additional formula) with 3 distinct predicates, each formula has at most 3 predicates; the additional formula is a unit one. The other instances can be described similarly. Each formula contains at most

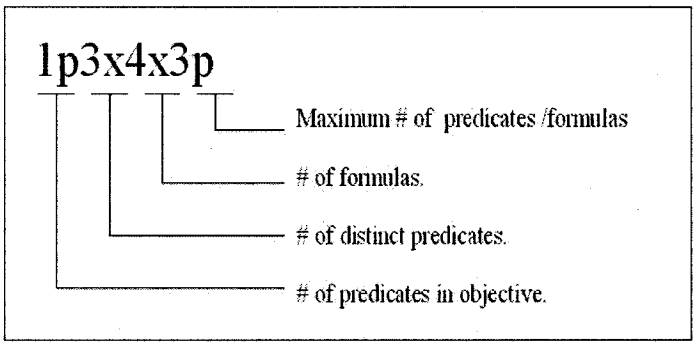

Figure 7: Problem labeling

3 variables.

In Table 8 and 9, we provide the lower and upper ($\underline{\pi}$ and $\overline{\pi}$) probability values, computing times for master and pricing problems, number of columns, average number of nodes in the search tree when solving the pricing problem. Moreover, in Table 10, the number of nodes in the search tree when solving the last pricing problem is also provided. It is observed that, number of columns required for minimization is higher than that of maximization. For minimization, instead of searching for an initial set of possible columns we start from scratch, i.e., with artificial columns in order to start with a feasible basis. At each iteration, an artificial column is deleted from the basis and a column (i.e., possible world) is entered in the basis. Hence, many columns are added to the restricted master problem in order to delete all the artificial columns from the basis. Therefore, we need to generate more columns to obtain an optimal basis for the minimization. On the other hand, we begin with the optimal basis obtained from the optimal solution of the minimization when solving the maximization problem. In spite of providing a good solution, it at least provides a feasible solution and avoids the use of artificial variables. Therefore, a smaller number of

columns is required to obtain the optimal basis for maximization. However, there are few exceptional cases, such as in 3p8x15x3p instance where for minimization the number of columns is smaller than for maximization for both the SOLFOPL and SOLFOPL$^+$ algorithms. The number of columns needed to solve each instance also depends on the complexity of problem structure, i.e., it also depends on the number of predicates, variables, etc used in a particular instance. If an additional formula has a higher number of predicates it is more likely to be difficult to solve.

For all solved instances, SOLFOPL and SOLFOPL$^+$ manage to find an optimal solution where SOLFOPL takes longer than SOLFOPL$^+$. Moreover, in SOLFOPL$^+$, the number of nodes in the search tree is much less than that of SOLFOPL. The reason for this has already been explained in Section 5.1.2. In order to evaluate and exhibit the performance of SOLPRICING and SOLPRICING$^+$, a graph is given in Figure 8. In this graph, we consider finding the lower bound of a particular pricing problem when solving the 2p11x25x3p instance. It is clear from the graph that, to obtain a possible world at each iteration the number of nodes is higher in SOLPRICING than that of SOLPRICING$^+$. In other words, by evaluating this graph, we can conclude about their efficiency and performance. For this solved instance, along the x-axis, number of columns for reaching the optimal solution is considered whereas number of nodes needed to find each of these columns is given along the y-axis. We see from the graph that, in the last iteration SOLFOPL$^+$ traverses less nodes than SOLFOPL as expected. Therefore, within a shorter period of time, we find a possible world with a negative reduced cost in SOLFOPL$^+$.

In Figure 9, another graph is given for the same problem instances in order to show the benefit of computing reduced cost as soon as we encounter null dual variables. This

graph can be evaluated for instance, when the number of columns along the x-axis is 18 the number of null dual variable along the y-axis for SOLPRICING is 16 (Figure 9) which visit or traverse 224 nodes (Figure 8) in order to obtain a possible column. Whereas, in case of SOLPRICING$^+$, the number of null dual variables is 13 and it visited only 26 nodes to obtain a possible world. The number of null dual values, i.e., zero dual values in SOLFOPL$^+$ is 11 and for SOLFOPL it is 15 at the last iteration.

In Table 11, a computational comparison is made between SOLFOPL$^+$ and AD-SOLFOPL on a number of consistent instances. We choose SOLFOPL$^+$ over SOLFOPL because it is more efficient. We provide the probability interval ($\underline{\pi}$ and $\overline{\pi}$) found by each algorithm and the total computing time. We can claim from the comparison that, even though AD-SOLFOPL finds a probability bound in a much shorter computing time than SOLFOPL$^+$, AD-SOLFOPL usually could not obtain the tightest probability bounds. By using SOLFOPL$^+$ we can achieve the tightest probability bound.

Table 8: Results of SOLFOPL using SOLPRICING algorithm.

Instances	Probability		CPU time (sec)		# Column		Average # nodes	
	$\underline{\pi}$	$\overline{\pi}$	Master	Pricing	Min	Max	Min	Max
1p3x4x3p	0.100	1.000	$< 10^{-2}$	0.08	6	5	7.571	4.33
2p4x7x2p	0.125	1.000	$< 10^{-2}$	0.39	7	6	23.75	7.14
3p4x8x3p	0.412	1.000	$< 10^{-2}$	0.66	11	4	23.08	11.80
2p5x10x3p	0.349	1.000	0.02	1.25	9	8	41.40	20.56
3p8x15x3p	0.255	0.967	$< 10^{-2}$	11.23	10	18	172.09	142.58
2p10x20x3p	0.300	1.000	$< 10^{-2}$	33.29	6	10	1193.86	19.55
2p11x25x3p	0.900	1.000	0.02	47.57	29	2	396.43	17.67

One decisive drawback of the algorithms is its computing time. It takes significant amount of time to find an optimal solution when the number of formulas grows large. This is because it is a NP-complete problem and it has to traverse through a large number of

Table 9: Results of SOLFOPL$^+$ using SOLPRICING$^+$ algorithm.

Instances	Probability		CPU time (sec)		# Column		Average # nodes	
	$\underline{\pi}$	$\overline{\pi}$	Master	Pricing	Min	Max	Min	Max
1p3x4x3p	0.100	1.000	0.01	0.08	12	4	4.92	5.60
2p4x7x2p	0.125	1.000	0.01	0.20	12	6	7.85	7.14
3p4x8x3p	0.412	1.000	0.03	0.34	14	4	10.80	19.20
2p5x10x3p	0.349	1.000	0.01	0.35	11	9	10.67	11.00
3p8x15x3p	0.255	0.967	0.02	2.33	12	20	15.00	52.14
2p10x20x3p	0.300	1.000	0.02	1.33	23	9	20.33	19.10
2p11x25x3p	0.900	1.000	0.02	5.27	35	1	69.11	2.00

Table 10: Results of SOLPRICING and SOLPRICING$^+$ algorithm with respect to number of nodes in the last solution of the pricing problem (min).

Instances	SOLPRICING	SOLPRICING$^+$
1p3x4x3p	20	3
2p4x7x2p	50	6
3p4x8x3p	9	3
2p5x10x3p	108	3
3p8x15x3p	38	3
2p10x20x3p	8064	3
2p11x25x3p	8614	1385

nodes while looking for a possible world with negative reduced cost. For instance, for 10 formulas, in the worst case it will traverse $(2^{11} - 1)$ nodes which grows exponentially as the number of formulas becomes large. For bigger instances this becomes painfully slow when waiting for all the iterations to complete. It also occupies a good amount of memory because of the nodes that it has to check before finding a possible world with a negative reduced cost. It is to be noted in our implementation, instead of going for the most negative reduced cost we stopped as soon as we got the first negative reduced cost with a possible one.

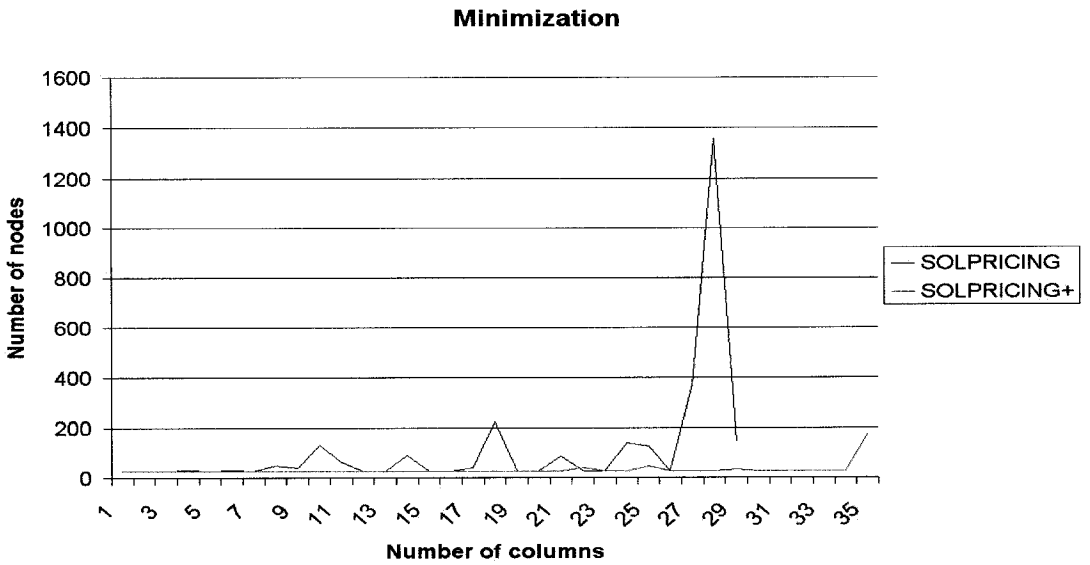

Figure 8: Comparison graph of SOLPRICING and SOLPRICING$^+$ for number of nodes with respect to number of columns for instance 2p11x25x3p.

The implementation part of this thesis was quite challenging as the algorithms did not get into details about a lot of key issues that is essential from programming point of view. It was very intriguing to reduce the exponential time complexity in the search tree. We spent countless hours on debugging. The solution schemes of the overall process was not so easy to understand. The algorithm was not easy to break into pieces for programming purposes.

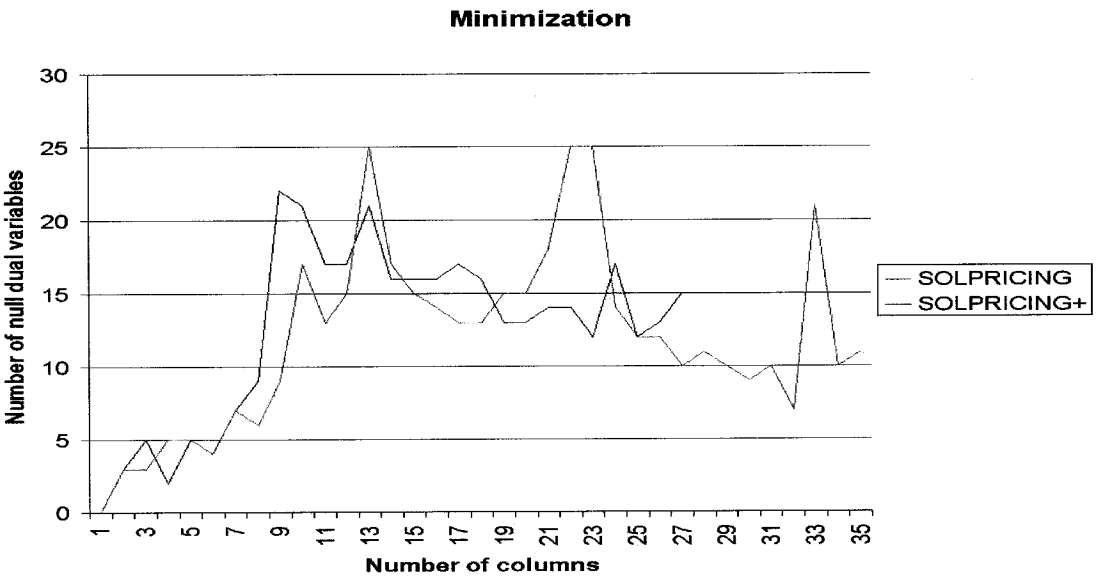

Figure 9: Comparison graph of SOLPRICING and SOLPRICING$^+$ for number of null dual variables with respect to number of columns for instance 2p11x25x3p.

Table 11: Comparison of SOLFOPL$^+$ and AD-SOLFOPL.

Instances	SOLFOPL$^+$			AD-SOLFOPL		
	Probability			Probability		
	$\underline{\pi}$	$\overline{\pi}$	CPU time(sec)	$\underline{\pi}$	$\overline{\pi}$	CPU time(sec)
1p3x4x3p	0.100	1.000	0.09	0.100	1.000	$< 10^{-2}$
2p4x7x2p	0.125	1.000	0.21	0.083	1.000	$< 10^{-2}$
2p5x10x3p	0.349	1.000	0.36	0.134	1.000	$< 10^{-2}$
3p8x15x3p	0.255	0.967	2.35	0.040	1.000	$< 10^{-2}$
2p10x20x3p	0.300	1.000	1.35	0.000	1.000	$< 10^{-2}$
2p11x25x3p	0.900	1.000	5.29	0.330	1.000	$< 10^{-2}$

Chapter 6

Conclusion and Future Work

The focus of this thesis was on reasoning under uncertainty models with first order logic where uncertainty is captured by probability values. We proposed a new mathematical modeling with a column generation formulation, i.e., a linear programming modeling for large scale and highly combinatorial problems. We showed that it allows the solution of small to medium size instances of probabilistic logic instances.

There is room improvement in the column generation algorithm proposed for solving the linear programming model, in particular with the algorithm for solving the pricing problem. By better exploiting on the one hand the order in which the formulas are selected and assigned truth values and on the other hand the observation that several dual variables are equal to zero, we believe that it is possible to increase significantly the size of the instances that can be solved in a reasonable amount of time. Another direction for improvement is to consider a "hot start" rather than a "cold start" for the column generation algorithm using, e.g., heuristic algorithms such as some anytime deduction type algorithms. It will reduce the number of times we need to solve the pricing problem, and as it is the most costly part

with respect to the computing time, it can also contribute significantly to solve much larger instances.

Time should also be spent on defining benchmark decidable instances. Many criticisms were made to probabilistic models toward the fact that when the size of the instances increase, if randomly generated, there are likely to be inconsistent. Is it related to the model or to the generation of the instances? There is not yet a clear answer.

With respect to generalization of the proposed mathematical model, as for probabilistic propositional logic, it will be easy to generalize it in order to handle the case of conditional probabilities. We also believe that the proposed mathematical model could be very useful in order to design new linear programming solution scheme for the classical second-order logic, often called *QBF*. Up to now, most of the QBF satisfiability checkers are based on Davis-Logemann-Loveland (DLL) algorithm. It would be interesting to incorporate the DLL based satisfiability checkers into our implementation.

Very few comparisons have been made among the different models for reasoning under uncertainty as it is quite difficult. There is no well accepted fair basis for comparison when uncertainty is captured using different means, see, e.g., probabilities vs. qualitative values in fuzzy sets. It would be therefore of great interest to investigate how to establish comparisons and compare the various models for reasoning under uncertainty in terms of expressiveness but also in terms of easiness for solution. The best model of expressiveness might be impossible to solve, and therefore compromise has to be made between those two objectives.

Bibliography

[1] K.A. Anderson and J.N. Hooker. Bayesian logic. *Decision Support System*, 11:191–210, 1994.

[2] B. Aspvall, M.F. Plass, and R.E. Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information Processing Letters*, 8(3):121–123, 1979.

[3] F. Bacchus. *Representing and Reasoning with Probabilistic Knowledge.* The MIT Press, Cambridge, Massachusetts, 1990.

[4] G. Boole. Proposed question in the theory of probabilities. *The Cambridge and Dublin Mathematical Journal*, 6(186), 1851.

[5] G. Boole. *The Laws of Thoughts.* Dover Publications, 1854.

[6] B.G. Buchanan and E.H. Shortliffe. *Rule-Based Expert Systems The MYCIN Experiments of the Stanford Heuristic Programming Project.* Addison-Wesley, 1985.

[7] C.L. Chang and R.C. Lee. *Symbolic Logic and Mechanical Theorem Proving.* Academic Press, 1973.

[8] P. Cheeseman. In defense of probability. In *Proceedings of the 9th IJCAI, Los Angeles*, volume 9, pages 1002–1009, 1985.

[9] A. Church. An unsolvable problem of elementary number theory. *American Journal of Mathematics*, 2:345–363, 1936.

[10] V. Chvatal. *Linear Programming.* W.H. Freeman and Company, 1983.

[11] W.L. Craig. Incompatibility, with respect to validity in every finite nonempty domain, of first order functional calculus. In *Proceedings of the International Congress of Mathematicians*, 1950.

[12] Y. Crama, P. Hansen, and B. Jaumard. The basic algorithm for pseudo-boolean programming revisited. *Discrete Applied Mathematics*, 29:171–186, 1990.

[13] P.S. de S. Andrade, J.C.F. da Rocha, D.P. Couto, A. da C. Teves, and F.G. Cozman. A toolset for propositional probabilistic logic. *Encontro Nacional de Inteligncia Artificial, Rio de Janeiro, RJ*, pages 1371–1380, 2007.

[14] F.J. Diez and M.J. Druzdzel. Reasoning under uncertainty. *Encyclopedia of Cognitive Science, Nadel, L.(Ed.) London: Nature Publishing Group*, pages 880–886, 2003.

[15] B. Dreben and W.D. Goldfarb. *The Decision Problem: Solvable Classes of Quantified Formulas.* Addison-Wesley, 1979.

[16] D. Dubois and H. Prade. *Possibility Theory.* Plenum Press, New York, 1988.

[17] De Finetti. *Theory of Probability Volume 2.* Wiley Interscience, first edition, 1990.

[18] De Finetti. *Theory of Probability Volume 1.* Wiley Interscience, first edition, 1991.

[19] A.M. Frisch and P. Haddawy. Anytime deduction for probabilistic logic. *Artificial Intelligence*, 69(1-2):93–122, 1994.

[20] G. Georgakopoulos, D. Kavvadias, and C.H. Papadimitriou. Probabilistic satisfiability. *Journal of Complexity*, 4:1–11, 1988.

[21] F. Glover. Tabu search - part I. *ORSA Journal on Computing*, 1:190–206, 1989.

[22] F. Glover. Tabu search - part II. *ORSA Journal on Computing*, 2:4–32, 1990.

[23] E. Grädel, P.G. Kolaitis, and M.Y. Vardi. On the decision problem for two-variable first-order logic. *The Bulletin of Symbolic Logic*, 3:53–69, 1997.

[24] I. Hacking. *Logic of Statistical Inference.* Cambridge University Press, Cambridge, U.K., 1965.

[25] T. Hailperin. Best possible inequalities for the probability of a logical function of events. *Monthly American Mathematics*, 72:343–359, 1965.

[26] T. Hailperin. Boole's logic and probability: Critical exposition from the standpoint of contemporary algebra, logic and probability theory. *Number 85 in studies in Logic and the Foundations of Mathematics Elsevier*, 3(25):198–212, 1976.

[27] T. Hailperin. *Boole's Logic and Probability.* North-Holland, Amsterdam, 1986.

[28] J.Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, December 1990.

[29] P.L. Hammer, I. Rosenberg, and S. Rudeanu. On the determination of the minima of pseudo-boolean functions (in romanian). *Studii si Cercetari Matematice*, 14:359–364, 1963.

[30] P.L. Hammer and S. Rudeanu. *Boolean Methods in Operations Research and Related Areas.* Springer-Verlag, New York, 1968.

[31] P. Hansen and B. Jaumard. Algorithms for the maximum satisfiability problem. *Computing*, 44:279–303, 1990.

[32] P. Hansen and B. Jaumard. *Probabilistic Satisfiability, Handbook on Algorithms for Uncertain and Defeasible Management Systems*, pages 321–368. J. Kohlas and S. Moral (eds.), Springer, Dordrecht, 2000.

[33] P. Hansen, B. Jaumard, and M.P.de Aragão. Boole's conditions of possible experience and reasoning under uncertainty. *Discrete Applied Mathematics*, 60:181–193, 1995.

[34] P. Hansen, B. Jaumard, M.P.de Aragão, and C.C. de Souza. Correctness of anytime deduction for probabilistic logic. *Les Cahiers du GERAD G-97-02, Groupe d'Etudes et de Recherche en Analyse des Decisions*, 1997.

[35] P. Hansen, B. Jaumard, G.B.D. Nguetsé, and M. P. de Aragão. Models and algorithms for probabilistic and bayesian logic. In *Proceedings of the 14th IJCAI-95*, volume 2, pages 1862–1868, 1995.

[36] P. Hansen and S. Perron. Merging the local and global approaches to probabilistic satisfiability. *to appear in International Journal of Approximate Reasoning.*

[37] U. Hustadt, R. A. Schmidt, and L. Georgieva. A survey of decidable first-order fragments and description logics. *Journal on Relation Methods in Computer Science*, 1:251–276, 2004.

[38] B. Jaumard, A. Fortin, I. Shahriar, and R. Sultana. Logic and algebraic languages for interoperability in multidatabase systems. *Fuzzy Information Processing Society, NAFIPS 2006, Annual meeting of the North American*, pages 341–346, 2006.

[39] B. Jaumard, P. Hansen, and M.P.de Aragão. Column generation methods for probabilistic logic. *ORSA Journal on Computing*, 3(2):135–148, 1991.

[40] B. Jaumard and A. Nongaillard. Automated mechanism design: using column generation for the design of multi-agent exchanges. *Web Intelligence and Agent Systems: An International Journal, To be printed*, 2007.

[41] B. Jaumard and A. Parreira. An anytime deduction algorithm for the probabilistic logic and entailment problems. *Fuzzy Information Processing Society, 2006. NAFIPS 2006, Reprinted: submitted to Elsevier*, Annual meeting of the North American.(3-6):347–354, 2007.

[42] D. Jovanović, N. Mladenović, and Z. Ognjanović. Variable neighborhood search for the probabilistic satisfiability problem. In *The 6th Metaheuristics International conference, MIC 2005*, August 2005.

[43] G.J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic Theory and Applications.* Prentice Hall PTR, first edition, 1995.

[44] K. Konol. An inference net compiler for the prospector rule-based consultation systems. In *Proceedings of the IJCAI*, volume 6, pages 487–489, 1979.

[45] L.V.S. Lakshmanan and F. Sadri. On a theory of probabilistic deductive databases. *Theory and Practice of Logic Programming*, 1(1):5–42, 2001.

[46] L.V.S. Lakshmanan, F. Sadri, and I.N. Subramanian. Logic and algebraic languages for interoperability in multidatabase systems. *The Journal of Logic Programming*, 33(2):101–149, 1997.

[47] S.L. Lauritzen and D.J. Spiegelhalter. Computation with probabilities in graphical structures and their application to expert systems. *Journal of the Royal Statistical Society B*, 50(2):157–224, 1988.

[48] K-S. Leung and M-F. Tse I. King. Ff99: A novel fuzzy first-order logic learning system. *IEEE*, 5:178–184, 1999.

[49] G.F. Luger. *Artificial Intelligence Structures and Strategies for Complex Problem Solving.* Pearson Education, fourth edition, 2002.

[50] T. Lukasiewicz. Probabilistic logic programming. In *13th Biennial European Conference on Artificial Intelligence*, pages 388–392, 1998.

[51] T. Lukasiewicz. Probabilistic logic programming with conditional constraints. *ACM Transactions on Computational Logic*, 2(3):289–339, 2001.

[52] B. Milch and S. Russell. First-order probabilistic languages: Into the unknown. In *Proceedings of the 16th International Conference on Inductive Logic Programming. Berlin: Springer*, 2007.

[53] N. Mladenović and P. Hansen. Variable neighborhood search. *Computational Operations Research*, 24(11):1097–1100, 1997.

[54] M. Newborn. Automated theorem proving: Theory and practice. *Springer*, 2005.

[55] R.T. Ng and V.S. Subrahmanian. Probabilistic logic programming. *Information and Computation*, 101(2):150–201, 1992.

[56] N.J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–87, 1986.

[57] N.J. Nilsson. Probabilistic logic revised. *Artificial Intelligence*, 59(1-2):39–42, 1993.

[58] R. Overbeck. A new class of automated theorem-proving algorithms. *Journal of the ACM*, 21:191–200, 1974.

[59] D. Page and A. Srinivasan. ILP: a short look back and a longer look forward. *Journal of Machine Learning Research*, 4:415–430, 2003.

[60] J. Pearl. How to do with probabilities what people say you can't. In *Proceedings of 2nd IEEE Conference on AI Applications, Miami, FL, 6-12, December 1985. Also in Charles L. Wesibin (Ed.), AI Applications, Amsterdam: North-Holland, 6-12, l988*, pages 6–12, 1985.

[61] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, California, 1988.

[62] A. Riazanov and A. Voronkov. Adaptive saturation-based reasoning. In *PSI: 4th International Andrei Ershov Memorial Conference, Revised Papers*, volume 2244, pages 1–12, 2001.

[63] A. Riazanov and A. Voronkov. Splitting without backtracking. In *IJCAI*, volume 1, pages 611–617, 2001.

[64] A. Riazanov and A. Voronkov. Limited resource strategy in resolution theorem proving. *Journal of Symbolic Computation*, 36:101–115, 2003.

[65] A. Riazanov and A. Voronkov. Efficient instance retrieval with standard and relational path indexing. In *19th International Conference on Automated Deduction*, volume 199, pages 228–252, 2005.

[66] S. Sahni. *Data Structures, Algorithms and Applications in Java.* McGraw-Hill, first edition, 2000.

[67] M.I. Shahriar. Analytical solution for first-order probabilistic logic. Master's thesis, Concordia University, Montreal, Canada, on-going.

[68] M. Smithson. *Ignorance and Uncertainty: Emerging Paradigms.* Springer Verlag, New York, first edition, 1989.

[69] B. Trakhtenbrot. The impossibility of an algorithm for the decision problem for finite models. *Dokl. Akad. Nauk SSSR, 70:596–572, 1950. English translation in: AMS Transl. Ser. 2*, 23(1063):1–6, 1950.

[70] A.M. Turing. On computable numbers, with an application to the entscheidungsproblem. In *London Mathematical Society*, volume 43, pages 544–546, 1937.

[71] M.A. Weiss. *Data Structures and Problem Solving using Java.* Addition Wesley, second edition, 2002.

[72] L. Wos and G. Robinson. Paramodulation and set of support. In *IRIA Symposium on Automatic Demonstration at Versailles, France*, 1968.

[73] L. Zadeh. Fuzzy sets. *Information and Control*, 8:338–353, 1965.

[74] L. Zadeh. Fuzzy sets as a basis for a theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.

[75] L. Zadeh. Is probability theory sufficient for dealing with uncertainty in AI: A negative view. In *Proceedings of the 1st Annual Conference on Uncertainty in Artificial Intelligence 4 North Holland*, pages 103–106, 1986.

Appendix

Generated Problem corresponds to the used example in Section 4.7 (equation (51)).

```
[POSITIVE FORMULAS]
fof(f_1,axiom,
( ! [X] : ? [Y] : ( big_p(X,Y) | big_q(Y) ))).
fof(f_2,axiom,
( ? [X] : ~ big_r(X) )).
fof(f_3,axiom,
( ? [Y] : ~ big_q(Y) )).
fof(f_4,axiom,
( ! [X,Y] : ( ~ big_p(X,Y) | big_r(X) | big_q(Y) ))).
fof(f_5,axiom,
( ? [X,Y] : ~ big_p(X,Y) )).
[NEGATIVE FORMULAS]
fof(f_1,axiom,
( ? [X] : ! [Y] : ( ~ big_p(X,Y) & ~ big_q(Y) ))).
fof(f_2,axiom,
( ! [X] :  big_r(X) )).
fof(f_3,axiom,
( ! [Y] :  big_q(Y) )).
fof(f_4,axiom,
( ? [X,Y] : ( big_p(X,Y) & ~ big_r(X) & ~ big_q(Y) ))).
fof(f_5,axiom,
( ! [X,Y] : big_p(X,Y) )).
[PROBABILITY]
0.85 0.9
0.55 0.6
0.55 0.6
0.65 0.7
1.0 1.0
[END]
```

Screen shot of a problem file 1p3x4x3p

An illustration of generating consistent probability values for a test instance with 7 first-order formulas.

Generation of consistent probability values.

F_i	W_1	W_2	W_3	W_4	W_5	W_6	W_7	W_8	W_9	W_{10}	W_{11}	W_{12}	Probability Value
F_1	1	1	0	0	0	1	1	0	0	0	0	1	= 5 x 1/12 = 0.42
F_2	1	1	0	0	0	1	1	0	0	0	1	1	= 6 x 1/12 = 0.50
F_3	1	1	0	1	1	1	0	0	0	1	0	1	= 7 x 1/12 = 0.58
F_4	1	1	0	1	1	1	1	0	0	0	1	0	= 7 x 1/12 = 0.58
F_5	1	1	1	0	1	1	1	0	0	1	1	1	= 9 x 1/12 = 0.75
F_6	1	1	1	1	0	1	0	0	0	1	1	1	= 8 x 1/12 = 0.67
F_7	1	1	1	1	1	1	1	1	0	1	1	1	= 11 x 1/12 = 0.92