SCALABLE, LIFTED MAXIMUM A POSTERIORI INFERENCE

by

Somdeb Sarkhel

APPROVED BY SUPERVISORY COMMITTEE:

Vibhav Gogate, Chair

Gopal Gupta

Sanda Harabagiu

Nicholas Ruozzi
SCALABLE, LIFTED MAXIMUM A POSTERIORI INFERENCE

by

SOMDEB SARKHEL, BS, MS

DISSERTATION
Presented to the Faculty of
The University of Texas at Dallas
in Partial Fulfillment
of the Requirements
for the Degree of

DOCTOR OF PHILOSOPHY IN
COMPUTER SCIENCE

THE UNIVERSITY OF TEXAS AT DALLAS
December 2016
ACKNOWLEDGMENTS

I would like to thank my advisor, Vibhav Gogate, without whom this dissertation would not have been possible. He has been an excellent advisor and always motivated me to be creative and to maintain high standards of research. He has also been a mentor who has helped me in every step of my Ph.D. journey. When I first arrived at UTD, he was generous enough to welcome me to his lab and support me through the ARO MURI grant (award number W911NF-08-1-0242), despite my lack of research experience. His constant motivation and innovative insight to seemingly unsolvable problems have helped me progress in my research path.

Next, I would like to thank members of my dissertation committee, Dr. Gopal Gupta, Dr. Sanda Harabagiu, and Dr. Nicholas Ruozzi, for their time, comments on my dissertation, constant encouragement, and many pieces of advice about life in and outside academia. I also wish to thank my collaborators, Deepak, Li, Tuan, Dr. Parag Singla and Dr. Nicholas Ruozzi for collaborating with me on various projects. I have gained a lot of insight into several different research areas as a result of these interactions.

I would also like to thank Deepak for his immense help in my research. Deepak has been a great friend, a proactive collaborator and a wise mentor. Without his help, many of my research endeavors would not have been successful.

I would like to thank various funding agencies: ARO, AFRL, NSF, and DARPA for providing me financial support through grant numbers W911NF-08-1-0242, FA8750-14-C-0021, 1528037, and, FA8750-14-C-0005. I would also like to thank the Department of Computer Science for partially supporting my first year of graduate study through Computer Science Scholarship.
Needless to say, without the love, patience, and support of my wife, Ritu, this dissertation would not have been possible. She has always given me strength and stood by me through the ups and downs of my Ph.D. journey. Were it not for her support I would not have survived in the Ph.D. program.

I would also like to thank my parents who always have encouraged me to pursue my dreams. They have given me the best possible opportunity and have taught me to be ambitious and not to give up. I am indebted for their teaching, encouragement, wishes, and love. I am also thankful to my extended family in India for their constant support throughout the Ph.D. program.

Finally, I will always cherish the friendships that I made at UT-Dallas. Deepak, Tahrima, Li, David, Luis, Tatiana, Tuan and several others have been great friends, and I have enjoyed countless interesting conversations with them about work and life in general. I thank them and will certainly miss their company as I finish my studies.

October 2016
This dissertation focuses on Markov logic networks (MLNs), a knowledge representation tool that elegantly unifies first-order logic (FOL) and probabilistic graphical models (PGMs). FOL enables compact representation while probability allows the user to model uncertainty in a principled manner. Unfortunately, although the representation is compact, inference in MLNs is quite challenging, as PGMs generated from MLNs typically have millions of random variables and features. As a result, even linear time algorithms are computationally infeasible.

Recently, there has been burgeoning interest in developing “lifted” algorithms to scale up inference in MLNs. These algorithms exploit symmetries in the PGM associated with an MLN, detecting them in many cases by analyzing the first-order structure without constructing the PGM, and thus have time and space requirements that are sub-linear when symmetries are present and can be detected. However, previous research has focused primarily on lifted marginal inference while algorithms for optimization tasks such as maximum-a-posteriori (MAP) inference are far less advanced. This dissertation fills this void, by developing next generation algorithms for MAP inference.
This dissertation presents several novel, scalable algorithms for MAP inference in MLNs. The new algorithms exploit both exact and approximate symmetries, and experimentally are orders of magnitude faster than existing algorithms on a wide variety of real-world MLNs. Specifically, this dissertation makes the following contributions:

- A key issue with existing lifted approaches is that one has to make substantial modifications to highly engineered, well-researched inference algorithms and software, developed in the PGM community over the last few decades. We address this problem by developing the “lifting as pre-processing” paradigm, where we show that lifted inference can be reduced to a series of pre-processing operations that compresses a large PGM to a much smaller PGM.

- Another problem with current lifted algorithms is that they only exploit exact symmetries. In many real-world problems, very few exact symmetries are present while approximate symmetries are abundant. We address this limitation by developing a general framework for exploiting approximate symmetries that elegantly trades solution quality with time and space complexity.

- Inference and weight learning algorithms for MLNs need to solve complex combinatorial counting problems. We propose a novel approach for formulating and efficiently solving these problems. We scale-up two approximate inference algorithms, Gibbs sampling and MaxWalkSAT and three weight learning algorithms, Contrastive Divergence, Voted Perceptron, and, Pseudo-log-likelihood learning.

- We propose novel approximate inference algorithms for accurate, scalable inference in PGMs having shared sub-structures but no shared parameters. We demonstrate both theoretically and experimentally that they outperform state-of-the-art approaches.
TABLE OF CONTENTS

ACKNOWLEDGMENTS .............................................. iv
ABSTRACT ......................................................... vi
LIST OF FIGURES ................................................ xii
LIST OF TABLES .................................................. xiv
CHAPTER 1 INTRODUCTION ....................................... 1
  1.1 Contributions ............................................. 2
CHAPTER 2 BACKGROUND ........................................ 7
  2.1 Representation ............................................. 7
    2.1.1 Propositional Logic .................................... 7
    2.1.2 First-Order Logic .................................... 9
    2.1.3 Probabilistic Graphical Models ....................... 11
    2.1.4 Markov Logic Networks ............................... 12
  2.2 Inference ................................................. 15
    2.2.1 Propositional MAP Inference Algorithms ............ 16
    2.2.2 Lifted Inference .................................... 20
  2.3 Learning .................................................. 21
    2.3.1 Weight Learning in MLNs ............................ 21
CHAPTER 3 LIFTED MAP FOR NON-SHARED MLN ................. 23
  3.1 Introduction ............................................. 23
  3.2 Lifted Formulation of MAP Inference ..................... 25
    3.2.1 Domain-Lifted MAP Inference over Non-Shared MLNs . 26
    3.2.2 MAP Inference over non-shared MLNs without Self-Joins . 29
  3.3 Extensions ................................................. 33
    3.3.1 Unit Propagation .................................... 33
    3.3.2 Pure Literal Elimination ............................ 33
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.4 Experiments</td>
<td>34</td>
</tr>
<tr>
<td>3.4.1 Cost vs. Time</td>
<td>35</td>
</tr>
<tr>
<td>3.4.2 Accuracy vs. Domain-Size</td>
<td>36</td>
</tr>
<tr>
<td>3.5 Conclusion</td>
<td>36</td>
</tr>
<tr>
<td>CHAPTER 4 LIFTED MAP USING INTEGER POLYNOMIAL PROGRAMMING</td>
<td>39</td>
</tr>
<tr>
<td>4.1 Introduction</td>
<td>39</td>
</tr>
<tr>
<td>4.2 Probabilistic Theorem Proving Based MAP Inference Algorithm</td>
<td>40</td>
</tr>
<tr>
<td>4.2.1 Integer Polynomial Programming formulation for Lifted MAP</td>
<td>45</td>
</tr>
<tr>
<td>4.2.2 Solving Integer Polynomial Programming Problem</td>
<td>48</td>
</tr>
<tr>
<td>4.3 Experiments</td>
<td>50</td>
</tr>
<tr>
<td>4.4 Conclusion</td>
<td>52</td>
</tr>
<tr>
<td>CHAPTER 5 APPROXIMATE LIFTED MAP USING PARTITIONING</td>
<td>54</td>
</tr>
<tr>
<td>5.1 Introduction</td>
<td>54</td>
</tr>
<tr>
<td>5.2 Scaling up the Partial Grounding Step using Set Partitioning</td>
<td>56</td>
</tr>
<tr>
<td>5.2.1 Generalizing the Partition Grounding Approach</td>
<td>61</td>
</tr>
<tr>
<td>5.3 Experiments</td>
<td>64</td>
</tr>
<tr>
<td>5.4 Conclusion</td>
<td>67</td>
</tr>
<tr>
<td>CHAPTER 6 EXPLOITING EFFICIENT COUNTING STRATEGIES FOR SCALABLE</td>
<td>69</td>
</tr>
<tr>
<td>INFERENCE AND LEARNING</td>
<td></td>
</tr>
<tr>
<td>6.1 Introduction</td>
<td>69</td>
</tr>
<tr>
<td>6.2 #sg as a CSP</td>
<td>72</td>
</tr>
<tr>
<td>6.2.1 Counting the Number of Solutions of the CSP</td>
<td>74</td>
</tr>
<tr>
<td>6.2.2 Junction Trees for Solution Counting</td>
<td>75</td>
</tr>
<tr>
<td>6.3 Application I: Gibbs Sampling</td>
<td>76</td>
</tr>
<tr>
<td>6.4 Application II: MaxWalkSAT</td>
<td>78</td>
</tr>
<tr>
<td>6.5 Inference Experiments</td>
<td>81</td>
</tr>
<tr>
<td>6.5.1 Setup</td>
<td>81</td>
</tr>
<tr>
<td>6.5.2 Results for Gibbs Sampling</td>
<td>82</td>
</tr>
<tr>
<td>6.5.3 Results for MaxWalkSAT</td>
<td>83</td>
</tr>
</tbody>
</table>
LIST OF FIGURES

3.1 Weights of all assignments to ground atoms and (lifted) groups for the non-shared MLN: $[R(x) \lor S(y), w_1]; [R(x), w_2];$ and $[S(y), w_3]$ with domains given by
\[ \Delta(x) = \Delta(y) = \{A, B\}. \] .............................................................. 26

3.2 The total weight of satisfied clauses as a function of counting assignment of $R$ and $S$. The plane is for illustration purpose only. .............................................................. 30

3.3 Cost vs. Time: Cost of unsatisfied clauses (smaller is better) against time for benchmark MLNs for different domain sizes. Notation used to label each figure: MLN-domainsize(number of ground clauses in the MLN). The standard deviation is plotted as error bars. For (b),(c),(e),(f),(h) and (i), no results could be obtained for propositional algorithms since they ran out of memory. .............................................................. 37

3.4 Accuracy vs. Domain-Size: The relative-gap i.e. \( \frac{|optimal - cost|}{optimal} \) is plotted for varying domain-sizes (smaller is better). Every algorithm was given 500 seconds of running-time. The propositional algorithms TUFFY, MWS and ILP run out of memory for larger domain-sizes. Note: We could not run this experiment for IE since the propositional algorithms ran out of memory when the domain-size exceeded 10. .............................................................. 38

4.1 Cost vs. Time: Cost of unsatisfied clauses(smaller is better) vs. time for different domain sizes. Notation used to label each figure: MLN(numvariables, numclauses, numevidences). Note: three quantities reported are for ground Markov network associated with the MLN. Standard deviation is plotted as error bars. .............................................................. 52

5.1 Exchangeable Partition Lattice corresponding to the domain \{1, 2, 3, 4\}. .............................................................. 63

5.2 Cost vs. Time: Cost of unsatisfied clauses(smaller is better) vs. time for different domain sizes. Notation used to label each figure: MLN(numvariables, numclauses). Note: the quantities reported are for ground Markov network associated with the MLN. Standard deviation is plotted as error bars. .............................................................. 67

5.3 Cost vs. Partition Size: Notation used to label each figure: MLN(numvariables, numclauses). .............................................................. 68

6.1 (a) A possible world of an MLN having only one formula: \( f = \forall xyz \ R(x, y) \lor S(y, z) \). The domain of each logical variable is \{A, B\}; (b) Functions $\phi_1$ and $\phi_2$ corresponding to $R(x, y)$ and $S(y, z)$ respectively; and (c) Function $\phi_3$ which is equal to the product of the two functions given in (b). The number of 1s in $\phi_3$ equals the number of groundings of $f$ that evaluate to False. (d) The constraint network for the formula .............................................................. 72
6.2 Results on benchmarks for Gibbs sampling using our approach. **SRate** is the sampling rate (#samples/second) and **C-Time** is the compilation time in seconds. Note that Alchemy timed out (2 hours) or ran out of memory on all the instances and therefore its results are not shown.

6.3 Results on benchmarks for MaxWalkSAT. For each system, we show **C-Time; FRate**, where **C-Time** is the compilation time (in seconds) for our system or the grounding time in Alchemy/Tuffy. **FRate** is the flip rate (#Flips/second). "−" denotes that the system ran out of time or memory.

6.4 Convergence plots. (a) shows a plot of convergence time vs. data size for Protein; (b) shows a plot of change in weights vs. training time (in hours) for WebKB. **ALY** corresponds to Alchemy and **CDw** corresponds to contrastive divergence with $i$-bound $w$.

6.5 Accuracy plots. (a) shows a plot of CLL vs. data size for Protein; (b) shows a plot of CLL vs. training time (in hours) for WebKB. **ALY** corresponds to Alchemy and **CDw** corresponds to contrastive divergence with $i$-bound $w$.

7.1 (a) dMRF associated with $f = \forall x, \forall y, \forall z \neg R(x, +y) \lor S(y, +z)$. The domain of each logical variable is $\{A, B\}$; (b) and (c): Constraints $\phi_1(x, y)$ and $\phi_2(y, z)$ corresponding to the possible world in which the ground atoms $R(A, B), R(B, A)$, and, $S(A, B)$ are true and the rest are false. (d) $\phi_3(x, z)$ corresponds to the parameters $\theta_{x,z}$ associated with the formula.

7.2 Cost vs. Time: Cost of unsatisfied clauses (smaller is better) vs. time for different domain sizes. Notation used to label each figure: MLN(numvariables, num-clauses). Note: the quantities reported are for ground Markov network associated with the MLN. Standard deviation is plotted as error bars.

7.3 Convergence of the Gibbs Sampler.

7.4 Accuracy of the Gibbs Sampler.
### LIST OF TABLES

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6.1</td>
<td>#sg complexities using various strategies. $M$ is the number of formulas, $V_i$ is the set of variables in the CSP encoded for the $i^{th}$ formula, $d$ is the domain-size of each variable and $w^*_i$ is the treewidth of the CSP encoded for the $i^{th}$ formula.</td>
</tr>
<tr>
<td>6.2</td>
<td>Time and Space Complexities for different learning algorithms (showing their scalability potential). $M$ is the number of gradient ascent steps, $N$ is the number of samples (contrastive divergence via Gibbs sampling) or MaxWalkSAT flips (voted perceptron), $F$ is the number of formulas in the MLN, $A$ is the total ground atoms in the MLN and $T$ be the number of IJGP iterations. $i$ is a parameter that controls the maximum cluster size for IJGP and $w^*$ is the maximum treewidth of the encoded CSPs.</td>
</tr>
<tr>
<td>6.3</td>
<td>Dataset sizes. #Evidence is number of true evidence</td>
</tr>
<tr>
<td>6.4</td>
<td>Results on benchmark datasets. The average and standard deviation of the CLL score for five-fold cross validation is shown. Alchemy-VP is the Alchemy version of VP, Alchemy-CD is the Alchemy version of CD using Gibbs sampling and Alchemy-MCSAT is a version of contrastive divergence using MC-SAT, Tuffy-MCSAT is the Tuffy version of contrastive divergence using MC-SAT. X means that we ran out of memory when running the algorithm. * indicates that Alchemy-CD gave us an error during weight learning. ($i = *$) reports the $i$-bound that gave the best result for our respective algorithms.</td>
</tr>
<tr>
<td>6.5</td>
<td>Effect of IJGP $i$-bound on convergence. Convergence times for the CD algorithm when using different IJGP bounds for approximate counting. Larger bounds imply larger complexity. X indicates that we could not converge. * indicates that counting is exact as that $i$-bound is larger than the treewidth of the formulas.</td>
</tr>
</tbody>
</table>
CHAPTER 1
INTRODUCTION

Statistical Relational Learning (SRL) (Getoor and Taskar, 2007; Milch et al., 2005; Pf- effer, 2001; De Raedt et al., 2007; Raghavan and Mooney, 2011; Broecheler et al., 2010; Richardson and Domingos, 2006) seeks to unify first-order logic and probability, and thus solve a fundamental problem in artificial intelligence. Unifying the two has several advantages. By using high-level abstract constructs such as predicates, which express properties of and relationships between objects, and quantifiers such as “for all ($\forall$)” and “there exists ($\exists$)”, first-order logic enables the application designer to compactly represent background knowledge, whereas probability theory makes it easier to capture the uncertainty involved in modeling a real-world problem.

In this dissertation, we focus on a popular SRL framework called Markov logic (Domingos and Lowd, 2009). A Markov logic network (MLN) is a collection of weighted first-order logic formulas. The formulas represent background knowledge and the weights, which are real numbers, express the probability of the corresponding formula being true in a possible world, all other things being equal. Given a set of constants that model objects in the real-world domain, the MLN represents a joint probability distribution over the ground atoms – all possible propositional atoms obtained by substituting each argument of each predicate with a constant. The MLN can then be used to answer any query over the ground atoms using probabilistic inference.

Unfortunately, despite significant progress in algorithms and techniques for probabilistic inference, several real-world MLNs are so complex that inference on them remains out of reach. Moreover, often NP-hard optimization tasks such as maximum-a-posteriori (MAP)
inference (finding the most probable assignment given data or evidence) are solved using algorithms that solve a much harder (∼P-hard) marginal inference task, and dedicated algorithms for the MAP inference are either not available or are far less advanced.

In this dissertation, we fill this void by developing novel scalable algorithms that significantly advance the state-of-the-art in MAP inference in MLNs. Specifically, we identify several key limitations of existing inference algorithms and address them by developing fundamental inference techniques that take advantage of exact and approximate symmetries in the MLN representation.

The road map of this dissertation is as follows. In the next section, we present a brief overview of our contributions. Chapter 2 provides the necessary background to follow the rest of our work, while Chapters 3-7 present our original research contributions. We conclude in Chapter 8 by summarizing our research accomplishments and presenting possible future directions.

1.1 Contributions

As iterated earlier, the main contribution of this dissertation is novel, scalable algorithms for MAP inference in MLNs. Currently (before the work presented in this dissertation), the state-of-the-art approach for solving this task is to use propositional approaches. These approaches first convert the MLN into a PGM by grounding the first-order representation and then run probabilistic inference algorithms, developed in the PGM community on the latter. Unfortunately, a typical PGM generated this way, has tens of millions of variables and features and as a result, even linear time approximate inference approaches such as loopy belief propagation \cite{murphy1999}, and Gibbs sampling \cite{geman1984} are computationally infeasible.

Therefore, recently there has been growing interest in developing lifted inference algorithms. These algorithms take advantage of relational structure or symmetries \cite{poole2003}.
In Chapter 3, we introduce “lifting as pre-processing” paradigm (Sarkhel et al., 2014b). The key idea is to construct a “lifted” Markov network such that the MAP solution over the lifted network can be used to construct a MAP solution over the MLN (in linear time), and its size is much smaller than the ground Markov network. We achieve this by introducing a new lifting rule, the “isolated singleton” rule. It states that for non-shared MLNs (i.e., MLNs in which first-order formulas have no shared terms), there exists a MAP solution in which all ground atoms of a predicate are either all true or all false. This rule enables us to reduce the MAP inference over the MLN to the MAP inference over a Markov network which has one Boolean variable for each predicate and one feature for each first-order formula. We can then use any off-the-shelf MAP solver (for graphical models) on the Markov network. Unlike propositional techniques, which operate on the much larger ground Markov network, the proposed technique operates on a much smaller network whose size is bounded by the number of first-order predicates (rather than the number of ground atoms).

Although the above approach applies to any arbitrary MLN, it is efficient only for non-shared MLNs and does not use existing research on lifted inference to the fullest extent.
Therefore, in Chapter 4, we propose another lifted MAP inference approach, based again on the lifting as pre-processing paradigm, and which subsumes our approach described in Chapter 3. Moreover, the new approach is at least as powerful as probabilistic theorem proving (Gogate and Domingos, 2011), a classic search-based lifted inference technique. The key idea in our approach is to reduce the lifted MAP inference problem to an equivalent Integer Polynomial Program (IPP) (Sarkhel et al., 2014a). Each variable in the IPP potentially refers to an assignment to a large number of ground atoms in the original MLN. Hence, the size of the search space of the generated IPP can be significantly smaller than the ground Markov network. To solve the generated IPP we convert it to an equivalent Integer Linear Program (ILP) using a classic conversion method outlined in (Watters, 1967). A desirable characteristic of our reduction is that we can use any off-the-shelf ILP solver to get an exact or approximate solution to the original problem.

In Chapter 5, we present schemes that address a key issue associated with lifted inference algorithms: when pre-conditions for lifted rules are not satisfied, the algorithms (partially) ground some first-order atoms in the MLN. The latter is computationally expensive and can significantly increase the complexity of inference. To circumvent this issue, we propose (Sarkhel et al., 2015) to exploit approximate in addition to “exact symmetries” in the first-order structure. Our approach is straight-forward: partition the ground atoms into groups and then force the inference algorithm to treat all atoms in each group as indistinguishable (symmetric). We prove that our proposed approach yields a consistent assignment that is a lower-bound on the MAP value. We show that the quality of the MAP solution can be improved systematically by refining the partitions. We further demonstrate how to improve the complexity of our refinement procedure by exploiting the exchangeability property of successive refinements. Specifically, we show that we can arrange the exchangeable refinements on a lattice, and explore the lattice using a heuristic search procedure to yield an efficient any-time, any-space algorithm. We show experimentally that our method is highly
scalable and yields close to optimal solutions in a fraction of the time as compared to existing approaches.

In chapter 6, we take a novel approach that is quite distinct from the typical “symmetry-exploiting” techniques used in chapters 3-5. We consider the task of counting the number of true groundings of a formula in a given world, a key sub-task in many local-search based algorithms for MAP inference such as MaxWalkSAT. Existing MLN systems solve this counting problem using the following naïve “generate-and-test” approach: generate all possible groundings of the formula and test if each is true in the given world. This naïve approach is the main reason for their poor scalability. In this chapter, we propose a novel, practical approach for solving this counting problem by encoding each formula as a CSP (Constraint Satisfaction Problem). The key advantages of our approach are that it does not ground the non-unit formulas and in most cases is orders of magnitude better than the “generate-and-test” approach. However, the real power of our encoding is its generality. That is, with this encoding, all guarantees, approximations and algorithms from graphical models and CSPs can be leveraged to solve the counting problem efficiently and thereby scale up inference and learning. We use this advanced counting technique to implement efficient, scalable approximate inference (Venugopal et al. 2015) and learning algorithms (Sarkhel et al. 2016). Our experiments clearly show that in several real-world MLN benchmarks, our approach is orders of magnitude more scalable than existing state-of-the-art inference algorithms.

In Chapter 7, we consider MLNs in which different groundings of a first-order formula have different weights. Such MLNs are used frequently in real-world domains such as information extraction and present a number of challenges for inference. In particular, the MLNs have no symmetries and as a result lifted inference algorithms perform poorly on them. Moreover, they render even advanced inference techniques, presented in chapter 6 computationally inefficient. We propose two different approaches for this realistic setting. First, we show that the problem of computing the weight of a world in these MLNs, a key sub-task in
inference algorithms, can be reduced to computing the partition function of a Markov network defined over the logical variables in the first-order formulas. This yields a general approach that is guaranteed to be more efficient than previously proposed approaches. Second, we propose an approximate approach having small computational complexity, which is useful when computing the partition function of the Markov network is computationally infeasible. We show experimentally that our new approaches, when used as a sub-routine in popular inference algorithms such as Gibbs sampling and MaxWalkSAT, significantly improve their accuracy and scalability.
In this chapter, we present brief background on representation, inference and learning in Markov logic. For more details on first-order logic, refer to (Russell and Norvig, 2003; Genesereth and Kao, 2013). For more details on probabilistic graphical models, refer to (Koller and Friedman, 2009; Darwiche, 2009). For a more detailed explanation of Markov logic, refer to (Domingos and Lowd, 2009). For an in-depth overview of local search based methods, refer to (Hoos and Stützle, 2004). For details on Monte Carlo methods, refer to (Liu, 2001).

2.1 Representation

2.1.1 Propositional Logic

Propositional logic is the study of propositions. A proposition is a statement or expression that must be either True or False. A propositional statement (or propositional formula, or simply formula) is constructed by connecting other propositions using logical connectives such as such as $\land$ (conjunction), $\lor$ (disjunction), $\neg$ (negation), $\Rightarrow$ (implication) and $\Leftrightarrow$ (equivalence). An atomic proposition (or simply an atom) is a proposition which does not contain any logical connective. A literal is defined as an atom or a negation of an atom.

In this thesis, we will focus on conjunctive and formulas only. A conjunctive formula is a conjunction of literals while a disjunctive formula is a disjunction of literals.

Each atom in propositional logic takes a value from the set $\{True, False\}$. Given an assignment $x$, we say that a formula $F$ is satisfied, if $x$ evaluates $F$ to true. A conjunctive formula is true if all of its literals are set to true. A disjunctive formula is true, if at least
one of its literals is true. The satisfying assignment is referred to as a model of the formula $F$.

**Example 1.** Let $p$, $q$, and, $r$ be propositional atoms, then $F = p \lor \neg q \lor r$ is a propositional formula. Here $p$, $\neg q$, $r$ are the three literal appearing in $F$. $p = True$, $q = True$, and, $r = False$ is a satisfying assignment (i.e., model) of $F$.

**Definition 1.** Satisfiability (or boolean satisfiability, SAT) is the problem of deciding whether a given formula is satisfiable.

The SAT problem is known to be NP-complete. It is the first formula to be shown to belong to the NP-complete class. There are many versions of the SAT problem, most notable among them is the Conjunctive Normal Form Satisfiability problem (CNF-SAT). We will describe it next.

**Definition 2.** A formula is in Conjunctive Normal Form (CNF) if it is a conjunction of clauses, where a clause is a disjunction of literals (namely, a disjunctive formula).

In the above example (Example 1) the formula $F$ contains only one clause and is in CNF. When the SAT problem is restricted to only formulas which are in CNF, it is known as the CNF-SAT. In this dissertation, we will always assume that a formula is in CNF and therefore by satisfiability, we mean CNF satisfiability. There are many extensions of the original SAT problem, and some of them are closely related to many inference problems discussed in this dissertation. We will briefly describe a few of them.

**Definition 3.** Maximum Satisfiability (or Max-SAT) is the optimization extension of CNF-SAT, which asks for the maximum number of clauses that can be simultaneously satisfied given a CNF formula.

**Example 2.** $F = (p \lor \neg q \lor r) \land (\neg p) \land q \land (\neg r)$ is a CNF with no satisfying assignment. However, an assignment $p = True$, $q = True$, and, $r = False$ satisfies three clauses (all except $(\neg p)$) and is a solution to the Max-SAT problem.
In the Max-SAT problem, it is assumed that all the clauses have the same importance. However, we may be interested in satisfying some clauses more than others. The weighted variant of Max-SAT (Weighted Max-SAT) models this by attaching a weight to each clause. For a given assignment if a clause is satisfied the weight is counted towards the assignment. The goal of the Weighted Max-SAT problem is to find an assignment that maximizes the total weight of satisfied clauses. Although in classical SAT literature it is assumed that the weights are positive integer numbers, in this dissertation, we will consider the general version in which weights are real numbers. Both Max-SAT and Weighted Max-SAT are NP-hard problems. However, many efficient solvers (both complete and incomplete) exist for them. Notable solvers include MiniMaxSAT (Heras et al., 2008), Clone (Pipatsrisawat and Darwiche, 2007), and MaxWalkSAT (Selman et al., 1996).

2.1.2 First-Order Logic

Propositional logic cannot express structures and relationships between and within an atomic proposition, and thus has limited representation power. First-order logic generalizes propositional logic by introducing three additional constructs: terms, predicates, and quantifiers. Throughout this dissertation, we will use finite Herbrand logic (cf. Genesereth and Kao, 2013), which is a strict subset of first-order logic. Unlike first-order logic, finite herbrand logic assumes that: (i) there are finitely many constants; (ii) there are no other objects in the domain other than those designated by the constants (domain-closure assumption); (iii) there are no function symbols; and (iv) different constants refer to different entities in the application domain (unique-name assumption). Furthermore, whenever we mention first-order logic in this dissertation, we imply finite Herbrand logic.

The syntax of a first-order logic formula is as follows: (i) A term is any constant or any variable (ii) An atomic formula is a predicate or its negation applied to one or more terms; (iii) Negation of a formula is a formula; (iv) If $f$ and $g$ are formulas then connecting them by
binary connectives yields a formula, in the same fashion as in propositional logic; and (v) If \( f \) is a formula and \( x \) is a variable then \( \forall x f \) and \( \exists x f \) are formulas.

Throughout, we assume that each predicate is typed, i.e., each term of a predicate is associated with a fixed set of constants. We will refer to this set as the domain of the term. We further restrict ourselves to the case where each first-order-logic formula is a clause (i.e., disjunction of predicates or their negation), and each variable in a formula is universally quantified.

We use lower case letters (e.g., \( x, y, z, \) etc.) to denote variables (we will call them logical variables to distinguish them from random variables). Constants (or objects) are denoted by strings that begin with an upper case letter (e.g., \( A, Ana, Bob, \) etc.). The domain of a variable \( x \) is denoted by \( \Delta(x) \). We denote predicates by strings in typewriter font (e.g., \( R, S, Smokes, \) etc.) followed by a parenthesized list of terms. We will use the notation \( i_R \) to denote the \( i \)-th argument of predicate \( R \).

We define a ground atom as an atomic predicate which does not have any logical variables, i.e., where each logical variable in the predicate is replaced by an object from the corresponding domains. Similarly, a ground formula is a first-order formula which contains only ground atoms. The process of grounding is to replace all the logical variables of a formula by object from corresponding domains. A first-order Knowledge Base (or KB) is a set of first-order logic formulas. A world is a truth assignment to all possible ground atoms of all predicates present in a KB. A ground KB is a KB containing all possible groundings of all of its formulas.

Example 3. \( \forall x \text{ Man}(x) \Rightarrow \text{ Mortal}(x) \) is a first-order logic formula. Let the domain associated with \( x \) be \( \Delta(x) = \{ \text{Socrates, Plato} \} \). Here, \( \text{Man(Socrates)}, \text{Mortal(Plato)} \) are some example of ground atoms. The above formula has only two possible ground formulas:

\[
\text{Man(Socrates)} \Rightarrow \text{Mortal(Socrates)} \quad \text{and} \quad \text{Man(Plato)} \Rightarrow \text{Mortal(Plato)}.
\]
An example of a world for the above formula is:

\[ \left\{ \text{Man}(Socrates) = \text{true}, \text{Mortal}(Socrates) = \text{false}, \right. \]
\[ \text{Man}(Plato) = \text{false}, \text{Mortal}(Plato) = \text{true} \].

2.1.3 Probabilistic Graphical Models

A key disadvantage of propositional and first-order logic is that they cannot model uncertainty. Uncertainty can stem from many different factors, and often is an inescapable aspect of most real-world problems. It can be caused by our lack of complete knowledge of the systems, or by the presence of noise in our observation. Because of this ubiquitous nature of uncertainty we need to reason about the probable states of a system. One natural way to model this is to use probabilistic models, which associate a number between 0 and 1, called the probability to all possible states of a system, such that the numbers add up to 1. This specification of all possible probabilities is often known as the joint probability distribution (or simply joint distribution).

Although, probabilistic modeling is a powerful technique for handling uncertainty, it is not compact. For example, a system containing only six binary variables requires us to specify \(2^6 = 64\) probability entries. Therefore, for a sufficiently complex system, specifying the joint distribution is not computationally feasible. Probabilistic Graphical Models (PGMs) address this issue by compactly encoding the joint distribution over many variables using a graph-based representation. Two of the most popular families of PGMs are: (i) Bayesian Network which uses a directed graph, and, (ii) Markov Network or Markov Random Field (MRF) which uses an undirected graph. The representation used in this dissertation, the Markov Logic Networks, is closely related to special type of MRFs, the log-linear Markov Random Fields. We will discuss them briefly next.
Log-Linear Markov Random Fields

We begin by defining our notation. We represent variables by capital letters (e.g., $X$), values in the domain of a variable by corresponding small letters (e.g., $x$) and an assignment of a value $x$ to a variable $X$ by $\bar{x}$. We represent sets of variables by bold capital letters (e.g., $X$) and assignment of values to all variables in the set $X$ by $\bar{x}$. For simplicity of presentation, we will assume that all domains are binary unless otherwise noted.

A log-linear probabilistic Markov random field, denoted by $\mathcal{M}$ is a triple $\langle X, F, \theta \rangle$ where $X = \{X_1, ..., X_n\}$ is a set of variables, $F = \{F_1, ..., F_m\}$ is a set of features and $\theta = \{\theta_1, \ldots, \theta_m\}$ is the set of weights (parameters) such that $\theta_i$ is the weight of feature $F_i$. $\mathcal{M}$ represents the following probability distribution

$$P(\bar{x}) = \frac{1}{Z} \exp \left( \sum_i \theta_i I_i(\bar{x}) \right)$$

(2.1)

where $I_i(\bar{x})$ is 1 if the assignment $\bar{x}$ evaluates $F_i$ to true or 0 otherwise, and

$$Z = \sum_\bar{x} \exp \left( \sum_i \theta_i I_i(\bar{x}) \right)$$

is the normalization constant, also known as the partition function.

2.1.4 Markov Logic Networks

Markov logic networks (MLNs) (Richardson and Domingos [2006] Domingos and Lowd [2009]) generalize first-order logic as well as log-linear MRFs, by attaching weights to each first-order logic formula. Formally, an MLN is a set of pairs $(f_i, \theta_i)$ where $f_i$ is a first-order logic formula and $\theta_i$ is a real number. Given a set of constants (the domain objects or objects), an MLN represents a “ground” log-linear model, which has one random variable for each ground atom and one propositional feature for each grounding of each formula. The weight associated with
a feature is the weight attached to the corresponding formula. The ground MLN represents the following probability distribution:

$$P(\omega) = \frac{1}{Z} \exp \left( \sum_i \theta_i N_{f_i}(\omega) \right)$$  \hspace{1cm} (2.2)$$

where $\omega$ is a world and $N_{f_i}(\omega)$ is the number of groundings of $f_i$ that evaluate to True given a world $\omega$ and $Z$ is the normalization constant. We will refer the problem of computing $N_{f_i}(\omega)$ as the $\#sg$ problem (short for number of satisfied grounding). Many iterative inference algorithms need to solve this $\#sg$ problem at every iteration. Unfortunately, it is NP-hard in general (naively, it can be solved in time that is exponential in the number of logical variables in $f_i$), and is the chief reasons for the poor scalability of many inference algorithms. Formally,

**Theorem 1.** Counting the number of true groundings of a first-order clause in a given world is $\#P$-complete in the length of the clause.

**Normal MLN**

Throughout this dissertation, we will assume that we are given an MLN in normal form (Jha et al., 2010) defined below.

**Definition 4.** A normal MLN is an MLN that satisfies the following two properties: (1) There are no constants in any formula, and (2) If two distinct atoms with the same predicate symbol have variables $x$ and $y$ in the same position then $\Delta(x) = \Delta(y)$.

Because of the second condition, in normal MLNs, we can associate domains with each argument of a predicate. This enables us to extend our $\Delta$ notation to arguments of a predicate. In other words, if $i_R$ is the $i$-th argument of the predicate $R$, then $\Delta(i_R)$ denotes the domain associated with $i_R$. Moreover, for inference purposes, in normal MLNs, we do
not have to keep track of the actual elements in the domain of a variable, all we need to know is the size of the domain \( |\Delta(i_{R})| \) [Jha et al. 2010]. Hence, we also introduced this compact notation \( D(i_{R}) = |\Delta(i_{R})| \), to denote the number of objects in the domain of \( i_{R} \).

**Example 4.** Consider an MLN having two formulas \((\text{Smokes}(x) \Rightarrow \text{Cancer}(x), \theta)\) and \((\text{Smokes}(\text{Bob}), \infty)\) (the infinite weight denotes that it is an evidence or observed atom). It is clear that this formula is not in the normal form. Its normal form has following three formulas: \((\text{Smokes}(x') \Rightarrow \text{Cancer}(x'), \theta)\), \((\text{Smokes}(y) \Rightarrow \text{Cancer}(y), \theta)\) and \((\text{Smokes}(y), \infty)\), where \( \Delta(x') = \Delta(x) \setminus \{\text{Bob}\} \) and \( \Delta(y) = \{\text{Bob}\} \).

Any MLN can always be converted to a normal MLN (since a ground MLN is trivially in the normal form). Henceforth, we will abuse notation and refer to normal MLNs as MLNs.

**Untied MLN**

In many practical settings (e.g., for many information extraction tasks such as event extraction [?]), the parameter tying assumption, namely the assumption that all groundings of a first-order formula have the same weight, is too strong. MLNs relax this constraint using the “+” operator. We learn a separate weight for each grounding of each logical variable associated with a “+” sign.

**Example 5.** Let \( f = \forall x, \forall y, \forall z \neg R(x, y) \lor S(y, z) \) is one of the formula of the MLN. Then to specify that different weights are attached to different groundings of \( x, z \) of \( f \), we use the following formula:

\[
f' = \forall x, \forall y, \forall z \neg R(+x, y) \lor S(y, +z)
\]

We will use \( \mathcal{X}_+ \) to denote the set of logical variable associated with the ‘+’ operator. To denote the weights (or parameters) associated with \( f \) we will use \( \theta_f \). A complete assignment to all the variables in \( \mathcal{X}_+ \) (denoted by \( \mathfrak{X}_+ \)) corresponds to exactly one parameter from the
parameter set $\theta$. Hence, we will denote each such individual parameter by $\theta_{x_+}$. For example, in the above example if $\Delta(x) = \{A, B\}$ and $\Delta(y) = \{C, D\}$ then the parameter set is, $\theta_f = \{\theta_{A,C}, \theta_{B,C}, \theta_{A,D}, \theta_{B,D}\}$.

For these MLNs we can rewrite Equation 2.2 as follows:

$$P(\omega) = \frac{1}{Z} \exp \left( \sum_i W_{f_i}(\omega) \right) \quad (2.3)$$

where, $W_{f_i} = \sum_{x_+} \theta_{x_+} N_{x_+}(\omega)$, is the total weight of the satisfied groundings of a first order formula $f_i$. We will refer the task of computing the expression $W_{f_i}$ as $\#\text{wsg}$ (short for, total weight of satisfied grounding). Since, $\#\text{wsg}$ generalizes $\#\text{sg}$, it is obvious from Theorem 1 that no efficient algorithm can exist for solving $\#\text{wsg}$ problem.

2.2 Inference

In probabilistic graphical models, the term inference (or probabilistic inference) usually refers to reasoning about probabilities of random variables in the model. Three common inference task in MLNs are:

(i) Computing the partition function, i.e.,

$$Z = \sum_{\omega} \exp \left( \sum_i \theta_i N_{f_i}(\omega) \right) \quad (2.4)$$

(ii) Finding the most probable state of the world $\omega$, that is finding a complete assignment to all ground atoms which maximizes the probability. This task is also called Maximum a Posteriori (MAP) inference and can be formally stated as follows:

$$\arg \max_{\omega} P(\omega) = \arg \max_{\omega} \frac{1}{Z} \exp \left( \sum_i \theta_i N_{f_i}(\omega) \right) \quad (2.5)$$
(iii) Computing the probability of a query variable given evidence, i.e., \( P(Q|E) \). This is often referred to as the marginal inference in literature.

The MAP inference task is an optimization problem or max-product problem, while the task of computing the partition function as well as the probability of evidence task is a counting problem or sum-product problem. All of these tasks are computationally intractable (NP-hard).

Since a ground MLN represents a probabilistic graphical model, inference in MLN can be carried out by first grounding it. This gives the flexibility to use any existing graphical model inference algorithm. However, this approach is not scalable as even for simpler MLNs the ground network can be quite large. For example, a simple MLN formula, \( \text{Friends}(x, y) \land \text{Friends}(y, z) \Rightarrow \text{Friends}(x, z) \) has 1 million groundings for only 100 objects in the domain. Hence we need highly scalable inference algorithms to be able to infer on any real-world MLN. However, before describing our scalable inference algorithms we will discuss some of the propositional inference algorithms for the Markov Logic.

### 2.2.1 Propositional MAP Inference Algorithms

Here we will discuss some common techniques for the propositional MAP inference task. For more details about the partition function computation or to compute the probability of evidence please refer to [Koller and Friedman 2009](#).

We start by converting the MAP problem from the max-product form to the max-sum form. Since the exponential function is a monotonically increasing function, the partition function \( Z \) is a constant for a given MLN , using Equation 2.5 we can rewrite the MAP problem as follows:

\[
\arg \max_{\omega} P(\omega) = \arg \max_{\omega} \left( \sum_i \theta_i N_{f_i}(\omega) \right)
\]  

From our previous discussion on extension of SAT problems, it is clear that the above formulation is a Weighted Max-SAT formulation. Hence we can use any weighted Max-SAT
solver to solve the MAP problem. Here we will discuss two popular methods for solving them, an exact approach and an approximate approach.

**ILP formulation of Weighted Max-SAT**

A popular approach for solving the Weighted Max-SAT problem is to first convert it into an *Integer Linear Program* (ILP), and then use optimized off-the-shelf ILP solver to solve them. Although there are many different ways of converting the Weighted Max-SAT problem to an ILP problem, we will describe here the most straightforward one due to Noessner et. al. (Noessner et al., 2013).

Let $F = \{(f, w_f)\}$ be the set of weighted clauses. For a given clause $f$, we have two sets $L^+(f)$ and $L^-(f)$ denoting sets of literals which appear unnegated or negated in the clause $f$ respectively. For each variable $g$ in the Weighted Max-SAT problem, we introduce a new boolean ILP variable $x_g$. For each hard clause (i.e., a clause with infinitely large weight) we add the following linear constraint to the ILP:

$$\sum_{g \in L^+(f)} x_g + \sum_{g \in L^-(f)} (1 - x_g) \geq 1$$

For every finitely weighted clause $f$ we introduce another binary ILP variable $z_f$. If the weight of the clause is positive, we add the following constraint to the ILP:

$$\sum_{g \in L^+(f)} x_g + \sum_{g \in L^-(f)} (1 - x_g) \geq z_f$$

However, if the weight of the clause is negative, then the added constraint is:

$$\sum_{g \in L^+(f)} x_g + \sum_{g \in L^-(f)} (1 - x_g) \leq (|L^+(f)| + |L^-(f)|)z_f$$
Finally, the objective of the ILP is:

\[
\arg \max \sum_{f \in F} w_f z_f
\]

The obtained ILP solution corresponds to a Weighted Max-SAT solution \( \bar{g} \) where \( g = \text{true} \) if the corresponding ILP variable \( x_g \) is 1 and \( g = \text{false} \) otherwise.

**Example 6.** Consider three weighted clauses \((x_1 \lor \neg x_2 \lor x_3, 1.5), (x_1 \lor \neg x_2, -0.5), \) and, \((\neg x_2 \lor x_3, \infty)\). The ILP formulation for this Weighted Max-SAT problem is as follows:

\[
\begin{align*}
\text{max} & \quad 1.5z_1 - 0.5z_2 \\
\text{Subject to} & \\
& x_1 + (1 - x_2) + x_3 \geq z_1 \\
& x_1 + (1 - x_2) \leq 2z_2 \\
& (1 - x_2) + x_3 \geq 1
\end{align*}
\]
MaxWalkSAT

Algorithm 1: MaxWalkSAT(\(F = \{(f_i, w_i)\}, p, \text{maxFlips}\))

**Initialize:** \(\bar{x} = \) a random assignment to all variables in \(F\).

\(\bar{x}_{Best} = \bar{x}\)

**for** flip = 1 to maxFlips **do**

**if** \(\bar{x}\) satisfies all clauses in \(F\) **then**

**return** \(\bar{x}\)

\(f_k = \) a random unsatisfied ground clause in \(F\)

**With** probability \(p\)

Flip a randomly selected literal of \(f_k\)

**Else**

Flip the literal of \(f_k\) that yields the maximum weight

**if** Satisfied weights better than \(\bar{x}_{Best}\) **then**

\(\bar{x}_{Best} = \bar{x}\)

**return** \(\bar{x}_{Best}\)

The most commonly used solver for propositional MAP inference in MLNs is MaxWalkSAT (Kautz et al., 1997). It is a weighted variant of WalkSAT, a local-search algorithm for satisfiability testing (Selman et al., 1996) and is the algorithm used in Alchemy (Kok et al., 2006) as well as Tuffy (Niu et al., 2011) for MAP inference.

The MaxWalkSAT algorithm given in Algorithm 1 takes as input a set of weighted clauses \(F\), an integer \(\text{maxFlips}\) which determines the maximum number of flips and a probability \(p\). The algorithm begins by randomly generating a complete assignment. Then, at each iteration, it selects an unsatisfied clause uniformly at random and flips the value assigned to one of its literal. With probability \(p\), the literal to be flipped is selected uniformly at random and with probability \(1 - p\), the literal when flipped maximizes the number of satisfied ground clauses is selected (greedy hill-climbing step which selects the best literal).
2.2.2 Lifted Inference

Since propositional inference algorithms need to ground an MLN to perform inference, they are not scalable. Inspired by resolution theorem proving (Robinson, 1965), lifted inference tries to achieve the goal of performing inference without grounding the MLN. Starting with the pioneering works of Poole (Poole, 2003), researchers have developed many lifted inference algorithms. Although most of these algorithms focus on marginal inference, recently development in lifted MAP inference has also picked up. Popular lifted inference algorithms for marginal inference are: First Order Variable Elimination (FOVE) (de Salvo Braz, 2007), Lifted Belief Propagation (Singla and Domingos, 2008), Probabilistic Theorem Proving (PTP) (Gogate and Domingos, 2011), Weighted First Order Model Counting (WFOMC) (Van den Broeck et al., 2011), Lifted Importance Sampling (Gogate et al., 2012), and, Lifted Gibbs Sampling (Venugopal and Gogate, 2012). Well-known lifted MAP inference algorithms include: Exploiting Uniform Assignments for Lifted MPE (Apsel and Brafman, 2012), Lifted Variational Inference using Graph Automorphism (Bui et al., 2013), Exploiting Symmetry for MAP Inference (Noessner et al., 2013) and Efficient Lifting of MAP LP Relaxations using k-Locality (Mladenov et al., 2014).

At a high level, lifted algorithms detect symmetries in the first-order specification using different lifted inference rules (Jha et al., 2010; Sarkhel et al., 2014b; Mittal et al., 2014) and then use these symmetries to infer over multiple objects simultaneously. One of the key challenges in lifted inference research is to identify these lifting rules. An ideal lifted inference algorithm is one which can infer only from the first-order specification and runs on any MLN irrespective of the domain size. These algorithms are usually referred to as the domain independent lifted algorithm. However, for an arbitrary structured MLN this is not possible (Van den Broeck, 2011a), and for these MLNs a desirable property is domain liftability (Van den Broeck, 2011a; Jaeger, 2015), which is formally defined as follows:
Definition 5. An MLN is domain liftable if an exact inference algorithm exists that can run on the MLN in time and space polynomial in the size of the domain.

2.3 Learning

The objective of learning algorithms is to construct or refine an MLN from the data automatically. The two types of learning are weight learning and structure learning. In weight learning, we try to find the optimal weights of an MLN from the data assuming that the structure of the MLN (i.e., the formulas in the MLN) is given. In structure learning, we learn the MLN formulas as well as their weights. Throughout this dissertation, we will assume the structure of the MLN is given. For an overview of structure learning algorithms, please refer to [Domingos and Lowd, 2009]. We briefly describe popular weight learning algorithms next.

2.3.1 Weight Learning in MLNs

Weights of an MLN can be learned generatively by maximizing the log-likelihood of the database or discriminatively by maximizing the conditional log-likelihood. In discriminative learning, we know which atoms are going to be the query atom at the time of inference, and hence we learn a distribution only over those atoms. Here we describe (and derive equations for) only generative learning noting that discriminative learning can be derived in a similar manner. The log-likelihood of an MLN is given by:

\[
\ell(\theta : \omega) = \log P_\theta(\omega) = \sum_i \theta_i N_i(\omega) - \log Z_\theta \\
= \sum_i \theta_i N_i(\omega) - \log \left( \sum_{\omega'} \exp \left( \sum_i \theta_i N_i(\omega') \right) \right) 
\]

(2.7)
Weights that optimize the log-likelihood (a convex function) can be learned using a standard gradient ascent procedure. The gradient of the log-likelihood w.r.t $\theta_i$ is given by:

$$
\frac{\partial}{\partial \theta_i} \ell(\theta : \omega) = N_i(\omega) - \sum_{\omega'} P_\theta(\omega') N_i(\omega')
$$

$$
= N_i(\omega) - \mathbb{E}_\theta[N_i(\omega)]
$$

where the sum is over all possible databases $\omega'$, and $P_\theta(\omega')$ is computed using the current weight vector $\theta = (\theta_1, ..., \theta_i, ...)$. The second term in the gradient requires (marginal) inference over the ground Markov network and is $\#P$-hard in the worst-case. Therefore, in practice, the expectation is approximated using either Markov Chain Monte Carlo (MCMC) sampling or MAP inference. The idea is to replace the expectation by either an average over the worlds sampled using a MCMC procedure or the MAP tuple. The former yields a class of algorithms called contrastive divergence (CD) ([Hinton, 2002]), while the latter yields a class of algorithms called Voted Perceptron (VP) ([Collins, 2002]).

An alternative approach is to change the likelihood (objective) function such that the gradient is tractable. An example of such a function is pseudo log-likelihood (PLL) ([Besag, 1975]), defined below:

$$
\ell_{PL}(\theta : \omega) = \sum_{l=1}^{n} \log P_\theta(X_l = v | MB_\omega(X_l))
$$

(2.8)

where $X_1, ..., X_n$ are the ground atoms and $MB_\omega(X_l)$ is the state of $X_l$’s Markov blanket in world $\omega$. Its gradient is given by:

$$
\frac{\partial}{\partial \theta_i} \ell_{PL}(\theta : \omega) = \sum_{l=1}^{n} \left( N_i(\omega) - \sum_{v \in \{0,1\}} P_\theta(X_l = v | MB_\omega(X_l)) N_i(\omega | [X_l=v]) \right)
$$

(2.9)

where $\omega | [X_l=v]$ denotes the world obtained from $\omega$ by changing the assignment of $X_l$ in it to $X_l = v$. 
CHAPTER 3
LIFTED MAP FOR NON-SHARED MLN

3.1 Introduction

In this chapter, we present a general approach for scaling up MAP inference in Markov logic networks (MLNs) using the “lifting as pre-processing” paradigm (Shavlik and Natarajan, 2009). In recent times, many “lifted inference” algorithms have been proposed to address the scalability issue of inference tasks in Markov logic. However, most of these algorithms either focus on the marginal inference task, a much harder task than the MAP task, or they require significant modification to propositional MAP algorithms. Hence, progress in MAP research has been slow, and dedicated MAP algorithms are far less advanced than algorithms for other tasks.

To circumvent this problem, in this chapter, we advocate the use of “lifting as pre-processing” paradigm for the MAP inference task. The key idea is to apply lifted inference as a pre-processing step and to construct a “lifted” Markov network such that the MAP solution in this lifted network corresponds to the MAP solution in the MLN, and its size is much smaller than the ground Markov network corresponding to that MLN.

We show that when the MLN contains no formulas having shared terms (we refer to such MLNs as a non-shared MLNs), MAP inference is domain liftable (Van den Broeck, 2011b), namely it is polynomial in the domain size of the logical variables in the MLN. In particular, we show that in non-shared MLNs, the set of full assignments having cardinality $O(2^{\sum_{i=1}^{n} d_i})$, where $n$ is the number of predicates in the MLN and $d_i$ is the number of possible groundings...
of the $i$-th atom in the MLN\(^1\) can be partitioned into $O(\prod_{i=1}^{n}(d_i + 1))$ subsets such that each element in a subset has the same probability. Thus, instead of performing a search for the MAP solution over $O(2\sum_{i=1}^{n}d_i)$ assignments as the ground inference algorithms do, we can carry out the search over an exponentially smaller $O(\prod_{i=1}^{n}(d_i + 1))$ space, yielding a lifted MAP inference algorithm.

We further extend this result by showing that if the non-shared MLN has no self-joins, namely each atom appears at most once in each formula, then one of the MAP solutions is guaranteed to lie at the following extreme points: for each atom, all of its groundings are either all true or all false. This helps us to further reduce the complexity of inference from $O(\prod_{i=1}^{n}(d_i + 1))$ to $O(2^n)$, i.e., the complexity of inference is independent of the domain size.

We utilize the results mentioned above by developing a mapping from the lifted search space to propositional search space of the same size. This helps us reformulate the MAP inference task over non-shared MLNs as propositional MAP inference over a Markov network such that: (i) the number of random variables in the Markov network is equal to the number of atoms in the MLN (i.e., equal to $n$) and (ii) the domain size of each random variable is either $d_i$ or 2 depending upon whether the non-shared MLN has self-joins or not. The two key features of this formulation are: (i) we can plug in any known propositional MAP inference algorithm for inference on this Markov network; and (ii) since the propositional algorithm operates in the lifted search space, it has the same performance guarantees as a lifted algorithm. Thus, by plugging in different propositional MAP inference algorithms, our approach yields a family of lifted MAP inference algorithms.

Our approach is quite general and can be extended to arbitrary MLNs that have shared terms in a straightforward way: simply ground all the shared terms of the MLN to obtain an equivalent non-shared MLN. The only caveat is that if all terms in all atoms of the MLN

---

\(^1\)Number of possible groundings of an atom is the product of the domain sizes of the logical variables appearing in it.
are shared, our approach will have the same complexity as ground inference but never worse (e.g., in the transitive formula $\forall x, y, z \ R(x, y) \land R(y, z) \Rightarrow R(x, z)$, all terms of $R$ are shared). Note that this is a limitation of not only our approach but lifted inference in general: it is only useful when symmetries are present.

We experimentally evaluated our approach on three benchmark MLNs: WebKB and Information Extraction MLNs available from the Alchemy [Kok et al., 2006] web page and the Student MLN created by us. We used two state-of-the-art MAP inference algorithms within our approach: (i) Gurobi [Gurobi, 2016], which is an integer linear programming solver and (ii) MaxWalkSAT [Kautz et al., 1997] which is a popular local search solver. Our experiments clearly show that our approach is significantly better regarding the solution quality and scalability than ground inference. In particular, as we increase the number of objects in the MLN, our algorithms are an order of magnitude faster and better in terms of solution quality.

3.2 Lifted Formulation of MAP Inference

In this section, we show that we can reduce MAP inference in a sub-class of MLNs, which we call non-shared MLNs, to MAP inference over an equivalent propositional MLN (or a propositional Markov network) such that the number of propositional variables in the propositional MLN is equal to the number of first order atoms in the non-shared MLN. This is in contrast to ground MAP inference in which the number of propositional variables is equal to the number of ground atoms.

We begin by defining non-shared MLNs.

**Definition 6.** A normal MLN is called a **non-shared MLN** if each of its formulas is non-shared. A formula $f_i$ is non-shared if every logical variable appears at most once in the formula. In other words, in a non-shared MLN, no logical variable is shared between the atoms in a formula.
Figure 3.1. Weights of all assignments to ground atoms and (lifted) groups for the non-shared MLN: \[ R(x) \lor S(y), w_1 \]; \[ R(x), w_2 \]; and \[ S(y), w_3 \] with domains given by \( \Delta(x) = \Delta(y) = \{A, B\} \).

For example, \( R(x) \lor S(y) \) is a non-shared formula. However, \( R(x) \lor S(x) \) is not because \( x \) is shared.

### 3.2.1 Domain-Lifted MAP Inference over Non-Shared MLNs

We show that MAP inference over non-shared MLNs is domain liftable (Van den Broeck, 2011b), namely inference over it is polynomial in the domain size of the logical variables in the MLN. The key reason that non-shared MLNs are domain liftable is that they contain several worlds with the same probability. We can group together these equi-probable worlds and perform MAP inference by just iterating over the groups, selecting the group with the maximum probability. The following example illustrates this grouping.

**Example 7.** Consider the non-shared MLN containing three formulas: \( R(x) \lor S(y), w_1 \); \( R(x), w_2 \); and \( S(y), w_3 \). Let \( \Delta(x) = \Delta(y) = \{A, B\} \). Figure 3.1 gives a truth table showing all possible assignments to the ground atoms as well as their weights. Figure 3.1 also shows nine equi-probable groups for these assignments. It turns out that each group can
be represented by a pair \((i, j)\) where \(i\) and \(j\) are the number of true groundings of \(R\) and \(S\) respectively. Namely, \(i, j \in \{0, 1, 2\}\). Thus, to compute the MAP tuple, we only have to iterate over 9 groups while the ground (naive) MAP inference algorithm will iterate over 16 assignments. In general, the number of groups will be equal to \((|\Delta(x)| + 1)(|\Delta(y)| + 1)\) while the number of possible assignments to the ground atoms equals \(2^{\Delta(x)} + |\Delta(y)|\).

We can generalize the ideas presented in Example 7 using the following theorem:

**Theorem 2.** Given a non-shared MLN \(\mathcal{M}\), let \(\omega_1\) and \(\omega_2\) be two worlds such that for each atom \(R\) in the MLN, the number of true groundings of \(R\) in \(\omega_1\) is equal to the number of true groundings of \(R\) in \(\omega_2\). Then, \(\Pr_{\mathcal{M}}(\omega_1) = \Pr_{\mathcal{M}}(\omega_2)\).

**Proof.** We will prove this theorem by leveraging the generalized binomial rule (Jha et al., 2010). The generalized Binomial rule states that if an atom \(R\) is non-shared (which is a special case of singleton atoms), then the MLNs obtained by conditioning on the following subset of assignments to all groundings of \(R\) are equivalent: the number of true groundings of \(R\) is the same in all the assignments in the subset. Moreover, according to the rule, the following two conditions hold:

- if the MLN is non-shared then the new MLN is also non-shared
- the number of formulas involving \(R\) satisfied by each assignment in the subset is the same.

Let \(R_1, \ldots, R_n\) be the atoms in the MLN \(\mathcal{M}\). Let \(d_i\) be the domain size of \(R_i\). Let \(R_i = j_{i,1}\) and \(R_i = j_{i,2}\) where \(j_{i,k} \in 2^{d_i}\), \(k \in \{1, 2\}\) denote the assignment to all groundings of \(R_i\) in the worlds \(\omega_1\) and \(\omega_2\) respectively. Let us condition the atoms along the order \(R_1, \ldots, R_n\). By the generalized Binomial rule, the MLN obtained by conditioning on \(R_1 = j_{1,1}\), denoted by \(\mathcal{M}|R_1 = j_{1,1}\) is equivalent to the MLN \(\mathcal{M}|R_1 = j_{1,2}\) obtained by conditioning on \(R_1 = j_{1,2}\) (since the number of true groundings of \(R_i\) is the same in both the assignments). Let
\( w(\mathcal{M}|R_i = j_{i,k}), k \in \{1, 2\} \) denote the sum of the weights of clauses satisfied by conditioning on the assignment \( R_i = j_{i,k} \). By the generalized Binomial rule, \( w(\mathcal{M}|R_1 = j_{1,1}) = w(\mathcal{M}|R_1 = j_{1,2}) \). Moreover, since all atoms in \( \mathcal{M}|R_1 = j_{1,1} \) and \( \mathcal{M}|R_2 = j_{2,1} \) are non-shared, it follows that the MLNs obtained by further conditioning on \( R_2 = j_{2,1} \) is the same as the one obtained by conditioning on \( R_2 = j_{2,2} \). By iteratively (inductively) applying this argument, we have:

\[
\sum_{i=1}^{n} w(\mathcal{M}|R_i = j_{1,1}, \ldots, R_{i-1} = j_{i-1,1}) = \sum_{i=1}^{n} w(\mathcal{M}|R_i = j_{1,2}, \ldots, R_{i-1} = j_{i-1,2}) \tag{3.1}
\]

In other words, the two worlds \( \omega_1 \) and \( \omega_2 \) have the same weight. Therefore, \( \Pr_M(\omega_1) = \Pr_M(\omega_2) \).

Theorem 2 yields the following lifted inference algorithm. Let \( \{R_1, R_2, \ldots, R_n\} \) be the atoms in the non-shared MLN. Let \( d_i \) denote the domain size of \( R_i \) (the domain of an atom equals the cartesian product of the domains of its logical variables). By Theorem 2 all the ground assignments of the MLN can be grouped into assignments of the form \( \langle (R_i, a_i) | i \in \{1, \ldots, n\} \rangle \) where \( a_i \in \{0, \ldots, d_i\} \) and the assignment indicates \( a_i \) groundings of \( R_i \) are true. We will refer to \( (R_i, a_i) \) as a counting assignment [Milch et al. 2008]. The algorithm iterates over all tuples of the form: \( \langle (R_1, a_1), \ldots, (R_n, a_n) \rangle \), computes the weight of the tuple, and returns the tuple with the maximum weight as the MAP tuple. This lifted algorithm is clearly more efficient than its propositional counterpart. The search space over which the propositional algorithm operates is bounded by \( O(2^{\sum_{i=1}^{n} d_i}) \) where \( n \) is the number of atoms in the MLN. On the other hand, the search space of the lifted algorithm is bounded by \( O(\prod_{i=1}^{n} (d_i + 1)) \). Since the search space is bounded polynomially by the domain size of the logical variables, we have:

**Theorem 3.** MAP inference in non-shared MLNs is domain liftable.
Although this represents a significant improvement over propositional MAP algorithms, it turns out that we can further reduce the search space for a sub-class of non-shared MLNs, namely non-shared MLNs without self-joins. We will present this result next.

### 3.2.2 MAP Inference over non-shared MLNs without Self-Joins

We illustrate the main idea in our result using the following example.

**Example 8.** Consider the MLN used in Example 1. Let \( w_1 = -4, \ w_2 = 5 \) and \( w_3 = 3 \). Assume that the domain of \( x, y \) is \( \{A, B, C, D, E\} \). If we iterate through all possible counting assignments to \( R \) and \( S \) and plot the total weight of satisfied clauses as a function of counting assignment of \( R \) and \( S \) we get the plot in Figure 3.2. Figure 3.2 shows that the function in the plot has only four extreme points: \((0, 0), (0, 5), (5, 0) \) and \((5, 5)\). These extreme points correspond to all groundings of \( R \) and \( S \) as either being all true or all false. Since the MAP value can only lie on these extreme points, we only have to evaluate these extreme points for computing the MAP value. It turns out that the MAP tuple is \( ((R, 0), (S, 0)) \).

We observe in the previous example that all the ground atoms of the predicate \( R \) (and the predicate \( S \)) have the same truth value. We will refer to this kind of assignment (i.e., all ground atoms having the same truth value) as a **uniform assignment** (Apsel and Brafman, 2012). This observation, that the atoms have a uniform assignment in the MAP state, holds not only for this example but for any non-shared MLN without self-joins, and we will prove this formally next.

**Lemma 1.** The sum of weights of satisfied clauses for a non-shared MLN without self-join is a multilinear function on the counting assignment of its predicates.

---

\(^2\)We say that a formula has no self-joins if a predicate symbol appears at most once in the formula.
Figure 3.2. The total weight of satisfied clauses as a function of counting assignment of $R$ and $S$. The plane is for illustration purpose only.

Proof. Consider a non-shared MLN $\mathcal{M}$ that contains $m$ weighted clauses $\{(C_i; w_i)\}_{i=1}^m$. Let $V(C_i)$ represent the set of all the atoms in the clause $C_i$. Let $V^+(C_i)$ represent the set of atoms which appear as positive literals in $C_i$. Let $V^-(C_i)$ represent the set of atoms appearing as negative literals. Given an atom $R$, let $(R, v_R)$ denote its counting assignment. It can be easily shown that the number of groundings of $C_i$ that are unsatisfied by the counting assignment is given by,

$$\prod_{R \in V^+(C_i)} (D(R) - v_R) \prod_{R \in V^-(C_i)} v_R$$

where we abused the notation $D(\cdot)$ slightly such that $D(R)$ represents the number of possible groundings of $R$. Clearly, the total number of possible groundings of $C_i$ is equal to $\prod_{R \in C_i} (D(R))$. Therefore, the sum of weights of satisfied clauses for $\mathcal{M}$ is given by,

$$\sum_{C_i} w_i \left( \prod_{R \in C_i} (D(R)) \right) - \left( \prod_{R \in V^+(C_i)} (D(R) - v_R) \prod_{R \in V^-(C_i)} v_R \right)$$
Clearly eq. 3.2 is a multilinear function in $v_R$ since $v_R$ never appears more than once in the product term (if there are no self-joins in $M$). □

**Lemma 2.** Consider a multilinear function $\Phi(v)$ defined over a tuple of variables $v = (v_1, \ldots, v_n)$. Let each $v_j$ take values from the set $\{0, \ldots, d_{v_j}\}$. Then, at least one of the solutions $v^*$ to the optimization problem $\arg\max_v \Phi(v)$ is such that each $v^*_j$ lies at the extremes i.e. $\forall j, v^*_j = 0$ or $v^*_j = d_{v_j}$.

**Proof.** We will prove the theorem using induction over $n$, the number of variables over which the multilinear function is defined. Clearly, the theorem holds true for $n = 1$ since a linear function of one variable has its maxima at the extremes. Assume the theorem holds for any multilinear function defined over $n - 1$ variables. Consider the function $\Phi(v)$ over the variables $v = (v_1, \ldots, v_n)$. By re-arranging terms, we can write:

$$\max_v \Phi(v) = \max_{v \setminus v_n} (\max_{v_n} \Phi(v))$$

Since $\Phi(v)$ is a multilinear function, it can be seen as a linear function of $v_n$ (holding other variables as constant). Hence, the inner expression on the right side is optimized at an extreme value of $v_n$ ($v_n = 0$ or $v_n = d_{v_n}$). Let $\Phi_0(v \setminus v_n)$ and $\Phi_{d_{v_n}}(v \setminus v_n)$, respectively, be the two possible resulting functions by substituting the values of 0 and $d_{v_n}$, for $v_n$ in $\Phi(v)$. In both the cases, we get a new function which is multilinear over $n - 1$ variables. Using the induction hypothesis, its maxima will lie at the extreme values of $v_1, v_2, \ldots, v_{n-1}$. Hence, one of the maxima of the original function $\phi(v)$ will lie at the extreme values of $v_1, \ldots, v_n$. Hence, proved. □

Note that above lemma states that at least one of the maxima of a multilinear function will lie at its extremes. It is still possible that there are other maxima which do not lie at the extremes (for instance, think of a constant function). As long as we are interested in
finding one of them, above lemma can be put to use. Lemma 1 and Lemma 2 allow us to prove our second main result stated below.

**Theorem 4.** For a non-shared MLN without self-joins, in at least one of the MAP solutions, all predicates have uniform assignments.

We can use Theorem 4 to formulate the MAP problem as a weighted Max-SAT problem as formalized in the following corollary.

**Corollary 1.** The MAP inference in a non-shared MLN $\mathcal{M}$ that contains no self-joins can be converted to an equivalent propositional weighted Max-SAT problem with number of variables equal to the number of first order atoms in $\mathcal{M}$.

**Proof.** Given a non-shared MLN, $\mathcal{M}$ with $m$ weighted clauses $\{(C_i; w_i)\}_{i=1}^m$ that contains no self-joins, we first construct a weighted propositional knowledge base $\mathcal{S}$. We create $\mathcal{S} = \{(C'_i; w'_i)\}_{i=1}^m$ with $v$ propositional variables where every $v_k \in v$ corresponds to a distinct atom $R_{v_k}$ in $\mathcal{M}$. All atoms in $\mathcal{M}$ have a corresponding variable in $v$ and vice versa. The assignment true to variable $v_k$ corresponds to the positive uniform assignment to $R_{v_k}$, i.e. $- (R_{v_k}, D(R_{v_k}))$ and assigning false to variable $v_k$ corresponds to the negative uniform assignment to $R_{v_k}$, i.e. $- (R_{v_k}, 0)$. $C'_i$ is constructed by replacing each atom in $C_i$ by its corresponding variable in $v$. The weight of the clause is computed as $w'_i = D(C'_i) \times w_i$, where again by abusing notation we use $D(C_i)$ to denote the number of possible groundings of $C_i$. For uniform assignment, all groundings of each clause $C_i$ is either satisfied or none of them are satisfied. Since whenever $C_i$ is satisfied $C'_i$ is also satisfied and as the weight of $C'_i$ is $D(C_i) \times w_i$ the sum of weights of satisfied clauses for a complete assignment will be same in both $\mathcal{M}$ and $\mathcal{S}$. As theorem 4 proves that the MAP solution consist of uniform assignments, it follows from equation 2.6 that the MAP inference in $\mathcal{M}$ is equivalent to solving the weighted Max-SAT problem over $\mathcal{S}$. Hence the corollary follows. \qed
The result of corollary 1 allows us to use any weighted Max-SAT solver to compute the MAP solution of a non-shared MLN without self-joins. This observation also means that for such MLNs, the optimal solution is independent of the number of objects in the MLN which makes MAP inference especially efficient in these cases.

3.3 Extensions

In this section, we propose heuristics to make our approach more practical. These heuristics can be considered as pruning techniques, which allow us to greatly reduce the size of the knowledge base. Moreover these heuristics can be applied to any arbitrary MLN. These can give us orders of magnitude of speedup. We propose to use these heuristics as a preprocessing step to simplify the MLN.

3.3.1 Unit Propagation

Repeated use of unit propagation (Davis and Putnam, 1960) is one of the key component of highly effective propositional satisfiability testing solvers. The idea in unit propagation is to resolve all clauses with unit clauses, and continue to do this until convergence, i.e., no further unit resolutions are possible. Although this heuristic is very effective for SAT solvers, for Max-SAT, it is not sound. However, this rule can be used for hard unit clauses. We can lift this rule in a straight forward manner, by resolving the hard unit clauses with other clauses. This heuristic in conjunction with the pure literal heuristic can greatly reduce the size of the MLN.

3.3.2 Pure Literal Elimination

The pure literal elimination rule for SAT formulas (Davis and Putnam, 1960) when lifted to MAP inference for MLNs, removes (i) Clauses guaranteed to be satisfied for all groundings;
and (ii) Atoms guaranteed to be false for all groundings. The following proposition specifies the pure literal elimination rule for MLNs.

**Proposition 1.** Given an MLN \( \mathcal{M} \), if a predicate \( \mathcal{S} \) appears in \( k \) clauses \( \mathbf{C} = \{ C_i; w_i \}_{i=1}^{k} \), (i) if \( w_i \geq 0, \forall 1 \leq i \leq k \) and \( \mathcal{S} \) either always occurs as a positive literal or always occurs as a negative literal in \( \mathcal{M} \), every \( C_i \in \mathbf{C} \) can be removed from \( \mathcal{M} \); and (ii) if \( w_i < 0, \forall 1 < i \leq k \) and \( \mathcal{S} \) either always occurs as a positive literal or always occurs as a negative literal in \( \mathcal{M} \), then every occurrence of \( \mathcal{S} \) can be removed from \( \mathcal{M} \).

### 3.4 Experiments

For our experiments, we implemented two lifted MAP algorithms, (i) An anytime exact solver based on Integer Linear Programming (L-ILP); and (ii) An anytime approximate solver based on WalkSAT architecture (L-MWS).

For the ILP encoding we have used the Weighted Max-SAT encoding as discussed in Chapter 2. We solved L-ILP using a parallelized ILP solver called Gurobi \cite{Gurobi2016} and implemented L-MWS using MaxWalkSAT \cite{Selman1996}, a randomized local-search algorithm. We compared both our algorithms with MaxWalkSAT which is the MAP inference algorithm implemented within two state-of-the-art MLN systems, Alchemy (MWS) and Tuffy (TUFFY) \cite{Niu2011}. Since both these systems produce approximate results, we implemented an exact MAP inference algorithm using Gurobi (ILP). All three algorithms, MWS, TUFFY and ILP work on the propositional search space, i.e. they ground the entire MLN before performing MAP inference.

We used three MLNs to evaluate our system,

(i) A **Student** MLN having four formulas:

\[
\text{Teaches}(teacher, course) \land \text{Takes}(student, course) \rightarrow \text{JobOffers}(student, company);
\]

\[
\text{Teaches}(teacher, course);
\]
Takes(student,course); and

¬JobOffers(student,company).

(ii) WebKB MLN (Kok et al. 2006) from the Alchemy web page, consisting of three predicates and six formulas.

(iii) Citation Information-Extraction (IE) MLN (Kok et al. 2006) from the Alchemy web page, consisting of five predicates and fourteen formulas.

In order to compare the performance and scalability of our algorithms, we ran two sets of experiments illustrated in Figure 3.3 and Figure 3.4. Figure 3.3 plots the solution quality (total weight of false clauses) achieved by each algorithm for varying time-bounds. Figure 3.4 plots the relative-gap between the optimal solution and the solution output by each algorithm, for varying domain-sizes. We describe the results of both our experiments below. All our experiments were run on a quad-core CentOS machine with 8GB RAM.

3.4.1 Cost vs. Time

The results for the Student MLN are shown in Figure 3.3 (a)-(c). We see that the lifted algorithms L-ILP and L-MWS are the best performing algorithms for all domain-sizes. At higher domain-sizes (100 and 500), the propositional solvers ILP, MWS and TUFFY ran out of memory. The performance of L-MWS was similar to L-ILP for domain-size equal to 30. For domain-sizes of 100 and 500, L-MWS gradually converges towards the optimal solution, whereas L-ILP was able to exactly solve the problem in less than 10 seconds.

The results for WebKB are shown in Figure 3.3 (d)-(f). Again, we can see that the lifted algorithms L-ILP and L-MWS outperform the propositional algorithms and are much more scalable. For the larger domain-sizes (100 and 500), MWS and TUFFY run out of memory. For domain-size 30 and 100, the performance of both the lifted algorithms L-ILP and L-MWS is quite similar.
The results for IE are shown in Figure 3.3 (g)-(i). For domain-size 30, the performance of both the lifted algorithms L-ILP and L-MWS is quite similar. For the domain-sizes 100 and 500, L-ILP was able to find the optimal solution while L-MWS was far from optimal.

3.4.2 Accuracy vs. Domain-Size

Figure 3.4 illustrates the variation in accuracy for each algorithm as the domain-size increases. Here, we gave each algorithm a fixed time-bound of 500 seconds and measured the relative-gap between the optimal solution \(\text{opt}\) and the best cost given by the algorithm \(c\) using \(\frac{|\text{opt} - c|}{\text{opt}}\). We see that both L-MWS and L-ILP are quite accurate and scale to much larger domain-sizes. On the other hand, there is a noticeable drop in the accuracy of the propositional algorithms, MWS, TUFFY and ILP as we increase the domain-size. For larger domain-sizes, the propositional algorithms run out of memory.

In summary, our experiments show that our two lifted algorithms L-MWS and L-ILP are far more scalable and accurate than propositional approaches. Since the two approaches are fundamentally different, L-ILP is a complete anytime solver while L-MWS is an approximate solver, as expected they perform differently on the benchmarks, with L-ILP being the superior approach. However, the main virtue of our approach is that we could use any off-the-shelf solver that is purely propositional in nature to perform lifted inference. This allows us to scale to large domain-sizes without implementing a new lifted solver. We believe that this abstraction greatly simplifies the development of lifted algorithms by benefitting from the advances made in propositional algorithms.

3.5 Conclusion

In this chapter, we proposed a general approach for lifting MAP inference in Markov Logic Networks (MLNs). We identified cases in which we can reduce lifted MAP inference to inference over an equivalent propositional theory such that the number of propositional
Figure 3.3. Cost vs. Time: Cost of unsatisfied clauses (smaller is better) against time for benchmark MLNs for different domain sizes. Notation used to label each figure: MLN-domainsize(number of ground clauses in the MLN). The standard deviation is plotted as error bars. For (b),(c),(e),(f),(h) and (i), no results could be obtained for propositional algorithms since they ran out of memory.
variables is equal to the number of first order atoms in the MLN. We used this observation in a straight-forward manner: convert the MLN to an equivalent propositional theory and then apply any propositional algorithm to solve it. For our experiments, we used two propositional algorithms, a complete, anytime algorithm (Gurobi) based on Integer Linear Programming (ILP) and a local-search algorithm called MaxWalksat. Our experiments clearly demonstrate the scalability and promise of our approach.
CHAPTER 4

LIFTED MAP USING INTEGER POLYNOMIAL PROGRAMMING

4.1 Introduction

In the previous chapter, we have proposed a Lifted MAP inference algorithm based on the “lifting as pre-processing” paradigm. However, our previous approach failed to exploit many recent advances in Lifted Inference (such as the “decomposer rule”, (Gogate and Domingos, 2011; Jha et al., 2010)). In this chapter, we exploit the lifting as pre-processing paradigm further to propose a novel lifted MAP inference approach which is at least as powerful as probabilistic theorem proving (Gogate and Domingos, 2011) and subsumes our previous method by using it as just another lifted inference rule. The key idea in our approach is to reduce the lifted MAP inference (maximization) problem to an equivalent Integer Polynomial Program (IPP). Each variable in the IPP potentially refers to an assignment to a large number of ground atoms in the original MLN. Hence, the size of search space of the generated IPP can be significantly smaller than the ground Markov network.

Our algorithm to generate the IPP is based on the following three lifted inference operations which incrementally build the polynomial objective function and its associated constraints: (1) Lifted decomposition (Gogate and Domingos, 2011) finds sub-problems with identical structure and solves only one of them; (2) Lifted conditioning (Gogate and Domingos, 2011) replaces an atom with only one logical variable (singleton atom) by a variable in the integer polynomial program such that each of its values denotes the number of the

\footnote{In the best case scenario, the number of variables in the IPP is independent of the domain size of the variables in the MLN.}
true ground atoms of the singleton atom in a solution; and (3) Partial grounding is used to simplify the MLN further so that one of the above two operations can be applied.

To solve the IPP generated from the MLN, we convert it to an equivalent zero-one Integer Linear Program (ILP) using a classic conversion method outlined in (Watters 1967). A desirable characteristic of our reduction is that we can use any off-the-shelf ILP solver to get an exact or approximate solution to the original problem. We used a parallel ILP solver, Gurobi (Gurobi 2016) for this purpose. We compared our approach with Alchemy (Kok et al. 2008) and Tuffy (Niu et al. 2011), two state-of-the-art MLN systems that perform MAP inference by grounding the MLN, as well as with our previous lifted MAP inference approach (Sarkhel et al. 2014b). Our experiment on multiple benchmark MLNs shows that our method is superior to Alchemy, Tuffy, and our previous approach in terms of scalability and accuracy.

4.2 Probabilistic Theorem Proving Based MAP Inference Algorithm

We motivate our approach by presenting in Algorithm 2 the most basic algorithm for lifted MAP inference. Algorithm 2 extends the probabilistic theorem proving (PTP) algorithm of Gogate and Domingos (Gogate and Domingos 2011) to MAP inference and integrates it with our isolated singleton rule as mentioned in the previous chapter (Sarkhel et al. 2014b). It is obtained by replacing the summation operator in the conditioning step of PTP by the maximization operator (PTP computes the partition function). Note that throughout the chapter, we will present algorithms that compute the MAP value rather than the MAP
assignment; the assignment can be recovered by tracing back the path that yielded the MAP value. We describe the steps in Algorithm 2 next, starting with some required definitions.

**Algorithm 2: PTP-MAP(MLN M)**

1. **if** $M$ **is empty** **then**
   - return 0

2. **Simplify**($M$)

3. **if** $M$ **has disjoint MLNs** $M_1, \ldots, M_k$ **then**
   - return $\sum_{i=1}^{k} \text{PTP-MAP}(M_i)$

4. **if** $M$ **has a decomposer** $d$ **such that** $D(i \in d) > 1$ **then**
   - return PTP-MAP($M|d$)

5. **if** $M$ **has an isolated atom** $R$ **such that** $D(i_R) > 1$ **then**
   - return PTP-MAP($M\{1_R\}$)

6. **if** $M$ **has a singleton atom** $A$ **then**
   - return $\max_{i=0}^{D(i_A)} \text{PTP-MAP}(M|(A, i)) + w(A, i)$

7. Heuristically select an argument $i_R$

   return PTP-MAP($M|G(i_R)$)

Two arguments $i_R$ and $j_S$ are called unifiable if they share a logical variable in a MLN formula. Clearly, unifiable, if we consider it as a binary relation $U(i_R, j_S)$ is symmetric and reflexive. Let $\mathcal{U}$ be the transitive closure of $U$. Given an argument $i_S$, let Unify($i_S$) denote the equivalence class under $\mathcal{U}$.

**Simplification.** In the simplification step, we simplify the predicates possibly reducing their arity (cf. [Jha et al., 2010; Gogate and Domingos, 2011] for details). An example simplification step is the following: if no atoms of a predicate share logical variables with other atoms in the MLN then we can replace the predicate by a new predicate having just one argument; the domain size of the argument is the product of domain sizes of the individual arguments.
Example 9. Consider a normal MLN with two weighted formulas: \( R(x_1, y_1) \lor S(z_1, u_1), w_1 \) and \( R(x_2, y_2) \lor S(z_2, u_2) \lor T(z_2, v_2), w_2 \). We can simplify this MLN by replacing \( R \) by a predicate \( J \) having one argument such that \( D(1_J) = D(1_R) \times D(2_R) \). The new MLN has two formulas: \( J(x_1) \lor S(z_1, u_1), w_1 \) and \( J(x_2) \lor S(z_2, u_2) \lor T(z_2, v_2), w_2 \).

**Decomposition.** If an MLN can be decomposed into two or more disjoint MLNs sharing no first-order atom, then the MAP solution is just a sum over the MAP solutions of all the disjoint MLNs.

**Lifted Decomposition.** Main idea in lifted decomposition [Gogate and Domingos, 2011] is to identify identical but disconnected components in ground Markov network by looking for symmetries in the first-order representation. Since the disconnected components are identical, only one of them needs to be solved and the MAP value is the MAP value of one of the components times the number of components. One way of identifying identical disconnected components is by using a decomposer [Gogate and Domingos, 2011; Jha et al., 2010], defined below.

**Definition 7. [Decomposer]** Given a MLN \( M \) having \( m \) formulas denoted by \( f_1, \ldots, f_m \), \( d = \text{Unify}(i_R) \) where \( R \) is a predicate in \( M \), is called a decomposer iff the following conditions are satisfied: (i) for each predicate \( R \) in \( M \) there is exactly one argument \( i_R \) such that \( i_R \in d \); and (ii) in each formula \( f_i \), there exists a variable \( x \) such that \( x \) appears in all atoms of \( f_i \) and for each atom having predicate symbol \( R \) in \( f_i \), \( x \) appears at position \( i_R \in d \).

Denoted by \( M|d \) the MLN obtained from \( M \) by setting domain size of all elements \( i_R \) of \( d \) to one and updating weight of each formula that mentions \( R \) by multiplying it by \( D(i_R) \). We can prove that:

**Proposition 2.** Given a decomposer \( d \), the MAP value of \( M \) is equal to the MAP value of \( M|d \).
Example 10. Consider a normal MLN $M$ having two weighted formulas $R(x) \lor S(x), w_1$ and $R(y) \lor T(y), w_2$ where $D(1_R) = D(1_S) = D(1_T) = n$. Here, $d = \{1_R, 1_S, 1_T\}$ is a decomposer. The MLN $M|d$ is the MLN having the same two formulas as $M$ with weights updated to $nw_1$ and $nw_2$ respectively. Moreover, in the new MLN $D(1_R) = D(1_S) = D(1_T) = 1$.

Isolated Singleton Rule. In the previous chapter (Sarkhel et al., 2014b) proved that if the MLN $M$ has an isolated predicate $R$ such that all atoms of $R$ do not share any logical variables with other atoms, then one of the MAP solutions of $M$ has either all ground atoms of $R$ set to true or all of them set to false, namely, the solution lies at the extreme assignments to groundings of $R$. Since we simplify the MLN, all such predicates $R$ have only one argument, namely, they are singleton. Therefore, the following proposition is immediate:

Proposition 3. If $M$ has an isolated singleton predicate $R$, then the MAP value of $M$ equals the MAP value of $M|\{1_R\}$ (the notation $M|\{1_R\}$ is defined just after the definition of the decomposer).

Lifted Conditioning over Singletons. Performing a conditioning operation on a predicate means conditioning on all possible ground atoms of that predicate. Naively it can result in exponential number of alternate MLNs that need to be solved, one for each assignment to all groundings of the predicate. However if the predicate is singleton, we can group these assignments into equi-probable sets based on number of true groundings of the predicate (counting assignment) (Gogate and Domingos, 2011; Jha et al., 2010; Milch et al., 2008). In this case, we say that the lifted conditioning operator is applicable. For a singleton $A$, we denote the counting assignment as the ordered pair $(A, i)$ which the reader should interpret as exactly $i$ groundings of $A$ are true and the remaining are false.

We denote by $M|(A, i)$ the MLN obtained from $M$ as follows. For each element $j_R$ in $\text{Unify}(1_A)$ (in some order), we split the predicate $R$ into two predicates $R_1$ and $R_2$ such that $D(j_{R_1}) = i$ and $D(j_{R_2}) = D(1_A) - i$. We then rewrite all formulas using these new predicate
symbols. Assume that $A$ is split into two predicates $A_1$ and $A_2$ respectively with $D(1_{A_1}) = i$ and $D(1_{A_2}) = D(1_A) - i$. Then, we delete all formulas in which either $A_1$ appears positively or $A_2$ appears negatively (because they are satisfied). Next, we delete all literals of $A_1$ and $A_2$ from all formulas in the MLN. The weights of all formulas (which are not deleted) remain unchanged except those formulas in which atoms of $A_1$ or $A_2$ do not share logical variables with other atoms. The weight of each such formula $f$ with weight $w$ is changed to $w \times D(1_{A_1})$ if $A_1$ appears in the clause or to $w \times D(1_{A_2})$ if $A_2$ appears in the clause.

The weight $w(A, i)$ is calculated as follows. Let $F(A_1)$ and $F(A_2)$ denote the set of satisfied formulas (which are deleted) in which $A_1$ and $A_2$ participate in. We introduce some additional notation. Let $V(f)$ denote the set of logical variables in a formula $f$. Given a formula $f$, for each variable $y \in V(f)$, let $i_R(y)$ denote the position of the argument of a predicate $R$ such that $y$ appears at that position in an atom of $R$ in $f$. Then, $w(A, i)$ is given by:

$$w(A, i) = \sum_{k=1}^{2} \sum_{f_j \in F(A_k)} w_j \prod_{y \in V(f_j)} D(i_R(y))$$

We can show that:

**Proposition 4.** Given an MLN $M$ having singleton atom $A$, the MAP-value of $M$ equals $\max_{i=0}^{D(1_{A})} MAP-value(M|(A, i)) + w(A, i)$.

**Example 11.** Consider a normal MLN $M$ having two weighted formulas $R(x) \lor S(x), w_1$ and $R(y) \lor S(z), w_2$ with domain sizes $D(1_R) = D(1_S) = n$. The MLN $M|(R, i)$ is the MLN having three weighted formulas: $S_2(x_2), w_1$; $S_1(x_1), w_2(n - i)$ and $S_2(x_3), w_2(n - i)$ with domains $D(1_{S_1}) = i$ and $D(1_{S_2}) = n - i$. The weight $w(R, i) = iw_1 + niw_2$.

**Partial grounding.** In the absence of a decomposer, or when the singleton rule is not applicable, we will have to partially ground a predicate. For this, we heuristically select an argument $i_R$ to ground. Let $M|G(i_R)$ denote the MLN obtained from $M$ as follows. For
each argument $i_S \in \text{Unify}(i_R)$, we create $D(i_S)$ new predicates which have all arguments of $S$ except $i_S$. We then update all formulas with the new predicates. For example,

**Example 12.** Consider a MLN with two formulas: $R(x, y) \vee S(y, z), w_1$ and $S(a, b) \vee T(a, c), w_2$. Let $D(2_R) = 2$. After grounding $2_R$, we get an MLN having four formulas: $R_1(x_1) \vee S_1(z_1), w_1$, $R_2(x_2) \vee S_2(z_2), w_1$, $S_1(b_1) \vee T_1(c_1), w_2$ and $S_2(b_2) \vee T_2(c_2), w_2$.

Since partial grounding will create many new clauses, we will try to use this operator as sparingly as possible. The following theorem is immediate from (Gogate and Domingos, 2011; Sarkhel et al., 2014b) and the discussion above.

**Theorem 5.** PTP-MAP$(M)$ computes the MAP value of $M$.

4.2.1 Integer Polynomial Programming formulation for Lifted MAP

Before explaining our Integer Polynomial Programming based approach let’s first define an Integer Polynomial Programming (IPP) problem formally:

Maximize $f(x_1, x_2, ..., x_n)$

Subject to $g_j(x_1, x_2, ..., x_n) \geq 0$ ($j = 1, 2, ..., m$)

where each $x_i$ takes finite integer values, and the objective function $f(x_1, x_2, ..., x_n)$, and each of the constraints $g_j(x_1, x_2, ..., x_n)$ are polynomials on $x_1, x_2, ..., x_n$. We will compactly represent an integer polynomial programming problem (IPP) as an ordered triple $\mathcal{I} = \langle f, G, X \rangle$, where $X = \{x_1, x_2, ..., x_n\}$, and $G = \{g_1, g_2, ..., g_m\}$.

PTP-MAP performs an exhaustive search over all possible lifted assignments in order to find the optimal MAP value. It can be very slow without proper pruning, and that is why branch-and-bound algorithms are widely used for many similar optimization tasks. The branch-and-bound algorithm maintains a global best solution found so far, as a lower
bound. If the estimated upper bound of a node is not better than the lower bound, the node is pruned and the search continues with other branches. However instead of developing a lifted MAP specific upper bound heuristic to improve Algorithm 2, we propose to encode the lifted search problem as an Integer Polynomial Programming (IPP) problem. This way we can use existing off-the-shelf advanced machinery, which includes pruning techniques, search heuristics, caching, problem decomposition and upper bounding techniques, to solve the IPP.

### Algorithm 3: SMLN-2-IPP(SMLN S)

```plaintext
if S is empty then
    return ⟨0, ∅, ∅⟩

Simplify(S)

if S has disjoint SMLNs then
    for disjoint SMLNs $S_i ... S_k$ in S do
        ⟨$f_i, G_i, X_i$⟩ = SMLN-2-IPP($S_i$)
    return ⟨$\sum_{i=1}^{k} f_i, \cup_{i=1}^{k} G_i, \cup_{i=1}^{k} X_i$⟩

if S has a decomposer d then
    return SMLN-2-IPP(S|d)

if S has a isolated singleton $R$ then
    return SMLN-2-IPP(S|{$i_R$})

if S has a singleton atom $A$ then
    Introduce an IPP variable ‘$i$’
    Form a constraint $g$ as ‘$(0 \leq i \leq D(1_A))$’
    ⟨$f, G, X$⟩ = SMLN-2-IPP(S|{$A, i$})
    return ⟨$f + w(A, i), G \cup \{g\}, X \cup \{i\}$⟩

Heuristically select an argument $i_R$

return SMLN-2-IPP(S|{$G(i_R)$})
```

At a high level, our encoding algorithm runs PTP-MAP schematically, performing all steps in PTP-MAP except the search or conditioning step. Before we present our algorithm,
we define schematic MLNs (SMLNs) – a basic structure on which our algorithm operates. SMLNs are normal MLNs with two differences: (1) weights attached to formulas are polynomials instead of constants and (2) Domain sizes of arguments are linear expressions instead of constants.

Algorithm 3 presents our approach to encode lifted MAP problem as an IPP problem. It mirrors Algorithm 2, with only difference being at the lifted conditioning step. Specifically, in lifted conditioning step, instead of going over all possible branches corresponding to all possible counting assignments, the algorithm uses a representative branch which has a variable associated for the corresponding counting assignment. All update steps described in the previous section remain unchanged with the caveat that in $S|(A, i)$, $i$ is symbolic (an integer variable). At termination, Algorithm 3 yields an IPP. Following theorem is immediate from the correctness of Algorithm 2.

**Theorem 6.** Given an MLN $M$ and its associated schematic MLN $S$, the optimum solution to the Integer Polynomial Programming problem returned by SMLN-2-IPP($S$) is the MAP solution of $M$.

**Proof.** The only difference between Algorithm 3 and Algorithm 2 is the conditioning step. At this step we introduce a variable to condition on. The domain of the variable is added as a constraint to the IPP and the satisfied weight is computed “symbolically”. However, when we solve the IPP we systematically look for assignments that maximizes this weight (which is the same step as the conditioning step of Algorithm 2), hence, solving the IPP yields the same solution as given by Algorithm 2. Since Algorithm 2 computes the MAP value correctly, the theorem follows.

In the next three examples, we show the IPP output by Algorithm 3 on some example MLNs.
Example 13. Consider an MLN having one weighted formula: \( R(x) \lor S(x), w_1 \) such that \( D(1_R) = D(1_S) = n \). Here, \( d = \{1_R, 1_S\} \) is a decomposer. By applying the decomposer rule, weight of the formula becomes \( nw_1 \) and domain size is set to 1. After conditioning on \( R \) objective function obtained is \( nw_1 r \) and the formula changes to \( S(x), nw_1(1 - r) \). After conditioning on \( S \), the IPP obtained has objective function \( nw_1 r + nw_1(1 - r)s \) and two constraints: \( 0 \leq r \leq 1 \) and \( 0 \leq s \leq 1 \).

Example 14. Consider an MLN having one weighted formula: \( R(x) \lor S(y), w_1 \) such that \( D(1_R) = nx \) and \( D(1_S) = ny \). Here \( R \) and \( S \) are isolated, and therefore by applying the isolated singleton rule weight of the formula becomes \( nxnyw_1 \). This is similar to the previous example; only weight of the formula is different. Therefore, substituting this new weight, IPP output by Algorithm 3 will have objective function \( nxnyw_1 r + nxnyw_1(1 - r)s \) and two constraints \( 0 \leq r \leq 1 \) and \( 0 \leq s \leq 1 \).

Example 15. Consider an MLN having two weighted formulas: \( R(x) \lor S(x), w_1 \) and \( R(z) \lor S(y), w_2 \) such that \( D(1_R) = D(1_S) = n \). On this MLN, the IPP output by Algorithm 3 has the objective function \( rw_1 + r^2w_2 + rw_2(n - r) + s_2w_1(n - r) + s_2w_2(n - r)^2 + s_1w_2(n - r)r \) and constraints \( 0 \leq r \leq n \), \( 0 \leq s_1 \leq 1 \) and \( 0 \leq s_2 \leq 1 \). The operations that will be applied in order are: lifted conditioning on \( R \) creating two new predicates \( S_1 \) and \( S_2 \); decomposer on \( 1_{S_1} \); decomposer on \( 1_{S_2} \); and then lifted conditioning on \( S_1 \) and \( S_2 \) respectively.

4.2.2 Solving Integer Polynomial Programming Problem

Although we can directly solve the IPP using any off-the-shelf mathematical optimization software, IPP solvers are not as mature as Integer Linear programming(ILP) solvers. Therefore, for efficiency reasons, we propose to convert the IPP to an ILP using the classic method outlined in (Watters, 1967) (we skip the details for lack of space). The method first converts the IPP to a zero-one Polynomial Programming problem and then subsequently linearizes
it by adding additional variables and constraints for each higher degree terms. Once the problem is converted to an ILP problem we can use any standard ILP solver to solve it. Next, we state a key property about this conversion in the following theorem.

**Theorem 7.** The search space for solving the IPP obtained from Algorithm 3 by using the conversion described in (Watters, 1967) is polynomial in the max-range of the variables.

**Proof.** Let \( n \) be number of variables of the IPP problem, where each of the variables has range from 0 to \((d-1)\) (i.e., for each variable \( 0 \leq v_i \leq d-1 \)). As we first convert everything to binary, the zero-one Polynomial Programming problem will have \( O(n \log_2 d) \) variables. If the highest degree of a term in the IPP problem is \( k \), we will need to introduce \( O(\log_2 d^k) \) binary variables (as multiplying \( k \) variables, each bounded by \( d \), will result in terms bounded by \( d^k \)) to linearize it. Since search space of an ILP is exponential in number of variables, search space for solving the IPP problem is:

\[
O(2^{(n \log_2 d + \log_2 d^k)}) = O(2^{n \log_2 d})O(2^{k \log_2 d}) = O(d^n)O(d^k) = O(d^{n+k})
\]

We conclude this section by summarizing the power of our new approach:

**Theorem 8.** The search space of the IPP returned by Algorithm 3 is smaller than or equal to the search space of the Integer Linear Program (ILP) obtained using the algorithm proposed in Chapter 3, which in turn is smaller than the size of the search space associated with the ground Markov network.

**Proof.** Since, one of the sub-steps of Algorithm 2 is the method described in Chapter 3, the search space of Algorithm 2 is at least as small as Chapter 3’s approach. Now as Algorithm 3 is a symbolic encoding of Algorithm 2, the search space for solving the encoded IPP is exactly the same as for Algorithm 2. Hence, the theorem follows.
4.3 Experiments

We used a parallelized ILP solver called Gurobi ([Gurobi](http://www.gurobi.com) 2016) to solve ILPs generated by our algorithm as well as by other competing algorithms used in our experimental study. We compared performance of our new lifted algorithm (which we call IPP) with four other algorithms from literature: Alchemy (ALY) ([Kok et al.](http://www.cs.columbia.edu/~kok) 2008), Tuffy([TUFFY](http://www.cs.columbia.edu/~kok) [Niu et al.](http://www.cs.columbia.edu/~kok) 2011), ground inference based on ILP (ILP), and lifted MAP (LMAP) algorithm presented in Chapter 3 ([Sarkhel et al.](http://www.cs.columbia.edu/~kok) 2014b). Alchemy and Tuffy are two state-of-the-art open source software for learning and inference in MLNs. Both of them first ground the MLN and then use an approximate solver, MaxWalkSAT ([Selman et al.](http://www.cs.columbia.edu/~kok) 1996) to compute MAP solution. Unlike Alchemy, Tuffy uses clever Database tricks to speed up computation. ILP is obtained by converting MAP problem over ground Markov network to an ILP. LMAP also converts the MAP problem to ILP, however its ILP encoding can be much more compact than ones used by ground inference methods because it processes “non-shared atoms” in a lifted manner (see Chapter 3 for details). We used following three MLNs to evaluate our algorithm:

(i) An MLN which we call **Student** that consists of following four formulas,

\[
\text{Teaches}(teacher, course) \land \text{Takes}(student, course) \rightarrow \text{JobOffers}(student, company);
\]

\[
\text{Teaches}(teacher, course);
\]

\[
\text{Takes}(student, course);
\]

\[
\neg \text{JobOffers}(student, company)
\]

(ii) An MLN which we call **Relationship** that consists of following four formulas,

\[
\text{Loves}(person1, person2) \land \text{Friends}(person2, person3) \rightarrow \text{Hates}(person1, person3);
\]

\[
\text{Loves}(person1, person2);
\]

\[
\text{Friends}(person1, person2);
\]

\[
\neg \text{Hates}(person1, person2);
\]
Citation Information-Extraction (IE) MLN \cite{Kok2008} from the Alchemy web page, consisting of five predicates and fourteen formulas.

To compare performance and scalability, we ran each algorithm on aforementioned MLNs for varying time-bounds and recorded solution quality (i.e., the total weight of false clauses) achieved by each. All our experiments were run on a third generation i7 quad-core machine having 8GB RAM.

For Student MLNs, results are shown in Fig 4.1(a)-(c). On the MLN having 161K clauses, ILP, LMAP and IPP converge quickly to the optimal answer while TUFFY converges faster than ALY. For the MLN with 812K clauses, LMAP and IPP converge faster than ILP and TUFFY. ALY is unable to handle this large Markov network and runs out of memory. For the MLN with 8.1B clauses, only LMAP and IPP are able to produce a solution with IPP converging much faster than LMAP. On this large MLN, all three ground inference algorithms, ILP, ALY and TUFFY ran out of memory.

Results for Relationship MLNs are shown in Fig 4.1(d)-(f) and are similar to Student MLNs. On MLNs with 9.2K and 29.7K clauses ILP, LMAP and IPP converge faster than TUFFY and ALY, while TUFFY converges faster than ALY. On the largest MLN having 1M clauses only LMAP, ILP and IPP are able to produce a solution with IPP converging much faster than other two.

For IE MLN results are shown in Fig 4.1(g)-(i) which show a similar picture with IPP outperforming other algorithms as we increase number of objects in the domain. In fact on the largest IE MLN having 15.6B clauses only IPP is able to output a solution while other approaches ran out of memory.

In summary, as expected, IPP and LMAP, two lifted approaches are more accurate and scalable than three propositional inference approaches: ILP, TUFFY and ALY. IPP not only scales much better but also converges much faster than LMAP, clearly demonstrating the power of our new approach.
Figure 4.1. Cost vs. Time: Cost of unsatisfied clauses (smaller is better) vs. time for different domain sizes. Notation used to label each figure: MLN(numvariables, numclauses, numevidences). Note: three quantities reported are for ground Markov network associated with the MLN. Standard deviation is plotted as error bars.

4.4 Conclusion

In this chapter we presented a general approach for lifted MAP inference in Markov logic networks (MLNs). The main idea in our approach is to encode MAP problem as an Integer Polynomial Program (IPP) by schematically applying three lifted inference steps to the MLN: lifted decomposition, lifted conditioning, and partial grounding. To solve the IPP, we propose to convert it to an Integer Linear Program (ILP) using the classic method outlined...
in (Watters 1967). The virtue of our approach is that the resulting ILP can be much smaller than the one obtained from ground Markov network. Moreover, our approach subsumes the method outlined in the previous chapter and is at least as powerful as probabilistic theorem proving (Gogate and Domingos 2011). Perhaps, the key advantage of our approach is that it runs lifted inference as a pre-processing step, reducing the size of the theory and then applies advanced propositional inference algorithms to this theory without any modifications. Thus, we do not have to lift explicitly (and efficiently implement) decades worth of research and advances in propositional inference algorithms, treating them as a black-box.
5.1 Introduction

In the previous two chapters, we have discussed two exact lifted MAP inference approaches. These two algorithms like many other lifted MAP inference algorithms (Apsel and Brafman, 2012; Bui et al., 2013; Hadji and Kersting, 2013; Noessner et al., 2013; Mladenov et al., 2014; Mittal et al., 2014; Apsel et al., 2014) identify symmetries in the first-order specification using lifted inference rules, and then use these symmetries to infer simultaneously over multiple symmetric objects. Unfortunately, in a vast majority of cases, the inference rules are unable to identify several useful symmetries (the rules are sound but not complete), either because the symmetries are approximate or because the symmetries are domain-specific and do not belong to a known type. In such cases, lifted inference algorithms partially ground some atoms in the MLN and search for a solution in this much larger partially propositionalized space.

In this chapter, we propose the following straight-forward yet principled approach for solving this partial grounding problem (Van den Broeck and Darwiche, 2013; Venugopal and Gogate, 2014): partition the ground atoms into groups and force the inference algorithm to treat all atoms in each group as indistinguishable (symmetric). For example, consider a first-order atom R(x) and assume that x can be instantiated to the following set of constants: \{1, 2, 3, 4, 5\}. If the atom possesses the so-called non-shared or single-occurrence symmetry (Sarkhel et al., 2014b; Mittal et al., 2014), then the lifted inference algorithm will search over only two assignments: all five groundings of R(x) are either all true or all false, to find the MAP solution. When no identifiable symmetries exist, the lifted algorithm will
inefficiently search over all possible 32 truth assignments to the 5 ground atoms and will be equivalent in terms of (worst-case) complexity to a propositional algorithm. In our approach, we would partition the domain, say as \{\{1, 3\}, \{2, 4, 5\}\}, and search over only the following 4 assignments: all groundings in each part can be either all true or all false. Thus, if we are lucky and the MAP solution is one of the 4 assignments, our approach will yield significant reductions in complexity even though no identifiable symmetries exist in the problem.

Our approach is quite general and includes the fully lifted and fully propositional approaches as special cases. For instance, setting the partition size \(k\) to 1 and \(n\) respectively where \(n\) is the number of constants will yield the same solution as the one output by the fully lifted and fully propositional approach. Setting \(k\) to values other than 1 and \(n\) yields a family of inference schemes that systematically explores the regime between these two extremes. Moreover, by regulating the size \(k\) of each partition, we can control the size of the ground theory, and thus space and time complexity of our algorithm.

We theoretically verify some of the properties of our approach and improve upon our basic idea in several ways. First, we prove that our proposed approach yields a consistent assignment that is a lower-bound on the MAP value. Second, we demonstrate how to improve the lower bound and thus the quality of the MAP solution by systematically refining the partitions. Third, we show how to improve the complexity of our refinement procedure further by exploiting the exchangeability property of successive refinements. Specifically, we show that the exchangeable refinements can be arranged on a lattice, which can then be searched via a heuristic search procedure to yield an efficient any-time, any-space algorithm for MAP inference. Finally, we demonstrate experimentally that our method is highly scalable and yields close to optimal solutions in a fraction of the time as compared to existing approaches. In particular, our results show that for even small values of \(k\) (\(k\) bounds the partition size), our algorithm yields close to optimal MAP solutions, clearly demonstrating the power of our approach.
5.2 Scaling up the Partial Grounding Step using Set Partitioning

Before presenting the key findings, we will describe some required definitions.

**Partition of a Set.** A collection of sets $\mathcal{C}$ is a partition of a set $X$ if and only if each set in $\mathcal{C}$ is nonempty, pairwise disjoint and the union of all sets equals $X$. The sets in $\mathcal{C}$ are called the *cells* or *parts* of the partition. If two elements, $a, b$, of the set appear in a same cell of a partition $\rho$ we denote them by the operator ‘$\sim_\rho$’, i.e., $a \sim_\rho b$. A partition $\alpha$ of a set $X$ is a *refinement* of a partition $\rho$ of $X$ if every element of $\alpha$ is a subset of some element of $\rho$. Informally, this means that $\alpha$ is a further fragmentation of $\rho$. We say that $\alpha$ is finer than $\rho$ (or $\rho$ is coarser than $\alpha$) and denote it as $\alpha \prec \rho$. We will also use the notation $\alpha \preceq \rho$ to denote that either $\alpha$ is finer than $\rho$, or $\alpha$ is the same as $\rho$. For example, let $\rho = \{\{1, 2\}, \{3\}\}$ be a partition of the set $X = \{1, 2, 3\}$ containing two cells $\{1, 2\}$ and $\{3\}$ and let $\alpha = \{\{1\}, \{2\}, \{3\}\}$ be another partition of $X$, then $\alpha$ is a refinement $\rho$, namely, $\alpha < \rho$.

**Domain Equivalence Class.** Let $i^R$ denote the $i$-th argument of predicate $R$. Given an MLN, two arguments $i^R$ and $j^S$ of its predicates $R$ and $S$ respectively are called *unifiable* if they share a logical variable in an MLN formula. Being symmetric and transitive, the unifiable relation splits the arguments of all the predicates into a set of *domain equivalence classes*.

**Example 16.** Consider a normal MLN $M$ having two weighted formulas $(R(x) \lor S(x, y), w_1)$ and $(R(z) \lor T(z), w_2)$. Here, we have two sets of domain equivalence classes $\{1^R, 1^S, 1^T\}$ and $\{2^S\}$.

A typical exact lifted MAP algorithm (such as the one presented in the previous chapter, Algorithm 2) has a few recursive steps and returns the optimal MAP value. The first two lines are the base case and the simplification step, in which the MLN is simplified by deleting redundant formulas, rewriting predicates by removing constants (so that lifted conditioning
can be applied) and assigning values to ground atoms whose values can be inferred using assignments made so far. The second step is the propositional decomposition step in which the algorithm recurses over disjoint MLNs (if any) and returns their sum. In the lifted decomposition step, the algorithm finds a domain equivalence class such that in the MAP solution all ground atoms of tare either all true or all false. To find such a class, rules given in (Jha et al., 2010; Sarkhel et al., 2014b; Mittal et al., 2014) can be used. In the lifted conditioning step, if there is an atom having just one argument (singleton atom), then the algorithm partitions the possible truth assignments to groundings of \( A \) such that, in each part all truth assignments have the same number of true atoms. In the algorithm, \( M|_{(A, i)} \) denotes the MLN obtained by setting \( i \) groundings of \( A \) to true and the remaining to false. \( w(A, i) \) is the total weight of ground formulas satisfied by the assignment. The final step in PTP-MAP is the partial grounding step and is executed only when the algorithm is unable to apply lifted inference rules. In this step, the algorithm heuristically selects a domain equivalence class and grounds it completely. For example,

**Example 17.** Consider an MLN with two formulas: \( R(x, y) \lor S(y, z), w_1 \) and \( S(a, b) \lor T(a, c), w_2 \). Let \( D(2_R) = 2 \). After grounding the equivalence class \( \{2_R, 1_g, 1_T\} \), we get an MLN having four formulas: \( (R(x_1, 1) \lor S(1, z_1), w_1) \), \( (R(x_2, 2) \lor S(1, z_2), w_1) \), \( (S(1, b_1) \lor T(1, c_1), w_2) \) and \( (S(2, b_2) \lor T(2, c_2), w_2) \)

1\(^1\)The constants can be removed by renaming the predicates yielding a normal MLN. For example, we can rename \( R(x_1, 1) \) as \( R_1(x_1) \). This renaming occurs in the simplification step.
Algorithm 4: Constrained-Ground (MLN \( M \), Size \( k \), Domain Equivalence Class \( U \))

\[ M' = M \]

Create a partition \( \pi \) of size \( k \) of \( \Delta(i_R) \) where \( i_R \in U \)

**foreach** predicate \( R \) such that \( \exists i_R \in U \) **do**

**foreach** cell \( \pi_j \) of \( \pi \) **do**

Add all possible hard formulas of the form \( R(x_1, \ldots, x_r) \Leftrightarrow R(y_1, \ldots, y_r) \)

such that \( x_i = y_i \) if \( i_R \not\in U \) and \( x_i = X_a, y_i = X_b \) if \( i_R \in U \) where \( X_a, X_b \in \pi_j \).

**return** \( M' \)

Partial grounding often yields a much bigger MLN than the original MLN and is the chief reason for the inefficiency and poor scalability of Algorithm LMAP. To address this problem, we propose a novel approach to speed up inference by adding additional constraints to the existing lifted MAP formulation. Our idea is as follows: reduce the number of ground atoms by partitioning them and treating all atoms in each part as indistinguishable. Thus, instead of introducing \( O(tn) \) new ground atoms where \( t \) is the cardinality of the domain equivalence class and \( n \) is the number of constants, our approach will only introduce \( O(tk) \) ground atoms where \( k << n \).

Our new, approximate partial grounding method (which will replace the partial grounding step in Algorithm 2) is formally described in Algorithm 4. The algorithm takes as input an MLN \( M \), an integer \( k > 0 \) and a domain equivalence class \( U \) as input and outputs a new MLN \( M' \). The algorithm first partitions the domain of the class \( U \) into \( k \) cells, yielding a partition \( \pi \). Then, for each cell \( \pi_j \) of \( \pi \) and each predicate \( R \) such that one or more of its arguments is in \( U \), the algorithm adds all possible constraints of the form \( R(x_1, \ldots, x_r) \Leftrightarrow R(y_1, \ldots, y_r) \)

such that for each \( i \): (1) we add the equality constraint between the logical variables \( x_i \) and \( y_i \) if the \( i \)-th argument of the predicate is not in \( U \) and (1) set \( x_i = X_a \) and \( y_i = X_b \) if \( i \)-th
argument of $R$ is in $U$ where $X_a, X_b \in \pi_j$. Since adding constraints restricts feasible solutions to the optimization problem, it is easy to show that:

**Proposition 5.** Let $M' =$ \text{Constrain-Ground}(M, k), where $M$ is an MLN and $k > 0$ is an integer, be the MLN used in the partial grounding step of Algorithm 2 (instead of the partial grounding step described in the algorithm). Then, the MAP value returned by the modified algorithm will be smaller than or equal to the one returned by Algorithm 2.

The following example demonstrates how Algorithm 4 constructs a new MLN.

**Example 18.** Consider the MLN in Example 17. Let $\{\{1, D_{2,R}\}\}$ be a 1-partition of the domain of $U$. Then, after applying Algorithm 4 the new MLN will have the following three hard formulas in addition to the formulas given in Example 17: (1) $R(x_3, 1) \Leftrightarrow R(x_3, 2)$, (2) $S(1, x_4) \Leftrightarrow S(2, x_4)$ and (3) $T(1, x_5) \Leftrightarrow T(2, x_5)$.

Although, adding constraints reduces the search space of the MAP problem, Algorithm 4 still needs to ground the MLN. This can be time consuming. Alternatively, we can group indistinguishable atoms together without grounding the MLN using the following definition:

**Definition 8.** Let $U$ be a domain equivalence class and let $\pi$ be its partition. Two ground atoms $R(x_1, \ldots, x_r)$ and $R(y_1, \ldots, y_r)$ of a predicate $R$ such that $\exists_i R \in U$ are equivalent if $x_i = y_i$ if $i \notin U$ and $x_i = X_a, y_i = X_b$ if $i \in U$ where $X_a, X_b \in \pi_j$. We denote this by $R(x_1, \ldots, x_r) \perp_{\pi_1} R(y_1, \ldots, y_r)$.

Notice that the relation $\perp_{\pi}$ is symmetric and reflexive. Thus, we can group all the ground atoms corresponding to the transitive closure of this relation, yielding a “meta ground atom” such that if the meta atom is assigned to true (false), all the ground atoms in the transitive closure will be true (false). This yields the \text{partition-ground} algorithm described as Algorithm 5. The algorithm starts by creating a $k$ partition of the domain of $U$. It then updates the domain of $U$ so that it only contains $k$ values, grounds all arguments of predicates
that are in the set $U$ and updates the formula weights appropriately. The formula weights should be updated because, when the domain is compressed, several ground formulas are replaced by just one ground formula. Intuitively, if $t$ (partially) ground formulas having weight $w$ are replaced by one (partially) ground formula $(f, w')$ then $w'$ should be equal to $wt$. The two for loops in Algorithm 5 accomplish this. We can show that:

**Algorithm 5: Partition-Ground** (MLN $M$, Size $k$, Domain Equivalence class $U$)

$M' = M$

Create a partition $\pi$ of size $k$ of $\Delta(i_R)$ where $i_R \in U$

Update the domain $\Delta(i_R)$ to $\{1, \ldots, k\}$ in $M'$

Ground all predicates $R$ such that $i_R \in U$

foreach formula $(f', w')$ in $M'$ such that $f$ contains an atom of $R$ where $i_R \in U$ do

Let $f$ be the formula in $M$ from which $f'$ was derived

foreach logical variable in $f$ that was substituted by the $j$-th value in $\Delta(i_R)$ to yield $f'$ do

$w' = w' \times |\pi_j|$ where $\pi_j$ is the $j$-th cell of $\pi$

return $M'$

**Proposition 6.** The MAP value output by replacing the partial grounding step in Algorithm 3 with Algorithm Partition-Ground, is the same as the one output by replacing the the partial grounding step in Algorithm 3 with Algorithm Constrained-Ground.

The key advantage using Algorithm Partition-Ground is that the lifted algorithm (LMAP) will have much smaller space complexity than the one using Algorithm Constrained-Ground. Specifically, unlike the latter, which yields $O(n|U|)$ ground atoms (assuming each predicate has only one argument in $U$) where $n$ is the number of constants in the domain of $U$, the former generates only $O(k|U|)$ ground atoms, where $k << n$.

The following example illustrates how algorithm partition-ground constructs a new MLN.

**Example 19.** Consider an MLN $M$, with two formulas: $(R(x, y) \lor S(y, z), w_1)$ and $(S(a, b) \lor T(a, c), w_2)$. Let $\Delta(2_R) = 3$ and $\pi = \{\{1, 2\}, \{3\}\} = \{\nu_1, \nu_2\}$. After grounding $2_R$ with
respect to \( \pi \), we get an MLN, \( M' \), having four formulas: 
\((R_{\nu_1}(x_1) \lor S_{\nu_1}(z_1), 2w_1), (R_{\nu_2}(x_2) \lor S_{\nu_2}(z_2), w_1), (S_{\nu_1}(b_1) \lor T_{\nu_1}(c_1), 2w_2) \) and \((S_{\nu_2}(b_2) \lor T_{\nu_2}(c_2), w_2) \). The total weight of grounding in \( M \) is 
\((3w_1\Delta(1_R)\Delta(2_S) + 3w_2\Delta(2_T)\Delta(2_S)) \) which is the same as in \( M' \).

The following example illustrates how the algorithm constructs a new MLN in presence of self-joins.

**Example 20.** Consider an MLN, \( M \), with the single formula: 
\( \neg R(x, y) \lor R(y, x), w \). Let \( \Delta(1_R) = \Delta(2_R) = 3 \) and \( \pi = \{\{1, 2\}, \{3\}\} = \{\nu_1, \nu_2\} \). After grounding \( 1_R \) (and also on \( \Delta(2_R) \), as they belong to the same domain equivalence class) with respect to \( \pi \), we get an MLN, \( M' \), having following four formulas: 
\((R_{\nu_1,\nu_1} \lor R_{\nu_1,\nu_1}, 4w), (R_{\nu_1,\nu_2} \lor R_{\nu_2,\nu_1}, 2w), (R_{\nu_2,\nu_1} \lor R_{\nu_1,\nu_2}, 2w) \) and 
\((R_{\nu_2,\nu_2} \lor R_{\nu_2,\nu_2}, w) \).

### 5.2.1 Generalizing the Partition Grounding Approach

Algorithm Partition-Ground allows us to group the equivalent atoms with respect to a partition and has much smaller space complexity and time complexity than the partial grounding strategy described in Algorithm 2. However, it yields a lower bound on the MAP value. In this section, we show how to improve the lower bound using refinements of the partition. The basis of our generalization is the following theorem:

**Theorem 9.** Given two partitions \( \pi \) and \( \phi \) of \( U \) such that \( \phi \preceq \pi \), the MAP value of the partially ground MLN with respect to \( \phi \) is less than or equal to the MAP value of the partially ground MLN with respect to \( \pi \).

**Proof.** Since the partition \( \phi \) is a finer refinement of \( \pi \), any candidate MAP assignment corresponding to the MLN obtained via \( \phi \) already includes all the candidate assignments corresponding to the MLN obtained via \( \pi \), and since the MAP value of both of these MLNs are a lower bound of the original MAP value, the theorem follows. \( \square \)
We can use Theorem 9 to devise a new any-time MAP algorithm which refines the partitions to get a better estimate of MAP values. Our approach is presented in Algorithm 6.

The algorithm begins by identifying all non-liftable domains, namely domains $U_i$ that will be partially grounded during the execution of Algorithm 2 and associating a 1-partition $\pi_i$ with each domain. Then, until there is timeout, it iterates through the following two steps. First, it runs the LMAP algorithm, which uses the pair $(U_i, \pi_i)$ in Algorithm partition-ground during the $i$-th partial grounding step, yielding a MAP solution $\mu$. Second, it heuristically selects a partition $\pi_j$ and refines it. From Theorem 9 it is clear that as the number of iterations is increased, the MAP solution will either improve or remain the same. Thus, Algorithm Refine-MAP is an anytime algorithm.

<table>
<thead>
<tr>
<th>Algorithm 6: Refine-MAP(MLN $M$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Let $\mathcal{U} = {U_i}$ be the non-liftable domains</td>
</tr>
<tr>
<td>Set $\pi_i = {\Delta(j_R)}$ where $j_R \in U_i$ for all $U_i \in \mathcal{U}$</td>
</tr>
<tr>
<td>$\mu = -\infty$</td>
</tr>
<tr>
<td><strong>while</strong> timeout has not occurred <strong>do</strong></td>
</tr>
<tr>
<td>$\mu$ = LMAP($M$)</td>
</tr>
<tr>
<td>/* LMAP uses the pair $(U_i, \pi_i)$ and Algorithm partition-ground for its $i$-th partial grounding step. */</td>
</tr>
<tr>
<td>Heuristically select a partition $\pi_j$ and refine it</td>
</tr>
<tr>
<td><strong>return</strong> $\mu$</td>
</tr>
</tbody>
</table>

Alternatively, we can also devise an any-space algorithm using the following idea. We will first determine $k$, the maximum size of a partition that we can fit in the memory. As different partitions of size $k$ will give us different MAP values, we can search through them to find the best possible MAP solution. A drawback of the any-space approach is that it explores a prohibitively large search space. In particular, the number of possible partitions of size $k$ for a set of size $n$ (denoted by $\binom{n}{k}$) is given by the so called Stirling numbers of the
\begin{align*}
\{\{1\}, \{2\}, \{3\}, \{4\}\} \\
\{\{1\}, \{2\}, \{3, 4\}\} \\
\{\{1\}, \{2, 3\}, \{4\}\} \\
\{\{1\}, \{2, 3, 4\}\}
\end{align*}

Figure 5.1. Exchangeable Partition Lattice corresponding to the domain \{1, 2, 3, 4\}.

second kind which grows exponentially with \(n\). (The total number of partitions of a set is
given by the Bell number, \(B_n = \sum_{k=1}^{n} \binom{n}{k}\)). Clearly, searching over all the possible partitions
of size \(k\) is not practical. Luckily, we can exploit symmetries in the MLN representation to
substantially reduce the number of partitions we have to consider, since many of them will
give us the same MAP value. Formally,

**Theorem 10.** Given two \(k\)-partitions \(\pi = \{\pi_1, \ldots, \pi_k\}\) and \(\phi = \{\phi_1, \ldots, \phi_k\}\) of \(U\) such that
\(|\pi_i| = |\phi_i|\) for all \(i\), the MAP value of the partially ground MLN with respect to \(\pi\) is equal to
the MAP value of the partially ground MLN with respect to \(\phi\).

\[\begin{align*}
\text{Proof.} \text{ A formula } f, \text{ when grounded on an argument } i_R \text{ with respect to a partition } \pi
\text{ creates } |\pi| \text{ copies of the formula. Since } |\phi| = |\pi| = k \text{ grounding on } i_R \text{ with respect to } \phi
\text{ also creates the same number of formulas which are identical up to a renaming of constants.}
\text{Furthermore, since } |\pi_i| = |\phi_i| \text{ (each of their parts have identical cardinality) and as weight of a ground formula is determined by the cell sizes (see Algorithm Partition-Ground) the ground formulas obtained using } \phi \text{ and } \pi \text{ will have same weights as well. As a result, MLNs obtained by grounding on any argument } i_R \text{ with respect to } \phi \text{ and } \pi \text{ are indistinguishable (subject to renaming of variables and constants) and the proof follows.} \]

From Theorem 10 it follows that the number of elements in cells and the number of cells of
a partition is sufficient to define a partially ground MLN with respect to that partition. Con-
secutive refinements of such partitions will thus yield a lattice, which we will refer to as Exchangeable Partition Lattice. The term “exchangeable” refers to the fact that two partitions containing same number of elements of cells and same number of cells are exchangeable with each other (in terms of MAP solution quality). Figure 5.1 shows the Exchangeable Partition Lattice corresponding to the domain \{1, 2, 3, 4\}. If we do not use exchangeability, the number of partitions in the lattice would have been \( B_4 = \binom{4}{1} + \binom{4}{2} + \binom{4}{3} + \binom{4}{4} = 1 + 7 + 6 + 1 = 15. \) On the other hand, the lattice has 5 elements.

Different traversal strategies of this exchangeable partition lattice will give rise to different lifted MAP algorithms. For example, a greedy depth-first traversal of the lattice yields Algorithm 6. We can also explore the lattice using systematic depth-limited search and return the maximum solution found for a particular depth limit \( d \). This yields an improved version of our any-space approach described earlier. We can even combine the two strategies by traversing the lattice in some heuristic order. For our experiments, we use greedy depth-limited search, because full depth-limited search was very expensive. Note that although our algorithm assumes normal MLNs, which are pre-shattered, we can easily extend it to use shattering as needed (Kisynski and Poole, 2009). Moreover by clustering evidence atoms together (Van den Broeck and Darwiche, 2013; Venugopal and Gogate, 2014) we can further reduce the size of the shattered theory (de Salvo Braz, 2007).

5.3 Experiments

We implemented our algorithm on top of the lifted MAP algorithm presented at Chapter 4 (Sarkhel et al., 2014a), which reduces lifted MAP inference to an integer polynomial program (IPP). We will call our algorithm P-IPP (which stands for partition-based IPP). We performed two sets of experiments. The first set measures the impact of increasing the partition size \( k \) on the quality of the MAP solution output by our algorithm. The second set compares the performance and scalability of our algorithm with several algorithms from
literature. All of our experiments were run on a third generation i7 quad-core machine having 8GB RAM.

We used following five MLNs in our experimental study:

(i) An MLN which we call **Equivalence** that consists of following three formulas:

\[
\text{Equals}(x,x),
\text{Equals}(x,y) \rightarrow \text{Equals}(y,x), \text{and}
\text{Equals}(x,y) \land \text{Equals}(y,z) \rightarrow \text{Equals}(x,z);
\]

(ii) The **Student** MLN from (Sarkhel et al., 2014b,a), consisting of four formulas and three predicates;

(iii) The **Relationship** MLN from (Sarkhel et al., 2014a), consisting of four formulas and three predicates;

(iv) **WebKB** MLN (Kok et al., 2008) from the Alchemy web page, consisting of three predicates and seven formulas; and

(v) **Citation Information-Extraction** (IE) MLN from the Alchemy web page (Kok et al., 2008), consisting of five predicates and fourteen formulas

We compared the solution quality and scalability of our approach with the following algorithms and systems: Alchemy (ALY) (Kok et al., 2008), Tuffy (TUFFY) (Niu et al., 2011), ground inference based on integer linear programming (ILP) and the IPP algorithm of Chapter 4 (Sarkhel et al., 2014a). Alchemy and Tuffy are two state-of-the-art open source software packages for learning and inference in MLNs. Both of them ground the MLN and then use an approximate solver, MaxWalkSAT (Selman et al., 1996) to compute the MAP solution. Unlike Alchemy, Tuffy uses clever Database tricks to speed up computation and in principle can be much more scalable than Alchemy. ILP is obtained by converting the
MAP problem over the ground Markov network to an Integer Linear Program. We ran each algorithm on the aforementioned MLNs for varying time-bounds and recorded the solution quality, which is measured using the total weight of the false clauses in the (approximate) MAP solution, also referred to as the cost. Smaller the cost, better the MAP solution. For a fair comparison, we used a parallelized Integer Linear Programming solver called Gurobi \cite{Gurobi2016} to solve the integer linear programming problems generated by our algorithm as well as by other competing algorithms.

Figure 5.2 shows our experimental results. Note that if the curve for an algorithm is not present in a plot, then it means that the corresponding algorithm ran out of either memory or time on the MLN and did not output any solution. We observe that Tuffy and Alchemy are the worst performing systems both in terms of solution quality and scalability. ILP scales slightly better than Tuffy and Alchemy. However, it is unable to handle MLNs having more than 30K clauses. We can see that our new algorithm P-IPP, run as an anytime scheme, by refining partitions, not only finds higher quality MAP solutions but also scales better in terms of time complexity than IPP. In particular, IPP could not scale to the equivalence MLN having roughly 1 million ground clauses and the relation MLN having roughly 125.8M ground clauses. The reason is that these MLNs have self-joins (same predicate appearing multiple times in a formula), which IPP is unable to lift. On the other hand, our new approach is able to find useful approximate symmetries in these hard MLNs.

To measure the impact of varying the partition size on the MAP solution quality, we conducted the following experiment. We first ran the IPP algorithm until completion to compute the optimum MAP value. Then, we ran our algorithm multiple times, until completion as well, and recorded the solution quality achieved in each run for different partition sizes. Figure 5.3 plots average cost across various runs as a function of $k$ (the error bars show the standard deviation). For brevity, we only show results for the IE and Equivalence MLNs. The optimum solutions for the three MLNs were found in (a) 20 minutes, (b) 6 hours
Figure 5.2. Cost vs. Time: Cost of unsatisfied clauses (smaller is better) vs. time for different domain sizes. Notation used to label each figure: MLN(numvariables, numclauses). Note: the quantities reported are for ground Markov network associated with the MLN. Standard deviation is plotted as error bars.

and (c) 8 hours respectively. On the other hand, our new approach P-IPP yields close to optimal solutions in a fraction of the time, and for relatively small values of $k \approx 5 - 10$.

5.4 Conclusion

Lifted inference techniques have gained popularity in recent years, and have quickly become the approach of choice to scale up inference in MLNs. A pressing issue with existing lifted
inference technology is that most algorithms only exploit exact, identifiable symmetries and resort to grounding or propositional inference when such symmetries are not present. This approach is problematic because grounding can blow up the search space. In this chapter, we proposed a principled, approximate approach for solving this grounding problem. The main idea in our approach is to partition the ground atoms into a small number of groups and then treat all ground atoms in a group as indistinguishable (from each other). This simple idea introduces new, approximate symmetries which can help speed-up the inference process. Although our proposed approach is inherently approximate, we proved that it has nice theoretical properties in that it is guaranteed to yield a consistent assignment that is a lower-bound on the MAP value. We further described an any-time algorithm which can improve this lower bound through systematic refinement of the partitions. Finally, based on the exchangeability property of the refined partitions, we demonstrated a method for organizing the partitions in a lattice structure which can be traversed heuristically to yield efficient any-time as well as any-space lifted MAP inference algorithms. Our experiments on a wide variety of benchmark MLNs clearly demonstrate the power of our new approach.
CHAPTER 6
EXPLOITING EFFICIENT COUNTING STRATEGIES FOR SCALABLE INFERENCE AND LEARNING

Acknowledgements

I wish to acknowledge the contributions of Deepak Venugopal in the work presented here. Deepak helped in the design of the Gibbs sampling algorithm, Contrastive Divergence algorithm and running corresponding experiments presented in this chapter.

6.1 Introduction

So far in this dissertation, we have only considered “Normal MLN”, i.e., MLN without any evidence. Lifted inference algorithms, like the ones presented in the previous three chapters, typically exploit symmetries in the Normal MLN to scale up inference. However, when an MLN has a significant amount of evidence atoms, converting it to a normal MLN can blow up the representation and inference. Hence, for the vast majority of real-world MLNs lifted inference techniques cannot scale-up. In this chapter, we consider an alternative and relatively unexplored approach for scalable inference. Our approach is orthogonal to existing lifted inference methods and does not require the MLNs to be in normal form.

Specifically, in this chapter, we identify the main computational bottleneck in MLN inference and learning algorithms. This bottleneck is counting the true groundings of a first-order formula \( f \) given a world \( \omega \). We observed that existing MLN systems such as Alchemy \((\text{Kok et al., 2006})\) and Tuffy \((\text{Niu et al., 2011})\) solve this counting problem using the following naive “generate-and-test” approach: generate each possible grounding and test whether it is true in \( \omega \). This naive approach is the chief reason for their poor scalability and
is particularly problematic because algorithms such as Gibbs sampling (Geman and Geman, 1984; Venugopal and Gogate, 2012) and MaxWalksat (Kautz et al., 1997) perform the above operation in every iteration.

We proposed a novel approach to solving this counting problem. The key advantages of our approach are that it never grounds the full MLN and in most cases is orders of magnitude better than the “generate-and-test” approach. Specifically, we encode each formula \( f \) as a CSP \( C \) such that the number of solutions to \( C \) can be directly used to count the satisfied groundings of \( f \). The main advantage of this encoding is that we can now leverage several years of advances in CSPs and graphical model inference and use virtually any exact/approximate inference algorithm along with its associated guarantees to efficiently solve the counting problem.

We demonstrate the power of our approach by showing that it greatly improves the computational complexity of two popular approximate inference algorithms: Gibbs Sampling and MaxWalkSAT. We show that in both these algorithms, the main computational steps involve solving the encoded CSP, where the constraints change over time (dynamic CSP). To solve this dynamic CSP, we compile an exact junction tree for the CSP and account for the changing constraints by modifying the junction tree messages. We evaluated both algorithms on a wide variety of MLN benchmarks and compared them with algorithms implemented in two existing MLN systems, Alchemy and Tuffy. Our experiments clearly show that our new algorithms are several orders of magnitude more scalable than legacy systems.

However, it quickly became evident to us that even this sophisticated approach is not enough for scaling up the learning task in MLN. The main reason for the poor scalability is the high polynomial complexity of algorithms used for computing the gradient. Specifically, to calculate the gradient we need to solve the previously mentioned counting problem multiple times. Exact algorithms for determining these counts have high polynomial complexity. For example, given a formula \( \forall d \forall a_1 \forall a_2 \text{HasAuthor}(d, a_1) \land \text{HasAuthor}(d, a_2) \Rightarrow \text{Coauthor}(a_1, a_2) \)
(if $a_1$ and $a_2$ are authors of document $d$, then they are likely to be coauthors), even our advanced exact counting algorithms will incur a complexity of $O(nm^2)$ where $n$ is number of documents and $m$ is the number of authors.

Currently, the most prevalent approach for achieving scalability in weight learning is to incorporate background knowledge in the MLN as hard evidence. Hard evidence substantially reduces the complexity of solving the counting problem, reducing it from $O(s^k)$ to $O(r^k)$ where $r << s$. This is because most ground formulas that contain evidence atoms will evaluate to either TRUE or FALSE, and they can be deleted from consideration. For example, in the entity resolution domain, Singla et al. (Singla and Domingos, 2006) achieve scalability by adding hard evidence, setting all atoms that are likely to be false (or true) with high probability to FALSE (or TRUE). These atoms are identified using the canopy approach of McCallum et al. (McCallum et al., 2000) – a cheap, domain-specific heuristic. The main drawbacks of using background knowledge are: (1) the added evidence may be incorrect and thus the approach introduces errors, and (2) the approach will not scale to large domains if the number of evidence atoms is small.

To scale-up the learning task in MLNs, in this chapter, we present a principled approach, which can scale seamlessly to large domains, even when background knowledge is not available. Our contributions are three-fold:

- We propose new objective functions for weight learning that approximate well-known functions such as likelihood, pseudo-likelihood, and contrastive divergence. The key idea is to use approximate instead of exact approaches for solving the counting problem. We analyze the complexity of our new algorithms and show that they are highly scalable.

- We provide theoretical bounds on how far our solutions are from the solutions obtained using existing exact methods.
The number of 1s in $\phi_3$ equals the number of groundings of $f$ that evaluate to False. (d) The constraint network for the formula

- We demonstrate experimentally that our new algorithms are superior to current weight learning algorithms implemented in Alchemy (Kok et al. 2008) and Tuffy (Niu et al. 2011), two state-of-the-art systems for learning and inference in Markov logic. More importantly, our results show that our new methods do not require background knowledge to scale to large domains.

6.2 \(\#sg\) as a CSP

We demonstrate our proposed CSP encoding on a simple MLN having just one clause: $f = \forall x, y, z \ R(x, y) \lor S(y, z)$. We will focus on counting the number of false groundings of $f$ given a world $\omega$ because it is easier to compute. Moreover, we can easily compute the number of true groundings $N_f(\omega)$ from it; $N_f(\omega)$ is equal to the number of all possible groundings (which is simply a product of the domain sizes) minus the number of false groundings.
Let us assume that the domain of each logical variable in $f$ is $\{A, B\}$. Each triple $(x, y, z)$ where each $x$, $y$ and $z$ can take values from the domain $\{A, B\}$ uniquely identifies each grounding of the formula. Consider a world $\omega$ shown in Figure 6.1(a). Let us associate two 0/1 functions $\phi_1(x, y)$ and $\phi_2(y, z)$ with $R(x, y)$ and $S(y, z)$ respectively. The 0/1 function has a value 1 iff the corresponding grounding is False in the world and 0 otherwise (see Figure 6.1(b)).

Given this set up, notice that if we take a product of the two functions $\phi_1(x, y)$ and $\phi_2(y, z)$, then the resulting function $\phi_3(x, y, z)$ will have a 1 associated with an entry $(x, y, z)$ iff both $R(x, y)$ and $S(y, z)$ are False. Since the ground formula $(x, y, z)$ evaluates to False iff both $R(x, y)$ and $S(y, z)$ are False, by extension $\phi_3(x, y, z) = 1$ implies that the ground formula $(x, y, z)$ is False. Therefore, we can count the number of groundings of $f$ that evaluate to False, by simply counting the number of ones in $\phi_3(x, y, z)$, which is the same as counting the number of solutions to the CSP having two functions $\phi_1(x, y)$ and $\phi_2(y, z)$.

Next, we will formalize this intuition and precisely define how to encode the $\#\text{SG}$ problem as a CSP solution counting problem.

**Encoding $f$-to-CSP.** Given a first-order clause $f$ and a world $\omega$, the corresponding CSP $\mathcal{C}$ has a variable for each (universally quantified) logical variable in $f$. The domain of each variable in $\mathcal{C}$ is the set of constants in the domain of the corresponding logical variable. For each atom $R(x_1, \ldots, x_u)$ in $f$, we have a relation $\phi$ in $\mathcal{C}$ defined as follows:

$$
\phi(\overline{x}_u) = \begin{cases} 
\omega_{R(X_1, \ldots, X_u)} & \text{if } R \text{ is negated in } f \\
\neg\omega_{R(X_1, \ldots, X_u)} & \text{Otherwise}
\end{cases}
$$

where $\overline{x}_u = (x_1 = X_1, \ldots, x_u = X_u)$ denotes an assignment to the CSP variables and $\omega_{R(X_1, \ldots, X_u)}$ is the projection of the world $\omega$ on the ground atom $R(X_1, \ldots, X_u)$, namely the truth-value of the ground atom $R(X_1, \ldots, X_u)$ in $\omega$. 
Table 6.1. #SG complexities using various strategies. \(M\) is the number of formulas, \(V_i\) is the set of variables in the CSP encoded for the \(i^{th}\) formula, \(d\) is the domain-size of each variable and \(w^*_i\) is the treewidth of the CSP encoded for the \(i^{th}\) formula.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Space Complexity</th>
<th>Time complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pre-Ground</td>
<td>(O(\sum_{i=1}^{M} d</td>
<td>V_i</td>
</tr>
<tr>
<td>Lazy-Ground</td>
<td>(O(1))</td>
<td>(O(\sum_{i=1}^{M} d</td>
</tr>
<tr>
<td>And-OR Tree</td>
<td>(O(M))</td>
<td>(O(\sum_{i=1}^{M} d</td>
</tr>
<tr>
<td>Junction Tree</td>
<td>(O(\sum_{i=1}^{M} d</td>
<td>V_i</td>
</tr>
</tbody>
</table>

By generalizing the arguments presented for the example MLN formula given above, we can show that:

**Theorem 11.** Let \(f\) be a first-order clause, \(x_1, \ldots, x_u\) be the (universally quantified) logical variables in \(f\), \(\omega\) be a world and let \(#\text{Sol}(C)\) denote the number of solutions of the CSP \(C\) obtained from \((f_i, \omega)\) using the \(f\)-to-CSP encoding. Then, \(N_f(\omega) = \prod_{j=1}^{u} |\Delta(x_j)| - #\text{Sol}(C)\) where \(\Delta(x_j)\) is the set of constants in the domain of \(x_j\).

### 6.2.1 Counting the Number of Solutions of the CSP

Since we have reduced the #SG problem to the CSP solution counting problem, it is clear that we can use any CSP/graphical model inference algorithm and leverage its advances and guarantees to efficiently compute the former. Table 6.2 shows the complexity bounds for various strategies and algorithms for solving the #SG problem.

Alchemy (Kok et al., 2006) uses the pre-ground strategy, namely it grounds all clauses that it cannot lift (Gogate and Domingos, 2011). Thus, its worst case time and space complexity bounds are exponential in the maximum number of variables in the MLN formulas. The pre-ground strategy is useful when there is a large amount of evidence. In presence of evidence, one can use unit propagation to remove all clauses that are either True or False. This reduces the space complexity as well as the time complexity of subsequent counting problems (e.g., when the counts are required at each iteration of Gibbs sampling).
An alternative strategy is to do lazy grounding, which reduces to solving the CSP using
the generate and test approach. In this approach, we count the solutions by generating
each tuple and testing whether it is a solution or not. Although this approach has constant
space complexity, its worst case time complexity is the same as the pre-ground approach.
Moreover, unlike the pre-ground approach, this approach is unable to take advantage of unit
propagation and the worst-case results apply to all subsequent operations.

A better, more powerful approach is to use advanced search and elimination techniques
such as AND/OR search \cite{dechter2007}, recursive conditioning \cite{darwiche2001}, junction tree propagation \cite{lauritzen1988}, as well as knowledge
compilation techniques such as arithmetic circuits \cite{darwiche2003} and AND/OR multi-
valued decision diagrams \cite{mateescu2008}. In this chapter, we focus on using the
junction tree algorithm for computing the solution counts, noting that in future, one can
use other advanced techniques mentioned above as well as approximate solution counting
approaches such as SampleCount \cite{gomes2007}, SampleSearch \cite{gogate2007} and generalized BP \cite{yedidia2005}.

6.2.2 Junction Trees for Solution Counting

We now briefly review the junction tree algorithm used to compute the number of solutions
\(#\text{Sol}(\mathcal{C})\) of \(\mathcal{C}\).

**Definition 9.** Given the CSP \(\mathcal{C}\), a **junction tree** is a tree \(\mathcal{T}(V,E)\) in which each vertex
\(V \in V\) (also called a cluster) and edge \(E \in E\) are labeled with a subset of variables, denoted
by \(L(V)\) and \(L(E)\) such that: (i) for every function \(\phi\) defined in \(\mathcal{C}\), there exists a vertex \(L(V)\)
such that \(S(\phi) \subseteq L(V)\) and (ii) for every variable \(x\) in \(\mathcal{C}\), the set of vertexes and edges in \(\mathcal{T}\)
that mention \(x\) form a connected sub-tree in \(\mathcal{T}\) (called the running intersection property).

Given a junction tree \(\mathcal{T}\) of \(\mathcal{C}\), we can compute the solution counts as well as the marginal
probability of each CSP variable (the fraction of solutions that the variable participates in)
by calibrating $\mathcal{T}$. We calibrate $\mathcal{T}$ by selecting a cluster as the root and performing sum-product message passing in two passes: from the leaves to the root (collect pass) and then from the root to the leaves (distribute pass). Formally, the message sent from cluster $i$ to cluster $j$, denoted by $m_{i \rightarrow j}$, is given by

$$m_{i \rightarrow j}(y) = \sum_z \prod_{\phi \in \Phi(V_i)} \phi(y, z) \prod_{k \in N(i) \setminus \{j\}} m_{k \rightarrow i}(y, z)$$

(6.1)

where $\Phi(V_i)$ is the set of functions assigned to vertex $V_i$, and $N(i)$ is the set of indexes of the neighbors of $V_i$ in $\mathcal{T}$.

The number of solutions to $\mathcal{C}$ can be computed from any vertex $V_k$ using the following equation:

$$\#\text{Sol}(\mathcal{C}) = \sum_x \prod_{\phi \in \Phi(V_k)} \phi(x) \prod_{j \in N(k)} m_{j \rightarrow k}(x)$$

(6.2)

The complexity of computing the solution counts using a junction tree is exponential in the maximum cluster size of the tree which equals treewidth plus 1. Next, we describe efficient implementations of two classical approximate inference algorithms, Gibbs sampling and MaxWalkSAT, using calibrated junction trees. \(^1\)

### 6.3 Application I: Gibbs Sampling

In Gibbs sampling, we start with a random world $\omega^{(0)}$. Then at each iteration $i > 0$, we compute the conditional distribution over a randomly chosen ground atom $R$ given $\omega_{-R}^{(i-1)}$ where $\omega_{-R}^{(i-1)}$ is the projection of $\omega^{(i-1)}$ on all ground atoms of the MLN except $R$. Then, we sample a new value for $R$, denoted by $\bar{R}$, from this conditional distribution and set $\omega^{(i)} = (\omega_{-R}^{(i-1)}, \bar{R})$. Note that for brevity, we have abused notation and denoted the ground atom $R(X_1, \ldots, X_r)$ as $R$.

\(^1\)Technically, Gibbs sampling can be implemented more efficiently with uncalibrated junction trees while MaxWalkSAT requires calibration.
The main computational bottleneck in Gibbs sampling is computing the conditional
distribution over the ground atom $\omega^{(i)}$. It is given by:

$$\Pr(R = j|\omega_{-R}^{(i)}) \propto \sum_{f_k \in F(R)} w_k N_{f_k}(\omega_{-R}^{(i)}, R = j)$$  \hspace{1cm} (6.3)

where $j \in \{0, 1\}$ and $F(R)$ is the set of first-order formulas in the MLN that contain $R$ (the
Markov Blanket of $R$).

Given a formula $f_k$ and a world $\omega$, let $C_k$ denote the constraint network obtained from
$(f_k, \omega)$ using the f-to-CSP encoding. Let $T_k$ denote the junction tree obtained from $C_k$. If we
calibrate the junction tree $T_k$, then we can easily compute $N_{f_k}(\omega)$ from it (see Eq.6.2). The
main challenge is computing $N_{f_k}(\omega')$ where $\omega' = (\omega_{-R}, \neg\omega_R)$. We describe how to compute
it next.

Consider the two CSPs $C_k$ and $C'_k$ obtained from $(f_k, \omega)$ and $(f_k, \omega')$ respectively using
the f-to-CSP encoding. Since $f_k$ defines the scope of the functions in the CSP, both CSPs
have functions defined over the same scope. Moreover, since $\omega$ and $\omega'$ differ only in a truth
assignment to one ground atom, the corresponding functions in $C_k$ and $C'_k$ differ only in at
most one entry. Thus, we can efficiently construct a calibrated junction tree $T'_k$ from $T_k$ by
appropriately propagating the changed entries. Specifically, assuming that all functions that
have changed are present in a cluster $V$ in $T_k$, we designate $V$ as the root and distribute
messages away from it, stopping propagation beyond a cluster if the new message and the
old message to the cluster are the same.
6.4 Application II: MaxWalkSAT

Algorithm 7: MaxWalkSAT($\mathcal{M}, p, \text{maxFlips}$)

**Initialize**: $\omega = \text{a randomly generated possible world of } \mathcal{M}$.

$\omega_{\text{Best}} = \omega$

**for** $\text{flip} = 1$ to $\text{maxFlips}$ **do**

- **if** $\omega$ satisfies all clauses in $\mathcal{M}$ **then**
  
  return $\omega$

- $f_k = \text{a random unsatisfied ground clause in } \mathcal{M}$

  **With** probability $p$
  
  Flip a randomly selected ground atom of $f_k$

  **Else**
  
  Flip the ground atom of $f_k$ that yields a world with the maximum weight

  **if** $\sum_i N_{f_i}(\omega) > \sum_i N_{f_i}(\omega_{\text{Best}})$ **then**

  $\omega_{\text{Best}} = \omega$

**return** $\omega_{\text{Best}}$

The MAP problem in Markov logic networks reduces to the problem of finding a possible world that maximizes sum of weights of satisfied clauses. Any weighted satisfiability solver can used to solve the MAP problem, however the most commonly used solver is MaxWalkSAT (Kautz et al. 1997). The latter is a weighted variant of WalkSAT, a local-search algorithm for satisfiability testing (Selman et al. 1996) and is the algorithm used in Alchemy as well as Tuffy for MAP inference.

For convenience, MaxWalkSAT algorithm is given in Algorithm 7. It takes as input an MLN $\mathcal{M}$, an integer $\text{maxFlips}$ which determines the maximum number of flips and a probability $p$. The algorithm begins by randomly generating a possible world. Then, at each iteration, it selects an unsatisfied ground clause uniformly at random and flips the value assigned to one of its atoms. With probability $p$, the atom (literal) to be flipped is selected uniformly at random and with probability $1 - p$, the atom when flipped maximizes
the number of satisfied ground clauses is selected (greedy hill-climbing step which selects the best atom).

The last two steps of flipping a random atom and the best atom can be accomplished by using the same approach described in the previous section. Namely, we maintain a calibrated junction tree for the current world $\omega$ and each formula $f_k$, updating it as necessary to determine the weight of the world after the flip. The most challenging step is selecting an unsatisfied ground formula uniformly at random and we describe a procedure next for accomplishing this.

**Selecting an Unsatisfied Clause uniformly at random:** We solve this problem using a two step procedure. In the first step, we select a first order formula $f_i$ and in the second step we select a ground clause of $f_i$ uniformly at random.

To select a first-order formula, we first compute the number of unsatisfied ground clauses for all first-order formulas (using the calibrated junction tree) and normalize them to yield a distribution over the formulas. We then sample a first-order formula from this distribution. Notice that the probability of selecting a first-order formula using this procedure is proportional to number of its unsatisfied groundings.

Let $f_i$ be the sampled first-order formula. To generate a ground clause of $f_i$ uniformly at random, we can sample the calibrated junction tree of $f_i$ using the junction tree solution sampling method described in [Dechter et al., 2002; Gogate 2009]. Sampling the junction tree yields a solution to the corresponding CSP $C_i$. Based on our encoding, every solution of $C_i$ corresponds to an unsatisfied clause of $f_i$.

**Selecting an Unsatisfied Clause in Presence of Evidence:** In presence of evidence, the solution sampling method described above cannot be used because the sampled unsatisfied clause can be *trivially unsatisfied*, i.e., it is unsatisfied and all of its atoms are evidence, hence none of them can be flipped. Thus, the solution sampling method must be modified so that it never generates a trivially unsatisfied formula.
Before explaining our method to solve this problem, we introduce some notation. Let $\mathcal{E}$, $\mathcal{E}_u$ and $\mathcal{E}_t$ denote the set of ground clauses, unsatisfied ground clauses and trivially unsatisfied ground clauses of $f$ respectively. From the definition, it is obvious that $\mathcal{E}_t \subseteq \mathcal{E}_u \subseteq \mathcal{E}$. In order to not generate a trivially unsatisfied clause, it is obvious that we should draw a solution uniformly at random from the set $(\mathcal{E}_u \setminus \mathcal{E}_t)$. We can use rejection sampling to find such a clause but this will be quite slow, especially when there is a large amount of evidence.

A more clever approach is to use two constraint networks. Formally, given a formula $f_k$ and the constraint network $\mathcal{C}_k$ obtained from it we define another constraint network (referred to as evidence network) $\mathcal{C}_k^e$ having the same variables as $\mathcal{C}_k$. Corresponding to each function $\phi(\overline{x}_u)$ in $\mathcal{C}_k$, we further define a function $\phi^e(\overline{x}_u)$ in $\mathcal{C}_k^e$ as follows:

$$
\phi^e(\overline{x}_u) = \begin{cases} 
\phi(\overline{x}_u) & \text{if } R(\overline{x}_u) \text{ is an evidence} \\
0 & \text{Otherwise}
\end{cases}
$$

where $\overline{x}_u = (x_1 = X_1, \ldots, x_u = X_u)$ and $R(\overline{x}_u)$ denotes the atom corresponding to $\phi(\overline{x}_u)$ in $\mathcal{C}_k$. From the construction of $\mathcal{C}_k^e$ it follows that any solution to $\mathcal{C}_k^e$ will be a trivially unsatisfied clause.

Since $\mathcal{E}_t \subseteq \mathcal{E}_u$, we have $|\mathcal{E}_u \setminus \mathcal{E}_t| = |\mathcal{E}_u| - |\mathcal{E}_t|$. Moreover, Since $\#Sol(\mathcal{C}_k) = |\mathcal{E}_u|$ and $\#Sol(\mathcal{C}_k^e) = |\mathcal{E}_t|$ we have $|\mathcal{E}_u \setminus \mathcal{E}_t| = \#Sol(\mathcal{C}_k) - \#Sol(\mathcal{C}_k^e)$.

Thus, we have a way of computing the number of non-trivial unsatisfied groundings of a formula. As outlined earlier for the non-evidence case, this can be used to sample a first-order formula $f_i$ (by sampling it with a probability proportional to this number). To generate a non-trivial unsatisfied grounding of $f_i$, we will compute a calibrated junction tree $(\mathcal{T}_k^e)$, and at the time of sampling a cluster $V$ in $\mathcal{T}_k$, we will exclude the trivially unsatisfied clauses by subtracting from it the counts in the corresponding cluster of $\mathcal{T}_k^e$. Note that the junction tree $\mathcal{T}_k^e$ needs to be calibrated only once since the evidence never changes.
6.5 Inference Experiments

6.5.1 Setup

We used 10 benchmark MLNs with varied structures and random evidence (< 25%) to evaluate the performance of our Gibbs sampling and MaxWalkSAT algorithms. We compared our system with two state-of-the-art MLN systems: Alchemy and Tuffy. Alchemy implements both Gibbs sampling as well as MaxWalkSAT whereas Tuffy only implements MaxWalkSAT. Of the 10 benchmarks, 5 were from Alchemy: webkb, entity resolution (er), segmentation (seg), protein interaction (protein) and coreference resolution (coref). Our 5 synthetic benchmarks are as follows:

(i) student: \( \neg \text{Student}(x, p) \lor \neg \text{Publish}(x, z) \lor \text{Cited}(z, u) \)

(ii) relation: \( \neg \text{Friends}(x, y) \lor \neg \text{Related}(y, z) \lor \text{Likes}(z, x) \)

(iii) longchain: \( \neg R_1(x_1, x_2) \lor \neg R_2(x_2, x_3) \ldots R_6(x_6, x_7) \)

(iv) transitive1: \( \neg \text{Likes}(x, y) \lor \neg \text{Likes}(y, z) \lor \text{Likes}(y, x) \)

(v) transitive2: \( \neg \text{Friends}(x, y) \lor \neg \text{Friends}(y, z) \lor \text{Friends}(z, x) \)

Each synthetic benchmark was designed to illustrate the influence of MLN structure on scalability. Specifically, though similar-looking, student has a smaller treewidth (for its encoded CSP) compared to relation. Longchain illustrates a large formula with small treewidth. Finally, though transitive1 and transitive2 appear similar, it turns out that in each step of Gibbs/MaxWalkSAT, very few messages of the junction tree underlying transitive1 need to be updated while for transitive2, nearly all messages need to be updated.

Figures 6.2 and 6.3 show our results for Gibbs sampling and MaxWalkSAT respectively. For our evaluation, we compute two metrics: the compilation time (C-Time) in seconds and the sampling/flip rate (SRate/FRate). C-time is the time taken to initialize our junction
<table>
<thead>
<tr>
<th>MLN</th>
<th>#Groundings</th>
<th>C-Time</th>
<th>SRate</th>
</tr>
</thead>
<tbody>
<tr>
<td>student-100</td>
<td>1.0003e+08</td>
<td>0</td>
<td>11397</td>
</tr>
<tr>
<td>student-500</td>
<td>6.25008e+10</td>
<td>0</td>
<td>496.72</td>
</tr>
<tr>
<td>student-1000</td>
<td>1e+12</td>
<td>0</td>
<td>117.901</td>
</tr>
<tr>
<td>relation-100</td>
<td>1.03e+06</td>
<td>0</td>
<td>7047.97</td>
</tr>
<tr>
<td>relation-500</td>
<td>1.2575e+08</td>
<td>1</td>
<td>274.901</td>
</tr>
<tr>
<td>relation-1000</td>
<td>1.003e+09</td>
<td>10</td>
<td>68.6392</td>
</tr>
<tr>
<td>longchain-100</td>
<td>1e+14</td>
<td>0</td>
<td>3235.09</td>
</tr>
<tr>
<td>longchain-500</td>
<td>7.8125e+18</td>
<td>0</td>
<td>126.147</td>
</tr>
<tr>
<td>longchain-1000</td>
<td>1e+21</td>
<td>1</td>
<td>31.836</td>
</tr>
<tr>
<td>transitive1-100</td>
<td>1.01e+06</td>
<td>0</td>
<td>88739.7</td>
</tr>
<tr>
<td>transitive1-500</td>
<td>1.2525e+08</td>
<td>0</td>
<td>24568.4</td>
</tr>
<tr>
<td>transitive1-1000</td>
<td>1.001e+09</td>
<td>0</td>
<td>8879.61</td>
</tr>
<tr>
<td>transitive2-100</td>
<td>1.01e+06</td>
<td>0</td>
<td>73.4589</td>
</tr>
<tr>
<td>transitive2-500</td>
<td>1.2525e+08</td>
<td>1</td>
<td>0.590163</td>
</tr>
<tr>
<td>webkb</td>
<td>1.01011e+10</td>
<td>1</td>
<td>183.836</td>
</tr>
<tr>
<td>seg</td>
<td>1.26038e+09</td>
<td>0</td>
<td>32.6557</td>
</tr>
<tr>
<td>er</td>
<td>1.59375e+09</td>
<td>21</td>
<td>5.83606</td>
</tr>
<tr>
<td>protein</td>
<td>5.00312e+08</td>
<td>1</td>
<td>116.672</td>
</tr>
<tr>
<td>coref</td>
<td>5.27568e+08</td>
<td>2</td>
<td>180.967</td>
</tr>
</tbody>
</table>

Figure 6.2. Results on benchmarks for Gibbs sampling using our approach. **SRate** is the sampling rate (#samples/second) and **C-Time** is the compilation time in seconds. Note that Alchemy timed out (2 hours) or ran out of memory on all the instances and therefore its results are not shown.

trees. **SRate** is the number of samples generated in a second for Gibbs sampling and **FRate** is the number of flips per second in MaxWalkSAT.

### 6.5.2 Results for Gibbs Sampling

Both **C-time** as well as **SRate** depends upon the structure as well as the #groundings in the MLN. We can see from Figure 6.2 that **C-time** was quite negligible for almost all the MLNs (at most 21 seconds). **SRate** depends upon the efficiency of updating the junction tree messages during Gibbs sampling. For example, in transitive1, we could generate 88,000 samples/second because each update operation is very efficient, while for transitive2 which has the same #groundings, we could generate only 73 samples/second. Similarly, the MLNs, student-100 and relation-500 have approximately the same #groundings, however, their **SRates** are vastly different due to the treewidth of their encoded CSPs. Student has
<table>
<thead>
<tr>
<th>MLN</th>
<th>#Groundings</th>
<th>Ours</th>
<th>Alchemy</th>
<th>Tuffy</th>
</tr>
</thead>
<tbody>
<tr>
<td>student-100</td>
<td>1.0003e+08</td>
<td>0:31629</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>student-500</td>
<td>6.25008e+10</td>
<td>0:252.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>student-1000</td>
<td>1e+12</td>
<td>0:72</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>relation-100</td>
<td>1.03e+06</td>
<td>0:2455.5</td>
<td>95:1000</td>
<td>75:300</td>
</tr>
<tr>
<td>relation-500</td>
<td>1.2575e+08</td>
<td>1:142.8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>relation-1000</td>
<td>1.003e+09</td>
<td>13:36</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>longchain-100</td>
<td>1e+14</td>
<td>0:928.3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>longchain-500</td>
<td>7.8125e+18</td>
<td>0:50</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>longchain-1000</td>
<td>1e+21</td>
<td>1:12.3</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>transitive1-100</td>
<td>1.01e+06</td>
<td>0:32082</td>
<td>75:350</td>
<td>65:100</td>
</tr>
<tr>
<td>transitive1-500</td>
<td>1.2525e+08</td>
<td>0:1032</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>transitive1-1000</td>
<td>1.001e+09</td>
<td>0:284.2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>transitive2-100</td>
<td>1.01e+06</td>
<td>0:30</td>
<td>75:200</td>
<td>65:150</td>
</tr>
<tr>
<td>transitive2-500</td>
<td>1.2525e+08</td>
<td>1:0.22</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>webkb</td>
<td>1.01011e+10</td>
<td>1:48.5</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>seg</td>
<td>1.26038e+09</td>
<td>0:8.6</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>er</td>
<td>1.59375e+09</td>
<td>26:1.2</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>protein</td>
<td>5.00312e+08</td>
<td>1:6.8</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>coref</td>
<td>5.27568e+08</td>
<td>3:6.3</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Figure 6.3. Results on benchmarks for MaxWalkSAT. For each system, we show C-Time:FRate, where C-Time is the compilation time (in seconds) for our system or the grounding time in Alchemy/Tuffy. FRate is the flip rate (#Flips/second). “-” denotes that the system ran out of time or memory.

treewidth 1 whereas relation has treewidth 2, therefore, while we could collect more than 11,000 samples in a second for student-100, we could only collect about 275 samples for relation-500. On the other hand, for the same treewidth, #groundings affects SRate. For example, longchain-1000 is almost 10 million times larger than longchain-100. Therefore, FRate on longchain-1000 is just 10% of the FRate on longchain-1000. Note that Figure 6.2 does not include results for Alchemy because it timed out (2 hours) or ran out of memory on all the instances.

6.5.3 Results for MaxWalkSAT

Figure 6.3 shows our results for MaxWalkSAT. C-Time is very similar to the Gibbs sampler. Again, FRate depends upon the MLN structure and the number of groundings because both affect the efficiency of the junction tree operations. For low tree-width benchmarks such as
student, we get much higher flip rates as compared to higher treewidth benchmarks such as relation. Transitive2 has the worst performance because, each update involves recomputing the junction tree messages from scratch. When these updates are efficient as in transitive1, **FRate** is several orders of magnitude higher. Both Alchemy and Tuffy did not work on most of the benchmarks except on three of the smallest-sized ones; our approach was slightly worse than Tuffy and Alchemy only on transitive2-100.

In summary, our results clearly demonstrate the superior scalability of our approach over the pre-grounding approaches used by Tuffy and Alchemy.

### 6.6 Application III: Scalable Weight Learning

Existing weight learning algorithms, even those that use alternative criteria such as CD, VP and PLL are impractical because all algorithms require solving the $\#SG$ problem in order to compute the gradient. As described earlier, even advanced algorithms ([Venugopal et al., 2015](#)) for solving $\#SG$ have high polynomial complexity, bounded by $O(n^{w^*+1})$ where $w^*$ is the treewidth of the associated constraint network. Specifically, in CD (and VP), in order to generate a a sample (MAP tuple) via Gibbs sampling (MaxWalksat), one has to visit all ground formulas at least once. In PLL, one has to solve the $\#SG$ problem, once for each ground atom. However, in terms of computational complexity, a key advantage of PLL over CD and VP is that the counts remain constant over iterations and therefore they can be precomputed. Thus, PLL will be more scalable than VP and CD if the gradient ascent procedure requires a large number of iterations to converge.

#### 6.6.1 Relaxed Learning Formulation with Approximate Counts

Next, we consider a simple modification of the likelihood function in which instead of computing the exact counts $N_i(\omega)$, we assume that we have access to a procedure that computes approximate counts $M_i(\omega)$ such that the absolute error between the two is bounded by $\epsilon$. 

Namely, $\forall \omega \mid N_i(\omega) - M_i(\omega) \leq \epsilon$ where $\epsilon \geq 0$. In this case, our relaxed learning objective is defined as:

$$\max_{\theta} \hat{\ell}(\theta : \omega) = \sum_i \theta_i M_i(\omega) - \log \hat{Z}_\theta$$

subject to $\forall \omega \mid N_i(\omega) - M_i(\omega) \leq \epsilon$

where $\hat{Z}_\theta = \sum_{\omega'} \exp(\sum_i \theta_i M_i(\omega'))$.

We show that this relaxed learning objective is a reasonable approximation to the original objective in that the difference in the log-likelihood between the parameters learned using the relaxed and original objective is bounded linearly by the sum of parameter values and $\epsilon$. In other words, if the data is generated from an MLN that is fairly smooth in that it does not have large weights, then the distribution learned using the relaxed objective will be fairly close to the one learned using the original objective. We assume that all weights are positive for simplicity of exposition. Note that this is not a limitation of our result since we can easily convert an arbitrary MLN to an equivalent MLN having positive weights by negating each clause having negative weight along with its weight. Formally, if we replace a weighted formula $(f, -\theta)$ by $(\neg f, \theta)$, where $0 < \theta < \infty$ then the distribution represented by the MLN does not change.

Before proving our main result, we first prove that given a setting of parameters $\theta$, the difference between the log-likelihood of the original objective and the relaxed one is bounded by $2 \sum_i \theta_i \epsilon$. Formally,

**Lemma 1**: $|\ell(\theta : \omega) - \hat{\ell}(\theta : \omega)| \leq 2 \sum_i \theta_i \epsilon$. 
Proof.

\[ \ell(\theta : \omega) = \sum \theta_i N_i(\omega) - \log(\sum_{\omega'} \exp(\sum \theta_i N_i(\omega'))) \]

\[ \leq \sum \theta_i (M_i(\omega) + \epsilon) - \log(\sum_{\omega'} \exp(\sum \theta_i M_i(\omega') \exp(-\sum \theta_i \epsilon))) \]

\[ = \sum \theta_i M_i(\omega) + \sum \theta_i \epsilon - \log(\sum_{\omega'} \exp(\sum \theta_i M_i(\omega')) \exp(-\sum \theta_i \epsilon))) \]

\[ = \sum \theta_i M_i(\omega) + \sum 2\theta_i \epsilon - \log(\sum_{\omega'} \exp(\sum \theta_i M_i(\omega')) \exp(-\sum \theta_i \epsilon))) \]

\[ = \hat{\ell}(\theta : \omega) + \sum 2\theta_i \epsilon \]

Similarly we can prove that \( \ell(\theta : \omega) \geq \hat{\ell}(\theta : \omega) - 2 \sum \theta_i \epsilon \)

\[ \square \]

**Theorem 12.** Let \( \alpha = (\alpha_1, \ldots, \alpha_m) \) and \( \beta = (\beta_1, \ldots, \beta_m) \) denote the parameters (weights) that optimize the function \( \ell(\theta : \omega) \) and \( \hat{\ell}(\theta : \omega) \) respectively (\( \alpha \) is the maximum likelihood estimate of \( \theta \)) and let \( \epsilon \) be an upper-bound on the error of the approximate counting algorithm used to estimate \( N_i(\omega) \), then

\[ \ell(\alpha : \omega) - \ell(\beta : \omega) \leq 2\epsilon \sum (\alpha_i + \beta_i) \]
Proof.

\[
\begin{align*}
\ell(\alpha : \omega) - \ell(\beta : \omega) & \\
\leq \hat{\ell}(\alpha : \omega) + \sum_i 2\alpha_i \epsilon - \left( \hat{\ell}(\beta : \omega) - \sum_i 2\beta_i \epsilon \right) \\
= 2\epsilon \sum_i (\alpha_i + \beta_i) - \left( \hat{\ell}(\beta : \omega) - \hat{\ell}(\alpha : \omega) \right) \\
\leq 2\epsilon \sum_i (\alpha_i + \beta_i) = c.\epsilon
\end{align*}
\]

where \( c = 2 \sum_i (\alpha_i + \beta_i) \). The last inequality is due to the fact that \( \beta \) optimizes the objective \( \hat{\ell}(\theta : \omega) \) and hence \( \hat{\ell}(\beta : \omega) \geq \hat{\ell}(\theta : \omega) \) for any \( \theta \).

The power of Theorem 12 is that as the approximate counting algorithm becomes more accurate, the bound specified in Theorem 12 becomes tighter. This allows us to leverage significant advances in approximate counting algorithms to improve the quality of our learning algorithms. Numerous studies have shown that algorithms such as loopy Belief propagation (Murphy et al., 1999) and its generalizations (Yedidia et al., 2005) often yield highly accurate estimates of the counts and we can use them instead in practice. In our experiments, we use IJGP to compute the approximate counts. It should be noted that IJGP is remarkably effective in practice in estimating the partition function. For instance, it won the 2010 and 2012 UAI approximate inference competitions (Elidan and Globerson, 2010, 2011)\(^2\).

\(^2\)Another option is to use sampling based algorithms for computing the counts including rejection sampling and advanced algorithms such as SampleSearch (Gogate and Dechter, 2011), and Appoxcount (Gomes et al., 2007). Unfortunately, these algorithms work with binary domains and are very inefficient at processing multi-valued variables.
6.6.2 Scalable Learning Algorithms using Approximate Counts

In this section, we describe how to scale up the contrastive divergence algorithm using IJGP as the solution counter.

**Algorithm 8: Scalable Contrastive Divergence**

(MLN $M$, $i$-bound for IJGP, Number of samples $N$, dataset or world $\omega$, learning rate $\eta$, update frequency $K$)

- Initialize all weights $\{\theta_j\}$ using a prior.
- /* Compute the data counts */
  foreach formula $f_j$ in $M$ do
    Convert $(f_j, \omega)$ to a constraint network $C_j$
    Compute $M_j(\omega)$ using the algorithm IJGP($i$)
    and store it in a lookup table.
    $\omega^{(0)} = \omega$
- /* Gradient Ascent */
  repeat
    /* Gibbs sampling using Approximate counts */
    for $t = 0$ to $N - 1$ do
      Select a random ground atom $S$
      Compute $\Pr(S|\omega^{(t)}_{\neg S})$ using Equation (6.4)
      Sample a value $S = s$ from $\Pr(S|\omega^{(t)}_{\neg S})$
      Set $\omega^{(t+1)} = (\omega^{(t)}_{\neg S}, S = s)$
    /* Weight update */
    foreach weight $\theta_j$ do
      $\theta_j = \theta_j + \eta(M_j(\omega) - \frac{1}{K} \sum_{t=1}^{K} M_j(\omega^{(t)}))$
  until convergence
  return $\{\theta_j\}$

Our approach for scaling up Contrastive Divergence is described in Algorithm 8. The algorithm takes as input a MLN $M$, an $i$-bound for IJGP, which determines its complexity,
number of samples $N$ for Gibbs sampling, and a dataset $\omega$ (world) and outputs the learned weights. The algorithm has three major steps: (1) computing the data counts, (2) the gradient ascent procedure, and (3) Gibbs sampling which is a sub-step of gradient ascent.

To scale up the computation of the data counts, we first convert each formula into a constraint network and then use IJGP($i$) instead of an exact algorithm to compute the counts $M_j(\omega)$. We store these counts in a look-up table since they will be used at each iteration of gradient ascent.

The most time consuming sub-step in the gradient ascent procedure is Gibbs sampling \cite{Geman and Geman, 1984}. In Gibbs sampling, we start with a complete assignment $\omega^{(0)}$ (world) and in each subsequent iteration $t$, we compute the conditional distribution over a randomly chosen ground atom $S$, given assignments to all other atoms except $S$, denoted by $\omega_{-S}$. To scale up Gibbs sampling, instead of using an exact algorithm to compute the conditional distribution, we use an approximate algorithm. Specifically, the distribution is computed using the following equation:

$$ P(S = s | \omega_{-S}^{(t)}) \propto \exp \left( \sum_{f_j \in MB(S)} \theta_j M_j(\omega_{-S}^{(t)}, S = s) \right) \quad (6.4) $$

where $s \in \{0,1\}$, $MB(S)$ are the formulas in the Markov Blanket of $S$ (namely the formulas that $S$ is involved in), $\theta_j$ is the (current) weight attached to $f_j$, $(\omega_{-S}^{(t)}, S = s)$ is the composition of the assignments $\omega_{-S}^{(t)}$ and $S = s$ and $M_j(\omega_{-S}^{(t)}, S = s)$ is computed by running IJGP($i$) over the constraint network associated with $f_j$ and $(\omega_{-S}^{(t)}, S = s)$.

Note that all CSPs associated with $f_j$ have the same function scopes (set of variables involved in a function is called its scope). Thus, we can use the same cluster graph to run Belief propagation via IJGP over all iterations, indexed by $t$. Moreover, since $\omega^{(t)}$ and $\omega^{(t+1)}$ differ only by an assignment to just one ground atom, only a few function (potential) values will be different between the CSPs of $f_j$ at iterations $t$ and $t + 1$. As a result, we can reuse the messages in iteration $t$ for message passing in iteration $t + 1$. In fact, if we use
Table 6.2. Time and Space Complexities for different learning algorithms (showing their scalability potential). $M$ is the number of gradient ascent steps, $N$ is the number of samples (contrastive divergence via Gibbs sampling) or MaxWalkSAT flips (voted perceptron), $F$ is the number of formulas in the MLN, $A$ is the total ground atoms in the MLN and $T$ be the number of IJGP iterations. $i$ is a parameter that controls the maximum cluster size for IJGP and $w^*$ is the maximum treewidth of the encoded CSPs.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Space Complexity</th>
<th>Time Complexity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Contrastive Divergence</td>
<td>$O(F \exp(\min(i, w^*)))$</td>
<td>$O(MNFT \exp(\min(i, w^* + 1)))$</td>
</tr>
<tr>
<td>Voted Perceptron</td>
<td>$O(F \exp(\min(i, w^*)))$</td>
<td>$O(MNFT \exp(\min(i, w^* + 1)))$</td>
</tr>
<tr>
<td>Pseudo Likelihood</td>
<td>$O(F \exp(\min(i, w^*)) + AF)$</td>
<td>$O(AFT \exp(\min(i, w^* + 1)) + MAF)$</td>
</tr>
</tbody>
</table>

advanced message passing strategy such as the one by Elidan et al. (Elidan et al., 2006), where messages are ordered by how much each message has changed from one iteration to the next, IJGP is likely to converge quickly. We used this advanced ordering strategy in our experiments and found that IJGP converges within 3 iterations in most cases. Moreover, we do not have to calculate a large number of messages because they are guaranteed not to change between subsequent iterations. For example, if all the incoming messages at a cluster have not changed, the outgoing message will be the same as in the previous iteration.

For lack of space, we skip the description of scaling up VP and PLL, noting that they can be similarly extended using the scalable CD algorithm detailed here as a guide. The time and space complexity of the various weight learning algorithms is described in Table 6.2.

### 6.7 Learning Experiments

The fundamental question that we seek to answer through our experiments is, can approximate counting (within learning) improve scalability while learning a highly accurate model? We perform a detailed evaluation to answer the above question and compare the performance of our learning algorithms with those implemented in Alchemy (Kok et al., 2008) and Tuffy (Niu et al., 2011), two state-of-the-art MLN learning systems. We conducted our experiments on three datasets publicly available on the Alchemy website. We used three datasets: WebKB, Entity Resolution (ER) and Protein Interaction (Protein). For sanity
Table 6.3. Dataset sizes. #Evidence is number of true evidence

<table>
<thead>
<tr>
<th>Dataset</th>
<th>#Clauses</th>
<th>#Atoms</th>
<th>#Parameters</th>
<th>#Evidence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smoker</td>
<td>125K</td>
<td>63K</td>
<td>5</td>
<td>7.5K</td>
</tr>
<tr>
<td>WebKB</td>
<td>892 million</td>
<td>20 million</td>
<td>64</td>
<td>227.5K</td>
</tr>
<tr>
<td>Protein</td>
<td>408 million</td>
<td>3.3 million</td>
<td>211</td>
<td>25.8K</td>
</tr>
<tr>
<td>ER</td>
<td>1.7 trillion</td>
<td>5.5 million</td>
<td>15</td>
<td>56K</td>
</tr>
</tbody>
</table>

check, we added another dataset called *Smoker*, that we generated randomly for the Friends and Smokers MLN in Alchemy. Table 6.3 shows the details of our datasets.

### 6.7.1 Methodology

We implemented three different algorithms: Contrastive Divergence (CD), Voted Perceptron (VP) and Pseudo Log Likelihood based learning (PLL). Each algorithm performed approximate counting using IJGP. In each algorithm, we implemented the diagonal newton method ([Lowd and Domingos 2007](#)) to adapt the learning rate dynamically. For all these algorithms, on all four datasets, we performed five-fold cross-validation. In each fold’s test set, we measured the conditional log-likelihood (CLL). The advantage of the CLL is that it directly measures the quality of the probability estimates produced. The CLL of a query predicate is the average log-probability over all ground atoms of the query predicate given the evidence. It should be noted that computing the CLL is also intractable. Therefore, we use Gibbs sampling to estimate the probability of each ground atom corresponding to the query predicate. We collected 100,000 Gibbs samples to estimate each probability.

For approximate counting within our learning algorithms, we used the IJGP algorithm. Specifically, for any formula $f$ and world $\omega$, when to estimate the number of satisfied groundings of $f$ in $\omega$, we run IJGP on the CSP encoding for $(f, \omega)$ for up to 10 iteration.

### 6.7.2 Results

Table 6.4 illustrates our results on the different benchmarks. For each benchmark, we divide the data into 5 sets and use cross-validation to obtain the CLL value for each fold. For each
Table 6.4. Results on benchmark datasets. The average and standard deviation of the CLL score for five-fold cross validation is shown. Alchemy-VP is the Alchemy version of VP, Alchemy-CD is the Alchemy version of CD using Gibbs sampling and Alchemy-MCSAT is a version of contrastive divergence using MC-SAT, Tuffy-MCSAT is the Tuffy version of contrastive divergence using MC-SAT. X means that we ran out of memory when running the algorithm. * indicates that Alchemy-CD gave us an error during weight learning. ($i = \ast$) reports the $i$-bound that gave the best result for our respective algorithms.

<table>
<thead>
<tr>
<th>System</th>
<th>WebKB</th>
<th>Protein</th>
<th>ER</th>
<th>Smoker</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alchemy-VP</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-1.21 ± 0.03</td>
</tr>
<tr>
<td>Alchemy-MCSAT</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>-1.21 ± 0.02</td>
</tr>
<tr>
<td>Alchemy-CD</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>*</td>
</tr>
<tr>
<td>Tuffy-MCSAT</td>
<td>-0.89 ± 0.05</td>
<td>X</td>
<td>-0.698 ± 0.04</td>
<td></td>
</tr>
<tr>
<td>CD</td>
<td>-0.66 ± 0.004 ($i = 2$)</td>
<td>-0.779 ± 0.0002 ($i = 2$)</td>
<td>-0.694 ± 1 × 10^{-6} ($i = 2$)</td>
<td>-0.695 ± 0.01 ($i = 2$)</td>
</tr>
<tr>
<td>VP</td>
<td>-0.91 ± 0.001 ($i = 3$)</td>
<td>-0.78 ± 0.001 ($i = 2$)</td>
<td>-0.693 ± 2 × 10^{-6} ($i = 3$)</td>
<td>-1.16 ± 0.18 ($i = 3$)</td>
</tr>
<tr>
<td>PLL</td>
<td>-0.72 ± 0.001 ($i = 3$)</td>
<td>-0.74 ± 0.0004 ($i = 3$)</td>
<td>-0.693 ± 2 × 10^{-5} ($i = 3$)</td>
<td>-1.61 ± 0.08 ($i = 3$)</td>
</tr>
</tbody>
</table>

Figure 6.4. Convergence plots. (a) shows a plot of convergence time vs. data size for Protein; (b) shows a plot of change in weights vs. time (in hours) for WebKB. ALY corresponds to Alchemy and CDw corresponds to contrastive divergence with $i$-bound $w$.

algorithm, we also put a time bound of 96 hours (4 days) and a memory bound of 8 GB. Table 6.4 shows the average CLL value obtained from the 5 folds along with the standard deviation. We compare the results of our 3 learning algorithms with 3 other algorithms implemented in Alchemy: VP, CD and a variant of CD using MC-SAT instead of Gibbs sampling, as well as the MC-SAT based learning algorithm implemented in Tuffy.
Table 6.5. Effect of IJGP i-bound on convergence. Convergence times for the CD algorithm when using different IJGP bounds for approximate counting. Larger bounds imply larger complexity. X indicates that we could not converge. * indicates that counting is exact as that i-bound is larger than the treewidth of the formulas.

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>i-bound</th>
<th>CLL</th>
<th>Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ER</td>
<td>1</td>
<td>-0.693</td>
<td>653</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-0.693</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>3*</td>
<td>-0.693</td>
<td>X</td>
</tr>
<tr>
<td>WebKB</td>
<td>1</td>
<td>-0.788</td>
<td>92,327</td>
</tr>
<tr>
<td></td>
<td>2*</td>
<td>-0.664</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>3*</td>
<td>-1.686</td>
<td>X</td>
</tr>
<tr>
<td>Protein</td>
<td>1</td>
<td>-0.9948</td>
<td>178.2</td>
</tr>
<tr>
<td></td>
<td>2*</td>
<td>-0.7797</td>
<td>293</td>
</tr>
<tr>
<td></td>
<td>3*</td>
<td>-0.7795</td>
<td>260</td>
</tr>
</tbody>
</table>

As seen in Table 6.4, we could only run Alchemy on the Smoker dataset, as it ran out of memory while creating the ground Markov network from the data for all other datasets. For the same reason we could only run Tuffy on WebKB and Smoker only. Among CD, VP and PLL, the results were fairly similar for WebKB, Protein and the ER datasets. CD was marginally better (larger CLL) than VP and PLL on these datasets. For the smokers problem, CD was much better than VP or PLL. The Alchemy learning algorithms performed
similar to VP and PLL for the Smoker dataset. However, we could not obtain the results for Alchemy-CD since the weights it learned were infinite for the Smoker dataset. In the case of Tuffy, our CD algorithm performed better than Tuffy (which uses MC-SAT as its inference subroutine) in both WebKB as well as the Smoker datasets. VP performed marginally worse than Tuffy for WebKB and PLL performed marginally better than Tuffy for WebKB. For the Smoker dataset, Tuffy performed better than VP and PLL.

As mentioned in earlier sections, we can control the space/time complexity in IJGP using a parameter \(i\)-bound that specifies the maximum cluster-size of the join graph that IJGP operates on. The complexity of learning increases as we increase the value of \(i\)-bound. Table 6.5 illustrates the results of using different \(i\)-bound values in IJGP with the CD algorithm. For larger \(i\)-bounds, the counts are exact (indicated by a * in Table 6.5). We can see that for the WebKB and ER datasets, for \(i\)-bound \(> 1\), the weights did not converge. For the Protein dataset, the weights converged for all values of \(i\)-bound. For \(i\)-bound \(= 1\), the algorithm converged very quickly on all three datasets. Furthermore, the CLL scores did not vary by much as we increase the \(i\)-bound. Thus, the difference in results when using approximate counts is almost as good as the results obtained using exact counts. Empirically, this suggests that we only need a “reasonably good estimate” of the partition function from each encoded constraint network to move along the correct gradient direction.

We further illustrate the scalability and accuracy of our approach through a set of 4 experiments shown in Figures 6.4 and 6.5. Owing to space constraints, we only show results for the Protein and WebKB datasets with the CD algorithm.

In Figure 6.4 (a), we vary the size of the training data by sub-sampling the original datasets according to the parameter \(k\) \((0 \leq k \leq 1)\) and plot the convergence times for different data-sizes. \(k\), known as Sampling fraction, is the ratio of sample size to population size. As Figure 6.4 (a) shows, Alchemy cannot scale up beyond \(k = 0.1\), Tuffy on the other hand is slightly more scalable and stops after \(k = 0.3\). In contrast our approach can easily
scale up to all values of $k$. Even more importantly, it converges much faster than Tuffy and Alchemy as seen for small $k$. Figure 6.4 (b) shows similar results and the weights learned by our approach converges rapidly while the variance in the weights learned by Tuffy is much larger. Further, as shown in our results, even for lower $i$-bound values (which means increased scalability), the convergence times during learning are fairly comparable.

Finally, Figure 6.5 illustrates the accuracy of our approach as compared to Alchemy and Tuffy. Fig 6.5 (a) shows the CLL for different sizes of training data (controlled using the parameter $k$). Similar to the previous result, Alchemy cannot process training data beyond $k = 0.1$ while Tuffy fails after $k = 0.3$. In contrast our approach is much more scalable. At the same time, the accuracy of our approach is higher as compared to Tuffy/Alchemy. Further, as shown in our results, using approximate counting ($i$-bound is less than treewidth) the learning accuracy is quite comparable to the accuracy obtained using exact counting ($i$-bound equal to treewidth). This proves, apart from our theoretical result, that the accuracy of our approach is not diminished as a result of our approximate counting. Similar results are observed in Figure 6.5 (b) where we illustrate CLL vs. training time.

6.8 Conclusion

For large MLNs a sub-step, which is typically a bottleneck, in several inference and learning algorithms is “counting the true groundings of a first-order formula in a possible world.” We proposed to solve this counting problem by encoding it as a CSP solution counting problem, thereby allowing us to exploit numerous advances, guarantees, principles and techniques in the active research area of CSP and probabilistic graphical models. We build on top of this encoding based approach by first developing a junction tree based exact CSP solution counting algorithm and applied it to two widely used MLN inference techniques, Gibbs sampling, and MaxWalkSAT, both of which require an answer to a dynamic version of the counting problem at each iteration. Our experiments with these algorithms on a wide variety
of MLN benchmarks with large domain sizes clearly showed that our approach was orders of magnitude more scalable than existing state-of-the-art inference systems. Then, to scale-up the learning task, we presented a principled approach using approximate counting for scaling up weight learning to larger datasets than is possible today. We empirically and theoretically showed the power of our approach and that it is not only orders of magnitude more scalable but also much more accurate when compared to state-of-the-art weight learning algorithms implemented in MLN software packages such as Alchemy and Tuffy.
CHAPTER 7

EXPLOITING SHARED STRUCTURE FOR EFFICIENT INFERENCE IN MLNS

7.1 Introduction

So far in this dissertation, we described research on inference algorithms for MLNs having tied parameters. In other words, we assumed that all groundings of a first-order formula have the same weight. The tied parameter assumption has enabled us to design many scalable inference approaches. However, for many practical application domains such as information extraction, this assumption is too strong; the induced models have too few parameters and are not rich enough to model complex dependencies in the application domain. For example, in order to specify that a word in a web-page determines the topic of a web-page, we can use the formula $\text{Word}(w, p) \Rightarrow \text{Topic}(p, t)$ and attach a weight $\alpha$ to it. However, this model is unrealistic because it assumes that the probability of a topic given a word is the same for all topics and words. A more plausible model will assign a different probability for each word-topic pair, and thus use $m \times n$ different weights, where $m$ is the number of words and $n$ is the number of topics. In MLN, this is represented using the ‘+’ operator. When a ‘+’ sign is attached to a logical variable, we learn a different weight for each grounding of the variable. In our example, we will represent the formula as $\text{Word}(+, w, p) \Rightarrow \text{Topic}(p, +t)$, and learn a different weight for each word-topic pair.

In this chapter, we present novel algorithms for scaling up inference in MLNs having ‘+’ operators. The key idea in our new approach is to extend our approach in chapter 6 to solve a key sub-task in inference algorithms, finding the weight of a world. Specifically, we make the following contributions.
1. We develop a graphical model encoding for MLN formulas with ‘+’ variables and prove that our new encoding is exact in that the partition function of the graphical model is equal to the sum of the weights of the satisfied groundings in a given world, and that our graphical model encoding is guaranteed to be more efficient as compared to the encoding proposed in the previous chapter (Venugopal et al., 2015). When the treewidth of the graphical model is small, inference is computationally efficient.

2. We develop an approximate graphical model encoding for MLN formulas with ‘+’ variables which is useful in cases when the treewidth of the exact encoding is large. The key idea is to cluster together groundings of a formula having similar weights, which yields a graphical model having smaller treewidth.

We evaluate our exact and approximate encoding on different MLN benchmarks and compare the effectiveness of our encoding with the encoding proposed by in the previous chapter. We show that new encoding substantially improves the scalability, convergence and accuracy of Gibbs sampling and MaxWalkSAT.

7.2 Exact Encoding for ‘+’ variable formulas

In Chapter 6 we have used constraint network to solve the #SG problem. In this section, we extend this approach to the #WSG problem for untied MLNs. We define this problem next.

Definition 10. Given a collection of (propositional) features \((g_j, w_j)\) which are groundings of a first-order formula \(f\), the #WSG problem is to compute \(\sum_j w_j g_j(\omega)\) for a given world \(\omega\), where \(g_j(\omega) = 1\) is \(g_j\) is true given \(\omega\).

Similar to the approach presented in chapter 6, we will create a dynamic Markov Random Field (dMRF)\(^1\) corresponding to each Markov Logic formula and show that the partition

\(^1\)We refer to the constructed MRF as dynamic because the potential entries depend on the given world \(\omega\).
function of the dMRF can be used to solve the \#WSG problem. It should be noted that our approach generalizes the aforementioned methods as a constraint network can be considered as an MRF whose potential values are either zero or one.

We demonstrate our proposed dMRF encoding on the following simple MLN having just one clause \( f = \forall x, \forall y, \forall z \neg R(+x, y) \lor S(y, +z) \). The domain of each logical variable is \( \{A, B\}^2 \). The formula has a different parameter (instead of only one parameter) corresponding to different groundings of the variable \( x \) and \( z \) (denoted by \( \theta_{x,z} \)). Instead of counting the number of false groundings of \( f \) (as done by Venugopal et. al.), we will focus on computing the total weight of false groundings of \( f \) given \( \omega \). Moreover, we can easily calculate the total weight of true groundings from it, which is equal to the total weight of all possible groundings minus the total weight of false groundings.

We create a Markov Random Field having three random variables, each corresponding to the logical variables \( x, y, z \), and having three potentials, \( \phi_1(x, y) \), \( \phi_2(y, z) \), and, \( \phi_3(x, z) \). Here, \( \phi_1(x, y) \) and \( \phi_2(y, z) \) are exactly the same as \( \phi_1 \) and \( \phi_2 \) respectively in Figure 6.1. For \( \phi_3(x, z) \) each potential entry indexed by \( x, z \) is equal to the parameter \( \theta_{x,z} \). Figure 7.1 shows the above setup.

Notice that if we take a product of the three functions \( \phi_1(x, y) \), \( \phi_2(y, z) \), and, \( \phi_3(x, z) \), then the resulting function \( \phi(x, y, z) \) will have \( \theta_{x,z} \) associated with an entry \( (x, y, z) \) iff \( R(x, y) \) is true and \( S(y, z) \) is false. Since the weight of a ground formula, indexed by \( (x, y, z) \), is given by \( \theta_{x,z} \) and it evaluates to false iff \( R(x, y) \) is true and \( S(y, z) \) is false, by extension \( \phi(x, y, z) = \theta_{x,z} \) implies that the ground formula \( (x, y, z) \) is false. Therefore, we can compute the total weight of groundings of \( f \) that evaluate to false, by simply summing all the entries in \( \phi(x, y, z) \), which is the same as computing the partition function of the constructed MRF.

To compute the total weight of true groundings we need to compute the total weight of all groundings minus the total weight of false groundings.

\(^{2}\)This is the same MLN as in Figure 6.1 with the ‘+’ operator added
Figure 7.1. (a) dMRF associated with \( f = \forall x, \forall y, \forall z \neg R(x, y) \lor S(y, z) \). The domain of each logical variable is \{A, B\}; (b) and (c): Constraints \( \phi_1(x, y) \) and \( \phi_2(y, z) \) corresponding to the possible world in which the ground atoms \( R(A, B) \), \( R(B, A) \), and, \( S(A, B) \) are true and the rest are false. (d) \( \phi_3(x, z) \) corresponds to the parameters \( \theta_{x,z} \) associated with the formula possible groundings, which in this case, equals \(|\Delta(y)| \times \sum_{x,z} \theta_{x,z} \). Subtracting the obtained partition function from this quantity will give us the total weight of true groundings.

Next, we will formalize this intuition and precisely define how to encode the \(#\text{wsg}\) problem as a partition function computation problem.

**MRF Encoding** Given a first-order clause \( f \) and a world \( \omega \), the corresponding dMRF \( G \) has a variable for each (universally quantified) logical variable in \( f \). The domain of each variable in \( G \) is the set of constants in the domain of the corresponding logical variable. For each atom \( R(x_1, \ldots, x_u) \) in \( f \), we have a potential \( \phi \) in \( G \) defined as follows:

\[
\phi(\bar{x}_u) = \begin{cases} 
\omega_R(X_1, \ldots, X_u) & \text{if } R \text{ is negated in } f \\
-\omega_R(X_1, \ldots, X_u) & \text{Otherwise}
\end{cases}
\]

where \( \bar{x}_u = (x_1 = X_1, \ldots, x_u = X_u) \) denotes an assignment to the dMRF variables and \( \omega_R(X_1, \ldots, X_u) \) is the projection of the world \( \omega \) on the ground atom \( R(X_1, \ldots, X_u) \), namely the truth-value of the ground atom \( R(X_1, \ldots, X_u) \) in \( \omega \). Finally if \( x_i, \ldots, x_j \) are variables associated with the ‘+’ operator and if the corresponding parameters are \( \theta_{x_i, \ldots, x_j} \) then we have a potential \( \psi \) in \( G \) defined as:

\[
\psi(x_i, \ldots, x_j) = \theta_{x_i, \ldots, x_j}
\]
By generalizing the arguments presented for the example MLN given above, we can show that:

**Theorem 13.** Let \( f \) be a first-order clause, \( x_1, \ldots, x_u \) be the (universally quantified) logical variables in \( f \), \( X_+ = \{ x_i, \ldots, x_j \} \) be the set of logical variables associated with the ‘+’ operator, \( \theta = \{ \overline{x}_+ \} \) is the set of parameters attached to \( f \) \( , \omega \) be a world and let \( Z_G \) denote the partition function of the dMRF \( G \) obtained from \( (f, \omega) \) using the MRF encoding. Then, \( W_f(\omega) = \prod_{x \in X_+} |\Delta(x)| \cdot \sum_{\overline{x}_+} \theta_{\overline{x}_+} - Z_G \) where \( \Delta(x) \) is the set of constants in the domain of \( x \).

Since we have reduced the \#wsg problem to the partition function calculation problem, it is clear that we can use any graphical model inference algorithm and leverage its advances and guarantees to compute the former efficiently. For example, we can use the junction tree algorithm for solving \#wsg. The time and space complexity of this algorithm will be exponential in the tree width of \( G \). The following theorem is straight forward and follows from the fact that the tree-width of dMRF is less than number of variables in the dMRF.

**Theorem 14.** The time and space complexity for solving \#wsg using junction tree algorithm is no-worse (i.e., at least same or better) than solving \#sg using same algorithm.

Since, many local-search based algorithm like MaxWalkSAT and Gibbs sampling require to solve \#wsg at very iteration we can efficiently implement them using our MRF encoding.

### 7.3 Approximate Encoding for ‘+’ variable formulas

In the encoding described in the previous section, we added a new potential to account for the fact that different groundings corresponding to the “+” variables have different weights. Note that, if all the weights are identical, then we could simply drop the weight potential from the graphical model encoding; this in turn can potentially reduce the treewidth. Similarly,
if a subset of weights are almost equal (to each other), then, in principle, we could create a graphical model for that subset of formulas without adding an additional weight potential. Here, we exploit this idea by creating clusters of formulas having similar weights.

We first cluster together objects corresponding to “+” variables of a formula based on whether they appear in groundings that have approximately similar weights. This naturally induces a clustering over the set of ground formulas. We then assume that weight of all the ground formulas that are in the same cluster is equal to the average weight of the formulas within that cluster. Thus, we can simply drop the weight potential in each graphical model encoding for the clustered formulas, which means that the treewidth of each graphical model is guaranteed to reduce or stay the same.

We illustrate our idea with a simple example. Let us assume that we have the following simple MLN with only one formula $f = \forall x, \forall y, \forall z \neg R(+x, y) \lor S(y, +z)$. The domain of each logical variable is $\{A, B, C, D\}$. There are 16 different parameters associated with $f$, each corresponding to a full assignment to $(x, z)$. Now let’s assume that $\theta_{A,A}, \theta_{A,B}, \theta_{B,A}, \theta_{B,B}$, have the same value (say $\theta$), in that case we can combine the four (partially) ground formulas associated with these parameters into a single formula $f' = \forall x', \forall y, \forall z' \neg R(x', y) \lor S(y, z')$ having a single parameter $\theta$ and $\Delta(x') = \Delta(z') = \{A, B\}$ and $\Delta(y) = \{A, B, C, D\}$. Hence, instead of having 16 parameters, we will have only 13 parameters. Observe that we can combine the ground formulas to $f'$ because the indices of the parameters induce a partition on the domain of $x$ and $y$, and all the indices of the parameters can be expressed as a Cartesian product of these two partitions. We will extend this intuition further to create partitions of the domains of $x, y$ such that the Cartesian product of each member of those partitions defines the indices of parameters belonging to a single cluster. All the parameters belonging to a cluster can be replaced by a single parameter (which is the mean of all the parameter values).

Formally, we are interested in finding the optimal joint clustering over all the “+” variables. We can define this as follows:
Let $f$ be a first-order clause, $x_1, \ldots, x_u$ be the (universally quantified) logical variables in $f$, $\mathcal{X}_+ = \{x_i, \ldots, x_j\}$ be the set of logical variables associated with the ‘+’ operator, and $\theta_{\mathcal{X}_+}$ are the set of parameters associated with different assignment to the variables in $\mathcal{X}_+$. For each variable $x_l \in \mathcal{X}_+$ we want to partition its domain $\Delta(x_l)$ into $k_l$ partitions $\mathcal{P}(l) = \{P_{l1}, \ldots, P_{k_l}\}$ such that the following objective function is minimized,

$$
\sum_{P^{(i)} \in \mathcal{P}(i)} \ldots \sum_{P^{(j)} \in \mathcal{P}(j)} \sum_{\theta_{\mathcal{X}_+} \in C_{\rho_{\mathcal{X}_+}}} ||\theta_{\mathcal{X}_+} - \mu_{C_{\rho_{\mathcal{X}_+}}}||^2_2
$$

(7.1)

where $\mathbf{X}_+ = (x_i = X_i, \ldots, x_j = X_j)$ is a complete assignment to the variables in $\mathcal{X}_+$, $\rho_{\mathbf{X}_+} = (P^{(i)}, \ldots, P^{(j)})$ is the partition tuple corresponding to $\mathbf{X}_+$ (i.e., $X_i \in P^{(i)}$ etc.), a cluster of parameters $C_{\rho_{\mathbf{X}_+}}$ is a set of parameters $\{\theta_{\mathbf{X}_+}| \rho_{\mathbf{X}_+}$ is the partition tuple corresponding to $\mathbf{X}_+\}$, and $\mu_{C_{\rho_{\mathbf{X}_+}}}$ is the average of all parameters in $C_{\rho_{\mathbf{X}_+}}$.

Solving the joint clustering problem is computationally hard (). Therefore, a naive approach which searches for all possible partitions of a given domain is not scalable since the number of possible partitions of size $k$ for a set of size $n$ (given by the so-called Stirling numbers of the second kind) grows exponentially with $n$. Hence, to solve this optimization
problem we propose a greedy approach which is inspired by the \textit{k-means} algorithm. Our method is described in Algorithm 9.

\begin{algorithm}
\caption{Cluster-Param($\theta_{x_+}$)}
Create an ordered set $O$ from $x_+$.
\begin{algorithmic}
\Procedure{}{}
\ForAll{$x \in O$}
\ForAll{domain value $d$ of $x$}
Randomly assign it to a cluster center.
\EndFor
\EndFor
\Repeat
\ForAll{$x \in O$}
Given cluster assignments to the rest of the variables fixed, find cluster centers for $x$ that minimizes Equation 7.1.
\EndFor
\Until{cluster assignment does not change}
\EndProcedure
\end{algorithmic}
\end{algorithm}

The algorithm begins by first selecting an order of variables. Then for each variable, we assign their domain values to some random cluster. Then assuming that the cluster assignment to the domains of all other variables remaining constant we determine the best possible cluster assignment to the domains of a variable selected from the variable ordering. We repeat the process until convergence.

\textbf{Theorem 15.} Algorithm 9 is guaranteed to converge to a local optima.

\textit{Proof.} Since for each variable’s cluster assignment always reduces Equation 7.1, Algorithm 9 is guaranteed to reach some local optima. \hfill \Box

Once we have obtained the joint-clusters, we can use Algorithm 10 to create a smaller number of constraint networks for the formula. The basic ida behind Algorithm 10 is that we can use the obtained partitions to ground the MLN formula with respect to them, and create a smaller number of (partially) ground MLN formulas, each of these formulas have a single weight attached to it, which is the mean (cluster center) of the clustered parameters.
These partially ground formulas now can be converted to a constraint-network by using the f-to-CSP method described in Chapter 6.

**Algorithm 10: Partition-Network**

(MLN function \( f \), Set of partitions \( \{ P^{(i)} \} \))

Let, \( \mathcal{G} = \{ \} \).

Let, \( \mathcal{X}_+ = \{ x_1, \ldots, x_m \} \) be the set of ‘+’ variables

Create set of ordered tuples \( \mathcal{O} = \{(P^{(1)}_{i_1}, \ldots, P^{(m)}_{i_m}) \} \) from \( P^{(1)} \times \ldots \times P^{(m)} \).

foreach \( (P^{(1)}_{i_1}, \ldots, P^{(m)}_{i_m}) \) in \( \mathcal{O} \) do

Create a formula \( f' \) by replacing each \( x_i \in \mathcal{X}_+ \) by a new logical variable \( x'_i \) such that \( \Delta(x'_i) = P^{(i)}_{i_i} \).

The weight of \( f' \) is given by \( \mu_{\mathcal{C}(P^{(1)}_{i_1}, \ldots, P^{(m)}_{i_m})} \).

Create a constraint network \( G \) for \( f' \) using f-to-CSP encoding.

Add \( G \) to \( \mathcal{G} \).

return \( \mathcal{G} \)

After each MLN formula is converted into a small set of constraint networks (by running Algorithm 10) we can compile each of these networks into Junction Trees for efficient counting. Note that we need to run the Algorithm 9 and 10 only once per formula and once we have compiled the Junction Trees are created we do not need to store the (partially) ground MLN formulas (or the constraint network).

### 7.4 Experiments

#### 7.4.1 Setup

We evaluate the graphical model encodings proposed in this chapter by using it within two inference algorithms. We use Gibbs sampling to compute marginal probabilities and MaxWalkSAT for MAP inference. Specifically, we perform the counting sub-step within both the inference algorithms (as detailed in the previous chapter) using our new graphical model encodings for the counting problem. We compare our exact encoding as well as approximate
encoding methods with the encoding approach proposed in the previous chapter (we refer to this as ‘#sg’). We conducted our experiments on the following three datasets:

(i) **Student** MLN having the formula \(\neg \text{Student}(x, +p) \lor \neg \text{Publish}(x, z) \lor \text{Cited}(z, +u)\)

(ii) **WebKB** MLN from the Alchemy web page

(iii) **Citation Information-Extraction** (IE) MLN form the Alchemy web page

Through our experiments we want to answer the following questions:

- Do our new encodings of the counting problem lead to more scalable inference algorithms as compared to state-of-the-art?

- How accurate is our approximate encoding in terms of inference results?

In our experiments, we implemented our system on top of the publicly available Magician system ([Venugopal et al., 2016](#)) that uses #sg. That is, we used the same counting subroutines as is used by Magician, but replaced the graphical model encoding used in Magician with our proposed exact and approximate encoding methods respectively.

**Results for MaxWalkSAT**

Figure 7.2 shows the performance of our approach in comparison with other methods. Here we have reported the ‘cost’ for the MAP objective function (the cost is typically reported when the MAP problem, which is a maximization problem, is solved as a minimization problem). The better the cost the better the performance of a solver. The cost is computed from the MAP tuple reported by each solver. Each solver was given 200 seconds for the dataset, which includes the time to ground/ pre-process the system. We have tested four of our solvers against ‘#sg’. They are ‘#wsg’, which is the exact encoding approach and three different ‘C-pct’s which are obtained by applying our clustering based encoding where the number of clusters is \(pct\%\) of the domains.
Figure 7.2. Cost vs. Time: Cost of unsatisfied clauses (smaller is better) vs. time for different domain sizes. Notation used to label each figure: MLN(numvariables, numclauses). Note: the quantities reported are for ground Markov network associated with the MLN. Standard deviation is plotted as error bars.

For all the datasets, all of our approaches are better than ‘#sg’. For the student datasets ‘#wsg’ outperforms ‘C-90’. However, ‘C-10’, which compresses the parameter space to 10% of the original, gives the best answer in all these datasets. This is because, as we have fewer clusters, our encoding yields a graphical model where counting becomes faster, which in turn can perform more flips/second result in a better exploration of the state-space for MaxWalkSAT.

Results for Gibbs Sampling
For Gibbs sampling, we evaluate the convergence of the sampler and the accuracy of the computed marginals.

For measuring convergence, we use the Gelman-Rubin (G-R) Statistic (Gelman and Rubin, 1992). For a well-mixed sampler, the G-R statistic should ideally decrease over time illustrating that the MCMC chain has mixed. To compute the G-R statistics, we set up 5 Gibbs samplers from random initialization points and measure the within chain and across chain variances for the marginal probabilities for 1000 randomly chosen query ground atoms. We compute the G-R statistics for each of the 1000 query atoms and measure the mean G-R statistic.

For measuring accuracy, we use the KL-divergence between the average true marginal probabilities of query ground atoms and the average approximate marginal probability computed using our encodings within the Gibbs sampler. Note that obtaining the true marginal probabilities is infeasible for our benchmarks in the case where we have arbitrary evidence. Therefore, for this experiment we did not use evidence in any of our benchmarks, in which case, the average true marginal probability is equal to 0.5.

Figure 7.3 compares the convergence of the Gibbs sampler for \#sg as well as different clustering based approximations. The labels for the clustering based encoding results are similar to that in the MaxWalkSAT graphs. As we see from these results, \( C - 10 \) has the best convergence and as we increase the number of clusters, the convergence gradually becomes worse. This is because, as we have fewer clusters, our complexity of counting goes down due to our encoding, and we can therefore in a fixed time interval, we can draw more samples which results in better convergence.

Figure 7.4 compares the accuracy of the Gibbs sampler for \#sg as well as different clustering based approximations. As in the previous case, for smaller number of clusters, we get higher accuracy within a fixed time period, since we can collect more samples/second. As we increase the number of clusters, for the same time interval we collect fewer samples, and therefore, the accuracy of computing the marginals is reduced.
In both Figure 7.3 and Figure 7.4, the performance of \#WSG was almost identical to C-90. For readability, we do not show the curves for \#WSG.
7.5 Conclusion

In this chapter, we developed two novel graphical model encodings for an MLN formula that has different weights for different subsets of its groundings. Both these encodings are useful in solving a computationally complex counting problem that manifests itself in several sampling and local-search based MLN inference algorithms. Namely, counting the sum of
weights of the groundings of an MLN formula that are satisfied in a given world (assignment to all variables in the MLN). The first encoding was an exact encoding such that the partition function of the graphical model corresponds exactly to the number of satisfied groundings in the MLN formula. The second was an approximate encoding that clustered formula groundings with approximately similar weights together, which in several cases, reduces the treewidth of the encoded graphical model. We showed through our experiments that our new encodings used within inference algorithms such as Gibbs sampling and MaxWalkSAT result in more scalable and accurate inference results.
CHAPTER 8
CONCLUSION

In this chapter, we conclude the dissertation by outlining our major contributions and providing the future directions that can further improve our investigation.

In this dissertation, we focused on Markov Logic Networks (MLNs) that combine first-order logic with probabilistic graphical models. MLNs have become increasingly popular to model complex applications due to their rich representational power. However, inference remains a key challenge for MLNs. Although there have been recent developments for scaling up inference in MLNs, the primary focus of the existing research is on marginal inference, and algorithms for optimization task like MAP inference are far less advanced. In the dissertation, we advance the state-of-the-art in MAP inference for MLNs by presenting fast, scalable and accurate inference algorithms that exploit logical structure in MLNs.

8.1 Contribution

The main contribution of this dissertation is novel, scalable algorithms for MAP inference in MLNs. The most popular approach for existing state-of-the-art MLN systems is propositional inference. These algorithms convert the MLN into a ground Markov Network. Unfortunately, this method is not scalable as a ground MLN can be massive even for simpler problems. To address this grounding problem recently many “lifted inference” algorithms have been proposed. These algorithms exploit symmetries found in MLN to achieve scalability. However, most of these algorithms suffer from following fundamental limitations:

- Designing new lifted inference algorithms require significant modifications to propositional algorithms and for optimal performance require lifting several steps of proposi-
tional algorithms. This especially affects optimization tasks like MAP inference as they need efficient heuristics (like variable ordering, upper/lower bounds) for scalability. As a result developing new efficient lifted algorithms is time-consuming.

- Most lifted inference rules are sound, but not complete as they exploit only exact symmetries. However, in most practical MLNs such symmetries do not exist. Hence any lifted inference algorithm, in the worst-case, will partially ground these MLNs and thus fail to scale-up.

- Most lifted inference algorithms cannot handle evidence as evidence breaks symmetries. Hence for these cases lifted inference algorithms cannot be used.

- Finally, most lifted algorithms only consider MLNs with tied parameters, i.e., all groundings of an MLN formula have the same weight. This assumption is seldom satisfied in most real world problems, and lifted inference algorithms, once again, are not efficient for these problems.

In this dissertation, we addressed all these limitations and designed novel scalable algorithms which significantly improved the state-of-the-art.

Our first contribution is the introduction of the “lifting as pre-processing” paradigm. The key idea behind this is to construct a “lifted” Markov network such that the MAP solution in this lifted network corresponds to the MAP solution in the MLN, and its size is much smaller than the ground Markov network. The main advantage of this approach is that we can use any existing off-the-shelf propositional inference algorithm to obtain a new lifted inference algorithm. We applied this method to obtain two new lifted MAP inference algorithms. First we identified a tractable subset of MLNs, called “Non-Shared MLN”, where the time complexity for MAP inference is in fact domain independent. Our key result stems from the observation that one of the MAP assignment for these MLNs can be
found using the “uniform assignment”, i.e., all ground atoms of a predicate are either all true or all false. We then provided two lifted MAP implementations, one where we have used exact Integer Linear Program based solver and another where we used approximate local search based solver. Our evaluation on a wide variety of datasets shows that our approach is significantly faster than existing state-of-the-art systems.

In our next lifted MAP algorithm (Sarkhel et al., 2014a), we build on top of the lifting as pre-processing paradigm, as well as the probabilistic theorem proving approach (Gogate and Domingos, 2011), a classic search-based lifted inference technique. The key idea in our approach is to reduce the lifted MAP inference problem to an equivalent Integer Polynomial Program (IPP). Each variable in the IPP potentially refers to an assignment to a large number of ground atoms in the original MLN. Hence, the size of the search space of the generated IPP can be significantly smaller than the ground Markov network. To solve the generated IPP we convert it to an equivalent Integer Linear Program (ILP) using a classic conversion method outlined in (Watters, 1967). We experimentally evaluate our approach on MLNs from different real-world problems. We demonstrate that our approach is faster and more accurate than not only existing propositional algorithms but also our previous lifted MAP method.

Our next contribution is an any-time and any-space approximate lifted MAP algorithm (Sarkhel et al., 2015) which exploits approximate symmetries in the first-order structure. To achieve this, we propose a new operator, “partition-ground”, which replaces the computationally expensive partial grounding step. Our approach is straight-forward: partition the ground atoms into groups and force the inference algorithm to treat all atoms in each group as indistinguishable (symmetric). We prove that our proposed approach yields a consistent assignment that is a lower-bound on the MAP value. We show that the quality of the MAP solution can be improved systematically by refining the partitions. We further demonstrate how to improve the complexity of our refinement procedure by exploiting the
exchangeability property of successive refinements. Specifically, we show that we can arrange the exchangeable refinements on a lattice, and can search via a heuristic search procedure to yield an efficient any-time, any-space algorithm for MAP inference. We show experimentally that our method is highly scalable and yields close to optimal solutions in a fraction of the time as compared to existing approaches.

Next, we consider MLNs in which lifted inference is not feasible. For these MLNs we take a new approach to scalability. We first consider a fundamental operation, the problem of evaluating the weight of a given world (i.e., counting the number of true groundings of a formula in a world) of an MLN. For many local-search based inference algorithms (e.g., MaxWalkSAT, Gibbs sampling) this step is very crucial and needs to be carried out at every iteration. Although this problem is known to be computationally intractable, existing MLN systems solve it using the following naive “generate-and-test” approach: generate each possible grounding and test whether it is true in a given world. This naive approach is the main reason for their poor scalability in these inference methods. We propose a novel, practical approach for solving this counting problem by encoding each formula as a CSP (Constraint Satisfaction Problem). Our approach never grounds the full MLN and in most cases is orders of magnitude faster than the “generate-and-test” approach. However, the most desirable characteristic of our encoding is that it enables us to leverage algorithms from graphical models and CSP literature along with their guarantees and approximations, for solving the counting problem efficiently. We use this advanced counting technique to implement efficient, scalable approximate inference [Venugopal et al., 2015] and learning algorithms [Sarkhel et al., 2016]. Our experiments clearly show that in several real-world MLN benchmarks, our approach is orders of magnitude more scalable than existing state-of-the-art systems.

Finally, we consider the most general and practical MLNs, i.e., MLNs without the shared parameter assumption. We show that even in these general settings we can achieve scalability.
We proposed two different approaches for this realistic setting. First, inspired by our work on advanced counting techniques, we propose an exact encoding which converts the counting problem to the partition function computation problem on an encoded MRF. We show that this encoding is no worse than our CSP based encoding, and in some cases, can be significantly faster than the previous encoding. Next, we present an approximate approach which clusters all possible groundings of an MLN formula to create a few MLN formulas. Each of these clustered formulas is assumed to have a single weight attached to them. This simplifying assumption helps us reduce the time complexity of the counting problem even further. We show through our experiments that our new encodings, when used within inference algorithms such as Gibbs sampling, and, MaxWalkSAT, result in more scalable and accurate inference results.

8.2 Directions for Future Research

Our research leaves plenty of room for improvements that can be pursued in the future.

8.2.1 Scaling up Weight learning for formulas with Shared structure

In Chapter 7 we considered the MLN formulas with shared structure but with different weights. These kind of formulas are quite popular among many Markov Logic application researchers. In many applications often tens of thousand of weights are attached to a single formula. This practice makes the learning task infeasible as at present there is no algorithm which can exploit the shared structure for faster learning. Moreover, adding many parameters increases the variance of the learned model significantly and requires a significant amount of data for learning them accurately.

We can easily extend our research on scalable inference (Chapter 7) to the parameter learning task. Since there are two distinct encoding approaches proposed in that chapter, we can have two very different parameter learning methods. First, we can extend our scalable
weight learning method (Sarkhel et al., 2016) with the proposed exact encoding to obtain a scalable learning approach for formulas with shared structure. However, the approximate encoding method cannot be extended in a straightforward way. One of the greatest challenges of using approximate encoding is its dependence on the clustering step. To perform the clustering step, we need to know the actual weights of each ground formula (which is the task we are trying to scale up).

One natural solution to this problem would be to randomly group formulas, instead of the clustering step. The weight learning can be achieved efficiently in this setting. However, the random grouping of formulas can introduce a significant bias to the learned model. We can easily remedy this high bias by learning a mixture of MLN, where each of the mixture components has randomly grouped formulas. Since each of the mixture components can be learned separately, this approach will allow us to achieve true scalability for the weight learning task in most practical MLNs.

8.2.2 Learning Feasible Structures

Structure learning task is a much harder problem than the weight learning problem. Here, the task is to learn formulas as well as its associated weights from the data. One popular approach for achieving scalable structure learning is to learn the so-called “tractable models” (Domingos and Webb, 2012), i.e., learn models where inference is tractable. However, these models lack the expressiveness of MLNs and can be difficult to use organically in many application domains.

In one of our recent papers (Chou et al., 2016), we have proposed a systematic approach to decrease the number of parameters in a graphical model by tying some of them together. Although this method was originally intended to reduce bias in learning it can be easily extended to learn structures as well. The basic idea in our paper was to cluster the learned weights to decrease the number of parameters. We can combine the features belonging to
a single cluster to obtain a new feature. This combining of features allows us to propose following recursive structure learning algorithm. We start with unit clauses (i.e., a single predicate). We learn weights of these clauses and then use clustering and combining operations together to obtain a new set of clauses. We repeat this process for a predetermined number of steps. Here, the weight learning step can be performed using our scalable learning approach \cite{Sarkhel2016}. 
REFERENCES


VITA

Somdeb Sarkhel earned his Bachelor of Engineering (B.E.) degree in Computer Science and Technology from the Indian Institute of Engineering Science and Technology (formerly Bengal Engineering and Science University) in August 2005. He entered the Ph.D. program in the Fall of 2011 at The University of Texas at Dallas (UTD) in the Statistical Relational Artificial Intelligence and Machine Learning Lab, working under the supervision of Dr. Vibhav Gogate. His primary research focus is to develop scalable algorithms for inference and learning in large complex graphical models. Somdeb has published seven papers at prominent machine learning and AI venues like NIPS, UAI, and AAAI. His research interest includes Artificial Intelligence, Machine Learning, Probabilistic Graphical Models, Statistical Relational Models, Combinatorial Optimization and Deep Learning.

Somdeb has more than six years of industry experience. Before joining UTD, he has worked as a software engineer at Tata Consultancy Services, India for six years. He has also worked as a research intern for Adobe Systems Inc., San Jose during his doctoral work at UTD. After completing his Ph.D., Somdeb will begin working at Adobe Systems, where he will use his machine learning skills to enhance the experience for digital marketing.