Efficient Prediction of Relational Structure and its Application to Natural Language Processing

Sebastian Riedel

Doctor of Philosophy

Institute for Communicating and Collaborative Systems
School of Informatics
University of Edinburgh

2009

Abstract

Many tasks in Natural Language Processing (NLP) require us to predict a relational structure over entities. For example, in Semantic Role Labelling we try to predict the 'semantic role' relation between a predicate verb and its argument constituents. Often NLP tasks not only involve related entities but also relations that are stochastically correlated. For instance, in Semantic Role Labelling the roles of different constituents are correlated: we cannot assign the agent role to one constituent if we have already assigned this role to another.

Statistical Relational Learning (also known as First Order Probabilistic Logic) allows us to capture the aforementioned nature of NLP tasks because it is based on the notions of entities, relations and stochastic correlations between relationships. It is therefore often straightforward to formulate an NLP task using a First Order probabilistic language such as Markov Logic. However, the generality of this approach comes at a price: the process of finding the relational structure with highest probability, also known as maximum *a posteriori* (MAP) inference, is often inefficient, if not intractable.

In this work we seek to improve the efficiency of MAP inference for Statistical Relational Learning. We propose a meta-algorithm, namely Cutting Plane Inference (CPI), that iteratively solves small subproblems of the original problem using any existing MAP technique and inspects parts of the problem that are not yet included in the current subproblem but could potentially lead to an improved solution. Our hypothesis is that this algorithm can dramatically improve the efficiency of existing methods while remaining at least as accurate.

We frame the algorithm in Markov Logic, a language that combines First Order Logic and Markov Networks. Our hypothesis is evaluated using two tasks: Semantic Role Labelling and Entity Resolution. It is shown that the proposed algorithm improves the efficiency of two existing methods by two orders of magnitude and leads an approximate method to more probable solutions. We also give show that CPI, at convergence, is guaranteed to be at least as accurate as the method used within its inner loop.

Another core contribution of this work is a theoretic and empirical analysis of the boundary conditions of Cutting Plane Inference. We describe cases when Cutting Plane Inference will definitely be difficult (because it instantiates large networks or needs many iterations) and when it will be easy (because it instantiates small networks and needs only few iterations).

Acknowledgements

First I would like to thank my supervisor, Ewan Klein, for the support and guidance he provided through the course of PhD studies, as well as the freedom he gave me to pursue my very own research goals. I also like to use this chance to apologise for the miserable English he had to correct so many times.

Many thanks also to my second supervisor, Miles Osborne, as well as my committee: Steve Renals and Mirella Lapata. I am particularly thankful to Mirella, who not only gave me much thoughtful and thorough advice, but also the encouragement I needed at times. Thank you to also to Pedro Domingos for agreeing to serve on my thesis committee and providing many helpful advices and insights during and after the viva.

I am very happy about the joint work I have done with my office mate James Clarke, and very grateful for his comments and suggestions regarding this thesis. Likewise, I am thankful for the constant input I got from Ivan Meza-Ruiz—without him this would all have been much more difficult.

I would like to thank my office mates: Abhishek, James and Ruken. Working with them was not only inspiring, it was also extremely joyful. In fact, my office was probably my favourite place in Edinburgh, and not only because it had a working heater.

A big thank you goes to three women that made my life so much easier and more enjoyable: Mikachan, for her love, support and refreshingly non-scientific view on things; Mogilinde, for all the love and support I feel when I am back home (and abroad), and Khaki, for her thoughtfulness and help with the small and big problems that can make the life of a confused scientist difficult. Finally, I am deeply thankful to my father and wish he could still be around.

Declaration

I declare that this thesis was composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

(*Sebastian Riedel*)

To Mika, Jutta, Katrin and Hartwig.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Many tasks in Natural Language Processing (NLP) can be cast into a relational framework: they all require us to predict a relational structure over objects. For example, in Coreference Resolution we search for a relation over phrases that refer to the same entity. For Dependency Parsing we need to infer a syntactic relation between the tokens of a sentence. When we do Semantic Role Labelling we are predicting a relation between verbs and their semantic arguments. In Word Alignment we are extracting a relation between the tokens of a sentence in one language and the tokens of a sentence in another language. In Relation Extraction we find relationships between the entities mentioned in a text.

The predominant approach to NLP is based on statistical models where stochastic variables represent the decisions to make in a particular task. In this light the above tasks not only consider *related* entities, but also *correlated* relations. For instance, in Coreference Resolution we know that if phrases A and B and phrases B and C refer to the same entity, then so do phrases A and C. During Dependency Parsing we cannot include an edge between a head word and a modifier if it would close a cycle of already included edges. In Semantic Role Labelling we cannot label two constituents as the agents of a verb. Note that correlations between relationships often come along with relationships between entities. For example, in Semantic Role Labelling we cannot label two constituents as the agent of a verb if the constituents are in the "in-same-sentence" relationship.

1.1 Statistical Relational Learning

Statistical Relational Learning [SRL, Getoor and Taskar, 2007] is a branch of Machine Learning that seeks to integrate probabilistic models and relational representations. Frameworks such as Markov Logic [Richardson and Domingos, 2005] and Bayesian Logic Programs [Kersting and De Raedt, 2000] allow us to describe the world in terms of entities, relations and stochastic correlations between relations. They combine ideas taken from the world of First Order Logic with concepts such as cliques and feature functions, drawn from the world of probabilistic models.

If we consider the relational nature of NLP applications as discussed above, and the correlations we need to model, the idea of using Statistical Relational Learning to tackle NLP tasks seems very intuitive. In the SRL paradigm the NLP developer would simply formulate her intuition about the correlations between relationships (such as "semantic-argument-of" relationships) in an SRL language. In turn this formulation, along with some training data, is provided to an SRL interpreter which learns the parameters of a stochastic model. At test time the SRL interpreter predicts the most likely relational structure with respect to the learnt model.

Such a *declarative approach* to Natural Language Processing has an important advantage: it allows the NLP researcher to focus on modelling the task, and Machine Learning researchers to focus on developing new inference and learning algorithms. In this view an SRL language like Markov Logic serves as interface between applications and Machine Learning technology, just as SQL serves as interface between applications and database technology, or VRML as interface between applications and chip technology. It is this decoupling of application and underlying technology that has dramatically increased the rate of progress in many fields [Domingos, to appear].

In fact, many NLP researchers have already made use of a declarative approach to NLP. At one point or another, they have probably used one of the numerous software packages for classification that implement techniques such as Support Vector Machines or Maximum Entropy models. Maybe they have also applied linear chain Conditional Random Field (CRF) toolkits, or Hidden Markov Model packages, that predict sequences of labels instead of the class of an individual item. In all these cases the NLP researcher only has to provide feature functions (or kernel functions) and the training data—both inference and training algorithms are already implemented.

While local classifiers and sequential models solve a surprising number of tasks really well, there are cases where their underlying independence assumptions are too

restrictive. For example, in the case of Semantic Role Labelling, the constraint that a verb can only have one agent breaks the assumption that we can label constituents as semantic arguments in an isolated fashion. Likewise, tackling complex problems such as Dependency Parsing by using sequential CRFs or local classifiers alone is likely to fail. It is hard to imagine how such approaches can enforce global properties of valid relational structures, such as the acyclicity property of dependency graphs. This is where Statistical Relational Learning comes into play.

In this thesis we will focus our efforts on a particular SRL language: Markov Logic. This is due to two reasons: firstly, Markov Logic unifies several other SRL approaches; secondly, it allows us to describe large and complex Markov Networks—a type of Graphical Model that has been successfully used in many recent NLP applications. Markov Logic can be seen either as template language for Markov Networks or as an extension to First Order Logic where certain rules can be violated to some extent. It is based on weighted First Order Logic formulae that describe a Markov Network and a log-linear distribution over possible relational structures. These structures are described through the notion of entity constants and predicates. Roughly speaking, the higher the weight of a formula, the higher will be the probability of each relational structure that satisfies the formula.

Statistical Relational Learning is also closely related to Structured Prediction [Taskar, 2004, Daumé III, 2006]. In this field of Machine Learning we ask how to perform inference and learning in domains where the structure of a problem imposes dependencies between statistical output variables, just as we described in the beginning of this chapter. One way of seeing Statistical Relational Learning is as a declarative approach to Structured Prediction in which we not only try to find learning and inference techniques for structured domains, but also ask how to provide these techniques in an application-independent fashion. Again, this allows the domain expert to focus on his or her task, and Machine Learning researchers to focus on inference and learning algorithms.

1.2 Semantic Role Labelling Example

Let us look at a particular NLP application, Semantic Role Labelling [Carreras and Marquez, 2005], in order to illustrate the potential benefits of an SRL language such as Markov Logic. Semantic Role Labelling refers to the task of identifying and classifying the arguments and modifiers of verbs in natural language text. For example,

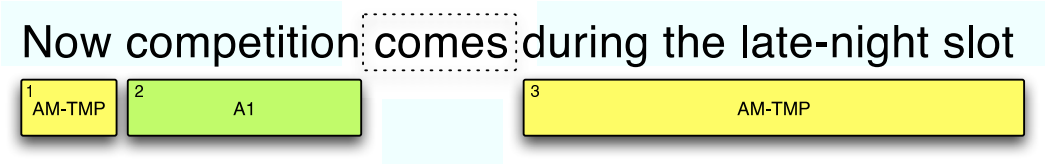


Figure 1.1: A semantic role labelling for sentence 1.2. “Now” is a temporal modifier of “comes”, competition is its A1 argument (the entity in motion / the ’comer’) and “during the late night slot” is another temporal modifier.

consider the sentence

Now competition comes during the late-night slot.

A correct role labelling for this sentence can be seen in figure 1.1. “Now” is a temporal modifier of “comes”, competition is its A1 argument (the entity in motion / the ’comer’) and “during the late night slot” is another temporal modifier.

The most effective way of Semantic Role Labelling to date is based on the output of a constituent parser. Each constituent is labelled with the type of argument or modifier it represents with respect to the verb in question.

It easy to formulate this task in Markov Logic. For example, we could introduce a binary predicate *role* that relates constituents to their semantic role with respect to a verb. Assuming that the integer constant i refers to constituent i in figure 1.1, for the example sentence the following set of atoms would be true: *role*(1,AM-TMP), *role*(2,A1) and *role*(3,AM-TMP).

We can now formulate our intuition or linguistic knowledge using weighted first order logic formulae. For example, we encode the assumption that prepositional phrases are often temporal modifiers (denoted by the constant AM-TMP) through the formula

$$type(i, \text{PP}) \Rightarrow role(i, \text{AM-TMP}) \tag{1.1}$$

This rule obviously does not hold all the time. For instance, the prepositional phrase “in the park” is surely not a temporal modifier. By assigning a finite weight to this formula we make sure that labellings where this rule is violated may be less likely but still valid role labellings.

A rule such as 1.1 can in fact be implemented using any local classifier. Essentially there is only one hidden variable: the role label of a constituent. In contrast, the type of each constituent is fully observed (again assuming our system uses the output of a constituent parser). However, Markov Logic also allows us to formulate more *global*

correlations between two or more hidden variables. For example, the rule

$$role(i, \text{A1}) \wedge i \neq j \Rightarrow \neg role(j, \text{A1}) \tag{1.2}$$

states that a verb cannot have more than one agent (or A1 argument). This rule would have a very large/infinite weight which indicates that labellings for which formula 1.2 is true are infinitely more probable than those for which the formula is false.

Note that there exist Semantic Role Labelling systems [Punyakanok et al., 2005] that already capture such constraints. However, by using and promoting a Markov Logic approach we gain the following advantages.

Firstly, we can improve the efficiency of predicting the most likely role assignment by exploiting the first order information encoded in a Markov Logic model. This allows us to tackle larger Role Labelling problems, such as the joint labelling of all arguments and all predicates of the same sentence [Riedel and Meza-Ruiz, 2008].

Secondly, we can reduce the amount of engineering necessary to implement a Semantic Role Labelling system. Punyakanok et al. [2005] use an Integer Linear Programming (ILP) formulation of Semantic Role Labelling. In their approach we need to write wrapper code that generates the ILP for each problem instance, send this ILP to a solver and convert the output of the solver back to a relational structure. In Markov Logic we only need to provide first order formulae and the observed relational structure to the Markov Logic interpreter, which will then return the predicted hidden relational structure.

Thirdly, in languages such as Markov Logic it is possible to automatically search for rules that improve the accuracy of a model. In the future this might allow us to design improved Semantic Role Labelling models with minimal engineering effort.

1.3 Maximum A Posteriori Inference

We have argued that Statistical Relational Learning, and therefore Markov Logic, is a good fit for NLP applications. However, up to now there has not been much work within NLP that uses Statistical Relational Learning languages such as Markov Logic [Riedel and Klein, 2005, Bunescu and Mooney, 2004, Poon and Domingos, 2007]. We believe that one reason for the lack of applied SRL in NLP is the complexity associated with the computational tasks an SRL interpreter has to perform. One such task is Maximum *a Posteriori* Inference.

An integral part of many NLP applications is the search for a hidden structure with maximal probability with respect to a probabilistic model and a given observation. In more formal terms, given a statistical model with parameters Θ and an observation $\mathbf{x}$ we search the hidden structure $\mathbf{y}$ out of a set possible structures $\mathcal{Y}$ with maximal probability[1]

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in \mathcal{Y}} Pr(\mathbf{y}|\mathbf{x}, \Theta) \tag{1.3}$$

This problem is usually referred to as either Maximum *A Posteriori* (MAP) inference (as we are inferring the solution $\hat{\mathbf{y}}$ with maximal *a posteriori* probability), *decoding* or the *search* problem. In Markov Logic, MAP inference amounts to predicting the relations between a set of entities that have maximal conditional probability.

If we cannot perform MAP inference in an efficient and accurate manner, a model can be trained arbitrarily well using sophisticated learning regimes and still not be useful because inference is slow, or even intractable. Or inference could be efficient but lead to poor results because we are simply not able to find what the model assumes to be most likely [Daumé III, 2006]. Conversely, if we have a method for efficient and accurate MAP inference, Online Learning methods [Collins and Roark, 2004, Crammer and Singer, 2003] allow us to readily learn the parameters of a model. We can thus argue that for Markov Logic to be useful for NLP, the availability of an efficient and accurate MAP inference method is both a necessary and sufficient condition.

1.4 Research Question

Richardson and Domingos [2006] propose the use of MaxWalkSAT [Kautz et al., 1997] as a MAP inference method for Markov Logic. In this thesis we will make two observations (see chapter 5):

- **MAP Inference with MaxWalkSAT can be slow for simple problems**. We apply MaxWalkSAT (MWS) to a state-of-the-art Markov Logic model for Semantic Role Labelling. MWS indeed finds solutions with maximal probability. However, this approach requires more than six seconds on average for each verb in a sentence.

- **MAP Inference with MaxWalkSAT can be very inaccurate for more complex problems**. We also apply MWS to a Joint Entity Resolution problem that

[1] Assuming that a unique maximum exists; otherwise we search for an arbitrary member of the set of optimal hidden structures.

requires inference to collectively match up to 500 citations in scientific papers along with the corresponding authors and venues. Here the Markov Networks that the Markov Logic model describes contain more than 250,000 nodes and millions of edges. In this scenario MWS (with a million steps) is reasonably fast (2 minutes per problem) but its accuracy is very poor. The solutions that MWS returns are significantly less likely than the optimal solution. This leads to a drop of 50% points in F1 measure for this task.

MaxWalkSAT is a method based on a random walk strategy that often leads to good solutions, but is not guaranteed to do so. To improve the accuracy of MAP inference in cases where MaxWalkSAT performs poorly we tried using Integer Linear Programming (ILP) as an alternative. ILP guarantees optimal solutions but is considered to be slow in general. In fact, perhaps not surprisingly, we make the following observation:

- **MAP Inference with ILP can be intractable for more complex problems**. We map the MAP problem in Markov Logic to an Integer Linear Program and run an off-the-shelf ILP solver on this program. In the case of Semantic Role Labelling this is roughly as efficient (or inefficient) as MWS (with the additional benefit that we know results to be optimal with ILP). However, when applied to the Joint Entity Resolution task, the ILP solver fails to converge.

If we cannot efficiently solve relatively small problems such as Semantic Role Labelling, and if we cannot accurately solve larger problems, such as Joint Entity Resolution, it seems unclear how Markov Logic (or any other Statistical Relational Learning language) can ever be useful for tasks such as the joint tackling of Parsing, Coreference Resolution, Word Sense Disambiguation and Semantic Role Labeling [Domingos, 2008]. Given the benefits of Statistical Relational Learning languages such as Markov Logic we mentioned in sections 1.1 and 1.2, this leads us to the following question that we will seek to answer with this work:

> How can we make accurate MAP inference for Markov Logic more efficient?

Because a Markov Logic model essentially describes a Markov Network, it is possible to consider MAP inference in Markov Logic as the problem of MAP inference in Markov Networks. Hence we could try to answer this question by using other Markov Network MAP algorithms from the literature, such as Belief Propagation and its variants [Murphy et al., 1999]. However, there are two main problems with this approach.

First, the networks we consider are not only large and densely connected, they are also partially deterministic. Solvers such Belief Propagation and its variants tend to perform poorly in such settings. Second, we will observe that due to the size and connectivity of the Markov Networks that Markov Logic describes, the process of instantiating these networks based on the first order representation takes a substantial amount of time. This time will remain constant if we only pick another Markov Network MAP method. In this work we will therefore follow a different route and introduce Cutting Plane Inference.

1.5 Cutting Plane Inference

While for simple classification tasks searching for the most probable $\mathbf{y}$ is trivial, for most other tasks it is very difficult. Usually the space of possible solutions $\mathcal{Y}$ is very large. For example, in Entity Resolution there is an exponential number of clusterings of records; in parsing we deal with an exponential number of parse trees. Simply enumerating all members of $\mathcal{Y}$ is thus not feasible. This is particularly true in the case of Statistical Relational Learning problems where the search space consists of all possible relations over a (potentially large) set of objects.

The problem of MAP inference is NP-hard[2] and thus in general we cannot expect to find the true optimal solution for general Graphical Models. Instead we usually follow one of two ways. Either we rely on approximation methods that return suboptimal solutions which are hopefully close to a true optimum. One example of an approximate method is MaxWalkSAT—it can be relatively efficient but does not give any guarantees of how good the returned solution might be. Or we make structural independence assumptions that give rise to exact but specialised search methods. For example, if we assume that stochastic variables are arranged in a linear chain and only directly depend on their neighbours (as in a Hidden Markov Model of linear-chain CRF) we can make use of the Viterbi algorithm to search for the MAP solution in linear time with respect to the number of variables. However, once we make such assumptions it is difficult to exploit additional correlations present in the data [Finkel et al., 2005].

This thesis seeks to show a third alternative. We present an algorithm that allows efficient and exact inference for a large class of problems. However, this class is not defined in terms of independence assumptions. Instead it considers different properties, such as the precision of a relaxed model that only considers local formulae, or the

[2] In fact MAP inference is NP^{PP}complete [Park, 2002] and thus even harder.

number of ground formulae we can prove or disprove with the formulae in the Markov Logic model.

We propose *Cutting Plane Inference* (CPI), a meta-algorithm for MAP inference in Markov Logic. CPI iteratively solves small sub-networks of the complete ground network using any MAP algorithm of choice (the *base solver*). It inspects parts of the complete network that are not yet included in the current partial network but could potentially lead to an improved solution. These parts are then added to the partial network, which is in turn solved again.

1.6 Contributions

This thesis makes the following core contributions:

Cutting Plane Inference We present a Cutting Plane Algorithm for MAP inference in Markov Logic that can be used in combination with any existing MAP method for Markov Networks, avoids the instantiation of the complete network to improve efficiency and accuracy of existing solvers and can be applied to MLNs with both soft and hard constraints (chapter 4).

Analysis of boundary conditions for Cutting Plane Inference We give answers to the following questions: when will Cutting Plane Inference definitely generate large partial networks, and when will partial networks be small? When will CPI definitely need many iterations, and when only a few (chapter 6)?

Note that the idea of using a cutting plane algorithm for MAP inference is not new by itself.[3] However, this is the first work that incorporates Cutting Planes into a Statistical Relational Learning framework. Moreover, our proposed method has several other advantages, such as a simple way to incorporate soft constraints and the possibility to plug-in any existing MAP inference method for Markov Networks.

We believe that Cutting Plane Inference will be important for a more widespread use of Markov Logic (and Statistical Relational Learning in general) within NLP, and will hopefully help to increase the rate of progress within the field. This hope is supported by the following findings we present in chapter 5 of this thesis:

- **Cutting Plane Inference improves efficiency of a base solver in several dimensions**. We show that CPI can dramatically improve the efficiency of a base

[3] However, note that most of the work on Cutting Plane Algorithms for MAP inference appeared in the course of the work on this thesis.

solver, in terms of (a) initialisation overhead, (b) time spent in optimisation algorithm, (c) memory overhead and (d) behaviour for increasing problem size.

- **Cutting Plane Inference improves accuracy of a reasonably accurate base solver**. We show that not only will inference be more efficient, it can also be more accurate, in case the base solver is already reasonably accurate; we also show that for a very inaccurate base solver the use of CPI leads to worse performance.

- **Efficient and exact MAP inference for Markov Logic is possible in large real world application**. We show that it is possible to find the true optimum for Markov Logic Networks that define Markov Networks with millions of edges and nodes.

An indicator of the future impact of Cutting Plane Inference can be observed in our recent work. Cutting Plane Inference has already been used in the first model for Semantic Role Labelling that jointly predicts which tokens are predicates and which are the semantic arguments for all tokens in a sentence [Riedel and Meza-Ruiz, 2008]. This model achieved the best exact-match score for out-of-domain data in the open track of the CoNLL 2008 shared task, and the second-best scores for in-domain data. Cutting Plane Inference has also been used in a state-of-the-art Spoken Language Understanding module for Dialogue systems [Meza-Ruiz et al., 2008b,a].

While we show that CPI improves MAP inference for two tasks, it is not immediately clear when we can expect this in general. We therefore also contribute a detailed analysis of the behaviour and boundary conditions of CPI. This analysis concerns two questions: how large are the partial networks during inference, and how many steps are needed until CPI terminates? The findings we present in this thesis will give answers to these questions; however, they require a certain amount of knowledge about Markov Logic and Cutting Plane Inference we cannot presuppose. The reader is therefore referred to chapter 6 and 7 for a summary of our analysis.

Along with the above core contributions and findings we also present a set of supplementary contributions:

A state-of-the-art Markov Logic Network for Semantic Labelling This shows how easily and compactly an accurate NLP model can be formulated in Markov Logic, declaratively and by integrating findings of existing work (section 5.3).

Separation in Cutting Plane algorithm as first order query processing This allows CPI implementations to make use of the "best of both worlds": existing query processing software and MAP algorithms implementation (section 4.5).

ILP Formulation for MAP inference in Markov Logic This mapping of Markov Networks to Integer Linear Programs uses significantly less variables (while yielding the same amount of constraints in our applications) as the conventional representation based on the Marginal Polytope (section 2.3.2.2).

Ordered Cutting Plane Inference We present a variant of Cutting Plane Inference that processes formulae in a user-defined order. This can lead to smaller partial problems for the base solver (section 4.6).

1.7 Thesis Overview

This work is formulated within the framework of Markov Logic, a Statistical Relational Learning language that combines Markov Networks and First Order Logic. We will thus proceed by first giving an introduction to Markov Networks in chapter 2 and describe methods for MAP inference and learning in such networks. In the course of this chapter we will use our Semantic Role Labelling example to show how Markov Networks can be used to model NLP applications.

In chapter 3 we will then introduce Markov Logic as a language to describe complex Markov Networks. We will also illustrate how to use Markov Logic to encode our intuitions about an NLP task, based on a Semantic Role Labelling example. Finally, we discuss MAP inference and learning for Markov Logic.

Chapter 4 will present Cutting Plane Inference in Markov Logic. This includes a brief introduction to the general idea of cutting planes. Furthermore, we will formally prove that Cutting Plane Inference for the complete problem is as accurate as the base solver for a subproblem. We will also introduce an extension of Cutting Plane Inference, namely Ordered Cutting Plane Inference, that processes formulae in a user-specified order.

In chapter 5 we hypothesise that Cutting Plane Inference can significantly improve both the efficiency and accuracy of existing methods. This is hypothesis is evaluated using two tasks: Semantic Role Labelling and Entity Resolution (a task very similar to Coreference Resolution). In the case of SRL we present a Markov Logic Network for Semantic Role Labelling, inspired by previous work in the domain, that achieves

state-of-the-art accuracy. In the case of Entity Resolution we re-use a Markov Logic Network from existing work.

In chapter 6 we present a set of theoretical and empirical observations that will help to understand the runtime behaviour of Cutting Plane Inference. This includes upper and lower bounds for the size of partial problems generated during CPI as well as the number of iterations necessary.

Finally we will summarise the contributions of this work along with directions for further research in chapter 7.

Note that we will label sections, tables, figures and other elements using a prefix which indicates the chapter that contains the corresponding section. For example, section 7.1 can be found in chapter 7.

Chapter 2

Markov Networks

When we reason about a problem it is often helpful to divide it into a set of sub-problems. For example, instead of trying to find all semantic arguments of a verb in a given sentence at once, we can try to find the role of each constituent individually. If we assume that we can assess each assignment of a role to a constituent with a score or probability function,[1] searching for all semantic arguments becomes a simple task: for each constituent we score all possible role assignments and pick the one with the highest score or probability. This task can be performed in $n \cdot m$ steps, where n it the number of constituents and m the number of possible labels. In contrast, naively searching through the set of complete role labellings for a verb takes m^n steps, one for each possible role labelling of all constituents in the sentence.

Likewise, learning the parameters of a probabilistic model that considers a labelling for a sentence as atomic event will inevitably lead to data sparsity problems because it is unlikely that we will see a particular sentence more than once. In contrast, there will be far more repeating events if we consider constituents and their roles individually.

However, when we break up a problem we run the risk of neglecting useful correlations between sub-problems. For instance, it is known that a verb cannot have more than one agent. By solving the role assignment as series of independent decisions this observation cannot be taken into account and the overall solution might violate this constraint [Punyakanok et al., 2005].

Graphical Models [Pearl, 1988] allow us to incorporate correlations between statistical variables in a principled fashion. Instead of assuming total isolation, Graphical Models define a local neighbourhood of interacting variables. In most Graphical Models, variables interact *directly* with only a few other variables. This often allows

[1] For instance, by using some off-the-shelf multi-class classifier software.

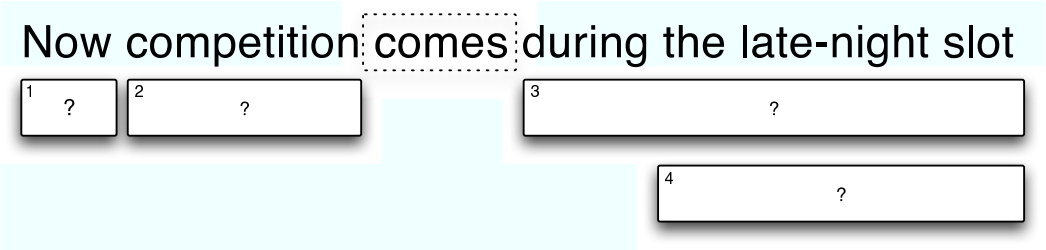


Figure 2.1: The four constituents to label for the given sentence.

learning and inference methods to remain efficient and also reduces the risk of data sparsity problems.

In this chapter we will present Bayesian and Markov Networks as well as a subset of inference and learning methods for Graphical Models. We will focus on Markov Networks, because they are a fundamental part of Markov Logic, and on those methods that have been used in the context of Markov Logic—for a complete introduction to Markov Networks the reader is referred to other work [Pearl, 1988].

2.1 Graphical Models

To introduce Graphical Models we will use the example problem formulated in section 1.2 . In figure 2.1 we see the sentence and constituents to be labelled. For simplicity let us assume that we only have two types of labels to assign: AM-TMP and A1. Let us also assume that a domain expert made the following observations that will help us to solve this task:

1. Semantic Role Labelling is a two step process where for each constituent two decisions are to be made: first we decide if the constituent is an argument (*argument identification*) and then which role it takes (*argument classification*).

2. A constituent can have at most one label.

3. A verb can only have one A1 argument.

4. A verb might have more than one AM-TMP argument, but a penalty is paid for this.[2]

[2]Note that encoding this assumption did not improve performance in our experiments; however, we add it here to illustrate how to encode soft correlations in Markov Networks (and Markov Logic Networks in the chapter 3).

5. The role label of a constituent is correlated with the Part of Speech tag of the head word[3] of the constituent.

6. Two overlapping constituents cannot both be arguments.

In the following we will try to capture these observations (or some of these) using both Bayesian Networks and Markov Networks.

2.1.1 Bayesian Networks

Bayesian Networks are used to represent statistical variables that interact in a causal or generative manner. A *Bayesian Network* is a directed acyclic graph in which nodes represent statistical variables. The directed edges of the graph indicate that there is a correlation between the parent variable (the variable at the start of the edge) and child variable (the variable at the end of the edge) such that the states of the parents cause or generate the state of the child. The graphical structure of a Bayesian Network is said to represent the "qualitative" part of a statistical model. The "quantitative" part is a collection of conditional probability distributions (CPDs) that define the probabilistic dependence of a variable on its parents.

More formally, a Bayesian Network is defined as follows [Ben-Gal, 2007]:

Definition 2.1. A *Bayesian Network B* is a pair (G,Θ) where

- G is a directed graph $(\mathbf{Y},E)$ where the vertices $\mathbf{Y} = (Y_i)_i$ are a family of random variables and the edges in E represent causal dependencies between the variables.

- Θ contains a conditional probability distribution $Pr_i(Y_i|\mathbf{PA}_v)$ for every variable Y_i in $\mathbf{Y}$, where $\mathbf{PA}_i$ is the set of parent variables of Y_i as defined by E.

A Bayesian Network defines the following joint probability distribution over the variables $\mathbf{Y}$ as follows

$$Pr(\mathbf{Y} = \mathbf{y}) = \prod_i Pr_i(y_i|\mathbf{pa}_i) \tag{2.1}$$

where $\mathbf{pa}_i$ is the part of $\mathbf{y}$ that represents the state of the parents $\mathbf{PA}_i$ of the variable Y_i.

[3] Roughly speaking, the head word is the word in a grammatical constituent that plays the same grammatical role as the whole constituent. All other words in the same constituent essentially just modify (or transitively modify) the head word. For example, in "the late-night slot" the head word is "slot", and both "the" and "late-night" modify "slot".

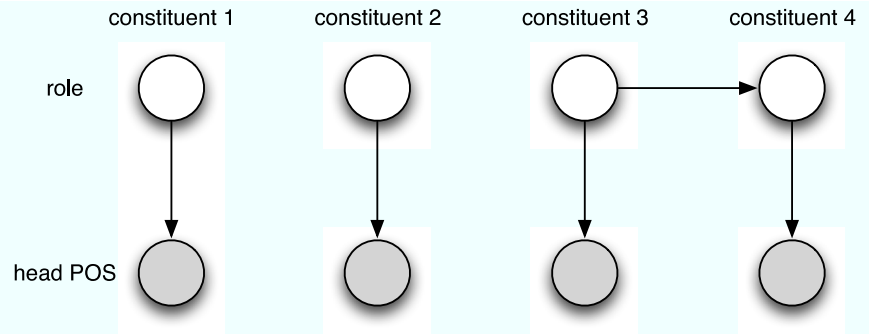


Figure 2.2: A Bayesian Network that captures the assumption that the head PoS tag of a constituent is correlated with the constituent's role and that overlapping constituents cannot both have semantic roles. The direction of the vertical arrows indicates that the semantic role generates the head POS tag of a constituent. The direction of the horizontal arrow indicates that the semantic role of constituent 3 generates the role of constituent 4.

For example, consider the sentence in figure 2.1 and assume that both the roles and the POS tags of the head words of constituents are the statistical variables of interest. In the following we refer to the variable that denotes the role of constituent i as $Y_{role,i}$ and the variable that denotes the head POS tag of constituent i as $Y_{pos,i}$. Note that we augment the set of possible roles with the NONE label, which is given to constituents that have no semantic role.

We can capture the assumptions that the head POS tag of a constituents is correlated with the constituent's role and that overlapping constituents cannot both have semantic roles using the Bayesian Network in figure 2.2. The "generative story" behind this network is the following: first we generate the semantic role of a top level candidate constituent. Based on this role we generate the POS tag of the head word of the constituent. If the constituent covers another candidate constituent (as constituent 3 does) then the role of this covered constituent is generated with respect to the role of the covering constituent.

The Bayesian Network contains three types of conditional probability distributions. The first one, $Pr_{role}(r)$ assigns a prior probability to the role r. The second one, $Pr_{pos}(p|r)$, generates a POS tag p with respect to a semantic role r. The third one, $Pr_{overlap}(r_1|r_2)$ generates a role label r_1 based on another label r_2. This distribution encodes assumption 6 by $Pr_{overlap}(NONE|r_2) = 1$ for every $r_2 \neq NONE$.

The resulting joint probability over all variables of the example sentence then be-

comes

$$Pr(\mathbf{Y}) = Pr_{overlap}\left(y_{role,4}|y_{role,3}\right)\prod_{i=1}^{4} Pr_{pos}\left(y_{pos,i}|y_{role,i}\right)\cdot\prod_{i=1}^{3} Pr_{role}\left(y_{role,i}\right) \quad (2.2)$$

The acyclic, causal structure gives rises to efficient means of inference and learning for Bayesian Networks. However, sometimes it is not immediately clear what this causal structure should look like. For example, incorporating the assumption that there can only be a single A1 role with respect to a given verb is significantly more difficult. Now there is a correlation between all roles in the sentence—but what is the causal relation? Do the roles in the beginning of the sentence generate those in the end? Or vice versa?

2.1.2 Markov Networks

In Markov Networks the restriction of relations to be causal is lifted. Instead of using conditional probability tables to represent correlations between variables, a Markov Network uses compatibility functions. Here we do not speak of variable states that generate the states of other variables. Instead, variables are in states that are compatible to different degrees. Accordingly, in the graphical representation directed edges are replaced by undirected edges. Likewise, compatibility functions are not defined over a node and its parents but over fully connected subgraphs, or *cliques*, of the Markov Network.

Markov Networks also lend themselves well to discriminative learning methods that try to maximize the prediction accuracy (or any other loss function) of the probabilistic model instead of the probability of the training data (see section 2.4). It has been shown that given sufficient data, discriminative learning generally leads to more accurate models that outperform their generative counterparts [Vapnik, 1995].

Formally, a Markov Network can be defined as follows:

Definition 2.2. A *Markov Network M* is a pair (G,Θ) where

- G is an undirected graph $(\mathbf{Y},E)$ where the vertices $\mathbf{Y} = (Y_i)_i$ are a family of random variables and each edge $\left(Y_i,Y_j\right)$ represents a correlation between Y_i and Y_j.
- Θ is a family of strictly positive *potential functions* $(\varphi_c)_c$ where each φ_c has the domain of some clique c in G.

y_1	y_2	$\varphi(y_1, y_2)$
0	0	10
0	1	10
1	0	10
1	1	1

Table 2.1: Example potential function that captures the observation that Y_1 and Y_2 are not very compatible.

Let $\mathbf{y}_{\{c\}}$ be the assignment to the set of random variables belonging to the clique k, then a Markov Network defines a distribution over assignments for the variables $\mathbf{Y}$ as follows:

$$Pr(\mathbf{Y} = \mathbf{y}) = \frac{1}{Z} \prod_c \varphi_c \left(\mathbf{y}_{\{c\}}\right) \tag{2.3}$$

where Z is a normalisation constant (the so-called *partition function*):

$$Z = \sum_{\mathbf{y}} \prod_c \varphi_c \left(\mathbf{y}_{\{c\}}\right) \tag{2.4}$$

Dividing by Z guarantees that summing over all possible assignments yields 1. Note that in Bayesian Networks this normalization is achieved by normalizing each term in the joint distribution. Hence we will say that Bayesian Networks are locally normalized, whereas Markov Networks are globally normalized.

Each potential function serves as a compatibility measure for the values of its domain. For example, assume that we have two binary variables Y_1 and Y_2 and that in almost every training instance at least one of these variables was not active. We can capture this observation using a Markov Network with graph $G = (\mathbf{Y}, E)$ with $\mathbf{Y} = (Y_1, Y_2)$ and $E = \{(Y_1, Y_2)\}$ and the potential function φ show in table 2.1.

Now assume that along with Y_1 and Y_2 we have two more variables, Y_3 and Y_4, and let us define one more potential $\varphi_{1,2,3,4}$ over all four variables. Figure 2.3 shows the graph of the corresponding Markov Network. It does not capture the fact there are two potentials defined over different cliques of the graph. When we want to graphically represent this information we use a *factor graph*, as shown in figure 2.4. Here circular nodes represent variables and square nodes represent the cliques that correspond to the potentials of the Markov Network. Each square node is connected to all variables of the clique that the corresponding potential is defined over.

In the following we will refer to direct neighbours of a variable Y_i in a Markov Network. We will use the following notation:

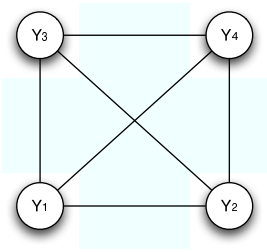


Figure 2.3: A fully connected Markov Network with four variables. It is not clear from this representation for which cliques we have defined potential functions. For example, there could be a potential for the clique $\{Y_1, Y_2\}$ and one for the clique $\{Y_1, Y_2, Y_3, Y_4\}$; however, there could also be only one potential: the one for $\{Y_1, Y_2, Y_3, Y_4\}$.

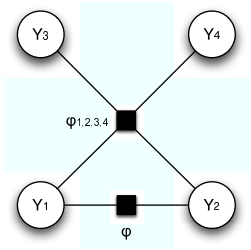


Figure 2.4: Factor graph of Markov Network.

Definition 2.3. The *Markov Blanket* $\mathbf{MB}_i$ of a variable Y_i is the set of variables directly connected with Y_i through the edges in E.

2.1.2.1 Loglinear model

An alternative representation of a Markov Network can be given by a loglinear model:

$$Pr(\mathbf{Y} = \mathbf{y}) = \frac{1}{Z} \exp\left(\sum_i f_i(\mathbf{y}) \cdot w_i\right) \tag{2.5}$$

where each f_i is a real valued *feature* function over $\mathbf{y}$ and w_i is its associated *weight*.[4]

For example, let's look at the potential function φ from table 2.1. We can represent the same function using the feature function

$$f_1(y_1, y_2) = \begin{cases} 1 & \text{if } \neg y_1 \vee \neg y_2 \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

[4]Note that we define feature functions over the complete input vector $\mathbf{y}$ because this will simplify further notation. However, in most cases feature functions will only consider a few subcomponents of $\mathbf{y}$ and so their actual domain is much smaller.

and the weight $w_1 = \log(10)$. This representation is not only often significantly more compact, it also shows a clear connection between propositional logic and Markov Networks. In the next chapter we will show how the logical condition within the feature function can be expressed using First Order Logic; this will lead us to Markov Logic.

2.1.2.2 Weighted Satisfaction Problems

The loglinear representation of Markov Networks highlights their similarity to weighted MAX-SAT problems [Borchers and Furman, 1999]. This similarity allows us to apply weighted MAX-SAT solvers, such as MaxWalkSAT (see section 2.3.1), in order to tackle the MAP inference (see section 2.3) problem for binary Markov Networks.

A weighted SAT problem can be formulated as follows. Given a family of Boolean variables $\mathbf{Y} = (Y_i)_i$ and a collection of weighted clauses $C = \left(c_j, w_j\right)_j$ where each c_j is a clause (disjunction over variables or negated variables of $\mathbf{Y}$) and each w_j is a real-valued weight associated with the clause, try to find the assignment of variables that maximises the total weight of the satisfied clauses in C.

Let $\mathbf{y}$ be such an assignment, then the objective function associated with the weighted SAT problem is

$$\sum_j w_j \cdot c_j(\mathbf{y}) \tag{2.7}$$

where we define $c_j(\mathbf{y})$ as

$$c_j(\mathbf{y}) = \begin{cases} 1 & \text{if } c_j \text{ is true in } \mathbf{y} \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

The similarity of the term 2.7 to the dot-product in the exponent of equation 2.5 shows that we can interpret the clauses of weighted MAX-SAT problem as feature functions of a loglinear model. For example, the clause $\neg y_1 \vee \neg y_2$ with weight w_i can be interpreted as the feature function f_1 in 2.6 with the same weight.

2.2 Semantic Role Labelling Example

Let us show how to solve the example problem formulated in section 1.2 using a Markov Network representation. While there are several ways to use variables to represent the decisions of the semantic role labelling problem, we pick a binary representation. This will simplify the introduction of Markov Logic Networks in chapter 3,

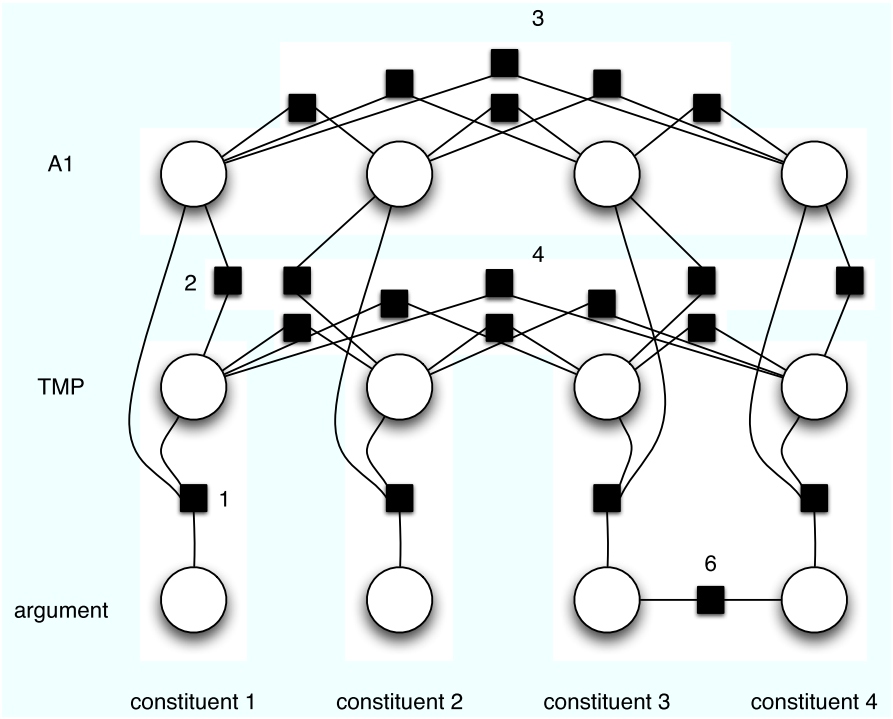


Figure 2.5: Markov Network for example Role Labelling problem. Note that this network only shows hidden variables and no local factors, and hence assumption 5 does not appear in this figure. The factors which capture the assumptions we made are labelled with the number of the corresponding assumption.

which are based on binary Markov Networks. Note, however, that a general Markov Network could represent the correlations we want to capture more compactly.

Figure 2.5 shows a factor graph of a Markov Network representation of this problem. The bottom layer consists of a set of binary variables that are active if and only if the corresponding constituents are semantic arguments of the verb, regardless of the semantic label they have. Note that this layer corresponds to the argument identification stage of many Semantic Role Labelling systems. We will refer to the variable that denotes whether constituent i is a semantic argument as Y_i.

The next layer contains binary nodes that indicate whether a constituent is labelled as AM-TMP argument. We will denote the variable that is active if and only if constituent i is labelled with AM-TMP with $Y_i^{\text{AM-TMP}}$. Likewise, the final layer contains binary nodes that are true if their corresponding constituent is labelled as A1. In this case we will denote the variable that is active if and only if constituent i is labelled with A1 with Y_i^{A1}. Note that the final two layers correspond to the argument classification of many Semantic Role Labelling systems.

Our model represents the assumptions made in section 2.1 using loglinear features. To ensure consistency between the predicate identification and classification stages we use the following type of feature:

$$f_i^1(\mathbf{y}) = \begin{cases} 1 & \text{if } y_i \Leftrightarrow y_i^{\text{AM-TMP}} \vee y_i^{\text{A1}} \\ 0 & \text{otherwise} \end{cases} \tag{2.9}$$

The assumption that a constituent cannot have more than one label is captured by a series of features $\left(f_i^2\right)_i$ with

$$f_i^2(\mathbf{y}) = \begin{cases} 1 & \text{if } \neg y_i^{\text{AM-TMP}} \vee \neg y_i^{\text{A1}} \\ 0 & \text{otherwise} \end{cases} \tag{2.10}$$

Likewise, the assumption that there cannot be more than one A1 argument for any given verb can be captured using the family of features $\left(f_{i,j}^3\right)_{i,j}$ with

$$f_{i,j}^3(\mathbf{y}) = \begin{cases} 1 & \text{if } \neg y_i^{\text{A1}} \vee \neg y_j^{\text{A1}} \\ 0 & \text{otherwise} \end{cases} \tag{2.11}$$

The assumption that AM-TMP modifiers are less likely to appear together can be captured using the features $\left(f_{i,j}^4\right)_{i,j}$ with

$$f_{i,j}^4(\mathbf{y}) = \begin{cases} 1 & \text{if } \neg y_i^{\text{AM-TMP}} \vee \neg y_j^{\text{AM-TMP}} \\ 0 & \text{otherwise} \end{cases} \tag{2.12}$$

The correlation between the fact that a constituent is a semantic argument and the Part of Speech tag of the constituent is represented using

$$f_{i,p}^5(\mathbf{y}, \mathbf{x}) = \begin{cases} 1 & \text{if } y_i \wedge x_i^{pos} = p \\ 0 & \text{otherwise} \end{cases} \tag{2.13}$$

Here x_i^{pos} is the POS tag of the head word of the i-th constituent. Note that we use the vector $\mathbf{x}$ to indicate that the POS tag information is *observed*. Finally, the feature that forbids overlapping constituents with role labels is

$$f^6(\mathbf{y}) = \begin{cases} 1 & \text{if } \neg y_3 \vee \neg y_4 \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

Together these features and their associated weights define the loglinear distribution

$$Pr(\mathbf{y},\mathbf{x}) = \frac{1}{Z}\log\left(s(\mathbf{y},\mathbf{x})\right) \tag{2.15}$$

with

$$\begin{aligned} s(\mathbf{y},\mathbf{x}) &= \sum_i f_i^1(\mathbf{y}) \cdot w_1 \\ &+ \sum_{i\neq j} f_{i,j}^2(\mathbf{y}) \cdot w_2 \\ &+ \sum_{i\neq j} f_{i,j}^3(\mathbf{y}) \cdot w_3 \\ &+ \sum_{i\neq j} f_{i,j}^4(\mathbf{y}) \cdot w_4 \\ &+ \sum_{i,p} f_{i,p}^5(\mathbf{y},\mathbf{x}) \cdot w_5^p \\ &+ f^6(\mathbf{y}) \cdot w_6 \end{aligned} \tag{2.16}$$

So far we have not described the weights of these features. However, note that the above distribution already encodes some information about the nature of the weights: for the first four types of features all instantiations have the same feature weight. For example, the weight of a feature function f_i^i is w_1 regardless of the constituent index i . This decision is based on the intuition that the corresponding constraints are independent of the identity of the constituents they apply to; hence they share the same weight. Moreover, since assumptions 1, 2, 3 and 6 are deterministic (they can never be violated), we can set the weights w_1, w_2, w_3 and w_6 to a very large number K. This results in a near-zero probability for all $\mathbf{y}$ that violate these constraints. [5]

The remaining weights are either hand-picked, or, more commonly, estimated from a set of example solutions as described in section 2.4.

Note that while we presented the nodes in the Markov Network as a linear chain, their connectivity is not linear anymore because in each layer all nodes are pairwise connected. This shows that no kind of linear model, such as linear chain Conditional Random Fields [Sutton et al., 2004], can be used to capture our assumptions. In such cases developers usually cannot use off-the-shelf software but have to implement learning and inference methods themselves. We will later see how this observation is part of the motivation behind languages such as Markov Logic.

[5] Note that we can also set K to $-\infty$ and replace each feature f with the feature $1-f$. This leads to zero probability for solutions that violate any of the hard constraints. However, notice that in this case the distribution technically does not correspond to a Markov Network, because it is not strictly positive anymore.

2.3 MAP Inference

Often we are given a set observations $\mathbf{x}$ for a set of variables $\mathbf{X}$ and are interested in the state $\mathbf{y}$ of the remaining variables $\mathbf{Y}$ that maximize the *a posteriori* probability

$$\begin{aligned} Pr(\mathbf{Y}=\mathbf{y}|\mathbf{X}=\mathbf{x}) &= \frac{Pr(\mathbf{y},\mathbf{x})}{Pr(\mathbf{x})} \\ &= \frac{1}{Z_{\mathbf{x}}}\prod_k \varphi_k\left(\mathbf{y}_{\{k\}},\mathbf{x}_{\{k\}}\right) \end{aligned} \tag{2.17}$$

with the normalization constant

$$Z_{\mathbf{x}} = \sum_{\mathbf{y}}\prod_k \varphi_k\left(\mathbf{y}_{\{k\}},\mathbf{x}_{\{k\}}\right) \tag{2.18}$$

In other words: we search for the state of hidden variables $\mathbf{Y}$ that has maximal probability if we assume the state of the observed variables $\mathbf{X}$ to be fixed. We will refer to this search as *Maximum A Posteriori* (MAP) inference.

Since the normalization constant $Z_{\mathbf{x}}$ is constant for a given $\mathbf{x}$, it is sufficient to directly optimize $\prod_k \varphi_k\left(\mathbf{y}_{\{k\}},\mathbf{x}_{\{k\}}\right)$. Moreover, it is often useful to consider the log-probability $\log\left(Pr(\mathbf{Y}=\mathbf{y}|\mathbf{X}=\mathbf{x})\right)$, which shares the its optimal arguments with the conditional probability in equation 2.17. Hence, in the case of loglinear models MAP inference amounts to the task of optimizing

$$\sum_i f_i(\mathbf{y},\mathbf{x})\cdot\theta_i \tag{2.19}$$

This representation will lead to an Integer Linear Programming formulation of MAP inference in Markov Networks, as presented in sections 2.3.2.1 and 2.3.2.2. Moreover, it is clear that in the case where the features f_i are based on logical clauses, MAP inference is equivalent to weighted MAX-SAT as defined in section 2.1.2.2.

In the following sections we will present the two methods for MAP inference in Markov Networks that we use in this thesis. We will also briefly cover alternative approaches but limit ourselves to those that have been used in the context of Markov Logic or could be readily implemented.

2.3.1 MaxWalkSAT

We have already highlighted the equivalence of weighted MAX-SAT problems and the MAP problem in binary loglinear models (with clausal features). This allows us to use methods for weighted MAX-SAT for MAP and vice versa. An inference method for

weighted MAX-SAT problems that has been used for loglinear models[6] is MaxWalkSAT [MWS, Kautz et al., 1997]. MWS is a stochastic search algorithm that has been shown to work well in many cases where systematic solvers perform poorly.

Algorithm 2.1 shows MWS in pseudo-code. MWS starts by randomly initializing a solution $\mathbf{y}$ which is then iteratively changed and, with a user-specified probability p, greedily improved. At each step a clause c which is not currently satisfied in $\mathbf{y}$ is picked randomly. Then we decide for which one of the variables in M that c covers we should change the current assignment. This change is usually referred to as a *flip*. The decision which variable to flip is based on a biased coin. If it shows heads we randomly pick one of the variables in c. If it shows tails we pick the variable in the current clause which, when flipped, maximally increases the overall sum of weights. To this end we calculate the possible gain $s_{a'}$ for every atom a' in c using the *deltaScore* function, based on the current solution $\mathbf{y}$.

The above process is repeated until a predefined number of iterations n_{flips} is reached; finally, of all solutions generated during MWS we return the one with the highest score. Optionally, we can repeat the above process $n_{restarts}$ times, each time starting with a different initialization, and return the best solution of all runs. This is helpful when the distribution has multiple local optima and a single MWS run can be trapped in one of these optima.

The operations performed in the main loop of MWS are simple and can be implemented efficiently. Hence it is often possible to perform millions of flips in seconds or fractions of seconds. This can lead to accurate solutions without high runtime requirements. However, there are two problems we noticed when using MWS.

First, the likelihood of picking a variable to flip depends on the *number* of unsatisfied clauses it is contained in, while intuitively it should mostly depend on the cumulative weight of these formulae. In other words, we want formulae with large weights to have a stronger impact on the solution than, for example, large sets of formulae with near-zero weight.

Let us illustrate this with an example. Assume a variable Y_1 that is contained as negative literal in many clauses such as $\neg Y_1 \vee Y_5$ with near-zero weights. It is also contained in the unit clause Y_1, which has a very large weight, but not contained in any other clauses.

Assume Y_1 is true in $\mathbf{y}$, then there can be several violated clauses such as $\neg Y_1 \vee$

[6] In fact it was used for MAP inference in Markov Logic; however, we present MWS in this chapter because of its propositional nature.

Y_5. It is relatively likely to pick one of these violated clauses in the next MWS step, merely because there are many of them. With some probability we then pick Y_1 to flip, even though this decreases the overall score significantly, because MWS allows us to make suboptimal steps. Changing the state of Y_1 renders the corresponding unit clause Y_1 unsatisfied. However, picking the variable Y_1 again in the next steps is rather improbable even though its weight is very large. This is due to the fact that there is only one formula that contains Y_1 as a positive literal, and picking it randomly out of a possibly large set of violated clauses is relatively unlikely.

The second problem we encounter when using MWS is the fact that the number of iterations and restarts has to be manually tuned. Hence there can be a lot of work to do when applying MWS to new problems.[7]

2.3.2 Integer Linear Programming

An Integer Linear Program [ILP, Winston and Venkataramanan, 2003] describes a constrained optimization problem of the following form

$$\begin{aligned} &\arg\max_{\mathbf{x}\in\mathbb{N}^n} \ \mathbf{c}^\top \mathbf{x} \\ &\forall i \in \{1,\dots,m\} : \mathbf{a}_i^\top \mathbf{x} \leq b_i \end{aligned} \tag{2.20}$$

where $\mathbf{c} \in \mathbb{R}^n$ is a *cost* vector and the $\mathbf{a}_i \in \mathbb{R}^n$ and b_i represent a linear inequality constraints. Here we are interested in 0-1 Linear Programs:

$$\begin{aligned} &\arg\max_{\mathbf{x}\in\{0,1\}^n} \ \mathbf{c}^\top \mathbf{x} \\ &\forall i \in \{1,\dots,m\} : \mathbf{a}_i^\top \mathbf{x} \leq b_i \end{aligned} \tag{2.21}$$

Integer Linear Programming has been used for several MAP inference tasks [Roth and Yih, 2005, Riedel and Clarke, 2006, Clarke and Lapata, 2007]. In contrast to MWS it can solve the MAP problem exactly: if an ILP solver terminates the returned assignment will be the true optimal solution. Moreover, the availability of generic ILP solvers, as free or commercial software, helps to solve complex MAP tasks with minimal engineering effort. However, a disadvantage of ILP-based inference are its memory and runtime requirements. Applying ILP to larger problems (corresponding

[7] Since large problems tend to require more flips until MWS reaches acceptable accuracy, in theory MWS needs to be tuned not only for different types of problems but also for problems of the same type with different size.

Algorithm 2.1 MaxWalkSAT algorithm.

Require: A Markov Network M with set of clausal features,
the number of restarts $n_{restarts}$,the number of flips n_{flips} and
an observation $\mathbf{x}$

$r \leftarrow 0$
while $r \leq n_{restarts}$ **do**
  $\mathbf{y} \leftarrow random$
  $i \leftarrow 0$
  while $i \leq n_{flips}$ **do**
    $c \leftarrow randomUnsatisfiedClause\,(M, \mathbf{y})$
    $u \leftarrow random\,(0,1)$
    if $u < p$ **then**
      for $a' \in c$ **do**
        $s_{a'} \leftarrow deltaScore\,(M, a', \mathbf{y}, \mathbf{x})$
      end for
      $a \leftarrow \max_{a' \in c} s_{a'}$
    else
      $a \leftarrow randomVariable\,(c)$
    end if
    $y_a \leftarrow 1 - y_a$
    $i \leftarrow i + 1$
  end while
  $r \leftarrow r + 1$
end while
return $\mathbf{y}$ with highest probability

to MAP tasks for networks with millions of edges) is often either very slow or strictly infeasible.

ILP solvers can be implemented in several ways. However, in this work we treat them as black boxes and do not explain how ILP solvers find the optimal assignments. Instead we will present two different ways of mapping the MAP inference problem to an ILP problem. First we will show a well-known mapping for generic discrete Markov Networks to Integer Linear Programs. Then we introduce a mapping designed for binary Markov Networks based on a logical representation of their features. This mapping is tailor-made for the kind of Markov Networks that Markov Logic (see chapter 3) describes.

2.3.2.1 Discrete Markov Networks

We can map every discrete Markov Network to an Integer Linear Program as follows [Taskar, 2004]. We introduce a binary variable $\mu_c\left(\mathbf{y}_{\{c\}}\right)$ for each clique c of the Markov Network (including the nodes of the network as singleton cliques) and each composite value $\mathbf{y}_c$ that can be assigned to the nodes in c. Note that the variables $\mu_c\left(\mathbf{y}_{\{c\}}\right)$ are sometimes referred to as marginals, as they represent the marginal probabilities of a (deterministic) distribution over $\mathbf{y}$.

Again, we seek to optimize the objective in equation 2.17. In terms of the variables we have just introduced, this amounts to optimizing

$$\sum \mu_c\left(\mathbf{y}_{\{c\}}\right) \log\left(\varphi\left(\mathbf{y}_{\{c\}}, \mathbf{x}\right)\right) \tag{2.22}$$

The variables $\mu_c\left(\mathbf{y}_{\{c\}}\right)$ have to be consistent with each other. This means that:

- For every clique of nodes there can only be one assignment. This leads to a constraint

$$\sum_{\mathbf{y}_{\{c\}}} \mu_c\left(\mathbf{y}_{\{c\}}\right) = 1 \tag{2.23}$$

  for every clique c.

- The assignments for cliques that share the same variables are consistent. That is, if a clique s has the assignment $\mathbf{y}_{\{s\}}$ then for any clique c that is contained in s the corresponding assignment $\mathbf{y}_{\{c\}}$ must match $\mathbf{y}_{\{s\}}$ for the nodes in c. This leads to a constraint

$$\mu_c\left(\mathbf{y}_{\{c\}}\right) = \sum_{\mathbf{y}_{\{s\}} \text{ consistent with } \mathbf{y}_{\{c\}}} \mu_s\left(\mathbf{y}_{\{s\}}\right)$$

for every clique s, every clique c that is contained in s and every assignment $\mathbf{y}_{\{c\}}$ for c. The constraint can be read as follows. If $\mu_c\left(\mathbf{y}_{\{c\}}\right)$ is active (equals one) there must be exactly one active $\mu_s\left(\mathbf{y}_{\{s\}}\right)$consistent with $\mu_c\left(\mathbf{y}_{\{c\}}\right)$; if $\mu_c\left(\mathbf{y}_{\{c\}}\right)$ is inactive (equals zero), no $\mu_s\left(\mathbf{y}_{\{s\}}\right)$ consistent with $\mu_c\left(\mathbf{y}_{\{c\}}\right)$can be active.

This yields the following Integer Linear Program

$$\begin{aligned} \max \quad & \sum \mu_c\left(\mathbf{y}_{\{c\}}\right) \log\left(\varphi\left(\mathbf{x}_{\{c\}}, \mathbf{y}_{\{c\}}\right)\right) \\ s.t. \quad & \sum_{\mathbf{y}_c} \mu_c\left(\mathbf{y}_{\{c\}}\right) = 1, \forall c \\ & \mu_c\left(\mathbf{y}_{\{c\}}\right) = \sum_{\mathbf{y}_{\{s\}} \text{ consistent with } \mathbf{y}_{\{c\}}} \mu_s\left(\mathbf{y}_{\{s\}}\right) \forall s, \forall c \subset s, \forall \mathbf{y}_{\{c\}} \end{aligned} \tag{2.24}$$

Note that the number of variables needed to represent the assignments of a clique with W members, where each can take V values, is W^V. Hence a potential function defined over 30 binary nodes (as we will encounter when looking at applications of Markov Logic in chapter 5) leads to 2^{30} variables in the ILP. The number constraints for a clique c with W members where each can take V values is at least $1+WV$. Here the $+1$ comes from the constraint that ensures that all $\mathbf{y}_{\{c\}}$ add up to one. The WV constraints come from the fact that there are at least W single-node sub-cliques in c (one for each member) and each of these can take V values.

2.3.2.2 Binary Markov Networks

The mapping we presented in the previous section may lead to ILPs which are prohibitively large because they need, for each potential function, at least as many variables as there are clique states. In this section we present a more compact and tailor-made mapping for binary Markov Networks represented with loglinear features—the type of Graphical Model Markov Logic is based on. By using this mapping we reduce the number of variables significantly. While fewer variables do not necessarily mean shorter runtime, they will always reduce the cost of setting up the the ILP in memory. We are not aware of existing work that uses this mapping. However, it can be seen as as an extension of the work of Mitchell et al. [1997] in which weighted clausal MAX-SAT problems are mapped to ILPs. While they only look at clauses we can take general propositional formulae into account.

This time we look at a log-linear representation of the MAP problem:

$$\arg\max_{\mathbf{y}} \frac{1}{Z} \exp\left(\sum_i f_i\left(\mathbf{y}, \mathbf{x}\right) \cdot \theta_i\right) \tag{2.25}$$

where $\mathbf{y}$ is a binary vector and every f_i is a binary feature function over $\mathbf{y}$ represented as a propositional formula such as

$$f_1(\mathbf{y},\mathbf{x}) = \begin{cases} 1 & \text{if } \neg y_1 \vee x_1 \\ 0 & \text{otherwise} \end{cases} \tag{2.26}$$

Let us start by replacing each feature function application $f_i(\mathbf{y},\mathbf{x})$ in equation 2.25 with a binary auxiliary variable λ_i and constrain $f_i(\mathbf{y},\mathbf{x})$ and λ_i to be equal. This leads to the optimization problem

$$\begin{aligned} \arg\max_{\mathbf{y}} \quad & \sum_i \theta_i \lambda_i \\ s.t \quad & \lambda_i = f_i(\mathbf{y},\mathbf{x}), i = 1 \ldots n \end{aligned} \tag{2.27}$$

with a linear objective function under a set of constraints.

In order to turn this into an ILP we need to transform each equality constraint into a set of linear constraints over $\mathbf{y}$ and the auxiliary variables $(\lambda_i)_i$. The constraint $\lambda_i = f_i(\mathbf{y},\mathbf{x})$ can be transformed to a linear constraint as follows:

1. Mapping each constraint to a logical equivalence of the auxiliary variable and the logical formula the feature is based on, such as

$$\lambda_1 \Leftrightarrow \neg y_1 \vee x_1 \tag{2.28}$$

2. Replacing the x_i variables by their values in $\mathbf{x}$ (i.e. either *true* or *false)*, leading to formulae such as

$$\lambda_1 \Leftrightarrow \neg y_1 \vee false \tag{2.29}$$

3. Transforming the logical equivalence into Conjunctive Normal Form (while eliminating disjunctions that are always true and literals that are always false), as in

$$(\neg\lambda_1 \vee \neg y_1) \wedge (\lambda_1 \vee y_1) \tag{2.30}$$

4. Replacing each disjunction by a linear constraint [Williams, 1999], for example

$$\begin{aligned} -1 \cdot \lambda_1 - 1 \cdot y_1 &\geq -1 \\ 1 \cdot \lambda_i + 1 \cdot y_1 &\geq 1 \end{aligned} \tag{2.31}$$

Note that we can simplify this program significantly for deterministic features f_i (i.e., with very large θ_i) by removing λ_i from the objective and using the logical formula

itself ($\neg y_1 \vee x_1$ in the above case) instead of the equivalence in step 1 as starting point for the constraints to add. We can also subsume the costs of variables λ_i that represent formulae which only contain a single hidden variable y_j to a cumulative cost of the variable y_j and omit the corresponding constraints (this would be possible in the example above).

Using this representation the number of variables can be reduced significantly. If n is the number of binary variables and m is the number of features, this mapping creates $n+m$ binary variables, one for each node and one for each feature. Thus even if the number of nodes included in a potential/feature is high, the number of ILP variables remains low because there is exactly one ILP variable for each feature. For example, even in the case of a clique with 30 binary variables we only need one variable per feature to represent the value of the potential, whereas we needed 2^{30} in the approach of section 2.3.2.1.

The number of constraints obviously depends on the complexity of each feature function. But in many cases (for example, in the models used in this thesis) the number of constraints to be created for each feature roughly corresponds to the number of nodes in the domain of the feature (i.e., number of members in the clique). For example, consider the formula $y_1 \vee \ldots \vee y_n$: converting $\lambda \Leftrightarrow y_1 \vee \ldots \vee y_n$ into Conjunctive Normal Form leads to one large disjunction $\neg\lambda \vee y_1 \vee \ldots \vee y_n$ and n small ones $\lambda \vee \neg y_i$ for $i = 1 \ldots n$. These disjunctions correspond to $n+1$ linear constraints. Hence, in such cases the number of constraints using this representation roughly corresponds to the number of constraints in section 2.3.2.1, leading to an ILP with significantly fewer variables and the same number of constraints.

2.3.3 Simulated Annealing

Another possible option for MAP inference in Markov Networks is *Simulated Annealing* [SA, Kirkpatrick et al., 1983], a general method for optimizing functions over large state spaces. In each step the SA algorithm replaces the current solution by a random "nearby" solution. This solution is chosen with a probability that depends on the difference between the corresponding function values and a *temperature* parameter T which is gradually decreased during the process. In analogy to annealing in metallurgy, where heat causes atoms to become unstuck from their initial positions, the current solution changes almost randomly when T is large, but goes increasingly "uphill" as T decreases to zero.

In contrast to greedy algorithms that always pick the highest scoring local neighbour, SA can avoid to get stuck in local maxima because it allows for downhill moves when the temperature is high. And in comparison with more sophisticated methods, such as Integer Linear Programming, it is easier to implement and may arrive at a reasonable solution in shorter time. However, no guarantees can be made as to how close the solution is to true optimum.

One way to apply Simulated Annealing for MAP inference in a Markov Network is to slightly modify a Gibbs Sampler [Geman and Geman, 1984]. In Gibbs Sampling without SA we generate samples from a distribution by iteratively sampling states for single variables based on their conditional probability with respect to the states of the remaining variables. To incorporate Simulated Annealing, we need to gradually lower the probability of choosing values with low conditional probability. For a binary Markov Network this can be achieved by sampling the value of a variable Y_i according to

$$Pr_T\left(y_i|\mathbf{mb}_i,\mathbf{x}\right) = \frac{\exp\left(\sum_i \theta_i f_i\left(y_i,\mathbf{mb}_i,\mathbf{x}\right)\right)^{1/T}}{\exp\left(\sum_i \theta_i f_i\left(1,\mathbf{mb}_i,\mathbf{x}\right)\right)^{1/T} + \exp\left(\sum_i \theta_i f_i\left(0,\mathbf{mb}_i,\mathbf{x}\right)\right)^{1/T}} \tag{2.32}$$

for a given temperate T.

Algorithm 2.2 shows Simulated Annealing with Gibbs Sampling in pseudo-code. Notice that it requires a *cooling schedule* $\mathbf{T} = \left\{T_1,\ldots,T_{n_{samples}}\right\}$ that gradually lowers temperature. Also note that although strictly speaking the Gibbs Sampler solves a different problem—the generation of the samples from the posterior—it is often used together with SA to find MAP solutions because it is easy to implement and has shown good results in practice [Finkel et al., 2005].

The Markov Networks described by Markov Logic Networks often contain deterministic (or near-deterministic) dependencies, such as "no more than one agent" constraints for Semantic Role Labelling or transitivity constraints for Coreference Resolution. In such settings a Gibbs Sampler often fails because we would need to change several variables at once in order jump from one consistent solution to another. This problem has led to the development of MC-Sat [Poon and Domingos, 2006], a sampler tailored to draw from distributions with deterministic dependencies. Another option for Simulated Annealing in (binary) Markov Networks would hence be to adapt MC-Sat in the same fashion as we have extended the Gibbs Sampler above.

Algorithm 2.2 Gibbs Sampling with Simulated Annealing for MAP inference.

Require: A Markov Network M, number of samples $n_{samples}$
and a cooling schedule $T = \{T_1, \ldots, T_{n_{samples}}\}$

$r \leftarrow 1$
while $r \leq n_{samples}$ **do**
  $i \leftarrow 1$
  $\mathbf{y} \leftarrow random$
  while $i \leq n_{variables}$ **do**
    pick node Y_i in M
    sample y_i from according to $Pr_T(y_i|\mathbf{mb}_i, \mathbf{x})$
    $i \leftarrow i + 1$
  end while
  $r \leftarrow r + 1$
end while
return $\mathbf{y}$ with highest probability

2.3.4 Alternative Methods

The MAP methods we presented here have either been applied in the context of Markov Logic (MaxWalkSAT, ILP), or could be readily applied because they are based on sampling algorithms that have already been implemented for Markov Logic (Simulated Annealing with Gibbs Sampling or MC-Sat). For other methods, such as (Max-Product) Belief Propagation and its variants we refer the reader to the literature [Murphy et al., 1999, Wainwright et al., 2003, Pearl, 1988, Globerson and Jaakkola, 2007].

2.4 Learning

One of the key tasks addressed by this thesis is finding a $\mathbf{y}$ that maximizes $Pr(\mathbf{y}|\mathbf{x}, \Theta)$ for a loglinear model when both the observation $\mathbf{x}$ and the weights Θ are already given. However, this is just one side of the story. Initially Θ is not available and has to be estimated from data. In this section we will therefore give a brief overview of the methods we used to learn the parameters Θ of the loglinear models used in this thesis.

Supervised learning is the process of finding a set of suitable parameters/weights Θ when we are given a sequence of training instances $\mathcal{D} = (\mathbf{y}_i, \mathbf{x}_i)_{i=1}^m$ where each $\mathbf{x}_i$ is an assignment to the observable variables and each $\mathbf{y}_i$ an assignment to the hidden vari-

ables in the probabilistic model. This is to be contrasted with *unsupervised learning*, where the $\mathbf{y}_i$ are not available. Our models in chapter 5 are trained in a setting where the $\mathbf{y}_i$ are available, and hence we will focus on supervised methods in this section.

Supervised training methods can be divided into two broad classes, discriminative and generative [Ng and Jordan, 2001]. In *generative learning* the aim is to find a set of parameters Θ so that the dataset $\mathcal{D}$ is likely to be generated by the distribution parametrised by Θ. More formally, we search for a set of parameters Θ that maximize the likelihood of the data $\mathcal{D}$, that is

$$\arg\max_{\Theta} Pr(\mathcal{D}|\Theta) \tag{2.33}$$

where we typically assume that the data $\mathcal{D}$ is *independent identically distributed* (i.i.d.):

$$Pr(\mathcal{D}|\Theta) = \prod_{i}^{m} Pr(\mathbf{y}_i, \mathbf{x}_i|\Theta) \tag{2.34}$$

Alternatively, we can try to maximise the *a posteriori* probability of the data

$$\arg\max_{\Theta} Pr(\mathcal{D}|\Theta) \cdot Pr(\Theta) \tag{2.35}$$

assuming that we have access to a *prior* probability for parameters, $Pr(\Theta)$.

In *discriminative learning* we are merely interested in the performance of the predictor

$$\mathbf{h}_{\Theta}(\mathbf{x}) = \arg\max_{\mathbf{y}} Pr(\mathbf{y}|\mathbf{x}, \Theta)$$

and we train weights to minimize the expected number of errors the predictor will make.[8] One way to pursue this is to maximize the conditional likelihood of the data (instead of the joint likelihood as above) [Lafferty et al., 2001]

$$\arg\max_{\Theta} Pr(\mathcal{D}_{\mathbf{y}}|\mathcal{D}_{\mathbf{x}}, \Theta) \tag{2.36}$$

where $\mathcal{D}_{\mathbf{x}} = (\mathbf{x}_i)_{i=1}^{m}$ and $\mathcal{D}_{\mathbf{y}} = (\mathbf{y}_i)_{i=1}^{m}$. In other words, we want to ensure that our model assigns the highest probability to the pair $(\mathcal{D}_{\mathbf{y}}, \mathcal{D}_{\mathbf{x}})$ when compared to all pairs with $\left(\mathcal{D}'_{\mathbf{y}}, \mathcal{D}_{\mathbf{x}}\right)$ and $\mathcal{D}_{\mathbf{y}} \neq \mathcal{D}'_{\mathbf{y}}$. We do not care whether the pair $(\mathcal{D}_{\mathbf{y}}, \mathcal{D}_{\mathbf{x}})$ has the highest probability in comparison to all pairs $(\mathcal{D}_{\mathbf{y}}, \mathcal{D}'_{\mathbf{x}})$ with $\mathcal{D}_{\mathbf{x}} \neq \mathcal{D}'_{\mathbf{x}}$. This usually means we can use the trained model to predict the hidden variables given some observation, but cannot use the model to generate observations based on the hidden variables in a way that is faithful to the training data.

[8] We can also minimise other loss functions such as an F1-based metric or a loss function that assigns different weight to different types of errors.

Instead of maximizing the probability of the data we can also try to enforce a margin [Taskar et al., 2004] between the score $s(\mathbf{y}_i, \mathbf{x})$ of the true solution and the score $s(\mathbf{y}', \mathbf{x})$ of any solution $\mathbf{y}'$ different to $\mathbf{y}_i$ for every instance i in the data. Here $s(\mathbf{y}, \mathbf{x}) = \sum_j \theta_j f_j(\mathbf{x}, \mathbf{y})$ (equivalent to equation 2.19). By enforcing a margin we try to avoid cases where the best solution $\mathbf{y}_i$ might have the highest probability but the probability of a wrong solution is only slightly lower—this can lead to errors at test time even if the test instances are almost identical to those of the training data.

2.4.1 Pseudo-Likelihood

In order to maximise the likelihood of the training data, it is necessary to calculate its gradient with respect to the parameters Θ several times. This is computationally expensive and often intractable [Richardson and Domingos, 2006] because it involves marginal inference in the current model.

A more efficient alternative is to maximize the pseudo-likelihood $Pr^*(\mathcal{D}|\Theta)$ of the data [Besag, 1975]. Assume that our training data is $\mathcal{D} = (\mathbf{y}_i)_i^m$. Then

$$Pr^*(\mathcal{D}|\Theta) = \prod_{i=1}^{m} Pr^*(\mathbf{y}_i|\Theta) \tag{2.37}$$

with

$$Pr^*(\mathbf{y}|\Theta) = \prod_{j=1}^{n} Pr\left(y_j|\mathbf{mb}_j, \Theta\right) \tag{2.38}$$

where $\mathbf{mb}_j$ denotes the state of the Markov Blanket of Y_j in $\mathbf{y}$.[9] That is, the pseudo-likelihood $Pr^*(\mathbf{y}|\Theta)$ of an assignment $\mathbf{y}$ is composed of the probabilities of all its components conditioned on the state of their neighbours.

The gradient of $Pr^*(\mathbf{y}|\Theta)$ can be efficiently calculated even for large and densely connected networks [Richardson and Domingos, 2006]; this means that optimizing the pseudo-likelihood of the training set is a practical option for estimating the weights of a Markov Network. However, generally we cannot expect the performance of the Pseudo-likelihood estimates for Θ to be as good as the performance of Maximum Likelihood or MAP estimates [Sutton and McCallum, 2007].

2.4.2 Online Learning

We mentioned that maximising the likelihood of the data can be computationally expensive. The problem is even more severe if we consider maximising the conditional

[9] Note that we subsumed $(\mathbf{y}, \mathbf{x})$ into $\mathbf{y}$ for notational reasons in order simplify the definition of pseudo-likelihood.

likelihood of the the training data. Here we not only need to perform marginal inference several times for each training instance, we also need to repeatedly calculate the normalizer $Z_{\mathbf{x}}$ (see section 2.3) for each training instance.

An alternative and often more practical option for discriminative weight training is Online Learning. Here the current model is updated in an instance-by-instance manner. For every instance i in the dataset we:

1. perform MAP inference to find the best hidden solution $\mathbf{y}'$ (or n-best solutions) given the current parameters Θ and the observation $\mathbf{x}_i$;

2. compare solution $\mathbf{y}'$ with the *gold* solution $\mathbf{y}_i$ and update weights Θ based on this comparison.

Note that in each step we use the model Θ that we have updated in the previous step. Once we reach the end of the corpus we have finished what is called an *epoch*. Usually we would run many such epochs (in our experiments we ran 4 epochs). After we have finished all epochs we can (a) return the last weight vector or (b) return the average of the weight vectors produced for each instance in each epoch. The averaged weight vector essentially corresponds to a model that uses all weight vectors to vote on the current instance; in practice option (b) is often preferred over (a) because voting-based approaches tend to be more robust than approaches based on a single model.

Online Learning algorithms can be distinguished by how they update the model in each step. A very simple way to update the weights Θ of a linear discriminative function $\Theta^{\top}\mathbf{f}(\mathbf{x},\mathbf{y})$ (or a corresponding log-linear model as presented in section 2.1.2.1) is the perceptron update rule [Collins and Roark, 2004, Collins, 2002]

$$\Theta_{i+1} = \Theta_i + \alpha \cdot (\mathbf{f}(\mathbf{x}_i,\mathbf{y}_i) - \mathbf{f}(\mathbf{x}_i,\mathbf{h}_{\Theta}(\mathbf{x}_i))) \tag{2.39}$$

where α is a weight that controls the learning rate. In each update the weight vector is moved further in the direction of the true feature vector $\mathbf{f}(\mathbf{x}_i,\mathbf{y}_i)$ and further away from the guessed feature $\mathbf{f}(\mathbf{x}_i,\mathbf{h}_{\Theta}(\mathbf{x}_i))$, assuming that $\mathbf{y}_i$ differs from $\mathbf{h}_{\Theta}(\mathbf{x}_i)$. If $\mathbf{y}_i = \mathbf{h}_{\Theta}(\mathbf{x}_i)$ then no update is performed.

The perceptron rule may sometimes be a suboptimal choice. Say we have trained our model Θ so far that $\mathbf{h}_{\Theta}$ predicts the correct $\mathbf{y}_i$ for the current observation $\mathbf{x}_i$. Then it is obvious that no update will be made. However, it is possible that the current optimum $\hat{\mathbf{y}}$ is only slightly better than the best runner-up $\mathbf{y}'$ in terms of score $s(\hat{\mathbf{y}},\mathbf{x})$. And while this was sufficient for this current example, examples in the test set only need to be slightly different in order to confuse the predictor $\mathbf{h}_{\Theta}$.

MIRA [Crammer and Singer, 2003] is an Online Learning algorithm that is designed to avoid this problem by establishing a large margin between the score of the gold solution $s(\mathbf{y}_i)$ and all wrong solutions $s(\mathbf{y}), \mathbf{y} \neq \mathbf{y}_i$. This is achieved by solving the quadratic program

$$\begin{aligned} \min \quad & \|\Theta_i - \Theta_{i-1}\|^2 \\ s.t. \quad & \Theta_i^\top \mathbf{f}(\mathbf{y}_i) - \Theta_i^\top \mathbf{f}(\mathbf{y}) \geq L(\mathbf{y}_i, \mathbf{y}) \quad \forall \mathbf{y} \neq \mathbf{y}_i \end{aligned} \tag{2.40}$$

in each iteration i. That is, in each step i MIRA tries to find a weight vector Θ_i which guarantees that the difference in score of the right solution $\mathbf{y}_i$ and a wrong solution $\mathbf{y}_j$ is as least as big as the (user-defined) loss $L(\mathbf{y}_i, \mathbf{y})$ we take when predicting $\mathbf{y}$ instead of $\mathbf{y}_i$, while trying to change Θ_{i-1} as little as possible. The loss function could, for example, be the number of wrong labels in a Semantic Role Labelling task.

In general the above Quadratic Program is too large to solve. For example, in a graph with n binary nodes there are $2^n - 1$ possible $\mathbf{y}'$ different from $\mathbf{y}_i$. Instead of solving the complete Quadratic Program one commonly only considers the set of k solutions $\mathbf{y}'$ with highest scores $s(\mathbf{y}')$ [McDonald et al., 2005].

The advantage of Online Learning algorithms in comparison to many other training methods (such as optimizing the conditional likelihood) is the relatively low computational overhead, provided that we can efficiently find or approximate the MAP solution for a given observation. It is also relatively easy to implement such algorithms because MAP inference is often already available (it is needed when we test the model).

2.5 Conclusion

In this chapter we have presented Markov Networks as undirected Graphical Models that describe probabilistic correlations between sets of random variables. We also showed how to represent Markov Networks as log-linear models in which potential functions are compactly described through feature functions. We illustrated the use of this representation based on the Semantic Role Labelling example in section 1.2. Finally we presented two methods for MAP inference and two methods for learning the weights of a Markov Network. This included the introduction of a novel ILP formulation for MAP inference in binary networks, tailored for the Markov Network framework we will present in chapter 3.

While we have shown how Markov Networks can be used to model our intuition about a problem, the reader may have noticed how complex and repetitive the cor-

responding networks can become. In the next chapter we will present a framework that allows us to represent our intuitions more compactly without having to explicitly construct complex Markov Networks. This will not only lessen the amount of code a developer has to write in order to make use of Markov Networks, it will also be helpful in order to improve the efficiency of MAP inference because in many cases the complete network does even need to be instantiated in order to find the most likely solution.

Chapter 3

Markov Logic

Markov Logic [Richardson and Domingos, 2006] is a combination of First Order Logic and Markov Networks. There are two possible ways of motivating and describing it (see figure 3.1). One starts at Markov Networks and asks how we can describe their structure in a compact fashion, make learning and inference more efficient, automate the process of finding good network structures and minimize the burden of constructing such networks. This resembles the step from Propositional Logic to First Order Logic and is indicated with "Abstraction" edges in figure see "Abstraction" edge between Markov Networks and Markov Logic in figure 3.1. The other way starts at First Order Logic and asks how to extend it in order to cope with the noise, fuzziness and ambiguities inherent to the real world. This requires the introduction of uncertainty and resembles the step from Propositional Logic to Markov Networks, as denoted by the "Uncertainty" edges. Both points of view will be described in this chapter after we present some terminology and concepts of First Order Logic in the next section.

Note that there are other flavours of logics and probabilistic frameworks we can plug into the boxes of Figure 3.1. For example, we could also look at Bayesian Networks and Bayesian Logic Programs [Kersting and De Raedt, 2000] and Prolog instead of First Order Logic. However, in this thesis we only consider Markov Logic, (a) because we want to make use of the advantages of Markov Networks we described in 2.1.2, and (b) because Markov Logic can be shown to generalise formalisms that combine logic and probability [Richardson and Domingos, 2006].

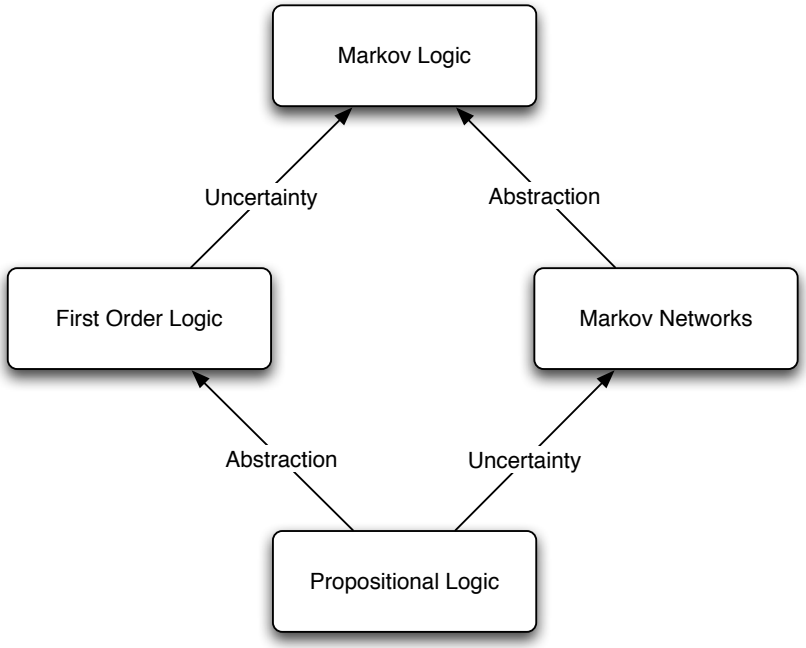


Figure 3.1: Relationships between Propositional Logic, First Order Logic, Markov Networks and Markov Logic.

3.1 First Order Logic

Logic is used to reason about the world in a formal way. It allows us to characterise the difference between valid and invalid arguments. More importantly for our purposes, it enables machines (formal systems) to reason about the world in an automatic fashion. This requires us to (a) define the syntax of a formal language that allows us to describe claims about the world (see section 3.1.1) and (b) formally define what sentences of this language mean (see section 3.1.2).

Simply put, *Propositional Logic* describes the world with atomic claims (propositions) that can be combined through logical connectives such as "and" "or" and "if-then". For example, we could state that "Constituent 1 overlaps with constituent 2 AND constituent 2 is an A1 argument" and "IF Constituent 1 overlaps with constituent 2 AND constituent 2 is an A1 argument THEN constituent 1 cannot be an A1 argument". A typical inference step in Propositional Logic would then deduce that "Constituent 1 cannot be an A1 argument".

The above statements are rather specific to "Constituent 1" and "Constituent 2", even though the claims would also hold for any sets of constituents. In *First Order Logic* (FOL) we can make more general statements, such as "If some constituent x overlaps with some constituent y and constituent x is an A1 argument, then constituent

y cannot be an A1 argument" by formally introducing the notions of *quantification* ("some person x") and *relations* between these ("x overlaps with y"). This has obvious advantages: it allows us to describe the world more succinctly and to make general statements that are applicable in a wide range of contexts.

In this section we will give an introduction to some of the terminology and concepts needed in later sections and chapters. Note that the definitions presented in this section will be tailored towards Markov Logic. In particular, we will define the semantics of a First Order Logic formula in terms of possible worlds (or Herbrand Interpretations [Genesereth and Nilsson, 1987]) instead of first order interpretations and variable assignments.[1]

3.1.1 Syntax

To define the syntax of First Order Logic we first need to describe its vocabulary.

Definition 3.1. A *vocabulary* $V = (P, F, C, Y)$ consists of:

- A finite set P of predicates, each with an associated arity.
- A finite set F of function constants, each with an associated arity.
- A finite non-empty set C of object constants.
- A finite set of variables Y.

Note that we will assume that all capitalized symbols, numerals and quoted strings are object constants. The type of other symbols can be determined from the context. If the function f or predicate p has arity n, we will often refer to them as f/n or p/n, respectively. Moreover, we will denote the arity of a predicate p or function f with $arity(p)$ or $arity(f)$, respectively.

For example, consider again the case of Semantic Role Labelling, where we want to label constituents with the role they play with respect to a given verb. A possible way to model this task could use a binary predicate *role/2* that relates a constituent to its label. Considering the example sentence we looked at in section 2.2, it might also be helpful to represent information about the Part of Speech (PoS) tag of the head

[1] In fact, First Order Logic as described in this chapter is closer to Herbrand Logic [Hinrichs and Genesereth, 2006], a formalism that combines First Order Logic and Herbrand semantics, than it is to traditional First Order Logic.

word of a constituent using a binary *pos/2* predicate that relates constituents to their head PoS.

It could be potentially useful to take the distance between the verb and the head word of a constituent into account. To this end we could introduce a function *distance* which relates each constituent to this distance. It could also be helpful to know the constituent's parent node which we could represent using a *parent* function (which could map the root constituent to some artificial node in order to be total).

Moreover, we will have to make statements about the properties of actual constituents. This requires us to use object constants that represent constituents, words, PoS tags etc. We will use integer constants to represent constituents, and as constants for PoS tags the PoS tags themselves (assuming they are capitalized). Finally, we will use the symbols $x, y, z, \ldots$ as variables to make more general statements about our domain.

The above requirements would lead to the following vocabulary

$$\begin{aligned} V \quad = \quad & (\{role, pos\}, \{distance, parent\}, \\ & \{1, 2, 3, \text{A1}, \text{AM-TMP}, \text{NN}, \text{IN}, \ldots\}, \{x, y, z, \ldots\}) \end{aligned} \tag{3.1}$$

As mentioned above, FOL allows us to make claims about the relations that objects participate in. Thus it needs to provide a language construct that denotes objects: a *term*.

Definition 3.2. An expression is a *term in V* if and only if it is

- a variable in V,
- an object constant in V or,
- a function constant in V with arity n applied to n terms in V.

For example, if we consider the vocabulary V in equation 3.1 then the following expressions are terms in V:

$$x, 1, \text{A1}, distance(1), distance(parent(1)), \ldots \tag{3.2}$$

Definition 3.3. A term that does not contain any variables is a *ground term*.

For example, $A1$, *distance(1)* and *distance(parent(1))* are ground terms, x and *parent(x)* are not.

We can now make statements about the relations that hold among objects. We will refer to such a statement as *formula* and define it recursively as follows.

Definition 3.4. An expression is a *formula in V* if and only if it is

- a predicate with arity n applied to n terms in V,
- $(\neg\phi)$ where ϕ is a formula,
- $(\phi \vee \psi)$ where ϕ and ψ are formulae,
- $(\phi \wedge \psi)$ where ϕ and ψ are formulae,
- $(\phi \Rightarrow \psi)$ where ϕ and ψ are formulae,
- $(\phi \Leftrightarrow \psi)$ where ϕ and ψ are formulae,
- $\forall x.\phi$ where ϕ is a formula and x a variable or
- $\exists x.\phi$ where ϕ is a formula and x a variable.

For instance, the expression

$$\forall x.pos\,(x, \text{IN}) \Rightarrow role\,(x, \text{AM-TMP}) \tag{3.3}$$

is a formula in V of equation 3.1.

A formula that matches the first rule of the above definition is called an *atom* or *positive literal*. A negated $(\neg\phi)$ atom is a *negative literal*. A formula that contains only ground terms is referred to as a *ground formula*. A ground formula which is an atom is a *ground atom*. In a *closed formula* all variables are quantified in an enclosing universal $\forall$ or existential $\exists$ quantification. By contrast, an *open formula* contains variables that are not quantified (free).

3.1.2 Possible World Semantics

One way of interpreting a formula such as

$$\forall x, y.role\,(x, \text{A1}) \wedge x \neq y \Rightarrow \neg role\,(y, \text{A1}) \tag{3.4}$$

is as a statement that expresses our belief about what the world looks like. There are many possible worlds, and we do not know exactly which of these worlds is the one we live in, but we do know that in each possible world there can only be one A1 (agent) argument for a verb. In other words, we can think of a first order formula (or a set of such formulae) as a constraint on the set of possible worlds.

In Markov Logic, and other probabilistic logic formalisms [Halpern, 1990], we go further. Instead of classifying worlds as either possible or impossible, we assign probabilities or degrees of belief to them. Before we can do so, we need to formalise the notion of a possible world and what it means for a formula to hold in such a world.[2]

Definition 3.5. A *possible world M* for a vocabulary V is a set of ground atoms in V that do not contain function symbols.

For example, the set

$$\{role(1, \mathrm{A1}), pos(1, \mathrm{IN})\} \tag{3.5}$$

is a possible world for the vocabulary V of equation 3.1. The set

$$\{role(1, \mathrm{A1}), pos(parent(1), \mathrm{IN})\} \tag{3.6}$$

is not a possible world because the ground term $parent(1)$ contains a function symbol.

Note that in practice we will often want to use *typed* predicates. These require that the arguments of ground atoms in a possible world have a certain type. For example, ground atoms of the *role/2* predicate should only have argument tuples of the type $Int \times Role$ where *Int* is the set of integer constants and *Role* is the set of possible role label constants.

Why do we forbid function symbols in the definition of a possible world? Because we have assumed the set of constants to be finite, the set of all ground atoms that do not contain function symbols is finite, too. Once we allow function symbols, the set of possible ground atoms becomes infinite. This would greatly complicate inference within this space, as well as the definition of a probability distribution over it.[3]

It is worthwhile mentioning that by restricting ourselves to a finite set of possible worlds we cannot be more expressive than Propositional Logic: there is no domain we can describe that we cannot describe with Propositional Logic, and vice versa [Hinrichs and Genesereth, 2006]. However, using a first order representation still has several advantages; for example, it allows us to describe our domain knowledge more compactly and do more efficient inference—a fact we will make use of in the algorithm presented in chapter 4.

[2]Note that we can also think of a possible world as a restricted version of a Herbrand Interpretation [Genesereth and Nilsson, 1987].

[3]Recent work [Singla and Domingos, 2007] has shown how to extend Markov Logic to infinite domains; however, it is not yet clear how inference in such domains should be performed. Moreover, in our applications the domains are indeed finite and hence finite possible worlds are sufficient for our purposes.

The fact that we do not allow function symbols in possible worlds does not mean that we cannot use function symbols in our formulae, as long as we know the value of each function applied to every possible tuple or arguments (with appropriate arity) in advance.[4] In this case the infinite number of terms we can construct from the functions and constants in our vocabulary can be ignored, because every term that contains function symbols can be recursively replaced by a constant. For example, if we know that $parent(1) = 2$ and $parent(2) = 3$, then the ground atom $role(parent(parent(1)))$ can be represented by the ground atom $role(3)$.

In Markov Logic we speak about formulae being true or false in a possible world. But what does it exactly mean for a formula such as the one in equation 3.4 to be true or false in terms of our previous definitions? We answer this question through the following definition.[5]

Definition 3.6. Let ϕ be a closed sentence, M a possible world in the vocabulary V and T a function over ground terms that maps each constant to itself and each function application recursively to the constant it corresponds to (assuming known functions). Then we say that M *satisfies* ϕ and write $\models_M \phi$ depending on the type of ϕ:

- $\models_M p(t_1, \ldots, t_n)$ for the ground terms $t_1, \ldots, t_n$ in V if and only if $p(T(t_1), \ldots, T(t_n)) \in M$.
- $\models_M \neg\phi$ if and only if $\not\models_M \phi$
- $\models_M \phi \wedge \psi$ if and only if $\models_M \phi$ and $\models_M \psi$
- $\models_M \phi \vee \psi$ if and only if $\models_M \phi$ or $\models_M \psi$
- $\models_M \phi \Rightarrow \psi$ if and only if $\not\models_M \phi$ or $\models_M \psi$
- $\models_M \phi \Leftrightarrow \psi$ if and only if either $\models_M \phi \wedge \psi$ or $\models_M \neg\phi \wedge \neg\psi$
- $\models_M \forall x.\phi(x)$ if and only if $\models_M \phi(c)$ for all constants c in V.
- $\models_M \exists x.\phi(x)$ if and only if $\models_M \phi(c)$ for some constants c in V.

For example, the possible world in equation 3.5 does not satisfy the sentence

$$\forall x.pos(x, \text{IN}) \Rightarrow role(x, \text{AM-TMP}) \tag{3.7}$$

[4] In other words, we may have only partial knowledge about the relations of the actual world, but we have complete knowledge about its functions.

[5] Note that the definition we give here essentially corresponds to the definition of satisfaction in Herbrand Logic [Hinrichs and Genesereth, 2006].

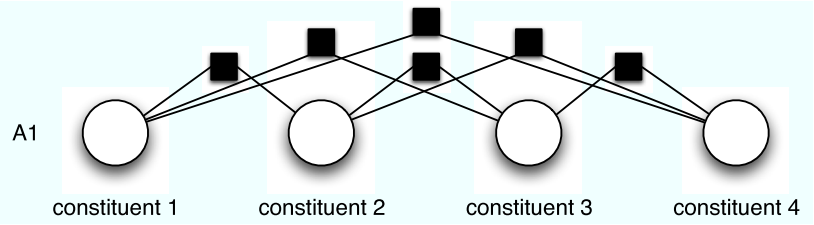


Figure 3.2: A part of the Markov Network for finding the semantic roles of the sentence in section 2.2.

because

$$pos\,(1,\text{IN}) \Rightarrow role\,(1,\text{AM-TMP}) \tag{3.8}$$

is not satisfied: the premise is true but the conclusion is not.

We say that an interpretation M satisfies the set of sentences Δ and write $\vDash_M \Delta$ if M satisfies each sentence in Δ.

Note that in the following we will often identify the binary vector $\mathbf{y} = \left(y_{p(\mathbf{a})}\right)$ with the possible world $\left\{p\,(\mathbf{a})\,|y_{p(\mathbf{a})} = 1\right\}$, where p ranges over the predicates in V and $\mathbf{a}$ over all constant tuples with an arity corresponding to the arity of p. If not mentioned otherwise, the components of $\mathbf{y}$ are ordered according to the lexicographical order on predicates and object constants. For instance, we could represent the possible world in equation 3.5 using $\mathbf{y} = (1,0,1,\ldots)$ if we assume that atoms are ordered as $\{role\,(1,\text{A1})\,, role\,(1,\text{AM-TMP})\,, pos(1,\text{IN}),\ldots\}$. Here y_1 and y_2 are active because *role*(1,A1) and *pos*(1,IN) appear in the first and third position.

3.2 Abstraction for Markov Networks

Markov Networks are propositional. They describe the probabilistic dependencies between atomic propositions. If we go back to the example given in section 2.2 we note that nodes represent propositions such as “constituent 1 has the role A1” and potentials/factors encode dependencies such as “constituent 1 has not more than one role”. A look at the Markov Network of figure 3.2, a part of figure 2.5 in chapter 2, shows that some nodes have similar interpretations and that some edges follow a certain pattern. Here each node indicates whether a certain constituent is labelled with the A1 role, and the edges which pairwise connect these nodes all forbid solutions in which two constituents have the same role.

Markov Logic, and other formalisms such as Relational Markov Networks [Taskar

et al., 2002] or Probabilistic Relational Models [Koller, 1999],[6] is about describing these regularities in a formal fashion. This has a few motivations:

Efficiency With an abstract description of the graphical structure and potentials of a Markov Network it is possible to avoid its full instantiation when doing inference and learning [Haddawy, 1994, de Salvo Braz et al., 2005, Poole, 2003].

Induction With a language that describes the structure of Markov Networks it is possible to reason about this structure and induce new structural patterns [Kok and Domingos, 2005, Richardson and Domingos, 2005].

Automation Modelling problems with Markov Networks requires us to construct highly repetitive structures for each problem instance to solve. With a description language for such structures we can create networks declaratively [Haddawy, 1994].

Decoupling A description language allows us to separate Machine Learning technology from Machine Learning applications and fosters independent progress of both [Domingos, to appear].

Compactness A description language allows us to describe Markov Networks compactly. This does not only lead to lower memory requirements when we want to store Markov Networks, it also allows others to comprehend such networks more easily.

It is the first of these reasons, *efficiency*, that we will further illustrate in this thesis—we will show how the efficiency (and accuracy) of MAP inference can be improved by exploiting a first order description of a Markov Network.

The support of structure *induction* is important if we want to further reduce the amount of work necessary to design Markov Networks that work well in practice. Often the design of a good feature set (i.e., the graphical structure of a Markov Network) is crucial to the performance of the network, and the automation of this process can lessen the amount of mechanical work carried out by the developer. Moreover, it can lead to networks that perform better than those designed by hand [Kok and Domingos, 2005].

Automation of the network construction process [Haddawy, 1994] also helps to reduce the amount of mechanical labor to be carried out by the developer. When using

[6] Note that Probabilistic Relational Models describe Bayesian Networks instead of Markov Networks.

Markov Networks we need to write new wrapper code that generates Markov Networks for every task we want to tackle (this also holds for other propositional languages). In Markov Logic this process can be automated. This does not directly improve the performance of our system; for example, the Markov Networks for Semantic Role Labelling we have described so far will not yield higher accuracy if we generate them automatically. However, this declarative approach can reduce development time and eventually lead to better performance because more time can be spent on understanding and modelling the problem.

In fields such as databases and programming languages,[7] *decoupling* of technology and applications has significantly increased the speed of development [Domingos, to appear]. Without decoupling, both the design of a successful applications and the development of new Machine Learning methods require Machine Learning experts as well as domain experts. With decoupling, Machine Learning experts can focus on Machine Learning methods, and domain experts can focus on modelling their domain. Note that while Markov Networks can already be used for decoupling, a higher order description language takes this further and allows us to provide more functionality for the application side (such as inference algorithms that exploit the regular and repetitive structure of Markov Networks).

Last but not least, languages such as Markov Logic help us to describe Markov Networks more *compactly*. For example, consider a binary Markov Network for Semantic Role Labelling (akin to the one in figure 2.5 but for a longer sentence). Such network could contain thousands of nodes, and much more edges. However, by using Markov Logic we can describe such network using a few weighted first order formulae (see section 5.3.1). This helps to reduce the memory footprint of a Markov Network. More importantly, it allows us to explain, comprehend and extend Markov Networks more easily.

3.3 Uncertainty for Logic

Until the late 50s, approaches to Natural Language Processing based on a statistical model of language were dominant [Lee, 2004]. In essence, these approaches assigned probabilities to hypotheses based on the detection of simple word patterns and their frequency in corpora. For example, Shannon [1948] calculated the probability of the

[7] Considering the widespread use of generic Machine Learning software such as CRF packages, one can argue that the same can already be observed in the field of Machine Learning.

word A at the next position within a sentence based on the previous word B and the frequency of seeing B followed by A in known texts of English. Likewise, Mosteller and Williams used the frequency of certain word patterns in text in order to determine whether it was Alexander Hamilton or James Madison who wrote some of the pseudonymous Federalist Papers [Lee, 2004].

However, there was a problem with this approach: there are word patterns which have never been seen before but are still likely to occur (this is often referred to as the *data sparsity* problem). Simple statistical models based on observed frequencies of word pattern assign zero probability to these patterns. Yet, humans are able to interpret sentences containing such word patterns, because they have access to both syntactic and semantic background knowledge [Chomsky, 1957]. To overcome this problem, researchers began to focus on incorporating (rule-based) background knowledge into their models of the world. SHRDLU [Winograd, 1971], a system that interprets and generates natural language sentence about an artificial block world, and the work on LADDER [Hendrix et al., 1978], a natural language interface to a database of Naval information, are two prominent examples of this approach.

Such systems made use of a knowledge base that captured the domain knowledge necessary to perform a certain task. A knowledge base would be formulated in a logic-like language such as PLANNER [Winograd, 1971]. However, PLANNER, PROLOG and other languages did not allow knowledge to partial: every rule in a knowledge base would always have to hold. Designing such rules in limited domains would not pose a problem. Yet, for more complex tasks, such as the parsing of general English text, it became increasingly difficult. Either a rule was very specific and would only apply in a very limited number of circumstances, or it would be general but violated in many cases.

By contrast, system like LADDER [Hendrix et al., 1978] allowed rules to be violated. In order to produce a single output, such systems took a user-defined order of rules into account: if there were several contradicting rules applicable in a certain context, the one with highest order would be chosen. However, after a knowledge base reached a critical size, the human cost of extending coverage would rise significantly because small changes to the rule base and the order of rules could erode the integrity of the whole system [Slocum, 1981].

In the beginning of the 80s, statistical methods became popular again. The success of statistical Speech Recognition showed that with more sophisticated computational resources and methods a probabilistic approach can outperform purely knowledge-

based systems [Lee, 2004]. Statistical approaches, such as the HMM-based Speech Recognition systems of IBM, still made use of rules (for example, embodied in the state transition matrices of an HMM). This time rules would contradict each other, and the hypothesis a system generates was not the one for which all rules hold, or rules of certain rank, but the one that maximises the product of the probabilities of all rules satisfied in the hypothesis.

However, since logic and knowledge based approaches were very successful for small domains without much noise or ambiguity, Nilsson [1986] started to ask how they could be extended in order to cope with the uncertainty that inevitably arises in large domains. In the following years other researchers would follow (see section 3.8). At the core of most of this work, is a change of semantics: a knowledge base does not describe a set of possible worlds anymore; instead, it describes a probability distribution over such worlds. In other words, we stop classifying hypotheses into being either true or false and give them a probability or score instead. Markov Logic is a very simple way of providing such semantics.

3.4 Markov Logic Networks

In First Order Logic a knowledge base is a set of formulae. In contrast, Markov Logic describes a knowledge base as a set of weighted formulae. This set of weighted formulae is referred to as a Markov Logic Network and defines a log-linear probability distribution over possible worlds.

Definition 3.7. Let V be a vocabulary, then a *Markov Logic Network (MLN) L* in V is a set of pairs $\{(\phi_i, w_i)\}_i$ where each ϕ_i is a First Order Logic formula in V and each w_i is a real value (the weight).

For a given set of constants a Markov Logic Network defines a Markov Network as follows.

Definition 3.8. Let V be a vocabulary and L a Markov Logic Network in V, then the *Ground Markov Network* $M_{L,V}$ is defined as follows:

1. $M_{L,V}$ contains one binary node $Y_{pred}^{a_1,\ldots,a_n}$ for each predicate *pred* in V and each tuple $a_1,\ldots,a_n$ of constants in V with $n = arity(pred)$ that represents the truth state of the ground atom $pred(a_1,\ldots,a_n)$.

2. $M_{L,V}$ contains one feature f_B^{ϕ} for each formula ϕ in the MLN L and each binding B of the free variables in ϕ with

$$f_B^{\phi}(\mathbf{y}) = \begin{cases} 1 & \text{if } \models_{\mathbf{y}} \phi[B] \\ 0 & \text{otherwise} \end{cases} \tag{3.9}$$

   where $\phi[B]$ is the ground formula we create by replacing each free variable in ϕ with the corresponding constant in the binding B and recursively expanding each existential quantification with a corresponding disjunction and each universal quantification with a corresponding conjunction.[8]

If not mentioned otherwise, we will assume that the vocabulary we use to ground the Markov Network consists of all symbols in the Markov Logic Network and denote the Ground Markov Network with M_L. Alternatively we sometimes specify a set of constants C and use $M_{L,C}$ to refer to the Ground Markov Network $M_{L,C}$ where V consists of the predicate, function and variable symbols in L and the constants C.[9]

The Ground Markov Network $M_{L,V}$ defines a loglinear distribution over the family of variables $\mathbf{Y} = \left(Y_{pred}^{\mathbf{a}}\right)$ where *pred* ranges over the set of predicates in V and $\mathbf{a}$ over the possible constant tuples in V for *pred*:

$$Pr(\mathbf{Y} = \mathbf{y}) = \frac{1}{Z} \exp\left(\sum_{(\phi,w) \in M} w \cdot \sum_{B \in \mathcal{B}(\phi)} f_B^{\phi}(\mathbf{y}) \right) \tag{3.10}$$

where $\mathcal{B}(\phi)$ contains all bindings of the free variables in ϕ with the constants in V.[10]

Since each $\mathbf{y}$ identifies a possible world, the equation 3.10 also describes a distribution over the set of possible worlds. Roughly speaking, a possible world becomes more likely the more groundings of formulae with positive weight are true and the more groundings with negative weight are false. More precisely, assume that there are two possible worlds $\mathbf{y}_1$ and $\mathbf{y}_2$ for which there is only one formula ϕ (with weight w) which holds in $\mathbf{y}_1$ but does not hold in $\mathbf{y}_2$. Then the probability of the possible world $\mathbf{y}_1$ is $\exp(w)$ times higher than the probability of $\mathbf{y}_2$.

It can also be easily shown that in the limit of infinite (positive) weights a Markov Logic Network corresponds to a First Order Logic knowledge base that strictly disallows any world in which formulae are violated [Richardson and Domingos, 2006].

[8] If ϕ contains no free variables we replace $\models_{\mathbf{y}} \phi[B]$ with $\models_{\mathbf{y}} \phi$ in the definition of f_B^{ϕ}.

[9] This is the notation used in the introduction of Markov Logic [Richardson and Domingos, 2006].

[10] If ϕ contains no free variables we can define $\mathcal{B}(\phi)$ to contain a single arbitrary binding.

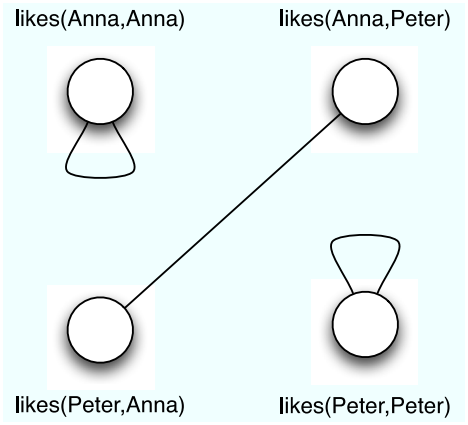


Figure 3.3: Example Ground Markov Network.

Before we present a more involved example of a Markov Logic Network in section 3.5 let us first consider a simple one to illustrate the relation between Markov Logic Networks and the Markov Networks they describe. Assume a predicate *likes/2,* two constants Anna and Peter and a formula ϕ

$$likes(x,y) \Rightarrow likes(y,x) \tag{3.11}$$

that captures the reflexivity of *likes*. Then the Markov Logic Network $L = \{(\phi, w)\}$ would yield the Ground Markov Network shown in figure 3.3 that contains the variables $Y_{likes}^{\text{Anna,Anna}}$, $Y_{likes}^{\text{Anna,Peter}}$,$Y_{likes}^{\text{Peter,Anna}}$ and $Y_{likes}^{\text{Peter,Peter}}$ and features such as

$$f^{\phi}_{\{x/\text{Anna,y/Peter}\}}(\mathbf{y}) = \begin{cases} 1 & \text{if } \vDash_{\mathbf{y}} likes(\text{Anna},\text{Peter}) \Rightarrow likes(\text{Peter},\text{Anna}) \\ 0 & \text{otherwise} \end{cases} \tag{3.12}$$

Note that sometimes a Markov Logic Network contains formulae that can be generalized to a single formula with additional free variables. However, because some of these formulae have different weights we cannot replace them by the generalized formula without changing the semantics of the Markov Logic Network. If we still want to describe such sets of formulae compactly we will use the generalized formula and mark all variables that uniquely define the weight of each instantiation using the '+' symbol [Kok et al., 2005]. For example, the formula

$$pos(x,+p) \Rightarrow arg(x,+a) \tag{3.13}$$

corresponds to a set of formulae of type

$$pos(x,P) \Rightarrow arg(x,A) \tag{3.14}$$

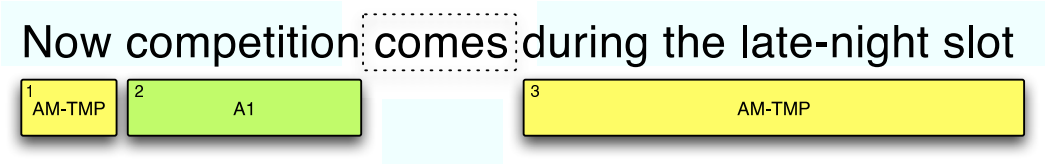


Figure 3.4: Correct role labelling for example sentence.

for each possible POS tag P and semantic label A, each with an individual weight $w_{P,A}$. For simplicity we will sometimes use an expression such as equation 3.13 and still refer to it as "one formula", even though it refers to a set of formulae.

Finally, when a formula ϕ has the weight w in a Markov Logic Network we will sometimes write

$$\phi \, \langle w \rangle$$

when we present ϕ.

3.5 Semantic Role Labelling Example

Let us come back to the example given in section 2.2. To remind the reader, we were trying to find a semantic role labelling for the sentence

> Now competition comes during the late-night slot.

Ideally, the role labelling we find looks like the one in figure 3.4. To this end we made a few assumptions based on our linguistic intuition. For example, we assumed that a verb can have at most one argument labeled A1. We went on to capture these assumptions in a Markov Network tailored to the given example sentence. In this section we will show how we can use Markov Logic to compactly describe this Markov Network, and all Markov Networks that capture the same assumptions.

First we define the predicates that represent the hidden role label information. We have said that there are two decisions to make: (a) whether a constituent is an argument and, if so, (b) what kind of label it should get. Hence we represent the hidden information using two predicates, *hasRole/1* and *role/2*. If *hasRole(x)* holds then the constituent x is an argument. If *role(x,r)* holds then the constituent x has the role r.

The observed information is captured using *pos/2* and *overlap/2*. The atom *pos(x,p)* holds if the head word of the constituent x has the Part of Speech tag p; the atom *overlap(x,y)* holds if the span of constituent x overlaps with the span of constituent y.

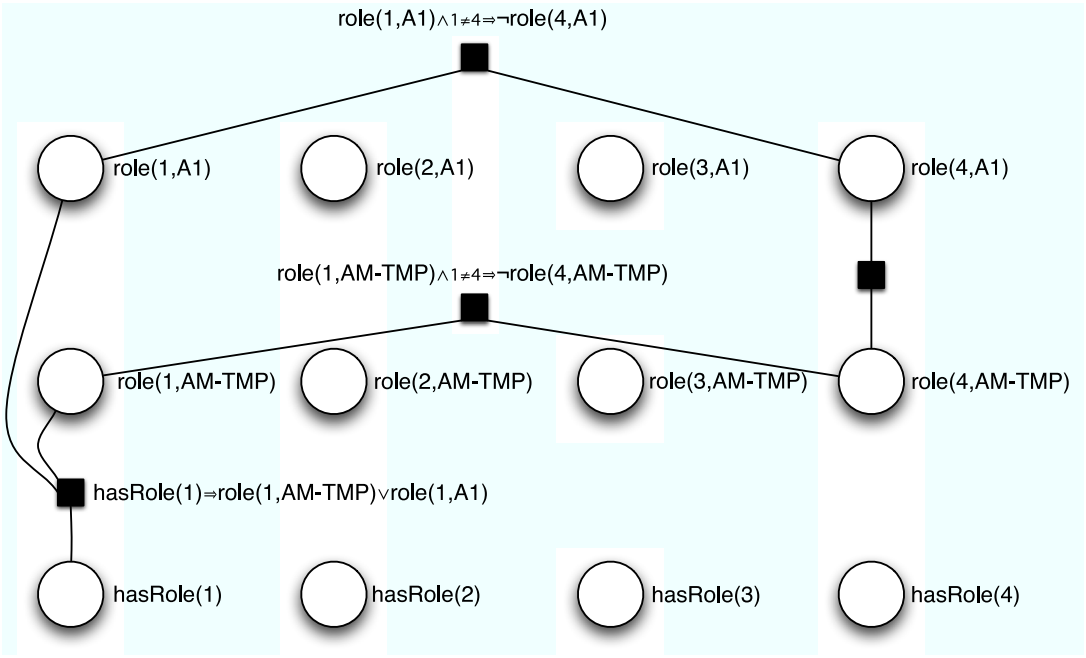


Figure 3.5: Partial Ground Markov Network for example Role Labelling problem. Note that this network only shows hidden ground atoms and a subset of ground formulae.

The following formula can be used to enforce the uniqueness assumption we made for the A1 role

$$role(i, \text{A1}) \wedge i \neq j \Rightarrow \neg role(j, \text{A1}) \tag{3.15}$$

And, for example, in order to forbid overlapping arguments the formula

$$overlap(i, j) \wedge hasRole(i) \Rightarrow \neg hasRole(j) \tag{3.16}$$

can be added to the MLN.

The remaining formulae can be captured in a similar manner. Note how close this formal representation of our assumptions is to the informal description we gave in section 2.2. However, by having a defined syntax and semantics for this representation, a computer can read the formulae and interpret them. For example, we can use this representation to learn weights or to infer the most likely world with respect to an observation. Moreover, it is possible to search for the best set of possible features by searching through the set of possible First Order Logic formulae [Kok and Domingos, 2005, Richardson and Domingos, 2005].

For the given MLN L and the set of constants $C = \{1, 2, 3, 4\}$ we show a part of the Ground Markov Network $M_{L,C}$ as a factor graph in figure 3.5. To ensure readability it contains only hidden ground atoms (for the predicates *hasRole/1* and *role/2*) and a

subset of ground formulae. Note that the label for each factor is the ground formula associated with its feature.

3.6 MAP Inference in Markov Logic

MAP inference in Markov Logic can be framed as follows: assume that the truth state $\mathbf{x}$ of a set of ground atoms is given, then find the truth state $\hat{\mathbf{y}}$ of the set of remaining ground atoms for which the probability $Pr(\hat{\mathbf{y}}|\mathbf{x})$ is maximal:

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} Pr(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} s(\mathbf{y},\mathbf{x}) \tag{3.17}$$

where

$$s(\mathbf{y},\mathbf{x}) = \sum_{(\phi,w)\in L} \sum_{B\in\mathcal{B}(\phi)} w \cdot f_B^{\phi}(\mathbf{y},\mathbf{x}) \tag{3.18}$$

and $\mathcal{B}(\phi)$ contains all bindings of the free variables in ϕ with the constants in V.

Instead of explicitly describing the state of each ground atom in $\mathbf{x}$ and $\mathbf{y}$, we will represent $\mathbf{x}$ and $\mathbf{y}$ as sets of ground atoms (i.e., possible worlds). This requires us define which ground atoms that are *not* in $\mathbf{x}$ are false, and for which the state is hidden. To this end we divide predicates into two types. Every ground atom of an *observed predicate* is true if and only if the ground atom is in $\mathbf{x}$. Likewise, every ground atom of an *hidden predicate* is true if and only if it is in $\mathbf{y}$.

For example, assume that we learnt a MLN for Semantic Role Labelling as described in section 3.5. In this case predicates such as *pos* and *overlap* are observed, and the *hasRole* and *role* predicates are hidden. Hence, if our observation is

$$\mathbf{x} = \{pos(1,\mathrm{RB}), pos(2,\mathrm{NN}), pos(3,\mathrm{IN}), pos(4,\mathrm{NN}), overlap(3,4)\} \tag{3.19}$$

then this means that *pos*(1,RB) is true but *pos*(1,NN) is false, while the state of *hasRole*(1) is unknown.

Although it is technically possible to individually define each ground atom as either hidden or observed, often this is not necessary[11] and makes our descriptions (and implementations) more complex. We will therefore use the notion of hidden and observed predicates throughout this thesis.

[11] If one wishes to define observed ground atoms of predicate *pred* individually one can introduce two new observed predicates: *predTrue* and *predFalse*. The ground atoms of *predTrue* (*predFalse*) represent the ground atoms of *pred* which we observed to be true (false). And by adding the deterministic rules

$$predTrue(x) \Rightarrow pred(x), predFalse(x) \Rightarrow \neg pred(x)$$

one can ensure that the final solution will still be consistent with the observation.

Note that for MAP inference a Ground Markov Network $M_{L,V}$ can be simplified: instead of all ground atoms, the network only needs to contain the *hidden* ground atoms. Moreover, we can remove all ground formulae (features) that will either always be false or always be true. For a given observation $\mathbf{x}$ we will refer to this simplified network as the *Conditional Ground Markov Network* $M_{L,V,\mathbf{x}}$. This simplification is valid because the log-linear score for any solution $\mathbf{y}$ with respect to $M_{L,V,\mathbf{x}}$ only differs by a constant when compared to the loglinear score of $(\mathbf{y},\mathbf{x})$ with respect to $M_{L,V}$.

The notion of MAP inference and the idea of hidden and observed predicates can also be found in applications of traditional logic. For example, in database languages such as DATALOG we find base tables (observed predicates) and views (hidden predicates). MAP inference here means to find the single model that is consistent with the view definitions and the content of the base tables [Hinrichs and Genesereth, 2006]. However, while in Markov Logic multiple worlds can have nonzero probability, for a database there is exactly one such world.

3.6.1 Propositional Inference

Each Conditional Ground Markov Network is finite because the vocabulary and hence the set of possible substitutions is finite. This means that one can solve the MAP problem for Markov Logic by instantiating the Conditional Ground Markov Network and then use a method from section 2.3 to find the optimal assignment of variables and hence the optimal state of ground atoms.

However, in many cases the Conditional Ground Markov Network can be very large. Moreover, an MLN would often define densely connected Markov Networks that contain many deterministic or near deterministic constraints (such as some of the formulae we presented in section 3.5). This leads to two problems:

1. Instantiating the Ground Markov Network requires a lot of memory and can soon become prohibitive both in terms of time and space requirements.

2. The densely connected and partly deterministic network causes solvers to be inaccurate and slow.

For example, consider the formula

$$role\,(i,\mathrm{A1}) \wedge i \neq j \Rightarrow \neg role\,(i,\mathrm{A1}) \tag{3.20}$$

and assume that we have 40 instead of 4 constituent candidates. This would lead to $O\left(40^2\right)$ factors that pairwise connect the *role(i,*A1*)* nodes, each one with a very high

weight. Solvers such as Belief Propagation are likely to fail in this setting [Sontag and Jaakkola, 2007].

Or consider a setting where we want to find pairs of citations referring to the same publication (as we do in chapter 5) and assume there are 400 in total. If we want to capture the transitivity property that holds for the "same publication" relation *same*/2, we need to add the formula

$$same(x,y) \wedge same(y,z) \Rightarrow same(x,z) \tag{3.21}$$

This formula leads to $O\left(400^3\right)$ deterministic features in the Ground Markov Network. Here not only is the network difficult to solve, its instantiation becomes difficult, too.

3.6.2 LazySAT

LazySAT [Singla and Domingos, 2006b] is an implementation of MaxWalkSAT with reduced memory requirements. Instead of grounding all formulae and predicates from the outset, LazySAT instantiates them lazily, only when they are needed.

Algorithm 3.1 shows LazySAT in pseudo code. LazySAT keeps only a small set of *active (ground) atoms A* and *active (ground) clauses C* in memory. The initial set of active atoms A is the set of atoms contained in the set of clauses which are not satisfied in a user specified (or random) initial solution $\mathbf{y}_0$. The initial set of active clauses C are all clauses which can be made unsatisfied by flipping zero or more active atoms.

After initialization we choose a random clause c from C. Then we pick an atom a of this clause. This atom is either randomly picked, or it is greedily chosen to maximize the total score when its truth state is flipped. Now LazySAT has to consider two cases:

1. The atom a' is active. In this case we can use the clauses in C and the states of atoms in A to calculate the improvement in score given the current solution $\mathbf{y}$ as in MaxWalkSAT using $deltaScore(C, A, a', \mathbf{y})$.

2. The atom a' is not active. In this case we cannot use the active atoms and clauses. However, since we have a first order representation of all possible clauses (in the Markov Logic Network) it is possible to calculate the improvement without grounding the whole network using $liftedDeltaScore(L, a', \mathbf{y})$.

Once we have picked an atom, either greedily or randomly, we flip the truth state of this atom in the current solution. Finally we update the set of active atoms and clauses: if the atom a' was not already active it is added to A and the clauses that could be made

active through a' are added to C. If a maximum number of flips have been performed the algorithm terminates (or is restarted again). Otherwise we go on to pick the next clause out of C and proceed as above.

LazySAT shows how a first order representation can help to make inference more efficient. Step 15 in algorithm 3.1 would not be possible if there was no first order representation of the Ground Markov Network. LazySAT improves MaxWalkSAT in two dimensions: it reduces memory requirements and the runtime of the initial network construction. The latter aspect is important because the initialization of the network can often dominate total runtime—we will observe this in chapter5. Note, however, that LazySAT has the same runtime requirements as MaxWalkSAT once the initial network construction phase is completed. Each flip in LazySAT is essentially equivalent to a flip in MaxWalkSAT in the same iteration. In fact, LazySAT needs slightly more time due to the higher cost of *liftedDeltaScore* in comparison to *deltaScore*.

Note that LazySAT is in fact a special case of the recently introduced Lazy Inference approach [Poon et al., 2008] that allows us to construct lazy versions of a wide range of algorithms. The general idea is avoid allocation for variables (such as ground atoms) and functions (such as ground clauses) as long as they are set to a default value (e.g., false for ground atoms, true for ground clauses).

3.7 Learning

Learning in Markov Logic can mean two things: we can either learn the weights of the formulae in a Markov Logic Network, or we can learn the formulae themselves. In fact, we can do both at the same time [Kok and Domingos, 2005, Huynh and Mooney, 2008].

To learn the weights of a Markov Logic Network we can apply any propositional method as presented in chapter 2. In this work we will restrict ourselves to this approach. However, note that it is also possible to learn weights more efficiently by exploiting the available first order information. For example, Koller [1999] shows how this can be achieved in the case of Probabilistic Relational Models.[12]

To learn the formulae of a Markov Logic Network we can use Inductive Logic Programming, a field that focuses on inducing logical rules from data. Richardson and Domingos [2005] train Markov Logic Networks by first learning their rules us-

[12]Note that their SRL formalism is based on Bayesian Networks so their results are not directly transferable to Markov Logic.

Algorithm 3.1 LazySAT algorithm

Require: A Markov Logic Network L, an initial solution $\mathbf{y}_0$,
the number of restarts $n_{restarts}$, the number of flips n_{flips}
and an observation $\mathbf{x}$

$r \leftarrow 0$
while $r \leq n_{restarts}$ **do**
  $A \leftarrow inUnsatisfiedClause\,(L, \mathbf{y}_0, \mathbf{x})$
  $C \leftarrow activatedBy\,(A)$
  $\mathbf{y}_A \leftarrow random$
  $i \leftarrow 0$
  while $i \leq n_{flips}$ **do**
    $c \leftarrow randomClause\,(C)$
    $u \leftarrow random\,(0,1)$
    if $u < p$ **then**
      for $a' \in c$ **do**
        if $a' \in A$ **then**
          $s_{a'} \leftarrow deltaScore\,(C, A, a', \mathbf{y}, \mathbf{x})$
        else
          $s_{a'} \leftarrow liftedDeltaScore\,(L, a', \mathbf{y}, \mathbf{x})$
        end if
      end for
      $a \leftarrow \min_{a' \in c} s_{a'}$
    end if
    if $Y_f \notin A$ **then**
      $A \leftarrow A \cup \{Y_a\}$
      $C \leftarrow C \cup activatedBy\,(A)$
    end if
    $Y_a \leftarrow 1 - Y_a$
    $i \leftarrow i + 1$
  end while
  $r \leftarrow r + 1$
end while
return y with highest probability

ing Inductive Logic Programming and then learning their weight parameters using Pseudo-Likelihood training. Kok and Domingos [2005] show that improvements can be achieved by integrating both steps.

Several other training methods exist or can be envisioned; however, this thesis focuses on MAP inference and we refer the reader to other work for more details [Richardson and Domingos, 2006, Mihalkova and Mooney, 2007].

3.8 Statistical Relational Learning

Markov Logic is not the first and only approach to combine logic with probability. Several others have been investigated in a field known as *Statistical Relational Learning* (SRL). SRL languages come in different flavours: they can be based on different probabilistic frameworks (e.g., Bayesian Networks or Markov Networks), based on different subsets of First Order Logic (e.g., Prolog or SQL) and make different assumptions about the number and identity of objects in the domain. It is beyond the scope of this thesis to give an exhaustive introduction into this SRL[13] formalisms. Instead we refer the reader the book of Getoor and Taskar [2007] and an overview paper by Milch and Russell [2007]. However, it should be pointed out that Markov Logic has been shown to generalise several other SRL frameworks, such as Knowledge Based Model Construction, Relational Probabilistic Models and Relational Markov Networks [Richardson and Domingos, 2006]. The MAP inference method presented in this thesis can therefore be directly applied within these frameworks, too.

3.9 Conclusion

In this chapter we have introduced Markov Logic, a Statistical Relational Learning language that uses weighted First Order Logic formulae to describe Markov Networks. We motivated Markov Logic from two perspectives: as a way of introducing abstraction to Markov Networks, and as a way of introducing uncertainty to First Order Logic.

We have argued that Markov Logic has several advantages over Markov Networks. First, it allows us to develop methods such as LazySAT that can perform inference in large Markov Networks without completely instantiating these potentially large networks. Second, with Markov Logic it is possible to induce new structural patterns

[13] Note that we use the abbreviation SRL for both Statistical Relational Learning and Semantic Role Labelling when the intended meaning is clear from the context.

(as opposed to just edges) from data. Third, by using Markov Logic we do not need to manually construct Markov Networks for new problem instances. Fourth, Markov Logic is better suited to decouple Machine Learning technology and AI applications, and hence to foster independent progress of both. Finally, Markov Logic Networks are simpler and more compact than their propositional counterparts—this helps us to explain, inspect and extend them more easily.

We also presented LazySAT, a variant of MaxWalkSAT that is tailored to Markov Logic and significantly reduces the memory overhead of inference by only instantiating only fractions of the complete Markov Network. In the next chapter we will introduce one of the core contributions of this thesis: a MAP inference method that not only improves memory efficiency but also speed and accuracy.

Chapter 4

Cutting Plane Inference for Markov Logic

As presented in section 3.6, MAP inference in Markov Logic amounts to finding a set of hidden ground atoms $\hat{\mathbf{y}}$ with maximum *a posteriori* probability given a set of observed ground atoms $\mathbf{x}$ and a Markov Logic Network L. More formally, we search for

$$\hat{\mathbf{y}} = \arg\max_{\mathbf{y}} Pr(\mathbf{y}|\mathbf{x}) = \arg\max_{\mathbf{y}} s(\mathbf{y},\mathbf{x}) \tag{4.1}$$

where

$$s(\mathbf{y},\mathbf{x}) = \sum_{(\phi,w)\in L} \sum_{B\in\mathcal{B}(\phi)} w \cdot f_B^{\phi}(\mathbf{y},\mathbf{x}) \tag{4.2}$$

evaluates the "consistency" of the joint possible world $(\mathbf{x},\mathbf{y})$ with respect to the formulae in L.

In theory every MAP problem in Markov Logic can be solved by completely grounding the Markov Logic Network and solving the MAP problem in the corresponding Ground Markov Network using any propositional MAP method of choice (see section 2). Richardson and Domingos [2005] proposed the use of MaxWalkSAT to solve the ground networks. In chapter 5 we apply MWS to two tasks, Semantic Role Labelling and Joint Entity Resolution. In both cases we found this approach to be slow, mostly due to the cost of grounding the complete network. In the case of Joint Entity Resolution we also observe MaxWalkSAT to be very inaccurate (both in terms of model score and F1 accuracy).

To overcome these problems one could try another propositional MAP algorithm from the literature and apply it to the full Ground Markov Network. However, there are two reasons not to follow this approach:

1. Some of the networks we are looking at in the context of Markov Logic are still significantly larger and more complex than the networks considered in current state-of-the-art work on propositional MAP inference, such as the networks considered by Sontag and Jaakkola [2007]. For example, in the case of Joint Entity Resolution we have to do MAP inference for networks with hundred thousands of nodes and millions of factors. In such settings the instantiation of networks becomes very difficult, let alone the search for the MAP solution.

2. The network we consider are partly deterministic. In other words, many formulae have a high weight and hence impose deterministic or near-deterministic constraints on the solution space. Many solvers, such as Belief Propagation and its variants, can perform poorly in such settings. On the other hand, solvers which can cope with deterministic constraints, such as ILP solvers, tend to be slow. For example, in the case of Semantic Role Labelling the ILP solver can take minutes to solve a single problem—this is too much when we have to process large corpora (or need MAP solutions in a Online Learning setting).

Hence in this work we follow a different approach: instead of improving a propositional MAP method we try to reduce the size and complexity of the propositional network and use an existing propositional method as is. This leads to the introduction of an incremental *meta* algorithm: *Cutting Plane Inference for Markov Logic* (CPI), inspired by the Cutting Plane method [Dantzig et al., 1954], which solves the complete Ground Network by incrementally solving a sequence of sub-networks using any given propositional MAP method.

CPI starts with a very simple subproblem of the complete Ground Network. For example, this problem could contain all ground features corresponding to the local formulae of the Markov Logic Network. This sub-problem is solved using any propositional method of choice. In turn CPI uses a first order query engine to find all ground formulae for which the corresponding feature-weight product ($w \cdot f_B^{\phi}(\mathbf{y}, \mathbf{x})$) is suboptimal with respect to the current solution. These are then added to the partial network of the previous step, provided they are not included already. The new network is solved, the corresponding solution is inspected and suboptimal ground formulae are added once again. This process is repeated until no more new ground formulae can be found or a maximum number of iterations is reached. Each CPI iteration produces one solution. Of all these solutions CPI returns the one with highest score.

Often the partial problems to be solved in the CPI loop are significantly smaller and

less complex than the complete Ground Markov Network. Thus we can expect solvers to perform better on those problems than on the complete problem. Moreover, often the current network changes only minimally from one step to the next. ILP solvers can resolve such incrementally changed problems very efficiently.

We will later show that the accuracy of CPI corresponds to the accuracy of the base solver on the final sub-problem. This also means that in the case of an exact base solver, such as an ILP solver, CPI itself will be exact. We will see in chapter 5 that this can lead to cases where exact MAP inference is feasible even if the complete Ground Network is densely connected and contains millions of factors.

In the remainder of this chapter we will first introduce the general idea of Cutting Planes. We will then present Cutting Plane Inference for Markov Logic and prove that its accuracy scales linearly with the accuracy of the base solver. We will also present a means to efficiently search for ground formulae for which the corresponding feature-weight product is suboptimal in a given solution. After presenting a variant of CPI that can process formulae in a different order we finally compare CPI to related work.

Note that the algorithm in section 4.2, parts of the discussion in section 4.5 and the analysis in section 4.3 also appear in [Riedel, 2008].

4.1 Background

Consider the class of 0-1 Linear Programs. As we have pointed out in chapter 2, they are particularly useful for solving MAP problems. A 0-1 Linear Program describes a constrained optimization problem of the following form.

$$\begin{aligned} &\arg\max_{\mathbf{x}\in\{0,1\}^n} \ \mathbf{c}^T\mathbf{x} \\ &\forall i \in \{1,\ldots,m\} : \mathbf{a}_i^T\mathbf{x} \leq b_i \end{aligned} \tag{4.3}$$

where $\mathbf{a}_i \in \mathbb{R}^n$,$b_i \in \mathbb{R}$ and $\mathbf{c} \in \mathbb{R}^n$.

Many problems of the form 4.3 cannot be solved directly because they contain a large number of constraints. For example, the Travelling Salesman Problem (TSP) can be formulated as a 0-1 Linear Program. However, the number of constraints needed is exponential with respect to the number of cities because one constraint for each possible cycle of cities is needed to ensure that the graph does not contain subtours.

However, despite the fact that the TSP is NP-hard, many real world instances of the TSP can be efficiently and exactly solved using a 0-1 Linear Programming formulation and the following algorithm [Dantzig et al., 1954]:

1. Solve;
2. Find violated constraints (i.e. subtours[1]);
3. Add constraints;
4. Go to (1) unless no more constraints were found.

This approach is generally referred to as a *Cutting Plane* algorithm. We will see in a minute why this is so. Note that many solvers have warm-starting capabilities: they can use the solution of the last problem to speed up solving the current, slightly modified problem.[2] This is extremely important for the efficiency of the Cutting Plane Method.

To illustrate the general idea behind Cutting Plane algorithms let us look at an example Linear Program, graphically shown in figure 4.1. The set of feasible solutions (those that satisfy all constraints) to this problem can be described as a polytope in $\mathbb{R}^2$. This polytope is indicated by the lines in figure 4.1, where the arrow attached to each line indicates in which half space the corresponding inequality holds. Let the point y' be the optimum of the corresponding Linear Program.

A Cutting Plane algorithm starts by solving the optimization problem with only a subset of the original inequalities. For example, we could start by only considering the two constraints that are drawn with solid lines in figure 4.2. Solving this optimization problem results in a solution y_1. This solution is inspected by an *oracle*: a method that finds a set of violated constraints or indicates that no violated constraints exist.

We find that there is a single violated constraint and add it to the problem. In other words, we *cut* the current feasible region by a hyperplane corresponding to the violated constraint. Our new LP can be seen in Figure 4.3. Solving this problem leads us to the true optimum $\mathbf{y}'$, even though we still have not included all constraints. The reason is the objective function: it points us to $\mathbf{y}'$—the further we move $\mathbf{y}$ to the left, the higher the objective function will be. Hence at this point the constraints to the right of the problem are redundant with respect to the problem.

In the previous example we saw how the objective function helps us avoid considering all constraints of the original problem. Obviously, the better the objective function is in guiding us into a feasible region, the more efficient the above algorithm is. And while in general there is no necessary connection between objective function

[1] Note that Dantzig et al. [1954] enforce solutions to be integer by using additional linear constraints that are known to hold for solutions of the TSP. These constraints are also added only when needed.

[2] For example, the Dual Simplex Algorithm that is used in many LP and ILP solvers can warm-start easily when a problem is extended by a set of constraints.

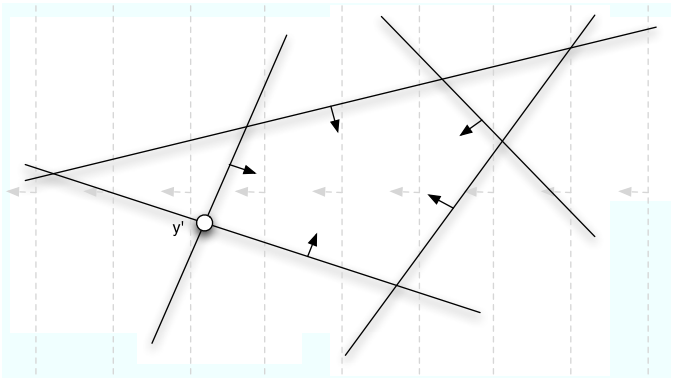


Figure 4.1: A Linear Program with two variables and 5 linear inequalities that define a polytope. The value of the objective function is constant on the dashed lines and increases in the direction of the arrows connected to these lines. y' is the optimal solution of the Linear Program.

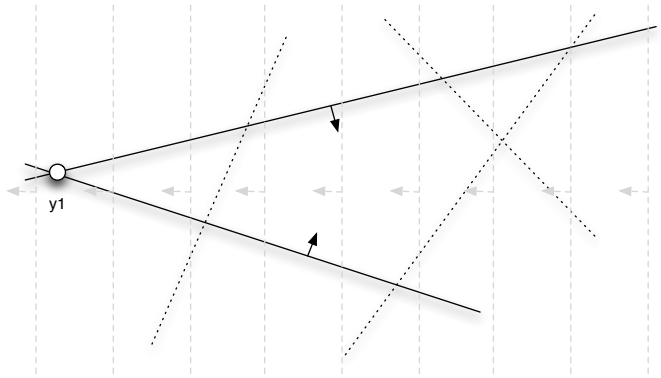


Figure 4.2: The feasible region of a Linear Program where three constraints have been removed from the original problem.

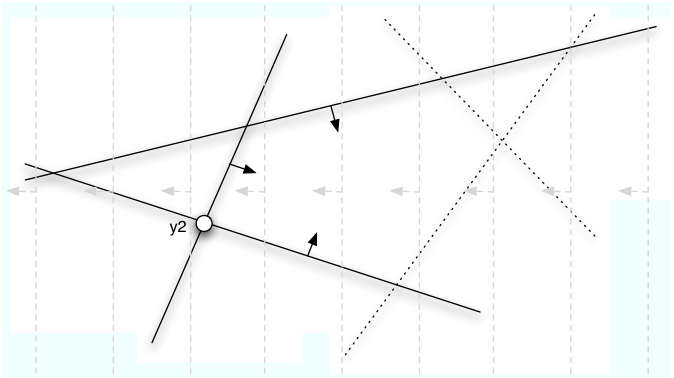


Figure 4.3: The feasible region after a violated constraint/cut has been added.

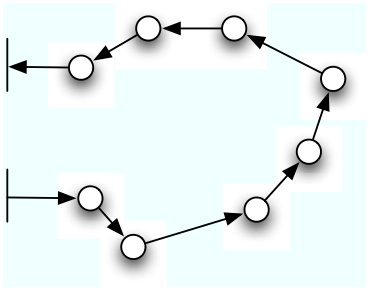


Figure 4.4: An “easy” Travelling Salesman Problem where the greedy solution is already cycle-free.

and constraints, in practice we often encounter cases where the objective function is a good guide.

s

For example, consider again the TSP. Imagine we want to find the shortest tour between the cities of an island. Here the objective function is the sum of distances between subsequent cities of the tour. As mentioned before, the TSP can be formulated as an Integer Linear Program which contains constraints that guarantee acyclicity. There is one constraint for each possible cycle, yielding an exponential number of constraints with respect to the number of cities.

However, most of these cities will be lined along the coast of this island. If we just solve the ILP without cycle constraints and try to minimize the sum of distances, then we will see that a good solution (in terms of the objective function) can be produced by simply connecting the cities along the coast (figure 4.4). This solution does not have any cycles to begin with. It can therefore be solved in one iteration using a cutting plane approach.[3]

In fact, the cutting plane method has been used to solve the TSP for all 24,978 cities in Sweden in an exact manner [Applegate et al., 2004], again despite the fact that the TSP is NP-hard. This suggests that finding the optimal solution to large optimization problems with complex structure, such as the MAP task for Markov Networks, does not need to be impractical per se.

Cutting Plane algorithms can also be seen as a counterpart to optimization algorithms that exploit a certain structure of the problem but fail to work when this structure is not present (such as Viterbi algorithms in linear chain Markov networks). Instead of requiring the structure of a problem to have certain properties, Cutting Plane al-

[3] A similar situation appears when solving the TSP for 50 cities in the USA, one in each state Dantzig et al. [1954]. Here the partial solution for the cities of all east coast states is trivial: we just have to travel from Florida up to Maine and visit each state along the way.

gorithms require the weights (or costs) to have certain properties—as this is a rather vague requirement we will try to make it more concrete in chapter 6.

4.2 Cutting Plane Inference

There is more than one way to use a Cutting Plane algorithm for MAP inference in Markov Logic. For example, one could use one of the ILP formulations we presented in section 2.3.2 and perform a Cutting Plane algorithm in terms of this representation, adding linear constraints of the ILP whenever necessary. However, this restricts the type of propositional solver we can use, because a partial ILP formulation with a subset of linear constraints makes sense to an ILP solver but not to a MaxWalkSAT or Belief Propagation method. And as there may be cases where using an ILP solver is impractical, even on a problem with reduced size, it can be helpful to plug-in a different propositional method instead. Moreover, by restricting ourselves to ILP we will not be able to exploit any further advances in propositional MAP inference made in the future, unless they are based on ILP.

If we want to be able to use arbitrary Markov Network (or weighted MAX-SAT) inference methods in our Cutting Plane algorithm we cannot solve a sequence of partial ILPs. Instead we need to solve a sequence of partial Markov Networks. To this end we present *Cutting Plane Inference* (CPI), a Cutting Plane algorithm that incrementally adds factors (edges) to a partial Ground Markov Networks and uses a propositional method of choice to solve the MAP problem in this partial networks.

Instead of searching for violated constraints, CPI searches for feature-weight products in

$$s(\mathbf{y},\mathbf{x}) = \sum_{(\phi,w)\in L} \sum_{B\in C^{n_\phi}} w \cdot f_B^\phi(\mathbf{y},\mathbf{x}) \tag{4.4}$$

that do not maximally contribute to the overall sum for the current solution. In other words, given a solution $\mathbf{y}$ and observation $\mathbf{x}$ we search for every variable assignment B for every formula ϕ for which the term $w \cdot f_B^\phi(\mathbf{y},\mathbf{x})$ is not optimal, i.e. there exists a $\mathbf{y}'$ for which $w \cdot f_B^\phi(\mathbf{y}',\mathbf{x})$ is larger than $w \cdot f_B^\phi(\mathbf{y},\mathbf{x})$. Formally we define:

Definition 4.1. For every set of hidden ground atoms $\mathbf{y}$, every observed set of ground atoms $\mathbf{x}$, formula ϕ and weight w the set Separate$(V,\phi,w,\mathbf{y},\mathbf{x})$ is the set of all of variable assignments B with constants in V so that

$$w \cdot f_B^\phi(\mathbf{y},\mathbf{x}) < \max_{\mathbf{y}'} w \cdot f_B^\phi(\mathbf{y}',\mathbf{x}) \tag{4.5}$$

If $B \in \text{Separate}(V, \phi, w, \mathbf{y}, \mathbf{x})$ we will say that the corresponding feature-weight product $w \cdot f_B^\phi(\mathbf{y}, \mathbf{x})$ is *locally suboptimal.*

In the terminology of the Cutting Plane method this step is often referred to as *separation*: it finds a set of constraints that separates feasible solutions from infeasible solutions. In our case this step will help to separate possible worlds with high score from those with low score.

A complete ground Markov Network contains a feature for each variable assignment of each formula in the Markov Logic Networks. The partial Markov Networks generated during CPI contain features for subsets of the variable assignments of each formula. To compactly represent these networks we will introduce the notion of a partial grounding.

Definition 4.2. A *partial grounding* $\mathbf{G} = \left(G_\phi\right)_{(\phi,w)\in L}$ maps each first order formula ϕ of an MLN L to a set of variable assignments G_ϕ. A partial grounding $\mathbf{G}$ induces a *partial score*

$$s_{\mathbf{G}}(\mathbf{y}, \mathbf{x}) = \sum_{(\phi,w)\in L} w \sum_{B\in G_\phi} f_B^\phi(\mathbf{y}, \mathbf{x}) \tag{4.6}$$

We are now ready to introduce Cutting Plane Inference (CPI). Algorithm 4.1 shows CPI in pseudo-code. It takes as input a Markov Logic Network L, a Vocabulary V, an observation $\mathbf{x}$, an initial partial grounding $\mathbf{G}^0$ and a propositional *base solver BS*. In each iteration i we consider a partial grounding $\mathbf{G}^i$ (for $i = 0$ this is the partial grounding $\mathbf{G}^0$ provided as input to the algorithm). In step 5 we find a solution $\mathbf{y} = \text{solve}\left(\mathbf{G}^{i-1}, \mathbf{x}\right)$ that maximises the partial score $s_{\mathbf{G}^{i-1}}$ (or approximately maximizes it) for the current partial grounding, using the base solver *BS*. Steps 6 and 7 compare the score of the current solution $\mathbf{y}$ with the score of the previous best solution $\mathbf{y}'$. If the score of the current solution is higher than the previous best score the current solution becomes the new best solution $\mathbf{y}'$.[4]

The loop in steps 9 and 10 finds the ground formulae which are locally suboptimal in the current solution $\mathbf{y}$ and adds them to the current partial grounding. Note that we can skip formulae which are already completely grounded. For example, if we ground all local formulae to create $\mathbf{G}^0$ there will be no need to search for new groundings of local formulae later.

[4]This is necessary in case the base solver is approximate. Assume the base solver returns completely random solutions and in iteration 1 its result is the optimal solution. In the second iteration the result is another solution, but with lower score, and in the third iteration the second solution is returned again. In this case the algorithm terminates but the last solution is not the best solution generated during CPI. However, in the case of exact inference the last solution is in fact the optimal solution; this will become clear in section 4.3.

We terminate the algorithm in step 12 if no more new ground formulae are found or a maximum number of iterations is reached. Alternatively we can terminate CPI when the sum

$$\sum_{(\phi,w)\in L} \sum_{B\in G_\phi^i \setminus G_\phi^{i-1}} |w| \tag{4.7}$$

of absolute weights for the new ground formulae is under some threshold—a choice that will be justified in section 4.3. The final result is the current best solution $\mathbf{y}'$.

Note that a natural choice for $\mathbf{G}^0$ are all groundings of formulae that only contain one hidden predicate when grounded. In this case maximizing $s_{\mathbf{G}^0}$ is trivial because the hidden variables do not interact. Moreover, using these local formulae often gives a very good first guess. For example, it is well known that in many NLP applications, such as PoS tagging, Named Entity Recognition and Semantic Role Labelling, features based on observable properties such as the orthography of words are very powerful.

Algorithm 4.1 Cutting Plane Inference

Require: A Markov Logic Network L, a vocabulary V, a base solver BS,
an observation $\mathbf{x}$, an initial grounding $\mathbf{G}^0$
and the maximum number of iterations $maxIterations$

1: $i \leftarrow 0$
2: $\mathbf{y}' \leftarrow \mathbf{0}$
3: **repeat**
4: $\quad i \leftarrow i+1$
5: $\quad \mathbf{y} \leftarrow$ solve $\left(\mathbf{G}^{i-1}, \mathbf{x}\right)$ using base solver BS.
6: $\quad$ **if** $s(\mathbf{y},\mathbf{x}) > s(\mathbf{y}',\mathbf{x})$ **then**
7: $\quad\quad \mathbf{y}' \leftarrow \mathbf{y}$
8: $\quad$ **end if**
9: $\quad$ **for** each $(\phi, w) \in L$ **do**
10: $\quad\quad G_\phi^i \leftarrow G_\phi^{i-1} \cup \text{Separate}\,(V, \phi, w, \mathbf{y}, \mathbf{x})$
11: $\quad$ **end for**
12: **until** $\mathbf{G}^i = \mathbf{G}^{i-1}$ or $i > maxIterations$
13: **return** $\mathbf{y}'$

4.3 Correctness

Most Cutting Plane algorithms for MAP inference [Riedel and Clarke, 2006, Anguelov et al., 2004, Sontag and Jaakkola, 2007] iteratively add *hard* constraints to an optimization problem. It is easy to see that once a solution does not violate any more hard constraints it must be optimal. However, Markov Logic Networks may also contain soft formulae that can be violated, albeit with a penalty. CPI can also be applied to such networks but it is not obvious how accurate its solutions will be. In this section we will show that CPI essentially inherits the accuracy of the base solver on the final partial problem (as opposed to the accuracy we would observe when applying the base solver to complete ground network). This also implies that CPI is exact if the base solver is. Moreover, we show how the accuracy of CPI depends on the number and weight of new locally suboptimal ground formulae if we terminate CPI before the partial problem converges (i.e., before $\mathbf{G}^i = \mathbf{G}^{i-1}$).

The following theorem shows that when CPI returns the solution $\mathbf{y}'$ of iteration i the error is bound by the sum of the error of the base solver on the last partial problem $\mathbf{G}^{i-1}$ and the term $\sum_{(\phi,w)\in L}\sum_{B\in G^i_\phi \setminus G^{i-1}_\phi}|w|$. In the summation term we consider each formula ϕ in the Markov Logic Network and add the absolute weight $|w|$ of ϕ for every grounding B which was found in the last separation step but has not been in the previous set of groundings G^{i-1}_ϕ for ϕ (i.e., B is in $G^i_\phi \setminus G^{i-1}_\phi$). Thus, if we have only a few newly found ground formulae, if these formulae have a low weight and if the base solver has determined a $\mathbf{y}'$ that has a high score with respect to the partial problem $\mathbf{G}^{i-1}$, then $\mathbf{y}'$ is also a high scoring solution for the complete Markov Network.

In particular, for an iteration with no more newly found groundings, the error is only bound by (in fact is equal to) the error of the base solver on the partial problem, which is likely to be much smaller and easier to solve than the original one. As mentioned before, this also shows that if the base solver is exact (like ILP) and no more groundings are found, CPI will be exact. If we choose a solution for which new ground formulae were found, the error bound is incremented by the sum of the absolute values of the weights of these ground formulae. Thus we still do well if the remaining formulae have low weight.

Theorem 4.1. *Let* $\hat{\mathbf{y}}$ *be an optimal solution,* $\mathbf{y}'$ *the solution returned by CPI taken from iteration i,* $\hat{\mathbf{y}}_{\mathbf{G}^{i-1}}$ *an optimal solution for the partial network* $\mathbf{G}^{i-1}$ *and*

$$b = \sum_{(\phi,w)\in L}\sum_{B\in G^i_\phi \setminus G^{i-1}_\phi}|w|$$

Then

$$s(\hat{\mathbf{y}},\mathbf{x}) - s(\mathbf{y}',\mathbf{x}) \leq s_{\mathbf{G}^{i-1}}(\hat{\mathbf{y}}_{\mathbf{G}^{i-1}},\mathbf{x}) - s_{\mathbf{G}^{i-1}}(\mathbf{y}',\mathbf{x}) + b$$

Proof. Let $\mathbf{G}^i \setminus \mathbf{G}^{i-1} = \left(G^i_\phi \setminus G^{i-1}_\phi\right)_\phi$ be the newly added groundings. Let C_ϕ be the set of all assignments of free variables in ϕ. Let $\overline{\mathbf{G}^i} = \left(C_\phi \setminus G^i_\phi\right)_\phi$ be the variable assignments that are not instantiated in the partial grounding $\mathbf{G}^i$. Then we can split $s(\hat{\mathbf{y}},\mathbf{x}) - s(\mathbf{y}',\mathbf{x})$ into three parts, a score difference for the previous ground formulae in $\mathbf{G}^{i-1}$, the new ground formulae in $\mathbf{G}^i \setminus \mathbf{G}^{i-1}$ and the remaining ground formulae $\overline{\mathbf{G}^i}$:

$$\begin{aligned} s(\hat{\mathbf{y}},\mathbf{x}) - s(\mathbf{y}',\mathbf{x}) &= s_{\mathbf{G}^{i-1}}(\hat{\mathbf{y}},\mathbf{x}) - s_{\mathbf{G}^{i-1}}(\mathbf{y}',\mathbf{x}) \\ &+ s_{\mathbf{G}^i\setminus\mathbf{G}^{i-1}}(\hat{\mathbf{y}},\mathbf{x}) - s_{\mathbf{G}^i\setminus\mathbf{G}^{i-1}}(\mathbf{y}',\mathbf{x}) \\ &+ s_{\overline{\mathbf{G}^i}}(\hat{\mathbf{y}},\mathbf{x}) - s_{\overline{\mathbf{G}^i}}(\mathbf{y}',\mathbf{x}) \end{aligned}$$

We know that $\mathbf{y}'$ solves $s_{\overline{\mathbf{G}^i}}$ optimally because each feature-weight product is locally optimal, and thus $s_{\overline{\mathbf{G}^i}}(\hat{\mathbf{y}},\mathbf{x}) - s_{\overline{\mathbf{G}^i}}(\mathbf{y}',\mathbf{x}) \leq 0$. Furthermore, in the worst case each term $w \cdot f^\phi_B(\mathbf{y}',\mathbf{x})$ in $s_{\mathbf{G}^i\setminus\mathbf{G}^{i-1}}(\mathbf{y}',\mathbf{x})$ is smaller by $|w|$ than each corresponding term in $s_{\mathbf{G}^i\setminus\mathbf{G}^{i-1}}(\hat{\mathbf{y}},\mathbf{x})$, leading to

$$s_{\mathbf{G}^i\setminus\mathbf{G}^{i-1}}(\hat{\mathbf{y}},\mathbf{x}) - s_{\mathbf{G}^i\setminus\mathbf{G}^{i-1}}(\mathbf{y}',\mathbf{x}) \leq \sum_{(\phi,w)\in L} \sum_{B\in\mathbf{G}^i_\phi\setminus\mathbf{G}^{i-1}_\phi} |w|$$

□

4.4 Semantic Role Labelling Example

Let us use the Semantic Role Labelling example we presented in the previous chapters to see Cutting Plane Inference in action. Note that we will assume an exact base solver (such as ILP) because CPI with an exact base solver is easier to follow.[5]

To remind the reader, we are looking at the following sentence:

> Now competition comes during the late-night slot.

A correct role labeling for this sentence can be seen in figure 4.5. “Now” is a temporal modifier of “comes”, “competition” is its A1 argument (the entity in motion / the comer) and “during the late night slot” is another temporal modifier.

[5] Obviously it is also desirable to use exact solvers whenever possible simply because they provide the most accurate solutions.

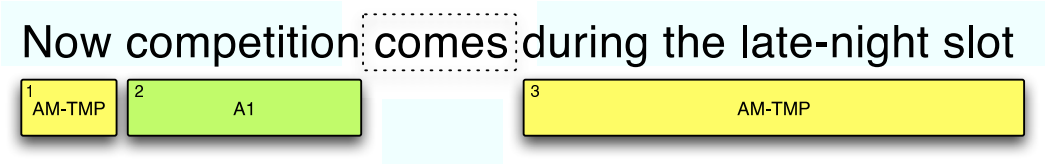


Figure 4.5: Correct role labelling for example sentence.

The input sentence can be expressed as a set of ground atoms (or possible world) $\mathbf{x}$ of the observable predicates. There are two such predicates we consider:[6] *pos*, which denotes the Part of Speech tag of the head of a given constituent, and *overlap*, which indicates whether two constituents are overlapping:

$$\mathbf{x} = \{pos(1,\text{RB}), pos(2,\text{NN}), pos(3,\text{IN}), pos(4,\text{NN}), overlap(3,4)\} \quad (4.8)$$

Note that along with the three actual arguments 1, 2 and 3 shown in figure there is one more candidate constituent: "the late-night slot". This constituent has the number 4.[7] The gold solution can be represented as a possible world, too:

$$\begin{aligned}\mathbf{y} = \quad \{&role(1,\text{AM-TMP}), role(2,\text{A1}), role(3,\text{AM-TMP}) \\ &hasRole(1), hasRole(2), hasRole(3)\}\end{aligned} \quad (4.9)$$

Recall that along with the hidden *role* predicate we are using an additional predicate *hasRole* which indicates whether a candidate is a semantic argument or not, regardless of its semantic role. This predicate will have a significant impact on the behaviour of CPI in this example.

The set of formulae we use is based on the assumptions discussed in chapter 2 and the formulae we presented in chapter 3. To ensure consistency between the *hasRole* and *role* predicates we use the formula

$$hasRole(i) \Leftrightarrow \exists r.role(i,r) \quad (4.10)$$

Furthermore we apply the formula

$$overlap(i,j) \Rightarrow \neg hasRole(i) \vee \neg hasRole(j) \quad (4.11)$$

which ensures that if two constituents are overlapping they cannot both have semantic roles and the formula

$$role(i,\text{A1}) \wedge i \neq j \Rightarrow \neg role(j,\text{A1}) \quad (4.12)$$

[6] In practice, and in the experiments in chapter 5, there will be a large set of observed predicates. However, for brevity we only consider two in this section.

[7] Again, we leave out details on how to pick the set of candidate constituents and refer the reader to chapter 5 for a more in-depth discussion.

which forbids cases where a verb has more than one A1 argument. All the above formulae are deterministic and need to always hold, hence we assign a very high positive weight to them. The final global formula,

$$role\,(i, \text{AM-TMP}) \wedge i \neq j \Rightarrow \neg role\,(j, \text{AM-TMP}) \tag{4.13}$$

with weight $w_{\text{AM-TMP}} > 0$ penalises cases where a verb has more than one temporal modifier. In contrast to the previous formulae, this formula is non-deterministic and the weight w_{tmp} needs to be learnt from data (or carefully tuned).

We mentioned in previous chapters that we assume a correlation between the semantic role of a constituent and the POS tag of the constituent's head word. This correlation is captured through a set of formulae, one for each possible POS tag p and role label p:

$$pos\,(x, +p) \Rightarrow role\,(x, +r) \tag{4.14}$$

where each formula has an individual weight $w^{role}_{p,r}$.

In the following we will assume that both $w^{role}_{\text{NN,A1}}$ and $w^{role}_{\text{IN,AM-TMP}}$ are strictly positive and that all other $w^{role}_{p,r}$ are negative. This means that cases where constituents have both the NN (IN) tag and the A1 (AM-TMP) role are rewarded while all other cases are penalised.[8] Furthermore, we assume that

$$w^{role}_{\text{RB,AM-TMP}} > w^{role}_{\text{RB},r} \tag{4.15}$$

for all $R \neq$ AM-TMP. This means that the penalty for assigning the AM-TMP role to an RB constituent is less than the penalty for any other role. Note that these assumptions are made in order to illustrate the behaviour of CPI and might not hold if the parameters of this simplified model were learnt from data.

We also assumed a correlation between the Part of Speech tag of the constituent's head word and the fact that a constituent is a semantic argument, regardless of its actual role. This correlation is captured using one formula

$$pos\,(i, +p) \Rightarrow hasRole\,(i) \tag{4.16}$$

for each POS tag P. Again, each of these formulae has an individual weight $w^{hasRole}_{p}$. Here we assume that $w^{hasRole}_{\text{RB}}$, $w^{hasRole}_{\text{NN}}$ and $w^{hasRole}_{\text{IN}}$ are strictly positive and all other

[8] In fact, the situation is slightly more complicated because an implication also holds whenever the premise is false. However, this doesn't have an effect on the relative impact of these formulae on the total score.

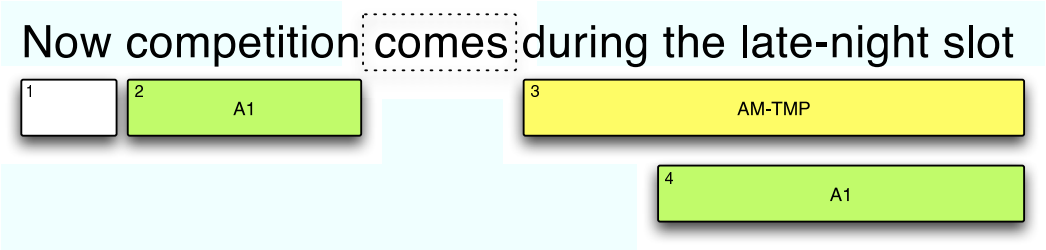


Figure 4.6: Solution after first CPI iteration. The solid line around constituent 1 indicates that this constituent is a semantic argument according to the *hasRole* predicate. The missing role name indicates that no role has been picked via the *role* predicate.

weights are negative. Moreover, we assume

$$w_{\text{IN}}^{hasRole} > w_{\text{NN}}^{hasRole} \tag{4.17}$$

and

$$w_{\text{RB}}^{hasRole} > -w_{\text{RB,AM-TMP}}^{role} \tag{4.18}$$

Roughly speaking, the first assumption says that in case of conflict it is better to use a prepositional phrase as semantic argument than a noun phrase. The second assumption implies that the reward for labelling an adverbial constituent with a semantic role (as opposed to "no role at all") is higher than the price we have to pay for labelling it as AM-TMP. Again these assumptions are made to best illustrate the Cutting Plane Inference algorithm and might not hold in practice.

CPI starts by solving the MAP problem for the partial network $\mathbf{G}^0$ that only contains local formulae. This yields the following solution

$$\begin{aligned}\mathbf{y} = \quad & \{role(2,\text{A1}), role(3,\text{AM-TMP}), role(4,\text{A1}), \\ & hasRole(1), hasRole(2), hasRole(3), hasRole(4)\}\end{aligned} \tag{4.19}$$

as shown in figure 4.6. Here both constituent 2 and 3 have been correctly labeled. However, the NN tag of the head ("slot") of constituent 4 together with a positive weight $w_{\text{NN,A1}}^{role}$ falsely suggest that constituent 4 is an A1 argument. For constituent 1 there is enough local confidence to decide that it has to have a label (that is, that $hasRole(1)$ has to hold) because $w_{\text{RB}}^{hasRole}$ is positive; however, there is not enough "local" confidence to pick the right label. That is,

$$w_{\text{RB},r}^{role} < 0 \tag{4.20}$$

for all possible role labels R. Thus it is "cheaper" to not pick a label at all.

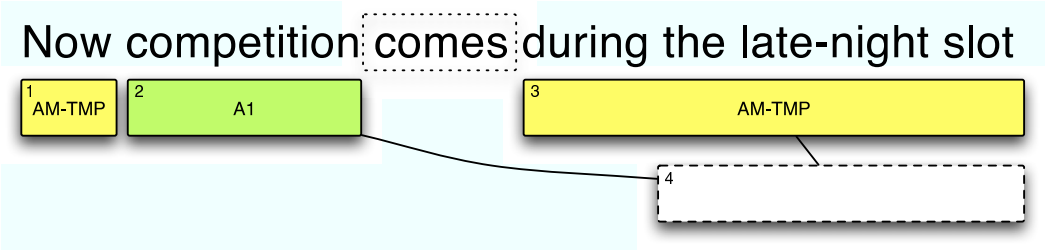


Figure 4.7: Labelling after iteration 2. The edges indicate grounded formulae between variables. The dashed edge for constituent 4 indicates that it is neither a semantic argument according to the *hasRole* predicate nor has a role assigned to according to the *role* predicate.

For the first solution the separation routine finds three ground formulae for which the corresponding features do not maximally contribute to the objective function. The first one is

$$hasRole(1) \Leftrightarrow \exists r.role(1,r) \tag{4.21}$$

for formula 4.10 because constituent 1 is supposed to be a semantic argument according to the *hasRole* predicate but has no role assigned to according to *role* predicate. This ground formula corresponds to the singleton set of bindings $\{\{i/1\}\}$ and is added to the partial grounding for formula 4.10 in $\mathbf{G}^1$ in step 10 of algorithm 4.1.

The second ground formula is

$$role(2,\text{A1}) \wedge 2 \neq 4 \Rightarrow \neg role(4,\text{A1}) \tag{4.22}$$

for formula 4.12 because both constituent 2 and 4 are labelled as A1. Hence we add $\{\{i/2, j/4\}\}$ to the partial ground of formula 4.12 in $\mathbf{G}^1$.

Finally, the third ground formula is

$$overlap(3,4) \Rightarrow \neg hasRole(3) \vee \neg hasRole(4) \tag{4.23}$$

because both constituent 3 and 4 have semantic roles even though they overlap. This means we have to add $\{\{i/3, j/4\}\}$ to the partial grounding of formula 4.11 in $\mathbf{G}^1$.

Note that in all three cases the separation routine had to search for bindings that render the corresponding formula false, because all three formulae have positive weights.

After adding these formulae to the partial problem and calling the propositional base solver we get

$$\begin{aligned}\mathbf{y} = \quad & \{role(1,\text{AM-TMP}), role(2,\text{A1}), role(3,\text{AM-TMP}), \\ & hasRole(1), hasRole(2), hasRole(3)\}\end{aligned} \tag{4.24}$$

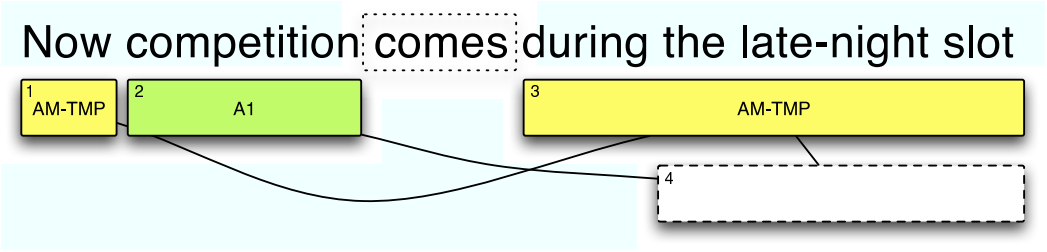


Figure 4.8: Labelling after iteration 3.

as shown in figure 4.7. Now formula 4.21 is satisfied—it must be because it is deterministic. We picked the role AM-TMP for constituent 1 because a) this role has the highest weight and b) removing $hasRole(1)$ to satisfy the hard constraint would result in a lower score because of $w^{hasRole}_{\text{RB}} > -w^{role}_{\text{RB,AM-TMP}}$. Formula 4.22 and 4.23 are satisfied because because both the *hasRole* atom and the *role* atom for constituent 4 were removed. This was the cheapest way to satisfy both formulae. However, since both the phrase "Now" and the phrase "during the late-night slot" have the role AM-TMP the soft formula

$$role(1,\text{AM-TMP}) \wedge 1 \neq 3 \Rightarrow \neg role(3,\text{AM-TMP}) \tag{4.25}$$

is violated and $\{\{i/1, j/3\}\}$ is added to the partial grounding for formula 4.13 in $\mathbf{G}^2$.

After solving the partial problem one more time, the solution is again

$$\begin{aligned}\mathbf{y} = \quad &\{role(1,\text{AM-TMP}), role(2,\text{A1}), role(3,\text{AM-TMP}), \\ &hasRole(1), hasRole(2), hasRole(3)\}\end{aligned} \tag{4.26}$$

and can be seen in figure 4.8, together with the added edges of the partial Markov Network. Again the formula

$$role(1,\text{AM-TMP}) \wedge 1 \neq 3 \Rightarrow \neg role(3,\text{AM-TMP}) \tag{4.27}$$

is violated because the penalty taken for violating this constraint is not high enough to overrule the global confidence (based on the weights of the local formulae and the already added hard constraints) in the current solution. However, since this formula was already added in the previous iteration no more *new* violated formulae are found and thus the algorithm terminates. The final solution $\mathbf{y}'$ is the solution of the last iteration—it has to be the true optimum because of theorem 4.1 and the fact that we used an exact base solver and no more new formulae were found.

4.5 Separation

An integral part of CPI is the separation step in which we need to find all bindings B for the free variables in a formula ϕ with weight w so that

$$w \cdot f_B^{\phi}(\mathbf{y}, \mathbf{x}) < \max_{\mathbf{y}'} w \cdot f_B^{\phi}(\mathbf{y}', \mathbf{x}) \tag{4.28}$$

for a given solution $\mathbf{y}$. In other words, given a solution $\mathbf{y}$, an observation $\mathbf{x}$ and, a formula ϕ and a weight w we search for every variable assignment B for which the term $w \cdot f_B^{\phi}(\mathbf{y}, \mathbf{x})$ is not optimal, i.e. there exists a $\mathbf{y}'$ for which $w \cdot f_B^{\phi}(\mathbf{y}', \mathbf{x})$ is larger than $w \cdot f_B^{\phi}(\mathbf{y}, \mathbf{x})$. Here the Statistical Relational Learning paradigm comes into play. In a (propositional) Markov Network we do not have any higher order descriptions of its features. Performing separation then means evaluating all features of the network. Moreover, we are also required to maintain the set of all features in memory because we need them in each iteration.[9] In cases where we have millions (or hundreds of millions, see chapter 5) of features this leads to runtime and memory requirements that can offset the potential gains achieved by simplifying the problem for the propositional base solver.[10]

However, in Markov Logic we can do better and exploit the first order representation of features. There are two cases to consider. If $w > 0$ we have to find bindings B with $f_B^{\phi}(\mathbf{y}, \mathbf{x}) = 0$, that is, bindings for which the ground formula $\phi[B]$ is false (or $\neg\phi[B]$ is true) in the possible world defined by $(\mathbf{x}, \mathbf{y})$. Correspondingly, for $w < 0$ we have to find bindings B for which $\phi[B]$ is true in $(\mathbf{x}, \mathbf{y})$. This observation simplifies the separation problem significantly because the problem of finding all bindings for which a formula ϕ holds in a possible world has been well studied in the database literature. It is known as the *evaluation problem* [Flum et al., 2002].

A naive separation routine iterates over all bindings for the formula ϕ and tests whether the formula holds (or doesn't hold for $w > 0$). Effectively this amounts to separation in a propositional model, albeit without the memory overhead. It is easy to see that this routine would take $O\left(|C|^{\phi_n}\right)$ steps where C is the set of constants and ϕ_n is the number of free variables in formula ϕ. However, many formulae can be processed more efficiently [Flum et al., 2002]. Roughly speaking, Flum et al. [2002] show that conjunctive queries (conjunctions of literals) can be processed in time $O(\|\phi\| \cdot \|(\mathbf{x}, \mathbf{y})\|)$

[9] We could also store them on disk; however, this would create significant additional IO overhead.

[10] For example, in the case of Joint Entity Resolution we will encounter a setting with 500^2 nodes and 500^3 features, each connecting 3 nodes. To represent a node we need 3 bytes (typically one would use 4) and hence one needs 9 bytes to represent a feature. This leads to approximately one gigabyte of memory needed for separation alone.

where $\|\phi\|$ is the length of the query (number of atoms) and $\|(\mathbf{x},\mathbf{y})\|$ is the size of the data,[11] provided that the queries are acyclic.[12] If the query is not acyclic, its tree-width[13] plays a major factor in determining its complexity.

In practice we solved the separation problem by storing the ground atoms of a predicate in a database table and mapping each formula to a database query, using additional database indices wherever possible. We observed that even for conjunctive formulae with high tree-width or different types of non-conjunctive formulae the cost of query evaluation was marginal when compared to the cost of numeric optimization.

4.6 Ordered Cutting Plane Inference

The bound on accuracy of CPI in theorem 4.1 is not based on the order in which we have added ground formulae. This means that we do not need to instantiate all newly found formulae all of the time. This can be crucial because the order in which we instantiate formulae can have a significant effect on the runtime of CPI.

For example, when applying CPI in the context of Semantic Role Labelling we might encounter a solution where many candidates have more than one role, violating the "no more than one role per constituent" constraint. We might also observe that the role A1 has been assigned to several candidate constituents. This violates the "no more than one A1 argument per predicate" rule.

Instead of adding both types of constraints in the current iteration i we could only add the "no more than one role per constituent" constraints. Enforcing these will result in a significantly sparser solution where A1 roles are less likely and in turn there might be fewer violated "not more than one A1 argument" constraints. Adding these constraints in iteration $i+1$ results a smaller network than the one we would have created if we had added all constraints in iteration i. Hence, instead of solving a potentially large network in iteration i we solve two simpler networks in iteration i and $i+1$. In other words: the order in which we process constraints allows us to trade off the size of the partial problems and the number of CPI iterations.[14]

[11] Or more precisely, the size of the encoding of the data. Hence it does not only depend on the number of atoms but also on the arity of each predicate because for each atom we need one field per argument.

[12] That is, the graph that connects variables of the formula when they appear in the same atom is acyclic.

[13] A number that measures the tree-likeness of a general graph.

[14] Note that it is possible that no constraints of the second type are violated in iteration $i+1$; Assuming that no other types of formulae need to be added, the number of CPI iterations remains the same, regardless of whether we process formulae in order or not. However, generally speaking there is no

This motivates the introduction of *Ordered CPI* . In Ordered CPI we provide a partition $(L_1, \ldots, L_m)$ of the MLN L with

$$\bigcup_i^m L_i = L \tag{4.29}$$

and

$$\forall i \neq j : L_i \cap L_j = \emptyset \tag{4.30}$$

that indicates the order in which constraints should be processed. For example, in the case of Semantic Role Labelling we could split the knowledge base L into three parts (L_1, L_2, L_3) where L_1 only contains the formula which ensures that constituents have at most one role, L_2 only contains the formula which ensures that a predicate has at most one semantic argument of type A1 , and L_3 contains all remaining formulae.

Ordered CPI is shown in Algorithm 4.2. It only differs from the generic CPI version in how separation is performed. Instead of checking for all formulae we start by looking at the formulae in L_1. If new ground formulae are found which are locally suboptimal for the current solution $\mathbf{y}$ we add these to the partial grounding $\mathbf{G}_i$, terminate the repeat loop and use $\mathbf{G}_i$ as in Algorithm 4.1. If no more new locally suboptimal ground formulae are found for L_1 we consider the formulae in L_2 and so on, until we either find some new ground formulae or have no more partitions to check.

4.7 Related Work

As we will shortly show, the idea of using Cutting Planes for MAP inference in probabilistic models is not new. However, Cutting Plane Inference, as proposed in this chapter, has a set of properties that can provide advantages over existing approaches. In particular, CPI

- is *compatible* with any propositional MAP solver,
- supports *hard and soft* constraints (without the need for branch-and-bound) ,
- uses *first order* information for separation,
- can be implemented with *off-the-shelf* optimization and database software and

guarantee that a ground formula that was violated (locally suboptimal) in iteration i and satisfied (locally optimal) in $i+1$ will not be violated at later iterations. Hence processing formulae in order may result in more than one extra iteration.

Algorithm 4.2 Ordered Cutting Plane Inference

Require: A partition $(L_1, \ldots, L_m)$ of a Markov Logic Network, a vocabulary V, a base solver *BS*, an observation $\mathbf{x}$, an initial grounding $\mathbf{G}^0$ and the maximum number of iterations *maxIterations*

$i \leftarrow 0$
$\mathbf{y}' \leftarrow \mathbf{0}$
repeat
  $i \leftarrow i + 1$
  $\mathbf{y} \leftarrow$ solve $\left(\mathbf{G}^{i-1}, \mathbf{x}\right)$ using base solver *BS*.
  if $s(\mathbf{y}, \mathbf{x}) > s(\mathbf{y}', \mathbf{x})$ **then**
    $\mathbf{y}' \leftarrow \mathbf{y}$
  end if
  $o \leftarrow 1$
  $\mathbf{G}^i \leftarrow \mathbf{G}^{i-1}$
  repeat
    for each $(\phi, w) \in L_o$ **do**
      $G^i_\phi \leftarrow G^i_\phi \cup \text{Separate}(V, \phi, w, \mathbf{y}, \mathbf{x})$
    end for
    $o \leftarrow o + 1$
  until $\mathbf{G}^i \neq \mathbf{G}^{i-1}$ or $o > m$
until $\mathbf{G}^i = \mathbf{G}^{i-1}$ or $i >$ *maxIterations*
return $\mathbf{y}'$

- is framed in a *Probabilistic First Order Logic* language.

Some existing methods share some of these properties but no method shares all of them. Also note that none of the prior work mentioned here presents an analysis of the number of iterations needed and the final problem size. Thus a major contribution of this thesis are the observations we present in chapter 6. We believe that they help to give a better understanding as to when MAP inference with Cutting Planes can work well, and it when it cannot.

4.7.1 Cutting Planes in Outer Loop

Many Cutting Plane algorithms for MAP inference can be characterized as follows: we are given a model, such as a Markov Network or weighted Finite State Machine, and a MAP method for this model, such as Belief Propagation or the Viterbi Algorithm. However, the model can be too complex to be solved directly, and hence an outer loop is introduced that passes relaxed versions of the original problem to the solver. Clearly, CPI is an instance of this class of algorithms.

Anguelov et al. [2004] use an outer loop Cutting Plane algorithm to handle a large set of hard constraints in a Markov Network and apply Belief Propagation to the partial networks. Likewise, Riedel and Clarke [2006] present an incremental approach to solving the MAP problem for Maximum Spanning Tree dependency parsing that can be seen as an outer loop Cutting Plane algorithm. In contrast to CPI, in both approaches we cannot incrementally add soft constraints, which appear frequently in NLP and other domains. Moreover, their methods are tailored to a specific model. This means that applying them to other models requires us to design and implement new separation methods. In contrast, CPI can be re-used easily as long as we can frame constraints in Markov Logic.

Tromble and Eisner [2006] present a Cutting Plane approach for MAP inference in Finite State Automata. They do consider soft constraints; however, while CPI processes soft constraints in the same fashion as hard constraints, their algorithm adds soft constraints using a branch-and-bound approach. Whenever a soft constraint is violated their algorithm branches and considers two cases: for the first branch the constraint is added as a hard constraint to the current problem (and all subsequent problems of the same branch); for the second branch the constraint remains violated and a penalty is given to the current solution (and all subsequent solutions of the same branch) based on the weight of the constraint.

While this approach can work well for problems with very few violated soft constraints, for cases with many soft constraints a lot of branching is required, which corresponds to many expensive calls to the optimizer. In contrast, CPI essentially leaves the handling of soft constraints to the propositional base solver. The solver might use a branch-and-bound scheme internally—then we don't do better; but it can also process the soft constraints in a more effective way. For example, in the ILP formulation of MAP inference in Markov Networks in section 2.3.2, weighted formulae are captured through a set of auxiliary variables and linear constraints. In many cases the LP relaxation of the corresponding ILP yields integer solutions and hence no branch-and-bounding is necessary within the ILP solver. Leaving the handling of soft constraints to the base solver also simplifies the implementation of CPI because no branch tree is required.

4.7.2 Cutting Planes for the Marginal Polytope

In section 2.3.2.1 we presented a mapping from Markov Networks to ILPs. A similar mapping exists for Linear Programs (LPs). In this case the set of valid marginal distributions for a Markov Network is described through a large set of linear constraints. This set is usually referred to as the *Marginal Polytope*. It can be shown that solutions within the Marginal Polytope which maximize the linear MAP objective are integer and solutions to the MAP problem. Hence we can use the Marginal Polytope to perform MAP inference with an LP solver instead of an ILP solver—this can be helpful because solving ILPs is more difficult than solving LPs.

However, the number of linear constraints needed to describe the Marginal Polytope is very large. Hence Sontag and Jaakkola [2007] present a Cutting Plane algorithm that incrementally add constraints that tighten an outer bound on the marginal polytope. They focus on a class of constraint that has to hold for the sum of all marginal probabilities over a cycle in the Markov Network.

In contrast to CPI their algorithm is tailored to be used with an LP solver because the linear cycle constraints are only meaningful in a Linear Program. For example, it is not immediately clear how to enforce such constraints in a MaxWalkSAT or Belief Propagation algorithm. This might also hold for future algorithms that solve MAP problems in Markov Networks. CPI, on the other hand, operates in terms of the Markov Network (and not in a lower level representation of it). This allows us to plug-in arbitrary base solvers.

Moreover, their separation algorithm finds violating cycles in $O(m \cdot n)$ time where m is the number of edges in the network and n the number of variables. Hence, for some Markov Logic applications, such as large scale Joint Entity Resolution (see chapter 5) with millions of variables and edges, separation becomes difficult.

Later Sontag et al. [2008] introduced a Cutting Plane algorithm for the Marginal Polytope that focuses on a different set of constraints. Effectively this algorithm adds constraints which enforce consistency between clusters of variables. However, in contrast to the work of Sontag and Jaakkola [2007] these constraints can be handled using a Belief Propagation variant. This is helpful in cases where even the relaxed Marginal Polytopes is too large to be efficiently solved with off-the-shelf LP solvers. However, again this method does not allow us to plug-in any MAP solver of choice. Moreover, the separation algorithm is essentially propositional: it iterates over set of predefined clusters and tests which ones would be helpful to improve the solution. In their work they use all 3-clusters after triangulation—for Markov Logic problems this would often lead to a large number of clusters and render separation difficult.

There is also a more general issue: when we want to directly use these methods for MAP inference in Markov Logic we need to instantiate the complete ground network and pass it to the algorithm. This can require both a lot of time and memory. In contrast, CPI can avoid this process because it is framed in terms of a Markov Logic Network and not in terms of the complete ground network; during CPI parts of this network will be instantiated, but only if this seems necessary.

Finally, we believe that Cutting Planes algorithms in the Marginal Polytope are orthogonal to our work in the sense that we can use them as base solver in CPI. This can lead to further improvements of CPI because in some cases we observed that ILP base solvers perform poorly even for small problems. Replacing the ILP solver with one of the algorithms we mentioned here might be helpful in such cases.

4.7.3 Cutting Planes in ILP Solvers

Note that many Integer Linear Programming solvers use Cutting Plane algorithms to ensure that solutions are integer. Instead of (or in combination with) using a branch-and-bound algorithm such solvers inspect the solution for fractional variables. If such variables exist, a set of linear constraints are added. These constraints are known to hold for the integer solution but are violated in the current fractional solution. By adding these constraints we hope that in the next solution fractional variables are less

likely.[15]

There is a strong connection between such algorithms and the Cutting Plane algorithms for the Marginal Polytope. In both cases integer constraints are enforced through large sets of linear constraints which are incrementally added. However, a generic ILP solver has to use general-purpose constraints known to hold for all ILPs while the work presented in section 4.7.2 makes use of properties of the Marginal Polytope.

Cutting Plane algorithms in ILP solvers share a similar disadvantage with those designed for the Marginal Polytope: if we use an ILP solver in order to find the MAP solution of a Ground Markov Network without CPI we are still required to instantiate the complete network. Again this can be difficult in cases where the networks are very large. Moreover, in contrast to application-specific constraints that enforce integer solutions, the general-purpose constraints used by generic ILP solvers are often not very effective. Moreover, even if they were effective, the large number of explicit application constraints (the set of constraints described in section 2.3.2) would still be difficult to solve. CPI, on the other hand, can reduce the set of application constraints by reducing the size of the Markov Network that is mapped to an ILP.

Finally, there is an interesting connection between Cutting Planes algorithms for Integer Linear Programming and resolution in a propositional knowledge base. [Hooker, 1988]. Roughly speaking, there is a type of cutting plane/linear inequality added in an ILP Cutting Plane algorithm that corresponds to the resolvent of two clauses in the resolution algorithm. However, the resolution algorithm only applies to deterministic knowledge bases and is not designed for MAP inference. Hence we cannot make any direct comparison between resolution with Cutting Planes and Cutting Plane Inference as proposed in this thesis.

4.7.4 Lazy Inference

CPI is also similar in nature to LazySAT and Lazy Inference [Poon et al., 2008] in general: both approaches seek to improve the performance of a propositional base solver by avoiding to instantiate the full ground network. However, while CPI only instantiates new parts of the ground network once the base solver has optimised the current partial network, Lazy Inference instantiates new parts of the network whenever they may be needed within the original algorithm to be made lazy. In this sense CPI is a

[15] It is this type of Cutting Plane Algorithm that was mentioned by Roth and Yih [2004] in their work on global inference for Natural Language Processing.

patronizing meta algorithm: instead of providing a base solver X with edges whenever X asks for them, it requires X to first solve the problem at hand, and only provides more edges once the problem is solved.

The approach of CPI is helpful when CPI indeed "knows better" than the base solver. This is the case when X will, if run without CPI, consider edges that CPI would never present to X. Here CPI instantiates only a subset of the edges activated during Lazy Inference. If this subset is sufficiently small, CPI can call X several times and still spend less time in X than Lazy Inference because smaller problems can be solved quicker. In section 5.3.4.1 of chapter 5 we can indeed observe that the total time in the base solver is dramatically reduced even though it is called more than twice on average. This is impossible with Lazy Inference (but does not mean that CPI is necessarily faster here because the time outside of the solver has to be considered, too). However, if X will not take into account the redundant edges that CPI avoids to instantiate, Lazy-X will never consider more edges than CPI. Since Lazy-X runs X only once, it is likely the better choice in this case.

To see how CPI can instantiate fewer edges than Lazy inference, consider the case where we use MaxWalkSat as a base solver. Now assume that MaxWalkSAT just picked an unsatisfied clause and randomly chose a member variable A to flip. Moreover, assume that A was false before the flip and will be false in each of the intermediate solutions inferred during CPI (because all sub-networks have a strong preference for A to be false). Finally, assume that A is a member of a clause c that CPI has not yet instantiated, and which becomes unsatisfied after flipping A. In this case LazySAT will activate the clause and CPI will not. In other words, if we give MaxWalkSat a bit more time, it would find out that A can actually remain false, and that c does not have to be instantiated.

Note that CPI has a practical advantage over Lazy Inference: it can consider the base solver as a black box. By contrast, Lazy Inference requires the developer to change the implementation of the algorithm to be made lazy. While adapting the implementation of a relatively simple algorithm such as MaxWalkSAT is straight-forward, it would be much more difficult for complex software such as an ILP solver. Moreover, if the solver is commercial and closed-source (such as the very fast CPLEX solver), it is impossible for us to convert it into a lazy version.

How CPI compares to Lazy Inference in practice is still an open question. In this work (and the experiments of chapter 5) we focus on the relative impact of CPI with respect to its base solver and therefore we do not answer this question here. However,

in future work we will perform a more detailed comparison of Lazy Inference and CPI that will investigate the relative advantages and disadvantages of CPI when applied to real world MLNs.

4.7.5 Lifted Belief Propagation

CPI and Lazy Inference do *lifted inference;* that is, they answer queries without materializing all atoms and formulae. Lifted Belief Propagation [Singla and Domingos, 2008] is another instance of lifted inference. Conventional (Loopy) Belief Propagation [BP, Murphy et al., 1999] is a message passing algorithm where the nodes and factors/features of a Markov Network repeatedly send messages to each other until the message values converge. We can then use the final messages to calculate the marginal probabilities of each variable (or their MAP state if we slightly alter the way to calculate messages). The insight behind Lifted Belief Propagation is the observation that in Ground Markov Networks many features and nodes send and receive the same messages. This allows Singla and Domingos to subsume several nodes and features into *supernodes* and *superfeatures,* respectively, that form a *lifted network* in which regular Belief Propagation (with slightly different messages) can be performed. Since this lifted network is often dramatically smaller than the original Ground Markov Network, significant improvements for the runtime of BP can be observed, despite the increased runtime for the construction of the lifted network.

Although both CPI and Lifted BP are similar in the sense that they work with simplified versions of the ground network, they both exploit different characteristics of the network. Lifted BP makes use of the fact that the network structure described by a Markov Logic Network can be very regular and repetitive, and hence BP will send the same messages in many cases. CPI, on the other hand, benefits from the fact that many features reward or punish the same properties of a solution. For example, in the case of Semantic Role Labelling many local features implicitly penalise violations of the "no more than one argument" constraint (cf. section 5.3). Here the regularity of a network is not crucial.

For both algorithms we can can construct scenarios where one fails while the other one does very well. For example, consider a case where for every ground atom there is a different set of active local formulae, but all these formulae implicitly punish violations of the global formulae. In this case CPI is done after one iteration. However, every node receives different messages and hence Lifted BP cannot simplify the net-

work. Now consider a Markov Logic Network with a predicate *pred* and the formulae $pred(x)$ and $pred(x) \wedge pred(y)$ where the first has a negative weight and the second a positive weight. If CPI starts by instantiating the local formula $pred(x)$ the first solution is the empty set of ground atoms. This violates all groundings of the global formulae, and hence CPI instantiates the complete network in the next step. However, in this case the network structure is extremely regular and repetitive, and Lifted BP can create a very small lifted network.

We think that CPI and Lifted BP can complement each other. For example, it could be possible to run CPI in an outer loop that creates lifted networks instead of Ground Markov Networks. In these networks we can then perform Lifted BP, or possibly lifted versions of ILP or MaxWalkSAT. This would allow us to exploit both the regularity and redundancy we find in many Ground Markov Networks.

4.7.6 Knowledge-Based Model Construction

A related approach to inference is followed in Knowledge-based Model Construction (KBMC) [Wellman et al., 1992, Ngo and Haddawy, 1997, Kersting and De Raedt, 2001]. Generally speaking, in KBMC we maintain a knowledge base (KB) that contains logical and probabilistic knowledge, often in the form of augmented Horn clauses. This knowledge is then used to construct a probabilistic (or decision-theoretic) model given some evidence and query propositions. This is to be contrasted with approaches that require the user to create one large model for all possible pairs of evidence and query, or manually construct a special model for each case.

For example, we could have a knowledge base with general information about causes and symptoms of a disease. Now were are given a certain patient, observe her symptoms and want to find out their cause. We then use the general information contained in the KB to generate a Graphical Model that correlates the symptoms and their potential causes for this particular patient. Usually this construction process grounds all clauses of the knowledge base that are *logically* related to the query and evidence, using forward and backward chaining. That is, we instantiate only those ground clauses that can prove evidence from the queries, and vice versa.

In both CPI and KBMC we avoid to fully ground a Graphical Model. Moreover, in section 6.4 we will see that CPI instantiates (a subset of) the refutations of ground formulae. Since the refutation a ground clause is equivalent to a proof of each atom in the body and a refutation of its head, the instantiated networks consist of subsets

of ground atom proofs, just as in KBMC. However, there is a conceptional difference. In KBMC network construction happens in advance and is purely based on logical properties of the KB; if a logical proof exists, it will be instantiated. By contrast, CPI instantiates proofs incrementally and only if enough (numerical) confidence exists—we illustrate this further in section 6.4.2. From another point of view, and in relation to what we have said in section 4.7.5, we do not (explicitly) consider logical connections between query and evidence but the partial redundancies of ground formulae in our model.

4.8 Conclusion

In this chapter we have presented Cutting Plane Inference (CPI), a method that iteratively solves small sub-problems of the complete ground network using a propositional base solver and adds ground formulae that are not yet included in the current network but could potentially lead to an improved solution. The process of finding ground formulae to be added (separation) is implemented through a first order query processing engine. This engine finds all groundings of a first order formulae which are satisfied (or not satisfied) in the current solution. We also presented a variation of CPI, namely ordered CPI, which processes formulae in a user-defined order.

We have showed that CPI is as accurate as the base solver, but on a smaller problem. This suggests that CPI can improve the accuracy of the base solver—this hypothesis is validated in chapter 5.

Finally we compared Cutting Plane Inference with related work and argued that CPI has advantages over previous approaches both as an MAP algorithm for Markov Logic and as a Cutting Plane algorithm for MAP inference.

This chapter has described and discussed CPI from a conceptual point of view. However, we have yet to show the effectiveness of CPI in practice—this will be done in the following chapter.

Chapter 5

Experimental Results

In this chapter we will evaluate Cutting Plane Inference in order to support the following hypothesis:

> CPI can significantly improve the efficiency of a propositional solver while remaining at least as accurate.

More precisely, given a propositional solver X, CPI with base solver X can be faster and use less memory while being at least as accurate as X by itself.

For the above hypothesis to hold in a more general fashion we should be able to observe improvements for more than one propositional method. Here we choose two: MaxWalkSAT and Integer Linear Programming. Both differ substantially; while the first is approximate and random, the latter is exact and mostly deterministic.[1]

We believe the above claim also holds across different applications. To support this assumption we choose to test CPI on two rather different applications. The first one is Semantic Role Labelling (SRL), the second is Joint Entity Resolution (ER). Both differ in several respects: the second task requires us to solve significantly larger problems than the first one; the second task uses nondeterministic global constraints, the first one does not; finally, the first task uses MLNs with significantly more formulae than the second one.

We will also try to use this chapter to gain a better understanding of the behaviour of CPI (but we refer the reader to chapter 6 for an in-depth investigation). Finally, we will seek to evaluate the impact of Ordered Cutting Plane Inference.

Note that the experiments presented in this chapter, apart from those that concern memory consumption and Ordered CPI, have been presented in [Riedel, 2008]. While

[1] We write "mostly" because an ILP solver can sometimes make random decisions to break ties or increase robustness.

the results reported here show the same trends, they differ slightly (in terms of CPI iterations for Semantic Role Labelling and score for MaxWalkSAT) due to minor bugs in the earlier version of our code.

5.1 Experimental Setup

As mentioned above, we seek to evaluate the impact of using CPI both in terms of efficiency and accuracy. In the following we present the metrics used to measure this impact. For both applications we use roughly the same experimental setup unless mentioned otherwise.

5.1.1 Efficiency

To evaluate the efficiency of CPI we choose to measure the time from the beginning of the inference procedure until the result is available. This includes the generation of ground networks, the separation routine and the execution of wrapper code to communicate with the base solver. This approach differs from previous work [Singla and Domingos, 2006b], where runtime is measured by the average time per MaxWalkSAT flip (including grounding). However, since it is not possible to evaluate the efficiency of an ILP solver in terms of MaxWalkSAT flips, we directly measure runtime.

In addition to the total runtime of inference we measure the time that is spent within the actual inference algorithm. This will allows us to evaluate where potential bottlenecks lie: in the propositional solver itself, or in the construction of the ground network and communication between propositional solver and Markov Logic engine.

Finally, we also evaluate efficiency in terms of memory usage. To this end we measure the number of global ground formulae in the final ground network. In the case of stand-alone solvers this corresponds to the number of global (non-initial) ground formulae in the complete ground network. This will (a) give us a platform- and implementation-independent metric for memory use, and (b) help us to understand how CPI manages to improve the efficiency and accuracy of its base solver. Note that the number of ground formulae we present does not include the number of initial ground formulae (in our cases these are the local formulae), because they are constant regardless of the system we observe.

In addition we will present the size of the Java virtual machine after all instances have been processed. The latter metric has to be taken with caution. It is machine-

dependent and also sensitive to memory fragmentation effects that arise when processing multiple instances in sequence.[2] Nevertheless, it is a measure that allows us to evaluate the impact of CPI in practice.

5.1.2 Linear Score

In order to evaluate how well a system predicts the most likely solution we look at the value of the linear scoring function

$$s\left(\mathbf{x},\mathbf{y}'\right) = \sum_{(\phi,w)\in L} w \cdot \sum_{B\in\mathcal{B}(\phi)} f_B^{\phi}\left(\mathbf{x},\mathbf{y}'\right) \tag{5.1}$$

for the system guess $\mathbf{y}'$.[3] This score is easier to calculate than the maximum *a posteriori* probability because no normalisation is required. Moreover, the actual probabilities will be close to zero and difficult to compare. When the solution violates deterministic constraints we will present both the number of hard constraint violations and the score based only on non-deterministic formulae. This allows us to judge the quality of a solution independent of the (very large but arbitrarily picked) weight of deterministic formulae.

5.1.3 Number of Iterations

We will also report the number of iterations each system needs. This number corresponds to the number of times the base solver is called. Thus for systems that do not use CPI this number equals one. The number of iterations does not help us in supporting our hypothesis. However, it will allow us better understand the behaviour of CPI.

[2] To improve the speed of grounding a network we make use of primitive arrays. To avoid the cost of creating these arrays we re-use them in multiple instances. Sometimes a new array has to be created because an old one is too small. In this case the new array will be created on the top of the heap while the old array will be garbage collected. This frees a fragment on the heap that can only be re-used when arrays are required that fit into it. Over time many such fragments will appear and memory usage rises.

[3] In fact, we use the score based on an MLN M' in which all global nondeterministic formulae are inverted according to the transformation in section 6.1.2. This yields equivalent scores (representing the same conditional probabilities). The main benefit of this approach is that scores are easier to calculate (for the type of formulae in our MLNs it is easier to search for groundings that violate the formula than for groundings that satisfy it). It also leads to lower numbers that are easier to present and compare.

5.1.4 Application Accuracy

Finally we evaluate the accuracy on the task at hand in terms of the F1 measure

$$F1 = \frac{2 \cdot P \cdot R}{P + R} \tag{5.2}$$

where recall R is the number of true ground atoms that were recovered (true positives, *#TP*) divided by the total number of true ground atoms (*#total*)

$$R = \frac{\#TP}{\#total} \tag{5.3}$$

and precision P is the number of true ground atoms that were recovered divided by the total number of atoms found (*#found*)

$$P = \frac{\#TP}{\#found} \tag{5.4}$$

Strictly speaking this does not measure the quality of the given inference method, but rather the quality of the trained model. However, it will help us to tell whether improvements in linear score accuracy are worthwhile in the sense that they improve application accuracy. It also allows us to show that our models perform at state-of-the-art level and are of practical significance.

5.2 Systems

Some of our results depend on the computational environment used and the actual implementation of the propositional solvers as well as their parameters. In this section we will thus provide some relevant details.

For all experiments we use our own Markov Logic implementation[4] running on a Pentium 4 at 2.8Ghz with 4Gb RAM (of which only 2.4Gb could be used in one process). It is completely written in Java with the exception of the ILP and MaxWalkSAT solver components.

5.2.1 MaxWalkSAT

Initially we used our own implementation of MaxWalkSAT (MWS) as part of our Markov Logic platform. However, existing work in Markov Logic [Richardson and Domingos, 2006, Singla and Domingos, 2006b] has been using the C implementation

[4]This implementation is available at *http://thebeast.googlecode.org*.

of Kautz et al. [1997]. This implementation is (a) much faster and (b) more accurate because it performs some additional heuristics, such as a special treatment of deterministic clauses. We therefore replaced our implementation with this off-the-shelf solver.

We will refer to MaxWalkSAT based systems using the schema *MWS-R#-F#,* where *R#* refers to the number of runs and *F#* refers to the number of flips. For example, the name MWS-10-10k refers to a MaxWalkSAT system using 10 restarts and 10,000 flips.

Note that for the problems we tackle in this chapter we have found that MWS performs best for a greedy-step probability of 0.5, and hence all our systems use this configuration.

5.2.2 Integer Linear Programming

For ILP we use the freely available toolkit *lp_solve*,[5] version 5.0.10, written in C. This solver uses the Simplex Method for Linear Programming, and a branch-and-bound algorithm to enforce integer constraints.

One advantage of using ILP is that at least in terms of accuracy there is no parameter tuning necessary because solvers return exact solutions by default. There are several parameters that optimize runtime, possibly trading off accuracy. However, in our experiments we used lp_solve as is, with its default parameters.

Systems that use Integer Linear Programming are denoted using the term *ILP*.

5.2.3 Cutting Plane Inference

The Cutting Plane Inference system uses the algorithm presented in chapter 4. In our experiments we use our own implementation of the database language D [Darwen and Date, 1995] to perform separation.[6]

The name *CPI-X* refers to a system that use Cutting Plane Inference with base solver *X*. For example, *CPI-MWS-10-10k* refers to a Cutting Plane Inference system using MaxWalkSAT with 10 runs of 10,000 flips as base solver.

In all cases the CPI system uses the local formulae as the initial network. This means that the first optimization problem can be trivially solved by optimizing ground

[5]This implementation is available at *http://sourceforge.net/projects/lpsolve.*

[6]In designing our system we tried to implement most operations, not just separation, through the execution of database statements. For example, ILP constraints and partial groundings are represented through database tables, and the process of generating a set of ILP constraints from partial groundings is realized through a database operation. This required some non-standard data types and database operators that we specifically designed for this purpose. However, the separation process does not require these extensions and could be implemented by an off-the-shelf database.

atom states locally. Hence for the system CPI-X we will use this local optimisation instead of solver X for the first partial problem. This means that if the system CPI-X performed n iterations the solver X was called $n-1$ times.

5.3 Semantic Role Labelling

Semantic Role Labelling (cf. section 1.2) refers to the task of identifying and classifying the arguments and modifiers of verbs, as in

$$[_{\text{AM-TMP}}\text{Now}]\ [_{\text{A1}}\text{competition}]\ \text{comes}\ [_{\text{AM-TMP}}\text{during the late-night slot}].$$

for the verb "comes". Labels such as A1 serve as placeholders for actual roles of the given verb, such as "comer" in the above case.

In this section we will present experiments based on a state-of-the-art Semantic Role Labelling model, inspired by previous work [Punyakanok et al., 2005, 2004] that exploits global correlations between the different semantic arguments of a verb. Essentially this model can be seen as an extension of the model we present in section 3.5 and based on the observations we mentioned in sections 1.2 and 2.2.

5.3.1 A Markov Logic Network for Semantic Role Labelling

We choose to follow previous work by using the constituents returned by a parser as candidate phrases. Before labelling we prune phrases using a high recall[7] heuristic [Xue and Palmer, 2004], which rules out constituents based on their position in the parse tree relative to the verb. For example, the siblings of the predicate are kept while their descendants are discarded unless their parents are prepositional phrases.

The Markov Logic Network we apply is very similar to the networks we have described in the examples of the previous chapters. Again we represent each candidate constituent using an integer constant. Along with these candidates we have a set of constants that represent possible labels,[8]

$$\mathit{Label} = \{\text{A0}, \text{A1}, \ldots\} \tag{5.5}$$

[7] That is, we are using a heuristic which rarely filters out constituents that are semantic arguments.

[8] Note that the interpretation of most of the possible semantic labels depends on the the predicate verb. For example, in most cases A0 refers to the agent of a verb, but in some cases in may be A1 that refers to the agent (as in out example sentence).

The predicate[9]

$$role : Int \times Label \tag{5.6}$$

denotes the relation between candidate constituents and their semantic roles. Ultimately we want to predict all ground atoms of this predicate. As we mentioned earlier, it can be helpful to introduce the auxiliary predicate

$$hasRole : Int \tag{5.7}$$

that indicates whether a candidate is or is not to be labelled, regardless of what label to be used.

Using this predicate along with a hard constraint

$$hasRole\,(i) \Rightarrow \exists l.role\,(i,r) \tag{5.8}$$

helped us to improve recall because it is possible to learn that a label is required even though the context does not clearly indicate which label to pick. Notice that this formula only works in one direction, in contrast to the equivalence formula we presented in chapter 3. We chose this formula because it yielded better performance in comparison to a formulae that ensures bidirectional consistency between *role* and *hasRole*.[10]

This approach is somewhat similar to how most state-of-the-art Semantic Role Labellers work: they first run a binary classifier to filter out candidates which are definitely not taking any roles in the verb frame. Then they pick a label for the remaining ones. This helps both in terms of efficiency and recall. However, note that errors made in the first stage cannot be recovered in the second stage. To avoid such situations we do not filter out any candidates; at any time we use all candidates provided by the pruning heuristic. Hence in our case the sole purpose of our *hasRole* predicate is to improve accuracy.

5.3.1.1 Local formulae

Our model contains a set of very effective local formulae inspired by previous work [Gildea and Jurafsky, 2002]. For example, we use a predicate

$$headword : Int \times Word \tag{5.9}$$

[9]Note that we are using typed predicates both for efficiency and to help the reader to better understand the semantics of a predicate by looking at its signature.

[10]One reason could be the following: during training the formulae for the *hasRole* predicate has to predict what a semantic argument is without the help of the *role* formulae. Hence we will make more errors for the *hasRole* predicate and are forced to improve the weights for the *hasRole* formulae more often. This might lead to better decisions at test time.

that maps each candidate constituent to its head word and a formula

$$headword(i, +w) \Rightarrow role(i, +r) \tag{5.10}$$

that allows us to learn correspondences between the label of a constituent and the head word of this constituent. Note that we are using the notation introduced in section 3.4. Here a formula with variables marked by the "+" symbol represents a set of formulae with distinct weights, one for each binding of the marked variables.

We also use syntax-based features such as the path [Xue and Palmer, 2004] from the constituent to be labelled to the predicate verb in the parse tree. In this case we use a predicate

$$path : Int \times Path \tag{5.11}$$

that maps candidate phrases to their path. Here Path consists of constants such as "$NP \uparrow S \downarrow VP \downarrow VV$",[11] which indicates that in order to get from the candidate constituent to the predicate verb we first go up, passing a NP node, then go down, passing an S node, and so on. We incorporate this information into our MLN by adding the formula

$$path(i, +p) \Rightarrow role(i, +r) \tag{5.12}$$

to our model.

The complete set of formulae is shown in the tables 5.1, 5.2 and 5.3. Table 5.1 lists all local formulae that consider exactly one property of the candidate constituent or target verb. Note that most formulae are used both for the *role* and *hasRole* hidden predicates. Table 5.2 shows all formulae that conjoin two properties of the candidate/target verb. Finally, table 5.3 lists all remaining formulae.

5.3.1.2 Global formulae

The local formulae by themselves are already quite effective. However, as previous work has shown [Toutanova et al., 2005, Punyakanok et al., 2004, 2005], significant improvements can be achieved by considering some more global properties of a role labelling.

One core constraint ensures that if we mark a candidate constituent to be a semantic argument (as denoted by the *hasRole* predicate) there has to be one semantic label l for this candidate:

$$hasRole(i) \Rightarrow \exists l.role(i, r) \tag{5.13}$$

[11] This is the notation used by Xue and Palmer [2004].

T	Description	*role*	*hasRole*
path	syntactic path between constituent and predicate	x	x
length	length of constituent	x	x
label	label of constituent	x	x
voice	voice of predicate verb	x	
subcat	subcategorisation frame	x	
frame	position of constituent in frame	x	x
short_frame	simplified position in frame		x
distance	distance in tokens		x
chunk_dist	distance in chunks	x	x
position	left or right side of verb	x	x
head	head word of constituent	x	x
first_word	first word of constituent	x	x
last_word	last word of constituent	x	x
first_pos	first POS of constituent	x	x
last_pos	last POS of constituent	x	x
parent_label	label of parent constituent	x	x
parent_head	head of parent constituent	x	x
pp_rightmost_head	head word of the rightmost PP	x	x
pp_rightmost_pos	pos of the rightmost PP	x	x
left_label	label of right neighbour	x	x
right_label	label of left neighbour	x	x

Table 5.1: Local formulae of the form $T(i,+t) \Rightarrow role(i,+r)$ and $T(i,+t) \Rightarrow hasRole(i)$.

T_1	T_2	role	hasRole
path	*predicate*	x	x
label	*predicate*	x	x
position	*voice*	x	x
position	*predicate*	x	x
head	*predicate*	x	x
frame	*predicate*	x	x

Table 5.2: Local Formulae of type $T_1(i,+t_1) \wedge T_2(i,+t_2) \Rightarrow role(i,+r)$ and $T_1(i,+t_1) \wedge T_2(i,+t_2) \Rightarrow hasRole(i)$.

Formula
$role(i,+r)$
$hasRole(i)$
$position(i,+x) \wedge predicate(i,+p) \wedge voice(i,+v) \Rightarrow role(i,+r)$
$position(i,+x) \wedge predicate(i,+p) \wedge voice(i,+v) \Rightarrow hasRole(i)$

Table 5.3: Additional local formulae

As we mentioned before, this formula only works in one direction, in contrast to the equivalence formula we presented in chapter 3.

One obvious attribute of a labelling is the fact that two overlapping constituents cannot be both be labelled. Assuming that we have a symmetric and irreflexive *overlap* predicate defined over all candidates that overlap, we can state this property quite easily using the deterministic formula:

$$overlap(i_1,i_2) \wedge hasRole(i_1) \Rightarrow \neg hasRole(i_2) \tag{5.14}$$

Because the *hasRole* atoms do not need to be consistent with the *role* atoms we are also required to add another formula that explicitly forbids overlapping arguments represented by the *role* predicate:

$$overlap(i_1,i_2) \wedge role(i_1,r_1) \Rightarrow \neg role(i_2,r_2) \tag{5.15}$$

Another powerful constraint is based on the observation that a verb cannot have more than one argument of each type. However, it can have multiple modifiers of the same type. For example, a verb must not have more than one agent. Yet, it can have two or more temporal modifiers. To capture this constraint we first introduce a

Formula
$overlap(i_1,i_2) \wedge hasRole(i_1) \Rightarrow \neg hasRole(i_2)$
$overlap(i_1,i_2) \wedge role(i_1,l_1) \Rightarrow \neg role(i_2,l_2)$
$argument(l) \wedge role(i_1,l) \wedge i_1 \neq i_2 \Rightarrow \neg role(i_2,l)$
$role(i,l_1) \wedge l_1 \neq l_2 \Rightarrow \neg role(i,l_2)$
$hasRole(i) \Rightarrow \exists l.role(i,l)$
$carg(l_c,l) \wedge role(i,l_c) \Rightarrow \exists j.role(j,l)$
$rarg(l_r,l) \wedge role(i,l_r) \Rightarrow \exists j.role(j,l)$

Table 5.4: Global formulae.

predicate

$$argument : Label \tag{5.16}$$

that is used to denote labels the refer to semantic arguments. Then we add the Markov Logic rule:

$$argument(l) \wedge role(i_1,l) \wedge i_1 \neq i_2 \Rightarrow \neg role(i_2,l) \tag{5.17}$$

Note that all global formulae are deterministic. We have trained models that contained soft constraints, such as the formula from our previous examples which penalizes solutions with multiple AM-TMP labels. Adding these constraints did not create a problem for Cutting Plane Inference. However, no soft constraint helped to improve accuracy.

We summarize all global formulae in table 5.4. The first six formulae have already been described, either in this section or in chapter 3. The last two formulae need some more explanation. Sometimes semantic arguments are not contiguous; that is, they can span two or more disconnected phrases. For example, in

> [$_{\text{A1}}$One troubling aspect of DEC's results], analysts said, [$_{\text{C-A1}}$was its performance in Europe].

"One troubling aspect of DEC's results ... was its performance in Europe" is itself a single semantic argument of the verb "to say".

In our training data the constituent that continues a semantic argument of type *X* is labelled as C-*X*. For instance, in example sentence above the constituent labelled with C-A1 continues the A1 argument. To avoid cases where there is a constituent labelled as C-*X* without a corresponding constituent labelled as *X* we add the following formula:

$$carg(l_c,l) \wedge role(i,l_c) \Rightarrow \exists j.role(j,l) \tag{5.18}$$

Here *carg*/2 holds between the label pairs (C-*X*,*X*) for all argument labels *X*.

A similar situation arises with referential arguments, such as "that" and "The deregulation" in the following example:[12]

> [$_{\text{A1}}$The deregulation] of railroad and trucking companies [$_{\text{R-A1}}$that] began in the 1980 enabled shippers to bargain for transportation.

Here "that" is an A1 argument of "began" but is actually referencing the A1 argument "The deregulation", and is hence labelled with R-A1. The final formula in table 5.4,

$$rarg(l_r, l) \wedge role(i, l_r) \Rightarrow \exists j.role(j, l) \tag{5.19}$$

ensures that whenever we label a candidate with R-*X* there must be a constituent that plays the role *X*. Here *rarg*/2 holds between the label pairs (R-*X*,*X*) for all argument labels *X*.

5.3.2 Datasets

For training we use the CoNLL 2005 dataset [Carreras and Marquez, 2005]. It consists of about 40,000 sentences, taken from the Wall Street Journal corpus. In total this amounts to about 90,000 verbs with labelled arguments and modifiers.

For testing, the CoNLL shared task provides two test sets, one drawn from the PTB Wall Street Journal (WSJ) corpus, the other from the Brown Corpus. The latter test set was provided to evaluate how well a system performs on one domain when trained on a slightly different one. As this is not a focus of this thesis we only use the WSJ test set.

In some cases we also restrict ourselves to the first 100 verb frames of the WSJ corpus. The reason for not using more instances were memory fragmentation problems that appeared when we were using ILP or MaxWalkSAT as the stand-alone solver and had to instantiate a complete ground network for every problem instance. CPI, on the other hand, had only minimal memory requirements and memory fragmentation was not a problem.

5.3.3 Learning

We learn the weights using the Online Learner MIRA [Crammer and Singer, 2003] that we presented in chapter 2 on page 13. The main reason for choosing Online

[12] Note that this example is taken from the official CoNLL 2005 shared task website.

Learning to train our weights for Semantic Role Labelling is the strong performance it has shown on several occasions [Collins, 2002, McDonald et al., 2005], and in the context of Markov Logic [Singla and Domingos, 2005]. Best results were achieved when we trained for 4 epochs and used the averaged weight vector. For inference during training we use CPI with ILP.

Note that in section 5.3.4 we run inference not only with ILP (and ILP used as base solver for CPI), but also with MaxWalkSAT. Hence there is a discrepancy between training and inference that has to be taken into consideration when comparing the accuracy of ILP and MaxWalkSAT. In other words, we may see higher accuracies (both in terms of score and F1 measure) for MaxWalkSAT if we had trained the model using MaxWalkSAT inside the Online Learner loop. However, the main concern of this thesis is the relative impact of Cutting Plane Inference, not a comparison between ILP and MaxWalkSAT, and hence we do not investigate this issue here.

5.3.4 Experiments

In the following we will first present and interpret our results averaged over the test set with 100 frames. Then we will empirically analyse how the runtime behaviour of CPI depends on the the number of candidate constituents.

5.3.4.1 Averaged Results

Tables 5.5 and 5.6 show the averaged results over the first 100 sentences of the WSJ test set for several systems in terms of speed and accuracy (table 5.5) and memory efficiency (table 5.6). Based on these results we can indeed conclude that CPI significantly improves the efficiency of a base solver, while remaining at least as accurate. In particular, we can make the following observations.

We first notice that the stand-alone ILP and MWS systems are accurate but slow.[13] The ILP system takes about 5.5 seconds on average to solve each instance. Since ILP is exact, the score of 0.834 is the optimal linear score (cf. section 5.1.2). When fully instantiated, the ground network has an average size of about 130,000 ground formulae, leading to a final memory usage of about 1.5 gigabyte.[14] The fastest stand-alone MWS system in table 5.5, namely MWS-10-1k with 10 runs of 1,000 flips, cannot reach solutions with maximal score. MWS-10-10k uses 10,000 instead of 1,000 flips per

[13] Here the term "accurate" refers to the accuracy in terms of model score, not F1 measure.

[14] Again, recall that this memory usage is not the average usage per problem but the size of the Java virtual machine after all 100 instances are processed.

System	Iterations	Time	Time(Solver)	Lin. Score	F1
MWS-10-1k	1	5.7	0.70	0.762	0.74
MWS-10-10k	1	6.5	1.5	0.832	0.79
ILP	1	4.6	0.71	0.834	0.79
CPI-MWS-10-1k	2.24	0.11	0.058	0.829	0.79
CPI-MWS-10-10k	2.24	0.14	0.093	0.834	0.79
CPI-ILP	2.25	0.065	0.024	0.834	0.79

Table 5.5: Speed and accuracy for different systems in terms of iterations (calls to the propositional optimizer), total time for each instance (in seconds), time spent in the propositional solver (in seconds), linear score of the system output and its F1 measure. All results are averaged over the first 100 examples in WSJ test set.

System	Formulae	Memory
MWS-10-1k	1.3×10^5	2.1
MWS-10-10k	1.3×10^5	2.0
ILP	1.3×10^5	1.5
CPI-MWS-10-1k	8.6	0.44
CPI-MWS-10-10k	8.6	0.41
CPI-ILP	8.6	0.44

Table 5.6: Memory Efficiency of different systems in terms of instantiated global ground formulae (averaged over the first 100 instances of the WSJ test set) and memory usage (in gigabyte) of the Java virtual machine after all 100 instances have been processed.

run, and this leads to optimal scores after an average of 6.5 seconds solving time and using about 2 gigabytes of RAM. Due to the relatively long solving times, both the ILP and MWS system would be unsuitable for Online Learning methods[15] and real-world application such as Dialogue Systems that cannot afford to make users wait 5 or more seconds for an answer.

We also observe that for stand-alone systems that initialisation costs dominate the actual solving time within the propositional algorithm. Both the ILP and the MWS solver only spend a small fraction of their time in the actual propositional MAP algo-

[15] In the WSJ training set there are about 90,000 instances to label, so with 6s per instance this would require about 6 days per training epoch. In our case we used 4 epochs—this would require more than 3 weeks training time.

rithm. The most time is thus consumed while generating the ground network, passing information to the MAP algorithm and translating between the propositional and relational representation.

Most importantly, we notice that CPI dramatically improves the efficiency of ILP and MWS. When we compare each stand-alone system with the corresponding CPI version we note that in all cases the total solving time is reduced by about two orders of magnitude. Moreover, memory usage is reduced by at least 70% for each solver.

The cause of this improvement is the dramatic reduction of instantiated ground formulae (see table 5.6). Instead of having to generate and process 130,000 ground formulae once, the CPI systems have to solve networks with about 9 ground formulae twice. Roughly speaking, the reason for this reduction is the local model. Because the initial solutions are "almost perfect", only a few formulae need to be instantiated. Once these formulae are added and the problem is resolved, the next solution already satisfies all constraints (leading to about 2 iterations on average). [16]

Also note how the CPI systems spend more of their time in the propositional MAP algorithm, instead of "wasting time" in initialisation of ground formulae that are not strictly necessary to solve the problem. However, by comparing the "time in solver" columns of base solvers and CPI systems we can also see that CPI not only reduces the overhead associated with generating large networks, it also reduces the workload for the propositional base solver.

Interestingly, CPI can also improve the accuracy of a weaker base solver. If we compare the scores of MWS-10-1k and CPI-MWS-10-1k it becomes clear that CPI increases the score accuracy significantly (and, since we use a model that assigns high probability to good solutions, the F1 measure). Intuitively this makes sense: accurately solving a problem with 8 ground formulae should be easier than solving a problem with 130,000 ground formulae.[17]

Finally, we can observe that the solutions found by CPI-ILP have maximal score (as defined through the results of the ILP system, which is known to be exact). This reconfirms the bound on score error we give in chapter 4 which predicts that CPI with exact base solver leads to optimal solutions.

Note that we also ran CPI-ILP on the full test set to compare our system with the state of art, yielding 0.77 F1 measure. When compared to the entries in the CoNLL

[16] In chapter 6 we will attempt to provide a more detailed description of this kind of behaviour.

[17] In particular if we consider that any formulae in the small network is also part of the large network and hence the small network cannot be "structurally" less complex than the large one.

shared task that only use the output of one parser our system would come out first [Carreras and Marquez, 2005]. However, the focus of these experiments is not to compare models for Semantic Role Labelling but inference methods for Markov Logic.

5.3.4.2 Scaling

As mentioned previously, the runtime of CPI is difficult to predict analytically and we cannot make claims such as "CPI solves SRL in $O\left(n^3\right)$ time". Hence one could argue that while CPI works well for small problems its runtime might increase drastically for larger problems. This would surely render CPI less useful. It is therefore helpful to evaluate how CPI performs with varying problem size n. If we see a trend (such as a linear dependency on n) we will still not have any guaranteed runtime bound, but our confidence in CPI will increase. In the following we will therefore present the observations we made when applying ILP and CPI-ILP to problems with different numbers of candidate constituents.[18]

Figure 5.1 shows the runtime for CPI-ILP for different numbers of candidate constituents when applied to the full WSJ test set of the CoNLL 2005 shared task.[19] The times are averaged over all problems with the corresponding size. Note that for larger sizes we have less data and thus the graph gets less smooth on the right end of the graph. For this problem we observe that CPI-ILP seems to scale linearly with the size of the problem.

In order to evaluate runtime behaviour of using ILP alone, as shown in figure 5.2, we again needed to restrict the set of problems to the first 100 instances of the test set. This was due to the excessive memory requirements when using the complete ILP formulation. Note that in this setting the range of runtimes is significantly larger. While CPI-ILP operates in the range of 0ms to 60ms, pure ILP needs 0ms to 25,000ms even though the largest problem size is only half as large as the largest problem size in the CPI-ILP experiments. Most importantly, we note that the runtime of the standalone ILP system is not linear; it rises faster for larger problems. This shows that CPI

[18]Note that in this section we will focus our attention to ILP since MWS is effectively optimal for this problem and therefore behaves very similar to ILP.

[19]Note that the number of candidates depends on the sentence length only in an indirect manner – the main factors that determines the number of candidate constituents are the size of the parse tree and the position of the predicate verb in this tree.

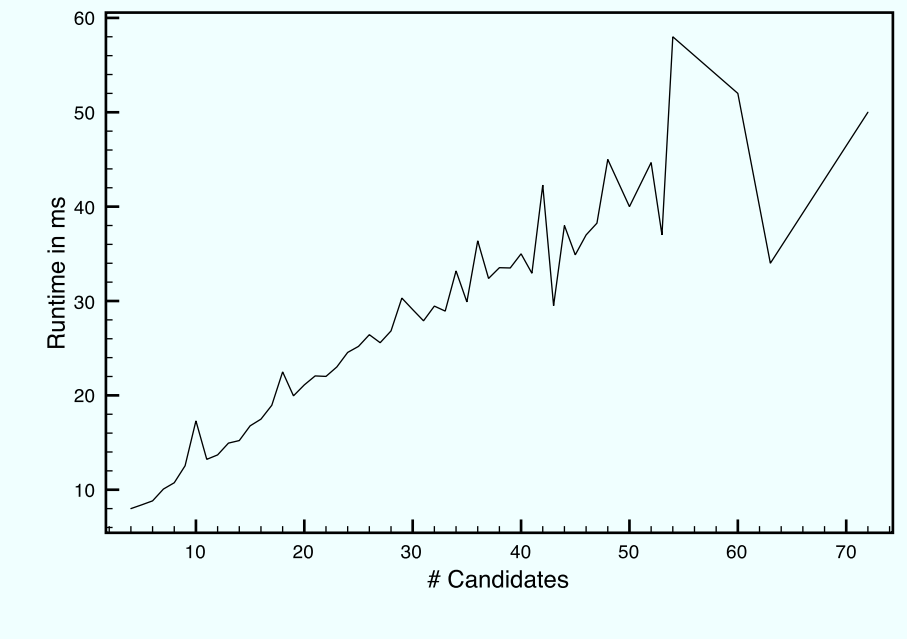


Figure 5.1: The average runtime for SRL problems with different number of candidate constituents, using CPI-ILP.

not only improves the average speed of MAP inference, it can also lead to a better behaviour when problem sizes increase.

Interestingly, the number of CPI iterations remains low even for larger problems. In terms of the number of iterations needed, figure 5.3 shows a quasi-constant behaviour. While rising for smaller numbers of candidates, it remains somewhat constant for problems with 20 to 40 candidates. From there on the graph becomes less smooth, again due to the lack of data. However, it seems unlikely that the number of iterations would become much larger than 3.

5.4 Joint Entity Resolution

Entity Resolution is the task of matching records that refer to the same entity. It has also been referred to as Record Linkage, Object Identification, De-duplication and Merge/Purge, to name only a few [Singla and Domingos, 2006a]. The problem of entity resolution always arises when data from different sources have to be integrated.

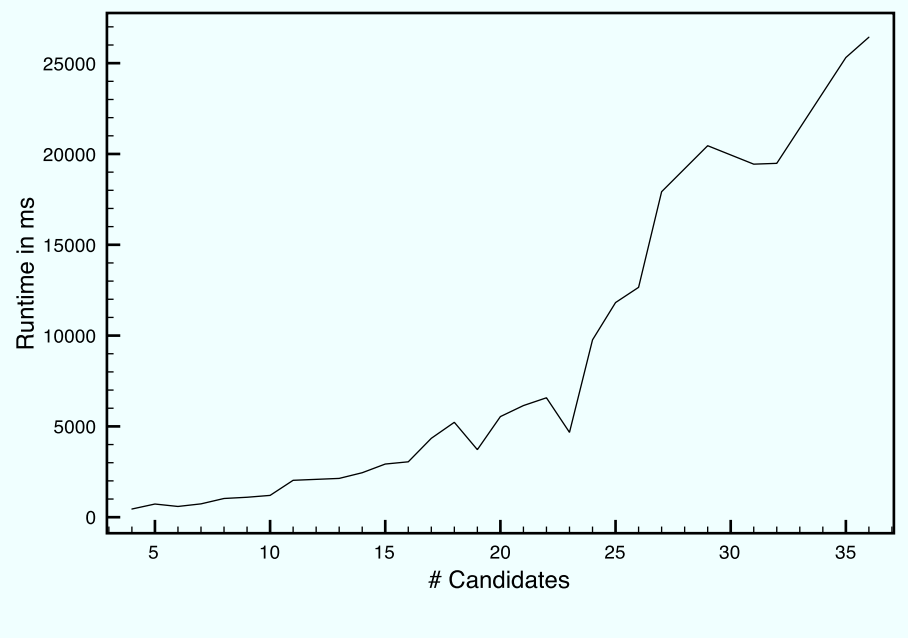


Figure 5.2: The average runtime for SRL problems with different numbers of candidate constituents, using the full ILP formulation.

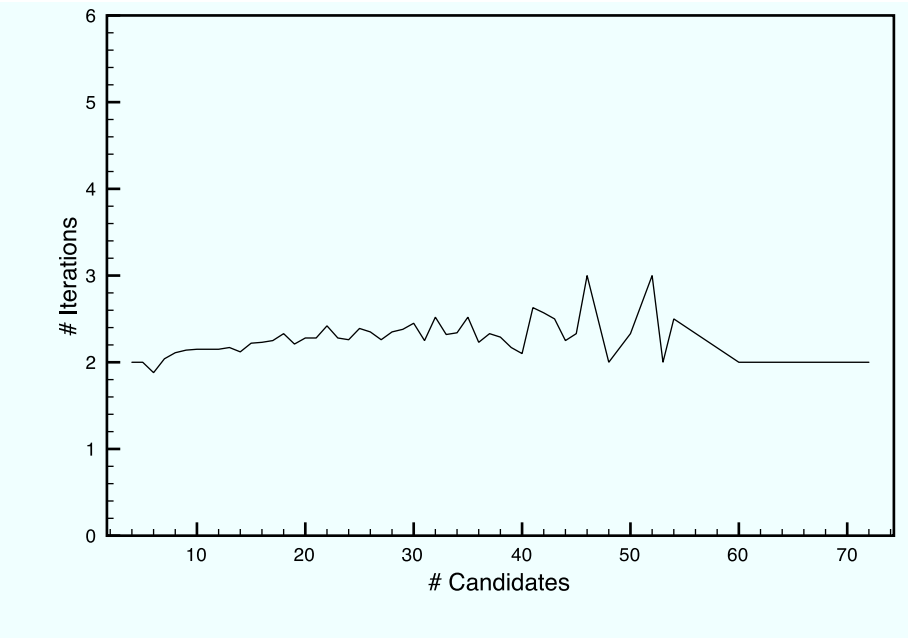


Figure 5.3: The average number of iterations for SRL problems with different number of candidate constituents.

For example, consider two companies that have been merged. In turn their customer databases are merged as well. Since the two companies had an overlapping set of customers, the resulting database contains duplicate records.

Finding duplicate records is not trivial. Consider the task of matching records that refer to persons. On one hand, several records might describe the same person with different (mis)spellings of his/her name, address and other attributes. On the other hand, we might find database records that look very similar yet still refer to different persons.

Note that Entity Resolution is similar in nature to the task of Coreference Resolution [Poon and Domingos, 2008] for Natural Language Processing. Here the goal is to find phrases in natural text that refer to the same entity. These phrases might be names of persons or other entities, possibly misspelled or abbreviated, as well as pronouns.

In this work we look at the process of matching citations in scientific publications. Citations of the same paper can differ substantially in many aspects. For example, paper titles might be misspelled or shortened, and author names might be abbreviated or partly discarded. By matching citations that refer to the same publication we can create citation databases such as Citeseer that help us to tell where a publication has been cited. It also allows us to evaluate the significance of a publication in terms of the number of its citations.

Instead of only searching for citation matches we also want to infer whether author strings refer to the same author, venue strings refer to the same venue and title strings refer to the same title. This has two advantages: first, we can answer questions such as "which papers did author X write" more accurately because we can find publications of X even if her name is misspelled in the corresponding citations; second, it has been shown that the resolution of entities of one type can help the resolution of entities of related types [Singla and Domingos, 2006a, Culotta and McCallum, 2005]. For example, if we know that two author strings match it is more likely that the corresponding citations match, too. This problem is especially suited for Statistical Relational Learning because it requires a joint model over several related decisions.

Here we choose this task for two reasons. First, it has been studied extensively in the Markov Logic literature [Poon and Domingos, 2006, Singla and Domingos, 2006b,a] and Markov Logic Networks and datasets are already available for this task. Second, it differs substantially from the Semantic Role Labelling task because here the size of Ground Markov Networks is significantly larger and the model contains formulae that are both global and nondeterministic. A successful application of CPI in

PID	Author	Title	Venue	VID
0	X. Li	Predicting the stock market	CIKM	10
1	X. Li	Predicting the stock market	Conf on Information Management	20
2	J. Smith	Semi-Definite Programming	CIKM	30
3	Smith, J.	Semi-Definate Programing	Conference on Info Management	40

Table 5.7: Four example citations referring to two different publications; taken from [Culotta and McCallum, 2005].

this domain shows that it suitable for a larger class of problems. It also demonstrates one advantage of Statistical Relational Learning: implementing an algorithm in this framework makes it immediately applicable to a wide range of problems. The only remaining tasks are to design a model and prepare the data.

5.4.1 Example

Before we go on to describe the Markov Logic Network we use for this task, let us first give a small motivating example, taken from [Culotta and McCallum, 2005]. Table 5.7 shows four citations. The goal of citation matching in this case is to determine that the citations with PID 0 and 1 should be merged because they refer to the same publication, and that the same holds for citation 2 and 3. In our joint task we also want to match citation attributes. For example, we need to find out that the venue mentions with VID 10, 20, 30 and 40 all refer to the same venue.

The given example also indicates the potential benefit of a joint approach. Since both author and title are identical for PID 0 and 1, we can predict that 0 and 1 indeed refer to the same publication. In this case we have strong evidence for VID 10 and 20 to match, even though they are very different in lexical terms. By transitivity and the fact that venue mention 20 is similar to mention 40 we can induce mentions 30 and 40 to match, too. This in turn gives us confidence in matching citation 2 and 3, which could be considered borderline because of different author, title and venue strings.

5.4.2 A Markov Logic Network for Citation Matching

We will now present a Markov Logic Network that models this problem. The network is taken from existing work [Singla and Domingos, 2006b].

We look at four types of constant: citations, venues, authors and titles. Based on

these types we define the following predicates:

$$sameCitation : Citation \times Citation \tag{5.20}$$
$$sameAuthor : Author \times Author$$
$$sameTitle : Title \times Title$$
$$sameVenue : Venue \times Venue$$

Each predicate holds for pairs of constants that refer to the same citation, author, title or venue, respectively.

We associate citations to their corresponding attributes using the following predicates:

$$author : Citation \times Author \tag{5.21}$$
$$venue : Citation \times Author$$
$$title : Citation \times Title$$

These predicates hold for a citation-attribute pair if and only if the citation has the corresponding attribute.

In addition, the model contains a set of predicates that indicate the similarity of citation attributes in terms of their literal content; that is, the similarity between title strings, author names and venue names. In order to denote the degree of similarity between attributes we use predicates such as

$$similarTitle20 : Title \times Title \tag{5.22}$$

that hold for pairs of constants if their TF-IDF distance is within a certain range. For example, *similarTitle20* holds for pairs of title constants whose corresponding titles have TF-IDF distance between 20% and 40%. The predicates for different similarity levels and attributes are defined accordingly.

Note that the *author*, *venue* and *title* predicate as well as all similarity predicates are fully *observed*. That is, we can extract them directly from the given database; no inference is needed to determine them. In contrast, the *sameCitation*, *sameAuthor*, *sameTitle* and *sameVenue* predicates are *hidden* in the sense that they are only known during training.

5.4.2.1 Local formulae

The Markov Logic Network of Singla and Domingos [2006b] contains a set of *local* formulae. Again, we refer to these formulae as being local because when grounded

they contain only one hidden atom. One such local formula is

$$similarTitle20(t_1,t_2) \Rightarrow sameTitle(t_1,t_2) \tag{5.23}$$

which can be read as follows: if a pair of citation titles has a similarity between 20% and 40% the corresponding citations have to refer to the same publication. If this is not the case, the weight of this formula is taken off the score for the corresponding possible world.

The model also contains formulae that directly associate citation matches with attribute similarities, such as

$$\begin{aligned} title(c_1,t_1) \wedge title(c_2,t_2) \wedge similarTitle20(t_1,t_2) \\ \Rightarrow sameCitation(c_1,c_2) \end{aligned} \tag{5.24}$$

These formulae can help Cutting Plane Inference to be more efficient. For example, in the data the titles of most citation pairs are not very similar. Because the formula that tests for low title similarity has a very negative weight, most citation pairs are therefore too expensive to match, at least if we only consider local formulae. This results in a relatively sparse matching graph as solution to the first MAP problem during CPI. We will see in chapter 6 that this guarantees that partial networks generated during CPI will remain small.

Because the partial networks remain sparse, we can also observe "island effects": the resulting final ground network will consist of several disconnected graphs that CPI effectively processes in parallel. This guarantees a low number of iterations. Section 6.6 describes this effect in more detail.

5.4.2.2 Global formulae

Along with the local formulae, the MLN of Singla and Domingos [2006b] also contains a set of global formulae. First, we associate attribute matches with citation matches:

$$\begin{aligned} title(c_1,t_1) \wedge title(c_2,t_2) \wedge sameTitle(t_1,t_2) \\ \Rightarrow sameCitation(c_1,c_2) \end{aligned} \tag{5.25}$$

This formula, along with the corresponding versions for author and venue attributes, is not local: when we ground the formula it contains two hidden ground atoms, one for the *sameTitle* predicate and one for the *sameCitation* predicate. The model also

contains the reversed version of formula 5.25:

$$title(c_1,t_1) \wedge title(c_1,t_2) \wedge sameCitation(c_1,c_2) \Rightarrow sameTitle(t_1,t_2) \tag{5.26}$$

Again, we also include versions of this formula for the author and venue attributes.

Finally, the network contains a formula that ensures transitivity of the *sameCitation* predicate in order to forbid inconsistent citation clusters:

$$sameCitation(c_1,c_2) \wedge sameCitation(c_2,c_3) \Rightarrow sameCitation(c_1,c_3) \tag{5.27}$$

This formula is deterministic and imposes a difficult problem for many inference methods [Poon and Domingos, 2006]. We follow Singla and Domingos [2006b] and do not include transitivity formulae for venues, authors and titles.[20]

5.4.3 Datasets

In our experiments we used a cleaned version [Singla and Domingos, 2005] of the Cora Database [Bilenko and Mooney, 2003], containing about 1200 citations of computer science articles. In total these citations refer to about 120 unique publications.

We also wanted to evaluate how CPI performs with increasing problem size. To this end we follow Singla and Domingos [2006b] and use a subset of the Bibserv.org corpus as additional test set. The Bibserv.org corpus consists of roughly half a million pre-segmented citations, donated by Citeseer, DBLP and others. The subset we are looking at contains about 20,000 citations. Bibserv.org does not contain any information about matching citations and hence we cannot evaluate F1 score. However, it can still serve to measure the efficiency of CPI.

The benefits of choosing an additional test set are the following. If we were to use the Cora dataset to evaluate CPI for, say, 500 citations we can only train a model on the remaining 700 citations. In contrast, by using the Bibserv.org corpus we can exploit the complete Cora dataset for training. Moreover, ideally we not only look at one problem with 500 citations but several such problems, randomly drawn from a dataset [Singla and Domingos, 2005]. When we only use the Cora dataset for this we need to learn several models: one for each sample of the data, using the remaining data as training

[20] Note that the main purpose of the Markov Logic Network of Singla and Domingos [2006b] is to evaluate MAP inference. In this light the restriction to a single transitivity clause only for citations makes sense: while further transitivity clauses for venues, authors and titles could improve the Entity Resolution accuracy, they do not significantly change the complexity of the problem.

set. Finally, by applying CPI to a different dataset we can get a better picture of how the behaviour of CPI is affected by the data it has to process.

5.4.4 Learning

Following Singla and Domingos [2005], we tested and trained using a 10-fold leave-one-out procedure while ensuring that folds do not contain split citation clusters. Each fold contains roughly 120 records. In order to match the citations, titles, authors and venues of one fold, we train a model by optimizing the Pseudo-Likelihood (compare section 2.4.1) with respect to the remaining 9 folds. In our experiments that use the Bibserv.org corpus we use a model trained on the complete Cora dataset, again using the Pseudo-Likelihood as objective.

5.4.5 Experiments

In the following we will first present and interpret our results averaged over the 10 Cora folds. Then we will empirically analyse how the runtime behaviour of CPI depends on the number of citations by using the Bibserv.org corpus. We will also use this corpus to evaluate the impact of Ordered CPI on runtime.

5.4.5.1 Averaged Results

Tables 5.8 and 5.9 show our results for Entity Resolution. Again we can observe that CPI makes accurate inference more efficient. While ILP on its own is impractical (the ILP solver crashed due to the size of the problem) it becomes usable and efficient through the use of CPI. In particular, we make the following observations.

Considering the results in table 5.8 we first observe that MaxWalkSAT is inaccurate for this model. For all presented MaxWalkSAT systems the score is far from optimal, regardless how many flips or number of restarts we use. When compared to the optimal solutions returned by CPI-ILP,[21] the MWS solutions have significantly lower score. This leads to a dramatic decrease in F1 measure.

The poor performance of MWS both in terms of score accuracy and F1 measure may be surprising. One possible explanation is the following. Consider a cluster A,B,C and D of citations all referring to the same publications. Assume currently (A,B), (B,C) and (C,D) are matched. This solution violates two ground instances of the transitivity

[21] Recall that we have shown CPI-ILP to be exact with theorem 4.1.

System	Iterations	Time	Time(Solver)	Lin. Score	F1
MWS-1-100k	1	2.2	0.30	-973	0.20
MWS-10-100k	1	4.3	2.5	-890	0.21
MWS-1-1000k	1	2.9	1.1	-970	0.22
ILP	1	60	58	N/A	N/A
CPI-MWS-10-100k	20	2.7	2.4	-1357(364)	0.27
CPI-ILP	6.7	1.73	0.95	3030	0.72

Table 5.8: Speed and accuracy for different systems in terms of iterations (calls to the propositional optimiser), total time for each instance (in minutes), time spent in the propositional solver (in minutes), linear score of the system output and its F1 measure. For the score of CPI-MWS-10-100k the number in brackets is the number of hard constraint violations. All results are averaged over the 10 Cora folds.

System	Formulae	Memory
MWS-1-100k	2.0×10^6	1.8
MWS-10-100k	2.0×10^6	1.9
MWS-1-1000k	2.0×10^6	1.7
ILP	2.0×10^6	2.2
CPI-MWS-10-100k	7.3×10^4	0.23
CPI-ILP	1.9×10^4	0.13

Table 5.9: Memory efficiency of different systems in terms of instantiated global ground formulae and memory usage (in gigabyte) of the Java virtual machine, both averaged over 10 Cora folds.

formulae. If MWS picks one of these (e.g., the one that implies (A,C)) and performs a greedy step, it would remove (B,C) since this satisfies both ground formulae. If MWS performs a random step there is only a 33% chance that the edge (A,C) is added.[22] Hence the overall chance of adding matches is low. This leads to sparse solutions. In fact, for MWS-10-100k the final precision in terms of citation matches is 0.86 while recall is only 0.15, indicating that we predict far too few matches.

Notice that Singla and Domingos [2005] successfully ran MaxWalkSAT for this task. However, in their MLN the transitivity clause is missing and hence the behaviour described above is not observed. Singla and Domingos [2006a] do run MWS with transitivity formulae, yet with discriminatively trained weights. These may help to improve the predictions of the local formulae (indicated by the strong performance they observe for MLNs without transitivity clause) and lead to more dense solutions in each MWS step. This in turn would make the scenario we described above less likely.

In table 5.8 we notice that for ILP no scores are available. When applied to this task, the ILP solver ran for about an hour only to finally crash due to the excessive memory requirements that arise when solving problems with about 2 million constraints. Hence for this problem ILP is effectively intractable.[23]

Here CPI comes into play. In tables 5.8 and 5.9 we see that with CPI exact inference for this problem becomes feasible. Moreover, we observe that CPI-ILP is even faster than MWS, an approximate method we showed to perform poorly on this task. We also notice a dramatic improvement in terms of memory usage when compared to the MWS systems. While propositional MAP inference required at least 1.7Gb of RAM, CPI-ILP only needs about 130Mb, reducing memory usage by an order of magnitude. It should be clear that this improvement is related to the fact that CPI reduces the number of ground formulae by two orders of magnitude, from two million to about 20,000. Based on this observation we can conclude CPI not only reduces grounding and initialization times, it can also significantly reduce the time the propositional base solver has to spend in order to find a solution.

Interestingly, CPI does not help MWS to perform better. First of all, CPI with MWS did not terminate in reasonable time. Hence in our experiments we forced CPI to stop after 20 iterations. This led to inference times of about 2.7 minutes, comparable to the solving times of the stand-alone MWS system. This yields results which not

[22] In the random step any atom of the clause is picked with equal probability.

[23] Note that it may be possible to solve these problem instances using a commercial solver. However, in any case optimizing an ILP problem with 2 million constraints will be difficult.

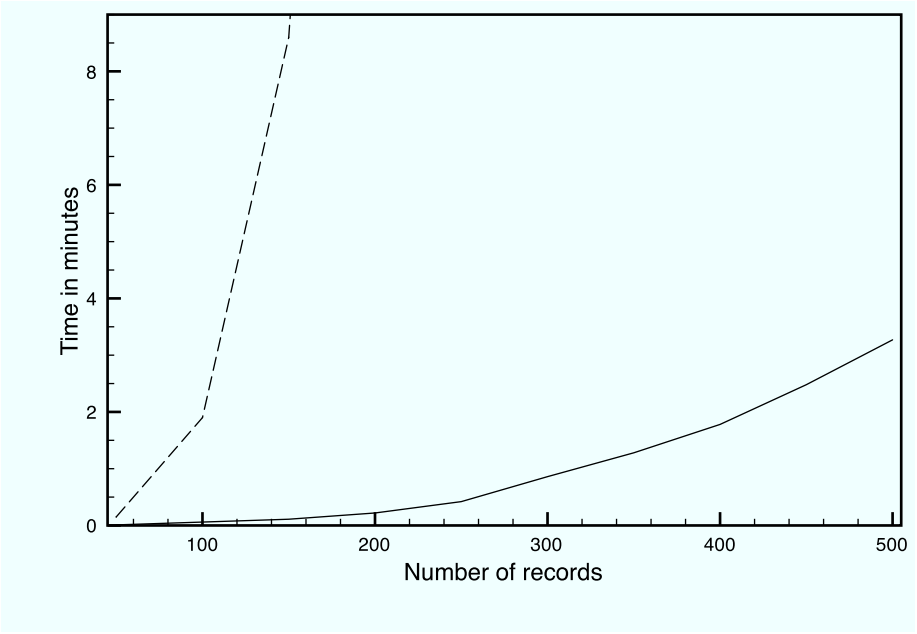


Figure 5.4: Runtime for different number of records. The solid line denotes the runtime of CPI-ILP, the dashed line the runtime of MWS-10-100k.

only have low (soft formula-based) scores and F1 measures, they also violate hard constraints. Hence CPI with MWS is not a practical solution for this MLN. We believe that this finding is related to the fact that MWS performs very poorly by itself. While subparts of the MLN model do well in predicting structure that is consistent with the remaining parts of the MLN,[24] the weak base solver cannot exploit this because it cannot find optimal or near-optimal solutions for these sub-models.

Note that an F1 score of around 72% (for all types of entities) is relatively weak result when compared to about 90% (for citations) and 84% (for venues) of Culotta and McCallum [2005], who used a slightly larger version of Cora. We believe that this is due to the Pseudo-Likelihood (PL) training regime we used in our experiments: Singla and Domingos [2005] showed a dramatic increase in performance (from 0.72 to 0.97 for the area under the precision/recall curve) when switching from PL to discriminative training.

5.4.5.2 Scaling

Figure 5.4 shows the average runtime for Bibserv.org subsets with different numbers of records: from 50 to 500, in steps of 50 records. For each size we use 5 subsets drawn from the 20,000 records mentioned in section 5.4.3. These subsets are identical

[24] This is what happens in the case of CPI-ILP.

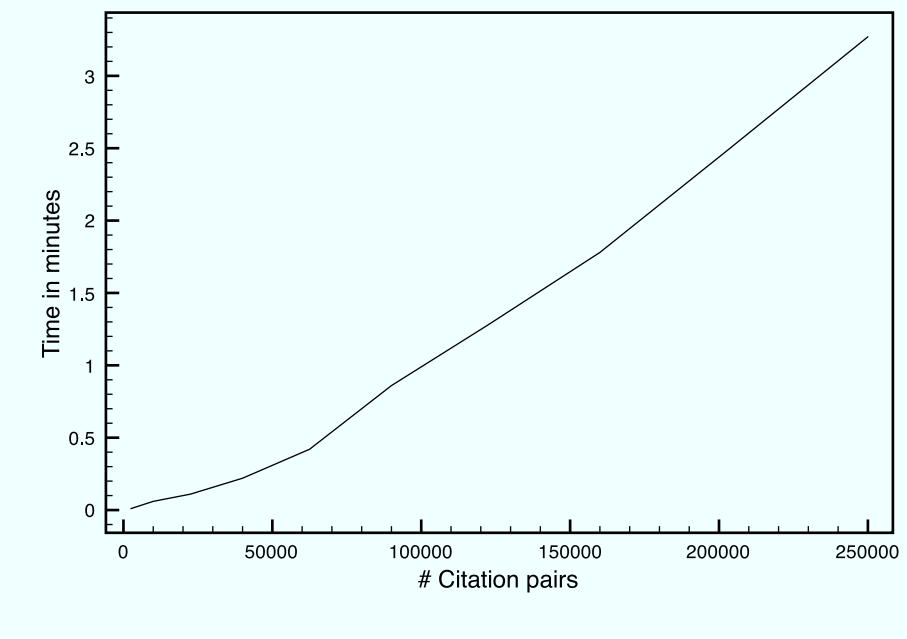


Figure 5.5: Runtime of CPI-ILP for different subsets of Bibserv.org, averaged over 5 instances for each number of citation pairs.

to those used by Singla and Domingos [2006b]. We contrast the runtime of the two most accurate systems, namely CPI-ILP and MWS-10-100k, for different numbers of records to match.

We notice in figure 5.4 that the CPI system is not only efficient on average, it also scales well with increasing problem size—better than an approximate method with poor accuracy. Figure 5.5 underlines this. It shows the runtime of CPI-ILP with increasing number of citation pairs; this corresponds to the number of decisions to make. Again CPI-ILP seems to scale linearly with the number of variables and thus quadratically with the number of citations (see section 5.3.4.2 for similar results in the case of Semantic Role Labelling). This happens despite the fact that the number of ground formulae in the complete network scales at least cubically with the number of citations (due to the transitivity clause).

Why does the runtime of plain MWS rise significantly faster? One explanation is the following. We already showed that one main bottleneck of inference with MWS is the work outside of the actual algorithmic code. For every problem instance we have to generate the complete ground network. The size of this network in terms of ground formulae scales cubically with the number of records, because the transitivity formula is quantified over three citation variables. Simply grounding this network therefore soon becomes increasingly difficult. In contrast, CPI is significantly faster because it

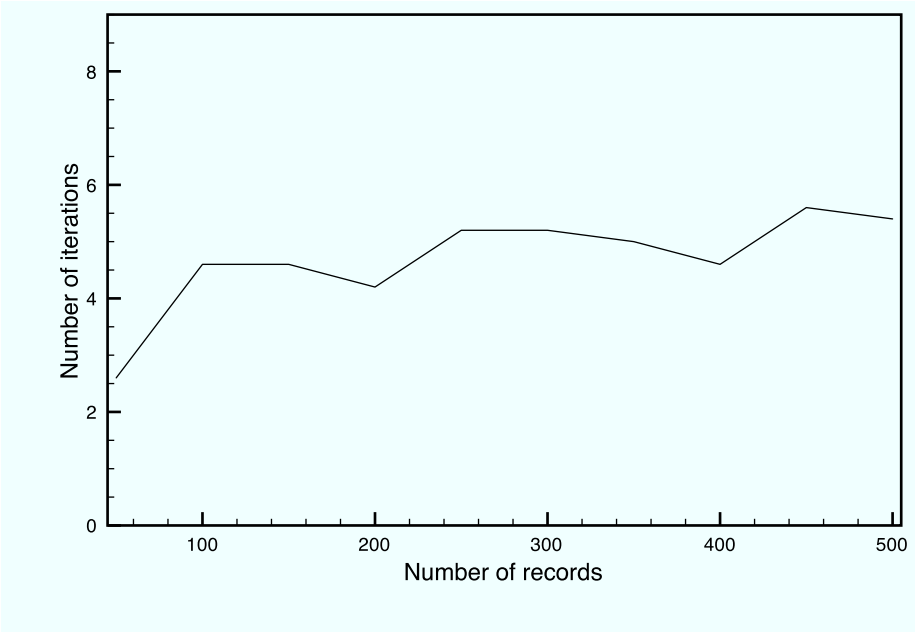


Figure 5.6: Number of CPI-ILP iterations for different number of records

only instantiates a small fraction of the complete network. This fraction is so small that grounding becomes easy and even multiple calls to the base solver are more efficient then one call for the complete network.

In contrast to the overall runtime in figure 5.4, the number of iterations in figure 5.6 stays almost constant over larger datasets. We can possibly explain this by again considering the island effects we discussed in 5.4.2.1 and will investigate further in chapter 6. With more data, especially if drawn from different domains, we will likely see more islands of similar citations, but not necessarily larger islands. We mentioned in 5.4.2.1 how these islands can lead to disconnected islands in the final ground network, and section 6.6 will show that the number of iterations does not depend on the number of such islands.

5.4.5.3 Ordered Cutting Plane Inference

As described in section 4.6 we can change the order in which we search for newly violated ground formulae and still produce the same result. In this section we will evaluate different orders for the formulae of our model. While there are many possible orders to choose from given the set of global formulae in the ER model presented above, we limit ourselves to three cases:

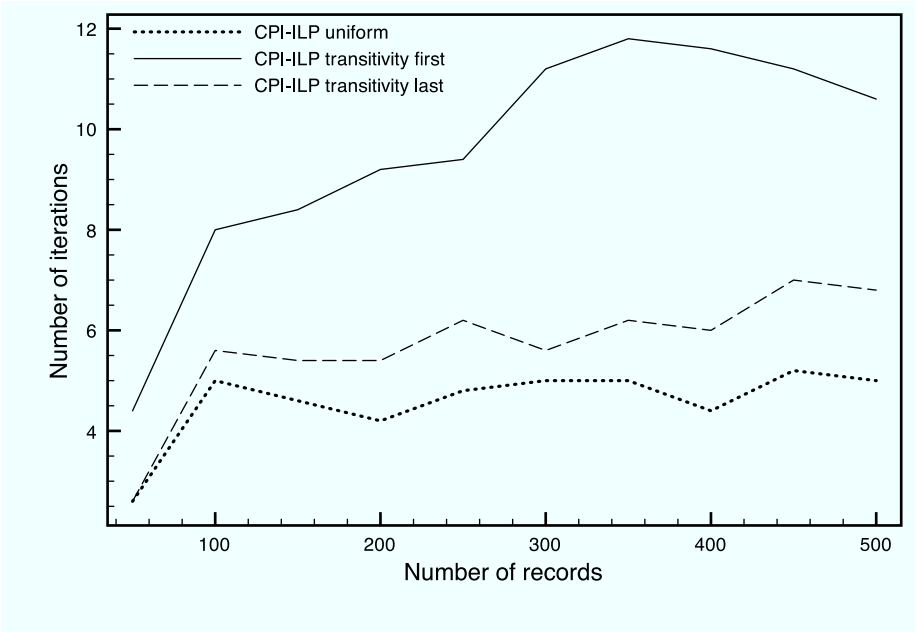


Figure 5.7: Number of iterations for different number of records.

uniform In each iteration we search for violations of all formulae. This corresponds to the plain CPI algorithm we have used so far.

transitivity first We begin by only looking at the transitivity (closure) formula and only if no more violations of this formula can be found are the remaining formulae taken into account. Note that these remaining formulae essentially propagate information between the *sameCitation* and *sameTitle*, *sameVenue* and *sameAuthor* atoms.

transitivity last We simply reverse "transitivity first" and apply the transitivity formula only if no other formulae need to be taken into account anymore.

These cases correspond to a distinction between two types of formulae: those that ensure transitivity and those that ensure consistency between attribute and citation matches. We chose these three types of order because they are easy to illustrate and explain, and because they lead to interesting changes in the behaviour of CPI. Note that in all cases we use the local formulae to generate the first guess.

Figure 5.7 shows the average number of iterations needed for databases of different sizes. The lowest number of iterations is needed when each formula has the same order;

that is, in each iteration we instantiate all violated formulae. This is expected since not taking formulae into account can never reduce the number of iterations: in the worst case we need to instantiate all formulae we have not considered in one step in some additional steps; in the best case all formulae are taken into account implicitly when dealing with the lower order formulae, yielding the same number of iterations as if we had explicitly used them.

Using the transitivity formula first yields the highest number of iterations. In this case the algorithm proceeds as follows. After generating the first guess based on the local formulae, it inspects the solution for violations of the transitivity clause and adds the corresponding instantiations to the model. Next time the solution will be transitive so now we need to make sure that the *sameCitation* atoms are consistent with the *sameAuthor*, *sameTitle* and *sameVenue* atoms, and vice versa. This, however, will cause some more transitivity violations because some newly found *sameAuthor* atoms indicate true *sameCitation* atoms for which we did not have enough evidence before. This process can go on for several iterations.

Slightly more iterations than in the default uniform ordering but less than in the case of "transitivity first" are used when we handle the transitivity clause at the end. We explain this by the fact that attribute matches are a very powerful indicator of citation matches and therefore the transitivity clause is violated less often once attribute and citation matches are consistent.

Figure 5.8 shows the runtime for each of the orderings introduced above. It shows that the number of additional iterations needed for the closure-first ordering does have a strong impact on the runtime from 250 records onward—this coincides with the increased number iterations we observe at problems of this size. However, it is interesting that while needing more iterations the "transitivity-last" order is not slower than the uniform order. In fact it is slightly faster.

Here we see that the number of iterations does not directly correlate with speed. For the "transitivity last" ordering the individual problems at each step are smaller, resulting in faster solving times and less data transfer between the separation and solving components. However, this increase in speed is offset by the increased number of iterations.

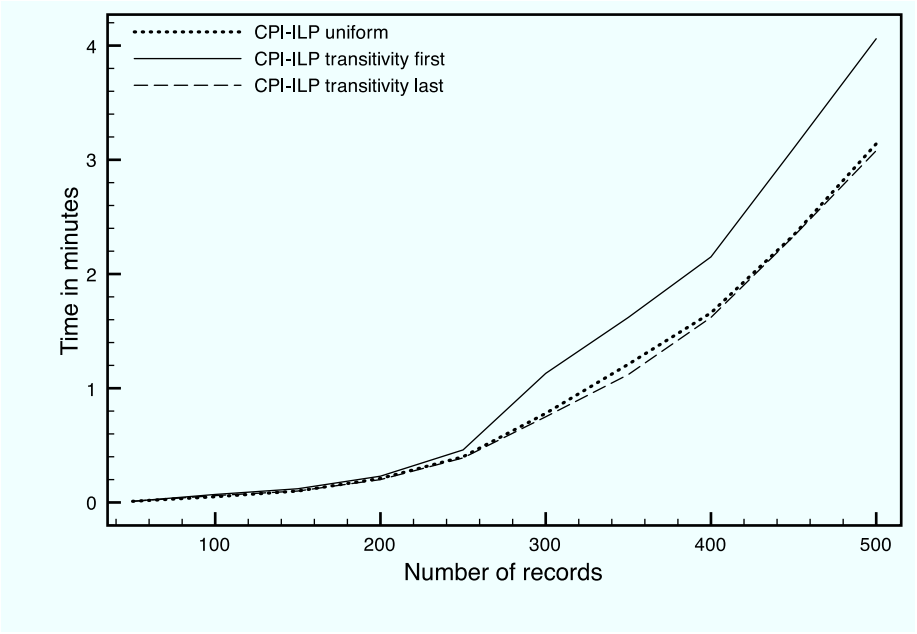


Figure 5.8: Runtime for different number of records

5.5 Conclusion

In the beginning of this chapter we claimed that CPI can significantly improve the efficiency existing propositional methods while remaining at least as accurate. This hypothesis was evaluated using two tasks: Semantic Role Labelling and Entity Resolution (a task very similar to Coreference Resolution). In the case of SRL we present a Markov Logic Network for Semantic Role Labelling, inspired by previous work in the domain, that achieves state-of-the-art accuracy. In the case of Entity Resolution we re-use a Markov Logic Network from previous work.

In both cases we showed that CPI increases the speed of two existing methods by two orders of magnitude while memory usage is reduced by at least 70%.[25] This goes along with a reduction of 4 orders of magnitude (for Semantic Role Labelling) and 2 orders of magnitude (for Entity Resolution) in terms of the number of ground formulae. In particular, CPI allowed us to find an optimal solution in a network with 250,000 nodes and 2 million edges while the base solver failed to even process the problem. This may be somewhat surprising, considering that the need for approximate inference in Markov Logic for all but the smallest domains has been taken for granted [Richardson and Domingos, 2006].

[25]This 70% reduction of memory usage was observed between the ILP and CPI-ILP systems for Semantic Role Labelling. All other memory usage improvements were larger.

In our experiments we observed that a significant amount of time during MAP inference in Markov Logic is spent on grounding a network (and communicating with the base solver). By dramatically reducing the size of the ground networks to be generated, CPI reduces this overhead substantially. In our Semantic Role Labelling experiments this was the major factor for improving runtime. However, in the case of Entity Resolution, CPI not only reduced the grounding overhead, it also made the optimization task for the base solver much easier. Here the time spent in the base solver was reduced from "not done even after an hour" to "less than one minute".

Moreover, we noticed that by using CPI we are also able to find optimal solutions if the base solver is exact (in line with our theoretical findings) and more probable solutions when the base solver is approximate. Only when the base solver was very inaccurate did CPI fail.

We also showed that not only does CPI perform faster on average, it can scale significantly better than the base method. In both applications we show that CPI's efficiency scales linearly with problem size (that is, the number of binary decisions to make). By contrast, the base solver becomes over-proportionally slower with larger problems.

We believe that this scaling behaviour will be very helpful when want to jointly solve several classical NLP problem with Markov Logic. For example, if we wanted formulate a Markov Network for lexicalised constituent parsing [Collins, 1997] we would need $O\left(K \cdot n^5\right)$ features, where n is the number of tokens in the sentence and K represents the number of possible constituent types. If we design a Markov Logic Network that jointly performs this type of parsing as well as Semantic Role Labelling, the Ground Markov Network will therefore contain at least $O\left(K \cdot n^5\right)$ ground formulae. Using a propositional solver will be difficult in this scenario, even if we disregard the increased solving times we can expect when applying the solver to larger problems, because the initialisation costs will dramatically rise with longer sentences. By contrast, CPI can do significantly better, in particular if the parsing model has a certain set of properties we will discuss in chapter 6.

Finally, we have presented evidence which suggests that by changing the order in which formulae are instantiated, the runtime of the base solver for partial networks can be reduced. However, in this case the advantage is offset by a larger number of iterations until convergence. We are currently applying CPI to several other tasks where different orders have shown to result in significantly shorter solving times, particularly

during Online Learning.[26] Future work will include a more detailed investigation of this phenomenon.

Note that so far we have said little about why CPI works. One reason why CPI performs well on the tasks we present here is the fact that subparts of our model do well in rewarding what other parts would reward, and penalise what other parts would penalise. If this was not the case, then solutions that optimize a partial network would lead to many locally suboptimal formulae.[27] However, this only leaves us with a very fuzzy class of problems that CPI is bound to solve well. In the next chapter we will try to describe this class more concretely and help users in cases where CPI performs poorly for their model.

[26]For example, assume that we train a model for Entity Resolution using an Online Learning algorithm. In the beginning the weights are not properly trained yet, and the initial matching graphs may connect many atoms. This can lead to many violated transitivity formulae and to large partial networks. Now assume that we have a set of formulae that ensure that citations do not have more than 5 matched citations, maybe because we have seen this in the data. Then adding these formulae first will help to avoid several transitivity formulae violations because the matching graphs will be sparser.

[27]When a formula is locally suboptimal in a possible world it essentially either rewards something that is not in the world, or penalizes something that is in the world.

Chapter 6

Analysis

In chapter 5 we showed two Markov Logic Networks and datasets for which CPI performs well. However, it is not clear what the general class of MLNs is that can be efficiently solved by CPI. This is surely a difficult question because the behaviour of CPI not only depends on the size or connectivity of the Ground Markov Network, but also on the chosen base solver and the weights of the formulae. Nevertheless, in this chapter we will try to characterize the class of problems that are solvable with CPI. Our characterization will not draw a complete picture of this class; instead, it is intended to be used as a guideline that allows users of CPI to alter an intractable MLN and make it more tractable.

We will look at two aspects of CPI that determine its efficiency: the size of the partial networks we generate and the number of iterations we need until CPI terminates. This excludes one important factor: the speed of the base solver. This has two reasons. First, it is generally difficult to predict this speed; for example, the worst case complexity of ILP is exponential in the size of the input, but ILP solvers can still be extremely efficient even if the size of a problem is large.[1] Second, if we consider CPI as a meta algorithm that improves the performance of a base solver, then it is not so important how fast this base solver is, but whether we can make it faster. This will primarily depend on the size of the networks CPI generates and the number of CPI iterations. However, note that the actual improvement also depends on how much faster the base solver becomes when we reduce the network size.

To characterize the runtime behaviour of CPI we will proceed in two steps: first we will present simple inner and outer bounds for the partial networks generated during

[1] This also holds for Linear Programming (LP): one of the most successful algorithms for LP is the Simplex Method, which has a worst case complexity that is exponential in the size of the problem. However, it usually performs much better than LP algorithms with polynomial complexity.

CPI. In other words, we will present networks that are contained in every partial network, and networks that contain every partial network. Then we will bound the number of iterations of CPI by asking how many steps are needed to instantiate the final partial network. In practice this will enable users to understand CPI in a two-stage process. First the user can get a rough idea about the final network by using the inner and outer bounds for the partial networks generated during CPI. Then based on this rough idea of the final network he or she can estimate lower and upper bounds on the number of iterations needed.

Before we present the mentioned bounds we will briefly introduce some prerequisites that will simplify our later explanations. We will also introduce a synthetic problem in order to illustrate the properties of CPI that we discuss in this chapter and evaluate CPI in a controlled environment.

6.1 Prerequisites

It will be convenient for us to represent the Ground Markov Network that a Markov Logic Network defines, and all partial networks generated during CPI, as sets of ground formulae instead of using the notions of graphs or sets of variable bindings (compare section 4.2). In particular, we will use the term $G_{L,V,\mathbf{x}}$ to describe the set of all ground formulae in the Conditional Ground Markov Network $M_{L,V,\mathbf{x}}$ (cf. section 3.6), and the term G^i to denote the set of ground formulae that corresponds to the partial grounding $\mathbf{G}^i$ at iteration i of CPI.

For example, for an MLN $L = \{(\phi, w)\}$ that only contains the formula ϕ

$$role(i, \text{AM-TMP}) \wedge i \neq j \Rightarrow \neg role(j, \text{AM-TMP}) \tag{6.1}$$

the initial partial grounding

$$\mathbf{G}^0 = (\{\{i/1, j/1\}, \{i/1, j/2\}\}) \tag{6.2}$$

corresponds to the set of ground formulae

$$\begin{aligned} G^0 = \{&role(1, \text{AM-TMP}) \wedge 1 \neq 1 \Rightarrow \neg role(1, \text{AM-TMP}), \\ &role(1, \text{AM-TMP}) \wedge 1 \neq 2 \Rightarrow \neg role(2, \text{AM-TMP})\} \end{aligned} \tag{6.3}$$

Moreover, the term $w(G)$ for a set of formulae $G \subseteq G_{L,V,\mathbf{x}}$ will refer to the sum $\sum_{g \in G} w(g)$ where $w(g)$ is the weight of the first order formula in L that was used to generate g. For example, in the above case we have $w\left(G^0\right) = 2w$. We define $w(\emptyset) = 0$.

We will sometimes make use of the ground literals that appear in a clause $c = l_1 \vee \ldots \vee l_n$ and denote these with $literals(c)$. The set of all ground atoms in a clause c will be referred to as $atoms(c)$.

We will also need the notion of entailment. A set of closed formulae Δ entails a closed formula ϕ if ϕ is true whenever all formulae in Δ are true. More formally, we write:

Definition 6.1. Let Δ be a set of closed formulae and let V be a vocabulary that is a superset of the vocabulary of Δ. Let ϕ be a closed formula. Then we say that Δ *entails* ϕ *with respect to vocabulary* V and write $\Delta \models \phi$ *wrt* V if and only if every possible world $\mathbf{y}$ based on V that satisfies Δ also satisfies ϕ. If the vocabulary V is clear from the context we simply write $\Delta \models \phi$.

Note that this definition only considers hidden possible worlds because in this chapter we exclusively look at the ground formulae $G_{L,V,\mathbf{x}}$ of the Conditional Ground Markov Network. These formulae do not contain any observed ground atoms and hence the state of the observed part of the world is irrelevant for our purposes. Moreover, this definition is consistent with the notion of entailment in Herbrand Logic [Hinrichs and Genesereth, 2006] and the traditional notion of entailment in First Order Logic if we assume domain closure and unique names.

Finally, we will need the notion of a prime implicant that allows us to distinguish between formulae in Δ with $\Delta \models \phi$ that are strictly required in order to entail ϕ and those that are redundant in this regard.

Definition 6.2. A set of closed formulae P is a *prime implicant* of the closed formula ϕ if and only if $P \models \phi$ and for every subset $P' \subset P$ $P' \not\models \phi$ holds.

6.1.1 Summarizing Local Formulae

In the following we will always transform the set of local formulae to an equivalent set of ground unit clauses with respect to a given observation. Assume an MLN L, an observation $\mathbf{x}$ and a hidden ground atom a. We can divide the set of local ground formulae L_a that contain the hidden atom a (and no other hidden atoms) as follows:

- L_a^+: the set of ground formulae that are true in $(\mathbf{x}, \mathbf{y})$ for all $\mathbf{y}$ with $a \in \mathbf{y}$ and false otherwise
- L_a^-: the set of ground formulae that are false in $(\mathbf{x}, \mathbf{y})$ for all $\mathbf{y}$ with $a \in \mathbf{y}$ and true otherwise

- $L_a^{\sim}$: the set of ground formulae that are either always false or always true in $(\mathbf{x},\mathbf{y})$ for all $\mathbf{y}$, regardless of $a \in \mathbf{y}$.

This definition allows us to define a *summarized* MLN L' as follows. Let w_ϕ denote the weight of the first order formulae associated with the ground formula ϕ and L_G the set of all non-local formulae in L, then

$$L' = M_G \cup \left\{ \left(a, \sum_{\phi \in L_a^+} w_\phi - \sum_{\phi \in L_a^-} w_\phi \right) | \text{ground atom } a \right\} \tag{6.4}$$

It should be clear that for any $\mathbf{y}$ the scoring functions $s_L(\mathbf{y},\mathbf{x})$ and $s_{L'}(\mathbf{y},\mathbf{x})$ only differ by a constant $K_\mathbf{x} = s_L(0,\mathbf{x}) - s_{L'}(0,\mathbf{x})$: if we change the state of any atom a in $\mathbf{y}$ from false to true, the local formulae will change their contribution by $\sum_{\phi \in L_a^+} w_\phi - \sum_{\phi \in L_a^-} w_\phi$ for both L and L'. This means that the$\mathbf{y}$ that maximizes $s_L(\mathbf{y},\mathbf{x})$ also maximizes $s_{L'}(\mathbf{y},\mathbf{x})$. Note that this also means that the conditional probabilities $Pr_L(\mathbf{y}|\mathbf{x})$ and $Pr_{L'}(\mathbf{y}|\mathbf{x})$ are identical.

To illustrate the transformation let us look at the following local formulae[2] taken from section 4.4:

$$\begin{aligned} pos(x,\mathrm{NN}) &\Rightarrow hasRole(x) \quad \langle w_1 \rangle \\ pos(x,\mathrm{RB}) &\Rightarrow hasRole(x) \quad \langle w_2 \rangle \\ pos(x,\mathrm{IN}) &\Rightarrow hasRole(x) \quad \langle w_3 \rangle \end{aligned} \tag{6.5}$$

For the first constituent $x = 1$ ("Now") the first and third formulae will always be true,[3] regardless of the state of hasRole(1). Hence the corresponding ground formulae will be in $L^{\sim}_{hasRole(1)}$. In contrast, the second ground formula

$$pos(1,\mathrm{RB}) \Rightarrow hasRole(1) \tag{6.6}$$

will be true if and only if *hasRole*(1) is true because the premise $pos(1,\mathrm{RB})$ holds for the observation $\mathbf{x}$. The formula is therefore a member of $L^+_{hasRole(1)}$ (while $L^-_{hasRole(1)}$ remains empty) and we add the formula-weight pair

$$(hasRole(1), w_2) \tag{6.7}$$

to the transformed MLN M' in equation 6.4. Likewise, for the second constituent $x = 2$ ("competition") the second and third formulae will always hold. Therefore we add

$$(hasRole(2), w_1) \tag{6.8}$$

[2] Again these formulae are called local because when grounded they only contain one hidden ground atom (*hasRole*).

[3] Because "Now" is neither a NN or IN and hence the premise is false.

to the transformed MLN M'. For the remaining constituents we proceed analogously.

The conversion we present here has two advantages:

- The partial network defined by the ground unit clauses alone can be solved very efficiently: the MAP solution includes a ground atom if and only if the corresponding ground unit clause has a positive score.
- By summarizing local formulae we can describe an MLN more compactly—this will simplify the analysis of CPI's runtime behaviour.

6.1.2 Negating Formulae with Negative Weights

In the following we will also assume that all formulae in a Markov Logic Network have positive weight. This can always be ensured by simply replacing each formula ϕ with negative weight $-w$ by its negation $\neg\phi$ and the positive weight w. To see that a transformed MLN L' is equivalent to the original MLN L, consider that both networks could assign different scores to the world $(\mathbf{x},\mathbf{y})$ but when $\mathbf{y}$ is changed the scores will change by the same amount: if a grounding $\phi[B]$ was false but is now true we will subtract w in both cases. In the original MLN this is the case because $\phi[B]$ will contribute $-w$. In the converted MLN this holds because the contribution w of $\neg\phi[B]$ is taken away. Hence the scores assigned by L and L' always differ by a fixed constant and the probability distributions defined through L and L' are identical.

To illustrate this transformation, let us assume that we have summarized the local formulae in our Semantic Role Labelling example and that this resulted in the ground unit clause $role(1,\text{AM-TMP})$ with negative weight -2. In the transformed MLN we would replace this formula with $\neg\text{role}(1,\textit{AM-TMP})$ with weight 2.

In the following we will use the term *positive MLN* for an MLN where each formula has a positive weight. In many cases we will present MLNs with positive and negative weights but will explain the behaviour of CPI in terms of the positive MLN.

6.2 Synthetic Data

Instead of using a real-world problem to analyse CPI, in this chapter we will use a synthetic problem. This allows us to test CPI in a controlled environment and illustrate some of its properties more easily. For an evaluation of CPI in a real-world setting, the reader is referred to chapter 5.

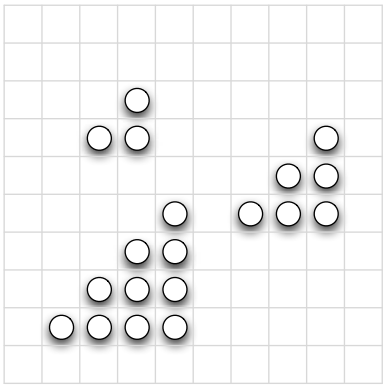


Figure 6.1: Grid with three triangles.

We will look at a 2D grid world and the task of inferring a set of blocks in this world. These blocks are represented through a binary predicate *block* defined over two integer arguments in $[1,n]$, where n is the number of columns/rows of the grid. The Markov Logic Network for this task favours solutions where blocks are arranged as triangles, such as those presented in figure 6.1. The formula that encourages triangle-like structure is

$$block(x,y) \wedge block(x+1,y) \Rightarrow block(x+1,y+1) \left\langle w_p \right\rangle \tag{6.9}$$

which requires a block in a cell whenever there is a block to the south and southwest of this cell. In the following we will refer to this formula as the *triangle formula*. We will sometimes require that every substructure has to be a complete triangle. In this case we set w_p to be very high.

Along with formula 6.9 we have n^2 ground unit clauses, one for each grid cell (x,y), described by:

$$block(+x,+y) \left\langle w_{x,y} \right\rangle \tag{6.10}$$

Note that each of these formulae can have distinct weight $w_{x,y}$. In practice we usually do not have an individual formula for each ground atom—otherwise the MLN would not generalize over differently shaped and size grids. However, we can understand this model as a Markov Logic Network with one global formulae (formula 6.9) and a set of local formulae that have been generated using the transformation in section 6.1.1. In the following we will usually draw the weights $w_{x,y}$ from a distribution. This simulates the scenario where different ground atoms have different local scores based on different local formulae that hold for each atom.

We chose this triangle world mainly because it allows us to easily visualise the theoretical bounds we present in this chapter. It will give us a sense of how a partial

network can grow, starting from the initial set of ground formulae. For instance, we will see that if the initial set of ground formulae are a set of ground atoms that represent a bottom row of a triangle, then the partial network essentially grows with the partial triangle we find in each iteration. The triangle world will also help us to visualize the fact that partial networks can only grow as far as the set of all ground formulae allows them to. For example, we will observe that triangles (and corresponding triangle formulae) have a maximum height that depends on the width of the rows represented by the positive ground atoms in the initial ground network.

The triangle world is also loosely related to the Entity Resolution task we presented in section 5.4. Roughly speaking, we can consider the blocks in a triangle as active citation matches, and the triangle formula 6.9 as a simplified version of the transitivity formula.[4] More generally, we can often use the triangle world to visualise the behaviour of CPI for MLNs in which rules are either of the form "something is true" or "if something is true, something else is also true".[5]

MLNs such as the one for Semantic Role Labelling presented in chapter 5 are more difficult to visualise in this world. To do so we would need to add additional rules that deactivate certain blocks if other blocks are active (corresponding to the "no duplicate A1 roles" argument); we would also need a rule which implies that if a certain block is active, at least one of a set of other blocks has to be active, too. This corresponds to the formula that requires us to assign a semantic label to a constituent whenever the constituent is marked by the *hasRole* predicate.

In the following we will usually sample the weights $w_{x,y}$ for formula 6.10 from some distribution and generate several triangle problems using this procedure. This will ensure that the behaviour we see is not a random artifact of a very specific set of weights. However, we do constrain the distributions in certain ways in order to reproduce what we think are characteristic behaviours of CPI.

6.3 Inner Bounds on Partial Networks

We will first ask the following question: given a MLN M, the initial grounding G^0 and the observation $\mathbf{x}$, can we predict a set of ground formulae that we will definitely

[4] We can also go on and consider venue, title and author matches as blocks in the grid. In this case the formulae that connect these matches to citation matches are again closely related to the triangle formula—they imply matches (blocks) based on the existence of other matches (blocks).

[5] More formally speaking, in this type of MLN all formulae are Horn clauses (disjunctions with at most one positive literal).

instantiate during CPI? In other words, is there a grounding G and iteration m with $G \subseteq G^i$ for all iterations i after m? We will call such a grounding an *inner bound* on the final partial grounding. Such a grounding can be very helpful for predicting when CPI will not work well: if G is a large part of the complete grounding, then CPI will essentially do the same amount of work as a purely propositional solver in every iteration after m. Hence just using a propositional solver can be faster than using CPI.

6.3.1 Bound Based on First Solution

The first inner bound is very simple and obvious.[6] However, it reflects one of the major limitations of CPI in its current form. Note that we will again use the notation $\models_{\mathbf{y}} g$ to indicate that a formula g is true (satisfied) in a possible world $\mathbf{y}$ (see section 3.1.2).

Theorem 6.1. *Let L be a positive MLN, V a vocabulary, $\mathbf{x}$ an observation, G^0 an initial grounding, BS an exact base solver and $\mathbf{y}^1 = \arg\max_{\mathbf{y}} s_{G^0}(\mathbf{x}, \mathbf{y})$ the unique MAP solution for the initial grounding. Then for the partial grounding G^i in every iteration $i > 0$ during Cutting Plane Inference with L, $\mathbf{x}$, G^0 and BS as input parameters the following holds*

$$\left\{ g \in G_{L,V,\mathbf{x}} \middle| \models_{\mathbf{y}^1} \neg g \right\} \subseteq G^i$$

Proof. Perform the first step of CPI. □

In other words, this theorem simply states that all formulae which are violated after solving the initial problem will be part of all future partial groundings.

We can use this theorem to illustrate the impact of the sign of the weight of a formula. For example, assume we are doing Semantic Role Labelling and the solution to the initial partial grounding is the empty solution. Now assume that we have a formula such as

$$role(i, \text{AM-TMP}) \wedge i \neq j \Rightarrow \neg role(i, \text{AM-TMP}) \tag{6.11}$$

If this formula has a positive weight we have to search for all pairs (i, j) for which the formula does not hold. If the solution contains no ground atoms, the premise will be false for each (i, j) because the premise *role(i,*AM-TMP*)* is always false in the empty solution. Hence the first partial grounding will not contain any ground formulae of this type. However, if the formula has negative weight we need to negate it in the positive

[6] Note that both theorems in section 6.3 are quite trivial observations. However, we believe that they need to be highlighted and we do so by formulating them as theorems.

MLN. The above theorem then tells us that every partial grounding will contain all groundings of the formula because its negation is always false in the empty solution.

Note that we could write similar theorems for all following solutions $\mathbf{y}^2, \mathbf{y}^3, \ldots$. However, the solution of the initial problem can often be easily predicted, in particular if the initial problem only consists of local formulae. For example, in the case where G^0 consists only of ground unit clauses with positive weights (see section 6.1.2) the initial solution is simply the set of unnegated ground atoms in G^0. In contrast, for any solution $\mathbf{y}^i$ for $i > 1$ we need to guess the MAP solutions of networks with highly interacting variables. Moreover, the structure of the networks in later iterations depends on these guessed MAP solutions. Therefore we will guess each future MAP solution with less and less confidence.

To further illustrate this aspect of CPI, we will look at our synthetic problem and set up an experiment in which we gradually change the weight of the rule in equation 6.9 from -2 to 2 while measuring the size of the final partial network. We will expect to see a significant change in the number of instantiated formulae when the weight changes from a positive value to a negative value, because at this point the formula will be negated and the above theorem will predict a larger inner bound.[7]

The initial partial grounding G^0 will consist of all ground unit clauses as described with equation 6.10. We will set the weights of these clauses so that the initial solution consists of the blocks on the bottom row of the grid. This can be achieved by giving a positive weight $w_{X,0}$ for all blocks on the bottom row and a negative weight $w_{X,Y>1}$ for all blocks on higher rows. We generate a sequence of such scenarios by sampling $w_{X,0}$ uniformly from $(0,2]$ and $w_{X,Y>1}$ uniformly from $[-1,0)$.[8]

In figure 6.2 we see the average number of ground formulae in the final partial network for different weights w_p. Note that here and in all following graphs we only show the number of groundings of the triangle formula 6.9 and do not count the initial ground formulae.

For each weight we averaged over 10 different assignments for the weights of the unit clauses, according to the distributions we described above. The dashed line indi-

[7] Strictly speaking the bound itself does not predict this significant change: we could theoretically have as many violated formulae for a positive weight as for a negative weight. However, the bounds we will present in section 6.4 we will show that for a positive weight we are guaranteed to have significantly less grounded formulae.

[8] By creating an active bottom row while all other rows remain inactive we are essentially emulating a citation matching scenario where the initial solution contains a cluster of citations that are all connected, though not necessarily directed. However, this is not strictly necessary to reproduce the behaviour we will observe in figure 6.2—any kind of sparse initial solution will lead to this effect.

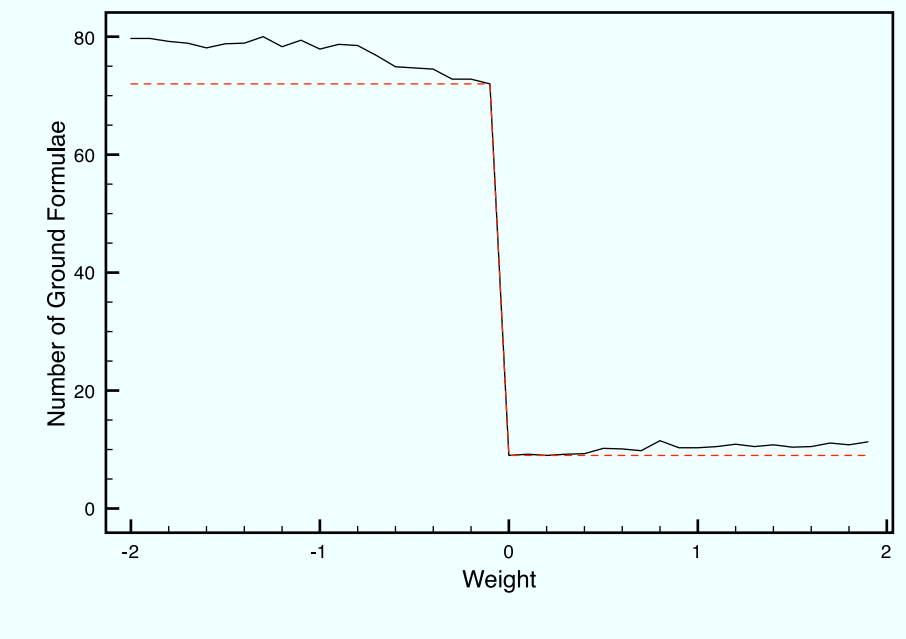


Figure 6.2: Number of ground formulae in the final problem when changing the weight of the triangle formula from -2 to 2, averaged over 10 instances (with different unit clause weights). The dashed line indicates the number of violated formulae in the initial solution $\mathbf{y}^1$. The black solid line indicates the number of ground formulae in the final network (without the local formulae).

cates the number of violated formulae in the initial solution $\mathbf{y}^1$. The black solid line indicates the number of ground formulae in the final network (without the local formulae). We can see how the behaviour of CPI in terms of the generated network changes dramatically at $w_p = 0$. The initial solution satisfies all but 9 of the 81 triangle ground formulae. In the case of a negative weight we have to add all 72 cases where the triangle formula is true. In contrast, for positive weights only the 9 groundings that don't satisfy the triangle formula are added. [9]

This behaviour can cause CPI to be slow and impractical for certain problems. However, often weights are bound to have the right sign. Coming back to the SRL example: if the formulae

$$role(x, \mathrm{A1}) \wedge x \neq y \Rightarrow \neg role(y, \mathrm{A1}) \tag{6.12}$$

that is supposed to encourage cases where there is not more than one A1 argument has a negative weight, it essentially encourages the opposite; that is, the formula rewards solutions with more than one A1 argument. This violates the assumptions we wanted to model. So in this case, and in other cases we encounter in chapter 5, the sign of the weight poses no problem for Cutting Plane Inference. Yet, in general models with hundreds of formulae the weights can interact in complex ways and their signs are not guaranteed to follow our intuition. An important future direction of research is thus to make CPI more robust with respect to the sign of formula weights.

6.3.2 Bound Based on Final Solution

The previous bound can sometimes be misleading. We can have a first solution $\mathbf{y}^1$ that violates only very few ground formulae. However, during the course of CPI, solutions that violate a large number of formulae may be generated. But as we mentioned, predicting the solutions of later iterations is difficult. So how can we further analyse CPI without having to guess how CPI will solve the complex networks that arise in later iterations?

One answer is to consider the optimal MAP solution. Certainly it is not easy to think of what this solution might be when we are given an MLN and observation—this

[9] Note that we see a larger number of formulae for negative weights that are close to -2 than for negative weights close to 0. The reason is the fact that in the initial solution the triangle formulae of the bottom row are violated and will therefore not be instantiated in the first iteration (because the weight of w_p is negative). The more negative w_p will get, the more likely will the second row from the bottom be active in the second solution (because this configuration violates the triangle formulae of the second row); and the more active atoms are active in the second row, the more triangle formulae of the bottom row are satisfied, and therefore instantiated (again because w_p is negative).

is actually the problem we want to solve with CPI. However, often we have good idea what the MAP solution *should* be if we had a perfect MLN: the true gold solution. For example, in the case of Semantic Role Labelling the user has an exact idea of the correct labelling. And if we know that this labelling violates the "no more than one AM-TMP modifier" constraint we also know that the final partial network will definitely contain the corresponding grounding.

More formally, we write the following:

Theorem 6.2. *Let L be a positive MLN, V a vocabulary,* $\mathbf{x}$ *an observation,* G^0 *an initial grounding, BS an exact base solver and* $\hat{\mathbf{y}}$ *the unique MAP solution for the given MLN M and observation* $\mathbf{x}$*. Then for the partial grounding* G^* *in the last iteration of Cutting Plane Inference with L,* $\mathbf{x}$*,* G^0 *and BS as input parameters the following holds*

$$\left\{ g \in G_{L,V,\mathbf{x}} | \vDash_{\hat{\mathbf{y}}} \neg g \right\} \subseteq G^*$$

Proof. Since CPI is exact for an exact base solver it will generate $\hat{\mathbf{y}}$ in one iteration and will add all violated formulae that are not yet in the partial problem. □

To illustrate that this bound can be useful when bound 6.1 fails—even though it discards all groundings that are added in all other iterations but the last—we will look at a triangle problem where only one ground formula is violated in the initial solution but a larger number in the MAP solution. This is a worst-case scenario, but it shows that a user of CPI can tailor his or her model so that the initial solution does not violate many formulae while CPI still generates large final networks.

In particular, we set up the MLN in such a way that the MAP solution is a partial triangle with height h. To this end we add one additional deterministic formula

$$y > h \wedge block(x,y) \Rightarrow block(x,y+1) \tag{6.13}$$

which ensures that for every active cell higher than h, the cell on top of it must be active, too.[10] The weights of the unit clauses are assigned according to figure 6.3: the grey blocks of the triangle until height h have a very large positive weight, the white blocks until height $n-1$ have a low positive weight. All other cells have a large negative weight. The weight w_p is set to a value smaller than the weight of the grey cells.

With this setting the MAP solution will be the partial triangle with height h. To understand this, consider that for every active block b above h, all blocks above b need

[10] Note that h is a fixed parameter in this formula.

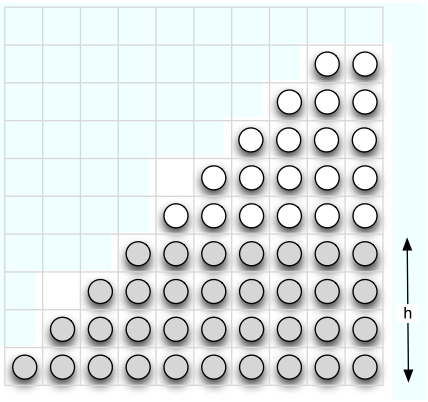


Figure 6.3: Visualization of unit clause weights: The grey circles have a very high positive weight, the white circles have a low positive weight. All other blocks have a large negative weight. The local solution is the triangle including the solid grey and white circles, without the top circle. The MAP solution is the partial triangle with height h.

to be active, too. Therefore the cost of activating a block b above h is affected by the negative weight of the top blocks. In our case the top cells (in fact the whole upper left triangle) have very negative weight and hence it is cheaper to just remove the block b. Now assume that all blocks above h are removed. The triangle formula is now violated several times at height h. However, because the weight of the triangle rule is lower than the weight for the blocks under h we get a higher overall score if we leave the corresponding formulae violated instead of removing blocks under h.

Note that the initial solution $\mathbf{y}^0$ is the triangle with height $n-1$ (as usual we assume that the initial partial grounding contains all local ground formulae). This means that in the initial solution there is only one violated triangle formula: the one that implies that the topmost block is active if the two blocks below it are active. There is also a set of violated groundings of formula 6.13 that appear at the border between the white blocks and the empty cells. Since the empty cells have very negative weights, and the weights of the white blocks are positive but not very large, the optimal choice is to remove the white blocks in order to satisfy formula 6.13. This process of removing white blocks is continued until the height h is reached.

Figure 6.4 shows the number of (triangle) ground formulae in the final partial network with respect to the height of the partial MAP triangle h. In all experiments we used a 20x20 grid. The solid black line shows the actual number of ground formulae; the dashed line shows how many ground formulae are violated (locally suboptimal) in the MAP solution and the dotted line shows how many formulae are violated in the

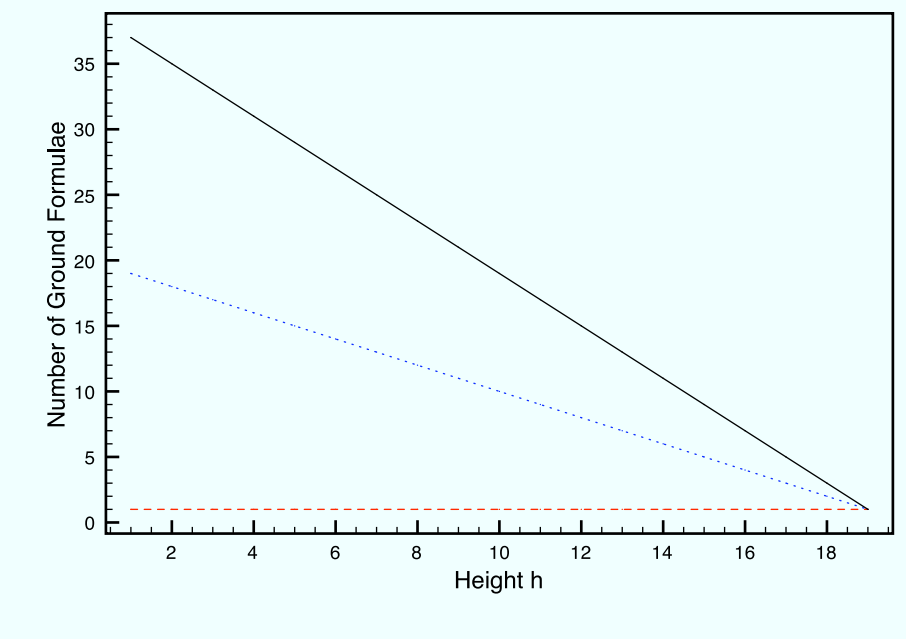


Figure 6.4: Number of ground formulae in the final problem when changing the height h of the partial MAP triangle. The solid black line represents the actual number of ground formulae (again only considering the triangle formulae), the dotted line represents the number of violated formulae in the MAP solution and the dashed line represents the number of violated formulae in the initial solution.

initial solution.

We observe that the bound based on the initial solution is not very helpful in this setting. On the other hand, the bound based on the MAP solution is much tighter. This cannot be said in general. For example, in the case of an MLN where all global formulae are deterministic the MAP solution will satisfy all formulae and hence the inner bound defined by theorem 6.2 will be the empty set.[11] However, we should have made clear that considering the MAP solution can be helpful when we assess the potential of CPI with respect to a certain model.

6.4 Outer Bounds on Partial Networks

In the previous section we tried to provide an inner bound for the partial networks generated during CPI (and hence a lower bound on their sizes). This can help us to answer the question when CPI might *not* be more efficient than the propositional base solver. However, we would also like to characterize cases where CPI can be more efficient. To this end we will look at outer bounds of the partial networks (and hence upper bounds their maximal sizes).

6.4.1 Bound Based on Entailment

Let us again assume that all formulae have positive weights. Then finding the locally suboptimal groundings of formulae amounts to finding the ground formulae that are violated in the current solution. Hence, if we can show that some ground formulae cannot be violated in any solution during CPI we know that these formulae will never be part of the networks generated during CPI.

In order to determine whether a ground formula g can be violated, we start with the following observation: assuming that the MAP solution is unique, a formula will only be violated if there is a set of formulae $P \subseteq G_L$ that logically entails $\neg g$ or, more formally, $P \models \neg g$.[12] This leads to the following theorem:

Theorem 6.3. *Let L be a positive MLN, V a vocabulary, $\mathbf{x}$ an observation, G^0 an initial grounding and BS an exact base solver. Assume that each partial network G^i has a unique MAP solution. Then for every iteration i during Cutting Plane Inference*

[11] This is the case for the SRL model in section 5.3.

[12] Note that when we say $P \models g$ we mean $P \models g$ with respect to vocabulary of the Markov Logic Network L.

with L, $\mathbf{x}$*,* G^0 *and BS as input parameters holds*

$$G^i \subseteq \{g \in G_{L,V,\mathbf{x}} | \exists P \subseteq G_{L,V,\mathbf{x}} : P \models \neg g\}$$

Proof. By contradiction: we assume that there exists a formula g that is violated in the unique MAP solution $\mathbf{y}$ at iteration i but has no $P \subseteq G_L$ with $P \models \neg g$. Let $K \subseteq G_L$ be the set of all formulae satisfied in $\mathbf{y}$, then $K \not\models \neg g$ because $K \subseteq G_L$. This means that there exists at least one world $\mathbf{y}'$ in which all formulae in K are true and the formula g is true as well. Because the same formulae K will be true in $\mathbf{y}'$ this solution has at least the same score as $\mathbf{y}$. If it has the same score, $\mathbf{y}$ cannot be the *unique* MAP solution. If it has a higher score, $\mathbf{y}$ cannot be a MAP solution. □

Note that the requirement that all MAP solutions during CPI are unique is often fulfilled. For example, frequently the weights learnt by numeric optimization routines are distinct and the MAP solutions defined by these weights are likely to be unique. However, generally we cannot guarantee uniqueness, particularly in settings where the weights have just been initialized and are still to be trained.[13] In such cases the behaviour of CPI becomes less predictable.

Theorem 6.3 can guarantee that partial networks will be significantly smaller than the complete network. Consider figure 6.5: here we have a setting where the sequence of w blocks (or the corresponding unit clauses) on the bottom row have a positive weight and all others have a negative weight.[14] The positive MLN then contains the positive ground unit clauses for the w blocks on the bottom row and negated ground unit clauses for all remaining blocks. The set of all possible ground formulae is the set of all groundings of the triangle formula and all unit clauses.

In this setting the only *block(x,y)* ground atoms we can prove using these ground formulae are those in the triangle of width and height w, denoted with dashed circles in figure 6.5. For example, it is possible to prove *block*(10,2) using

$$\{block(9,1), block(10,1), block(9,1) \wedge block(10,1) \Rightarrow block(10,2)\} \tag{6.14}$$

but not *block*(2,2) because neither *block*(1,1) nor *block*(2,1) are available. Therefore we can only ever violate the ground formulae within this triangle (assuming that the partial MAP solutions are unique). This implies that the grid size has no impact on the number of global ground formulae; as long as w is small we are guaranteed to only instantiate a small fraction of the complete network.[15]

[13] Note that such cases can be avoided through randomisation of the initial set of weights.

[14] Again, this resembles an Entity Resolution scenario where the initial (local) matching indirectly

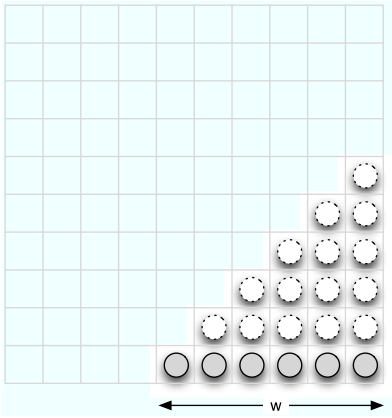


Figure 6.5: A triangle problem where the w blocks in the bottom row have a positive weight and all remaining blocks have a negative weight. The dashed blocks show which blocks we can prove using the bottom row of blocks and the triangle ground formulae.

To further illustrate this behaviour we evaluate the size of the final partial network based on a 10x10 grid and different values for the triangle width w. The solid black line in figure 6.6 shows the number of (triangle) ground formulae in the final network, averaged over 10 random weight assignments. Here we generate the corresponding positive weights by uniformly drawing a value from $\big(0,10\big]$ and the negative weights by drawing from $\big[-1,0\big)$.[16] The dashed line corresponds to the number of ground formulae in the full network we would have to solve if we are not using CPI. Finally, the dotted line denotes the number of ground formulae that can be violated according to theorem 6.3.

The graph shows that the size and connectivity of the complete network, represented by the dotted line, can be a poor indicator of the difficulty of inference with CPI. In contrast, by considering what we can prove or disprove we get a tighter bound on the number of ground formulae and hence a better picture of why CPI works or not.

Note that we will observe the same behaviour in cases where all ground formulae in the positive MLN are Horn clauses such as the triangle formula 6.9 or the transitivity formula

$$sameCitation(c_1,c_2) \wedge sameCitation(c_2,c_3) \Rightarrow sameCitation(c_1,c_3) \qquad (6.15)$$

for Entity Resolution. Roughly speaking, the smaller the set of positive ground unit

connects several citations in a cluster, but these citations are not yet all pairwise connected.

[15]Note that in practice in can be difficult to explicitly calculate this bound for a given MLN. However, often the user will at least have a rough idea of what can be entailed, and what cannot. In this cases the bound will serve as a guideline in order to design more efficient MLN models.

[16]We pick these intervals in such a way that the graph does not suggest tightness of the bound. If the positive values were smaller the bound would even be closer to the actual number of ground formulae.

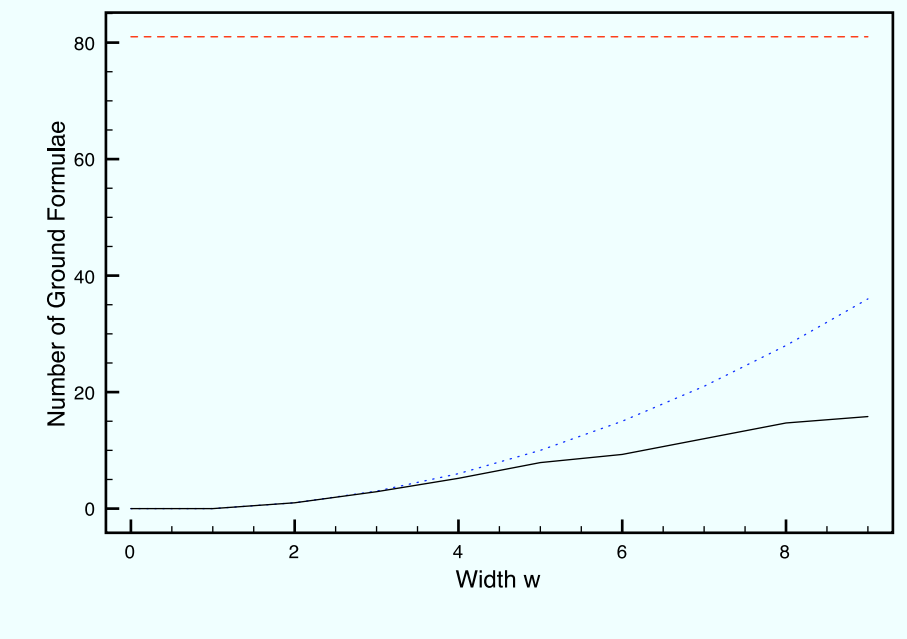


Figure 6.6: Number of ground formulae when changing the width of the positive bottom row. The solid black line in figure 6.6 shows the number of ground formulae in the final network, averaged over 10 random local weight assignments. The dashed line corresponds to the number of ground formulae in the full network. The dotted line denotes the number of ground formulae that can be violated according to theorem 6.3.

clauses, the smaller we can guarantee the final partial network to be. This can be understood when we consider that in the case of no positive ground unit clauses (for example, no *block*(x,y) clauses) no other positive atoms can be proved and hence no Horn clause will ever be violated and instantiated.

6.4.2 Bound Based on Weights

In the above case we observed that for increasing width w there is a growing discrepancy between the number of formulae that can be disproved (the dotted line) and those which are actually instantiated (the black line). How can this be explained? Let us look at a formula g at a certain height of the triangle. In order to prove $\neg g$ (and hence instantiate g), all blocks in the triangle under g have to be active. However, only those blocks on the bottom row have a positive weight, all others have a negative weight. This implies that with greater height, violating a formula means activating more and more negative blocks. When the total cost of these negative atoms becomes larger than the cost of the positive blocks it will be cheaper to just remove the positive blocks.

This suggests that a formula can only be instantiated if the "strength" of its disproofs is sufficiently large. In the following we will define this notion of strength. Note that the following definition is only helpful if there is only one prime implicant[17] of $\neg g$ in G_L. The definition could be extended in order to take this into account. However. for brevity and simplicity we focus on the case with only one proof.

Definition 6.3. Let L be a positive MLN, V a vocabulary, $\mathbf{x}$ an observation, G^0 an initial grounding, and g a ground formula so that there exists exactly one prime implicant $P_{\neg g} \subseteq G_{L,V,\mathbf{x}}$ of $\neg g$. Let for any $S \subseteq P_{\neg g}$ the set $D_{S,g}$ be the set of ground formulae $d \in G^0$ so that $P_{\neg g} \models \neg d$ and for all sets of ground formulae $K \subseteq G_{L,V,\mathbf{x}}$ holds $K \setminus S \not\models \neg d$. Then the instantiation support $isup(g)$ of g is defined as:

$$isup(g) = \min_{S \subseteq P_{\neg g}} w(S) - w\left(D_{S,g}\right)$$

If there is no prime implicant of g we define $isup(g)$ to be 0.

In this definition every set $S \subseteq P_{\neg g}$ is a set of formulae that need to be true if g is false (because it is a subset of the only prime implicant of $\neg g$). The set $D_{S,g}$ is a set of initial ground formulae that will always be false if $\neg g$ is entailed and which cannot be implied to be false by any set of formulae that does not contain S. We will see in the following theorem that the formulae in S *support* $\neg g$ (and hence the instantiation of g) and that the formulae in $D_{S,g}$ *inhibit* $\neg g$.

Theorem 6.4. *Let L be a positive MLN, V a vocabulary, $\mathbf{x}$ an observation, G^0 an initial grounding and BS an exact base solver. Assume that each partial network G^i defines a unique MAP solution and that for each $g \in G_{L,V,\mathbf{x}}$ there is not more than one prime implicant of $\neg g$. Then for every iteration i during Cutting Plane Inference with the parameters L, G^0, $\mathbf{x}$ and BS* holds

$$G^i \subseteq \{g \in G_{L,V,\mathbf{x}} | isup(g) > 0\}$$

Proof. By contradiction: assume $\mathbf{y}$ is the unique MAP solution at iteration i, that g is violated in $\mathbf{y}$ ($\models_{\mathbf{y}} \neg g$) and that $isup(g) \leq 0$. From the proof of theorem 6.3 we know that the set K of all formulae satisfied in $\mathbf{y}$ entail $\neg g$. Because there is only one prime implicant $P_{\neg g}$ of $\neg g$ this implicant must be contained in K. Let $S \subseteq P_{\neg g}$ be the set of ground formulae that minimises $w(S) - w\left(D_{S,g}\right)$. We know that no $d \in D_{S,g}$ can be active in $\mathbf{y}$ because $P_{\neg g} \models \neg d$. We create a new set of formulae $K' = K \setminus S \cup D_{S,g}$.

[17] Compare section 3.1.2.

Because no set of formulae without S can imply $\neg d$ for any formula $d \in D_{S,G}$, the formulae in K' are consistent. This means that there exists a possible world $\mathbf{y}'$ which satisfies all formulae in K', possibly more. The world $\mathbf{y}'$ has a score of at least $w(K) - w(S) + w\left(D_{S,g}\right)$. If $isup(g) \leq 0$ the score of $\mathbf{y}'$ is therefore as high or higher than the score of $\mathbf{y}$, contradicting the assumption that $\mathbf{y}$ is the unique MAP solution. □

Calculating the instantiation support of a formula is a difficult problem by itself (it is another optimization task). However, it can be easier to calculate a lower bound for $isup(g)$. This lower bound can then be used to calculate an outer bound on $\{g \in G_L | isup(g) > 0\}$. In order to get a lower bound on $isup(g)$ we just have to find one support set S and the corresponding set $D_{S,g}$ that inhibits $\neg g$,

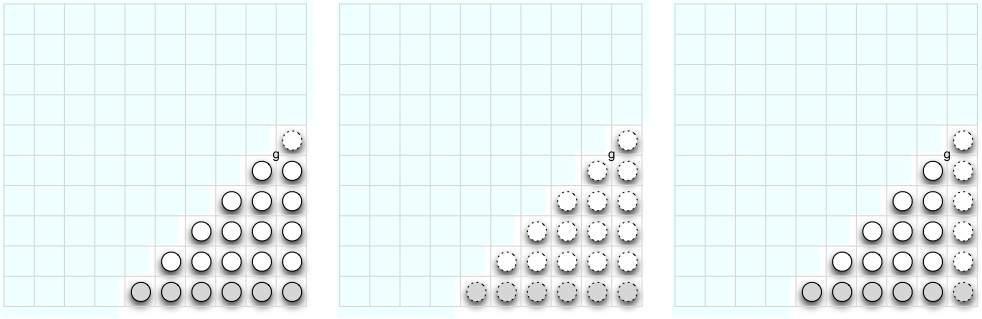
For example, consider the case of formula g in figure 6.7 a) that implies the existence of the dashed block given the existence of the two blocks under it. Figure a) shows a solution that violates g because the blocks under the dashed block are active while the dashed block itself is not. In this figure the white blocks have negative weight and the grey ones have positive weight. Hence $\mathbf{G}^0$ contains the positive atoms of the bottom row and the negated atoms of the rows above the bottom row. In this case we can think of at least two support sets S and corresponding inhibits $D_{S.g}$:

- S is the set of positive atoms of the bottom row, because every proof of $\neg g$ requires these to be active. The inhibit $D_{S,g}$ of this set is the set of all negated block atoms for the rest of the triangle (minus the top block), because corresponding positive atoms have to be true if $\neg g$ is proven but cannot entailed to be true without the bottom row atoms in S. This configuration can be seen in sub-figure b).

- S only contains the bottom row atom at the far right of the triangle. Surely this atom always has to be true if we prove $\neg g$. The inhibit $D_{S,g}$ is the set of negated atoms for the blocks in the rightmost column under the top block of the triangle, because the corresponding positive atoms have to be true if $\neg g$ is proven but cannot entailed to be true without the rightmost block atom on the bottom row. This configuration can be seen in sub-figure c).

In the first case we lose the score of the complete bottom row but win the score of all negated atoms in the triangle. In the second case we only lose the score of one positive atom and win the scores of the negated atoms in the column "under g".

In figure 6.8 we see the actual number of groundings for formula 6.9 compared to an upper bound based on the column and triangle bound of $isup(g)$. The controlled



(a) Violated g (b) Pyramid-based (c) Column-based

Figure 6.7: Figure a) shows a solution that violates formula g. Here the dark circles have positive weight and the white circles have negative weight. Figure b) and c) show two ways of removing the proof of $\neg g$: b) remove all positive and negative atoms under it; c) remove the column under it. Both cases will not violate any triangle formulae under g.

parameter is the interval $[0,m]$ from which we sample the positive weights of the bottom row atoms. The higher these weights are, the more difficult it will be to remove the support sets while still improving score. Note that there is still a discrepancy between the actual number of ground formulae and the upper bounds. We could further improve the bound by looking for better ways to "remove the proof". However, even in this case we can observe how not only the existence of a proof of $\neg g$ controls whether g will be instantiated, but also the weight this proof was given.

In practice this bound tells us that we can guarantee even smaller partial problems if we manage to design and train an MLN that penalises violations through local formulae (or any other type of formula we pick as initial groundings). Let us consider the Semantic Role Labelling example to illustrate this. Assume that the initial ground network contains the ground unit clauses *hasRole*(1) and *hasRole*(2) which indicate that constituent 1 and 2 should be semantic arguments. Further assume that there are only two possible semantic labels, A1 and AM-TMP, and that the ground unit clauses

$$\{role(1,\text{A1}), \neg role(1,\text{AM-TMP}), \neg role(2,\text{A1}), \neg role(2,\text{AM-TMP})\} \tag{6.16}$$

are also part of the initial ground network. In this case we prove the violation of the ground clause

$$role(1,\text{A1})\, 1 \neq 2 \Rightarrow role(2,\text{A1}) \tag{6.17}$$

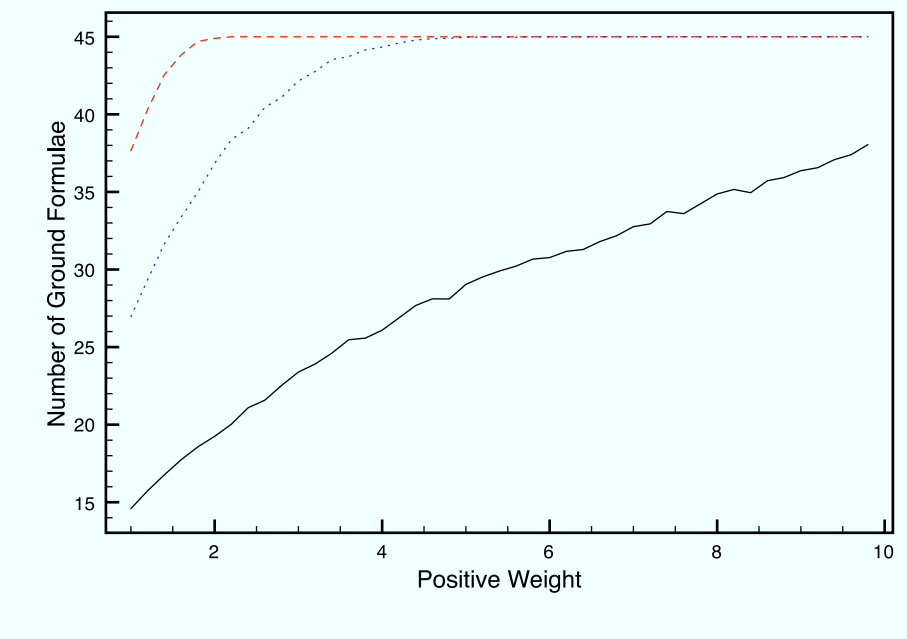


Figure 6.8: The solid line shows the number of instantiated ground formulae with increasing positive weight for the bottom row atoms. The dotted line represents the column-based upper bound, the dashed line represents the triangle-based bound.

that forbids duplicate A1 arguments using the following set of ground formulae:

$$\{role\,(1,\text{A1})\,, hasRole\,(2)\,, \neg role\,(2,\text{AM-TMP})\,, \\ hasRole\,(2) \Rightarrow role\,(2,\text{A1}) \vee role\,(2,\text{AM-TMP})\} \tag{6.18}$$

Hence according to theorem 6.3 formula 6.17 could be part of the final network. However, if we guarantee that the ground unit clause *role*(2,A1) has a very negative weight then we can "remove the proof" and always gain score because we are removing *role*(2,A1) from the solution. This leads to the question of how we can ensure that the corresponding ground unit clause has very negative weight. Often this can be achieved by carefully designing local features. In the above case only one constituent can be the true A1 argument, say constituent 1. Then most likely there will be some local properties which indicate that constituent 2 cannot be the A1 argument. For example, the constituent might appear on the right side of the verb. In this case an A1 label (assuming that A1=agent) is less likely. If we have a local formula that captures this observation, then the local score of *role*(2,A1) will be decreased.

6.5 Lower Bound on Number of Iterations

Sometimes we can guarantee that the partial problems generated during CPI will be relatively small. However, if CPI needs hundreds of iterations we do not gain much. Likewise, even if the partial problems are relatively large, CPI can still do well if it needs only few iterations. Hence the size of the partial problem alone is insufficient in order to predict whether CPI works well. In this section we will therefore first try to predict how many iterations CPI will at least take. In the next section we will then bound the number of iterations CPI will at most take.

How many iterations will it at least take to instantiate the final network G^*? Generally speaking we cannot predict exactly what this final network will look like. However, the bounds in the previous sections allow us to get a rough idea of G^*. When we want to predict or understand the behaviour of CPI we can use this rough idea in place of the exact G^*.[18]

We will first show that a minimal number of iterations can be determined if we know when certain literals first become true, i.e. in which iteration the solution will satisfy certain literals the first time. Note that this bound only applies to cases where the ground network that CPI instantiates is completely clausal (all ground formulae are disjunctions). This is the case for all experiments and applications we investigate in this thesis. It is possible to extend the bound to the more general case of first order ground formulae, but we omit this step for simplicity.

Lemma 6.1. *Let L be a positive MLN, V a vocabulary, $\mathbf{x}$ an observation, G^0 an initial grounding, BS an exact base solver and $G^* \subseteq G_{L,V,\mathbf{x}}$ the final network generated by Cutting Plane Inference with L, $\mathbf{x}$, G^0 and BS as input parameters. Let $first(l)$ be the iteration in which the ground literal l first becomes true and let G^* be clausal. Then for the iteration $\iota(G^*)$ in which G^* is completely instantiated holds*

$$\iota(G^*) \geq \max_{g \in G^*} \max_{l \in literals(g)} first(\neg l)$$

Proof. A clause $g = l_1 \vee \ldots \vee l_n$ is violated and instantiated if and only if all l_i are false. This cannot happen before each literal has become false for the first time. Hence g is instantiated not before $\max_{l \in literals(g)} first(\neg l)$ and G^* cannot be instantiated before all its formulae are instantiated. □

[18] Note that when we say a formula is instantiated in iteration i we mean that it was first violated in the solution for the partial network G^i (i.e., after the i-th call to the base solver).

Note that once a network G^* is completely instantiated CPI needs one more iteration that solves the MAP problem in the instantiated network (and finds no more violated formulae that have not already been added).

Lemma 6.1 is only helpful if we can provide a lower bound on the iteration $first(l)$ in which a literal l first becomes active. Fortunately this is possible in the case of clausal ground networks if we consider the following graphical representation of a set of clauses G^*. Let us represent each ground atom that appears in G^* as circle, each clause in G^0 as a solid square and each clause not in G^0 as a dashed square. Finally, let us add a directed edge (a,c) from atom a to clause c if and only if $\neg a \in literals(c)$ and a directed edge (c,a) from clause c to atom a if and only if $a \in literals(c)$.[19]

For example, consider the triangle scenario and a case where CPI instantiates the following global formulae

$$\begin{aligned}\{&block(1,1) \wedge block(2,1) \Rightarrow block(2,2),\\ &block(2,1) \wedge block(3,1) \Rightarrow block(3,2),\\ &block(2,2) \wedge block(3,2) \Rightarrow block(3,3)\}\end{aligned} \tag{6.19}$$

while the local formulae

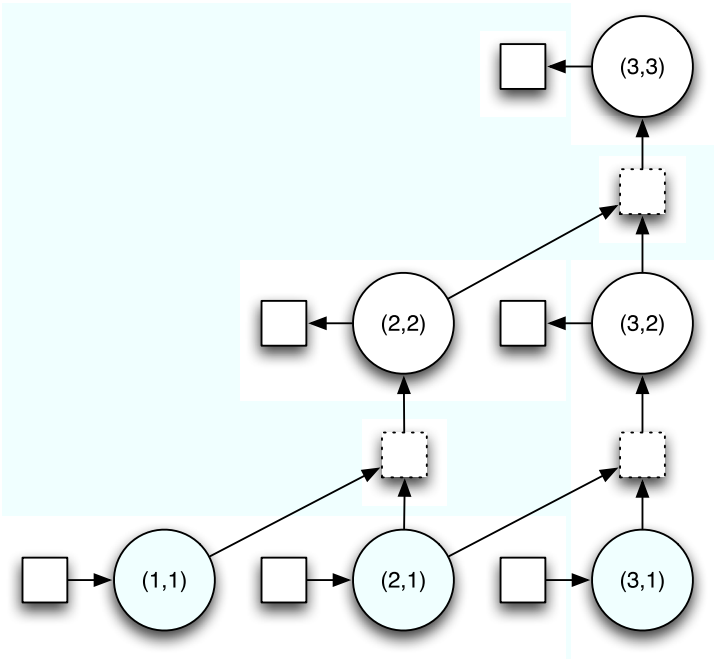
$$\begin{aligned}\{&block(1,1), block(2,1), block(3,1),\\ &\neg block(2,2), \neg block(3,2), \neg block(3,3)\}\end{aligned} \tag{6.20}$$

are in G^0. Then the corresponding graph is shown in figure 6.9.

Based on this graph we can now present a lower bound on the iteration in which an atom first becomes true or false. This bound can then be easily plugged into theorem 6.1 to allow us to make statements about the minimal number of iterations. However, first we need to introduce the notion of incoming and outgoing support paths in this graph.

Definition 6.4. An *incoming (outgoing) support path of an atom a* is a directed path in the clause graph that ends (begins) in a and that starts at a clause with no negated (non-negated) atoms. A *support path of a literal l* is the incoming path of l if l is an atom and the outgoing path of $\neg l$ if l is a negated atom. The set of all support paths of a literal l is referred to as $sup(l)$. The length $len(p)$ of a support path p is the number of non-initial clauses (dashed squares) it passes.

[19] Note that this graph is very similar to the factor graphs we introduced in chapter 2.



(a) clause graph

Figure 6.9: A graphical representation of clauses in equation 6.19 and 6.20. Each circle represents an atom and each square a clause. An atom is a member of clause if there is directed edge from the corresponding square to the circle of the atom. If the edge points from the atom to the clause the atom appears in the clause as negated atom. Dashed clauses are not initially instantiated.

For example, in figure 6.10 b) we can see an incoming support path of *block*(3,3) and an outgoing support path of *block*(3,1) (which happens to be the support path of $\neg block\,(3,1)$).

We will first show that if a literal l has no support path and all partial networks have a unique MAP solution, then l will never become true.[20]

Lemma 6.2. *Let l be a ground literal for which the corresponding ground atom appears in the final network G^* generated by CPI with an exact base solver. Further assume that every MAP solution during CPI is unique. If $sup\,(l)$ is empty then l will never be true in any MAP solution found during CPI with exact base solver.*

Proof. By contradiction: let A be the set of atoms which appear on any directed path that ends in atom a. We show that for any solution $\mathbf{y}$ with $a \in \mathbf{y}$ the solution $\mathbf{y}' = \mathbf{y}/A$ has a score at least as high as $\mathbf{y}$ and hence $\mathbf{y}$ cannot be a unique MAP. The only clauses that can become false when removing the atoms A are those that contain some atoms in A. However, all these clauses also contain negated atoms (otherwise there would be an incoming support path of a) and the corresponding atoms must be in A. Since they are also false in $\mathbf{y}'$ no clause that was not violated in $\mathbf{y}$ can be violated in $\mathbf{y}'$ and the score of $\mathbf{y}'$ is at least as high as the score of $\mathbf{y}$. In the case of $\neg a$ we can proceed analogously. □

Now we can focus our attention on those literals that have support paths.

Lemma 6.3. *Let l be a ground literal and $first\,(l)$ the iteration where l becomes true for the first time. If every MAP solution during CPI is unique and exact, then*

$$first\,(l) \geq \min_{p \in sup(l)} len\,(p) + 1$$

Proof. By induction for minimal path length d. $d = 0$: the bound is trivially true because a cannot become true before the first iteration. $d \rightarrow d+1$: assume the theorem holds for all atoms with minimal incoming support path length d. Assume an atom a with minimal incoming length $d+1$. This atom can only be true in a solution $\mathbf{y}$ if there is at least one clause c with $a \in literals\,(c)$ and all literals $b \in literals\,(c) \setminus \{a\}$ are false in $\mathbf{y}$; otherwise for the solution $\mathbf{y}' = \mathbf{y} \setminus \{a\}$ all clauses that are true in $\mathbf{y}$ are also true in $\mathbf{y}'$ and therefore it has at least the same score as $\mathbf{y}$. Since all incoming paths of a have a length of at least $d+1$, for each clause c with $a \in literals\,(c)$ there must be

[20] Note that this observation is closely related to theorem 6.3: if an atom cannot become true then all clauses that contain negations of this atom cannot be violated and will not be instantiated.

at least one atom a' with $\neg a \in literals(c)$ for which all incoming paths have a length of at least d. Hence a' cannot become true before $d+1$ iterations and c cannot be violated/instantiated before $d+1$ iterations because $\neg a' \in c$. Therefore c cannot exist and a cannot become true before $d+2$ iterations. The case of $\neg a$ can be approached analogously. □

Now we can easily prove the following theorem.

Theorem 6.5. *Let L be a positive MLN, V a vocabulary, $\mathbf{x}$ an observation, G^0 an initial grounding, BS an exact base solver and $G^* \subseteq G_{L,V,\mathbf{x}}$ the final network generated by Cutting Plane Inference with L, $\mathbf{x}$, G^0 and BS as input parameters. Let G^* be clausal. Then for the iteration $\iota(G^*)$ in which G^* is completely instantiated holds*

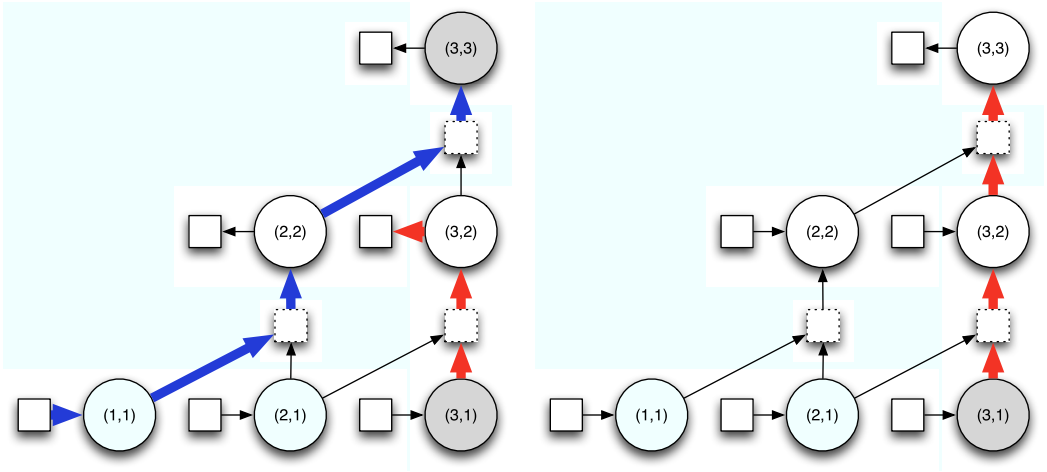
$$\iota(G^*) \geq \max_{g \in G^*} \max_{l \in literals(g)} \min_{p \in sup(\neg l)} len(p) + 1$$

Proof. From lemma 6.1 we follow that a formula $g \in G^*$ cannot become false before each literal $l \in literals(g)$ first becomes false. From lemma 6.2 we follow that this can only happen if the negated literals have a support path, and hence $sup(\neg l)$ is non-empty and l will not become false before $len(p)+1$ iterations for all $p \in sub(\neg l)$ because of lemma 6.3. □

In other words, theorem 6.5 says that by adding one to the maximum of the lengths of all minimal support paths of all literals appearing in G^* we get a lower bound on the number of iterations needed to instantiate G^*.

Assume we want to instantiate the ground formulae G^* in figure 6.10 a). If we consider the topmost clause we notice that *block*(3,3) is a member of this clause and has a minimal support path length of 2 (one such path is shown with bold lines). We also notice that $\neg block(3,2)$ is a member of this clause, and has a minimal support path length of 0. If we proceed in the same fashion for all literals of all clauses in G^* we find that the maximum length of all minimal support paths is 2 and hence the minimal number of iterations needed to instantiate G^* is 3.

In figure b) we have changed the sign of the ground unit clauses in the middle layer: now G^* contains $\neg$*block*(2,2) and $\neg$*block*(3,2). Now *block*(3,3) has a shorter support path of length 1. However, *block*(3,1) has lost its previous shortest support path of length 1; now its shortest support path has length 2 (shown with bold lines starting in *block*(3,1)). Hence we find that the maximum length of all minimal support paths is again 2; this results in at least 3 CPI iterations.



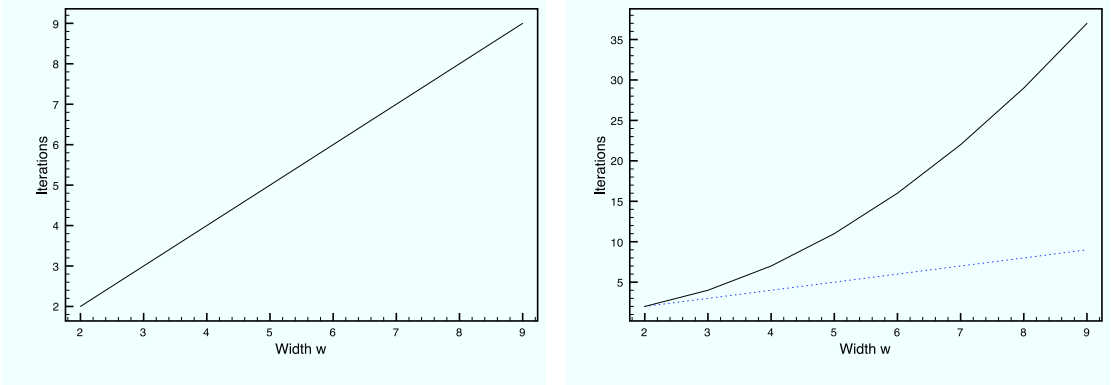
(a) negative middle blocks (b) positive middle blocks

Figure 6.10: Figure a) shows an incoming path of *block*(3,3) and an outgoing path of *block*(3,1). Figure b) shows an outgoing path of *block*(3,1) when the middle blocks are positive.

We can illustrate this behaviour empirically by setting up a similar experiment to the one in section 6.4.1.[21] However, this time we assign so much weight to the positive atoms in the bottom row that we are guaranteed to instantiate the full triangle. This leads to instantiations G^* similar to the one in figure 6.9. Based on figure 6.10 a) we know that in a triangle with width and height w, where all blocks but the bottom ones have negative weights, the length of all support paths of the topmost ground atom is $w-1$. This is also the maximum of all minimal support path lengths in G^*. Hence we need at least w iterations to instantiate G^* according to theorem 6.5.

In figure 6.11 a) we see that in this case the bound is tight. CPI needs exactly w iterations to instantiate the complete triangle of width w. To illustrate that this is not always the case we change the weights of the blocks inside the triangle to be all positive, with the exception of the topmost block. Again we make sure that the complete triangle will be instantiated. In this case our graphs and paths look like the ones depicted in 6.10 b). Here the triangle formulae between the bottom row and the second row can only be violated if the blocks on the second row are false. For a corresponding triangle with width w the bound 6.5 predicts that this cannot happen before $w-1$ iterations. Therefore CPI will again need at least w iterations to finish. However, as figure 6.11 b) shows, this time CPI does need significantly more iterations the larger

[21] Note that we again generate weights randomly (while still making sure that the complete triangle will be instantiated) and average over 10 cases.



(a) Only bottom row is positive

(b) only top block is negative

Figure 6.11: Number of iterations with increasing triangle width w. In figure a) we see the number of iterations for the case where only the bottom row blocks are positive. The solid black line shows both the actual number of iterations and the lower bound based on when atoms right below the top block can become true the first time. In figure b) we see the number of iterations for the case where all atoms are positive except for the topmost atom of the triangle. The solid black line shows the actual number of iterations while the dashed line represents the lower bound based on when atoms on the second row can become false for the first time.

the triangle becomes.

Why is this bound not tight in the case of figure 6.11 b)? Let us look at figure 6.10 b) to understand how CPI proceeds in this scenario. In the first iteration the topmost triangle formula is violated and will be instantiated. In the next iteration we need to satisfy this formula (assuming that it has a high weight) and if the topmost block has a very negative weight, the optimal choice will be to remove either *block*(2,2) or *block*(3,2). Say we remove *block*(2,2), then this will lead to the violation of the formula on the bottom left of the triangle; this formula is in turn added to the partial network. If both *block*(1,1) and *block*(2,1) have a very high weight, then its best to satisfy the triangle formula by activating *block*(2,2) again, and satisfying the topmost triangle formula by removing *block*(3,2). This will result in the violation and instantiation of the final formula on the bottom right side of the triangle. However, we needed three iterations while the bound 6.5 predicted 2.

This example shows that in the case of positive internal blocks we cannot instantiate the formulae in a bottom-up fashion. Instead we instantiate them one-by-one in a top-down manner—this leads to significantly more iterations. It will be important future

work to find out how this behaviour can be predicted using the graphical structure of the ground network.

In practice this bound tells us that CPI needs many iterations if certain atoms can only be explained by a "long proof". Let us again consider the Semantic Role Labelling task to illustrate this. Assume that the MAP solution for a problem with two candidate constituents contains the ground atoms *role*(1,A1) and *role*(2,C-A1), indicating that constituents 1 and 2 represent one discontinuous argument of type A1. However, the ground network does not contain the unit clause *role*(1,A1), the unit clause *role*(2,C-A1), nor the unit clause *hasRole*(1). The only positive unit clause it contains is *hasRole*(2), all other unit clauses are negated. In this case we can only prove *role*(1,A1) through the set of formulae

$$\{hasRole(2), \neg role(2,\text{A1}), hasRole(2) \Rightarrow role(2,\text{A1}) \vee role(\text{C-A1}), \quad role(2,\text{C-A1}) \Rightarrow role(1,\text{A1})\} \tag{6.21}$$

(note that we are assuming that A1 and C-A1 are the only possible argument labels). If we draw the clause graph for this example (see figure 6.12) we will see that the ground clauses

$$hasRole(2) \Rightarrow role(2,\text{A1}) \vee role(2,\text{C-A1}) \tag{6.22}$$

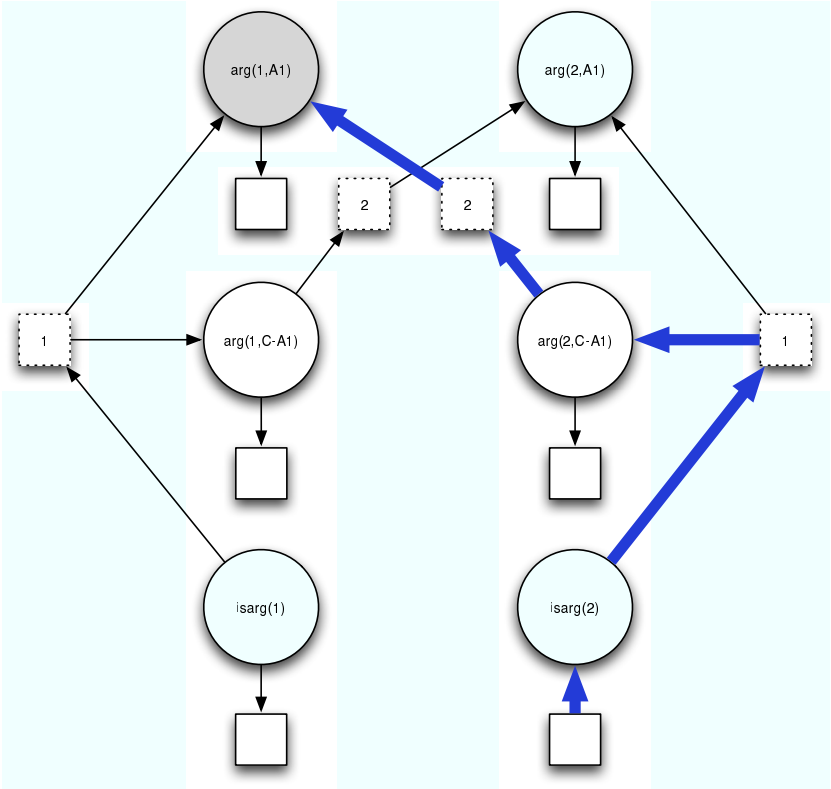
and

$$role(2,\text{C-A1}) \Rightarrow role(1,\text{A1}) \tag{6.23}$$

appear on the only path that starts at a positive ground atom and ends in *role*(1,A1). Both are not in the initial ground network, and hence the length of the corresponding support path is 2. Therefore theorem 6.5 tell us that CPI will at least need 3 iterations. Note that we only see very few iterations (2.2 iterations on average) for CPI-ILP in our SRL application in section 5.3. This tells us that our model has short explanations for all semantic arguments.

6.6 Upper Bound on Number of Iterations

In the previous section we have seen that the number of iterations that CPI will at least need to instantiate a set of formula $\mathbf{G}$ depends on when atoms can first become true or false. And we have seen that the iteration in which an atom can become first true or false depends on the minimal length of incoming and outgoing paths. We could also show that the number of iterations CPI will at most need depends on the iteration in



(a) clause graph

Figure 6.12: A graphical representation of the clauses for a simple SRL problem. The clauses (squares) denoted by the number 1 imply that there has to be a semantic label if a constituent is a semantic argument according to the *hasRole* predicate. The clauses denoted by the number 2 require that there is an constituent labelled as A1 if some constituent is labelled as C-A1. The only incoming path of *role*(1,A1) that starts at a clause with no negated atoms is marked by bold (blue) arrows.

which atoms change their truth state for the final time. However, bounding the iteration when an atom can become true or false for the last time turned out to be significantly more difficult—it not only depends on the length of incoming or outgoing paths of an atom but also on the interaction between these paths.

We will therefore use this section to present a simpler bound on the number of CPI iterations. It is easy to see that the number of iterations cannot be higher than the number of ground formulae in $\mathbf{G}$ because CPI adds at least one formula in each iteration—by definition it terminates if no formula is added. But we can also say more: if $\mathbf{G}$ contains "islands" of disconnected ground formulae then the number of iterations is bounded by the maximum number of formulae these islands consist of. This indicates that CPI processes disconnected partial networks in parallel. Formally we write

Theorem 6.6. *Let L be a positive MLN, V a vocabulary, $\mathbf{x}$ an observation, G^0 an initial grounding, BS an exact base solver and $G^* \subseteq G_{L,V,\mathbf{x}}$ the final network generated by Cutting Plane Inference with L, $\mathbf{x}$, G^0 and BS as input parameters. Let $M = \{M_1, \ldots, M_n\}$ be a partition of G^* so that for every $i \neq j$ and for all $g_i \in M_i$ and $g_j \in M_j$ $atoms(g_i) \cap atoms(g_j) = \emptyset$ then for the iteration $\iota(G^*)$ in which G^* is completely instantiated holds*

$$\iota(G^*) \leq \max_i \left\| M_i \setminus G^0 \right\|$$

Proof. Each M_i is an independent network. Once no more formulae are instantiated for a certain M_i, the optimum $\mathbf{y}_i$ for this partial network will remain constant and no more formulae can be instantiated at any later point. Hence in order to fully instantiate M_i there needs to be at least one formulae instantiated in every iteration. □

We illustrate this bound by setting up a MLN that results in several disconnected triangles in $\mathbf{G}$. This is achieved by placing a set of p triangles of height and width 6 next to each other, as shown in figure 6.13. When we increase the number of triangles we increase total number of ground formulae in the complete ground network $\mathbf{G}$; however, the number of formulae in each island remains constant. Therefore we expect the number of iterations to remain constant, too. Note that the complete ground networks we would have to generate for propositional MAP inference are *not* disconnected—the islands only appear because CPI does not need to instantiate the complete graph.

The results can be seen in figure 6.14: as expected, the number of iterations remains constant. However, note that we do need less than the number of ground formulae

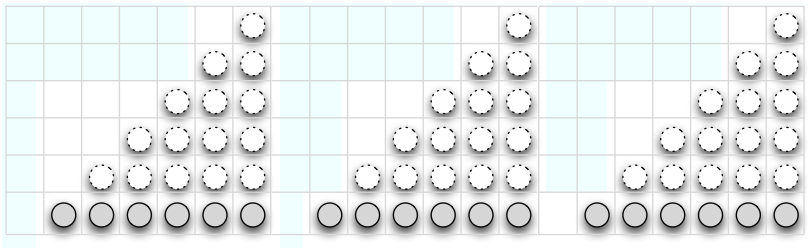


Figure 6.13: A single grid with several triangles.

in each island, as shown through the gap between the dotted line (upper bound) and the solid black line (actual number of iterations). The dashed lines shows how many ground formulae exist in the **G**. The gap between the solid black line and the dashed line shows that the number of iterations can be completely independent of the actual number of ground formulae we have to instantiate.

Island effects can appear in real world applications. For example, consider the Entity Resolution task in section 5.4, where we had to find pairs of citations that refer to the same publication. The corresponding MLN contained a formula

$$\begin{aligned} &title(c_1,t_1) \wedge title(c_2,t_2) \wedge similarTitle20(t_1,t_2) \\ &\Rightarrow sameCitation(c_1,c_2) \end{aligned} \tag{6.24}$$

which implies that two citations match if their titles have a TF-IDF distance between 0% and 20%. This formula has a negative weight for which the absolute value is an order of magnitude larger than any of the positive weights assigned to the formulae that test other local properties of a potential match. This means that if two titles are not similar at all, the penalty we have to take for this will dominate the reward from all other local evidence. As a result the summarized initial ground unit clauses will not contain positive $sameCitation(C_1,C_2)$ clauses for citation pairs (C_1,C_2) that have very different titles.

We can think of the positive unit clauses $sameCitation(C_1,C_2)$ as edges between citations in a matching graph (see figure 6.15). If this graph contains disconnected islands then the final partial network will also contain islands (because we can never disprove the transitivity formulae that connect citations from different matching graph islands). And by looking at the data we observe that the initial matching graph indeed contains many islands—on the 10 Cora folds described in section 5.4 we count 10.5 islands on average.

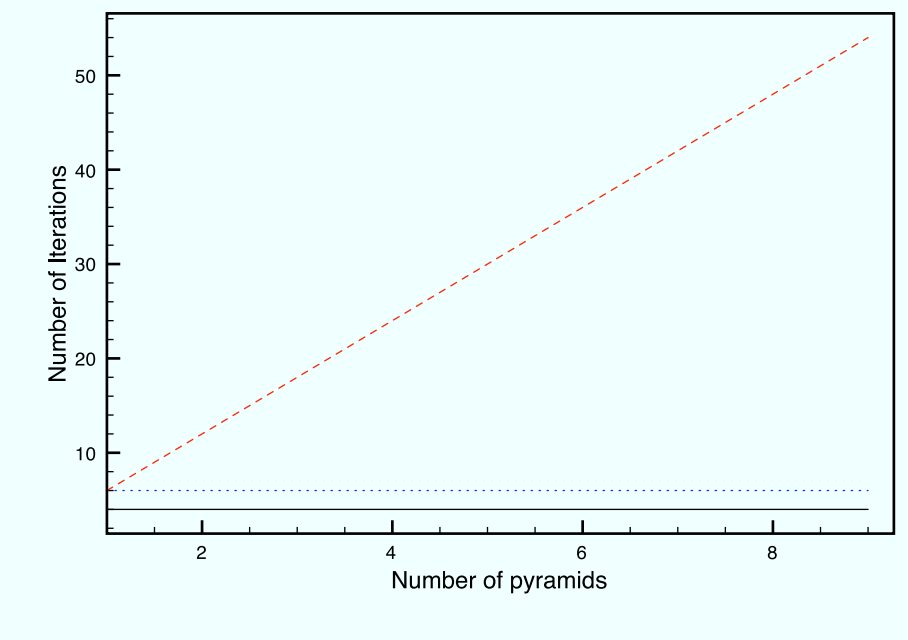


Figure 6.14: Number of iterations with increasing number of triangles of size 5. The solid black line denotes the actual number of iterations, the dotted line is the number of ground formulae in each triangle and the dashed line denotes the number of ground formulae in the final partial network $\mathbf{G}^*$.

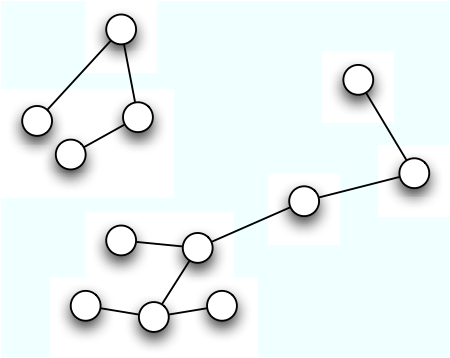


Figure 6.15: A graph that connects citation pairs if there exists a positive *sameCitation*(C_1, C_2) clause in the summarized initial ground network.

6.7 CPI with Approximate Base Solver

All previous observations are based on the assumption that the base solver for CPI is exact. If we lift this requirement the previous theorems do not hold anymore because they make use of the fact that the current CPI solution is truly optimal. For example, with an approximate solver such as MaxWalkSAT we do not need to disprove a formula in order to instantiate it. It could be "randomly" violated just because the solver performed randomized operations.

Yet, intuitively we would expect that the closer the accuracy of an approximate solver gets to the accuracy of an exact solver, the more similar the solutions in each iteration will be. And because the theorems in this chapter are based on properties of these solutions, we believe that the more accurate an approximate base solver is, the closer the behaviour CPI for this base solver will be to the behaviour of exact CPI. It will be important future work to thoroughly analyse approximate CPI because there will likely be cases where exact CPI is bound to be impractical.

6.8 Conclusion

In this chapter we presented a theoretical analysis of the runtime behaviour of CPI and illustrated our findings using a synthetic problem and examples taken from chapter 5. We have shown the following properties of CPI:

- The partial networks generated during CPI will be large if our Markov Logic Network contains formulae that are frequently violated in the initial solution (and this largely depends on the sign of weights). Likewise, the final network will be large if many formulae are violated in the MAP solution.

- The partial networks generated during CPI are guaranteed to be small if many ground formulae cannot be disproven using the set of all ground formulae. The partial networks will also be small if the MLN allocates a large amount of weight for formulae in the initial grounding that contradict potential refutations of the formulae to be instantiated.

- CPI will need many iterations if ground literals have long incoming or outgoing paths in a directed graphical representation of the ground network. Here the length of a path roughly corresponds to the number of explanation steps that are necessary to show why a literal is true or false.

- CPI shows a parallel behaviour if the final ground network consists of disconnected islands.

Knowing such boundary conditions can be very useful in practice. For example, if the user observes that CPI is slow for a given MLN, he or she can check whether the MLN has some of the properties that are guaranteed to lead to large partial ground networks according to our findings in section 6.3. There might be a formula that is violated often in the initial solution and has a positive weight. By enforcing that the corresponding weight is nonpositive, it is possible (but not guaranteed) that partial networks become significantly smaller. If CPI is still slow, the user can try to adapt the model even further and ensure that it has properties that will definitely lead to small partial networks, for example by adding more local formulae that can help to contradict violations of global formulae.

Likewise, if CPI needs many iterations, it may be possible that the model can only explain certain facts through long explanations that involve several non-initial formulae. Again, if we add more local formulae (or if we extend the initial grounding) then shorter explanations of these facts might arise. This certainly does not guarantee a lower number of iterations, but it makes a lower number possible. Moreover, it may again help us to create smaller ground networks that contain disconnected islands and hence lead to a more parallel behaviour with fewer iterations.

Chapter 7

Conclusion

In this chapter we will first summarise the contributions of this thesis and the insights gained. Then we present directions for further research.

7.1 Contributions

Let us begin by presenting the core contributions of this thesis. We will divide this section into three parts. The first two correspond to our two core contributions: Cutting Plane Inference, and a theoretical and empirical analysis of its runtime behaviour. The final section describes supplementary contributions of this thesis.

7.1.1 Cutting Plane Inference

The first core contribution of this thesis is presented in chapter 4. We introduce a novel meta-algorithm, namely Cutting Plane Inference (CPI), for Maximum *a posteriori* Inference in Markov Logic. Cutting Plane Inference iteratively solves small sub-problems of the full ground network using a propositional base solver and adds ground formulae that are not yet included in the current network but could potentially lead to an improved solution. The process of finding ground formulae to be added (separation) is implemented through a first order query processing engine, which finds all groundings of a first order formulae which are satisfied (or not satisfied) in the current solution. Hence we can see CPI as hybrid method that combines numeric optimisation and query processing—this allows us to draw from the best of both worlds and shows one of the benefits of combining probabilistic models and logic.

We prove that CPI has an important theoretical property: its accuracy is essentially

the accuracy of the used base method, but on a smaller problem. In particular, this means that CPI performs exact inference if the base solver performs exact inference. For example, this means that if we use ILP as base method we are guaranteed to find the optimal solution.

CPI is novel from the following point of view: it is the first MAP inference method for Markov Logic that, by avoiding the full instantiation of the Ground Markov Network, not only improves memory efficiency (as LazySAT [Singla and Domingos, 2006b] does) but also speed and accuracy of inference. Moreover, in contrast to LazySAT, which is essentially a memory-efficient implementation of MaxWalkSAT, Cutting Plane Inference can be combined with any given method for MAP inference in propositional Markov Networks.

When compared to Lifted Belief Propagation [Singla and Domingos, 2008], a recently introduced method that improves the speed of marginal inference by working in a much simpler "lifted version" of the complete Ground Markov Network, we argued that CPI exploits a different type of redundancies in Ground Markov Networks. While Lifted Belief Propagation makes use of the repetitive and regular nature of many networks, CPI's efficiency is based on the fact that many factors in a network reward or punish the same properties of a solution.

CPI is also novel when we compare it to other methods that use a Cutting Plane algorithm for MAP inference. In the methods proposed by Sontag and Jaakkola [2007] and Sontag et al. [2008] MAP inference in Markov Networks is also performed with the help of Cutting Planes. However, these methods still require us to explicitly construct the complete network and do not take first order information into account. Moreover, in both cases separation costs increase substantially when the Markov Network gets larger. In problems such as the Entity Resolution task we presented in chapter 5 separation is therefore likely to be difficult. In contrast, the separation routine in CPI exploits first order information and is not directly influenced by the actual network size.

Moreover, instead of incrementally building a propositional Markov Network, the methods of Sontag et al. iteratively tighten outer bounds on the Marginal Polytope. This means that in contrast to CPI these methods cannot simply use another existing method for MAP inference for Markov Networks in order to solve the optimization problem in each iteration.

Compared to work [Tromble and Eisner, 2006, Riedel and Clarke, 2006, Anguelov et al., 2004] which is not based on the Marginal Polytope we argued that CPI has the

following advantages. Firstly, it provides a generic separation routine that requires no additional implementation effort for new models. Secondly, it supports soft constraints and compared to the work of Tromble and Eisner 2006 it does not require expensive branch-and-bound operations to do so. Finally, it is the only Cutting Plane algorithm that is framed in terms of a Statistical Relational Learning language—this makes it directly applicable to a large set of relational problems.

In chapter 5 we state that CPI can significantly improve the efficiency and accuracy of existing propositional methods. This hypothesis is evaluated using two tasks: Semantic Role Labelling and Entity Resolution (a task very similar to Coreference Resolution). In the case of Semantic Role Labelling we present a Markov Logic Network, inspired by previous work in the domain, that achieves state-of-the-art accuracy. In the case of Entity Resolution we re-use a Markov Logic Network from previous work

In both cases we show that CPI increases the speed of two existing methods by two orders of magnitude and the memory usage by at least 70%. Moreover, we show that by using CPI we are also able to find optimal solutions if the base solver is exact (in line with our theoretical findings) and more probable solutions when the base solver is approximate (while still being reasonably accurate). In particular, CPI allowed us to find a the optimal solution in network with 250.000 nodes and millions of edges while the base solver failed to even process the problem.

We also show that not only does CPI perform faster on average, it can scale significantly better than the base method. In both applications we show that CPI's efficiency scales linearly with problem size (that is, the number of binary decisions to make). By contrast, the base solver becomes over-proportionally slower with larger problems. This makes CPI a good candidate for jointly tackling several NLP tasks, or large NLP tasks such as Machine Translation or Parsing, which are currently out of scope for generic inference methods.

Generally speaking, we believe that the introduction of CPI will lead to a more widespread use of Statistical Relational Learning in NLP. Languages such as Markov Logic allow researchers to focus on the modelling of the problem instead of the development of inference/search or learning methods. This attractive property is often offset by the fact that generic inference is often either slow/infeasible or inaccurate. With CPI more Markov Logic models become tractable, both in terms of inference, and in terms of training (through online learning). This makes Markov Logic (or any Statistical Relational Learning language for which we implement CPI) more attractive.

Finally, we presented a variant of Cutting Plane Inference that processes formulae in a user-defined order that can reduce the workload of the base solver.

7.1.2 Empirical and Theoretical Runtime Analysis

The second major contribution is the analysis of the runtime behaviour of the proposed Cutting Plane Inference algorithm both in theoretical and practical terms. To our knowledge this work is the first which asks and answers questions about how the size of partial problems and number of iterations of a Cutting Plane Algorithm for MAP inference depends on the input problem.

We first stressed, both theoretically and empirically, an important property of CPI: CPI will create large partial problems, and therefore offset its potential advantage of using the base solver only, if many formulae are violated (assuming positive weight) or not violated (assuming negative weight) in the first solution. This leads to a behaviour that is strongly correlated with the sign of the weights of formulae, and significantly reduces the set of models that can be efficiently processed through CPI. Surely the above observation also holds for the solutions of all intermediate iterations of CPI. However, the nature of these solutions is often hard to predict without solving complex optimisation problems. In contrast, the first solution can often be trivially calculated (in particular if all initial formulae are local).

We also pointed out that the final partial problem will be large if there are many violated formulae in the true MAP solution. Again we could argue that in order to apply this finding in practice we would need to solve the MAP problem before we can (lower) bound the size of the final partial problem. However, often we have a clear picture about what the MAP solution *should* be if our model was reasonably accurate: it should be close to the true gold solution. Hence this observation allows us to make claims such as “if the model is accurate and contains many formulae that are frequently violated in gold solutions, CPI will likely not work well/create large problems”. This then could be understood as “don’t use CPI here because what good is it to not use a good model?”

Our second observation is that CPI problems are guaranteed to be small if the set of all ground formulae have low “expressive power”. That is, if the violation (non-violation) of a formula cannot be proven by some of the remaining ground formulae in the complete Ground Markov Networks, it will not be instantiated. In many cases (in particular if all ground formulae are Horn) this means that if the set of positive ground

unit clauses (usually the result of summarising local formulae) is small, the final partial problem will be small, too.

Moreover, partial problems are also guaranteed to be small if the model has a low of confidence (in the form of weight) in those ground formulae that can disprove (prove) other ground formulae, and a high confidence in those ground formulae that contradict these proofs.

We also introduced a lower bound on the number of iterations that will be required for solving a problem with CPI. This bound is formulated in terms of a measure of length in a directed graphical representation of a clausal Ground Markov Network. More abstractly, we can understand this bound as follows: if atoms states (either true or false) can only be explained/proved in a large number of steps/ground formulae, CPI will need many iterations.

Finally, we gave evidence that the number of iterations maximally needed depends on the size of independent clusters in the final network we generate. Hence CPI can be seen as partly parallel way to solve the MAP problem. It also means that CPI takes into account the "effective" independence of variables, not only based on the graphical structure, but also based on the cost/weight structure of the problem.

We hope that these findings will (a) help us to understand when CPI works or does not work, (b) help us to constrain the model structure and parameters of CPI in cases it does not work and (c) show where CPI needs to be improved.

7.1.3 Supplementary Contributions

In this thesis (and in [Riedel, 2008]) we also introduced a compact mapping from binary Markov networks to Integer Linear Programs. This mapping uses significantly fewer variables (while yielding the same amount of constraints in our applications) as the conventional representation based on the Marginal Polytope. Moreover, the work presented here is fact the first that investigates Integer Linear Programming (ILP) for Inference in Markov Logic. We believe that ILP is an important addition to the line of MAP methods for Markov Logic, both as base solver for CPI and in isolation, because ILP is exact and can often quickly find MAP solutions for non-trivial problems. Moreover, even if ILP may not scale up as well as MaxWalkSAT or other heuristic solvers, it can help to inspect and validate their performance.

We have also presented the first Markov Logic Network for Semantic Role Labelling. This network leads to very competitive performance, yielding results better

than the best single-parse systems of the CoNLL Shared Task 2005. However, the performance of this MLN is not its only merit. Compared to its competitors, our MLN is much simpler. Others need to train classifiers, connect argument identification and classification components and integrate results with n-best rerankers or ILP solver (that require programs to generate ILPs). By contrast, we simply define a set of intuitive First Order Logic formulae and let the Markov Logic interpreter do the rest. This also means that we can provide a very compact representation of our complete SRL system in the form of our MLN file. This representation can be easily inspected, extended, or used as basis for reproducing our results.

7.2 Future Work

This work provides several possibilities for follow-up research we will present in the following sections.

7.2.1 Reduce Sensitivity to Sign of Weights

One of the main shortcomings of CPI is its sensitivity with respect to the sign of weights (cf. section 6.3). Just by simply changing the sign of a formula from being positive to negative, or vice versa, the size of the partial problems can change from "only a few ground formulae" to "almost all ground formulae". In such cases it is likely to be more efficient to just use the base solver on its own.

This restriction does not always pose a problem to the user. For example, in the case of Semantic Role Labelling the constraint that forbids more than one agent for a single predicate has a high positive weight. Changing this weight to be negative can easily lead to very large ground networks that would cause CPI to be slow. However, assigning a negative weight to such a formula seems intuitively wrong, as it would encourage the opposite of what know about the problem. This suggest that while a large set of Markov Logic Networks might not be solvable with CPI, many of these networks may not be *desirable* or useful, either.

Nevertheless, in the case of formulae with low positive or negative weight the situation is not as obvious and one cannot say whether a weight should be positive or negative based on the nature of the formula alone. In such cases it is the interaction with other formulae that determines the weight (and its sign) of a formula.

It might be possible to transform Markov Logic Networks that cannot be solved

with CPI into equivalent, or near-equivalent, networks which can be solved. Moreover, while wrong signs may lead to large networks, it is possible that these networks are highly regular and repetitive. In this case it could be helpful to integrate Lifted Belief Propagation [Singla and Domingos, 2008] into the Cutting Plane Inference framework. There are at least two ways of doing so. One way is to run CPI and use Max-Product Lifted Belief Propagation (or any other lifted MAP algorithm) as base solver. The other way is to first create a lifted network that leads to the same results as the original network for a given MAP solver X. Then we perform CPI with base solver X for this lifted network. Future work could investigate the relative merits of these two approaches.

7.2.2 Tighter Upper Bounds on Number of Iterations

We believe that the upper bound on the number of iterations can be further improved. It seems that the maximum number of iterations can be more accurately determined if we not only take the explanation length into account (as in the case of the lower bound on iterations) but also the number of explanations. In ongoing experiments we have observed that during CPI different explanations are instantiated, and whenever these explanations interfere (explain the same ground literal) formulae are instantiated in serial/one-by-one fashion.

It will also be worthwhile to thoroughly analyse if and how the fact that weights in a Markov Network are trained (as opposed to random or user-specified) impacts the efficiency of inference. For example, intuitively we expect trained local weights to more likely encourage solutions consistent with a set of global hard constraints than random local weights—they have been trained to encourage gold solutions that are consistent with these constraints. However, so far no theoretical or empirical support for this claim has been given.[1]

7.2.3 Use Cutting Plane Algorithm as Base Solver

We have used ILP and MaxWalkSAT as base solver in our experiments. ILP has the advantage of guaranteed exact inference. However, in ongoing experiments we have found that in certain situations ILP becomes slow even if the problems generated during CPI are small and only few iterations are needed. In such cases the ILP solver

[1] However, during learning we have often observed that a model becomes faster the longer we have trained its weights.

seems to perform a large amount of internal branch and bound operations.

An alternative to ILP could be the MAP version of the Cutting Plane algorithm of Sontag and Jaakkola [2007] or the algorithm of Sontag et al. [2008]. These methods are also guaranteed to be optimal, but may be significantly faster.

Note that some ILP solvers also use Cutting Plane algorithms internally, but in contrast to the work of Sontag and Jaakkola [2007] and Sontag et al. [2008] they are not specialised for MAP inference. They are also not supported by the open source ILP solver we use.

7.2.4 New Applications

It is often straightforward to formulate a Markov Logic Network for a new task. However, due to the complexity of such models it will sometimes be impossible to train this network efficiently, or run inference in it. As mentioned in section 7.1 we believe that CPI will increase the set of problems we both model and efficiently process within Markov Logic. It will be be interesting to further explore this set and evaluate the utility of Markov Logic and CPI. It might be possible to look at larger problems, such as Machine Translation, Parsing, or tasks that involve several of such NLP sub-tasks.

The work of Riedel and Meza-Ruiz [2008] can be seen as step into this direction. They present a Markov Logic Network that models the complete Semantic Role Labelling process, jointly labelling several arguments of several predicates as well as predicate sense and frame. This system achieved state of the art performance and won CoNLL 2008 best runner-up (including best results for out of domain results).

7.2.5 Interaction between Learning and CPI

We have mentioned that the performance of CPI depends on the weights of formulae. The behaviour of CPI changes dramatically when weights change their sign, and the performance is also affected by how much weight is assigned to explanations of formula violations. This leads to the question: can we design a training method that ensures that the model is accurate *and* still efficient to process with CPI? This question is related to work by Daumé III [2006] who presents an approach to structured prediction that tightly couples search and inference.

This also leads to a more theoretical question: what is the the expressiveness of the class of Markov Logic Networks that can be efficiently processed with CPI. Are

there Markov Logic Networks for which no CPI-friendly Markov Logic Networks with similar performance can be created or learnt?

7.2.6 Inferring Marginal Probabilities

In this work we have only tried to find the most likely set of ground atoms. However, in many cases users need to infer the marginal probability that a ground atom is true. For example, it may be that the most likely semantic role for a given constituent is A1 but it is almost equally likely that the constituent has the role A2. In this case all later components should handle the provided role label with caution.

A simple way of extending CPI to calculate marginal probabilities is the following. We could run a base-solver for marginal inference on a sub-network and find all ground atoms with probability close to zero/one, based on some probability threshold. These atoms are then stipulated to be false/true. In turn, all ground formulae that are either violated or cannot be proven by these atoms are added to the network. We then call the propositional base solver again, and continue just as in CPI.

The intuition behind this approach is that the satisfied ground formulae which are not yet added would only encourage the atoms to remain in their current "clipped" state. Hence, by adding them, the deterministic marginals would only get more deterministic, and the amount of probability mass that could be used to change other marginals is limited by the remaining uncertainty of the deterministic marginals. This bounds the error that is made by not including the satisfied formulae.

The Cutting Plane algorithm introduced by Sontag and Jaakkola [2007] may also be suitable for marginal inference in Markov Logic. However, it might be difficult to apply it to networks with a large number of edges and nodes, such as the one in our Entity Resolution application. An interesting area for further research is therefore to combine the ideas from our work, i.e. a separation routine that exploits the information provided by a first order model of the network, with their Cutting Plane Algorithm in the Marginal Polytope.

7.2.7 Continuous Domains

Many real-world applications require not only Boolean variables and features but also numerical ones. For example, in robotic map building we have to infer the positions of wall segments, along with their types and relations to other segments. To adequately model this problem, Markov Logic Networks have been extended to *Hybrid*

Markov Logic Networks [HMLN, Wang and Domingos, 2008]. Here we can consider numerical properties and formulae along with their Boolean counterparts. An example numerical property would be $x(s)$ for the start x-position of a segment s. An example numerical formulae would be $start(s,l) \cdot (x(s) = x(l))$. This formula adds the value $-(x(s) - x(l))^2$ to the score of a possible world in case the wall segment s starts a line of segments l, and zero otherwise. For an explanation of the used notation we refer the reader to [Wang and Domingos, 2008].

One possible area of future work is the use of Mixed Integer Linear Programming (MILP) for Hybrid Markov Logic Networks. Mixed Integer Linear Programs seem particularly suited for Hybrid Markov Logic Networks since they can contain both discrete and real variables. However, we cannot use quadratic terms in the objective, and hence we would need to approximate such terms (for example by using piecewise linear approximations). Alternatively we could replace the MILP solver by a Mixed Integer Quadratic Programming solver; however, free/open source solvers of this type are hard to find.

We have seen how ILP did not scale up for the large problems we tackled in section 5.4, and for MILP we may observe the same behaviour. Hence CPI might also be helpful for Hybrid Markov Logic Networks, and would need to be extended for this scenario. Conceptually this is easy: we still only need to add those ground factors which do not maximally contribute to the objective (assuming that for each factor the maximal contribution is finite). For example, the maximum contribution of $start(s,l) \cdot (x(s) = x(l))$ is zero, and thus for a given solution $\mathbf{y}$ we would need to add those groundings of the formula for which both $start(s,t)$ and $x(s) \neq x(l)$holds in $\mathbf{y}$—only in this case the contribution of the corresponding ground factor would be smaller than zero.

However, while separation in the above example would be straight-forward (it is still a Boolean first order query evaluation problem), this cannot be said in general. One could simply test all variable assignments for each formula, but this would likely not scale up well. Future work on CPI for Hybrid Markov Logic Networks would therefore need to focus on efficient separation in the presence of numerical features.

7.2.8 Comparison with Lazy Inference

In section 4.7.4 of chapter 4 we compared Cutting Plane Inference to Lazy Inference [Poon et al., 2008] and showed that they are closely related. Lazy Inference

provides edges to the base solver when needed, CPI only once the solver has found the solution for the current network and the edges are not redundant with respect to this network. If the lazy version of base solver tends to activate such redundant nodes, CPI may have an advantage. If the base solver is aware of this redundancy, this advantage would disappear. However, how both algorithms compare in practice still remains to be seen. In future work we will therefore investigate the relative advantages and disadvantages of CPI compared to Lazy Inference when applied to real world MLNs.

Bibliography

D. Anguelov, D. Koller, P. Srinivasan, S. Thrun, H.-C. Pang, and J. Davis. The correlated correspondence algorithm for unsupervised registration of nonrigid surfaces. In *Advances in Neural Information Processing Systems (NIPS '04)*, pages 33–40, 2004.

David Applegate, Robert Bixby, Vašek Chvátal, William Cook, and Keld Helsgaun. Georgia tech tsp webpage for sweden, 2004. URL `http://www.tsp.gatech.edu/sweden/`.

Irad Ben-Gal. Bayesian networks. In F. Ruggeri, F. Faltin, and R. Kenett, editors, *Encyclopedia of Statistics in Quality and Reliability*. John Wiley & Sons, 2007.

Julian Besag. Statistical analysis of non-lattice data. *The Statistician*, 24(3):179–195, 1975.

M. Bilenko and R. Mooney. Adaptive duplicate detection using learnable string similarity measures. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '03)*, pages 39–48, 2003.

B. Borchers and J. Furman. A two-phase exact algorithm for MAX-SAT and weighted MAX-SAT problems. *Journal of Combinatorial Optimization*, 2:299–306, 1999.

Razvan Bunescu and Raymond J. Mooney. Collective information extraction with relational Markov networks. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL' 04)*, pages 439–446, 2004.

X. Carreras and L. Marquez. Introduction to the conll-2005 shared task: Semantic role labeling. In *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL' 05)*, pages 152–164, 2005.

Noam Chomsky. *Syntactic Structures*. Mouton, 1957.

James Clarke and Mirella Lapata. Modelling compression with discourse constraints. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '07)*, pages 1–11, 2007.

Michael Collins. Three generative, lexicalised models for statistical parsing. In *Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics (ACL' 97)*, pages 16–23, 1997.

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '02)*, volume 10, pages 1–8, 2002.

Michael Collins and Brian Roark. Incremental parsing with the perceptron algorithm. In *Proceedings of the 42nd Annual Meeting of the Association for Computational Linguistics (ACL' 04)*, pages 111–118, 2004.

Koby Crammer and Yoram Singer. Ultraconservative online algorithms for multiclass problems. *Journal of Machine Learning Research*, 3:951–991, 2003. ISSN 1533-7928.

Aron Culotta and Andrew McCallum. Joint deduplication of multiple record types in relational data. In *Proceedings of the 14th ACM international conference on Information and knowledge management (CIKM '05)*, pages 257–258, New York, NY, USA, 2005. ACM. ISBN 1-59593-140-6.

G. B. Dantzig, R. Fulkerson, and S. M. Johnson. Solution of a large-scale traveling salesman problem. *Operations Research*, 2:393–410, 1954.

Hugh Darwen and C. J. Date. The third manifesto. *SIGMOD Record*, 24(1):39–49, 1995.

Hal Daumé III. *Practical Structured Learning Techniques for Natural Language Processing*. PhD thesis, University of Southern California, Los Angeles, CA, August 2006.

R. de Salvo Braz, E. Amir, and D. Roth. Lifted first-order probabilistic inference. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence (IJCAI '05)*, pages 1319–1125, 2005.

P. Domingos. *Artificial Intelligence: The First Hundred Years*, chapter What's Missing in AI: The Interface Layer. AAAI Press, to appear.

Pedro Domingos. Structured machine learning: Ten problems for the next ten years. *Machine Learning*, 73(1):3–23, 2008.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL' 05)*, pages 363–370, June 2005.

Jörg Flum, Markus Frick, and Martin Grohe. Query evaluation via tree-decompositions. *Journal of the ACM (JACM)*, 49(6):716–752, 2002.

Stuart Geman and Donald Geman. Stochastic relaxation, gibbs distributions and the bayesian restoration of images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(6):721–741, November 1984.

Michael Genesereth and Nils J. Nilsson. *Logical Foundations of Artificial Intelligence*. Morgan Kaufmann, 1987.

Lise Getoor and Ben Taskar. *Introduction to Statistical Relational Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2007.

Daniel Gildea and Daniel Jurafsky. Automatic labeling of semantic roles. *Computional Linguistics*, 28(3):245–288, 2002. ISSN 0891-2017.

Amir Globerson and Tommi Jaakkola. Fixing max-product: Convergent message passing algorithms for map lp-relaxations. In *Advances in Neural Information Processing Systems (NIPS '07)*, pages 553–560, 2007.

P. Haddawy. Generating bayesian networks from probability logic knowledge bases. In R. Lopez de Mantaras and D. Poole, editors, *Proceedings of the 10th Annual Conference on Uncertainty in AI (UAI '94)*, pages 262–269. Morgan Kaufmann, 1994.

Joseph Y. Halpern. An analysis of first-order logics of probability. *Artificial Intelligence*, 46(3):311–350, 1990. ISSN 0004-3702.

Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. Developing a natural language interface to complex data. *ACM Transactions on Database Systems (TODS)*, 3(2):105–147, 1978. ISSN 0362-5915.

Timothy Hinrichs and Michael Genesereth. Herbrand logic. Technical report, Stanford Logic Group, 2006.

J. N. Hooker. Generalized resolution and cutting planes. *Annals of Operations Research*, 12(1-4):217–239, 1988. ISSN 0254-5330.

Tuyen N. Huynh and Raymond J. Mooney. Discriminative structure and parameter learning for markov logic networks. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 416–423, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-205-4.

Henry Kautz, Bart Selman, and Yueyen Jiang. A general stochastic approach to solving problems with hard and soft constraints. In D. Du, J. Gu, and P. Pardalos, editors, *The Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 573–586. American Mathematical Society, 1997.

K. Kersting and L. De Raedt. Towards combining inductive logic pro- gramming with bayesian networks. In *Proceedings of the Eleventh International Conference on Inductive Logic Programming*, pages 118—131, 2001.

Kristian Kersting and Luc De Raedt. Bayesian logic programs. In J. Cussens and A. Frisch, editors, *Proceedings of the Work-in-Progress Track at the 10th International Conference on Inductive Logic Programming*, pages 138–155, 2000.

S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

Stanley Kok and P. Domingos. Learning the structure of markov logic networks. In *Proceedings of the Twenty-Second International Conference on Machine Learning*, pages 441–448, 2005.

Stanley Kok, Parag Singla, Matthew Richardson, and Pedro Domingos. The Alchemy system for statistical relational AI. Technical report, University of Washington, 2005.

Daphne Koller. Probabilistic relational models. In S. Džeroski and P. Flach, editors, *Proceedings of the 9th International Conference on Inductive Logic Programming (ILP '99)*, volume 1634, pages 3–13. SV, 1999.

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the 18st International Conference on Machine Learning (ICML' 01)*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

Lillian Lee. "i'm sorry dave, i'm afraid i can't do that": Linguistics, statistics, and natural language processing circa 2001. In Committee on the Fundamentals of Computer Science: Challenges, Computer Science Opportunities, and National Research Council Telecommunications Board, editors, *Computer Science: Reflections on the Field, Reflections from the Field*, pages 111–118. The National Academies Press, 2004.

R. McDonald, K. Crammer, and F. Pereira. Online large-margin training of dependency parsers. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL' 05)*, pages 91–98, 2005.

Ivan Meza-Ruiz, Sebastian Riedel, and Oliver Lemon. Spoken language understanding in dialogue systems, using a 2-layer markov logic network: improving semantic accuracy. Late Breaking Abstracts of Londial '08: Workshop on the semantics and pragmatics of dialogue, 2008a.

Ivan Vladimir Meza-Ruiz, Sebastian Riedel, and Oliver Lemon. Accurate statistical spoken language understanding from limited development resources. In *Proceedings of 2008 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '08)*, pages 5021–5024, 2008b.

Lilyana Mihalkova and Raymond J. Mooney. Bottom-up learning of markov logic network structure. In *Proceedings of the 24nd International Conference on Machine Learning (ICML' 07)*, pages 625–632. ACM, 2007. ISBN 978-1-59593-793-3.

Brian Milch and Stuart Russell. First-order probabilistic languages: Into the unknown. In *Proceedings of the 17th International Conference on Inductive Logic Programming (ILP '07)*, pages 10–24, 2007.

John Mitchell, S. Joy, and B. Borchers. A branch-and-cut algorithm for max-sat and weighted max-sat. In D. Du, J. Gu, and P. Pardalos, editors, *Satisfiability Problem: Theory and Applications*, volume 35 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 519–536. American Mathematical Society, 1997.

Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *Proceedings of the 15th Annual Conference on Uncertainty in AI (UAI '99)*, pages 467–475. Morgan Kaufmann, 1999.

Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Advances in Neural Information Processing Systems (NIPS '01)*, pages 841–848, 2001.

L. Ngo and P. Haddawy. Answering queries from context-sensitive probabilistic knowledge bases. *Theoretical Computer Science*, (171):147—177, 1997.

Nils J. Nilsson. Probabilistic logic. *Artificial Intelligence*, 28(1):71–88, 1986. ISSN 0004-3702.

J. Park. Map complexity results and approximation methods. In *Proceedings of the 18th Annual Conference on Uncertainty in AI (UAI '02)*, pages 388–396, 2002.

Judea Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1988. ISBN 0-934613-73-7.

David Poole. First-order probabilistic inference. In *Proceedings of the 18th International Joint Conference on Artificial Intelligence (IJCAI '03)*, pages 985–991, 2003.

H. Poon and P. Domingos. Sound and efficient inference with probabilistic and deterministic dependencies. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence (AAAI '06)*, pages 458–463. AAAI Press, 2006.

Hoifung Poon and Pedro Domingos. Joint inference in information extraction. In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI '07)*, pages 913–918, 2007.

Hoifung Poon and Pedro Domingos. Joint unsupervised coreference resolution with markov logic. In *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '08)*, pages 650–659, 2008. URL `http://www.cs.washington.edu/homes/hoifung/papers/poon08b.pdf`.

Hoifung Poon, Pedro Domingos, and Marc Summer. A general method for reducing the complexity of relational inference and its application to mcmc. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI '08)*, 2008.

V. Punyakanok, D. Roth, and W. Yih. Generalized inference with multiple semantic role labeling systems. In Ido Dagan and Dan Gildea, editors, *Proceedings of the 9th Conference on Computational Natural Language Learning (CoNLL' 05)*, pages 181–184, 2005.

Vasin Punyakanok, Dan Roth, Wen tau Yih, and Dav Zimak. Semantic role labeling via integer linear programming inference. In *Proceedings of the 20th international conference on Computational Linguistics (COLING '04)*, pages 1346–1352, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

Matt Richardson and Pedro Domingos. Markov logic networks. *Machine Learning*, 62:107–136, 2006.

Matthew Richardson and Pedro Domingos. Markov logic networks. Technical report, University of Washington, 2005.

Sebastian Riedel. Improving the accuracy and efficiency of MAP inference for markov logic. In *Proceedings of the 24th Annual Conference on Uncertainty in AI (UAI '08)*, 2008.

Sebastian Riedel and James Clarke. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '06)*, pages 129–137, 2006.

Sebastian Riedel and Ewan Klein. Genic interaction extraction with semantic and syntactic chains. In *Learning Language in Logic Workshop, ICML 2005*, pages 69–74, 2005.

Sebastian Riedel and Ivan Meza-Ruiz. Collective semantic role labelling with markov logic. In *Proceedings of the 12th Conference on Computational Natural Language Learning (CoNLL' 08)*, pages 193–197, 2008.

D. Roth and W. Yih. A linear programming formulation for global inference in natural language tasks. In *Proceedings of the 8th Conference on Computational Natural Language Learning (CoNLL' 04)*, pages 1–8, 2004.

D. Roth and W. Yih. Integer linear programming inference for conditional random fields. In *Proceedings of the 22nd International Conference on Machine Learning (ICML' 05)*, pages 737–744, 2005.

Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27:379–423, 1948.

Parag Singla and Pedro Domingos. Discriminative training of markov logic networks. In *Proceedings of the 20th AAAI Conference on Artificial Intelligence (AAAI '05)*, pages 868—873, 2005.

Parag Singla and Pedro Domingos. Entity resolution with markov logic. In *Proceedings of the International Conference on Data Mining (ICDM '06)*, pages 572–582, 2006a.

Parag Singla and Pedro Domingos. Memory-efficient inference in relational domains. In *Proceedings of the 21st AAAI Conference on Artificial Intelligence (AAAI '06)*, 2006b.

Parag Singla and Pedro Domingos. Markov logic in infinite domains. In *Proceedings of the 23rd Annual Conference on Uncertainty in AI (UAI '07)*, pages 368—37, 2007.

Parag Singla and Pedro Domingos. Lifted first-order belief propagation. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI '08)*, pages 1094–1099, 2008.

Jonathan Slocum. A practical comparison of parsing strategies. In *Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics (ACL' 81)*, pages 1–6, Morristown, NJ, USA, 1981. Association for Computational Linguistics.

D. Sontag and T. Jaakkola. New outer bounds on the marginal polytope. In *Advances in Neural Information Processing Systems (NIPS '07)*, pages 1393–1400, 2007.

David Sontag, T. Meltzer, A. Globerson, T. Jaakkola, and Y. Weiss. Tightening LP relaxations for MAP using message passing. In *Proceedings of the 24th Annual Conference on Uncertainty in AI (UAI '08)*, 2008.

Charles Sutton and Andrew McCallum. Piecewise pseudolikelihood for efficient training of conditional random fields. In *Proceedings of the 24nd International Conference on Machine Learning (ICML' 07)*, pages 863–870, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-793-3.

Charles Sutton, Khashayar Rohanimanesh, and Andrew McCallum. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. In *Proceedings of the 21st International Conference on Machine Learning (ICML' 04)*, pages 99–107, 2004.

B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proceedings of the 18th Annual Conference on Uncertainty in AI (UAI '02)*, pages 485–492, 2002.

Ben Taskar. *Learning Structured Prediction Models: a Large-Margin approach*. PhD thesis, Stanford University, 2004.

Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin markov networks. In Sebastian Thrun, Lawrence Saul, and Bernhard Schölkopf, editors, *Advances in Neural Information Processing Systems (NIPS '04)*, pages 25—32, Cambridge, MA, 2004. MIT Press.

Kristina Toutanova, Aria Haghighi, and Christopher D. Manning. Joint learning improves semantic role labeling. In *Proceedings of the 43rd Annual Meeting of the Association for Computational Linguistics (ACL' 05)*, pages 589–596, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

Roy W. Tromble and Jason Eisner. A fast finite-state relaxation method for enforcing global constraints on sequence decoding. In *Joint Human Language Technology Conference/Annual Meeting of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL '06)*, pages 423–430, 2006.

Vladimir N. Vapnik. *The Nature of Statistical Learning Theory*. Springer, 1995.

Martin J. Wainwright, Tommi Jaakkola, and Alan S. Willsky. Tree-based reparameterization framework for analysis of sum-product and related algorithms. *IEEE Transactions on Information Theory*, 49:1120–1146, 2003.

Jue Wang and Pedro Domingos. Hybrid markov logic networks. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence (AAAI '08)*, 2008.

M. Wellman, J. S. Breese, and R. P. Goldman. From knowledge bases to decision models. *Knowledge Engineering Review*, (7), 1992.

H. Paul Williams. *Model Building in Mathematical Programming*. Wiley, 4th edition, 1999.

Terry Winograd. Procedures as a representation for data in a computer program for understanding natural language. Technical report, MIT, 1971.

Wayne L. Winston and Munirpallam Venkataramanan. *Introduction to Mathematical Programming*. Brooks/Cole, 2003.

Nianwen Xue and Martha Palmer. Calibrating features for semantic role labeling. In *Proceedings of the Conference on Empirical methods in natural language processing (EMNLP '04)*, pages 88–94, 2004.