EVALUATING EXACT AND APPROXIMATE ALGORITHMS FOR INTEGER LINEAR PROGRAMMING FORMULATIONS OF MAP INFERENCE

BY

BHARGAV MANGIPUDI

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2017

Urbana, Illinois

Advisor:

Professor Dan Roth

Abstract

Structured prediction tasks involve an inference step which allows for producing coherent label assignments to the output structure. This can be achieved by constraining the output using prior knowledge about the domain. This paradigm is called Constrained Conditional Models; and it involves augmenting the learning of conditional models with declarative constraints. The MAP inference problem in CCM framework can be solved by formulating an Integer Linear Programming problem. This ILP formulation is generally relaxed to an Linear Programming problem by dropping the integrality constraints and making it tractable. In this work, we evaluate other approximate inference algorithms for the MAP estimate for structured prediction task in the CCM framework. We model the constrained structured prediction problem as a factor graph and use different graphical models based algorithms. We evaluate these methods for the quality of their solution and the computation time over some NLP tasks with varying complexity. For large-scale problems, the tradeoff between inference time and the approximateness of the solution is a crucial aspect. Furthermore, these inference solvers are provided as black-box implementations in Saul, which is a declarative programming language for structured prediction tasks.

To my parents and sister who have always supported me

Acknowledgments

This project would not have been possible without the support of many people. Many thanks to my adviser, Prof. Dan Roth, for guiding me through my work and providing invaluable feedback. I would also like to thank Parisa Kordjamshidi and Daniel Khashabi for introducing me to the Saul framework and for all the help during the process. I really enjoyed working with all the members of the Cognitive Computation Group. Mark Sammons has been very helpful with the software issues and administrative problems. I would also like to thank Ben Zhou for work on the ACE Entity-Relation classifiers.

Finally, I would like to express my gratitude to my family and friends for helping and supporting me.

Table of Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

AD3	Alternating Directions Dual Decomposition
BP	Belief Propagation
CCM	Constrained Conditional Model
ILP	Integer Linear Programming
JNI	Java Native Interface
MAP	Maximum a posteriori
SRMP	Sequential Reweighted Message Passing

Chapter 1

Introduction

Structured prediction is a machine learning framework that deals with structured output spaces. Structures in the output space capture dependencies between objects that would otherwise be ignored by local discriminative models. Most of the discriminative tasks in Natural Language Processing and linguistics have some *output structure* that needs to be reasoned about. These could be linear chains, trees, or other forms. *Saul* [Kordjamshidi et al., 2016, Kordjamshidi et al., 2015] is a declarative programming language that was designed with an aim to make it easier to represent structured prediction problem using a domain-specific language build over Scala. A major aspect of Saul is to let the user reason about the *structured output spaces* easily.

Constrained Conditional Model (CCM) [Chang et al., 2008] is a paradigm that discusses the formalism to add global constraints (from domain knowledge) to ensure a *coherent* prediction for the output-structure. Saul makes it easier for the user to represent constraints on the output space using the expressive power of first order logic. Typically, constraints are realized using an ILP formulation, which is solved by most solvers using an LP-relaxation and a branch-and-bound strategy. Since, these constraints are essentially restrictions on discrete output space configurations, we can model them using graphical models. Graphical models let you decompose the optimization objective over smaller structures and patterns in the output. If you consider the joint distribution of the structured output spaces, constraints can be seen as a way to restrict some configurations (or assignments) in the output space. Following this analogy, we model the problem as a graphical model with factors to model the output-space configurations. Thus, we can achieve the same effect as using a ILP by restricting local assignments at the factors in the factor graph.

In this work, we experiment with approximate inference algorithms based on graphical models as alternatives to the ILP approach. We evaluate the performance and quality of results on a few NLP tasks of different complexity. Additionally, these inference algorithms are provided in the Saul toolkit as *black-box* algorithms. In Saul, the users can write constraints in a first order logic syntax and get the flexibility of choosing between different solvers.

Chapter 2

MAP Inference in Structured Prediction

2.1 Structured Prediction

Structured prediction is a class of classification problems where the output labels are a very large set with some sort of *structure*. The size of the output classes is large enough for enumeration-based methods to be intractable or inefficient. Thus, incorporating the *structure* in the solution process is paramount. The output is generally not a single discrete value but a set of values, $y = (y_1, y_2, \dots y_L)$ for a sequence of L variables. Thus, the entire output space can be represented as $\mathcal{Y} \subset \mathcal{Y}_1 \times \mathcal{Y}_2 \times \ldots \times \mathcal{Y}_L$. In structured prediction, all the output labels $\mathcal{Y}$ are predicted *jointly*, capturing any correlations between them.

Consider the natural language processing task of inducing a syntactic parse tree from an input sentence. The input to the problem is a list of words in a sentence, while the output is a tree representing the constituent parsing information. In this problem, the output is a structured space of all possible parse trees that can be derived from that sentence. An example parse tree for the sentence: '`John hit the ball.`' is shown in Figure 2.1.

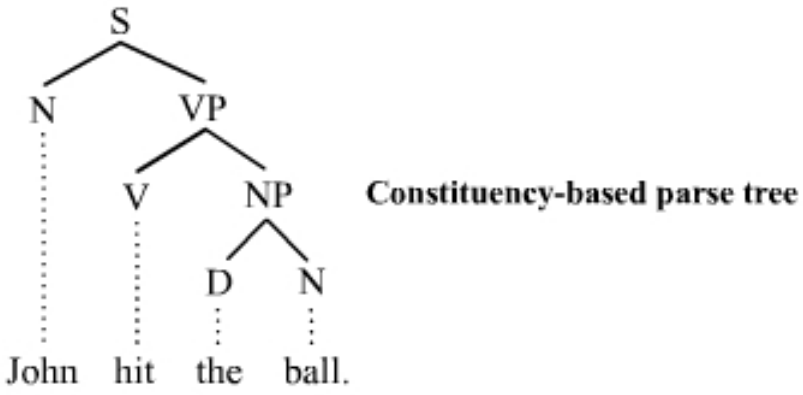

Figure 2.1: Parse tree example

2.2 MAP Inference

MAP inference in structured prediction tasks aims to find the most probable assignments to the structure according to a probabilistic model. Consider the scenario where the input x has output $y \in Y$, where Y represents the set of all possible outputs. The MAP assignment for the problem would be the assignment that maximizes the model's score. If we use $\phi(x)$ to represent the feature vector for the input and w to represent the weights of the classifier, then we can represent the MAP assignment as:

$$\hat{y} = \arg\max_{y \in Y} \; F(x, y) \;=\; \arg\max_{y \in Y} \; w \cdot \phi(x) \tag{2.1}$$

In the case of structured prediction, the score $F(x, y)$ can be decomposed over the structure in the output. Suppose that the output y can be decomposed into smaller sub-structures $y_\delta \in \Phi(y)$, then we can write the objective as:

$$\hat{y} = \arg\max_{y \in Y} \sum_{\delta \in \Phi(y)} F(x, y_\delta) \;=\; \arg\max_{y \in Y} \sum_{\delta \in \Phi(y)} w_\delta \cdot \phi(x, y_\delta) \tag{2.2}$$

For structured prediction problems, where the output space is exponential in the size of the input, MAP inference is NP-hard [Shimony, 1994]. This makes it attractive to use approximate inference models which have a faster runtime at the expense of getting a sub-optimal solution.

2.3 Constrained Conditional Model

Constrained Conditional Model [Chang et al., 2008] is a training paradigm that augments the learning objective to also include global constraints that are induced from prior knowledge or domain expertise. The objective function that is optimized in CCM is shown below:

$$\arg\max_{y} \; \lambda \cdot F(x, y) - \sum_{i=1}^{K} \rho_i d(y, 1_{C_i(x)}) \tag{2.3}$$

$F(x, y)$ denotes the score of the assignment y for the input instance x. The term $C_i(x)$ denotes a single global constraint in the problem (out of the K constraints). For each constraint, $d(y, 1_{C_i(x)})$ captures the measure of constraint violation. The overall goal is to maximize the weighted sum of assignment score which is penalized by constraint violation. λ and ρ are hyper-parameters that need to be adjusted according to the training scenario.

This objective function can be expressed using an ILP objective by introducing binary indicator variables for each (classifier, label) pair [Roth and Yih, 2007]. The ILP objective for the *assignment cost* when each output variable has $|Y|$ labels and the problem has L output variables, can be written as:

$$\begin{aligned} \text{maximize} \quad & \sum_{i=1}^{L} \sum_{l \in |Y|} y_{i,l} \cdot F(x, y_{i,l}) \\ \text{subject to} \quad & y_{i,l} \in \{0,1\} \qquad \forall i \in \{1,2,\ldots,L\} \\ \text{and} \quad & \sum_{l \in |Y|} y_{i,l} = 1 \qquad \forall i \in \{1,2,\ldots,L\} \end{aligned}$$

Constraints can be incorporated into this objective by introducing additional variables that represent the penalty for not satisfying constraints. This is represented by the $d(y, 1_{C_i(x)})$ term. For incorporating hard-constraints that should never be violated, ρ can be assigned a very high value (say ∞), so that the penalty of violating the constraint is high and a violating assignment is never preferred. In the chapter 3, we discuss the methods to optimize the same objective using factor graphs.

The ability to incorporate constraints into the MAP objective leads to some interesting training paradigms. In this work, we look at one particular paradigm called `L+I`, which stands for *Learning + Inference*. The `L+I` paradigm involves first training local (independent) classifiers for each part of the structure. These local classifiers can be training using any discriminative classification approach. These local assignments are then augmented with constraints in the *Inference* step so that the final MAP assignments are more coherent for the problem.

2.4 Algorithms Studied

In Saul, the inference algorithms are abstracted away from the end-user and can be used as pluggable *black-box* components of the system. This gives the user flexibility to choose the appropriate algorithm for their inference task. In this section we discuss the some exact and approximate inference algorithms briefly.

2.4.1 Integer Linear Programming (ILP)

Formulating the constrained inference problem as an ILP over local classifier scores is well studied in [Roth and Yih, 2004], [Chang et al., 2008] and [Rizzolo, 2011]. ILP-based inference has always been available in Saul and the results of ILP on a few NLP tasks was presented in [Kordjamshidi et al., 2016]. If the ILP has the form $max\ c^T x$ such that $Ax = b$, where A, b and c have all integer entries and A is totally

unimodular, then every feasible solution is guaranteed to be integral. [Rizzolo, 2011] discusses approaches to ensure that the ILP formulation for constraints has a unimodular matrix A. We follow the ILP formulation from [Rizzolo, 2011] and use Gurobi [Gurobi Optimization, 2016] and ojAlgo[1] software packages to solve the ILP problem formulation.

Exact MAP-inference in an ILP is a NP-hard problem, but most solvers relax the integrality condition and solve the *tractable* LP problem. Approximate integral solutions can be obtained using different strategies like rounding the solutions or using a branch-and-bound routine to search of better solutions. This approach is called the LP-MAP inference. Solving as linear programs makes the problem tractable but the solution may not be integral or meaningful. We do not provide experimental results for LP-MAP in this work.

2.4.2 (Loopy) Belief Propagation

Graphical models are probabilistic models which expresses a complete distribution over a multi-dimensional space using a graph-based representation, which is a compact or factorized representation of a set of independences that hold in the specific distribution.

We use the factor graph representation [Kschischang et al., 2001] for graphical models, which subsumes *undirected* Markov networks and *directed* Bayesian networks. A factor graph is an undirected bipartite graph between nodes representing variables (drawn as a *circle*) and nodes representing functions (or factors) between variables (drawn as a *square*). Example of a factor-graph used for error-correcting codes is shown in figure 2.2.

Belief Propagation is an efficient algorithm to solve inference problems in graphical models based on passing local messages. It was first introduced for "Bayesian Networks" by Judea Pearl in [Pearl, 1986] and in the book [Pearl, 1988]. Belief Propagation has been studied extensively in literature, especially for problems in statistical physics and computer vision.

We will discuss the factor graph representation and the Max-Product algorithm for MAP inference in factor graphs. A factor graph model $G = (V, F, E)$, where V is the set of variables, F is the set of factors and E is the edges in the graph. The graph G represents the *joint probability distribution* of all variables x. The set of factors and edges control how the probability distribution decomposes over the set of variables.

$$\mu(x) = \frac{1}{Z} \prod_{a \in F} \psi_a(x_{\partial a}) \prod_{i \in V} \psi_i(x_i)$$

Consider the factor graph for the parity check code in figure 2.2. This graph represents a real-world scenario where the factor graph can be used for correcting single bit error over an unreliable transmission

[1] `http://www.ojalgo.org/`

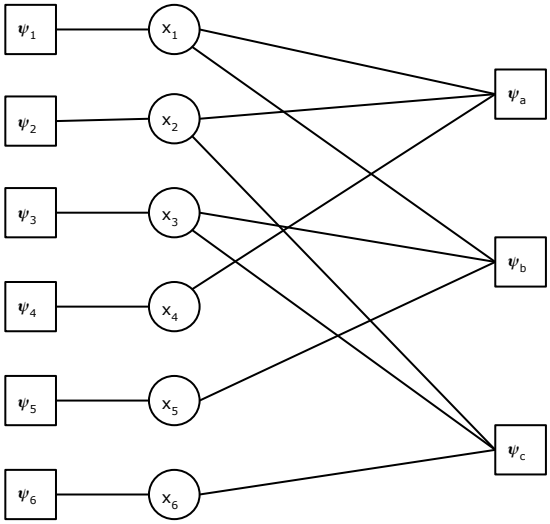

Figure 2.2: Factor graph for $N = 6$, $k = 3$ parity check code

network. The graph contains six binary variable nodes: $(x_1, x_2, \ldots, x_6)$. Each of these nodes are connected to a unary factor: $(\psi_1, \psi_2, \ldots, \psi_6)$. Each of these unary factors represents the probability of each bit flipping over a network. The ternary factors: (ψ_a, ψ_b, ψ_c) are used to enforce the parity over the variables connected to each factor. A valid transmission always satisfies the parity constraint. When we receive a codeword over the unreliable network, we can do a MAP inference on the factor graph with the code word as initialization and get the original message.

Max-BP is an iterative algorithm which involves sending messages from 'variable nodes' to 'factor nodes' and vice-versa. Each message is an approximation of the marginal probability of the variable node. These *belief* messages are exchanges till they converge or keep cycling.

Message from variables to factors

$$\nu_{i\to a}^{(t+1)}(x_i) = \prod_{b\in\partial i\setminus\{a\}} \tilde{\nu}_{b\to i}^{(t)}(x_i)$$

Message from factors to variables

$$\tilde{\nu}_{a\to i}^{(t+1)}(x_i) = \max_{x_{\partial a\setminus\{i\}}} \prod_{j\in\partial a\setminus\{i\}} \nu_{j\to a}^{(t)}(x_j)\psi_a(x_{\partial a})$$

To obtain the global MAP assignment configuration, we just need to store the *argmax* for each message

sent from a factor to variable for back-tracking.

$$\delta_{a\to i}(x_i) = \arg\max_{x_{\partial a\setminus\{i\}}} \prod_{j\in\partial a\setminus\{i\}} \nu_{j\to a}(x_j)\psi_a(x_{\partial a})$$

On a tree-shaped graphical model, BP algorithms is guaranteed to converge to the correct marginal posterior probabilities and MAP assignment. For graphical models with loops, an approximate algorithm called the *loopy* BP runs belief propagation algorithm assuming the graph does not have cycles. This algorithm is not guaranteed to converge to the optimal solution. [Weiss and Freeman, 2001] evaluate the max-product algorithm on some arbitrary factor graphs and have observed good emperical performance on graphs with loops. Their analysis also show some specific graph structures where BP does not converge and keeps cycling between a few states. Thus, the MAP results on loopy graphs are approximate and BP is not guaranteed to converge.

We use the Factorie [McCallum et al., 2009] toolkit[2] for the factor graph representation and the Loopy Belief Propagation algorithm.

2.4.3 Sequential Reweighted Message Passing (SRMP)

Tree Reweighted Message passing (TRW) is an algorithm introduced in [Wainwright et al., 2005a] and [Wainwright et al., 2005b] as an alternative to the *max-product* algorithm tailored to hyper-trees and general graphical models. Since MAP inference on trees using message passing is exact and tractable, TRW proposes a *lower-bound* on the MAP objective for graphs with loops as a convex combination of the problem on the set of spanning trees of the graph. Thus, instead of maximizing the MAP objective for a loopy-graph, the algorithm uses message passing updates to maximize the MAP problem on each of the spanning trees.

[Kolmogorov, 2006] proposes an improvement to the TRW algorithm called the Tree Reweighted Message passing - Sequential (TRW-S). TRW-S imposes a *sequence* on the nodes in the spanning trees and performs sequential message passing updates. [Kolmogorov, 2015] introduced Sequential Reweighted Message Passing (SRMP) as a generalization of TRW-S from pairwise to higher-order graphical models. SRMP is a family of message passing algorithms which includes *Convex Max-Product* and *TRW-S*

In this work we use SRMP as an advanced message-passing belief propagation algorithm for MAP inference. SRMP had C++ implementation[3] provided by the author. We implemented JNI-wrappers over the native implementation and provide an inference solver implementation in Saul.

[2] `http://factorie.cs.umass.edu/`
[3] `https://github.com/opengm/SRMP`

2.4.4 Alternating Directions Dual Decomposition (AD3)

AD3 [Martins et al., 2011] is an algorithm for *approximate* MAP inference on factor graphs which is based on the alternating directions method of multipliers (ADMM). The algorithm divides the maximization problem into local sub-problems which are solved independently. In a factor graph, each factor is treated as a *local* sub-problem. Thus, the MAP problem at each problem is solved independently. The solutions of the local sub-problems are then gathered to compute a global parameter update. The key characteristic of this algorithm is that each local sub-problem has a quadratic regularizer which leads to faster convergence than other dual decomposition algorithms. The algorithm details are explained in detail in [Martins et al., 2015].

AD3 had C++ implementation[4] provided by the author. The implementation also provides closed-form solutions for some *hard-constraint* factors so that the local MAP/QP subproblem can be evaluated efficiently. AD3 has shown good performance on complex tasks like protein design, dependency parsing [Martins and Almeida, 2014] etc.

We implemented a JNI-wrapper over the native implementation and provide an inference solver implementation in Saul.

2.4.5 Probabilistic Soft Logic (PSL)

Probabilistic Soft Logic [Bach et al., 2013] is a framework for developing probabilistic models for reasoning about relational and structural data. PSL provides a Groovy-based syntax for declarative representation of the relational models. Relation models can be defined in terms of predicates, which can be thought of as tuples in a database. PSL lets you define *rules* over predicates using a subset of first order logic expressions. There rules along with some weights facilitate constrained learning and inference. The optimization problem is case as a hinge-loss based markov random field and solved using an appropriate solver like ADMM, BooleanSAT, etc. In the end of chapter 3, we compare PSL with other approaches.

Note: We do not provide experimental results for PSL in this thesis.

[4] `https://www.cs.cmu.edu/~ark/AD3/`

Chapter 3

Constrained Inference Formulation

In Saul [Kordjamshidi et al., 2015], constraints over the structured output-space are represented in First order logic. The following FOL constructs are allowed[1] in Saul for representing constraints: Conjunction (AND), Disjunction (OR), Implication (IMPLY), Double Implication (EQUAL), Negation (NOT). We also have the following quantification constructs: Atleast, Atmost, and Exactly.

Using first order logic constructs in Saul gives the user more flexibility and expressivity in declaring constraints. Variables in first order logic are induced from a local classifiers' scores on an instance. These variables are then combined into rules using the operations mentioned above. Complex constraints can be further constructed by using the operations on the variables recursively.

The code snippet below represents a constraint for the Entity-Relation example. The constraint is defined on three classifiers' application on three instances. The following base variables are created:

1. *LivesInClassifier* applied to instance `x` for the *Lives-In* relation.
2. *PersonClassifier* applied to instance `x.e1`, the first entity of the relation.
3. *LocationClassifier* applied to instance `x.e2`, the second entity of the relation.

Listing 3.1: Example constraint in Saul.

```
// if x is lives-in relation, then its first argument should be person,
// and second argument should be location.
def livesInConstraint(x: ConllRelation) = {
  (LivesInClassifier on x isTrue) ⟹
  ((PersonClassifier on x.e1 isTrue) and (LocationClassifier on x.e2 isTrue))
}
```

Note: Detailed description of the different formulations for the Entity-Relation example is discussed in the appendix A.

[1] Latest Saul documentation is at `https://github.com/CogComp/saul/blob/master/saul-core/doc/SAULLANGUAGE.md`

Transformation to ILP formulation was well studied previously in [Roth and Yih, 2004, Chang et al., 2008, Rizzolo, 2011]. As part of this work, we look at transforming the MAP objective and constraints to a factor graph so that we can use techniques for inference in graphical models. For most structured prediction tasks, the factor graph would contain loops and thus, the inference performed would be approximate. In the following sections, we look at the different implementation and some analysis of the different approaches.

3.1 Transformation to Factor Graph

In this section, we describe the process of transforming the MAP objective (Equation 2.3) with local classifiers and the FOL constraints into a factor-graph representation that can be used for Max-Belief Propagation and SRMP algorithms. We use a method similar to the one used in AD3 [Martins et al., 2015] for their local sub-problems.

3.1.1 Incorporating the MAP objective

From the objective in the equation 2.3, we see that the local classifier scores are considered in the first part of the objective. This is similar to the *assignment cost* in [Roth and Yih, 2007]; though we aim to maximize the negative of the cost (the score) in constrast to minimizing the cost. In our factor graph, we introduce a variable *node* for each binary classifier's application on a each instance participating in the constrained problem. These variable nodes capture all the initial variables in the first order logic representation. To incorporate the *local* classifier scores into the factor graph, we introduce *unary* factors to each of these nodes with their log-potential initialized to the classifier score.

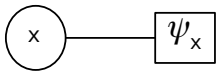

Figure 3.1: Unary factor for a variable node

x	Factor Potentials
true	$\log(P(x = \text{true}))$
false	$\log(1 - P(x = \text{true}))$

Table 3.1: Unary factor potentials

Thus, if we just have binary variable nodes with unary factors, we get the following joint-distribution for the BP factor graph:

$$\begin{aligned}\mu(x_1, x_2, \ldots, x_V) &= \frac{1}{Z}\prod_{i=1}^{V}\psi_i(x_i = v_i) \\ &\propto \log(\frac{1}{Z}\prod_{i=1}^{V}P(x_i = v_i)) \\ &\propto \sum_{i\in V}\log(P(x_i = v_i)) - log(Z) \\ &\propto \sum_{i\in V}\log(P(x_i = v_i))\end{aligned} \tag{3.1}$$

This is similar to the first part of the MAP objective in equation 2.3, hence the assignment in this case is same as the local classifier predictions. In the following sections, we look at incorporating *constraints* into this graphical model joint-distribution factorization.

For multi-class classifiers, we introduce binary variables nodes for each of the individual labels. We then introduce an XOR-constraint on these binary variables for a single consistent assignment.

3.1.2 Factors representation for incorporating constraints

In this section, we will discuss the conversion for different first-order-logic constructs to corresponding factor graph components. We consider the simplest case where there are only two variables involved - x and y. Some of the constraints also deal with an output variable z. Output variables are intermediate variables used for connecting different constraints together.

Exclusive-OR: XOR

Table 3.2 shows the factor potentials that enforces mutual exclusivity for assignment to the variables. We use *negative* infinity log-potential value to indicate a hard-constraint that must not be violated by a MAP assignment. Since an assignment with $-\infty$ potential forces low objective value, such assignments are avoided during message passing updates.

x	y	Factor Potentials
0	0	-∞
0	1	1
1	0	1
1	1	-∞

Table 3.2: Factor potentials for XOR Factor

Factor XOR is mostly used for exclusivity of assignments. For example: making the assignment of the binary nodes of a multi-label classifier to take consistent values. For multi-class classifier, we can extend the XOR factor by representing them as a disjunction of $O(n)$ conjunctive clauses. Each of the base conjunctions only allows one variable to take the label 1.

Conjunction: AND

Table 3.3 shows the factor potentials that enforces the conjunction constraint between two variables. This is a hard-constraint that forces both assignments to 1. Soft version of the AND constraint can be enforced using the AND-OUT factor that follows.

x	y	Factor Potentials
0	0	-∞
0	1	-∞
1	0	-∞
1	1	1

Table 3.3: Factor potentials for AND Factor

Conjunction with Output: AND-OUT

AND-OUT extends the AND factor with an output variable. The output variable z represents the result of the $x \cap y$. Table 3.4 shows the factor-potential for this factor.

z	x	y	Factor Potentials for $z = x \cap y$
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	-∞
1	0	0	-∞
1	0	1	-∞
1	1	0	-∞
1	1	1	1

Table 3.4: Factor potentials for AND-OUT Factor

Further, the variable z can be constrained by adding it to other factors or attaching *unary* factors to it.

Disjunction: OR

OR-factor enforces the disjunction constraint on two variables. Table 3.5 shows the factor potentials for the OR-factor.

x	y	Factor Potentials
0	0	-∞
0	1	1
1	0	1
1	1	1

Table 3.5: Factor potentials for OR Factor

Disjunction with Output: OR-OUT

Similar to AND-OUT, this factor extends OR with an output variable. This leads to a soft-constraint that can be constrained by the rest of the factor graph.

z	x	y	Factor Potentials
0	0	0	1
0	0	1	-∞
0	1	0	-∞
0	1	1	-∞
1	0	0	-∞
1	0	1	1
1	1	0	1
1	1	1	1

Table 3.6: Factor potentials for OR-OUT Factor

AtLeast and AtLeast with Output

Implementing the AtLeast-based quantifiers is not efficient in factor graphs. We can implement these quantifiers in terms of a disjunction of an *exponential* $\binom{n}{k}$ conjunction clauses. This make factor-graph based approaches un-effective.

Note: Empirical observation that most NLP tasks only have `AtLeast(1)` or `AtMost(1)`, which can be cast into simpler and efficient DNF forms.

Generalization

Though we only discuss factors with two input variable, this approach can be extended to arbitrary length constraints by introducing intermediate variables that can be forced to a hard true. Let x, y, z be three binary variable nodes. To represent $x \cap y \cap z$, we can use the following factors:

- AND-OUT($t_1 = x \cap y$)
- AND($t_1 \cap z$)

This composability makes it easy to programmatically transform constraints with arbitrary complexity.

Note: For numerical computation consistency, instead of using negative infinity, we use -100 for the in-compatible assignment potential at each factor. This has a similar effect but leads to well-behaved computations.

3.1.3 Transforming General Constraints

In the previous sections, we look translating single constraints and handling complex single constraints. Now, we will briefly look at handling multiple independent constraints.

If we have multiple constraints that need to hold simultaneously, we can express them as *conjunctions* of first order expressions. We call the expressions in this conjunction a *top-level* conjunction, as each of these expressions must hold independently.

i. Simplify first order constraints and group them as a top-level *conjunction* of expressions ;

ii. Identify all (classifier, instance) pairs in the problem ;

for *each classifier in the problem* **do**

  iii. Create a binary variable node for each (classifier, instance, label) tuple and attach a *unary* factor with log-potential scores from local classifiers ;

  iv. If the classifier is a multi-class classifier, add a XOR factor for variables from step iii.

end

for *each top-level constraint* **do**

  v. Recursively convert lower-level expressions to factor graphs while creating intermediate variables as required. ;

  Lower-level expressions typically use the `-OUT` output factors for creating intermediate variables ;

  vi. Add the top-level constraints using variable nodes from step v. ;

end

Algorithm 3.1: Pseudo-code for transforming inference problem to a factor graph

3.1.4 Model Equivalence

In section 3.1.1, we saw how the factor graph approach and the ILP approach have similar MAP objectives (without constraints); and hence, have the same MAP solution. In this section, we will look at the factor graph objectives when constraints are involved.

Consider the factor graph objective where V represents the set of variable nodes and F represents the factor nodes. We can separate the set of variables into two classes: classifier variables and intermediate variable. Classifier variables are created from a classifier's scores on an instance, whereas intermediate variables are used for chaining constraints. We can separate the two sets of variables in to sets: C and I respectively.

$$
\begin{aligned}
\mu(x) &= \frac{1}{Z}\prod_{i\in V}\psi_i(x_i)\prod_{a\in F}\psi_a(x_{\partial a}) \\
&= \frac{1}{Z}\Big(\prod_{c\in C}\psi_c(x_c)\Big)\Big(\prod_{i\in I}\psi_i(x_i)\Big)\prod_{a\in F}\psi_a(x_{\partial a}) \\
&\propto \log(\frac{1}{Z}\Big(\prod_{c\in C}\psi_c(x_c)\Big)\Big(\prod_{i\in I}\psi_i(x_i)\Big)\prod_{a\in F}\psi_a(x_{\partial a})) \\
&\propto \sum_{c\in C}\log(P(x_c)) + \sum_{i\in I}\log(P(x_i)) + \sum_{a\in F}\log(P(x_{\partial a})) - \log(Z) \\
&\propto \sum_{c\in C}\log(P(x_c)) + \sum_{i\in I}\log(P(x_i)) + \sum_{a\in F}\log(P(x_{\partial a})) \\
&\propto \underbrace{\sum_{c\in C}\log(P(x_c))}_{\textit{assignment score}} + \underbrace{\sum_{i\in I}\log(P(x_i)) + \sum_{a\in F}\log(P(x_{\partial a}))}_{\textit{constraint-satisfaction score}}
\end{aligned}
$$

Thus, when the constrained are satisfied, the *constraint score* should be bounded to a constant and we can find the MAP assignment to variables in C under those conditions.

3.1.5 Handling Soft Constraints

For handling soft top-level constraints, instead of a negative infinity, we can use a higher value so that all assignments are feasible with different penalties for assignments. By tweaking this hyper-parameter for each constraint, the extent of satisfaction for each constraint can be enforced.

3.2 Transformation to AD3

Transforming the objective and constraints to AD3 is similar to the method used for factor graphs. AD3 works with a factor graph representation as its input. Also, in AD3, each factor is solved as a separate *local* sub-problem, whose results are then gathered for global parameter updates and to ensure consistency in overlapping variables. AD3 implementation provides closed-form solutions for the local sub-problems (factor types) that we discussed in section 3.1. Thus, the *ATLEAST* factor is realized efficiently in AD3 because the local sub-problem is solved in the closed-form using a budget assignment approach.

3.3 Analysis

Table 3.7 (below) summarizes the differences between various inference solvers.

Algorithm	AND/OR	XOR	ATLEAST	Exactness	Remarks
ILP	Yes	Yes	Yes	Yes	NP-hard. Solvers use iterative approach like branch-and-bound to improve solutions.
LP-MAP	Yes	No	Yes	Approx.	LP is tractable and efficient. Rounding solutions may cause constraint violation.
Max-BP	Yes	No	No	Approx.	May not converge to the optimal. Objective might lower during iterations (or cycle between some fixed values).
SRMP	Yes	No	No	Approx.	May not converge to the optimal. Bound on objective does not decrease with iterations.
AD3	Yes	Yes	Yes	Approx.	Iterative algorithm, which can converge to the optimal.
PSL	Yes	Yes*	No	Approx.	XOR is available as a rule on predicates. Can only be used for (classifier, instance) predicates.

Table 3.7: Factor potentials for OR-OUT Factor

ILP and AD3 (dual-decomposition) are the most flexible and efficient algorithms.

Note: MaxBP, SRMP and PSL does not support AtLeast/AtMost and Existential quantifier rules in their implementation. AtLeast/AtMost quantifiers can be implemented by adding an exponential $\binom{n}{k}$ number of rules in the DNF form.

Chapter 4

Experiments

To evaluate the different inference techniques, we use the following NLP tasks. For each task, we implement the *L+I* paradigm [Roth and Yih, 2004], where individual (*local*) classifiers are trained using the training data; and then constraints are imposed during the inference process. Thus, constraints helps in making sure that assignments to individual instances are consistent according to the structure of the output space and additional domain-knowledge about the task.

Using three NLP tasks, we study the performance and efficiency of the different inference algorithms. We also evaluate a simple Beam Search based solver that tried to incorporate the constraints as per of the structures kept in the beam. We discuss the amount of constraint violation in the approximate algorithms, and effectively, demonstrate how the choice of the algorithm affect of quality of the results and inference time.

4.1 Entity Relations: UIUC ER Dataset

We look at the task of Entity and Relation identification. The data for our experiments was derived from [Roth and Yih, 2004], which is an annotation of a set of sentences from TREC documents. The dataset is available publicly[1]. In the dataset, there are 1,987 sentences which contain 4,645 entities, and 6,909 intra-sentence pairs of entities. The entity labels include 1,648 person entities, 1,872 location entities, and 858 organization entities. The relation labels include 420 located in, 394 work for, 451 org based in, 529 live in, and 270 kill. We train independent binary classifiers for each entity and relation type independently. The performance of the *local* binary classifiers is shown in the first row in the results. For the $L + I$ approach, we use the following constraints and perform inference as a post-processing step using scores from local classifiers:

1. **Works-For Relation Consistency:** This constraint enforces that a *Works-For* relation can only exists between a *Person* entity and a *Organization* entity.

[1] `https://cogcomp.cs.illinois.edu/page/resource_view/43`

2. **Lives-In Relation Consistency:** This constraint enforces that a *Lives-In* relation can only exists between a *Person* entity and a *Location* entity.

3. **Works-For::Lives-In Relation Exclusivity:** This constraint implies that a relation cannot be both *Works-For* and *Lives-In* simultaneously.

Table 4.1 shows the results of 5-fold cross-validation for the *constrained* entity classifiers; and table 4.2 shows the results of the *constrained* relation classifiers on the same folds.

	Person (PER)			Location (LOC)			Organization (ORG)			
Experiment	P	R	F1	P	R	F1	P	R	F1	Time (secs)
Local Classifiers	92.99	83.91	88.22	90.36	78.56	84.04	91.22	64.43	75.52	0.866
L + Beam (Size 5)	92.67	85.93	89.17	89.55	79.67	84.32	89.95	69.11	78.16	2.619
L + Beam (Size 10)	92.67	85.98	89.20	89.55	79.67	84.32	89.95	69.11	78.16	2.589
L + ILP-oJAlgo	92.67	85.98	89.20	89.55	79.67	84.32	89.95	69.11	78.16	4.851
L + ILP-Gurobi	92.67	85.98	89.20	89.55	79.67	84.32	89.95	69.11	78.16	4.004
L + MaxBP	92.73	85.98	89.23	89.60	79.67	84.35	90.07	69.11	78.21	3.671
L + AD3	92.67	85.98	89.20	88.55	79.67	84.32	89.93	69.00	78.09	2.915
L + SRMP	92.67	85.98	89.20	89.55	79.67	84.32	89.93	69.00	78.09	2.701

Table 4.1: ER-UIUC: 5-fold CV evaluation results: Entity Classifiers

	Lives In Relation			Works For Relation			
Experiment	Precision	Recall	F1 score	Precision	Recall	F1 score	Time (secs)
Local Classifiers	74.43	63.15	68.33	76.15	66.08	70.76	0.290
L + Beam Search (Size 5)	90.49	60.27	72.35	90.69	55.86	69.14	2.142
L + Beam Search (Size 10)	90.46	60.08	72.20	90.36	56.11	69.23	2.269
L + ILP-oJAlgo	90.46	60.08	72.20	90.36	56.11	69.23	4.503
L + ILP-Gurobi	90.46	60.08	72.20	90.36	56.11	69.23	3.978
L + MaxBP	89.97	60.27	72.18	89.33	56.36	69.11	3.541
L + AD3	90.46	60.08	72.20	90.32	55.86	69.03	2.474
L + SRMP	90.46	60.08	72.20	90.32	55.86	69.03	2.437

Table 4.2: ER-UIUC: 5-fold CV evaluation results: Relation Classifiers

4.2 Entity Relations: ACE Dataset

In this section, we look at the task of Entity and Relation type identification for the ACE 2005 dataset [Walker et al., 2006]. ACE dataset contains 7 coarse entities types and 6 coarse relation types. We only consider the *broadcast news* and *newswire* sections of the English corpus. We train independency local classifiers for entities and relations and enforce constraints during inference to refine the label assignments.

For constrained prediction, we enforce constraints on the permissible range of entities that are associated with each relation type. This requires using domain-knowledge about the relations. One example of such constraint would be restricting the `Family` relation to only exist between two `Person` entities. Such a restriction can be expressed using first order logic constructs and enforced during inference time.

Tables 4.1 and 4.2 show the results of adding constraints to joint-inference of entities and relations using different inference solvers. We evaluate the solvers for performance on the task and inference time.

Experiment	Precision	Recall	F1 score	Time (secs)
Local Classifiers	80.94	80.94	80.94	8.33
L + ILP - Beam Search (100)	80.94	80.94	80.94	40.36
L + ILP - Gurobi	81.63	81.63	81.63	103.66
L + AD3	81.42	81.42	81.42	48.97
L + MaxBP*	73.26	73.26	73.26	1320.73
L + SRMP*	77.10	77.10	77.10	1270.20

Table 4.3: ER-ACE - 2005: 5-fold CV evaluation - Entity Classifier

Experiment	Precision	Recall	F1 score	Time (secs)
Local Classifiers	64.21	50.68	56.65	73.50
L + ILP - Beam Search (100)	64.21	50.68	56.65	163.55
L + ILP - Gurobi	65.14	44.99	53.22	490.20
L + AD3	64.79	43.12	51.78	209.87
L + MaxBP*	47.23	40.62	43.68	1265.20
L + SRMP*	54.67	43.22	48.28	1140.42

Table 4.4: ER-ACE - 2005: 5-fold CV evaluation - Relation Classifier

Note: MaxBP and SRMP solvers fall back to the local classifier prediction if they all factors are not satisfied or if they timeout.

Table 4.5 shows the most frequent constraint violations for different algorithms. The number of constraint violation is measure using a similar 5-fold CV approach on the total of about 5000 relations. You can note that ILP and AD3 solutions violate the minumum number of constraints. Note: that even Gold annotation does not satisfy all constraints. There are few failure cases where the constraints are not satisfied.

Constraint	Gold Annotation	Local Classifier	ILP - Gurobi	AD3
'Employment' relation *legal* arguments	4	267	29	29
'Geographical' relation *legal* arguments	0	232	59	59
'Located' relation *legal* arguments	10	193	36	36
Others	12	437	86	86

Table 4.5: ER-ACE: Most frequently violated constraints.

4.3 Semantic Role Labeling

Semantic Role Labeling [Palmer et al., 2010] is the task of assigning semantic roles to constituents or phrases in a sentence. For this experiment, we study the task of constrained *Verb* Semantic Role Labeling. We use the features from [Punyakanok et al., 2008] (semantic parse features); and follow the methodology of generating argument candidates using heuristics from [Xue and Palmer, 2004] and filter candidates using a binary candidate identification classifier. Then, we train argument type classifiers that assign scores to all filtered arguments. The evaluation is performed on the section 23 of the WSJ treebank dataset.

We use the following constraints during the inference step for consistency of the semantic role assignment to each argument:

1. **No-Overlap Constraint:** Semantic arguments are labeled on non-embedding constituents in the syntactic parse tree; and hence, should be non-overlapping.

2. **Legal Arguments Constraint:** For each *known* predicate, we constraint the argument of that predicate to take values from the *legal* argument classes from PropBank Frames.

3. **No-Duplicate Core Arguments Constraint:** No duplicate argument classes for core semantic roles: A0-A5, AA.

4. **Referential Arguments Constraint:** If there is an *R-arg* labeled argument, there should also some other argument which is labeled *arg* in the same sentence.

5. **Continuation Arguments Constraint:** If there is an *C-arg* labeled argument, there should also some other argument which is labeled *arg* in the same sentence.

While evaluating the Max-BP and SRMP algorithms on the SRL problem, we found that a lot of the factors were not satisfied. This is because the factor graph for SRL problems are large (partially due to large label set and complex constraints), which leads to poor convergence for the message passing algorithms. In such scenarios, when we do not get a *valid* assignment, we use the local scores of the classifiers directly for MaxBP and SRMP.

Tables 4.6 and 4.7 show the results of evaluation of the different inference solvers on the test section of the WSJ PennTreebank dataset. Results presented in table 4.6 use the Charniak syntactic parse for extracting features while table 4.7 use the Stanford parse for extracting features.

Experiment	Precision	Recall	F1 score	Time (in seconds)
Local Classifiers (Greedy No-Overlap)	74.04	68.71	71.27	111
L + ILP - Beam Search (size 10)	73.40	66.90	70.00	302
L + ILP - Beam Search (size 100)	73.98	67.37	70.52	316
L + ILP - ojAlgo	76.48	69.53	72.84	514
L + ILP - Gurobi	76.48	69.53	72.84	406
L + AD3	75.88	69.19	72.38	300
L + MaxBP*	56.26	43.71	49.20	4700
L + SRMP*	57.90	48.20	52.61	7200

Table 4.6: SRL: PennTreebank/WSJ test set (23) - Argument Type - PARSE_CHARNIAK

Experiment	Precision	Recall	F1 score	Time (in seconds)
Local Classifiers (Greedy No-Overlap)	73.61	58.85	65.41	76
L + ILP - Beam Search (size 10)	71.93	56.60	63.35	285
L + ILP - Beam Search (size 100)	72.01	56.68	63.43	279
L + ILP - ojAlgo	75.88	59.35	66.60	473
L + ILP - Gurobi	75.88	59.35	66.60	370
L + AD3	75.86	59.35	66.60	294
L + MaxBP*	52.10	41.37	46.12	4700
L + SRMP*	54.04	45.29	49.23	7200

Table 4.7: SRL: PennTreebank/WSJ test set (23) - Argument Type - PARSE_STANFORD

To further evaluate the quality of results and the amount of approximation in constraint satisfaction, we look at the number of constraints violated by each algorithm. Results are presented in tables 4.8 and 4.9 for the models trained on Charniak parse and Stanford parse respectively.

Constraint	Greedy	ILP - Gurobi	AD3	Beam(5)	Beam(100)	MaxBP*	SRMP*
No-Overlap	0	0	0	2	0	0	0
Legal Arguments	193	0	0	2	0	340	253
No-Duplicate Core Args	308	0	0	2	1	523	638
Referential Args	546	0	0	11	0	1046	840
Continuation Args	94	0	0	2	0	227	342

Table 4.8: SRL: Number of constraint violated by algorithm - PARSE_CHARNIAK

Constraint	Greedy	ILP - Gurobi	AD3	Beam(5)	Beam(100)	MaxBP*	SRMP*
No-Overlap	0	0	0	0	0	0	0
Legal Arguments	820	0	0	6	2	843	750
No-Duplicate Core Args	170	0	0	11	2	240	214
Referential Args	416	0	0	5	1	572	471
Continuation Args	33	0	0	0	0	121	97

Table 4.9: SRL: Number of constraint violated by algorithm - PARSE_STANFORD

From evaluating the set of constraints violated, we see that the solutions from Gurobi and AD3 do not violate any of the constraints. Whereas the other algorithms satisfy constraints with varying amounts of approximation. Modeling the constraints into the Beam Search leads to better constraint-satisfaction than the *greedy* no-overlap approach but the overall performance is worse than the greedy baseline. Belief Propagation based solvers perform poorly as they do not scale with the size of the factor graph. We use a 10-second timeout for each problem and then assign the base classifier's prediction for variables which are not properly satisfied by the BP solution.

ILP approach and AD3 both have about $1 - 1.5$ percentage point improvement in F-1 score from the baseline after adding constraints. This shows the general efficacy of the CCM approach. AD3 approach is consistently faster than the industrial ILP solver, proving to be a viable alternative for large scale structured problems.

Chapter 5

Discussion and Future Work

After evaluating the different approaches to Structured Inference with constraints, we see that using ILP and dual-decomposition (AD3) are the most versatile and have good performance across the NLP tasks. Belief Propagation and SRMP suffer with the problem that they do not scale to large factor graphs with cycles.

AD3 has *comparable* performance to ILP in most cases and handles constraints correctly. But the results of ILP and AD3 differ on some instances. This happens because the AD3 algorithms is restricted to terminate in 10 seconds per problem. For some cases where the number of SRL argument candidates are higher, the algorithm returns a sub-optimal solution which satisfies all constraints. As part of future work, we should evaluate the difference between the algorithms against the size of the problem carefully.

Max-BP and SRMP perform poorly when the constraint contains the AtLeast/Atmost/Exactly quantifiers. In such scenarios, the factor graphs can grow exponential in size and hence, inference becomes inefficient. Especially, they performed much worse on the SRL task where the number of variables in the factor graph consisted of about $5,000 - 10,000$ nodes and factors.

These algorithms need further analysis on more complex tasks. Tasks like Dependency Parsing can be a suitable candidate to evaluate the efficacy of the methods.

Further, the factor graph algorithms can be improved using specialized message updates for some factors. For example: [Smith and Eisner, 2008] discusses an approach to perform efficient message passing updates for XOR and AtMost One factors.

All the experiments in this work perform inference as a post-processing approach after training independent classifiers. We can use the inference techniques during training in a structured SVM training approach [Chang et al., 2015]. Since structured prediction and constrained inference are powerful frameworks to capture global dependencies on structured output spaces, this area is important and requires more rigorous experimentation.

References

[Bach et al., 2013] Bach, S., Huang, B., London, B., and Getoor, L. (2013). Hinge-loss markov random fields: Convex inference for structured prediction. *arXiv preprint arXiv:1309.6813*.

[Chang et al., 2015] Chang, K.-W., Upadhyay, S., Chang, M.-W., Srikumar, V., and Roth, D. (2015). Illinoissl: A java library for structured prediction. *arXiv preprint arXiv:1509.07179*.

[Chang et al., 2008] Chang, M., Ratinov, L., Rizzolo, N., and Roth, D. (2008). Learning and inference with constraints. In *Proc. of the Conference on Artificial Intelligence (AAAI)*.

[Gurobi Optimization, 2016] Gurobi Optimization, I. (2016). Gurobi optimizer reference manual.

[Kolmogorov, 2006] Kolmogorov, V. (2006). Convergent tree-reweighted message passing for energy minimization. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1568–1583.

[Kolmogorov, 2015] Kolmogorov, V. (2015). A new look at reweighted message passing. *IEEE transactions on pattern analysis and machine intelligence*, 37(5):919–930.

[Kordjamshidi et al., 2016] Kordjamshidi, P., Khashabi, D., Christodoulopoulos, C., Mangipudi, B., Singh, S., and Roth, D. (2016). Better call saul: Flexible programming for learning and inference in nlp. In *Proc. of the International Conference on Computational Linguistics (COLING)*.

[Kordjamshidi et al., 2015] Kordjamshidi, P., Wu, H., and Roth, D. (2015). Saul: Towards declarative learning based programming. In *Proc. of the International Joint Conference on Artificial Intelligence (IJCAI)*.

[Kschischang et al., 2001] Kschischang, F. R., Frey, B. J., and Loeliger, H.-A. (2001). Factor graphs and the sum-product algorithm. *IEEE Transactions on information theory*, 47(2):498–519.

[Martins and Almeida, 2014] Martins, A. F. and Almeida, M. S. (2014). Priberam: A turbo semantic parser with second order features. In *Proceedings of the 8th International Workshop on Semantic Evaluation (SemEval 2014)*, pages 471–476.

[Martins et al., 2011] Martins, A. F., Figeuiredo, M. A., Aguiar, P. M., Smith, N. A., and Xing, E. P. (2011). An augmented lagrangian approach to constrained map inference.

[Martins et al., 2015] Martins, A. F., Figueiredo, M. A., Aguiar, P. M., Smith, N. A., and Xing, E. P. (2015). Ad3: alternating directions dual decomposition for map inference in graphical models. *Journal of Machine Learning Research*, 16:495–545.

[McCallum et al., 2009] McCallum, A., Schultz, K., and Singh, S. (2009). FACTORIE: Probabilistic Programming via Imperatively Defined Factor Graphs.

[Palmer et al., 2010] Palmer, M., Gildea, D., and Xue, N. (2010). Semantic role labeling. *Synthesis Lectures on Human Language Technologies*, 3(1):1–103.

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

[Pearl, 1988] Pearl, J. (1988). *Probabilistic reasoning in intelligent systems: networks of plausible inference*. Morgan Kaufmann, San Mateo (Calif.).

[Punyakanok et al., 2008] Punyakanok, V., Roth, D., and Yih, W. (2008). The importance of syntactic parsing and inference in semantic role labeling. *Computational Linguistics*, 34(2).

[Rizzolo, 2011] Rizzolo, N. (2011). Learning based programming.

[Roth and Yih, 2004] Roth, D. and Yih, W. (2004). A linear programming formulation for global inference in natural language tasks. In Ng, H. T. and Riloff, E., editors, *Proc. of the Conference on Computational Natural Language Learning (CoNLL)*, pages 1–8. Association for Computational Linguistics.

[Roth and Yih, 2007] Roth, D. and Yih, W. (2007). Global inference for entity and relation identification via a linear programming formulation.

[Shimony, 1994] Shimony, S. E. (1994). Finding maps for belief networks is np-hard. *Artificial Intelligence*, 68(2):399–410.

[Smith and Eisner, 2008] Smith, D. A. and Eisner, J. (2008). Dependency parsing by belief propagation. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 145–156. Association for Computational Linguistics.

[Wainwright et al., 2005a] Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. (2005a). Map estimation via agreement on trees: message-passing and linear programming. *IEEE transactions on information theory*, 51(11):3697–3717.

[Wainwright et al., 2005b] Wainwright, M. J., Jaakkola, T. S., and Willsky, A. S. (2005b). A new class of upper bounds on the log partition function. *IEEE Transactions on Information Theory*, 51(7):2313–2335.

[Walker et al., 2006] Walker, C., Strassel, S., Medero, J., and Maeda, K. (2006). Ace 2005 multilingual training corpus. *Linguistic Data Consortium, Philadelphia*, 57.

[Weiss and Freeman, 2001] Weiss, Y. and Freeman, W. T. (2001). On the optimality of solutions of the max-product belief-propagation algorithm in arbitrary graphs. *IEEE Transactions on Information Theory*, 47(2):736–744.

[Xue and Palmer, 2004] Xue, N. and Palmer, M. (2004). Calibrating features for semantic role labeling. In *EMNLP*, pages 88–94.

Appendix A: In-depth UIUC ER Example

In this section, we look at the ILP and Factor Graph realizations of the UIUC-ER example. For the constraints mentioned in Section 4.1, we consider the following example:

`Dole is a resident of N.C.`

where `Dole` and `N.C.` are two entities represented by E_1 and E_2 respectively. Let R_{12} represent the relation of E_1 with E_2.

variable	PER	LOC	ORG
E_1	0.45	0.5	0.2
E_2	0.3	0.7	0.6

variable	Lives-In	Works-For
R_{12}	0.7	0.4

Table A.1: Binary Classifier Scores

In the following sections, we look at the ILP formulation and the factor graph approach to solve the *structured MAP* inference for this Entity-Relation example. We consider the simpler case where the constraints are interpreted as hard-restrictions on possible assignments.

Local classifiers

For this sample sentence, we show the scores provided by local classifiers in table A.1. If you only consider the local classifiers, the MAP assignment is:

- $E_1 = Location$
- $E_2 = Location$
- $R_{12} = Lives - In$

ILP formulation

- **MAP objective:** We have two entities, E_1 and E_2 and three binary entity classifiers. Hence, we introduce 6 binary variables for each entity. Also for the relation R_{12}, we will have 4 binary variables. Each of these variables take values $\in \{0, 1\}$.

$$e_{1,PER=true}\ ,\ e_{1,PER=false}\ ,\ e_{1,LOC=true}\ ,\ e_{1,LOC=false}\ ,\ e_{1,ORG=true}\ ,\ e_{1,ORG=false},$$

$$e_{2,PER=true}\ ,\ e_{2,PER=false}\ ,\ e_{2,LOC=true}\ ,\ e_{2,LOC=false}\ ,\ e_{2,ORG=true}\ ,\ e_{2,ORG=false}$$

$$r_{12,LivesIn=true}\ ,\ r_{12,LivesIn=false}\ ,\ r_{12,WorksFor=true}\ ,\ r_{12,WorksFor=false}$$

So, the ILP Objective is:

$$\begin{aligned}
&0.45 \cdot e_{1,PER=true} + 0.55 \cdot e_{1,PER=false} + \\
&0.5 \cdot e_{1,LOC=true} + 0.5 \cdot e_{1,LOC=false} + \\
&0.2 \cdot e_{1,ORG=true} + 0.8 \cdot e_{1,ORG=false} + \\
&0.3 \cdot e_{2,PER=true} + 0.3 \cdot e_{2,PER=false} + \\
&0.7 \cdot e_{2,LOC=true} + 0.3 \cdot e_{2,LOC=false} + \\
&0.6 \cdot e_{2,ORG=true} + 0.4 \cdot e_{2,ORG=false} + \\
&0.7 \cdot r_{12,LivesIn=true} + 0.3 \cdot r_{12,LivesIn=false} + \\
&0.4 \cdot r_{12,WorksFor=true} + 0.6 \cdot r_{12,WorksFor=false}
\end{aligned}$$

subject to

$$\begin{aligned}
&e_{1,PER=true} + e_{1,PER=false} = 1 \\
&e_{1,LOC=true} + e_{1,LOC=false} = 1 \\
&e_{1,ORG=true} + e_{1,ORG=false} = 1 \\
&e_{2,PER=true} + e_{2,PER=false} = 1 \\
&e_{2,LOC=true} + e_{2,LOC=false} = 1 \\
&e_{2,ORG=true} + e_{2,ORG=false} = 1 \\
&r_{12,LivesIn=true} + r_{12,LivesIn=false} = 1 \\
&r_{12,WorksFor=true} + r_{12,WorksFor=false} = 1
\end{aligned}$$

Next, we add the constraints to the ILP.

- **Works-For relation consistency constraint:** This constraint enforces that a *Works-For* relation can only exists between a *Person* entity and a *Organization* entity. This can be represented by the following inequality.

$$e_{1,PER=true} + e_{2,ORG=true} > 2 \cdot r_{12,WorksFor=true}$$

If the assignment on the right evaluate to 1, then it forces both the variables on the left to be 1. Note that the other direction is not imposed by this inequality.

- **Lives-In relation consistency constraint:** This constraint enforces that a *Lives-In* relation can only exists between a *Person* entity and a *Location* entity. Similar to the above constraint, this can be represented by:

$$e_{1,PER=true} + e_{2,LOC=true} > 2 \cdot r_{12,LivesIn=true}$$

- **Works-For::Lives-In relation exclusivity:** This constraint implies that a relation cannot be both *Works-For* and *Lives-In* simultaneously.

$$r_{12,LivesIn=true} + r_{12,WorksFor=true} \leq 1$$

This equation only allows atmost one of these two variable to take the `true` label.

Solving the ILP problem with these inequalities leads to a solution that is more coherent than the prediction made by the local classifiers. The solution leads to the following assignment:

  - $E_1 = Person$
  - $E_2 = Location$
  - $R_{12} = Lives - In$

Factor Graph formulation

We construct the factor graph for the problem following the algorithm 3.1. This factor graph depicts how the joint probability distribution (of two entities and one relation) is factorized into smaller parts defined by the constraints. *Circular* nodes represent binary variables. Shaded variable nodes represent binary variable

nodes for each base classifier; whereas unshaded nodes represent intermediate variables created to support logic-factors. Unary factors encode local classifier scores whereas squares with *rounded corners* represent logic-factors for this formulation.

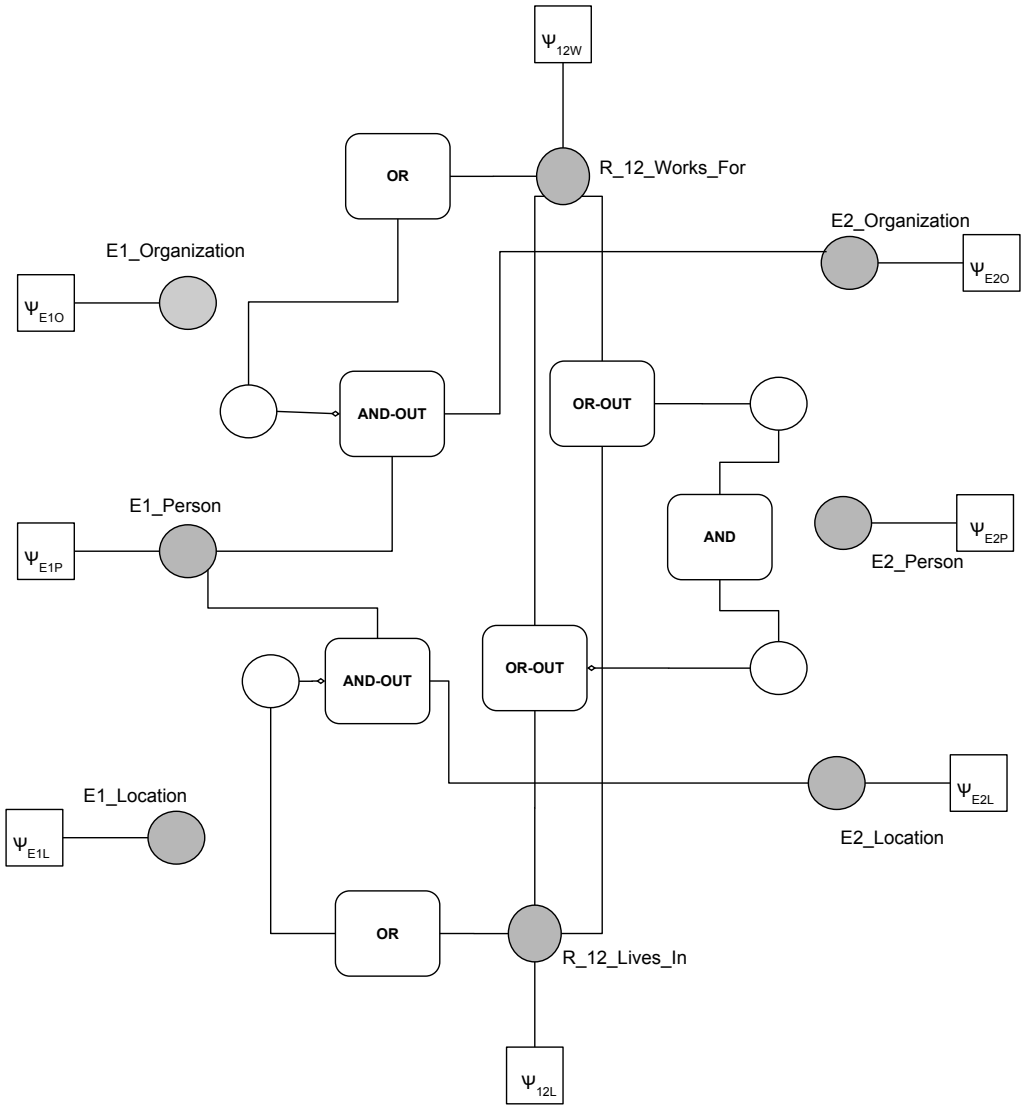

Figure A.1: Factor graph for an ER:UIUC problem instance

Some of the variable nodes combine to form additional variables and factor nodes. We create 4 intermediate variable nodes and 7 logic-factor nodes. This factor graph representation can be solved using any of the graphical models inference methods.