20 Global Inference for Entity and Relation Identification via a Linear Programming Formulation

Dan Roth and Wen-tau Yih

Natural language decisions often involve assigning values to sets of variables, representing low-level decisions and context-dependent disambiguation. In most cases there are complex relationships among these variables representing dependencies that range from simple statistical correlations to those that are constrained by deeper structural, relational, and semantic properties of the text.

In this chapter we study a specific instantiation of this problem in the context of identifying named entities and relations between them in free-form text. Given a collection of discrete random variables representing outcomes of learned local predictors for entities and relations, we seek an optimal global assignment to the variables that respects multiple constraints, including constraints on the type of arguments a relation can take, and the mutual activity of different relations.

We develop a linear programming formulation to address this global inference problem and evaluate it in the context of simultaneously learning named entities and relations. We show that global inference improves stand-alone learning; in addition, our approach allows us to efficiently incorporate expressive domain and task-specific constraints at decision time, resulting, beyond significant improvements in the accuracy, in "coherent" quality of the inference.

20.1 Introduction

In a variety of AI problems there is a need to learn, represent, and reason with respect to definitions over structured and relational data. Examples include learning to identify properties of text fragments such as functional phrases and named entities, identifying relations such as "A is the assassin of B" in text, learning to classify molecules for mutagenicity from atom-bond data in drug design, learning to identify 3D objects in their natural surrounding, and learning a policy to map goals to actions in planning domains.

Learning to make decisions with respect to natural language input is a prime source of examples for the need to represent, learn, and reason with structured and relational data [12, 13, 16, 18]. Natural language tasks presents several challenges to statistical relational learning (SRL). It is necessary (1) to represent structured domain elements in the sense that their internal (hierarchical) structure can be encoded, and learning functions in these terms can be supported, and (2) it is essential to represent concepts and functions relationally, in the sense that different data instantiations may be abstracted to yield the same representation, so that evaluation of functions over different instantiations will produce the same output. Moreover, beyond having to deal with structured *input*, in many natural language understanding tasks there is a rich relational structure also on the *output* of predictors. Natural language decisions often depend on the outcomes of several different but mutually dependent predictions. These predictions must respect some constraints that could arise from the nature of the data or from domain-specific conditions. For example, in part-of-speech (POS)tagging, a sentence must have at least one verb, and cannot have three consecutive verbs. These facts can be used as constraints. In named entity recognition, "no entities overlap" is a common constraint used in various works [37]. When predicting whether phrases in sentences represent entities and determining their type, the relations between the candidate entities provide constraints on their allowed (or plausible) types, via selectional restrictions.

While the classifiers involved in these global decisions need to exploit the relational structure in the input [30], we will not discuss these issues here, and will focus here instead on the task of inference with the classifiers' outcomes. Namely, this work is concerned with the *relational structure over the outcomes of predictors*, and studies natural language inferences which exploit the global structure of the problem, aiming at making decisions which depend on the outcomes of several different but mutually dependent classifiers.

Efficient solutions to problems of this sort have been given when the constraints on the predictors are *sequential* [25, 15]. These solutions can be categorized into the following two frameworks. The first, which we call *learning global models*, trains a probabilistic model under the constraints imposed by the domain. Examples include variations of hidden Markov models (HMMs), conditional models, and sequential variations of Markov random fields (MRFs) [21]. The other framework, *inference with classifiers* [28], views maintaining constraints and learning component classifiers as separate processes. Various local classifiers are trained without the knowledge of the global output constraints. The predictions are taken as input to an inference procedure which is given these constraints and then finds the best global prediction. In addition to the conceptual simplicity and modularity of this approach, it is more efficient than the global training approach, and seems to perform better experimentally in some tasks [37, 26, 32].

Typically, efficient inference procedures in both frameworks rely on dynamic programming (e.g., Viterbi), which works well for sequential data. However, in many important problems, the structure is more general, resulting in computationally intractable inference. Problems of these sorts have been studied in computer vision, where inference is generally performed over low-level measurements rather than over higher-level predictors [22, 3].

This work develops a novel *inference with classifiers* approach. Rather than being restricted to sequential data, we study a fairly general setting. The problem is defined in terms of a collection of discrete random variables representing binary relations and their arguments; we seek an optimal assignment to the variables in the presence of the constraints on the binary relations between variables and the relation types. Following ideas that were developed recently in the context of approximation algorithms [8], we model inference as an optimization problem, and show how to cast it in a linear programming (LP) formulation. Using existing numerical packages, which are able to solve very large LP problems in a very short time[1], inference can be done very quickly.

Our approach could be contrasted with other approaches to sequential inference or to general MRF approaches [21, 35]. The key difference is that in these approaches, the model is learned globally, under the constraints imposed by the domain. Our approach is designed to address also cases in which some of the local classifiers are learned (or acquired otherwise) in other contexts and at other times, or incorporated as background knowledge. That is, some components of the global decision need not, or cannot, be trained in the context of the decision problem. This way, our approach allows the incorporation of constraints into decisions in a dynamic fashion and can therefore support task-specific inference. The significance of this is clearly shown in our experimental results.

We develop our model in the context of natural language inference and evaluate it here on the problem of *simultaneously* recognizing named entities and relations between them.

For instance, in the sentence `J. V. Oswald was murdered at JFK after his assassin, R. U. KFJ shot...`, we want to identify the *kill (KFJ, Oswald)* relation. This task requires making several local decisions, such as identifying named entities in the sentence, in order to support the relation identification. For example, it may be useful to identify that Oswald and KFJ are *people*, and JFK is a *location*. This, in turn, may help to identify that a *kill* action is described in the sentence. At the same time, the relation *kill* constrains its arguments to be *people* (or at least, not to be *location*s) and helps to enforce that Oswald and KFJ are likely to be *people*, while JFK may not.

In our model, we first learn a collection of "local" predictors, e.g., entity and relation identifiers. At decision time, given a sentence, we produce a global decision

1. For example, CPLEX [11] is able to solve a linear programming problem of 13 million variables within 5 minutes.

that optimizes over the suggestions of the classifiers that are active in the sentence, known constraints among them and, potentially, domain-specific or task-specific constraints relevant to the current decision. Although a brute-force algorithm may seem feasible for short sentences, as the number of entity variables grows, the computation becomes intractable very quickly. Given n entities in a sentence, there are $O(n^2)$ possible binary relations between them. Assume that each variable (entity or relation) can take l labels ("none" is one of these labels). Thus, there are l^{n^2} possible assignments, which is too large to explicitly enumerate even for a small n.

When evaluated on simultaneous learning of named entities and relations, our approach not only provides a significant improvement in the predictors' accuracy; more importantly, it provides *coherent* solutions. While many statistical methods make "incoherent" mistakes (i.e., inconsistency among predictions) that no human ever makes, as we show, our approach improves also the *quality* of the inference significantly.

The rest of the chapter is organized as follows. Section 20.2 formally defines our problem and section 20.3 describes the computational approach we propose. Experimental results are given in section 20.5, including a case study that illustrates how our inference procedure improves the performance. We introduce some common inference methods used in various text problems as comparison in section 20.6, followed by some discussions and conclusions in section 20.7.

20.2 The Relational Inference Problem

We consider the relational inference problem within the *reasoning with classifiers* paradigm, and study a specific but fairly general instantiation of this problem, motivated by the problem of recognizing named entities (e.g., persons, locations, organization names) and relations between them (e.g. work_for, located_in, live_in). Conceptually, the entities and relations can be viewed, taking into account the mutual dependencies, as shown in figure 20.1, where the nodes represent entities (e.g., phrases) and the links denote the binary relations between the entities. Each entity and relation has several properties. Some of the properties, such as words inside the entities and POS tags of words in the context of the sentence, are easy to acquire. However, other properties like the semantic types (i.e., class labels, such as "people" or "locations") of phrases are difficult. Identifying the labels of entities and relations is treated here as a learning problem. In particular, we learn these target properties as functions of all other properties of the sentence.

To describe the problem in a formal way, we first define sentences and entities as follows.

Definition 20.1 Sentence and Entities
A sentence S is a linked list which consists of words w and entities $\mathcal{E}$. An entity can be a single word or a set of consecutive words with a predefined boundary. Entities in a sentence are labeled as $\mathcal{E} = \{E_1, E_2, \cdots, E_n\}$ according to their order, and

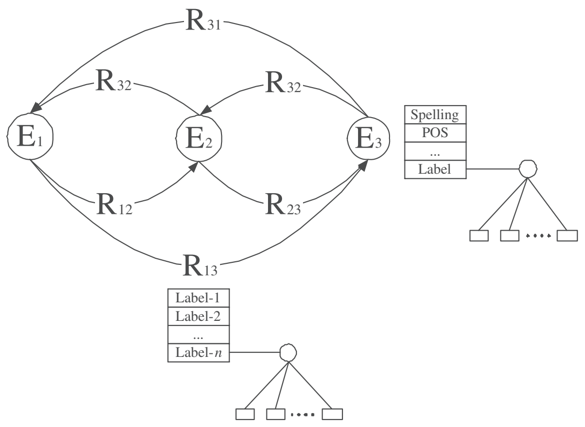

Figure 20.1 A conceptual view of entities and relations.

they take values (i.e., labels) that range over a set of entity types $\mathcal{L}_\mathcal{E}$. The value assigned to $E_i \in \mathcal{E}$ is denoted $f_{E_i} \in \mathcal{L}_\mathcal{E}$.

Notice that determining the entity boundaries is also a difficult problem – the *segmentation* (or *phrase detection*) problem [1, 25]. Here we assume it is solved and given to us as input; thus we only concentrate on classification.

$\underset{\textbf{E1}}{\boxed{\text{Dole}}}$'s wife , $\underset{\textbf{E2}}{\boxed{\text{Elizabeth}}}$, is a native of $\underset{\textbf{E3}}{\boxed{\text{Salisbury , N.C.}}}$

Figure 20.2 A sentence that has three entities.

Example 20.1
The sentence in figure 20.2 has three entities: E_1 = "Dole", E_2 = "Elizabeth," and E_3 = "Salisbury, N.C."

A relation is defined by the entities that are involved in it (its arguments). Note that we only discuss binary relations.

Definition 20.2 Relations
A (binary) relation $R_{ij} = (E_i, E_j)$ represents the relation between E_i and E_j, where E_i is the first argument and E_j is the second. In addition, R_{ij} can range over a set of entity types $\mathcal{L}_\mathcal{R}$. We use $\mathcal{R} = \{R_{ij}\}_{\{1 \le i,j \le n; i \ne j\}}$ as the set of binary relations on the entities $\mathcal{E}$ in a sentence. Two special functions $\mathcal{N}^1$ and $\mathcal{N}^2$ are used to indicate the argument entities of a relation R_{ij}. Specifically, $E_i = \mathcal{N}^1(R_{ij})$ and $E_j = \mathcal{N}^2(R_{ij})$.

Note that in this definition, the relations are directed (e.g., there are both R_{ij} and R_{ji} variables). This is because the arguments in a relation often take different roles and have to be distinguished. Examples of this sort include *work_for*, *located_in* and *live_in*. If a relation variable R_{ij} is predicted as a mutual relation (e.g., *spouse_of*), then the corresponding relation R_{ji} should be also assigned the same label. This additional constraint can be easily incorporated in our inference framework. Also notice that we simplify the definition slightly by not considering self-relations (e.g., R_{ii}). This can be relaxed if this type of relations appears in the data.

Example 20.2
In the sentence given in figure 20.2, there are six relations between the entities: R_{12} = ("Dole", "Elizabeth"), R_{21} = ("Elizabeth," "Dole"), R_{13} = ("Dole," "Salisbury, N.C."), R_{31} = ("Salisbury, N.C.," "Dole"), R_{23} = ("Elizabeth," "Salisbury, N.C."), and R_{32} = ("Salisbury, N.C.," "Elizabeth")

We define the types (i.e., classes) of relations and entities as follows.

Definition 20.3 Classes
We denote the set of predefined entity classes and relation classes as $\mathcal{L}_{\mathcal{E}}$ and $\mathcal{L}_{\mathcal{R}}$ respectively. $\mathcal{L}_{\mathcal{E}}$ has one special element, *other_ent*, which represents any unlisted entity class. Similarly, $\mathcal{L}_{\mathcal{R}}$ also has one special element, *other_rel*, which means the involved entities are irrelevant or the relation class is undefined.

When it is clear from the context, we use E_i and R_{ij} to refer to the entity and relation, as well as their types (class labels). Note that each relation and entity variable can take only one class according to definition 20.3. Although there may be different relations between two entities, it seldom occurs in the data. Therefore, we ignore this issue for now.

Example 20.3
Suppose $\mathcal{L}_{\mathcal{E}}$ = { *other_ent*, *person*, *location* } and $\mathcal{L}_{\mathcal{R}}$ = { *other_rel*, *born_in*, *spouse_of* }. For the entities in figure 20.2, E_1 and E_2 belong to *person* and E_3 belongs to *location*. In addition, relation R_{23} is *born_in*, R_{12} and R_{21} are *spouse_of*. Other relations are *other_rel*.

Given a sentence, we want to predict the labels of a set $\mathcal{V}$ which consists of two types of variables – entities $\mathcal{E}$ and relations $\mathcal{R}$. That is, $\mathcal{V} = \mathcal{E} \cup \mathcal{R}$. However, the class label of a single entity or relation depends not only on its local properties but also on the properties of other entities and relations. The classification task is somewhat difficult since the predictions of entity labels and relation labels are mutually dependent. For instance, the class label of E_1 depends on the class label of R_{12} and the class label of R_{12} also depends on the class label of E_1 and E_2. While we can assume that all the data is annotated for training purposes, this cannot be assumed at evaluation time. We may presume that some local properties, such as the words or POS tags, are given, but none of the class labels for entities or relations are.

To simplify the complexity of the interaction within the graph but still preserve the characteristic of mutual dependency, we abstract this classification problem in the following probabilistic framework. First, the classifiers are trained independently and used to estimate the probabilities of assigning different labels given the observation (that is, the easily classified properties in it). Then, the output of the classifiers is used as a conditional distribution for each entity and relation, given the observation. This information, along with the constraints among the relations and entities, is used to make global inference.

In the task of entity and relation recognition, there exist some constraints on the labels of corresponding relation and entity variables. For instance, if the relation is *live_in*, then the first entity should be a *person*, and the second entity should be a *location*. The correspondence between the relation and entity variables can be represented by a bipartite graph. Each relation variable R_{ij} is connected to its first entity E_i , and second entity E_j. We define a set of constraints on the outcomes of the variables in $\mathcal{V}$ as follows.

Definition 20.4 Constraints
A constraint is a function that maps a relation label and an entity label to either 0 or 1 (contradict or satisfy the constraint). Specifically, $\mathcal{C}^1 : \mathcal{L}_\mathcal{R} \times \mathcal{L}_\mathcal{E} \rightarrow \{0, 1\}$ constrains values of the first argument of a relation. $\mathcal{C}^2$ is defined similarly and constrains values of the second argument.

Note that while we define the constraints here as Boolean functions, our formalism allows us to associate weights with constraints and to include statistical constraints [32]. Also note that we can define a large number of constraints, such as $\mathcal{C}^R : \mathcal{L}_\mathcal{R} \times \mathcal{L}_\mathcal{R} \rightarrow \{0, 1\}$ which constrain the labels of two relation variables. For example, we can define a set of constraints on a mutual relation *spouse_of* as $\{(\textit{spouse\_of}, \textit{spouse\_of}) = 1$, $(\textit{spouse\_of}, l_r) = 0$, and $(l_r, \textit{spouse\_of}) = 0$ for any $l_r \in \mathcal{L}_\mathcal{R}$, where $l_r \neq \textit{spouse\_of}\}$. By enforcing these constraints on a pair of symmetric relation variables R_{ij} and R_{ji}, the relation class *spouse_of* will be assigned to either both R_{ij} and R_{ji} or none of them. [In fact, as will be clear in section 20.3, the language used to describe constraints is very rich – linear (in)equalities over $\mathcal{V}$.]

We seek an inference algorithm that can produce a coherent labeling of entities and relations in a given sentence. Furthermore, it optimizes an objective function based on the conditional probabilities or other confidence scores estimated by the entity and relation classifiers, subject to some natural constraints. Examples of these constraints include whether specific entities can be the argument of specific relations, whether two relations can occur together among a subset of entity variables in a sentence, and any other information that might be available at the inference time. For instance, suppose it is known that entities A and B represent the same location; one may like to incorporate an additional constraint that prevents an inference of the type: “C lives in A; C does not live in B.”

We note that a large number of problems can be modeled this way. Examples include problems such as chunking sentences [25], coreference resolution and sequencing problems in computational biology, and the recently popular problem of semantic role labeling [5, 6]. In fact, each of the components of our problem here, namely the separate task of recognizing named entities in sentences and the task of recognizing semantic relations between phrases, can be modeled this way. However, our goal is specifically to consider interacting problems at different levels, resulting in more complex constraints among them, and exhibit the power of our method.

20.3 Integer Linear Programming Inference

The most direct way to formalize our inference problem is using MRFs [9]. Rather than doing that, for computational reasons, we first use a fairly standard transformation of MRFs to a discrete optimization problem (see [19] for details). Specifically, under weak assumptions we can view the inference problem as the following optimization problem, which aims at minimizing the objective function that is the sum of the following two cost functions.

Assignment cost This is the cost of deviating from the assignment of the variables $\mathcal{V}$ given by the classifiers. The specific cost function we use is defined as follows: Let l be the label assigned to variable $u \in \mathcal{V}$. If the posterior probability estimation is $p = P(f_u = l|\bar{x})$, where $\bar{x}$ represents the input feature vector, then the assignment cost $c_u(l)$ is $-\log p$.

Constraint cost This is the cost imposed by breaking constraints between neighboring nodes. The specific cost function we use is defined as follows: Consider two entity nodes E_i, E_j and their corresponding relation node R_{ij}; that is, $E_i = \mathcal{N}^1(R_{ij})$ and $E_j = \mathcal{N}^2(R_{ij})$. The constraint cost indicates whether the labels are consistent with the constraints. In particular, we use: $d^1(f_{E_i}, f_{R_{ij}})$ is 0 if $(f_{R_{ij}}, f_{E_i}) \in \mathcal{C}^1$; otherwise, $d^1(f_{E_i}, f_{R_{ij}})$ is ∞ [2]. Similarly, we use d^2 to force the consistency of the second argument of a relation.

Since we are looking for the most probable global assignment that satisfies the constraints, the overall cost function we optimize for a global labeling f of all variables is

$$C(f) = \sum_{u \in \mathcal{V}} c_u(f_u) + \sum_{R_{ij} \in \mathcal{R}} \left[d^1(f_{R_{ij}}, f_{E_i}) + d^2(f_{R_{ij}}, f_{E_j}) \right] \tag{20.1}$$

Unfortunately, this combinatorial problem (20.1) is computationally intractable even when placing assumptions on the cost function [19]. The computational approach we adopt is to develop a *linear programming* formulation of the problem, and then solve the corresponding *integer linear programming* (ILP) problem[3]. Our LP formulation is based on the method proposed by Chekuri et al. [8]. Since the objective function (20.1) is not a linear function in terms of the labels, we introduce

2. In practice, we use a very large number (e.g., 9^{15}).
3. In this chapter, ILP only means *integer linear programming*, not inductive logic programming.

new binary variables to represent different possible assignments to each original variable; we then represent the objective function as a linear function of these binary variables.

Let $x_{\{u,i\}}$ be an indicator variable, defined to be 1 if and only if variable u is labeled i and 0 otherwise, where $u \in \mathcal{E}, i \in \mathcal{L}_{\mathcal{E}}$ or $u \in \mathcal{R}, i \in \mathcal{L}_{\mathcal{R}}$. For example, $x_{\{E_1,person\}} = 1$ when the label of entity E_1 is *person*; $x_{\{R_{23},spouse\_of\}} = 0$ when the label of relation R_{23} is not *spouse_of*. Let $x_{\{R_{ij},r,E_i,e_1\}}$ be an indicator variable representing whether relation R_{ij} is assigned label r and its first argument, E_i, is assigned label e_1. For instance, $x_{\{R_{12},spouse\_of,E_1,person\}} = 1$ means the label of relation R_{12} is *spouse_of* and the label of its first argument, E_1, is *person*. Similarly, $x_{\{R_{ij},r,E_j,e_2\}} = 1$ indicates that R_{ij} is assigned label r and its second argument, E_j, is assigned label e_2. With these definitions, the optimization problem can be represented as the following integer linear program.

$$\min \sum_{E\in\mathcal{E}} \sum_{e\in\mathcal{L}_{\mathcal{E}}} c_E(e) \cdot x_{\{E,e\}} + \sum_{R\in\mathcal{R}} \sum_{r\in\mathcal{L}_{\mathcal{R}}} c_R(r) \cdot x_{\{R,r\}}$$

$$+ \sum_{\substack{E_i,E_j\in\mathcal{E}\\E_i\neq E_j}} \left[\sum_{r\in\mathcal{L}_{\mathcal{R}}} \sum_{e_1\in\mathcal{L}_{\mathcal{E}}} d^1(r,e_1) \cdot x_{\{R_{ij},r,E_i,e_1\}} + \sum_{r\in\mathcal{L}_{\mathcal{R}}} \sum_{e_2\in\mathcal{L}_{\mathcal{E}}} d^2(r,e_2) \cdot x_{\{R_{ij},r,E_j,e_2\}} \right],$$

subject to:

$$\sum_{e\in\mathcal{L}_{\mathcal{E}}} x_{\{E,e\}} = 1 \qquad \forall E \in \mathcal{E} \tag{20.2}$$

$$\sum_{r\in\mathcal{L}_{\mathcal{R}}} x_{\{R,r\}} = 1 \qquad \forall R \in \mathcal{R} \tag{20.3}$$

$$x_{\{E,e\}} = \sum_{r\in\mathcal{L}_{\mathcal{R}}} x_{\{R,r,E,e\}} \qquad \begin{array}{l}\forall E \in \mathcal{E}, e \in \mathcal{L}_{\mathcal{E}},\\ \forall R \in \{R : E = \mathcal{N}^1(R) \text{ or } E = \mathcal{N}^2(R)\}\end{array} \tag{20.4}$$

$$x_{\{R,r\}} = \sum_{e\in\mathcal{L}_{\mathcal{E}}} x_{\{R,r,E,e\}} \qquad \forall R \in \mathcal{R}, r \in \mathcal{L}_{\mathcal{R}}, E = \mathcal{N}^1(R) \tag{20.5}$$

$$x_{\{R,r\}} = \sum_{e\in\mathcal{L}_{\mathcal{E}}} x_{\{R,r,E,e\}} \qquad \forall R \in \mathcal{R}, r \in \mathcal{L}_{\mathcal{R}}, E = \mathcal{N}^2(R) \tag{20.6}$$

$$x_{\{E,e\}} \in \{0,1\} \qquad \forall E \in \mathcal{E}, e \in \mathcal{L}_{\mathcal{E}} \tag{20.7}$$

$$x_{\{R,r\}} \in \{0,1\} \qquad \forall R \in \mathcal{R}, r \in \mathcal{L}_{\mathcal{R}} \tag{20.8}$$

$$x_{\{R,r,E,e\}} \in \{0,1\} \qquad \forall R \in \mathcal{R}, r \in \mathcal{L}_{\mathcal{R}}, E \in \mathcal{E}, e \in \mathcal{L}_{\mathcal{E}} \tag{20.9}$$

Equations (20.2) and (20.3) require that each entity or relation variable can only be assigned one label. Equations (20.4), (20.5), and (20.6) assure that the assignment to each entity or relation variable is consistent with the assignment to its neighboring variables. Equations (20.7), (20.8), and (20.9) are the integral constraints on these binary variables.

There are several advantages of representing the problem in an LP formulation. First of all, linear (in)equalities are fairly general and are able to represent many types of constraints (e.g., the decision time constraint in the experiment in section 20.5). More importantly, an ILP problem at this scale can be solved very quickly using current numerical packages, such as Xpress-MP [42] or CPLEX [11]. We introduce the general strategies of solving an ILP problem here.

20.4 Solving Integer Linear Programming

To solve an ILP problem, a straightforward idea is to *relax* the integral constraints. That is, replacing (20.7), (20.8), and (20.9) with

$$x_{\{E,e\}} \geq 0 \qquad \forall E \in \mathcal{E}, e \in \mathcal{L}_{\mathcal{E}} \tag{20.10}$$

$$x_{\{R,r\}} \geq 0 \qquad \forall R \in \mathcal{R}, r \in \mathcal{L}_{\mathcal{R}} \tag{20.11}$$

$$x_{\{R,r,E,e\}} \geq 0 \qquad \forall R \in \mathcal{R}, r \in \mathcal{L}_{\mathcal{R}}, E \in \mathcal{E}, e \in \mathcal{L}_{\mathcal{E}}, \tag{20.12}$$

If linear programming relaxation (LPR) returns an integer solution, then it is also the optimal solution to the ILP problem. In fact, it can be shown that the optimal solution of a linear program is always integral if the coefficient matrix of its standard form is *unimodular* [34].

Definition 20.5
A matrix $\mathbf{A}$ of rank m is called *unimodular* if all the entries of $\mathbf{A}$ are integers, and the determinant of every square submatrix of $\mathbf{A}$ of order m is in 0,+1,-1.

Theorem 20.6 Veinott and Dantzig
Let $\mathbf{A}$ be an (m,n)-integral matrix with full row rank m. Then the polyhedron $\{\mathbf{x}|\mathbf{x} \geq 0; \mathbf{A}\mathbf{x} = \mathbf{b}\}$ is integral for each integral vector $\mathbf{b}$, if and only if $\mathbf{A}$ is unimodular.

Theorem 20.6 indicates that if a linear program is in its standard form, then regardless of the cost function and the integral vector $\mathbf{b}$, the optimal solution is an integer point if and only if the coefficient matrix $\mathbf{A}$ is unimodular.

When LPR returns a noninteger solution, the ILP problem is usually handled by one of the two strategies: *rounding* and *search*.

The goal of *rounding* is to find an integer point that is close to the noninteger solution. Under some conditions of the cost function, which do not hold in our problem, a well-designed rounding algorithm can be shown that the rounded solution is a good approximation to the optimal solution [19, 8]. Nevertheless, in general, the outcomes of the rounding procedure may not even be a legitimate solution to the problem.

To find the optimal solution of an ILP problem, a *search* approach based on the idea of *branch and bound* divides an ILP problem into several LP subproblems, and uses the noninteger solutions returned by an LP solver to reduce the search space.

When LPR finds a noninteger solution, it splits the problem on the noninteger variable. For example, suppose variable x_i is fractional in a noninteger solution to the ILP problem $\min\{cx : x \in S, x \in \{0,1\}^n\}$, where S is the linear constraints. The ILP problem can be split into two sub-LPR problems, $\min\{cx : x \in S \cap \{x_i = 0\}\}$ and $\min\{cx : x \in S \cap \{x_i = 1\}\}$. Since any feasible solution provides an upper bound and any LPR solution generates a lower bound, the search tree can be effectively cut.

One technique that is often combined with *branch and bound* is *cutting plane*. When a noninteger solution is given by LPR, it adds a new linear constraint that makes the noninteger point infeasible, while still keeping the optimal integer solution in the feasible region. As a result, the feasible region is closer to the ideal polyhedron, which is the convex hull of feasible integer solutions. The most well-known cutting plane algorithm is Gomory's fractional cutting plane method [41], for which it can be shown that only a finite number of additional constraints are needed. Moreover, researchers developed different cutting plane algorithms for different types of ILP problems. One example is [40], which only focuses on binary ILP problems.

In theory, a search-based strategy may need several steps to find the optimal solution. However, LPR always generates integer solutions for *all* the (thousands of) cases we have experimented with, even though the coefficient matrix in our problem is not unimodular.

20.5 Experiments

We describe below two sets of experiments on the problem of simultaneously recognizing entities and relations. In the first, we view the task as a knowledge acquisition task – we let the system read sentences and identify entities and relations among them. Given that this is a difficult task which may require quite often information beyond the sentence, we consider also a "forced decision" task, in which we simulate a question-answering situation – we ask the system, say, "Who killed whom?" and evaluate it on identifying correctly the relation and its arguments, given that it is known that somewhere in this sentence this relation is active. In addition, this evaluation exhibits the ability of our approach to incorporate task specific constraints at decision time. At the end of this section, we will also provide a case study to illustrate how the inference procedure corrects mistakes both in entity and relation predictions.

20.5.1 Data Preparation

We annotated the named entities and relations in some sentences from the TREC documents. In order to effectively observe the interaction between relations and entities, we chose 1437 sentences[4] that have at least one active relation. Among those sentences, there are 5336 entities, and 19,048 pairs of entities (binary relations). Entity labels include 1685 *persons*, 1968 *locations*, 978 *organizations*, and 705 *other_ent*. Relation labels include 406 *located_in*, 394 *work_for*, 451 *orgBased_in*, 521 *live_in*, 268 *kill*, and 17,007 *other_rel*. Note that most pairs of entities have no active relations at all. Therefore, relation *other_rel* significantly outnumbers others. Examples of each relation label and the constraints between a relation variable and its two entity arguments are shown in table 20.1.

Table 20.1 Relations of interest and the corresponding constraints

Relation	Entity1	Entity2	Example
located_in	loc	loc	(New York, US)
work_for	per	org	(Bill Gates, Microsoft)
orgBased_in	org	loc	(HP, Palo Alto)
live_in	per	loc	(Bush, US)
kill	per	per	(Oswald, JFK)

20.5.2 Tested Approaches

In order to focus on the evaluation of our inference procedure, we assume the problem of *segmentation* (or *phrase detection*) [1, 25] is solved, and the entity boundaries are given to us as input; thus we only concentrate on their classifications.

We evaluate our LP-based inference procedure by observing its effect in different approaches of combining the entity and relation classifiers. The first approach is to train entity and relation classifiers *separately*. In particular, the relation classifier does not know the labels of its entity arguments, and the entity classifier does not know the labels of relations in the sentence, either. For the entity classifier, one set of features is extracted from words within a size 4 window around the target phrase. They are (1) words, POS tags, and conjunctions of them; and (2) bigrams and trigrams of the mixture of words and tags. In addition, some other features are extracted from the target phrase, which are listed in table 20.2.

For the relation classifier, there are three sets of features:

1. features similar to those used in the entity classification are extracted from the two argument entities of the relation;
2. conjunctions of the features from the two arguments;
3. some patterns extracted from the sentence or between the two arguments.

4. The data used here is available by following the data link from `http://L2R.cs.uiuc.edu/∼cogcomp/`
5. We collected names of famous places, people, and popular titles from other data sources in advance.

Table 20.2 Some features extracted from the target phrase

Symbol	Explanation
icap	the first character of a word is capitalized
acap	all characters of a word are capitalized
incap	some characters of a word are capitalized
suffix	the suffix of a word is "ing," "ment," etc.
bigram	bigram of words in the target phrase
len	number of words in the target phrase
place[5]	the phrase is/has a known place's name
prof[5]	the phrase is/has a professional title (e.g., Lt.)
name[5]	the phrase is/has a known person's name

Table 20.3 Some patterns used in relation classification

Pattern	Example
arg_1 , arg_2	San Jose, CA
arg_1 , ⋯ a ⋯ arg_2 *prof*	John Smith, a Starbucks manager ⋯
in/at arg_1 in/at/, arg_2	Officials in Perugia in Umbria province said ⋯
arg_2 *prof* arg_1	CNN reporter David McKinley ⋯
arg_1 ⋯ native of ⋯ arg_2	Elizabeth Dole is a native of Salisbury, N.C.
arg_1 ⋯ based in/at arg_2	⋯ a manager for Kmart based in Troy, Mich. said ⋯

Some features in category 3 are "the number of words between arg_1 and arg_2 ," "whether arg_1 and arg_2 are the same word," or "arg_1 is the beginning of the sentence and has words that consist of all capitalized characters," where arg_1 and arg_2 represent the first and second argument entities respectively. Table 20.3 presents some patterns we use.

The learning algorithm used is a regularized variation of the Winnow update rule incorporated in SNoW [29, 31, 4], a multiclass classifier that is specifically tailored for large-scale learning tasks. SNoW learns a sparse network of linear functions, in which the targets (entity classes or relation classes, in this case) are represented as linear functions over a common feature space. While SNoW can be used as a classifier and predicts using a winner-take-all mechanism over the activation value of the target classes, we can also rely directly on the raw activation value it outputs, which is the weighted linear sum of the active features, to estimate the posteriors. It can be verified that the resulting values provide a good source of probability estimation. We use softmax [2] over the raw activation values as conditional probabilities. Specifically, suppose the number of classes is n, and the raw activation values of class i is act_i. The posterior estimation for class i is derived by the following equation

$$p_i = \frac{e^{act_i}}{\sum_{1 \le j \le n} e^{act_j}}.$$

In addition to the *separate* approach, we also test several pipeline models, which we denote $E \rightarrow R$, $R \rightarrow E$ and $E \leftrightarrow R$. The $E \rightarrow R$ approach first trains the basic entity classifier (E), which is identical to the entity classifier in the *separate* approach. Its predictions on the two entity arguments of a relation are then used conjunctively as additional features (e.g., *person–person* or *person–location*) in learning the relation classifier (R). Similarly, $R \rightarrow E$ first trains the relation classifier (R); its output is then used as additional features in the entity classifier (E). For example, the additional feature could be "this entity is predicted as the first argument of a *work_for* relation." The $E \leftrightarrow R$ model is the combination of the above two. It uses the entity classifier in the $R \rightarrow E$ model and the relation classifier in the $E \rightarrow R$ model as its final classifiers.

Although the true labels of entities and relations are known during training, only the *predicted* labels are available during evaluation on new data (and in testing). Therefore, rather than training the second-stage pipeline classifiers on the available *true* labels, we train them on the *predictions* of the previous stage classifiers. This way, at test time the classifiers are being evaluated on data of the same type they were trained on, making the second-stage classifier more tolerant to the mistakes [6]. The need to train pipeline classifiers this way has been observed multiple times in natural language processing (NLP) research, and we also have validated it in our experiments. For example, when the relation classifier is trained using the true entity labels, the performance is usually worse than when training it using the predicted entity labels.

The last approach, *omniscient*, tests the conceptual upper bound of this entity-relation classification problem. It also trains the two classifiers separately. However, it assumes that the entity classifier knows the *correct* relation labels, and similarly the relation classifier knows the *right* entity labels as well. This additional information is then used as features in training and testing. Note that this assumption is unrealistic. Nevertheless, it may give us a hint on how accurately the classifiers with global inference can achieve. Finally, we apply the LP-based inference procedure to the above five models, and observe how it improves the performance.

20.5.3 Results

We test the aforementioned approaches using five fold cross-validation. For each approach, we also perform a paired t-test on its F_1 scores before and after inference. Tables 20.4 and 20.5 show the performance of each approach in *recall*, *precision*, and F_1.

The results show that the inference procedure consistently improves the performance of the five models, both in entities and relations. One interesting observation

6. In order to derive similar performance in testing, ideally the previous stage classifier should be trained using a different corpus. We didn't take this approach because of data scarcity.

Table 20.4 Results of the entity classification in different approaches. Experiments are conducted using five fold cross-validation. Numbers in boldface indicate that the p-values are smaller than 0.1. Symbols † and ‡ indicate significance at 95% and 99% levels respectively. Significance tests were computed with a two-tailed paired t-test.

Approach	person			location			organization		
	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1
Separate	89.5	89.8	89.4	87.0	91.5	89.0	67.6	91.3	77.0
Separate w/ Inf	90.5	90.6	90.4	88.6	91.8	**90.1**	71.0	91.2	**79.4**
E → R	89.5	89.8	89.4	87.0	91.5	89.0	67.6	91.3	77.0
E → R w/ Inf	89.7	90.1	**89.7**†	87.0	91.7	89.1	69.0	91.2	78.0
R → E	89.1	88.7	88.6	88.1	89.8	88.9	71.4	89.3	78.7
R → E w/ Inf	88.6	88.6	88.3	88.2	89.4	88.7	72.1	88.5	79.0
E ↔ R	89.1	88.7	88.6	88.1	89.8	88.9	71.4	89.3	78.7
E ↔ R w/ Inf	89.5	89.1	**89.0**	88.7	89.7	**89.1**	72.0	89.5	79.2
Omniscient	94.9	93.7	94.2	92.4	96.6	94.4	88.1	93.5	90.7
Omniscient w/ Inf	96.1	94.2	**95.1**‡	94.0	97.0	**95.4**	88.7	94.9	91.7

is that the *omniscient* classifiers, which know the correct entity or relation labels, can still be improved by the inference procedure. This demonstrates the effectiveness of incorporating constraints, even when the learning algorithm may be able to learn them from the data.

One of the more significant results in our experiments, we believe, is the improvement in the *quality* of the decisions. As mentioned in section 20.1, incorporating constraints helps to avoid inconsistency in classification. It is interesting to investigate how often such mistakes happen without global inference, and see how effective the global inference is.

For this purpose, we define the *quality* of the decision as follows. For a relation variable and its two corresponding entity variables, if the labels of these variables are predicted correctly and the relation is active (i.e., not *other_rel*), then we count it as a *coherent* prediction. *Quality* is then the number of *coherent* predictions divided by the sum of *coherent* and *incoherent* predictions. When the inference procedure is not applied, 5% to 25% of the predictions are incoherent. Therefore, the quality is not always good. On the other hand, our global inference procedure takes the natural constraints into account, so it never generates incoherent predictions. If the relation classifier has the correct entity labels as features, a good learner should learn the constraints as well. As a result, the quality of *omniscient* is almost as good as *omniscient with inference*.

Another experiment we performed is the *forced decision* test, which boosts the F_1 score of the “kill” relation to 86.2%. In this experiment, we assume that the system knows which sentences have the “kill” relation at the decision time, but it does not know which pair of entities have this relation. We force the system to determine

Table 20.5 Results of the relation classification in different approaches. Experiments are conducted using five-fold cross-validation. Numbers in boldface indicates that that the p-values are smaller than 0.1. Symbols † and ‡ indicate significance at 95% and 99% levels respectively. Significance tests were computed with a two-tailed paired t-test.

Approach	located_in			work_for			orgBased_in		
	Rec	Prec	F_1	Rec	Prec	F_1	Rec	Prec	F_1
Separate	53.0	43.3	45.2	41.9	55.1	46.3	35.6	85.4	50.0
Separate w/ Inf	51.6	56.3	**50.5**‡	40.1	74.1	**51.2**	35.7	90.8	50.8
$E \rightarrow R$	56.4	52.5	50.7	44.4	60.8	51.2	42.1	77.8	54.3
$E \rightarrow R$ w/ Inf	55.7	53.2	50.9	42.9	72.1	**53.5**†	42.3	78.0	54.5
$R \rightarrow E$	53.0	43.3	45.2	41.9	55.1	46.3	35.6	85.4	50.0
$R \rightarrow E$ w/ Inf	53.0	49.8	**49.1**†	41.6	67.5	**50.4**	36.6	87.1	51.2
$E \leftrightarrow R$	56.4	52.5	50.7	44.4	60.8	51.2	42.1	77.8	54.3
$E \leftrightarrow R$ w/ Inf	55.7	53.9	51.3	42.3	72.0	**53.1**	41.6	79.8	54.3
Omniscient	62.9	59.5	57.5	50.3	69.4	58.2	50.3	77.9	60.9
Omniscient w/ Inf	62.9	61.9	**59.1**	50.3	79.2	**61.4**†	50.9	81.7	**62.5**‡

Approach	live_in			kill		
	Rec	Prec	F_1	Rec	Prec	F_1
Separate	39.7	61.7	48.0	81.5	75.3	77.6
Separate w/ Inf	41.7	68.2	**51.4**†	80.8	82.7	81.4
$E \rightarrow R$	50.0	58.9	53.5	81.5	73.0	76.5
$E \rightarrow R$ w/ Inf	50.0	57.7	53.0	80.6	77.2	78.3
$R \rightarrow E$	39.7	61.7	48.0	81.5	75.3	77.6
$R \rightarrow E$ w/ Inf	40.6	64.1	49.4	81.5	79.7	80.1
$E \leftrightarrow R$	50.0	58.9	53.5	81.5	73.0	76.5
$E \leftrightarrow R$ w/ Inf	49.0	59.1	53.0	81.5	77.5	**79.0**†
Omniscient	56.1	61.7	58.2	81.4	76.4	77.9
Omniscient w/ Inf	57.3	63.9	**59.9**	81.4	79.9	79.9

which of the possible relations in a sentence (i.e., which pair of entities) has this "kill" relation

by adding the following linear inequality.

$$\sum_{R \in \mathcal{R}} x_{\{R,kill\}} \geq 1$$

This is equivalent to saying that at least one of the relation variables in the sentence should be labeled as "kill." Since this additional constraint only applies to on the sentences in which the "kill" relation is active, the inference results of other sentences are not changed. Note that it is a realistic situation (e.g., in the context of question answering) in that it adds an external constraint, not present at the time of learning the classifiers, and it evaluates the ability of our inference algorithm to cope with it. The results exhibit that our expectations are correct.

20.5.4 Case Study

Although tables 20.4 and 20.5 clearly demonstrate that the inference procedure improves the performance, it is interesting to see *how* it corrects the mistakes by examining a specific case. The following sentence is taken from a news article in our corpus. The eight entities are in boldface, labeled E_1 to E_8.

> At the proposal of the **Serb Radical Party**$_{|E_1}$, the Assembly elected political **Branko Vojnic**$_{|E_2}$ from **Beli Manastir**$_{|E_3}$ as its speaker, while **Marko Atlagic**$_{|E_4}$ and Dr. **Milan Ernjakovic**$_{|E_5}$, **Krajina**$_{|E_6}$ **Serb Democratic Party**$_{|E_7}$ (**SDS**$_{|E_8}$) candidates, were elected as deputy speakers.

Table 20.6 shows the probability distribution estimated by the basic classifiers, the predictions before and after the inference, along with the true labels. Table 20.7 provides this information for the relation variables. Because the values of most of them are *other_rel*, we only show a small set of them here.

Table 20.6 Example: Inference effect on entities' predictions: the true labels, the predictions before and after inference, and the probabilities estimated by the basic classifiers.

	Label	before Inf.	after Inf.	other	person	loc.	org
E_1	Org	Org	Org	0.21	0.13	0.06	**0.60**
E_2	Per	Other	Other	**0.46**	0.16	0.33	0.05
E_3	Loc	Loc	Loc	0.29	0.25	**0.31**	0.15
E_4	Per	Other	Other	**0.37**	0.20	0.33	0.10
E_5	***Per***	***Loc***	***Per***	0.10	0.31	**0.36**	0.23
E_6	Loc	Loc	Loc	0.24	0.05	**0.61**	0.10
E_7	***Org***	***Per***	***Org***	0.15	**0.41**	0.03	0.40
E_7	Org	Org	Org	0.35	0.17	0.11	**0.37**

In this example, the inference procedure corrects two variables – E_5 (Milan Ernjakovic) and E_7 (Serb Democratic Party). If we examine the probability distribution of these two entity variables in table 20.6, it is easy to see that the classifier has difficulty deciding whether E_5 is a person's name or location, and whether E_7 is a person or organization. The strong belief that there is a *work_for* relation between these two entities (see the row R_{57} in table 20.7) enables the inference procedure to correct this mistake. In addition, several relation predictions are also corrected from *work_for* to *other_rel* because they lack the support of the entity classifier.

Note that not every mistake can be rectified, as several *work_for* relations are misidentified as *other_rel*. This may be due to the fact that the relation *other_rel* can take any types of entities as its arguments. In some rare cases, the inference

Table 20.7 Example: Inference effect on relations' predictions: the true labels, the predictions before and after inference, and the probabilities estimated by the basic classifiers.

	Label	before Inf.	after Inf.	other_rel	located_in	work_for	org_in	live_in	kill
R_{23}	kill	other_rel	other_rel	**0.66**	0.10	0.03	0.03	0.11	0.08
R_{37}	***other_rel***	***work_for***	***other_rel***	0.38	0.07	**0.41**	0.02	0.10	0.02
R_{47}	work_for	other_rel	other_rel	**0.65**	0.05	0.19	0.02	0.06	0.03
R_{48}	work_for	other_rel	other_rel	**0.83**	0.06	0.03	0.02	0.04	0.03
R_{51}	other_rel	work_for	work_for	0.36	0.06	**0.42**	0.01	0.13	0.02
R_{52}	***other_rel***	***work_for***	***other_rel***	0.24	0.15	**0.28**	0.04	0.22	0.07
R_{56}	***other_rel***	***work_for***	***other_rel***	0.23	0.16	**0.35**	0.01	0.22	0.02
R_{57}	work_for	work_for	work_for	0.26	0.07	**0.44**	0.01	0.21	0.02
R_{58}	work_for	other_rel	other_rel	**0.58**	0.06	0.14	0.02	0.17	0.02
R_{67}	work_for	other_rel	other_rel	**0.67**	0.06	0.19	0.02	0.05	0.01
R_{68}	work_for	other_rel	other_rel	**0.76**	0.09	0.04	0.04	0.05	0.02

procedure may change a correct prediction to a wrong label. However, since this seldom happens, the overall performance is still improved after inference.

One interesting thing to notice is the efficiency of this ILP inference in practice. Using a Pentium III 800MHz machine, it takes less than 30 seconds to process all the 1437 sentences (5336 entity variables and 19,048 relation variables in total).

20.6 Comparison with Other Inference Methods

In this section, we provide a broader view of inference methods and place the ILP approach described in this chapter in this context. Our approach to the problem of learning with structured output decouples learning and inference stages. As mentioned earlier, this is not the only approach. In other approaches (e.g., [39, 36]), training can be done globally, coupled with the inference. Coupling training and inference has multiple effects on performance and time complexity, which we do not discuss here (but see [32, 26] for some comparative discussion) as we concentrate on the inference component. Inference is the problem of determining the *best* global output $\hat{\mathbf{y}} \in \mathcal{F}(\mathbf{Y})$ given model parameters $\boldsymbol{\lambda}$, according to some cost function f, where $\mathbf{Y}$ is the output space and $\mathcal{F}(\mathbf{Y}) \subseteq \mathbf{Y}$ is the subset of $\mathbf{Y}$ that satisfy some constraints. Formally, if $\mathbf{x}$ represents the input data, then $\hat{\mathbf{y}}$ is decided as follows:

$$\hat{\mathbf{y}} = \text{argmax}_{\mathbf{y}\in\mathcal{F}(\mathbf{Y})} f(\mathbf{x}, \mathbf{y}; \boldsymbol{\lambda}).$$

The efficiency and tractability of the inference procedure dictate the feasibility of the whole framework. However, whether there exists an efficient and exact inference algorithm highly depends on the problem's structure. Polynomial-time algorithms usually do not exist when there are complex constraints among the output variables (just like the entity/relation problem described in this chapter). In this section, we briefly introduce several common inference algorithms in various text-processing problems, and contrast them with our ILP approach.

20.6.1 Exact Polynomial-time Methods

Most polynomial-time inference methods are based on dynamic programming. For linear chain structures, the Viterbi algorithm and its variations are the most popular. For tree structures, different cubic-time algorithms have been proposed. Although replacing these algorithms with the ILP approach does not necessarily make the inference more efficient in practice, as we show below, the ILP framework does provide these polynomial-time algorithms an easy way to incorporate additional "declarative" constraints, which may not be possible to express within the original inference algorithm. We describe these methods here and sketch how they can be formulated as an integer linear programming problem.

20.6.1.1 The Viterbi Algorithm

Linear-chain structures are often used for sequence labeling problems, where the task is to decide the label of each token. For this problem, HMMs [27], conditional sequential models and other extensions [25], and conditional random fields [21] are commonly used. While the first two methods learn the state transition between a pair of consecutive tokens, conditional random fields relax the directionality assumption and train the potential functions for the size-1 (i.e., a single token) and size-2 (a pair of consecutive tokens) cliques. In both cases, the Viterbi algorithm is usually used to find the most probable sequence assignment.

We describe the Viterbi algorithm in the linear-chain conditional random fields setting as follows. Suppose we need to predict the labels of a sequence of tokens, $t_0, t_1, \cdots, t_{m-1}$. Let $\mathcal{Y}$ be the set of possible labels for each token, where $|\mathcal{Y}| = m$. A set of $m \times m$ matrices $\{M_i(\mathbf{x})|i = 0, \ldots, n-1\}$ is defined over each pair of labels $y', y \in \mathcal{Y}$

$$M_i(y', y|\mathbf{x}) = \exp(\sum_j \lambda_j f_j(y', y, \mathbf{x}, i)),$$

where λ_j are the model parameters and f_j are the features. By augmenting two special nodes y_{-1} and y_n before and after the sequence with labels `start` and `end` respectively, the sequence probability is

$$p(\mathbf{y}|\mathbf{x}, \boldsymbol{\lambda}) = \frac{1}{Z(\mathbf{x})} \prod_{i=0}^{n} M_i(y_{i-1}, y_i|\mathbf{x}).$$

$Z(\mathbf{x})$ is a normalization factor that can be computed from the M_i's but is not needed in evaluation. We only need to find the label sequence $\mathbf{y}$ that maximizes the product of the corresponding elements of these $n+1$ matrices. The Viterbi algorithm is the standard method that computes the most likely label sequence given the observation. It *grows* the optimal label sequence incrementally by scanning the matrices from position 0 to n. At step i, it records all the optimal sequences ending at a label $y, \forall y \in \mathcal{Y}$ (denoted by $\mathbf{y}_i^*(y)$), and also the corresponding product $P_i(y)$. The recursive function of this dynamic programming algorithm is

1. $P_0(y) = M_0(\mathsf{start}, y|\mathbf{x})$ and $\mathbf{y}_0^*(y) = y$.
2. for $1 \leq i \leq n$, $\mathbf{y}_i^*(y) = \mathbf{y}_{i-1}^*(\hat{y}).(y)$ and $P_i(y) = \max_{y' \in \mathcal{Y}} P_{i-1}(y')M(y', y|\mathbf{x})$, where $\hat{y} = \text{argmax}_{y' \in \mathcal{Y}} P_{i-1}(y')M(y', y|\mathbf{x})$ and "." is the concatenation operator.

The optimal sequence is therefore $\mathbf{y}_{n-1}^* = [\mathbf{y}_n^*]_{0..n-1}$, which is the best path to the end symbol but taking only position 0 to position $n-1$.

The solution that Viterbi outputs is in fact the shortest path in the graph constructed is as follows. Let n be the number of tokens in the sequence, and m be the number of labels each token can take. The graph consists of $nm + 2$ nodes and $(n-1)m^2 + 2m$ edges. In addition to two special nodes start and end that denote the start and end positions of the path, the label of each token is represented by a node v_{ij}, where $0 \leq i \leq n-1$, and $0 \leq j \leq m-1$. If the path passes node v_{ij}, then label j is assigned to token i. For nodes that represent two adjacent tokens $v_{(i-1)j}$ and $v_{ij'}$, where $0 \leq i \leq n$, and $0 \leq j, j' \leq m-1$, there is a directed edge $x_{i,jj'}$ from $v_{(i-1)j}$ to $v_{ij'}$, with the cost $-\log(M_i(jj'|\mathbf{x}))$.

Obviously, the path from start to end will pass exactly one node on position i. That is, exactly one of the nodes $v_{i,j}, 0 \leq j \leq m-1$, will be picked. Figure 20.3 illustrates the graph. Suppose that $\mathbf{y} = y_0 y_1 \cdots y_{n-1}$ is the label sequence determined by the path. Then

$$\text{argmin}_{\mathbf{y}} - \sum_{i=0}^{n-1} \log(M_i(y_{i-1}y_i|\mathbf{x})) = \text{argmax}_{\mathbf{y}} \prod_{i=0}^{n-1} M_i(y_{i-1}y_i|\mathbf{x}).$$

Namely, the nodes in the shortest path are exactly the labels returned by the Viterbi algorithm.

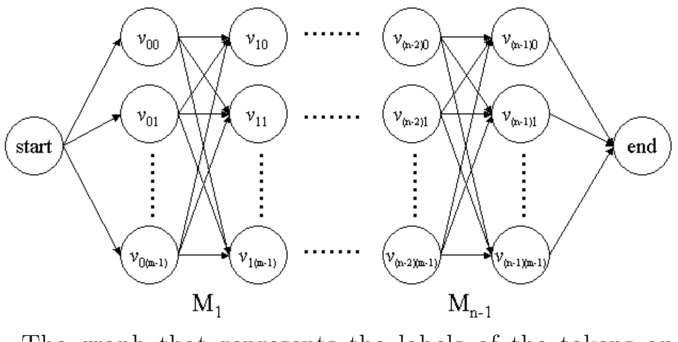

Figure 20.3 The graph that represents the labels of the tokens and the state transition (also known as the *trellis* in hidden Markov models).

The Viterbi algorithm can still be used when the matrix is slightly modified to incorporate simple constraints. For example, in the task of information extraction, if the label of a word is the *beginning* of an entity (`B`), *inside* an entity (`I`), or *outside* any entity (`O`), a token label `O` immediately followed by a label `I` is not a valid labeling. The constraint can be incorporated by changing the corresponding transitional probability or matrix entries to 0 [10, 20]. However, more general, nonMarkovian constraints cannot be resolved using the same trick.

Recently, Roth and Yih [32] proposed a different inference approach based on ILP to replace the Viterbi algorithm. The basic idea there is to use integer linear programming to find the shortest path in the trellis (e.g., figure 20.3). Each edge of the graph is represented by an indicator variable to represent whether this edge is in the shortest path or not. The cost function can be written in terms of a linear function of these indicator variables. In this ILP, linear (in)equalities are added to enforce that the values of these indicator variables represent a legitimate path. This ILP can be solved simply by LP relaxation because the coefficient matrix is totally unimodular. However, the main advantage of this new setting is its ability to allow more general constraints that can be encoded either in linear (in)equalities or in the cost function. Interested readers may see [32] for more details.

20.6.1.2 Constraint Satisfaction with Classifiers

A second efficient inference algorithm for linear sequence tasks that has been used successfully for natural language and information extraction problems is *constraint satisfaction with classifiers* (CSCL) [25]. This method was first proposed for shallow parsing – identifying atomic phrases (e.g., base noun phrases) in a given sentence. In that case, two classifiers are first trained to predict whether a word ***opens*** (`O`) a phrase or ***closes*** (`C`) a phrase. Since these two classifiers may generate inconsistent predictions, the inference task has to decide which `OC` pairs are indeed the boundaries of a phrase.

We illustrate their approach by the following example. Suppose a sentence has six tokens, $t_1, \cdots, t_6$, as indicated in figure 20.4. The classifiers have identified three opens (O) and three closes (C) in this sentence (i.e., the open and close brackets). Among the OC pairs (t_1, t_3), (t_1, t_5), (t_1, t_6), (t_2, t_3), (t_2, t_5), (t_2, t_6), (t_4, t_5), (t_4, t_6), the inference procedure needs to decide which of them are the predicted phrases, based on the cost function. In addition, the chosen phrases should not overlap or embed with each other. Let the predicate "this pair is selected as a phrase" be represented by an indicator variable $x_i \in X$, where $|X| = 8$ in this case. They associate a cost function $c : X \rightarrow R$ with each variable (where the value $c(x_i)$ is determined as a function of the corresponding OC classifiers), and try to find a solution that minimizes the overall cost, $\sum_{i=1}^{n} c(x_i)x_i$.

This problem can be reduced elegantly to a shortest path problem by the following graph construction. Each open and close word is represented by an `O` node and a `C` node. For each possible `OC` pair, there is a direct link from the corresponding open

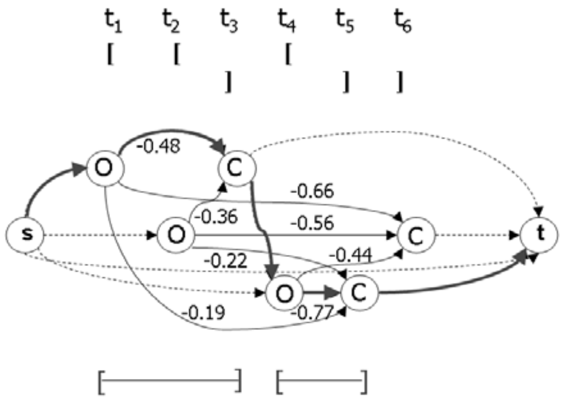

Figure 20.4 Identifying phrases in a sentence using the constraints satisfaction with classifiers (CSCL) approach, courtesy of Vasin Punyakanok.

node `O` to the close node `C`. Finally, one source (`s`) node and one target (`t`) node are added. Links are added from `s` to each `O` and from each `C` to `t`. The cost of an `OC` link is $-p_i$, where p_i is the probability that this `OC` pair represents a phrase, estimated by the `O` and `C` classifiers.

Because the inference process is also done by finding the shortest path in the graph, the ILP framework described in [32] is applicable here as well.

20.6.1.3 Clause Identification

The two efficient approaches mentioned above can be generalized beyond the sequential structure, to tree structures. Cubic-time dynamic algorithms are often used for inference in various tree-structure problems, such as parsing [17] or clause identification [38]. As an example, we discuss the inference approach proposed by Carreras et al. [7], in the context of clause identification. Clause identification is a partial parsing problem. Given a sentence, a clause is defined as a sequence of words that contains a subject and a predicate [38]. In the following example sentence taken from the Penn yreebank [23], each pair of corresponding parentheses represents a clause. The task is thus to identify all the clauses in the sentence.

(The deregulation of railroads and trucking companies (that (began in 1980)) enabled (shippers to bargain for transportation).)

Although the problem looks similar to shallow parsing, the constraints between the clauses are weaker – clauses may not overlap, but a clause can be embedded in another. Formally speaking, let w_i be the ith word in a sentence of n words. A clause can be defined as a pair of numbers (s, t), where $1 \leq s \leq t \leq n$, which represents the word sequence $w_s, w_{s+1}, \ldots, w_t$. Given two clauses $c_1 = (s_1, t_1)$ and $c_2 = (s_2, t_2)$, we say that these two clauses overlap iff $s_1 < s_2 \leq t_1 < t_2$ or $s_2 < s_1 \leq t_2 < t_1$.

Similarly to the approach presented throughout this chapter, in [7, 6], this problem is solved by combining learning and inference. Briefly speaking, each candidate clause $c = (s, t)$ in the targeted sentence is associated with a score, *score*(c), estimated by the classifiers. Let $\mathbf{C}$ be the set of all possible clauses in the given sentence, $\mathcal{F}(\mathbf{C})$ all possible subsets of $\mathbf{C}$ that satisfy the nonoverlapping constraint. Then the *best* clause prediction is defined as

$$\mathbf{c}^* = \text{argmax}_{\mathbf{c} \in \mathcal{F}(\mathbf{C})} \sum_{c \in \mathbf{c}} score(c).$$

Carreras et al. [7] proposed a dynamic programming algorithm to solve this inference problem. In this algorithm, two 2D matrices are maintained: `best-split[s,t]` stores the optimal clause predictions in $w_s, w_{s+1}, \ldots, w_t$; `score[s,t]` is the score of the clause (s, t). By filling the table recursively, the optimal clause prediction can be found in $O(n^3)$ time.

As in the previous cases discussed in this section, it is clear that this problem can be represented as an ILP. Each candidate clause (s, t) can be represented by an indicator variable $x_{s,t}$. The cost function is the sum of the score times the corresponding indicator variable, namely $\sum(score(s, t) \cdot x_{s,t})$. Suppose clause candidates (s_1, t_1) and (s_2, t_2) overlap. The nonoverlapping constraint can be enforced by adding a linear inequality, $x_{s_1,t_1} + x_{s_2,t_2} \leq 1$.

20.6.2 Generic Methods – Search

As discussed above, exact polynomial time algorithms exist for specific constraint structures; however, the inference problem typically becomes computationally intractable when additional constraints are introduced, or more complex structures are needed. A common computational approach to the inference problem in this case is *search*. Following the definition in [33], search is used to find a legitimate state transition path from the initial state to a goal state while trying to minimize the cost. The problem can be treated as consisting of four components: state space, operators (the legitimate state transitions), goal-test (a function that examines whether a goal state is reached), and path-cost-function (the cost function of the whole path). Figure 20.5 depicts a generic search algorithm.

To solve the entity-relation problem described in this chapter, we can define the state space as the set of all possible labels of the entities and relations (namely, $\mathcal{L}_\mathcal{E}$ and $\mathcal{L}_\mathcal{R}$), plus "*undecided.*" In the initial state, the values of all the variables are "*undecided.*" A legitimate operator changes an entity or relation variable from "*undecided*" to one of the possible labels, subject to the constraints. The goal-test evaluates whether every variable has been assigned a label, and the path-cost is the sum of the assignment cost of each variable.

The main advantage of inference using search is its generality. The cost function need not be linear. The constraints can also be fairly general: as long as the decision

Algorithm 1

```
generic-search(problem, enqueue-func)

  nodes ← MakeQueue(MakeNode(init-state(problem))
  while (node is not empty)
      node ← RemoveFront(nodes)
      if (goal-test(node)) then return node
      next ← Operators(node)
      nodes ← enqueue-func(problem, nodes, next)
  end
  return failure

end
```

Figure 20.5 The generic search algorithm, adapted from [33].

on whether a state violates constraints can be evaluated efficiently, they can be used to define the operators.

The main disadvantage, however, is that there is no guarantee of optimality. Despite this weakness, it has been shown that *search* is a successful approach in some tasks empirically. For instance, Moore [24] applied beam search to find the best word alignment given a linear model learned using voted perceptron. Recently, Daumé and Marcu [14] demonstrated an approximate large margin method for learning structured output, where the key inference component is search.

In contrast, our ILP approach may or may not be able to replace this search mechanism, depending on the specific cost function. Nevertheless, in several real-world problems, we observed that our ILP method may not be slower than search, but is guaranteed to find the optimal solution.

20.7 Conclusion

We presented a linear-programming based approach for global inference in cases where decisions depend on the outcomes of several different but mutually dependent classifiers. Even in the presence of a fairly general constraint structure, deviating from the sequential nature typically studied, this approach can find the optimal solution efficiently.

Contrary to general search schemes (e.g., beam search), which do not guarantee optimality, the LP approach provides an efficient way of finding the optimal solution. The key advantage of the LP formulation is its generality and flexibility; in particular, it supports the ability to incorporate classifiers learned in other contexts, "hints" supplied, and decision-time constraints, and reason with all these

for the best global prediction. In sharp contrast with the typically used pipeline framework, our formulation does not blindly trust the results of some classifiers, and therefore is able to overcome mistakes made by classifiers with the help of constraints.

Our experiments have demonstrated these advantages by considering the interaction between entity and relation classifiers. In fact, more classifiers can be added and used within the same framework. For example, if coreference resolution is available, it is possible to incorporate it in the form of constraints that force the labels of the coreferred entities to be the same (but, of course, allowing the global solution to reject the suggestion of these classifiers). Consequently, this may enhance the performance of entity-relation recognition and, at the same time, correct possible coreference resolution errors. Another example is to use chunking information for better relation identification; suppose, for example, that we have available chunking information that identifies Subj+Verb and Verb+Object phrases. Given a sentence that has the verb "murder," we may conclude that the subject and object of this verb are in a "kill" relation. Since the chunking information is used in the global inference procedure, this information will contribute to enhancing its performance and robustness, relying on having more constraints and overcoming possible mistakes by some of the classifiers. Moreover, in an interactive environment where a user can supply new constraints (e.g., a question-answering situation) this framework is able to make use of the new information and enhance the performance at decision time, without retraining the classifiers. As we have shown, our formulation supports not only improved accuracy but also improves the "coherent" quality of the decisions. We believe that it has the potential to be a powerful way for supporting natural language inference.

Acknowledgments

Most of this research was done when Wen-tau Yih was at the University of Illinois at Urbana-Champaign. This research was supported by NSF grants CAREER IIS-9984168 and ITR IIS-0085836, an ONR MURI award and by the Advanced Research and Development Activity (ARDA)'s Advanced Question Answering for Intelligence (AQUAINT) program.

References

[1] S. Abney. Parsing by chunks. In R. Berwick, S. Abney, and C. Tenny, editors, *Principle-Based Parsing: Computation and Psycholinguistics*, pages 257–278. Kluwer, Dordrecht, Netherlands, 1991.

[2] C. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Oxford, UK, 1995.

[3] Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1222–1239, 2001.

[4] A. Carlson, C. Cumby, J. Rosen, and D. Roth. The SNoW learning architecture. Technical Report UIUCDCS-R-99-2101, University of Illinois at Urbana-Champaign Computer Science Department, May 1999.

[5] X. Carreras and L. Màrquez. Introduction to the CoNLL-2004 shared tasks: Semantic role labeling. In *Proceedings of the Conference on Natural Language Learning*, 2004.

[6] X. Carreras and L. Màrquez. Introduction to the CoNLL-2005 shared task: Semantic role labeling. In *Proceedings of the Conference on Natural Language Learning*, 2005.

[7] X. Carreras, L. Màrquez, V. Punyakanok, and D. Roth. Learning and inference for clause identification. In *Proceedings of the European Conference on Machine Learning*, 2002.

[8] C. Chekuri, S. Khanna, J. Naor, and L. Zosin. Approximation algorithms for the metric labeling problem via a new linear programming formulation. In *Symposium on Discrete Algorithms*, 2001.

[9] R. Chellappa and A. Jain. *Markov Random Fields: Theory and Application.* Academic Press, April 1993.

[10] H. Chieu and H. Ng. A maximum entropy approach to information extraction from semi-structure and free text. In *Proceedings of the National Conference on Artificial Intelligence*, 2002.

[11] CPLEX. ILOG, Inc. http://www.ilog.com/products/cplex/, 2003.

[12] C. Cumby and D. Roth. Relational representations that facilitate learning. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, 2000.

[13] C. Cumby and D. Roth. On kernel methods for relational learning. In *Proceedings of the International Conference on Machine Learning*, 2003.

[14] H. Daumé III and D. Marcu. Learning as search optimization: Approximate large margin methods for structured prediction. In *Proceedings of the International Conference on Machine Learning*, 2005.

[15] T. Dietterich. Machine learning for sequential data: A review. In *Structural, Syntactic, and Statistical Pattern Recognition*, pages 15–30. Springer-Verlag, 2002.

[16] Y. Even-Zohar and D. Roth. A classification approach to word prediction. In *Proceedings of the Annual Meeting of the North American Association of Computational Linguistics*, 2000.

[17] M. Johnson. PCFG models of linguistic tree representations. *Computational Linguistics*, 24(4):613–632, 1998.

[18] R. Khardon, D. Roth, and L. G. Valiant. Relational learning for NLP using linear threshold elements. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 1999.

[19] J. Kleinberg and E. Tardos. Approximation algorithms for classification problems with pairwise relationships: Metric labeling and markov random fields. In *IEEE Symposium on Foundations of Computer Science*, 1999.

[20] T. Kristjannson, A. Culotta, P. Viola, and A. McCallum. Interactive information extraction with constrained conditional random fields. In *Proceedings of the National Conference on Artificial Intelligence*, 2004.

[21] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the International Conference on Machine Learning*, 2001.

[22] A. Levin, A. Zomet, and Y. Weiss. Learning to perceive transparency from the statistics of natural scenes. In *Proceedings of Neural Information Processing Systems*, 2002.

[23] M. P. Marcus, M. A. Marcinkiewicz, and B. Santorini. Building a large annotated corpus of English: the Penn treebank. *Computational Linguistics*, 19(2):313–330, 1993.

[24] R. C. Moore. A discriminative framework for bilingual word alignment. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2005.

[25] V. Punyakanok and D. Roth. The use of classifiers in sequential inference. In *Proceedings of Neural Information Processing Systems*, 2001.

[26] V. Punyakanok, D. Roth, W. Yih, and D. Zimak. Learning and inference over constrained output. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2005.

[27] L.R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2), February 1989.

[28] D. Roth. Reasoning with classifiers. In *Proceedings of the European Conference on Machine Learning*, 2002.

[29] D. Roth. Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of the National Conference on Artificial Intelligence*, 1998.

[30] D. Roth and W. Yih. Relational learning via propositional algorithms: An information extraction case study. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2001.

[31] D. Roth and W. Yih. Probabilistic reasoning for entity and relation recognition. In *Proceedings of the International Conference on Computational Linguistics*, 2002.

[32] D. Roth and W. Yih. Integer linear programming inference for conditional random fields. In *Proceedings of the International Conference on Machine Learning*, 2005.

[33] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ, 1995.

[34] A. Schrijver. *Theory of Linear and Integer Programming.* Wiley Interscience Series in Discrete Mathmatics. John Wiley & Sons, Hoboken, NJ, 1986.

[35] B. Taskar, P. Abbeel, and D. Koller. Discriminative probabilistic models for relational data. In *Proceedings of Uncertainty in Artificial Intelligence*, 2002.

[36] B. Taskar, D. Klein, M. Collins, D. Koller, and C. Manning. Max-margin parsing. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, 2004.

[37] E. F. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Conference on Natural Language Learning*, 2003.

[38] E. F. Tjong Kim Sang and H. Déjean. Introduction to the CoNLL-2001 shared task: Clause identification. In Walter Daelemans and Rémi Zajac, editors, *Proceedings of the Conference on Natural Language Learning*, pages 53–57, 2001.

[39] I. Tsochantaridis, T. Hofmann, T. Joachims, and Y. Altun. Support vector machine learning for interdependent and structured output spaces. In *Proceedings of the International Conference on Machine Learning*, 2004.

[40] X. Wang and A. Regan. A cutting plane method for integer programming problems with binary variables. Technical Report UCI-ITS-WP-00-12, University of California, Irvine, 2000.

[41] L. Wolsey. *Integer Programming.* John Wiley & Sons, Hoboken, NJ, 1998.

[42] Xpress-MP. Dash Optimization. http://www.dashoptimization.com/products.html, 2003.