Global Learning of Typed Entailment Rules

Jonathan Berant
Tel Aviv University
Tel Aviv, Israel
jonatha6@post.tau.ac.il

Ido Dagan
Bar-Ilan University
Ramat-Gan, Israel
dagan@cs.biu.ac.il

Jacob Goldberger
Bar-Ilan University
Ramat-Gan, Israel
goldbej@eng.biu.ac.il

Abstract

Extensive knowledge bases of entailment rules between predicates are crucial for applied semantic inference. In this paper we propose an algorithm that utilizes transitivity constraints to learn a globally-optimal set of entailment rules for typed predicates. We model the task as a graph learning problem and suggest methods that scale the algorithm to larger graphs. We apply the algorithm over a large data set of extracted predicate instances, from which a resource of typed entailment rules has been recently released (Schoenmackers et al., 2010). Our results show that using global transitivity information substantially improves performance over this resource and several baselines, and that our scaling methods allow us to increase the scope of global learning of entailment-rule graphs.

1 Introduction

Generic approaches for applied semantic inference from text gained growing attention in recent years, particularly under the Textual Entailment (TE) framework (Dagan et al., 2009). TE is a generic paradigm for semantic inference, where the objective is to recognize whether a target meaning can be inferred from a given text. A crucial component of inference systems is extensive resources of *entailment rules*, also known as *inference rules*, i.e., rules that specify a directional inference relation between fragments of text. One important type of rule is rules that specify entailment relations between predicates and their arguments. For example, the rule *'X annex Y → X control Y'* helps recognize that the text *'Japan annexed Okinawa'* answers the question *'Which country controls Okinawa?'*. Thus, acquisition of such knowledge received considerable attention in the last decade (Lin and Pantel, 2001; Sekine, 2005; Szpektor and Dagan, 2009; Schoenmackers et al., 2010).

Most past work took a "local learning" approach, learning each entailment rule independently of others. It is clear though, that there are global interactions between predicates. Notably, entailment is a *transitive* relation and so the rules $A \rightarrow B$ and $B \rightarrow C$ imply $A \rightarrow C$.

Recently, Berant et al. (2010) proposed a *global* graph optimization procedure that uses *Integer Linear Programming (ILP)* to find the best set of entailment rules under a transitivity constraint. Imposing this constraint raised two challenges. The first of *ambiguity:* transitivity does not always hold when predicates are ambiguous, e.g., *X buy Y → X acquire Y* and *X acquire Y → X learn Y*, but *X buy Y ↛ X learn Y* since these two rules correspond to two different senses of *acquire*. The second challenge is *scalability:* ILP solvers do not scale well since ILP is an NP-complete problem. Berant et al. circumvented these issues by learning rules where one of the predicate's arguments is instantiated (e.g., *'X reduce nausea → X affect nausea'*), which is useful for learning small graphs on-the-fly, given a target concept such as *nausea*. While rules may be effectively learned when needed, their scope is narrow and they are not useful as a generic knowledge resource.

This paper aims to take global rule learning one step further. To this end, we adopt the representation suggested by Schoenmackers et al. (2010), who learned inference rules between *typed predicates*, i.e., predicates where the argument types (e.g., *city* or *drug*) are specified. Schoenmackers et al. uti-

lized typed predicates since they were dealing with noisy and ambiguous web text. Typing predicates helps disambiguation and filtering of noise, while still maintaining rules of wide-applicability. Their method employs a local learning approach, while the number of predicates in their data is too large to be handled directly by an ILP solver.

In this paper we suggest applying global optimization learning to open domain typed entailment rules. To that end, we show how to construct a structure termed *typed entailment graph*, where the nodes are typed predicates and the edges represent entailment rules. We suggest scaling techniques that allow to optimally learn such graphs over a large set of typed predicates by first *decomposing* nodes into components and then applying *incremental ILP* (Riedel and Clarke, 2006). Using these techniques, the obtained algorithm is guaranteed to return an optimal solution. We ran our algorithm over the data set of Schoenmackers et al. and release a resource of 30,000 rules[1] that achieves substantially higher recall without harming precision. To the best of our knowledge, this is the first resource of that scale to use global optimization for learning predicative entailment rules. Our evaluation shows that global transitivity improves the F_1 score of rule learning by 27% over several baselines and that our scaling techniques allow dealing with larger graphs, resulting in improved coverage.

2 Background

Most work on learning entailment rules between predicates considered each rule independently of others, using two sources of information: *lexicographic resources* and *distributional similarity*.

Lexicographic resources are manually-prepared knowledge bases containing semantic information on predicates. A widely-used resource is WordNet (Fellbaum, 1998), where relations such as *synonymy* and *hyponymy* can be used to generate rules. Other resources include NomLex (Macleod et al., 1998; Szpektor and Dagan, 2009) and FrameNet (Baker and Lowe, 1998; Ben Aharon et al., 2010).

Lexicographic resources are accurate but have low coverage. Distributional similarity algorithms use large corpora to learn broader resources by assuming that semantically similar predicates appear with similar arguments. These algorithms usually represent a predicate with one or more vectors and use some function to compute argument similarity. Distributional similarity algorithms differ in their *feature representation*: Some use a *binary* representation: each predicate is represented by one feature vector where each feature is a pair of arguments (Szpektor et al., 2004; Yates and Etzioni, 2009). This representation performs well, but suffers when data is sparse. The *binary-DIRT* representation deals with sparsity by representing a predicate with a pair of vectors, one for each argument (Lin and Pantel, 2001). Last, a richer form of representation, termed *unary*, has been suggested where a different predicate is defined for each argument (Szpektor and Dagan, 2008). Different algorithms also differ in their similarity function. Some employ symmetric functions, geared towards *paraphrasing* (bi-directional entailment), while others choose directional measures more suited for entailment (Bhagat et al., 2007). In this paper, We employ several such functions, such as *Lin* (Lin and Pantel, 2001), and *BInc* (Szpektor and Dagan, 2008).

Schoenmackers et al. (2010) recently used distributional similarity to learn rules between *typed predicates*, where the left-hand-side of the rule may contain more than a single predicate (horn clauses). In their work, they used Hearst-patterns (Hearst, 1992) to extract a set of 29 million *(argument, type)* pairs from a large web crawl. Then, they employed several filtering methods to clean this set and automatically produced a mapping of 1.1 million arguments into 156 types. Examples for *(argument, type)* pairs are *(EXODUS, book)*, *(CHINA, country)* and *(ASTHMA, disease)*. Schoenmackers et al. then utilized the types, the mapped arguments and tuples from TextRunner (Banko et al., 2007) to generate 10,672 typed predicates (such as *conquer(country,city)* and *common in(disease,place)*), and learn 30,000 rules between these predicates[2]. In this paper we will learn entailment rules over the same data set, which was generously provided by

[1] The resource can be downloaded from http://www.cs.tau.ac.il/~jonatha6/homepage_files/resources/ACL2011Resource.zip

[2] The rules and the mapping of arguments into types can be downloaded from http://www.cs.washington.edu/research/sherlock-hornclauses/

Schoenmackers et al.

As mentioned above, Berant et al. (2010) used global transitivity information to learn small *entailment graphs*. Transitivity was also used as an information source in other fields of NLP: Taxonomy Induction (Snow et al., 2006), Co-reference Resolution (Finkel and Manning, 2008), Temporal Information Extraction (Ling and Weld, 2010), and Unsupervised Ontology Induction (Poon and Domingos, 2010). Our proposed algorithm applies to any sparse transitive relation, and so might be applicable in these fields as well.

Last, we formulate our optimization problem as an Integer Linear Program (ILP). ILP is an optimization problem where a linear objective function over a set of integer variables is maximized under a set of linear constraints. Scaling ILP is challenging since it is an NP-complete problem. ILP has been extensively used in NLP lately (Clarke and Lapata, 2008; Martins et al., 2009; Do and Roth, 2010).

3 Typed Entailment Graphs

Given a set of typed predicates, entailment rules can only exist between predicates that share the same (unordered) pair of types (such as *place* and *country*)[3]. Hence, every pair of types defines a graph that describes the entailment relations between predicates sharing those types (Figure 1). Next, we show how to represent entailment rules between typed predicates in a structure termed *typed entailment graph*, which will be the learning goal of our algorithm.

A typed entailment graph is a directed graph where the nodes are *typed predicates*. A typed predicate is a triple $p(t_1, t_2)$ representing a predicate in natural language. p is the lexical realization of the predicate and the types t_1, t_2 are variables representing argument types. These are taken from a set of types T, where each type $t \in T$ is a bag of natural language words or phrases. Examples for typed predicates are: *conquer(country,city)* and *contain(product,material)*. An *instance* of a typed predicate is a triple $p(a_1, a_2)$, where $a_1 \in t_1$ and $a_2 \in t_2$ are termed *arguments*. For example, *be common in(ASTHMA,AUSTRALIA)* is an instance of *be common in(disease,place)*. For brevity, we refer to typed entailment graphs and typed predicates as *entailment graphs* and *predicates* respectively.

Edges in typed entailment graphs represent entailment rules: an edge (u, v) means that predicate u entails predicate v. If the type t_1 is different from the type t_2, mapping of arguments is straightforward, as in the rule '*be find in(material,product)* $\rightarrow$ *contain(product,material)*'. We term this *a two-types entailment graph*. When t_1 and t_2 are equal, mapping of arguments is ambiguous: we distinguish *direct-mapping edges* where the first argument on the left-hand-side (LHS) is mapped to the first argument on the right-hand-side (RHS), as in '*beat(team,team)* $\xrightarrow{d}$ *defeat(team,team)*', and *reversed-mapping edges* where the LHS first argument is mapped to the RHS second argument, as in '*beat(team,team)* $\xrightarrow{r}$ *lose to(team,team)*'. We term this *a single-type entailment graph*. Note that in single-type entailment graphs reversed-mapping loops are possible as in '*play(team,team)* $\xrightarrow{r}$ *play(team,team)*': if team A plays team B, then team B plays team A.

Since entailment is a transitive relation, typed-entailment graphs are transitive: if the edges (u, v) and (v, w) are in the graph so is the edge (u, w). Note that in single-type entailment graphs one needs to consider whether mapping of edges is direct or reversed: if mapping of both (u, v) and (v, w) is either direct or reversed, mapping of (u, w) is direct, otherwise it is reversed.

Typing plays an important role in rule transitivity: if predicates are ambiguous, transitivity does not necessarily hold. However, typing predicates helps disambiguate them and so the problem of ambiguity is greatly reduced.

4 Learning Typed Entailment Graphs

Our learning algorithm is composed of two steps: (1) Given a set of typed predicates and their instances extracted from a corpus, we train a (local) *entailment classifier* that estimates for every pair of predicates whether one entails the other. (2) Using the classifier scores we perform global optimization, i.e., learn the set of edges over the nodes that maximizes the global score of the graph under transitivity and background-knowledge constraints.

Section 4.1 describes the local classifier training

[3] Otherwise, the rule would contain unbound variables.

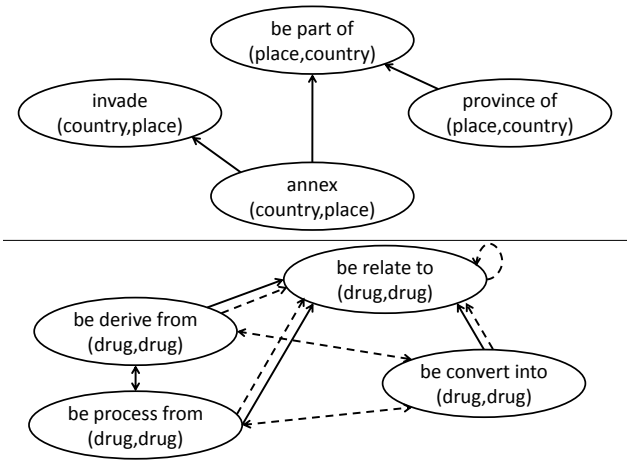

Figure 1: *Top:* A fragment of a two-types entailment graph. *bottom:* A fragment of a single-type entailment graph. Mapping of solid edges is direct and of dashed edges is reversed.

procedure. Section 4.2 gives an ILP formulation for the optimization problem. Sections 4.3 and 4.4 propose scaling techniques that exploit graph sparsity to optimally solve larger graphs.

4.1 Training an entailment classifier

Similar to the work of Berant et al. (2010), we use "distant supervision". Given a lexicographic resource (WordNet) and a set of predicates with their instances, we perform the following three steps (see Table 1):

1) Training set generation We use WordNet to generate positive and negative examples, where each example is a pair of predicates. Let P be the set of input typed predicates. For every predicate $p(t_1, t_2) \in P$ such that p is a single word, we extract from WordNet the set S of synonyms and direct hypernyms of p. For every $p' \in S$, if $p'(t_1, t_2) \in P$ then $p(t_1, t_2) \rightarrow p'(t_1, t_2)$ is taken as a positive example.

Negative examples are generated in a similar manner, with direct co-hyponyms of p (sister nodes in WordNet) and hyponyms at distance 2 instead of synonyms and direct hypernyms. We also generate negative examples by randomly sampling pairs of typed predicates that share the same types.

2) Feature representation Each example pair of predicates (p_1, p_2) is represented by a feature vector, where each feature is a specific distributional similarity score estimating whether p_1 entails p_2. We compute 11 distributional similarity scores for each pair of predicates based on the arguments appearing in the extracted arguments. The first 6 scores are computed by trying all combinations of the similarity functions *Lin* and *BInc* with the feature representations *unary*, *binary-DIRT* and *binary* (see Section 2). The other 5 scores were provided by Schoenmackers et al. (2010) and include *SR* (Schoenmackers et al., 2010), *LIME* (McCreath and Sharma, 1997), *M-estimate* (Dzeroski and Brakto, 1992), the standard *G-test* and a simple implementation of *Cover* (Weeds and Weir, 2003). Overall, the rationale behind this representation is that combining various scores will yield a better classifier than each single measure.

Type	**example**
hyper.	*beat(team,team)* $\rightarrow$ *play(team,team)*
syno.	*reach(team,game)* $\rightarrow$ *arrive at(team,game)*
cohypo.	*invade(country,city)* $\nrightarrow$ *bomb(country,city)*
hypo.	*defeat(city,city)* $\nrightarrow$ *eliminate(city,city)*
random	*hold(place,event)* $\nrightarrow$ *win(place,event)*

Table 1: Automatically generated training set examples.

3) Training We train over an equal number of positive and negative examples, as classifiers tend to perform poorly on the minority class when trained on imbalanced data (Van Hulse et al., 2007; Nikulin, 2008).

4.2 ILP formulation

Once the classifier is trained, we would like to learn all edges (entailment rules) of each typed entailment graph. Given a set of predicates V and an entailment score function $f : V \times V \rightarrow \mathbb{R}$ derived from the classifier, we want to find a graph $G = (V, E)$ that respects transitivity and maximizes the sum of edge weights $\sum_{(u,v)\in E} f(u, v)$. This problem is NP-hard by a reduction from the NP-hard *Transitive Subgraph* problem (Yannakakis, 1978). Thus, employing ILP is an appealing approach for obtaining an optimal solution.

For two-types entailment graphs the formulation is simple: The ILP variables are indicators X_{uv} denoting whether an edge (u, v) is in the graph, with the following ILP:

$$\hat{G} = arg\,max \sum_{u \neq v} f(u,v) \cdot X_{uv} \quad (1)$$

$$\text{s.t. } \forall_{u,v,w \in V}\ X_{uv} + X_{vw} - X_{uw} \leq 1 \quad (2)$$

$$\forall_{u,v \in A_{yes}}\ X_{uv} = 1 \quad (3)$$

$$\forall_{u,v \in A_{no}}\ X_{uv} = 0 \quad (4)$$

$$\forall_{u \neq v}\ X_{uv} \in \{0,1\} \quad (5)$$

The objective in Eq. 1 is a sum over the weights of the eventual edges. The constraint in Eq. 2 states that edges must respect transitivity. The constraints in Eq. 3 and 4 state that for known node pairs, defined by A_{yes} and A_{no}, we have background knowledge indicating whether entailment holds or not. We elaborate on how A_{yes} and A_{no} were constructed in Section 5. For a graph with n nodes we get $n(n-1)$ variables and $n(n-1)(n-2)$ transitivity constraints.

The simplest way to expand this formulation for single-type graphs is to duplicate each predicate node, with one node for each order of the types, and then the ILP is unchanged. However, this is inefficient as it results in an ILP with $2n(2n-1)$ variables and $2n(2n-1)(2n-2)$ transitivity constraints. Since our main goal is to scale the use of ILP, we modify it a little. We denote a direct-mapping edge (u,v) by the indicator X_{uv} and a reversed-mapping edge (u,v) by Y_{uv}. The functions f_d and f_r provide scores for direct and reversed mappings respectively. The objective in Eq. 1 and the constraint in Eq. 2 are replaced by (Eq. 3, 4 and 5 still exist and are carried over in a trivial manner):

$$arg\,max \sum_{u \neq v} f_d(u,v) X_{uv} + \sum_{u,v} f_r(u,v) Y_{uv} \quad (6)$$

$$\text{s.t. } \forall_{u,v,w \in V}\ X_{uv} + X_{vw} - X_{uw} \leq 1$$

$$\forall_{u,v,w \in V}\ X_{uv} + Y_{vw} - Y_{uw} \leq 1$$

$$\forall_{u,v,w \in V}\ Y_{uv} + X_{vw} - Y_{uw} \leq 1$$

$$\forall_{u,v,w \in V}\ Y_{uv} + Y_{vw} - X_{uw} \leq 1$$

The modified constraints capture the transitivity behavior of direct-mapping and reversed-mapping edges, as described in Section 3. This results in $2n^2 - n$ variables and about $4n^3$ transitivity constraints, cutting the ILP size in half.

Next, we specify how to derive the function f from the trained classifier using a probabilistic formulation[4]. Following Snow et al. (2006) and Berant et al. (2010), we utilize a probabilistic entailment classifier that computes the posterior $P_{uv} = P(X_{uv} = 1 | F_{uv})$. We want to use P_{uv} to derive the posterior $P(G|F)$, where $F = \cup_{u \neq v} F_{uv}$ and F_{uv} is the feature vector for a node pair (u,v).

Since the classifier was trained on a balanced training set, the prior over the two entailment classes is uniform and so by Bayes rule $P_{uv} \propto P(F_{uv}|X_{uv} = 1)$. Using that and the exact same three independence assumptions described by Snow et al. (2006) and Berant et al. (2010) we can show that (for brevity, we omit the full derivation):

$$\hat{G} = arg\,max_G \log P(G|F) = \quad (7)$$

$$arg\,max \sum_{u \neq v} (\log \frac{P_{uv} \cdot P(X_{uv}=1)}{(1-P_{uv})P(X_{uv}=0)}) X_{uv}$$

$$= arg\,max \sum_{u \neq v} (\log \frac{P_{uv}}{1-P_{uv}}) X_{uv} + \log \eta \cdot |E|$$

where $\eta = \frac{P(X_{uv}=1)}{P(X_{uv}=0)}$ is the prior odds ratio for an edge in the graph. Comparing Eq. 1 and 7 we see that $f(u,v) = \log \frac{P_{uv} \cdot P(X_{uv}=1)}{(1-P_{uv})P(X_{uv}=0)}$. Note that f is composed of a likelihood component and an *edge prior* expressed by $P(X_{uv} = 1)$, which we assume to be some constant. This constant is a parameter that affects graph sparsity and controls the trade-off between recall and precision.

Next, we show how sparsity is exploited to scale the use of ILP solvers. We discuss two-types entailment graphs, but generalization is simple.

4.3 Graph decomposition

Though ILP solvers provide an optimal solution, they substantially restrict the size of graphs we can work with. The number of constraints is $O(n^3)$, and solving graphs of size > 50 is often not feasible. To overcome this, we take advantage of graph sparsity: most predicates in language do not entail one another. Thus, it might be possible to decompose graphs into small components and solve each

[4] We describe two-types graphs but extending to single-type graphs is straightforward.

Algorithm 1 Decomposed-ILP

Input: A set V and a function $f : V \times V \rightarrow \mathbf{R}$
Output: An optimal set of directed edges E^*

1: $E' = \{(u, v) : f(u, v) > 0 \vee f(v, u) > 0\}$
2: $V_1, V_2, ..., V_k \leftarrow$ connected components of $G' = (V, E')$
3: **for** $i = 1$ **to** k **do**
4: $\quad E_i \leftarrow$ ApplyILPSolve(V_i,f)
5: **end for**
6: $E^* \leftarrow \bigcup_{i=1}^{k} E_i$

component separately. This is formalized in the next proposition.

Proposition 1. *If we can partition a set of nodes V into disjoint sets U, W such that for any crossing edge (u, w) between them (in either direction), $f(u, w) < 0$, then the optimal set of edges E_{opt} does not contain any crossing edge.*

Proof Assume by contradiction that E_{opt} contains a set of crossing edges E_{cross}. We can construct $E_{new} = E_{opt} \setminus E_{cross}$. Clearly $\sum_{(u,v)\in E_{new}} f(u, v) > \sum_{(u,v)\in E_{opt}} f(u, v)$, as $f(u, v) < 0$ for any crossing edge.

Next, we show that E_{new} does not violate transitivity constraints. Assume it does, then the violation is caused by omitting the edges in E_{cross}. Thus, there must be a node $u \in U$ and $w \in W$ (w.l.o.g) such that for some node v, (u, v) and (v, w) are in E_{new}, but (u, w) is not. However, this means either (u, v) or (v, w) is a crossing edge, which is impossible since we omitted all crossing edges. Thus, E_{new} is a better solution than E_{opt}, contradiction. □

This proposition suggests a simple algorithm (see Algorithm 1): Add to the graph an undirected edge for any node pair with a positive score, then find the connected components, and apply an ILP solver over the nodes in each component. The edges returned by the solver provide an optimal (not approximate) solution to the optimization problem.

The algorithm's complexity is dominated by the ILP solver, as finding connected components takes $O(V^2)$ time. Thus, efficiency depends on whether the graph is sparse enough to be decomposed into small components. Note that the edge prior plays an important role: low values make the graph sparser and easier to solve. In Section 5 we empirically test how typed entailment graphs benefit from decomposition given different prior values.

Algorithm 2 Incremental-ILP

Input: A set V and a function $f : V \times V \rightarrow \mathbf{R}$
Output: An optimal set of directed edges E^*

1: ACT,VIO $\leftarrow \phi$
2: **repeat**
3: $\quad E^* \leftarrow$ ApplyILPSolve(V,f,ACT)
4: $\quad$ VIO $\leftarrow$ violated(V, E^*)
5: $\quad$ ACT $\leftarrow$ ACT $\cup$ VIO
6: **until** |VIO| $= 0$

From a more general perspective, this algorithm can be applied to any problem of learning a sparse transitive binary relation. Such problems include Co-reference Resolution (Finkel and Manning, 2008) and Temporal Information Extraction (Ling and Weld, 2010). Last, the algorithm can be easily parallelized by solving each component on a different core.

4.4 Incremental ILP

Another solution for scaling ILP is to employ incremental ILP, which has been used in dependency parsing (Riedel and Clarke, 2006). The idea is that even if we omit the transitivity constraints, we still expect most transitivity constraints to be satisfied, given a good local entailment classifier. Thus, it makes sense to avoid specifying the constraints ahead of time, but rather add them when they are violated. This is formalized in Algorithm 2.

Line 1 initializes an *active* set of constraints and a *violated* set of constraints (*ACT;VIO*). Line 3 applies the ILP solver with the active constraints. Lines 4 and 5 find the violated constraints and add them to the active constraints. The algorithm halts when no constraints are violated. The solution is clearly optimal since we obtain a maximal solution for a less-constrained problem.

A pre-condition for using incremental ILP is that computing the violated constraints (Line 4) is efficient, as it occurs in every iteration. We do that in a straightforward manner: For every node v, and edges (u, v) and (v, w), if $(u, w) \notin E^*$ we add (u, v, w) to the violated constraints. This is cubic in worst-case but assuming the degree of nodes is bounded by a constant it is linear, and performs very

fast in practice.

Combining *Incremental-ILP* and *Decomposed-ILP* is easy: We decompose any large graph into its components and apply Incremental ILP on each component. We applied this algorithm on our evaluation data set (Section 5) and found that it converges in at most 6 iterations and that the maximal number of active constraints in large graphs drops from $\sim 10^6$ to $\sim 10^3 - 10^4$.

5 Experimental Evaluation

In this section we empirically answer the following questions: (1) Does transitivity improve rule learning over typed predicates? (Section 5.1) (2) Do *Decomposed-ILP* and *Incremental-ILP* improve scalability? (Section 5.2)

5.1 Experiment 1

A data set of 1 million TextRunner tuples (Banko et al., 2007), mapped to 10,672 distinct typed predicates over 156 types was provided by Schoenmackers et al. (2010). Readers are referred to their paper for details on mapping of tuples to typed predicates. Since entailment only occurs between predicates that share the same types, we decomposed predicates by their types (e.g., all predicates with the types *place* and *disease*) into 2,303 *typed entailment graphs*. The largest graph contains 118 nodes and the total number of potential rules is 263,756.

We generated a training set by applying the procedure described in Section 4.1, yielding 2,644 examples. We used SVMperf (Joachims, 2005) to train a Gaussian kernel classifier and computed P_{uv} by projecting the classifier output score, S_{uv}, with the sigmoid function: $P_{uv} = \frac{1}{1+exp(-S_{uv})}$. We tuned two SVM parameters using 5-fold cross validation and a development set of two typed entailment graphs.

Next, we used our algorithm to learn rules. As mentioned in Section 4.2, we integrate background knowledge using the sets A_{yes} and A_{no} that contain predicate pairs for which we know whether entailment holds. A_{yes} was constructed with syntactic rules: We normalized each predicate by omitting the first word if it is a modal and turning passives to actives. If two normalized predicates are equal they are synonymous and inserted into A_{yes}. A_{no} was constructed from 3 sources (1) Predicates differing by a single pair of words that are WordNet antonyms (2) Predicates differing by a single word of negation (3) Predicates $p(t_1, t_2)$ and $p(t_2, t_1)$ where p is a transitive verb (e.g., *beat)* in VerbNet (Kipper-Schuler et al., 2000).

We compared our algorithm (termed ILP_{scale}) to the following baselines. First, to 10,000 rules released by Schoenmackers et al. (2010) (*Sherlock*), where the LHS contains a single predicate (Schoenmackers et al. released 30,000 rules but 20,000 of those have more than one predicate on the LHS, see Section 2), as we learn rules over the same data set. Second, to distributional similarity algorithms: (a) *SR:* the score used by Schoenmackers et al. as part of the *Sherlock* system. (b) *DIRT:* (Lin and Pantel, 2001) a widely-used rule learning algorithm. (c) *BInc:* (Szpektor and Dagan, 2008) a directional rule learning algorithm. Third, we compared to the entailment classifier with no transitivity constraints (*clsf*) to see if combining distributional similarity scores improves performance over single measures. Last, we added to all baselines background knowledge with A_{yes} and A_{no} (adding the subscript X_k to their name).

To evaluate performance we manually annotated all edges in 10 typed entailment graphs - 7 two-types entailment graphs containing 14, 22, 30, 53, 62, 86 and 118 nodes, and 3 single-type entailment graphs containing 7, 38 and 59 nodes. This annotation yielded 3,427 edges and 35,585 non-edges, resulting in an empirical edge density of 9%. We evaluate the algorithms by comparing the set of edges learned by the algorithms to the gold standard edges.

Figure 2 presents the precision-recall curve of the algorithms. The curve is formed by varying a score threshold in the baselines and varying the edge prior in ILP_{scale}[5]. For figure clarity, we omit *DIRT* and *SR*, since *BInc* outperforms them.

Table 2 shows micro-recall, precision and F_1 at the point of maximal F_1, and the Area Under the Curve (AUC) for recall in the range of 0-0.45 for all algorithms, given background knowledge (knowledge consistently improves performance by a few points for all algorithms). The table also shows results for the rules from $Sherlock_k$.

[5] we stop raising the prior when run time over the graphs exceeds 2 hours. Often when the solver does not terminate in 2 hours, it also does not terminate after 24 hours or more.

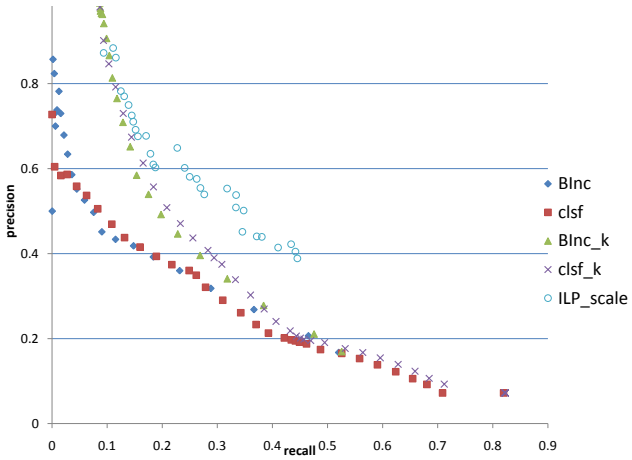

Figure 2: Precision-recall curve for the algorithms.

	micro-average			
	R (%)	P (%)	F_1 (%)	AUC
ILP_{scale}	**43.4**	42.2	**42.8**	**0.22**
$clsf_k$	30.8	37.5	33.8	0.17
$Sherlock_k$	20.6	**43.3**	27.9	N/A
$BInc_k$	31.8	34.1	32.9	0.17
SR_k	38.4	23.2	28.9	0.14
$DIRT_k$	25.7	31.0	28.1	0.13

Table 2: micro-average F_1 and AUC for the algorithms.

Results show that using global transitivity information substantially improves performance. ILP_{scale} is better than all other algorithms by a large margin starting from recall .2, and improves AUC by 29% and the maximal F_1 by 27%. Moreover, ILP_{scale} doubles recall comparing to the rules from the *Sherlock* resource, while maintaining comparable precision.

5.2 Experiment 2

We want to test whether using our scaling techniques, *Decomposed-ILP* and *Incremental-ILP*, allows us to reach the optimal solution in graphs that otherwise we could not solve, and consequently increase the number of learned rules and the overall recall. To check that, we run ILP_{scale}, with and without these scaling techniques (termed ILP^{-}).

We used the same data set as in Experiment 1 and learned edges for all 2,303 entailment graphs in the data set. If the ILP solver was unable to hold the ILP in memory or took more than 2 hours for some graph, we did not attempt to learn its edges. We ran ILP_{scale} and ILP^{-} in three density modes to examine the behavior of the algorithms for different graph densities: (a) $\log\eta = -0.6$: the configuration that achieved the best recall/precision/F_1 of 43.4/42.2/42.8. (b) $\log\eta = -1$ with recall/precision/F_1 of 31.8/55.3/40.4. (c) $\log\eta = -1.75$: A high precision configuration with recall/precision/F_1 of 0.15/0.75/0.23 [6].

$\log\eta$	# unlearned	# rules	△	Red.
-1.75	9/0	6,242 / 7,466	20%	75%
-1	9/1	16,790 / 19,396	16%	29%
-0.6	9/3	26,330 / 29,732	13%	14%

Table 3: Impact of scaling techinques (ILP^{-}/ILP_{scale}).

In each run we counted the number of graphs that could not be learned and the number of rules learned by each algorithm. In addition, we looked at the 20 largest graphs in our data (49-118 nodes) and measured the ratio r between the size of the largest component after applying *Decomposed-ILP* and the original size of the graph. We then computed the average $1-r$ over the 20 graphs to examine how graph size drops due to decomposition.

Table 3 shows the results. Column *# unlearned* and *# rules* describe the number of unlearned graphs and the number of learned rules. Column △ shows relative increase in the number of rules learned and column *Red.* shows the average $1-r$.

ILP_{scale} increases the number of graphs that we are able to learn: in our best configuration ($\log\eta = -0.6$) only 3 graphs could not be handled comparing to 9 graphs when omitting our scaling techniques. Since the unlearned graphs are among the largest in the data set, this adds 3,500 additional rules. We compared the precision of rules learned only by ILP_{scale} with that of the rules learned by both, by randomly sampling 100 rules from each and found precision to be comparable. Thus, the additional rules learned translate into a 13% increase in relative recall without harming precision.

Also note that as density increases, the number of rules learned grows and the effectiveness of decomposition decreases. This shows how *Decomposed-ILP* is especially useful for sparse graphs. We re-

[6] Experiment was run on an Intel i5 CPU with 4GB RAM.

lease the 29,732 rules learned by the configuration $\log \eta = -0.6$ as a resource.

To sum up, our scaling techniques allow us to learn rules from graphs that standard ILP can not handle and thus considerably increase recall without harming precision.

6 Conclusions and Future Work

This paper proposes two contributions over two recent works: In the first, Berant et al. (2010) presented a *global* optimization procedure to learn entailment rules between predicates using transitivity, and applied this algorithm over small graphs where all predicates have one argument instantiated by a target concept. Consequently, the rules they learn are of limited applicability. In the second, Schoenmackers et al. learned rules of wider applicability by using *typed predicates*, but utilized a *local* approach.

In this paper we developed an algorithm that uses *global* optimization to learn widely-applicable entailment rules between typed predicates (where *both* arguments are variables). This was achieved by appropriately defining entailment graphs for typed predicates, formulating an ILP representation for them, and introducing scaling techniques that include graph decomposition and incremental ILP. Our algorithm is guaranteed to provide an optimal solution and we have shown empirically that it substantially improves performance over Schoenmackers et al.'s recent resource and over several baselines.

In future work, we aim to scale the algorithm further and learn entailment rules between untyped predicates. This would require explicit modeling of predicate ambiguity and using approximation techniques when an optimal solution cannot be attained.

Acknowledgments

This work was performed with financial support from the Turing Center at The University of Washington during a visit of the first author (NSF grant IIS-0803481). We deeply thank Oren Etzioni and Stefan Schoenmackers for providing us with the data sets for this paper and for numerous helpful discussions. We would also like to thank the anonymous reviewers for their useful comments. This work was developed under the collaboration of FBK-irst/University of Haifa and was partially supported by the Israel Science Foundation grant 1112/08. The first author is grateful to IBM for the award of an IBM Fellowship, and has carried out this research in partial fulllment of the requirements for the Ph.D. degree.

References

J. Fillmore Baker, C. F. and J. B. Lowe. 1998. The Berkeley framenet project. In *Proc. of COLING-ACL*.

Michele Banko, Michael Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. In *Proceedings of IJCAI*.

Roni Ben Aharon, Idan Szpektor, and Ido Dagan. 2010. Generating entailment rules from framenet. In *Proceedings of ACL*.

Jonathan Berant, Ido Dagan, and Jacob Goldberger. 2010. Global learning of focused entailment graphs. In *Proceedings of ACL*.

Rahul Bhagat, Patrick Pantel, and Eduard Hovy. 2007. LEDIR: An unsupervised algorithm for learning directionality of inference rules. In *Proceedings of EMNLP-CoNLL*.

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31:273–381.

Ido Dagan, Bill Dolan, Bernardo Magnini, and Dan Roth. 2009. Recognizing textual entailment: Rational, evaluation and approaches. *Natural Language Engineering*, 15(4):1–17.

Quang Do and Dan Roth. 2010. Constraints based taxonomic relation classification. In *Proceedings of EMNLP*.

Saso Dzeroski and Ivan Brakto. 1992. Handling noise in inductive logic programming. In *Proceedings of the International Workshop on Inductive Logic Programming*.

Christiane Fellbaum, editor. 1998. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press.

Jenny Rose Finkel and Christopher D. Manning. 2008. Enforcing transitivity in coreference resolution. In *Proceedings of ACL-08: HLT, Short Papers*.

Marti Hearst. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of COLING*.

Thorsten Joachims. 2005. A support vector method for multivariate performance measures. In *Proceedings of ICML*.

Karin Kipper-Schuler, Hoa Trand Dang, and Martha Palmer. 2000. Class-based construction of verb lexicon. In *Proceedings of AAAI/IAAI*.

Dekang Lin and Patrick Pantel. 2001. Discovery of inference rules for question answering. *Natural Language Engineering*, 7(4):343–360.

Xiao Ling and Daniel S. Weld. 2010. Temporal information extraction. In *Proceedings of AAAI*.

Catherine Macleod, Ralph Grishman, Adam Meyers, Leslie Barrett, and Ruth Reeves. 1998. NOMLEX: A lexicon of nominalizations. In *Proceedings of COLING*.

Andre Martins, Noah Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of ACL*.

Eric McCreath and Arun Sharma. 1997. ILP with noise and fixed example size: a bayesian approach. In *Proceedings of the Fifteenth international joint conference on artificial intelligence - Volume 2*.

Vladimir Nikulin. 2008. Classification of imbalanced data with random sets and mean-variance filtering. *IJDWM*, 4(2):63–78.

Hoifung Poon and Pedro Domingos. 2010. Unsupervised ontology induction from text. In *Proceedings of ACL*.

Sebastian Riedel and James Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of EMNLP*.

Stefan Schoenmackers, Oren Etzioni Jesse Davis, and Daniel S. Weld. 2010. Learning first-order horn clauses from web text. In *Proceedings of EMNLP*.

Satoshi Sekine. 2005. Automatic paraphrase discovery based on context and keywords between ne pairs. In *Proceedings of IWP*.

Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. 2006. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of ACL*.

Idan Szpektor and Ido Dagan. 2008. Learning entailment rules for unary templates. In *Proceedings of COLING*.

Idan Szpektor and Ido Dagan. 2009. Augmenting wordnet-based inference with argument mapping. In *Proceedings of TextInfer*.

Idan Szpektor, Hristo Tanev, Ido Dagan, and Bonaventura Coppola. 2004. Scaling web-based acquisition of entailment relations. In *Proceedings of EMNLP*.

Jason Van Hulse, Taghi Khoshgoftaar, and Amri Napolitano. 2007. Experimental perspectives on learning from imbalanced data. In *Proceedings of ICML*.

Julie Weeds and David Weir. 2003. A general framework for distributional similarity. In *Proceedings of EMNLP*.

Mihalis Yannakakis. 1978. Node-and edge-deletion NP-complete problems. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 253–264, New York, NY, USA. ACM.

Alexander Yates and Oren Etzioni. 2009. Unsupervised methods for determining object and relation synonyms on the web. *Journal of Artificial Intelligence Research*, 34:255–296.