Efficient Global Learning of Entailment Graphs

Jonathan Berant*
Stanford University

Noga Alon**
Tel Aviv University

Ido Dagan†
Bar-Ilan University

Jacob Goldberger‡
Bar-Ilan University

Entailment rules between predicates are fundamental to many semantic-inference applications. Consequently, learning such rules has been an active field of research in recent years. Methods for learning entailment rules between predicates that take into account dependencies between different rules (e.g., entailment is a transitive relation) have been shown to improve rule quality, but suffer from scalability issues, that is, the number of predicates handled is often quite small. In this article, we present methods for learning transitive graphs that contain tens of thousands of nodes, where nodes represent predicates and edges correspond to entailment rules (termed entailment graphs). Our methods are able to scale to a large number of predicates by exploiting structural properties of entailment graphs such as the fact that they exhibit a "tree-like" property. We apply our methods on two data sets and demonstrate that our methods find high-quality solutions faster than methods proposed in the past, and moreover our methods for the first time scale to large graphs containing 20,000 nodes and more than 100,000 edges.

1. Introduction

Performing textual inference is at the heart of many semantic inference applications, such as Question Answering (QA) and Information Extraction (IE). A prominent generic

* Computer Science Department, Stanford University, Stanford, CA 94305.
Web: http://www.stanford.edu/∼joberant.
** Blavatnik School of Computer Science, Tel Aviv University, Tel Aviv, 6997801, Israel.
Web: http://www.tau.ac.il/∼nogaa.
† Computer Science Department, Bar-Ilan University, Ramat-Gan, 52900, Israel.
Web: http://www.cs.biu.ac.il/∼dagan.
‡ Engineering Department, Bar-Ilan University, Ramat-Gan, 52900, Israel.
Web: http://www.eng.biu.ac.il/goldbej.

Submission received: 23 November 2013; revised version received: 26 September 2014; accepted for publication: 25 November 2014.

doi:10.1162/COLI_a_00220

© 2015 Association for Computational Linguistics

paradigm for textual inference is Textual Entailment (Dagan et al. 2013). In textual entailment, the goal is to recognize, given two text fragments termed **text** and **hypothesis**, whether the hypothesis can be inferred from the text. For example, the text *Cyprus was invaded by the Ottoman Empire in 1571* implies the hypothesis *The Ottomans attacked Cyprus*.

Semantic inference applications such as QA and IE crucially rely on entailment rules (Ravichandran and Hovy 2002; Shinyama and Sekine 2006; Berant and Liang 2014) or equivalently, inference rules—that is, rules that describe a directional inference relation between two fragments of text. An important type of entailment rule specifies the entailment relation between natural language predicates, for example, the entailment rule *X invade Y*⇒*X attack Y* can be helpful in inferring the aforementioned hypothesis. Consequently, substantial effort has been made to learn such rules (Lin and Pantel 2001; Sekine 2005; Szpektor and Dagan 2008; Schoenmackers et al. 2010; Melamud et al. 2013).

Textual entailment is inherently a transitive relation, that is, the rules $x \Rightarrow y$ and $y \Rightarrow z$ imply the rule $x \Rightarrow z$. For example, from the rules *X reduce nausea* ⇒ *X help with nausea* and *X help with nausea* ⇒ *X associated with nausea* we can infer the rule *X reduce nausea* ⇒ *X associated with nausea* (Figure 1). Accordingly, Berant, Dagan, and Goldberger (2012) proposed taking advantage of transitivity to improve learning of entailment rules. They modeled learning entailment rules as a graph optimization problem, where nodes are predicates and edges represent entailment rules that respect transitivity. To solve this optimization problem, they formulated it as an Integer Linear Program (ILP) and used an off-the-shelf ILP solver to find an exact solution. Indeed, they showed that applying global transitivity constraints results in more accurate graphs (known as **entailment graphs**) than methods that ignore the property of transitivity.

Although using an off-the-shelf ILP solver is straightforward, finding the optimal set of edges respecting transitivity is NP-hard, and practically, transitivity constraints impose substantial restrictions on the scalability of the methods. In fact, in some cases finding the optimal set of edges for entailment graphs with even ∼50 nodes was quite slow.

In this article, we develop algorithms for learning entailment graphs that take advantage of structural properties such as transitivity, but substantially reduce the computational cost of inference, thus allowing us to solve the aforementioned optimization problem on very large graphs.

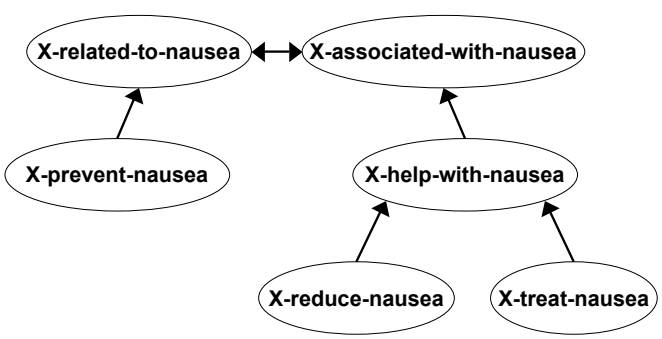

Figure 1
A fragment of an entailment graph about the concept *nausea* from the data set used by Berant, Dagan, and Goldberger. Edges that can be inferred by transitivity are omitted for clarity.

Our method contains two main steps. The first step is based on a **sparsity** assumption that even if an entailment graph contains many predicates, most of them do not entail one another, and thus we can decompose the graph into smaller components that can be solved more efficiently. For example, the predicates *X parent of Y*, *X child of Y* and *X relative of Y* are independent from the predicates *X works with Y*, *X boss of Y*, and *X manages Y*, and thus we can consider each one of these two sets separately. We prove that finding the optimal solution for each of the smaller components results in a global optimal solution for our optimization problem.

The second step proposes a polynomial heuristic approximation algorithm for finding a transitive set of edges in each one of the smaller components. It is based on a novel modeling assumption that entailment graphs exhibit a "tree-like" property, which we term **forest reducible**. For example, the graph in Figure 1 is not a tree, because the predicates *X related to nausea* and *X associated with nausea* form a cycle. However, these two predicates are synonymous, and if they were merged into a single node, then the graph would become a tree. We propose a simple iterative approximation algorithm, where in each iteration a single node is deleted from the graph and then inserted back in a way that improves the objective function value. We prove that if we impose a constraint that entailment graphs must be forest reducible, then each iteration can be performed in linear time. This results in an algorithm that can scale to entailment graphs containing tens of thousands of nodes.

We apply our algorithm on two data sets. The first data set includes medium-sized entailment graphs where predicates are **typed**, that is, the arguments are restricted to belong to a particular semantic type (for instance, X_{person} *parent of* Y_{person}). We show that using our algorithm, we can substantially improve runtime, suffering from only a slight reduction in the quality of learned entailment graphs. The second data set includes a much larger graph containing 20,000 untyped nodes, where applying state-of-the-art methods that use an ILP solver is completely impossible. We run our algorithm on this data set and demonstrate that we can learn knowledge bases with more than 100,000 entailment rules at a higher precision compared with local learning methods.

The article is organized as follows. In Section 2 we survey prior work on the learning of entailment rules. We first focus on **local** methods (Section 2.1), that is, methods that handle each pair of predicates independently of other predicates, and then describe **global** methods (Section 2.2), that is, methods that take into account multiple predicates simultaneously. Section 3 is the core of the article and describes our main algorithmic contributions. After formalizing entailment rule learning as a graph optimization problem, we present the two steps of our algorithm. Section 3.1 describes the first step, in which a large entailment graph is decomposed into smaller components. Section 3.2 describes the second step, in which we develop an efficient heuristic approximation based on the assumption that entailment graphs are forest reducible. Section 4 describes experiments on the first data set containing medium-sized entailment graphs with typed predicates. Section 5 presents an empirical evaluation on a large graph containing 20,000 untyped predicates. We also perform a qualitative analysis in Section 5.4 to further elucidate the behavior of our algorithm. Section 6 concludes the article.

This article is based on previous work (Berant, Dagan, and Goldberger 2011; Berant et al. 2012), but expands over it in multiple directions. Empirically, we present results on a novel data set (Section 5) that is by orders of magnitude larger than in the past. Algorithmically, we present the Tree-Node-And-Component-Fix algorithm, which is an extension of the Tree-Node-Fix algorithm presented in Berant et al. (2012) and achieves best results in our experimental evaluation. Theoretically, we provide an NP-hardness

proof for the *Max-Trans-Forest* optimization problem presented in Berant et al. (2012) and an ILP formulation for it. Last, we perform in Section 5.4 an extensive qualitative analysis of the graphs learned by our local and global algorithms from which we draw conclusions for future research directions.

2. Background

In this section we describe prior work relevant for entailment rule learning. First, we describe **local** methods that estimate entailment for a pair of predicates, focusing on methods that we employ in this article. Then, we describe **global** methods that perform inference over a larger set of predicates. Specifically, we provide details on the method and optimization problem developed by Berant, Dagan, and Goldberger (2012) for which we propose scalable inference algorithms in this article. Last, we survey some other related work in NLP that uses global inference.

2.1 Local Learning

In local learning, given a pair of predicates (i, j) we would like to determine whether $i \Rightarrow j$. The main sources of information utilized in the past for local learning were (1) lexicographic resources, (2) co-occurrence, and (3) distributional similarity. We briefly describe the first two and then expand more on distributional similarity, which is the most commonly used source of information.

Lexicographic resources are manually built knowledge bases from which semantic information may be extracted. For example, the hyponymy, toponymy, and synonymy relations in WordNet (Fellbaum 1998) can be used to detect entailment between nouns and verbs. Although WordNet is the most popular lexicographic resource, other resources such as CatVar, Nomlex, and FrameNet have also been utilized to extract inference rules (Meyers et al. 2004; Budanitsky and Hirst 2006; Szpektor and Dagan 2009; Coyne and Rambow 2009; Ben Aharon, Szpektor, and Dagan 2010).

Pattern-based methods attempt to identify the semantic relation between a pair of predicates by examining their co-occurrence in a large corpus. For example, the sentence *people snore while they sleep* provides evidence that *snore* $\Rightarrow$ *sleep*. While most pattern-based methods focused on identifying semantic relations between nouns (for example, Hearst patterns [Hearst 1992]), several works (Pekar 2008; Chambers and Jurafsky 2011; Weisman et al. 2012) attempted to extract relations between predicates as well. Chklovsky and Pantel (2004) used pattern-based methods to generate the commonly used VerbOcean resource.

Both lexicographic as well as pattern-based methods suffer from limited coverage. Distributional similarity is therefore used to automatically construct broad-scale resources. Distributional similarity methods are based on the "distributional hypothesis" (Harris 1954) that semantically similar predicates occur with similar arguments. Quite a few methods have been suggested (Lin and Pantel 2001; Szpektor et al. 2004; Bhagat, Pantel, and Hovy 2007; Szpektor and Dagan 2008; Yates and Etzioni 2009; Schoenmackers et al. 2010), which differ in terms of the specifics of the ways in which predicates are represented, the features that are extracted, and the function used to compute feature vector similarity. Next, we elaborate on the methods that we use in this article.

Lin and Pantel (2001) proposed an algorithm for learning paraphrase relations between binary predicates, that is, predicates with two variables such as *X treat Y*. For each binary predicate, Lin and Pantel compute two sets of features F_x and F_y,

which are the words that instantiate the arguments X and Y, respectively, in a large corpus. Given a predicate u and its feature set for the X variable F_x, every feature $f_x \in F_x$ is weighted by pointwise mutual information between the predicate and the feature: $w(f_x) = \log \frac{Pr(f_x|u)}{Pr(f_x)}$, where the probabilities are computed using maximum likelihood over the corpus. Given two predicates u and v, the *Lin* measure (Lin 1998) is computed for the variable X in the following manner:

$$Lin_x(u,v) = \frac{\sum_{f \in F_x^u \cap F_x^v} [w_x^u(f) + w_x^v(f)]}{\sum_{f \in F_x^u} w_x^u(f) + \sum_{f \in F_x^v} w_x^v(f)} \tag{1}$$

The measure is computed analogously for the variable Y and the final distributional similarity score, termed DIRT, is the geometric average of the scores for the two variables: If $DIRT(u,v)$ is high, this means that the templates u and v share many "informative" arguments and so it is possible that $u \Rightarrow v$.

Szpektor and Dagan (2008) suggested two modifications to DIRT. First, they looked at **unary predicates**, that is, predicates with a single variable such as *X treat*. Secondly, they computed a directional score that is more suited for capturing entailment relations compared to the symmetric *Lin* score. They proposed that if for two unary predicates $u \Rightarrow v$, then relatively many of the features of u should be covered by the features of v. This is captured by the asymmetric *Cover* measure (Weeds and Weir 2003):

$$Cover(u,v) = \frac{\sum_{f \in F^u \cap F^v} w^u(f)}{\sum_{f \in F^u} w^u(f)} \tag{2}$$

The final directional score, termed *BInc* (Balanced Inclusion), is the geometric average of the *Lin* measure and the *Cover* measure.

Both Lin and Pantel as well as Szpektor and Dagan compute a similarity score using a single argument. However, it is clear that although this alleviates sparsity problems, considering pairs of arguments jointly provides more information. Yates and Etzioni (2009), Schoenmackers et al. (2010), and even earlier, Szpektor et al. (2004), presented methods that compute semantic similarity based on pairs of arguments.

A problem common to all local methods presented above is predicate ambiguity—predicates may have different meanings and different entailment relations in different contexts. Some works resolved the problem of ambiguity by representing predicates with argument variables that are **typed** (Pantel et al. 2007; Schoenmackers et al. 2010). For example, argument variables in the work of Schoenmackers et al. were restricted to belong to one of 156 types, such as *country* or *profession*. A different solution that has attracted substantial attention recently is to represent the various contexts in which a predicate can appear in a low-dimensional latent space (for example, using Latent Dirichlet Allocation [Blei, Ng, and Jordan 2003]) and infer entailment relations between predicates based on the contexts in which they appear (Ritter, Mausam, and Etzioni 2010; ÓSéaghdha 2010; Dinu and Lapata 2010; Melamud et al. 2013). In the experiments presented in this article we will use the representation of Schoenmackers et al. in one experiment, and ignore the problem of predicate ambiguity in the other.

2.2 Global Learning

The idea of global learning is that, by jointly learning semantic relations between a large number of natural language phrases, one can use the dependencies between the relations to improve accuracy. A natural way to model that is with a graph where nodes are phrases and edges represent semantic similarity. Snow, Jurafsky, and Ng (2006) presented one of the early examples of global learning in the context of learning noun taxonomies. In their work, they enforced a transitivity constraint over the taxonomy using a greedy inference procedure and demonstrated that this improves the quality of the taxonomy. Transitivity constraints were also enforced by Yates and Etzioni (2009), who proposed a clustering algorithm for learning undirected synonymy relations. Nakashole, Weikum, and Suchanek (2012) learned a taxonomy of binary predicates and also enforced transitivity with a greedy algorithm.

Berant, Dagan, and Goldberger (2012) developed a global method for learning entailment relations between predicates. In this article we present scalable approximation algorithms for the optimization problem they propose, and so we provide further detail on their method. The input to their algorithm is a large corpus and a lexicographic resource, such as WordNet, and the output is a set of entailment rules that respect the constraint of transitivity. The algorithm is composed of two main steps. In the first step, a set of predicates is extracted from the corpus and a local entailment classifier is trained based on examples automatically generated from the lexicographic resource. At this point, we can derive for each pair of predicates (i, j) a score $w_{ij} \in \mathbb{R}$ that estimates whether $i \Rightarrow j$.

In the second step of their algorithm, they construct a graph, where predicates are nodes and edges represent entailment rules. Using the local scores they look for the set of edges E that maximizes the objective function $\sum_{(i,j)\in E} w_{ij}$ under the constraint that edges respect transitivity. They show that this optimization problem is NP-hard and find an exact solution using an ILP solver over small graphs. They also avoid problems of predicate ambiguity by partially contextualizing the predicates. In this article, we present efficient and scalable heuristic approximation algorithms for the optimization problem they propose.

In recent years, there has been substantial work on approximation algorithms for global inference problems. Do and Roth (2010) suggested a method for the related task of learning taxonomic relations between terms. Given a pair of terms, they construct a small graph that contains the two terms and a few other related terms, and then impose constraints on the graph structure. They construct these small graphs because their work is geared towards scenarios where relations are determined on-the-fly for a given pair of terms and no global knowledge base is ever explicitly constructed. Because they independently construct a graph for each pair of terms, their method easily produces solutions where global constraints, such as transitivity, are violated.

Another approximation method that violates transitivity constraints is LP relaxation (Martins, Smith, and Xing 2009). In LP relaxation, binary variables are replaced by continuous variables, transforming the problem from an ILP to a Linear Program (LP), which is polynomial. An LP solver is then applied, and variables that are assigned a fractional value are rounded to their nearest integer, so many violations of transitivity may occur. The solution when applying LP relaxation is not a transitive graph; we will show in Section 4 that our approximation method is substantially faster.

Global inference has gained popularity in recent years in NLP, and a common approximation method that has been extensively utilized is dual decomposition (Sontag, Globerson, and Jaakkola 2011). Dual decomposition has been successfully applied in

tasks such as Information Extraction (Reichart and Barzilay 2012), Machine Translation (Chang and Collins 2011), Parsing (Rush et al. 2012), and Named Entity Recognition (Wang, Che, and Manning 2013). To the best of our knowledge, it has not yet been applied for the task of learning entailment relations. The graph decomposition method we present in Section 3.1 can be viewed as an ideal case of dual decomposition, where we can decompose the problem into disjoint components in a way that we do not need to ensure consistency of the results obtained on each component separately.

3. Efficient Inference

Our goal is to learn a large knowledge base of entailment rules between natural language predicates. Following Berant, Dagan, and Goldberger (2012), we formulate this task as a graph-learning problem, where, given the nodes of an entailment graph, we would like to find the best set of edges that respect a global transitivity constraint.

Our main modeling assumption is that textual entailment is a transitive relation. Berant, Dagan, and Goldberger (2012) have demonstrated that this modeling assumption holds for *focused entailment graphs*, that is, graphs in which predicates are disambiguated by one of their arguments. For example, a graph that focuses on the concept *nausea* might contain a en entailment rule between predicates such as *X prevents nausea* $\Rightarrow$ *X affects nausea*, where the predicate *X prevent Y* is disambiguated by the instantiation of the argument *nausea*. In this work, we will examine this modeling assumption for more general graphs. In Section 4 we show that transitivity holds in *typed entailment graphs*, that is, graphs where each predicate specifies the semantic type of its arguments (for example, X_{drug} *prevent* $Y_{symptom}$). In Section 5.4, we examine the assumption of transitivity when predicates do not carry any typing information (for example, *X prevent Y*); and we observe that in this setting the assumption of transitivity is often violated because of the predicate ambiguity. Nevertheless, we show that even in this set-up we can empirically take advantage of the transitivity assumption.

Let V be a set of predicates, which are the nodes of the entailment graph, and let $w : V \times V \rightarrow \mathbb{R}$ be an entailment weighting function. Given a pair of predicates (i, j), a positive w_{ij} indicates local tendency to decide that i entails j, whereas a negative w_{ij} indicates local tendency to decide that i does not entail j. We want to find a global entailment transitive graph that is most consistent with the local cues. Formally, our goal is to find the directed graph $\mathcal{G} = (V, E)$ that maximizes the sum of edge weights $\sum_{(i,j)\in E} w_{ij}$, under the constraint that the graph is transitive—that is, for every triple of nodes (i, j, k), if $(i, j) \in E$ and $(j, k) \in E$, then $(i, k) \in E$.

Berant, Dagan, and Goldberger (2012) proved that this optimization problem (which we term Max-Trans-Graph) is NP-hard, and provided an ILP formulation for it. Let x_{ij} be an indicator for whether $i \Rightarrow j$, then $x = \{x_{ij} : i \neq j\}$ are the variables of the following ILP:

$$\max_{x} \sum_{i \neq j} w_{ij} x_{ij} \tag{3}$$

$$\begin{aligned} s.t. \quad & \forall i, j, k \in V \qquad x_{ij} + x_{jk} - x_{ik} \leq 1 \\ & \forall i, j \in V \qquad x_{ij} \in \{0, 1\} \end{aligned}$$

The objective function is the sum of weights over the edges of $\mathcal{G}$, and the constraint $x_{ij} + x_{jk} - x_{ik} \leq 1$ on the binary variables enforces transitivity (i.e., $x_{ij} = x_{jk} = 1$ implies that $x_{ik} = 1$). The weighting function w is trained separately using supervised learning methods and we describe the details of training for each one of our experiments in Sections 4 and 5.

There is a simple probabilistic modeling that motivates the score (3) that we optimize. Assume that for each pair of nodes (i, j), we are given a probability $p_{ij}(1) = p(x_{ij} = 1)$ that $i \Rightarrow j$ (the probability that $i \not\Rightarrow j$ is denoted by $p_{ij}(0) = 1 - p_{ij}(1)$). Assuming a uniform probability over graphs, the posterior probability (which can also be viewed as the likelihood) of a graph, represented by an edge-set x, is:

$$p(x) \propto \prod_{i \neq j} p_{ij}(x_{ij}) \tag{4}$$

It can be easily verified that:[1]

$$\log p(x_{ij}) = \log \frac{p_{ij}(1)}{p_{ij}(0)} x_{ij} + \log p_{ij}(0) \tag{5}$$

Hence,

$$\log p(x) = \sum_{i \neq j} \log p(x_{ij}) = \sum_{i \neq j} w_{ij} x_{ij} + \text{const} \tag{6}$$

such that 'const' is a scalar that does not depend on x and

$$w_{ij} = \log \frac{p_{ij}(1)}{p_{ij}(0)} \tag{7}$$

The optimal entailment graph, therefore, is $\arg\max_x p(x) = \arg\max_x \sum_{i \neq j} w_{ij} x_{ij}$, where the maximization is over all the transitive graphs. Hence the most likely transitive entailment graph is obtained as the solution of the ILP maximization problem (3).

Berant, Dagan, and Goldberger solved this optimization problem with an ILP solver, but because ILP is NP-hard, this does not scale well; the number of variables is $O(|V|^2)$ and the number of constraints is $O(|V|^3)$. Thus, even a graph with 80 nodes (predicates) has more than half a million constraints. In this section, we describe a heuristic algorithm that empirically provides high-quality solutions for this optimization problem in graphs with tens of thousands of nodes.

Our algorithm contains two main steps. The first step (Section 3.1) is based on a structural assumption that entailment graphs are relatively sparse—that is, most predicates do not entail one another. This allows us to decompose the graph into smaller components in a way that guarantees that an exact solution for each one of the components results in an exact solution for Max-Trans-Graph. However, often even after decomposition, components are too large and finding an exact solution is still intractable.

The second step (Section 3.2) proposes a heuristic algorithm for finding a good solution for each one of the components. This step is based on an observation that

1 We assume that $p_{ij}(1), p_{ij}(0) \in (0, 1)$ and thus $\log p(x_{ij})$ is well defined.

entailment graphs exhibit a "tree-like" property and are very similar to a novel type of graph, which we term **forest-reducible graph**. We utilize this property to develop an iterative efficient approximation algorithm for learning the graph edges, where each iteration takes linear time. We also prove that finding the optimal forest-reducible graph is NP-hard.

In Sections 4 and 5 we apply our algorithm on two different data sets and show that using transitivity substantially improves performance and that our methods dramatically improve scalability, allowing us to increase the scope of global learning of entailment graphs.

3.1 Entailment Graph Decomposition

The first step of our algorithm takes advantage of graph sparsity: Most predicates in language do not entail one another. Thus, it might be possible to decompose entailment graphs into small components and solve each component separately.

Let V_1, V_2 be a partitioning of the nodes V into two disjoint non-empty subsets. We term any directed edge, whose two nodes belong to different subsets, a **crossing edge**.

Proposition 1
If we can partition a set of nodes V into disjoint sets V_1, V_2 such that no crossing edge has a positive weight, then the optimal set of edges that is the solution of the ILP problem (3) does not contain any crossing edge.

Proof Assume by contradiction that the edge-set of the optimal solution E_{opt} contains a non-empty set of crossing edges E_{cross}. We can construct $E_{new} = E_{opt} \setminus E_{cross}$. Clearly $\sum_{(i,j)\in E_{new}} w_{ij} \geq \sum_{(i,j)\in E_{opt}} w_{ij}$, as $w_{ij} \leq 0$ for any crossing edge.

Next, we show that E_{new} does not violate transitivity constraints. Assuming it does, then the violation is caused by omitting the edges in E_{cross}. Thus, there must be, without loss of generality, nodes $i \in V_1$ and $k \in V_2$ such that for some node j, (i, j) and (j, k) are in E_{new}, but (i, k) is not. However, this means either (i, j) or (j, k) is a crossing edge, which is impossible because we omitted all crossing edges. Thus, E_{new} is a better solution than E_{opt}, contradiction. Hence, E_{cross} is empty. ■

This proposition suggests a simple algorithm (see Algorithm 1): Construct an undirected graph with the node set V and with an edge connecting i and j if either $w_{ij} > 0$ or $w_{ji} > 0$, then find its connected components, and finally solve each component separately. If an ILP solver is used to find the edges of each component separately,

Algorithm 1 Decompose

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

```
E' = {(i,j) : w_ij > 0 ∨ w_ji > 0}
V_1, V_2, ..., V_k ← connected components of the undirected graph G' = (V, E')
for l = 1, ..., k do
    E_l ← Solve the ILP Problem (3) restricted to V_l
end for
E* ← ∪_{l=1}^{k} E_l
```

Algorithm 2 SolveCuttingPlane

Input: A set V and a weighting function $w : V \times V \rightarrow \mathbb{R}$
Output: An optimal set of directed edges E^*
1: ACT $\leftarrow \phi$
2: **repeat**
3: $E^* \leftarrow$ Solve ILP (3) using only the constraint subset ACT
4: VIO $\leftarrow$ violated(V, E^*)
5: ACT $\leftarrow$ ACT $\cup$ VIO
6: **until** VIO = ϕ

then we obtain an optimal (not approximate) solution to the optimization problem (3) for the whole graph. Finding the undirected edges (Line 1) and computing connected components (Line 2) can be performed in $O(V^2)$. Thus, in this case the efficiency of the algorithm is dominated by the application of an ILP solver (Line 4).

If the entailment graph decomposes into small components, one could obtain an exact solution with an ILP solver, applied on each component separately, without resorting to any approximation. To further extend scalability in this setting we use a **cutting-plane** method (Kelley 1960). Cutting-plane methods have been used in the past, for example, in dependency parsing (Riedel and Clarke 2006). The idea is that even if we omit all transitivity constraints, we still expect most transitivity constraints to be satisfied, given a good weighting function w. 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 (*ACT*). 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 because we obtain a maximal solution for a less-constrained problem that does not violate any transitivity constraint.

A pre-condition for using cutting-plane methods is that computing the violated constraints (Line 4) is efficient, as it occurs in every iteration. We do that in a straightforward manner: For all edges (i, j) and (j, k) that are in the current solution E^*, if $(i, k) \notin E^*$ we add $x_{ij} + x_{jk} - x_{ik} \leq 1$ to the violated constraints. This is cubic in worst-case and performs very fast in practice.

To conclude, an exact algorithm for Max-Trans-Graph decomposes the graph into components and uses Algorithm 2 to find an exact solution for each component. However, because the problem is NP-hard, this will still fail once components become large. Next, we describe the second step of our method, which replaces Algorithm 2 with an efficient approximation that can scale to much larger graphs. We will show in Section 4 that the solutions obtained by the approximation algorithm are almost as good as the exact solution on a data set where finding an exact solution is feasible.

3.2 Efficient Tree-Based Approximation

The approximation we present in this section is based on a conjecture that entailment graphs exhibit a "tree-like" property, that is, they can be *reduced* into a structure similar to a directed forest, which we term **forest-reducible graph** (FRG). Although FRGs are a more constrained class of directed graphs, we prove that restricting our optimization

problem to FRGs does not make the problem fundamentally easier, that is, the problem remains NP-hard (see Appendix). Then, we present in Section 3.2.2 our iterative approximation algorithm, where in each iteration a node is removed and re-attached back to the graph in a locally optimal way. Combining this scheme with our conjecture about the graph structure yields a linear algorithm for node re-attachment.

Thus, the contribution of this section is twofold: First, we define a novel modeling assumption about the tree-like structure of entailment graphs and empirically demonstrate its validity. Second, we exploit this assumption to develop a polynomial approximation algorithm for learning entailment graphs that can scale to much larger graphs than in the past.

3.2.1 Forest-Reducible Graph. The predicate entailment relation, described by our entailment graphs, is typically from a "semantically specific" predicate to a more "general" one. Thus, intuitively, the topology of an entailment graph is expected to be "tree-like." In this section we first formalize this intuition and then empirically analyze its validity. This property of entailment graphs is an interesting topological observation on its own, but also enables the efficient approximation algorithm of Section 3.2.2.

For a directed edge $i \Rightarrow j$ in a directed acyclic graph (DAG), we term the node i a **child** of node j, and j a **parent** of i.[2] A **directed forest** is a DAG where all nodes have no more than one parent. A strongly connected component in a directed graph is a set of nodes such that there is a path from every node in the set to any other node. In entailment graphs, strongly connected components correspond to semantically equivalent predicates.

The entailment graph in Figure 2a (subgraph from the data set described in Section 4) is clearly not a directed forest—it contains a cycle of size two comprising the nodes *X common in Y* and *X frequent in Y*, and in addition the node *X be epidemic in Y* has three parents. However, we can convert it to a directed forest by applying the following operations. Any directed graph $\mathcal{G}$ can be converted into a **Strongly-Connected-Component** (SCC) graph in the following way: Every strongly connected component is contracted into a single node, and an edge is added from SCC S_1 to SCC S_2 if there is an edge in $\mathcal{G}$ from some node in S_1 to some node in S_2. The SCC graph is always a DAG (Cormen et al. 2002), and if $\mathcal{G}$ is transitive, then the SCC graph is also transitive. The graph in Figure 2b is the SCC graph of the one in Figure 2a, but is still not a directed forest because the node *X be epidemic in Y* has two parents.

The *transitive closure* of a directed graph $\mathcal{G}$ is obtained by adding an edge from node i to node j if there is a path in $\mathcal{G}$ from i to j. The **transitive reduction** of $\mathcal{G}$ is obtained by removing all edges whose absence does not affect its transitive closure. In DAGs, the result of transitive reduction is unique (Aho, Garey, and Ullman 1972). We thus define the **reduced graph** $\mathcal{G}_{red} = (V_{red}, E_{red})$ of a directed graph $\mathcal{G}$ as the transitive reduction of its SCC graph. The graph in Figure 2c is the reduced graph of the one in Figure 2a and is a directed forest. We say a graph is a forest-reducible graph if its reduced graph is a directed forest.

We now hypothesize that entailment graphs are approximately FRGs. The intuition is that the predicate on the left-hand-side of an entailment rule has a more specific meaning than the one on the right-hand-side. For instance, in Figure 2a *X be epidemic in Y* (where *X* is a type of disease and *Y* is a country) is more specific than *X common in Y* and

2 In standard graph terminology an edge is from a parent to a child. We choose the opposite definition to conflate edge direction with the direction of the entailment operator '$\Rightarrow$'.

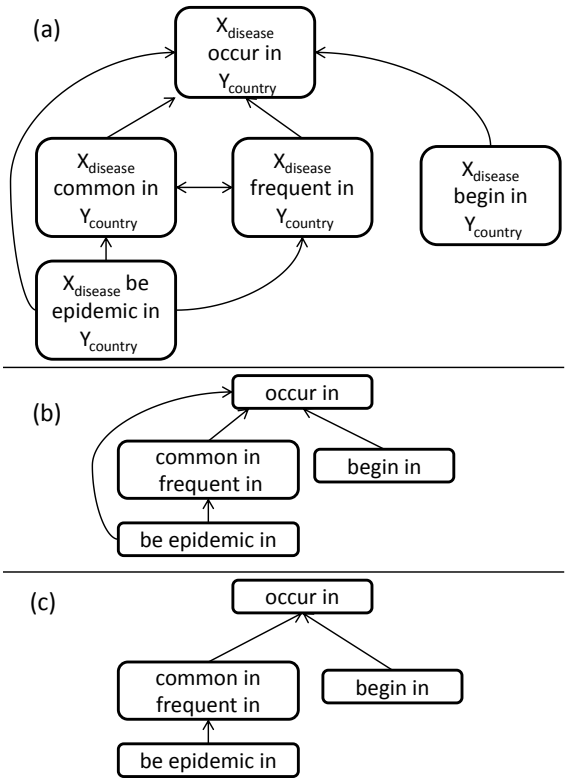

Figure 2
A fragment of an entailment graph (a), its SCC graph (b), and its reduced graph (c). Nodes are predicates with typed variables, which are omitted in (b) and (c) for compactness.

X frequent in Y, which are equivalent, while *X occur in Y* is even more general. Accordingly, the reduced graph in Figure 2c is an FRG. We note that this is not always the case: For example, the entailment graph in Figure 3 is not an FRG, because *X annex Y* entails both *Y be part of X* and *X invade Y*, while the latter two do not entail one another. However, we hypothesize that this scenario is rather uncommon. Consequently, a natural variant of the Max-Trans-Graph problem is to restrict the required output graph of the optimization problem (3) to an FRG. We term this problem Max-Trans-Forest.

To test whether our hypothesis holds empirically, we performed the following analysis. We sampled seven gold standard typed entailment graphs (that is, graphs where

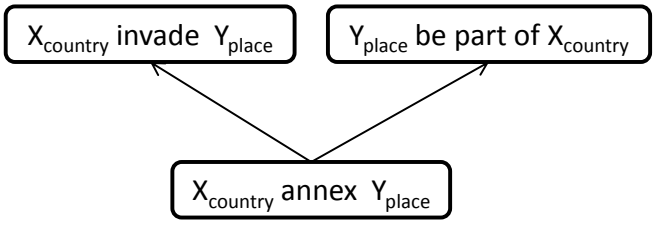

Figure 3
A fragment of an entailment graph that is not an FRG.

predicates specify the semantic type of their arguments, such as X_{drug} *prevents* $Y_{symptom}$) from the data set described in Section 4, manually transformed them into FRGs by deleting a minimal number of edges, and measured recall over the set of edges in each graph (precision is naturally 1.0, as we only delete gold-standard edges). The lowest recall value obtained was 0.95, illustrating that deleting a very small proportion of edges converts a typed entailment graph into an FRG. Further support for the practical validity of this hypothesis is obtained from our experiments on typed entailment graphs in Section 4. In these experiments we show that exactly solving Max-Trans-Graph and Max-Trans-Forest (with an ILP solver) results in nearly identical performance. In Section 5.4 we qualitatively analyze the validity of the FRG assumption in untyped entailment graphs (where predicates are of the form *X prevents Y*) and find that this assumption does not always hold, because of predicate ambiguity.

An ILP formulation for Max-Trans-Forest is simple—a transitive graph is an FRG if all nodes in its reduced graph have no more than one parent. It can be verified that this is equivalent to the following statement: For every triplet of nodes i, j, k, if $i \Rightarrow j$ and $i \Rightarrow k$, then either $j \Rightarrow k$ or $k \Rightarrow j$ (or both). This constraint can be stated in a linear form: $x_{ij} + x_{ik} - x_{jk} - x_{kj} \leq 1$. Therefore, adding this new type of constraint to the ILP given in Equation (3) results in a formulation for Max-Trans-Forest:

$$\max_{x} \sum_{i \neq j} w_{ij} x_{ij} \tag{8}$$

$$\begin{aligned} \textit{such that} \quad & \forall i,j,k \in V \quad && x_{ij} + x_{jk} - x_{ik} \leq 1 \\ & \forall i,j,k \in V \quad && x_{ij} + x_{ik} - x_{jk} - x_{kj} \leq 1 \\ & \forall i,j \in V \quad && x_{ij} \in \{0,1\} \end{aligned}$$

A natural question is whether there is a simpler (polynomial) solution for Max-Trans-Forest that avoids the need for an ILP solver. In the Appendix we prove that Max-Trans-Forest is also an NP-hard problem by a polynomial reduction from the X3C problem (Garey and Johnson 1979). Readers who are not interested in the proof can safely skip the Appendix.

3.2.2 Approximation Algorithm. In this section we present Tree-Node-Fix and Tree-Node-And-Component-Fix, which are efficient approximation algorithms for Max-Trans-Forest, as well as Graph-Node-Fix, an approximation for Max-Trans-Graph.

Tree-Node-Fix. The scheme of Tree-Node-Fix (TNF) is the following. First, an initial FRG is constructed, using some initialization procedure. Then, at each iteration a single node v is *re-attached* (see below) to the FRG in a way that improves the objective function. This is repeated until the value of the objective function can no longer be improved by re-attaching a node.

Re-attaching a node v is performed by removing v from the graph (deleting v and all its adjacent edges) and connecting it back with a better set of edges, while maintaining the constraint that it is an FRG. This is done by considering all possible edges from/to the other graph nodes and choosing the *optimal* subset, while the rest of the graph edges remain fixed. For example, in Figure 2, one way of re-attaching the node *X common in Y* is to add it as a direct child of *X occur in Y* that is not a synonym of *X frequent in Y*. This will result in deletion of the edges *X common in Y* $\Rightarrow$ *X frequent in Y*, *X frequent in Y* $\Rightarrow$ *X common in Y*, and also *X be epidemic in Y* $\Rightarrow$ *X common in Y*, because otherwise the

resulting graph will not be an FRG. We will show that re-attachment can be efficiently performed in linear time using dynamic programming.

Formally, let $S_{v-in} = \sum_{i \neq v} w_{iv} x_{iv}$ be the sum of scores over v's incoming edges and $S_{v-out} = \sum_{k \neq v} w_{vk} x_{vk}$ be the sum of scores over v's outgoing edges. Re-attachment amounts to optimizing a linear objective:

$$\underset{x(v)}{\operatorname{argmax}}(S_{v\text{-}in} + S_{v\text{-}out}) \tag{9}$$

while maintaining the FRG constraint, where the variables $x(v) \subseteq x$ are indicators for all pairs of nodes involving v. We approximate a solution for Equation (3) by iteratively optimizing the simpler objective (9). Clearly, at each re-attachment the value of the objective function cannot decrease, because the optimization algorithm considers the previous graph as one of its candidate solutions.

We now show that re-attaching a node v is linear. To analyze v's re-attachment, we consider the structure of the directed forest $\mathcal{G}_{red}$ just *before* v is re-inserted, and examine the possibilities for v's insertion relative to that structure. We start by defining some helpful notations. Every node $c \in V_{red}$ is a strongly connected component in $\mathcal{G}$. Let $v_c \in c$ be an arbitrary representative node in c (we can choose any node $v_c \in c$, because the graph $\mathcal{G}$ is transitive, and c is a strongly connected component). We denote by $S_{v\text{-}in}(c)$ the sum of weights from all nodes in c and their descendants to v, and by $S_{v\text{-}out}(c)$ the sum of weights from v to all nodes in c and their ancestors:

$$S_{v\text{-}in}(c) = \sum_{i \in c} w_{iv} + \sum_{k \notin c} w_{kv} x_{kv_c} \tag{10}$$

$$S_{v\text{-}out}(c) = \sum_{i \in c} w_{vi} + \sum_{k \notin c} w_{vk} x_{v_c k} \tag{11}$$

Note that $\{x_{v_c k}, x_{kv_c}\}$ are edge indicators in $\mathcal{G}$ and not $\mathcal{G}_{red}$.

There are several cases for re-attaching v that we need to consider:

1. Inserting v into an existing component $c \in V_{red}$ (case 1, see Figure 4a).
2. Inserting v as a new component, which breaks into two subcases:
   (a) Forming a new component that contains only v, where v is a child of a component c (case 2a, see Figure 4b).
   (b) Forming a new component that contains only v, where v is not a child of any component c, and so is a new root in $\mathcal{G}_{red}$ (case 2b, see Figure 4c).

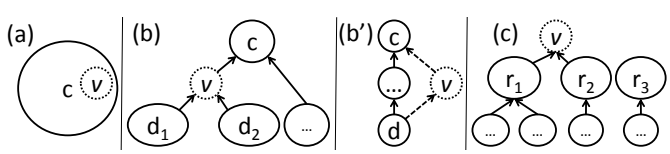

Figure 4
(a) Inserting v into a component $c \in V_{red}$. (b) Inserting v as a child of c and a parent of a subset of c's children in $\mathcal{G}_{red}$. (b') A node d that is a descendant but not a child of c cannot choose v as a parent, as v becomes its second parent. (c) Inserting v as a new root.

Note that v cannot form a new component that contains other nodes as well, because the rest of the graph is fixed.

To find the optimal way of re-attaching v, we need to compute the score Equation (9), for each of the three cases, and choose the best one. We now describe how this score is computed in each of the three cases.

Case 1: Inserting v into a component $c \in V_{red}$. In this case we add in $\mathcal{G}$ edges from all nodes in c and their descendants to v and from v to all nodes in c and their ancestors. The score (9) in this case is

$$s_1(c) \triangleq S_{v\text{-}in}(c) + S_{v\text{-}out}(c) \tag{12}$$

Case 2a: Inserting v as a child of some $c \in V_{red}$. Once c is chosen as the parent of v, choosing v's children in $\mathcal{G}_{red}$ is substantially constrained. A node that is not a descendant of c cannot become a child of v, because this would create a new path from that node to c and would require by transitivity to add a corresponding directed edge to c (but all graph edges not connecting v are fixed). Moreover, only a direct child of c can choose v as a parent instead of c (Figure 4b), because for any other descendant of c, v would become a second parent, and $\mathcal{G}_{red}$ will no longer be a directed forest (Figure 4b'). Thus, this case requires adding in $\mathcal{G}$ edges from v to all nodes in c and their ancestors; additionally, for each new child of v, denoted by $d \in V_{red}$, we add edges from all nodes in d and their descendants to v. Crucially, although the number of possible subsets of c's children in $\mathcal{G}_{red}$ is exponential, the fact that they are independent trees in $\mathcal{G}_{red}$ allows us to go over them one by one, and decide for each one whether it will be a child of v or not, depending on whether $S_{v\text{-}in}(d)$ is positive. Therefore, the score (9) in this case is:

$$s_2(c) \triangleq S_{v\text{-}out}(c) + \sum_{d \in child(c)} \max(0, S_{v\text{-}in}(d)) \tag{13}$$

where $child(c)$ are the children of c.

Case 2b: Inserting v as a new root in $\mathcal{G}_{red}$. Similar to case 2a, only roots of $\mathcal{G}_{red}$ can become children of v. In this case for each chosen root r we add in $\mathcal{G}$ edges from the nodes in r and their descendants to v. Again, each root can be examined independently. Therefore, the score (9) of re-attaching v is:

$$s_3 \triangleq \sum_{r} \max(0, S_{v\text{-}in}(r)) \tag{14}$$

where the summation is over the roots of $\mathcal{G}_{red}$.

It can be easily verified that $S_{v\text{-}in}(c)$ and $S_{v\text{-}out}(c)$ satisfy the recursive definitions:

$$S_{v\text{-}in}(c) = \sum_{i \in c} w_{iv} + \sum_{d \in child(c)} S_{v\text{-}in}(d),\ c \in V_{red} \tag{15}$$

$$S_{v\text{-}out}(c) = \sum_{i \in c} w_{vi} + S_{v\text{-}out}(p),\ c \in V_{red} \tag{16}$$

where p is the parent of c in $\mathcal{G}_{red}$. These recursive definitions allow computing in linear time $S_{v\text{-}in}(c)$ and $S_{v\text{-}out}(c)$ for all c (given $\mathcal{G}_{red}$) using dynamic programming, before going over the cases for re-attaching v. $S_{v\text{-}in}(c)$ is computed going over V_{red} leaves-to-root (post-order), and $S_{v\text{-}out}(c)$ is computed going over V_{red} root-to-leaves (pre-order).

Re-attachment is summarized in Algorithm 3. Computing an SCC graph is linear (Cormen et al. 2002), and it is easy to verify that transitive reduction in FRGs is also linear (Line 1). Computing $S_{v\text{-}in}(c)$ and $S_{v\text{-}out}(c)$ (Lines 2–3) is also linear, as explained. Cases 1 and 2b are trivially linear and in case 2a we go over the children of all nodes in V_{red}. As the reduced graph is a forest, this simply means going over all nodes of V_{red}, and so the entire algorithm is linear.

Because re-attachment is linear, re-attaching all nodes is quadratic. Thus if we bound the number of iterations over all nodes, the overall complexity is quadratic. This is dramatically more efficient and scalable than applying an ILP solver. Empirically, we found that TNF converges after 5–10 iterations.

Tree-Node-and-Component-Fix. Assuming that entailment graphs are FRGs allows us to use the node re-attachment operation in linear time. However, this assumption also enables performing other graph operations efficiently. We now suggest a natural extension to the TNF algorithm that better explores the space of FRGs.

A disadvantage of TNF is that it re-attaches a *single* node in every iteration. Consider, for instance, the reduced graph in Figure 5. In this graph, nodes l, m, and n are direct children of node k, but suppose that in the optimal solution they are all children of node j. Reaching the optimal solution would require three independent re-attachment operations, and it is not clear that each of the three alone would improve the objective function value. However, if we allow re-attachment operations over components in the SCC graph, then we would be able to re-attach the strong connectivity component containing the nodes l, m, and n in a single operation. Thus, the idea of our extended TNF algorithm is to allow re-attachment of both *nodes* and *components*. We term this algorithm Tree-Node-and-Component-Fix (TNCF).

There are many ways in which this intuition can be implemented and our TNCF algorithm uses one possible variant. In TNCF we first perform node re-attachment until convergence as in TNF (after initialization), but then compute the SCC graph and perform component re-attachment until convergence. Component re-attachment is identical to node re-attachment, except that we are guaranteed that the reduced graph is a forest. After performing component re-attachment until convergence, we again

Algorithm 3 Computing optimal re-attachment

Input: FRG $\mathcal{G} = (V, E)$, weighting function w, node $v \in V$
Output: optimal re-attachment of v

1: remove v and compute $G_{red} = (V_{red}, E_{red})$.
2: for all $c \in V_{red}$ in post-order compute $S_{v\text{-}in}(c)$ (Eq. 15)
3: for all $c \in V_{red}$ in pre-order compute $S_{v\text{-}out}(c)$ (Eq. 16)
4: case 1: $s_1 = \max_{c \in V_{red}} s_1(c)$ (Eq. 12)
5: case 2a: $s_2 = \max_{c \in V_{red}} s_2(c)$ (Eq. 13)
6: case 2b: compute s_3 (Eq. 14)
7: re-attach v according to $\max(s_1, s_2, s_3)$.

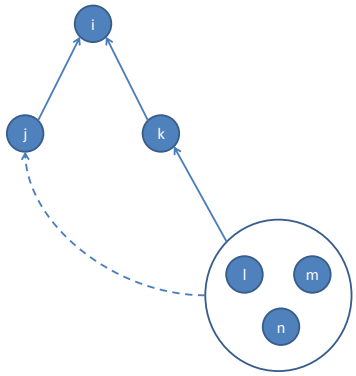

Figure 5
An example graph where single node re-attachment is a disadvantage.

perform node re-attachments and then component re-attachments and so on, until the entire process converges.

Graph-Node-Fix. The re-attachment strategy described above can be applied without assuming that the graph is an FRG. Next, we show Graph-Node-Fix (GNF), a similar algorithm that uses the re-attachment strategy to obtain an approximate solution for the ILP problem Max-Trans-Graph (3). In this more general case, finding the optimal re-attachment of a node v is done with an ILP solver. Nevertheless, this ILP is simpler than Equation (3), because we consider only candidate edges involving v. Figure 6 illustrates the three types of possible transitivity constraint violations when re-attaching v. The left side depicts a violation when $(i,k) \notin E$, expressed by the constraint in Equation (18), and the middle and right depict two violations when the edge $(i,k) \in E$, expressed by the constraints in Equation (19). Thus, the ILP is formulated by adding the following constraints to the objective function (9):

$$\max_{x(v)} \quad S_{v\text{-}in} + S_{v\text{-}out} \tag{17}$$

$$s.t. \qquad \forall_{i,k \in V \setminus \{v\}} \quad \text{if } (i,k) \notin E,\ x_{iv} + x_{vk} \leq 1 \tag{18}$$

$$\forall_{i,k \in V \setminus \{v\}} \quad \text{if } (i,k) \in E,\ x_{vi} \leq x_{vk},\ x_{kv} \leq x_{iv} \tag{19}$$

$$\forall_{i,k \in V \setminus \{v\}} \quad x_{iv}, x_{vk} \in \{0,1\}$$

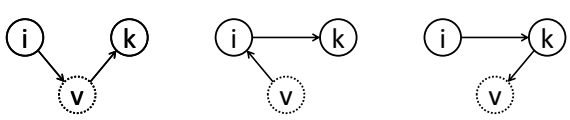

Figure 6
Three types of transitivity constraint violations, when re-attaching the node v to a graph containing the nodes i and k.

Complexity is exponential because of the ILP solver; however, the ILP size is reduced by an order of magnitude to $O(|V|)$ variables and $O(|V|^2)$ constraints. To summarize, the complexity of GNF is higher than the complexity of TNF, but it does not require the assumption that entailment graphs are FRGs.

4. Experimental Evaluation: Typed Entailment Graphs

In this and the next section we empirically evaluate the algorithms presented in Section 3 on two different data sets. Our first data set, presented in this section, comprises medium-sized graphs for which obtaining an exact solution is possible, and we demonstrate that our approximation methods substantially improve runtime while causing only a small degradation in performance compared with the optimal solution. The graphs are also particularly suited for global optimization because graph predicates are *typed*, which substantially reduces their ambiguity. The second data set (Section 5) contains a graph with tens of thousands of untyped nodes, where exact inference is completely impossible. We show that our methods scale to this graph and that transitivity improves performance even when predicates are untyped. The resulting graph contains more than 100,000 entailment rules that can be utilized in downstream semantic applications.

The input to the algorithms presented in Section 3 is a set of nodes V and a weighting function $w : V \times V \rightarrow \mathbb{R}$. We describe how those are constructed before presenting an experimental evaluation.

4.1 Typed Entailment Graphs

As mentioned in Section 2, one of the challenges in global optimization is that transitivity does not always hold when predicates are ambiguous. Schoenmackers et al. (2010) proposed an algorithm for learning inference rules between typed predicates, a representation that substantially reduces ambiguity. We adopt their representation and learn **typed entailment graphs**. A typed entailment graph (see Figure 7) is a directed graph where the nodes are typed predicates. A **typed predicate** is a triple (t_1, p, t_2), or simply $p(t_1, t_2)$, representing a predicate in natural language. p is the lexical realization of the predicate and the typed variables t_1, t_2 indicate that the arguments of the predicate belong to the semantic types t_1, t_2. Semantic types 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 (a_1, p, a_2), or simply $p(a_1, a_2)$, where $a_1 \in t_1$ and $a_2 \in t_2$

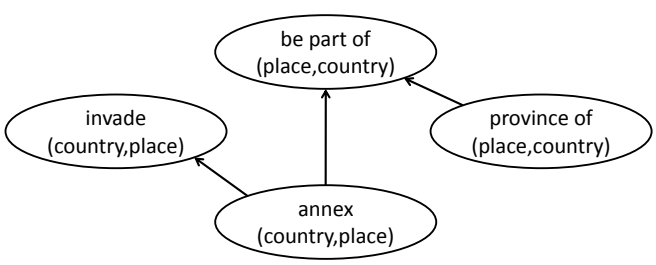

Figure 7
A fragment of a typed entailment graph.

are termed **arguments**. For example, *be common in(asthma,australia)* is an instance of *be common in*(DISEASE,PLACE).

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), as otherwise, the rule would contain unbound variables. Hence, given a set of typed predicates we can immediately decompose them into disjoint subsets—all typed predicates sharing the same pair of types define a separate graph that describes the entailment relations between those predicates (see Figure 7).

Edges in typed entailment graphs represent entailment rules in the usual way. 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). If t_1 and t_2 are equal we need to specify how arguments are mapped. This is done by splitting each node into two: For example, the node *beat*(TEAM,TEAM) is split into two typed predicates $beat(X_{team},Y_{team})$ and $beat(Y_{team},X_{team})$. This allows us to specify a rule where argument order is reversed such as $beat(X_{team},Y_{team}) \Rightarrow lose\ to(Y_{team},X_{team})$. This also accommodates rules such as $play(X_{team},Y_{team}) \Rightarrow play(Y_{team},X_{team})$: If team A plays team B, then team B plays team A.

To create typed entailment graphs, we used a data set that was generously provided by Schoenmackers et al. (2010). Schoenmackers et al. produced a mapping of 1.1 million arguments into 156 types (Examples for (*argument*, TYPE) pairs are (*exodus*, BOOK), (*china*, COUNTRY), and (*asthma*, DISEASE)), and then utilized the types, the mapped arguments, and 1 million TextRunner tuples (Banko et al. 2007) to generate a set of 10,672 typed predicates.[3] Because entailment can only occur between predicates that share the same types, we decomposed the 10,672 typed predicates into 2,303 typed entailment graphs. The largest graph contains 188 nodes and the total number of potential rules is 263,756.[4] The advantage of typing predicates is that it substantially reduces ambiguity, while still maintaining rules of wide applicability.

4.2 Training a Local Entailment Classifier

The weighting function w is derived from an entailment score provided by a local classifier. Given a local classifier that provides an entailment score s_{ij} for a pair of predicates (i, j), we define $w_{ij} = s_{ij} - \lambda$, where λ is a prior that controls graph sparseness: As λ increases, w_{ij} decreases and becomes negative for more pairs of predicates, rendering the graph more sparse. The probabilistic interpretation of λ is as follows. For each two nodes let q be a prior probability that an edge exists. For large values of q the graph tends to be dense, and vice versa. Defining λ as $\log \frac{1-q}{q}$, the modified weight function is:

$$w_{ij} = \log \frac{p(x_{ij} = 1)}{p(x_{ij} = 0)} - \log \frac{1-q}{q} = s_{ij} - \lambda \tag{20}$$

Given the weight function w, the task is to find the maximum a posteriori global graph that satisfies predefined constraints such as transitivity.

3 Readers are referred to their paper for details on mapping of tuples to typed predicates. The mapping of arguments into types can be downloaded from `http://www.cs.washington.edu/research/sherlock-hornclauses/`.

4 In more detail, 1,714 graphs have less than 5 nodes, 326 graphs have 5–10 nodes, 145 graphs have 10–20 nodes, 92 graphs have 20–50 nodes, 18 graphs have 50–100 nodes, and 7 graphs have at least 100 nodes.

Training is similar to the method proposed by Berant, Dagan, and Goldberger (2012), and we briefly describe it here. The input for training is a lexicographic resource, for which we use WordNet, and a set of tuples, for which we use the 1 million typed TextRunner tuples provided by Schoenmackers et al. We perform the following steps:

1. **Training set generation** Positive examples are generated using WordNet synonyms and hypernyms. Negative pairs are generated using WordNet direct co-hyponyms (sister terms), but we also utilize Word hyponyms at distance 2. In addition, we generate negative examples by randomly sampling pairs of typed predicates that share the same types. Table 1 provides an example for each type of automatically generated training example. It has been noted in the past that the WordNet verb hierarchy contains a certain amount of noise (Richens 2008; Roth and Frank 2012). However, we use WordNet only to generate examples for training a statistical classifier, and thus we can tolerate some noise in the generated examples. In fact, we have noticed that simple variants in training set generation do not result in substantial differences in classifier performance.

2. **Feature representation** Each example pair of typed predicates (p_1, p_2) is represented by a feature vector, where each feature is a distributional similarity score estimating whether p_1 entails p_2.

   We compute 11 distributional similarity scores for each pair of typed predicates, based on their arguments in the input set of tuples. The first six scores are computed by trying all combinations of two similarity functions *Lin* and *BInc* with three types of feature representations (see Section 2):

   (a) A feature is a pair of arguments. For example, a feature for the typed predicate *invade*(COUNTRY,CITY) might be *(germany,leningrad)* or *(england,paris)*.

   (b) Predicate representation is binary, and each typed predicate has two feature vectors, one for the *X* slot and one for the *Y* slot. Similarities are computed for each vector separately and are then combined by a geometric average (as in DIRT [Lin and Pantel 2001]). For example, the predicate *invade*(COUNTRY,CITY) will have a feature vector for its X slot with features such as *germany* and *england*, and a feature vector for its Y slot with features such as *leningrad* and *paris*.

   (c) Binary typed predicates are decomposed into two unary typed predicates, and similarity is computed separately for each unary

Table 1
Automatically generated training set examples.

Type	Example
direct hypernym	*beat*(TEAM,TEAM) $\Rightarrow$ *play*(TEAM,TEAM)
direct synonym	*reach*(TEAM,GAME) $\Rightarrow$ *arrive at*(TEAM,GAME)
direct cohyponym	*invade*(COUNTRY,CITY) $\not\Rightarrow$ *bomb*(COUNTRY,CITY)
hyponym (distance=2)	*defeat*(CITY,CITY) $\not\Rightarrow$ *eliminate*(CITY,CITY)
random	*hold*(PLACE,EVENT) $\not\Rightarrow$ *win*(PLACE,EVENT)

predicate. Then, similarity scores are combined by a geometric average. For example, the binary typed predicate *invade*(COUNTRY, CITY) will be decomposed into two unary predicates, one where the first argument of *invade* has the type COUNTRY (and the type of the second argument is unspecified), and another where the second argument of *invade* has the type CITY (and the first argument in unspecified).

The other five scores were provided by Schoenmackers et al. (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.

3. **Training** We sub-sample negative examples and train over an equal number of positive and negative examples. We used SVMperf (Joachims 2005) to train a Gaussian kernel classifier that provides an output score, s_{ij}. We tuned the two SVM parameters using 5-fold cross validation on a development set of two typed entailment graphs.

4. **Prior knowledge** For a small number of pairs of predicates, we might have prior knowledge of whether one entails the other. Berant, Dagan, and Goldberger (2012) integrated prior knowledge by adding hard constraints to the ILP. Because not all of our algorithms use an ILP solver, we integrate prior knowledge by modifying the local classifier score. For pairs of predicates i, j for which we have prior knowledge that i entails j (termed **positive local constraints**), we set $s_{ij} = \infty$. For pairs of predicates i, j for which we have prior knowledge that i does not entail j (termed **negative local constraints**), we set $s_{ij} = -\infty$.

   To generate prior knowledge constraints we normalized each predicate by omitting the first word if it is a modal and turned passives into actives. If two normalized predicates are equal, they are a positive local constraint. Negative local constraints were constructed from three sources (1) Predicates differing by a single pair of words that are WordNet antonyms, (2) Predicates differing by a single word of negation, and (3) Predicates $p(t_1, t_2)$ and $p(t_2, t_1)$ where p is a transitive verb (for example, *beat*) in VerbNet (Kipper, Dang, and Palmer 2000).

   As reported by Berant, Dagan, and Goldberger (2012), we find that injecting prior knowledge into the graph improves performance, as it provides a good starting point for the inference procedure.

4.3 Experimental Evaluation

To evaluate performance, we manually annotated all edges in 10 typed entailment graphs containing 14, 14, 22, 30, 53, 62, 76, 86, 118, and 118 nodes. This annotation yielded 3,427 edges and 35,585 non-edges, resulting in an empirical edge density of 9%. The data set is publicly available and can be downloaded from `http://www-nlp.stanford.edu/joberant/homepage_files/resources/Acl2011Exp.rar`.

We implemented the following algorithms for learning graph edges, where in all of them the graph is first decomposed into components as described in Section 3.1.

No-trans Use local scores without transitivity constraints—an edge (i, j) is inserted iff $w_{ij} > 0$, or in other words iff $s_{ij} > \lambda$.

Exact-graph Find the optimal solution for Max-Trans-Graph in each component by applying an ILP solver in a cutting-plane method, as described in Section 3.1.

Exact-forest Find the exact solution for Max-Trans-Forest (see Equation 8) in each component by applying an ILP solver in a cutting-plane method.

LP-relax Solve Max-Trans-Graph approximately by applying an LP-relaxation (see Section 2) on each graph component. We apply the LP solver within the same cutting-plane method to allow for a direct comparison. As mentioned, our goal is to present a method for learning transitive graphs, whereas LP-relax produces solutions that violate transitivity. However, we run it on our data set to obtain empirical results, and to compare runtimes against TNF.

Graph-Node-Fix (GNF) Initialize each component in the following way: If the graph is very sparse, that is, $\lambda \geq C$ for some constant C (set to 1 in our experiments), then solving the graph exactly is not an issue and we use Exact-graph. Otherwise, we initialize by applying Exact-graph in a sparse configuration, that is, $\lambda = C$.

Tree-Node-Fix (TNF) Initialize as in GNF, except that if it generates a graph that is not an FRG, it is corrected by a simple heuristic: For every node in the reduced graph $\mathcal{G}_{red}$ that has more than one parent, we choose from its current parents the single one whose SCC is composed of the largest number of nodes in $\mathcal{G}$.

We do not present the results of TNCF in this experiment, because for medium-sized graphs it provides results that are almost identical to TNF.

The No-trans baseline is a state-of-the-art local learning algorithm. It uses state-of-the-art local scores such as DIRT (Lin and Pantel 2001), BInc (Szpektor and Dagan 2008), Cover (Weeds and Weir 2003), and SR (Schoenmackers et al. 2010) and trains a classifier using these scores. Berant, Dagan, and Goldberger (2010) have demonstrated empiricially that training a classifier over multiple similarity scores improves performance compared with using just a single similarity score.

The Exact-graph algorithm is the state-of-the-art global method presented by Berant, Dagan, and Goldberger (2012). It finds the optimal solution for Max-Trans-Graph, and our goal is to show that we can obtain good performance more efficiently using our approximation methods. Last, LP-relax is a standard approximation method for solving ILPs (Martins, Smith, and Xing 2009).

We note that a trivial baseline would be to initialize edges based on whether $s_{ij} > \lambda$ and then compute the transitive closure in each component. We empirically found that this adds edges too aggressively and results in very low precision.

We use the Gurobi optimization package[5] as our ILP solver in all experiments. The experiments were run on a multi-core 2.5GHz server with 32GB of RAM.

We evaluate algorithms by comparing the set of gold-standard edges with the set of edges learned by each algorithm. We measure recall, precision, and F_1 for various values of the sparseness parameter λ, and compute the area under the precision-recall curve (AUC) generated. Efficiency is evaluated by comparing runtimes.

We first focus on runtimes and show that TNF is substantially more efficient than other baselines that use transitivity. Figure 8 compares runtimes of Exact-graph, GNF, TNF, and LP-relax as $-\lambda$ increases and the graph becomes denser. Note that the y-axis

5 www.gurobi.com

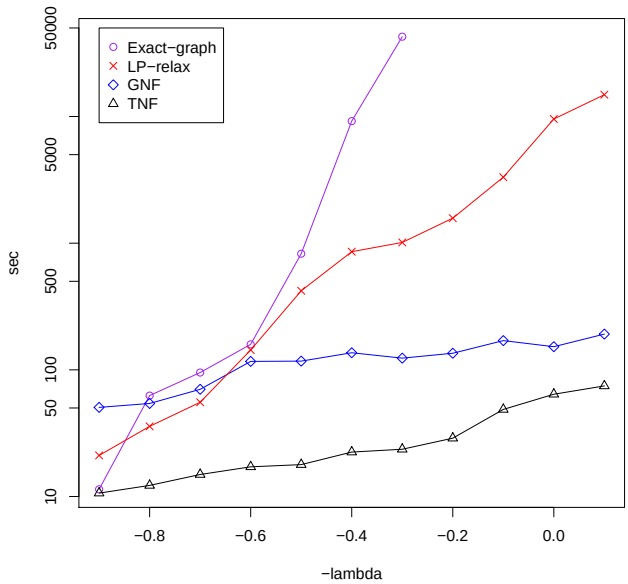

Figure 8
Runtime in seconds of the global algorithms for various $-\lambda$ values.

is in logarithmic scale. Clearly, Exact-graph is extremely slow and runtime increases quickly. For $\lambda = 0.3$, runtime was already 12 hours and we were unable to obtain results for $\lambda < 0.3$, whereas in TNF we easily got a solution for any λ. When $\lambda = 0.6$, where both Exact-graph and TNF achieve best F_1, TNF is 10 times faster than Exact-graph. When $\lambda = 0.5$, TNF is 50 times faster than Exact-graph, and so on. Most importantly, runtime for GNF and TNF increases much more slowly than for Exact-graph. Comparing runtimes for TNF and GNF, we see that the gap between the algorithms decreases as $-\lambda$ increases. However, for reasonable values of λ, TNF is about four to seven times faster than GNF, and we were unable to run GNF on large graphs, as we report in Section 5.

Runtime of LP-relax is also bad compared with TNF and GNF. Runtime increases more slowly than Exact-graph, but still very fast compared with TNF. When $\lambda = 0.6$, LP-relax is almost 10 times slower than TNF, and when $\lambda = -0.1$, LP-relax is 200 times slower than TNF. This points to the difficulty of scaling LP-relax to large graphs. Last, Exact-forest is the slowest algorithm and because it is an approximation of Exact-graph we omit if from the figure for clarity.

We now examine the quality of the learned graphs and validity of our modeling assumptions. Figure 9 (left) shows a pair-wise comparison of Exact-graph and Exact-forest. As is evident, the two curves are very similar, and the maximal F_1 on the curve and AUC are almost identical. This provides further support for our modeling assumption that entailment graphs are roughly forest-reducible. Figure 9 (right) shows a similar comparison for TNF and GNF. We observe again that the curves are similar and performance is almost identical (maximal F_1: 0.41, AUC: 0.31), illustrating that using the FRG assumption when using our approximation algorithm is empirically effective.

In Figure 10, we compare the performance of Exact-graph, which exactly solves Max-Trans-Graph, to our most efficient approximation algorithm, TNF, and to No-trans, a baseline that does not use transitivity at all (GNF and LP-relax are omitted from the

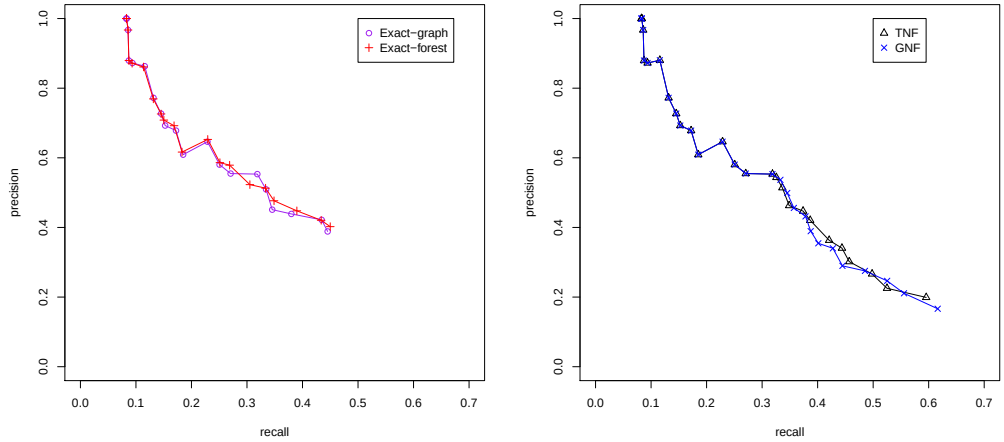

Figure 9
Pair-wise comparison of the precision-recall curves of Exact-graph vs. Exact-forest, and GNF vs. TNF.

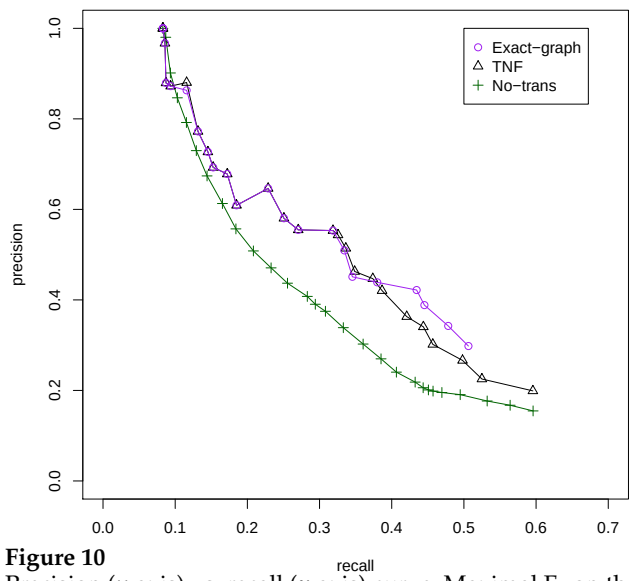

Figure 10
Precision (y-axis) vs. recall (x-axis) curve. Maximal F_1 on the curve is 0.43 for Exact-graph, 0.41 for TNF, and 0.34 for No-trans. AUC in the recall range 0–0.5 is 0.32 for Exact-graph, 0.31 for TNF, and 0.26 for No-trans.

figure to improve readability). We observe that both Exact-graph and TNF substantially outperform No-trans and that TNF's graph quality is only slightly lower than Exact-graph (which is extremely slow). We report in the caption the maximal F_1 on the curve and AUC in the recall range 0–0.5 (the widest range for which we have results for all algorithms). Note that compared with Exact-graph, TNF reduces AUC by merely a point and the maximal F_1 score by two points only.

As for LP-relax, results are just slightly lower than Exact-graph (maximal F_1: 0.43, AUC: 0.32), but its output is not a transitive graph, and as shown above runtime is quite slow.

To summarize, in this section we have empirically evaluated the algorithms presented in Section 3 on medium-sized graphs for which we can find an optimal solution. Our main findings are as follows:

1. Finding the optimal solution for Exact-graph and Exact-forest results in very similar graphs. This supports our assumption that entailment graphs are approximately forest-reducible.
2. Running GNF and TNF also yields very similar graphs, illustrating that using the FRG assumption when using our re-attachment approximation scheme is effective.
3. Our efficient approximation algorithm, TNF, is substantially faster compared with running an ILP solver that exactly solves Max-Trans-Graph.
4. TNF results in only a slight decrease in quality of learned graphs compared with Exact-graph.
5. TNF, which respects the transitivity constraint, learns graphs of higher quality compared with the local baseline, No-trans.

These findings lead us to believe that the algorithms presented in Section 3 can scale to large entailment graphs. In the next section, we empirically evaluate on a much larger data set for which using ILP solvers is impractical.

5. Experimental Evaluation: Untyped Entailment Graphs

In this section we evaluate our methods on a graph with 20,000 nodes. Again, we describe how the nodes V and the weighting function w are constructed.

5.1 Untyped Entailment Graphs

The nodes of the entailment graph we learn in this section are not typed. Although ambiguity is a problem in this setting, we will show that nevertheless transitivity constraints can improve results compared with a state-of-the-art local entailment classifier.

The nodes of the graph were generated from a set of two billion tuples of the form *(*arg_1*,predicate,*arg_2*)* that were extracted by the Reverb open information extraction system (Fader, Soderland, and Etzioni 2011) over the Clueweb09[6] data set and were generously provided by the authors of Reverb.

We performed some simple preprocessing over the extracted tuples. Each tuple is accompanied by a confidence value and we discarded tuples with confidence smaller than 0.5. Next, we normalized predicates using a procedure that omits unnecessary content such as modals and adverbs. The entire normalization procedure is publicly available as part of the Reverb project.[7] Last, we normalized arguments by replacing pronouns and determiners by the tokens *PRONOUN* and *DET*. We also ran the BIU

6 http://lemurproject.org/clueweb09.php/.
7 https://github.com/knowitall/reverb-core/blob/master/core/src/main/java/edu/washington/cs/knowitall/normalization/RelationString.java.

number normalizer[8] and replaced numbers larger than one by the token *NUM*. After these three steps, we are left with a total of 960 million tuples of which 291 million are distinct.

The number of distinct predicates in the set of extracted tuples is 103,315. Because this is still a large number, we restrict the predicate set to the 10,000 predicates that appear with the highest number of pairs of arguments. As previously explained, each predicate is split into two (for example, *defeat* is split into *X defeat Y* and *Y defeat X*), and so the final number of entailment graph nodes is 20,000.

5.2 Training a Local Entailment Classifier

As with typed entailment graphs, the weighting function w is obtained by training a classifier that provides a score s_{ij} for all pairs of predicates[9] and defining $w_{ij} = s_{ij} - \lambda$. This setting, computing the scores s_{ij}, involves the following steps:

1. **Training set generation.** We use the data set released by Zeichner, Berant, and Dagan (2012), which contains 6,567 entailment rule applications annotated for their validity by crowdsourcing. For example, the data set marks that *The exercises alleviate pain* ⇒ *The exercises help ease pain* is a valid rule application, whereas *Obama want to boost the defense budget* ⇒ *Obama increase the defense budget* is an invalid rule application. We extract a single rule from each rule application, for example, from the rule application *The exercises alleviate pain* ⇒ *The exercises help ease pain* we extract the rule *X alleviate Y* ⇒ *X help ease Y*. We use half of the data set for training, resulting in 1,224 positive examples and 2,060 negative examples. Another two training examples are *X unable to pay Y* ⇒ *X owe Y* and *X own Y* ⇏ *Y be sold to X*.

2. **Feature representation.** Each pair of predicates (p_1, p_2) is represented by a feature vector where the first six are distributional similarity features identical to the first six features described in Section 4.2. In addition, for pairs of predicates for which at least one distributional similarity feature is non-zero, we add lexicographic features computed from WordNet (Fellbaum 1998), VerbOcean (Chklovski and Pantel 2004), and CatVar (Habash and Dorr 2003), as well as string-similarity features. Table 2 provides the exact details of these features. A feature is computed for a pair of predicates (p_1, p_2) where in this context a predicate is a pair *(pred,rev)*: *pred* is the lexical realization of the predicate, and *rev* is a Boolean indicating whether arg_1 is *X* and arg_2 is *Y* or vice versa. Overall, each pair of predicates is represented by 27 features.

3. **Training.** After obtaining a feature representation for every pair of predicates, we train a Gaussian kernel SVM classifier that optimizes F_1 (SVMperf implementation [Joachims 2005]), and tune the parameters C and γ by a grid search combined with 5-fold cross validation. We use the trained local classifier to compute a score s_{ij} for all pairs of predicates.

8 `http://u.cs.biu.ac.il/~nlp/downloads/normalizer.html`.

9 We do not need to run the classifier on $10,000^2$ pairs of predicates, because for the majority of predicate pairs the features are all zeros, and for this set of pairs we can run the classifier only once.

Table 2
Definition of lexicographic and string-similarity features. We denote by the string *pred* the lexical realization of the predicate, and by the Boolean indicator *rev* whether arg_1 is X and arg_2 is Y, or vice versa. Normalized edit-distance is edit-distance divide by the sum of lengths of pred_1 and pred_2.

Name	Type	Source	Description
synonym	binary	WordNet	$rev_1 = rev_2 \wedge$ $pred_1$ is a synonym of $pred_2$.
loose synonym	binary	WordNet	$rev_1 = rev_2 \wedge$ $pred_1$ and $pred_2$ are identical except for a pair of words (w_1, w_2) where w_1 is a synonym of w_2.
hypernym	binary	WordNet	$rev_1 = rev_2 \wedge$ $pred_1$ is a hypernym of $pred_2$ at distance ≤ 2.
loose hypernym	binary	WordNet	$rev_1 = rev_2 \wedge$ $pred_1$ and $pred_2$ are identical except for a pair of words (w_1, w_2) where w_1 is a hypernym of w_2 at distance ≤ 2.
hyponym	binary	WordNet	$rev_1 = rev_2 \wedge$ $pred_1$ is a hyponym of $pred_2$ at distance ≤ 2.
loose hyponym	binary	WordNet	$rev_1 = rev_2 \wedge$ $pred_1$ and $pred_2$ are identical except for a pair of words (w_1, w_2) where w_1 is a hyponym of w_2 at distance ≤ 2.
co-hyponym	binary	WordNet	$rev_1 = rev_2 \wedge$ $pred_1$ is a co-hyponym of $pred_2$ at distance ≤ 2.
loose co-hyponym	binary	WordNet	$rev_1 = rev_2 \wedge$ $pred_1$ and $pred_2$ are identical except for a pair of words (w_1, w_2) where w_1 is a co-hyponym of w_2 at distance ≤ 2.
entailment	binary	WordNet	$rev_1 = rev_2 \wedge$ $pred_1$ verb-entails $pred_2$ (distance ≤ 1).
loose entailment	binary	WordNet	$rev_1 = rev_2 \wedge$ $pred_1$ and $pred_2$ are identical except for a pair of words (w_1, w_2) where w_1 verb-entails w_2 (distance ≤ 1).
stronger	binary	VerbOcean	$rev_1 = rev_2 \wedge$ $pred_1$ is stronger-than $pred_2$.
loose stronger	binary	VerbOcean	$rev_1 = rev_2 \wedge$ $pred_1$ and $pred_2$ are identical except for a pair of words (w_1, w_2) where w_1 is stronger-than w_2.
rev-stronger	binary	VerbOcean	$rev_1 = rev_2 \wedge$ $pred_2$ is stronger-than $pred_1$.
loose rev-stronger	binary	VerbOcean	$rev_1 = rev_2 \wedge$ $pred_1$ and $pred_2$ are identical except for a pair of words (w_1, w_2) where w_2 is stronger-than w_1.
CatVar	binary	CatVar	$pred_1$ and $pred_2$ contain a pair of content words (w_1, w_2) that are either identical or derivationally-related in CatVar.
remove word	binary	String	$rev_1 = rev_2 \wedge$ removing a single word from $pred_1$ will result in $pred_2$.
add word	binary	String	$rev_1 = rev_2 \wedge$ adding a single word to $pred_1$ will result in $pred_2$.
remove adj.	binary	String	$rev_1 = rev_2 \wedge$ removing a single adjective from $pred_1$ will result in $pred_2$.
add adj.	binary	String	$rev_1 = rev_2 \wedge$ adding a single adjective to $pred_1$ will result in $pred_2$.
Edit	real	String	if $rev_1 = rev_2$, the normalized edit-distance between $pred_1$ and $pred_2$, otherwise 1.
Reverse	binary	String	$rev_1 = rev_2$.

4. **Prior knowledge.** We automatically generate local constraints for pairs of predicates for which we know with high certainty whether the first entails the second or not. We define and compute constraints over pairs of predicates (p_1, p_2), where again a predicate p is a pair $(pred, rev)$. We start with negative local constraints (Examples in Table 3):

   (a) *cousin:* (p_1, p_2) are cousins if $rev_1 = rev_2$ and $pred_1 = pred_2$, except for a single pair of words w_1 and w_2, which are cousins in WordNet—that is, they have a common hypernym at distance 2.

   (b) *indirect hypernym:* (p_1, p_2) are indirect hypernyms if $rev_1 = rev_2$ and $pred_1 = pred_2$, except for a single pair of words w_1 and w_2, and w_1 is a hypernym at distance 2 of w_2 in WordNet.

Table 3
Examples for negative local constraints. The column ⇒/⇔ indicates whether the constraint is directional or symmetric.

Name	⇒/⇔	Example
cousin	⇔	*X beat Y* ⇏ *X fix Y*
indirect hypernym	⇒	*X give all kind of* ⇏ *X sell all kind of Y*
antonym	⇔	*X announce death of* ⇏ *X announce birth of Y*
uncertainty implication	⇒	*X want to analyze Y* ⇏ *X analyze Y*
negation	⇔	*X do no harm to Y* ⇏ *X do harm to Y*
transitive opposite	⇔	*X abandon Y* ⇏ *Y abandon X*

(c) *antonym:* (p_1, p_2) are antonyms if $rev_1 = rev_2$ and $pred_1 = pred_2$, except for a single pair of words w_1 and w_2, and w_1 is an antonym of w_2 in WordNet.

(d) *uncertainty implication:* holds for (p_1, p_2) if $rev_1 = rev_2$ and concatenating the words *want to* to $pred_2$ results in $pred_1$.

(e) *negation:* Negation holds for (p_1, p_2) and (p_2, p_1), if $rev_1 = rev_2$ and in addition removing a single negation word (*not*, *no*, *never*, or *nt*) from $pred_1$ results in $pred_2$. Negation also holds for (p_1, p_2) and (p_2, p_1), if $rev_1 = rev_2$ and in addition replacing in $pred_1$ the word *no* for the word *a* results in $pred_2$.

(f) *transitive opposites:* (p_1, p_2) are transitive opposites if $pred_1 = pred_2$ and $rev_1 \neq rev_2$ and $pred_1$ is a transitive verb in VerbNet.

Next, we define the positive local constraints (Examples in Table 4):

(a) *determiner:* The determiner constraint holds for (p_1, p_2) and (p_2, p_1) if $rev_1 = rev_2$ and omitting a determiner (*a* or *the*) from $pred_1$ results in $pred_2$.

(b) *positive implication:* Positive implication holds for (p_1, p_2) if $rev_1 = rev_2$ and concatenating the words *manage to* or *start to* or *start* or *decide to* or *begin to* to $pred_2$ results in $pred_1$.

Table 4
Examples for positive local constraints. The column ⇒/⇔ indicates whether the constraint is directional or symmetric.

Name	⇒/⇔	Example
determiner	⇔	*X be the answer to Y* ⇒ *X be answer to Y*
positive implication	⇒	*X start to doubt Y* ⇒ *X doubt Y*
passive-active	⇔	*X command Y* ⇒ *Y be commanded by X*

(c) *passive-active:* The passive-active constraint holds for (p_1, p_2) and (p_2, p_1) if p_2 is the passive form of p_1.

Again, local constraints are integrated into our model by changing the score $s_{ij} = \infty$ for positive local constraints and $s_{ij} = -\infty$ for negative local constraints.

5.3 Experimental Evaluation

To evaluate performance we use the second half of the data set released by Zeichner, Berant, and Dagan (2012) as a test set. This test set contains 3,283 annotated examples, where 1,734 are covered by the 10,000 nodes of our graph (649 positive examples and 1,085 negative examples). As usual, for each algorithm we compute recall and precision with respect to the gold standard at various points by varying the sparseness parameter λ.

Most methods used over typed graphs in Section 4 (*Exact-graph*, *Exact-forest*, *LP-relax*, and *GNF*) are intractable because they require an ILP solver and our graph does not decompose into components that are small enough. Therefore, we compare the *No-trans* local baseline, where transitivity is not used, with the efficient global methods TNF and TNCF. Recall that in Section 4 we initialized TNF by applying an ILP solver in a sparse configuration on each graph component. In this experiment the graph is too large to use an ILP solver and so we initialize TNF and TNCF with a simple heuristic we describe next.

We begin with an empty graph and first sort all pairs of predicates (i, j) for which $w_{ij} > 0$, according to their weight w. Then, we go over predicate pairs one-by-one and perform two operations. First, we verify that inserting (i, j) into the graph does not violate the FRG assumption. This is done by going over all edges $(i, k) \in E$ and checking that for every k either $(j, k) \in E$ or $(k, j) \in E$. If this is the case, then the edge (i, j) is a valid candidate edge as the resulting reduced graph will remain a directed forest, otherwise adding it will result in i having two parents in the reduced graph, and we do not insert this edge. Second, we compute the **transitive closure** T_{ij}, which contains all node pairs that must be edges in the graph in case we insert (i, j), due to transitivity. We compute the change in the value of the objective function if we were to insert T_{ij} into the graph. If this change improves the objective function value, we insert T_{ij} into the graph (and so the value of the objective function increases monotonically). We keep going over the candidate edges from highest score to lowest until the objective function can no longer be improved by inserting a candidate edge. We term this initialization *HTL-FRG*, because we scan edges from high-score to low-score and maintain an FRG. We add this initialization procedure as another baseline.

Figure 11 presents the precision-recall curve of all global algorithms compared with the local classifier for $0.05 \leq \lambda \leq 2.5$. One evident property is that No-trans reaches higher recall values than the global algorithms, which means that adding a global transitivity constraint prevents adding correct edges that have positive weight, since this would cause the addition of many other edges that have negative weight. A possible reason for that is predicate ambiguity, where positive weight edges connect connectivity components through an ambiguous predicate. This suggests that a natural direction for future research is to develop algorithms that do not impose transitivity constraints for rules $i \Rightarrow j$ and $j \Rightarrow k$, if each rule refers to a different meaning of j. One possible direction is to learn latent "sense" or "topic" variables for each rule (as suggested by Melamud et al. Melamud et al. [2013]). Then, we can use the methods presented in this article

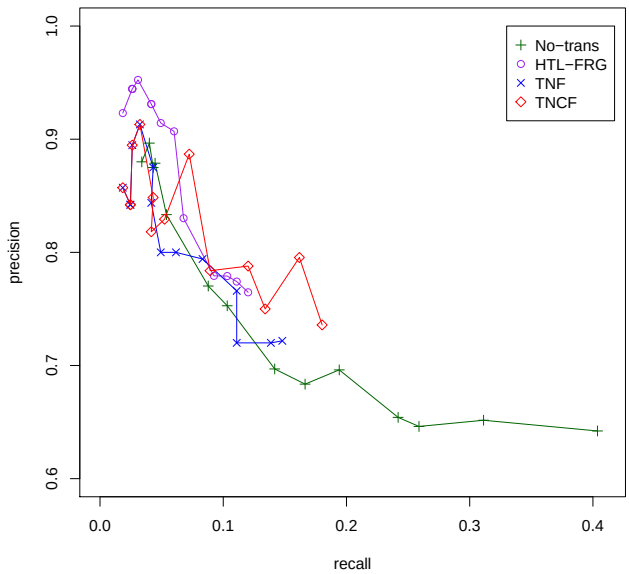

Figure 11
Precision-recall curve comparing global algorithms to the local algorithm.

to impose transitivity constraints for each sense separately, and easily allow violations of transitivity across different senses. We provide concrete examples for such cases of ambiguity in our qualitative analysis in Section 5.4.

Although global algorithms are limited in their recall, for recall values of 0.1–0.2 they substantially improve precision over No-trans. To better observe this result, Table 5 presents the results for the recall range 0.1–0.25. Comparing TNCF and No-trans shows that the TNCF algorithm improves over No-trans by 5–10 precision points:

Table 5
Recall, precision, F_1, and the number of learned rules for No-trans and several global algorithms for parameters of λ for which recall is in the range 0.1–0.25.

method	λ	recall	precision	F_1	# of rules
No-trans	1	0.1	0.75	0.18	39,872
No-trans	0.8	0.14	0.7	0.24	57,276
No-trans	0.6	0.17	0.68	0.27	65,232
No-trans	0.4	0.19	0.7	0.3	77,936
No-trans	0.2	0.24	0.65	0.35	116,832
HTL-FRG	0.2	0.09	0.78	0.17	48,986
HTL-FRG	0.15	0.1	0.78	0.18	55,756
HTL-FRG	0.1	0.11	0.77	0.19	65,802
HTL-FRG	0.05	0.12	0.76	0.21	84,380
TNF	0.2	0.11	0.77	0.19	61,186
TNF	0.15	0.11	0.72	0.19	71,674
TNF	0.1	0.14	0.72	0.23	88,304
TNF	0.05	0.15	0.72	0.25	127,826
TNCF	0.2	0.12	0.79	0.21	66,240
TNCF	0.15	0.13	0.75	0.23	80,450
TNCF	0.1	0.16	0.8	0.27	102,565
TNCF	0.05	0.18	0.74	0.29	156,208

In the recall range of 0.1–.2, the precision of No-trans is 0.68–0.75, whereas the precision of TNCF is 0.74–0.8. The number of rules learned by TNCF in this recall range is about 100,000.

We use HTL-FRG as an initialization method for TNCF, which on its own is not a very good approximation algorithm. Comparing HTL-FRG with No-trans, we observe that it is unable to reach high recall values and it only marginally improves precision compared with No-trans. Applying TNF over HTL-FRG also provides disappointing results—it increases recall compared with HTL-FRG, but precision is not better than No-trans. Indeed, it seems that although performing node re-attachments monotonically improves the objective function value, it is not powerful enough to learn a graph that is better with respect to our gold standard. On the other hand, adding component re-attachments (TNCF) allows the algorithm to better explore the space of graphs and learn a graph with higher recall and precision than HTL-FRG.

Figure 12 presents the runtime for HTL-FRG, TNF, and TNCF (y-axis in logarithmic scale), all of which yield an FRG and can be run on a large untyped graph. Again, we were unable to obtain results with Exact-graph, Exact-forest, LP-relax, and GNF, because using an ILP solver on a graph with 20,000 nodes was impossible. As expected, HTL-FRG is much faster than both TNF and TNCF, and TNCF is somewhat slower than TNF. However, as mentioned earlier, TNCF is able to improve both precision and recall compared with TNF and HTL-FRG. Note that when $\lambda = 1.2$, runtime increases suddenly for both TNF and TNCF. This is because at this point a large connected component is formed, as we explain next.

Figure 13 presents the results of graph decomposition as described in Section 3.1. Evidently, when $\lambda = 0$ the largest component constitutes almost all of the graph, which contains 20,000 nodes. Note that when $\lambda = 1.2$, the size of the largest component increases suddenly to more than half of the graph nodes. Additionally, TNCF obtained recall of 0.1–0.2 when $\lambda \leq 0.2$, and in this case the number of nodes in the largest

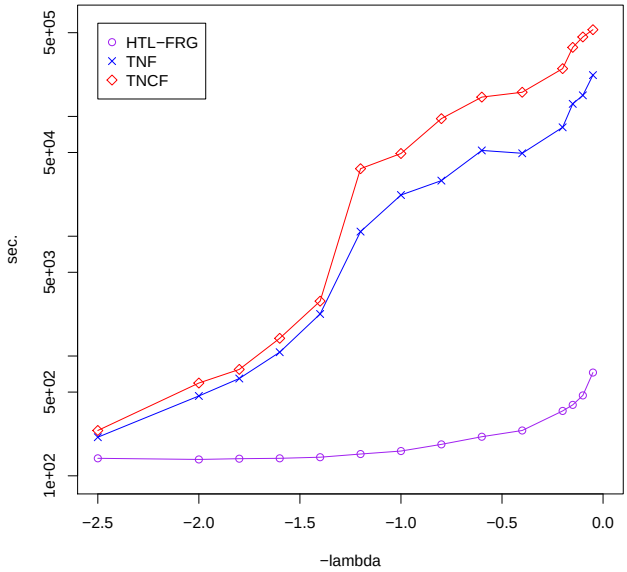

Figure 12
Runtime in seconds for various $-\lambda$ values for global algorithms.

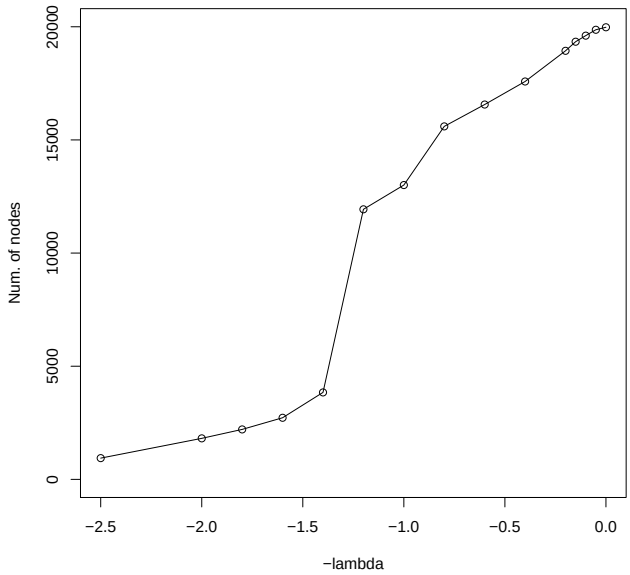

Figure 13
The number of nodes in the largest component as a function of the sparseness prior $-\lambda$.

component is 90% of the total number of graph nodes. Thus, contrary to the typed graphs, untyped graphs that contain ambiguous predicates do not decompose well into small components.

To summarize, we demonstrated that our efficient global methods scale to a graph with 20,000 nodes and observed that even when predicates are ambiguous we are able to improve precision for moderate values of recall. Nevertheless, there are indications that our modeling assumptions are violated when applied over a large graph with ambiguous predicates. Transitivity and FRG constraints limit the recall we are able to reach, and the graph does not decompose very well into components. Thus, in future work we would like to apply our algorithms over graphs where predicate ambiguity is modeled and so transitivity constraints can be properly applied.

5.4 Qualitative Analysis

In the previous section we quantitatively evaluated global methods for learning entailment graphs over a large set of untyped predicates. Our results revealed interesting issues that we would like to further investigate:

1. Why do methods that assume entailment graphs are transitive and forest-reducible have limited recall?
2. How much does node re-attachment affect graph structure?

In this section, we try to answer these questions by performing a qualitative analysis that allows us to gain better insight on the behavior of our algorithms.

Question 1: Transitivity and FRG assumptions. We would like to understand why using global algorithms such as TNCF results in limited recall. Our hypothesis is that because predicates in the graph are untyped and potentially ambiguous, violations of the

assumptions are possible, which results in recall reduction. To examine this hypothesis, we perform two qualitative analyses:

1. We analyze graphs containing only gold-standard edges, and manually identify cases where the modeling assumptions are violated.
2. We compare the local algorithm, No-trans, to the global algorithm, TNCF. If our hypothesis is correct, then we expect TNCF to remove correct edges that are inserted by No-trans due to ambiguous predicates.

We start by constructing a graph with all edges from our gold-standard data set and search for transitivity and FRG violations caused by ambiguous predicates. Note that the manually annotated gold-standard edges are only a subset of the full set of correct edges, which is actually much larger. Thus, we cannot automatically identify violations, because the graph is missing many true edges. Instead, we manually go over candidate violations and check if indeed this case is a result of a violation of our modeling assumptions, and if so then our global algorithm cannot predict the correct set of edges.

One example is the pair of rules *seek* $\Rightarrow$ *apply for* and *apply for* $\Rightarrow$ *file for*. The first rule was annotated in the rule application *Others seek medical care* $\Rightarrow$ *Others apply for medical care* (Recall that crowdsourcing was used to annotate rule applications, and rules were extracted from these applications.) The second rule was extracted from the rule application *Students apply for more than one award* $\Rightarrow$ *Student file for more than one award*. This causes a transitivity violation because *Students seek more than one award* $\not\Rightarrow$ *Student file for more than one award*. Evidently, the meaning of the predicate *apply for* is context-dependent. Another example is the pair of rules *contain* $\Rightarrow$ *supply* and *supply* $\Rightarrow$ *serve* annotated in the applications *The page contains links* $\Rightarrow$ *The page supplies links* and *Users supply information or material* $\Rightarrow$ *Users serve information or material*. Clearly, *contain* $\not\Rightarrow$ *serve* and the transitivity violation is caused by the fact that the predicate *supply* is context-dependent—in the first context the subject is inanimate, whereas in the second context the subject is human.

A good example for an FRG violation is the predicate *come from*. The following three entailment rules are annotated by the turkers: *come from* $\Rightarrow$ *be raised in*, *come from* $\Rightarrow$ *be derived from*, and *come from* $\Rightarrow$ *come out of*. These correspond to the following rule applications: *The Messiah comes from Judah* $\Rightarrow$ *The Messiah was raised in Judah*, *The colors come from the sun light* $\Rightarrow$ *The colors are derived from the sun light*, and *The truth comes from the book* $\Rightarrow$ *The truth comes out of the book*. Clearly, *be raised in* $\not\Rightarrow$ *be derived from* and *be derived from* $\not\Rightarrow$ *be raised in*, so this is an FRG violation. Indeed, the three applications correspond to different meanings of the predicate *come from*, which depend on context. A second example is the pair of rules *be dedicated to* $\Rightarrow$ *be committed to* and *be dedicated to* $\Rightarrow$ *contain information on*. Again, this is an FRG violation because *be committed to* $\not\Rightarrow$ *contain information on* and *contain information on* $\not\Rightarrow$ *be committed to*. This violation occurs because of the ambiguity of the predicate *be dedicated to*, which can be resolved by knowing whether the subject is human or is an object that carries information (for example, *The web site is dedicated to the life of lizards*).

This first analysis revealed particular cases where the transitivity and FRG assumptions do not hold. In the next analysis, we would like to directly compare cases where global and local methods disagree with one another. We hypothesize that because predicates are not disambiguated, then global algorithms will delete correct edges due

to an ambiguous predicate. We set the sparseness parameter $\lambda = 0.2$ and compare the set of edges learned by No-trans to the set of edges learned by TNCF.

Examining the disagreements, we observe that, as expected, in 90% of the cases TNCF deletes an edge that was inserted by No-trans. We divide these deleted edges into two categories in the following manner: We compute the weakly connected components of the graph learned by TNCF,[10] and then for each edge inserted by No-trans and deleted by TNCF we check whether it connects two different weakly connected components or not. There are two reasons for focusing on this property of deleted edges. First, we hypothesize that most deleted edges will connect two different connected components, because such edges result in many violations of transitivity. Second, weakly connected components usually correspond to separate semantic meanings, and thus a correct edge that connects two weakly connected components probably involves an ambiguous predicate.

Indeed, out of all edges where No-trans inserts an edge and TNCF does not, 76.4% connect weakly connected components in the global graph. This proves that deleting such edges is the main cause for disagreement between TNCF and No-trans. Now, we can take a closer look at these edges, and check whether indeed when a correct edge is deleted, it is because of an ambiguous predicate.

We randomly sample five cases where TNCF erroneously deleted an edge connecting weakly connected components—these are cases where ambiguity is likely to occur. For comparison, we also sample five cases where TNCF was right and No-trans erred. Table 6 shows the samples where No-trans is correct. The first column describes the rule and the second column specifies the score s_{ij} provided by the local classifier No-trans. The last two columns detail two examples for predicates that are in the same weakly connected component with the rule LHS and RHS in the graph learned by TNCF. These two columns provide a flavor for the overall meaning of predicates that belong to this component. Table 7 is equivalent and shows the samples where TNCF is correct.

Example 1 in Table 6 already demonstrates the problem of ambiguity. The LHS for this rule application in the crowdsourcing experiment was *your parents turn off comments* and indeed in this case the RHS *your parents cut off comments* is inferred. However, we can see that the LHS component revolves around the *turning off* or *shutting off* of appliances, for example, whereas the RHS component deals with a more explicit and physical meaning of *cutting*. This is because the predicate *X cut off Y* is ambiguous and consequently TNCF omits this correct edge. Example 5 also demonstrates well the problem of ambiguity—the LHS of this rule application is *This site was put by fans*, which in this context entails the RHS *This site was run by fans*. However, the more common sense of *be put by* does not entail the predicate *be run by*; this sense is captured by the predicates in the LHS component *Y help put X* and *X be placed by Y*. Example 4 is another good example where the ambiguous predicate is *X be open to Y*—the RHS component is concerned with the physical state of being opened, whereas the LHS component has a more abstract sense of *availability*. The LHS of the rule application in this case was *The services are available to museums*. Examples 2 and 3 are cases where the problem is not in predicate ambiguity but in the connected component itself. Thus, out of five randomly sampled examples where TNCF deleted an edge that connects two weakly connected components, three can be attributed to predicate ambiguity.

10 A weekly connected component in a directed graph is a set of nodes S such that for every pair of nodes $u, v \in S$ there is an *undirected* path from u to v and from v to u, that is, there is a path when edge directions are ignored.

Table 6
Correct edges inserted by No-trans and omitted by TNCF that connect weakly connected components in TNCF. An explanation for each column is provided in the body of the section.

Example	**Rule**	s_{ij}	**LHS component**	**RHS component**
1	*X turn off Y ⇒ X cut off Y*	1.32	*X shut off Y, X cut on Y*	*X chop Y, X slice Y*
2	*X be arrested in Y ⇒ X be captured in Y*	0.27	*X be killed on Y, X be arrested on Y*	*X be delivered in Y, X be provided in Y*
3	*X die of Y ⇒ X be diagnosed with Y*	0.88	*X gain on Y, X run on Y*	*X be treated for Y, X have symptom of Y*
4	*X be available to Y ⇒ X be open to Y*	0.27	*X be offered to Y, X be provided to Y*	*X open for Y, X open up for Y*
5	*X be put by Y ⇒ X be run by Y*	0.41	*Y help put X, X be placed by Y*	*Y operate X, Y control X*

Table 7
Erroneous edges inserted by No-trans and omitted by TNCF that connect weakly connected components in TNCF. An explanation for each column is provided in the body of the section.

Example	**Rule**	s_{ij}	**LHS component**	**RHS component**
1	*X want to see Y ⇒ X want to help Y*	0.83	*X need to visit Y, X need to see Y*	*X be a supporter of Y, X need to support Y*
2	*X cut Y ⇒ X fix Y*	0.88	*X chop Y, X slice Y*	*X cure Y, X mend Y*
3	*X share Y ⇒ X understand Y*	0.62	*X partake in Y, Y be shared by X*	*Y be recognized by X, X realize Y*
4	*X build Y ⇒ X rebuild Y*	1.42	*X construct Y, Y be manufactured by X*	*X regenerate Y, X restore Y*
5	*X measure Y ⇒ X weigh Y*	0.65	*X quantify Y, X be a measure of Y*	*X count for Y, X appear Y*

Table 7 demonstrates cases where TNCF correctly omitted an edge and consequently decomposed a weakly connected component into two. These are classical cases where the global constraint of transitivity helps the algorithm avoid errors. In Example 1, although $s_{ij} = 0.83$ (which is higher than $\lambda = 0.2$), the edge is not inserted because this would cause the addition of wrong edges from the LHS component that deals with *seeing* and *visiting* to the RHS component, which is concerned with *help* and *support*. Example 2 is a case of a pair of predicates that are *co-hyponyms* or even perhaps *antonyms*. Transitivity helps TNCF avoid this erroneous rule.

To conclude, our analysis reveals that applying structural constraints encourages global algorithms to omit edges. Although this is often desirable, it can also prevent correct rules from being discovered because of problems of ambiguity. Thus, as discussed in Section 5.3, we believe that an important direction for future research is to apply our global graph learning methods over context-sensitive representations (such as the one presented by Melamud et al. [2013]).

Question 2: Node re-attachment. We would like to understand the effect of applying TNF over the entailment graph, or more precisely, check whether TNF substantially alters the set of graph edges compared with an initialization with HTL-FRG. First, we run both

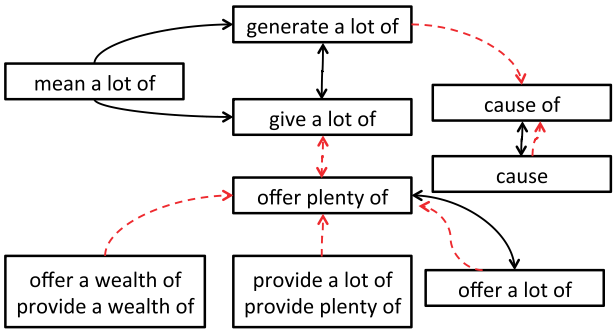

Figure 14
A fragment from the graphs learned by HTL-FRG (black solid edges) and TNF (red dashed edges).

HTL-FRG and TNF with sparseness parameter $\lambda = 0.2$ and compute the proportion of edges that is in their intersection. HTL-FRG learns 48,986 edges and TNF learns 61,186 edges. The number of edges in the intersection is 35,636. This means that 27% of the edges learned by HTL-FRG were removed (13,350 edges) and in addition 25,550 edges were added into the graph. Clearly, this indicates that TNF changes the learned graph substantially.

We exemplify this with a few fragments of graphs learned by HTL-FRG and TNF. Figure 14 shows 11 predicates,[11] where the black solid edges correspond to HTL-FRG edges and red dashed edges correspond to TNF edges. Nodes that contain more than a single predicate represent an agreement between TNF and HTL-FRG that all predicates in the node entail one another. Clearly, TNF substantially changes the initialization generated by HTL-FRG. HTL-FRG creates an erroneous component in which *give a lot of* and *generate a lot of* are synonymous and *mean a lot of* entails them. TNF disassembles this component and determines that *give a lot of* is equivalent to *offer plenty of*, while *generate a lot of* entails *cause of*. Moreover, TNF disconnects the predicate *mean a lot of* completely. TNF also identifies that the predicates *offer a wealth of*, *provide a wealth of*, *provide a lot of*, and *provide plenty of* entail *offer plenty of*. Of course, TNF sometimes causes errors—for example, HTL-FRG decided that *cause* and *cause of* are equivalent, but TNF deleted the correct entailment rule *cause of* $\Rightarrow$ *cause*.

Figure 15 provides another interesting example for an improvement in the graph due to the application of TNF. In the initialization, HTL-FRG determined that the predicate *encrypt* entails the predicates *convert* and *convince* rather than the predicates *code* and *encode*. This is because the local score provided by the classifier for *(encrypt, convert)* is 0.997—slightly higher than the local score for *(encrypt, encode)*, which is 0.995. Therefore, HTL-FRG inserts the wrong edge *encrypt* $\Rightarrow$ *convert* and then it is unable to insert the correct rule *encrypt* $\Rightarrow$ *encode* because of the FRG assumption. After HTL-FRG terminates, TNF goes over the nodes one by one and is able to determine that overall the predicate *encrypt* fits better as a synonym of the predicates *code* and *encode* and re-attaches it in the correct location. As an aside, notice that both HTL-FRG and TNF are able to identify in this case the correct directionality of the rule *compress* $\Rightarrow$ *encode*. The

11 We do not explicitly write the X and Y variables for brevity, and thus we focus only on predicates where the X variable is a subject.

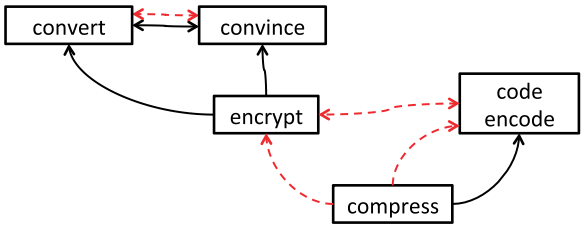

Figure 15
A fragment from the graphs learned by HTL-FRG (black solid edges) and TNF (red dashed edges).

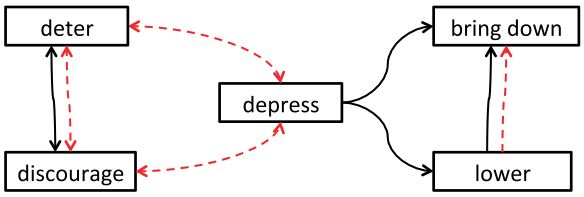

Figure 16
A fragment from the graphs learned by HTL-FRG (black solid edges) and TNF (red dashed edges).

local score given by the classifier for *(compress, encode)* is 0.65, whereas the local score for *(encode, compress)* is –0.11.

As a last example, Figure 16 demonstrates again the problem of predicate ambiguity. The predicate *depress* can occur in two contexts that, though related, instigate different entailment relations. The first context is when the object of the predicate (the *Y* argument) is a *person* and then the meaning of *depress* is similar to *discourage*. A second context is when the object is some *phenomenon*, for example, *The serious economical situation depresses consumption levels*. In initialization, HTL-FRG generates a set of edges that is compatible with the latter context, determining that the predicate *depress* entails the predicates *lower* and *bring down*. TNF re-attaches *depress* as a synonym of *discourage* and *deter* (because this improves the objective function), resulting in a set of edges that corresponds to the first meaning of *depress* mentioned above. The methods we use in this article do not allow learning the rules for both contexts because this would result in violations of the transitivity and FRG assumptions. This again emphasizes the importance of learning graphs where predicates are marked by their various senses, which will result in a model that can directly benefit from the methods suggested in this article.

6. Conclusions

The problem of language variability is at the heart of many semantic applications such as Information Extraction, Question Answering, Semantic Parsing, and more. Consequently, learning broad coverage knowledge bases of entailment rules and paraphrases (Ganitkevitch, Van Durme, and Callison-Burch 2013) has proved crucial for systems that perform inference over textual representations (Stern and Dagan 2012; Angeli and Manning 2014).

In this article, we have presented algorithms for learning entailment rules between predicates that can scale to large sets of predicates. Our work builds on prior work by Berant, Dagan, and Goldberger (2012), who defined the concept of entailment graphs, and formulated entailment rule learning as a graph optimization problem, where the graph has to obey the structural constraint of transitivity.

Our main contribution is a heuristic polynomial approximation algorithm that can learn entailment graphs containing tens of thousands of nodes. The algorithm is based on two main observations: first, that entailment graphs are sparse, and second, that entailment graphs are forest-reducible. This leads to an efficient algorithm in which at the first step the graph is decomposed into smaller components, and in the second step we find a set of edges for each component. The second step is an iterative approximation algorithm in which at each iteration a node or component of the graph is deleted and then re-attached in a way that improves the overall objective function.

We performed a thorough empirical evaluation, in which we ran our algorithm on two separate data sets. On the first data set we witnessed a substantial improvement in runtime at a relatively small cost in performance. On the second data set, we show that our method scales to an untyped entailment graph containing 20,000 nodes, much larger than was previously possible using state-of-the-art global methods. We see that using a global algorithm improves precision for modest values of recall, but that local methods can achieve higher recall than global methods. We perform an analysis and conclude that the main reason for this limitation is predicate ambiguity, which results in graphs that do not adhere to the structural constraints of transitivity and forest reducibility.

In future work, we hope to address the problem of predicate ambiguity. A natural direction is to combine methods that model the context-sensitivity of predicates with global structural assumptions. In this way, one can apply structural constraints for a set of predicates, given that they appear in similar contexts. A second important direction for future work is to demonstrate that using entailment rules learned by our method improves performance in downstream semantic applications such as QA, IE, and Recognizing Textual Entailment.

Appendix A. Max-Trans-Forest Is NP-hard

In this appendix we show that the Max-Trans-Forest problem (8) is NP-hard. We prove that the following decision problem (which is clearly easier than Max-Trans-Forest) is NP-hard: Given a set of nodes V, a function $w : V \times V \rightarrow \mathbb{R}$, and an integer number α, is there a transitive forest-reducible (FRG) subgraph $G = (V, E)$ such that $\sum_{e \in E} w(e) \geq \alpha$? In this appendix we use the standard terminology that for a directed edge $i \Rightarrow j$, the node i is termed the *parent* of node j, and the node j is termed the *child* of i. We start with a simple polynomial reduction from the following decision problem.

Max-Sub-FRG: Given a directed graph $G = (V, E)$, a function $w : E \rightarrow \mathbb{Z}_+$, and a positive integer α, is there a transitive FRG subgraph $G' = (V, E')$ such that $\sum_{e \in E'} w(e) \geq \alpha$?

Max-Sub-FRG $\leq_p$ Max-Trans-Forest: Given an instance $(G = (V, E), w, \alpha)$ of Max-Sub-FRG, we construct the instance $(G' = (V, E'), w', \alpha)$ of Max-Trans-Forest such that $w'(u, v) = w(u, v)$ if $(u, v) \in E$ and $-\infty$ otherwise. We need to show that $(G = (V, E), w, \alpha) \in$ Max-Sub-FRG iff $(G' = (V, E'), w', \alpha) \in$ Max-Trans-Forest. This is trivial: If there is a transitive FRG subgraph of G whose sum of edges $\geq \alpha$, then choosing the same edges will yield a transitive FRG subgraph of G' whose sum of edges $\geq \alpha$. Similarly, any

transitive FRG subgraph of G' whose sum of edges $\geq \alpha$ cannot use any $-\infty$ edges, and therefore the edges of this FRG are in E and this defines a subgraph of G whose sum of edges $\geq \alpha$.

Note that for Max-Sub-FRG, maximizing the sum of weights of edges in the subgraph is equivalent to minimizing the sum of weights of edges not in the subgraph. We now denote by z the sum of weights of the edges deleted from the graph.

Next we show a polynomial reduction from the NP-hard Exact Cover by 3-sets (X3C) problem (Garey and Johnson 1979) to Max-Sub-FRG. The X3C is defined as follows. Given a set X of size $3n$, and m subsets, $S_1, S_2, .., S_m$ of X, each of size 3, decide if there is a collection of n S_i's whose union covers X.

X3C $\leq_p$ Max-Sub-FRG: Given an instance (X, S) of X3C problem, we construct an instance $(G = (V, E), w, z)$ of the Max-Sub-FRG problem as follows (an illustration of the construction is given in Figure A.1). First, we construct the vertices V: We construct $x_1, .., x_{3n}$ vertices, corresponding to the points of X, m vertices $s_1, .., s_m$ corresponding to the subsets S, m additional vertices $t_1, t_2, .., t_m$, and one more vertex a. Next we construct the edges E and the weight function $w : E \to \mathbb{Z}_+$.

1. For each $1 \leq i \leq m$, we add an edge (t_i, s_i) of weight 2.
2. For each $1 \leq i \leq m$, we add an edge (a, s_i) of weight 1.
3. For each $1 \leq j \leq 3n$, we add an edge (a, x_j) of weight $M = 4m$.
4. For each $1 \leq i \leq m$, if $s_i = \{x_p, x_q, x_r\}$, we add 3 edges of weight 1: (s_i, x_p), (s_i, x_q), and (s_i, x_r).

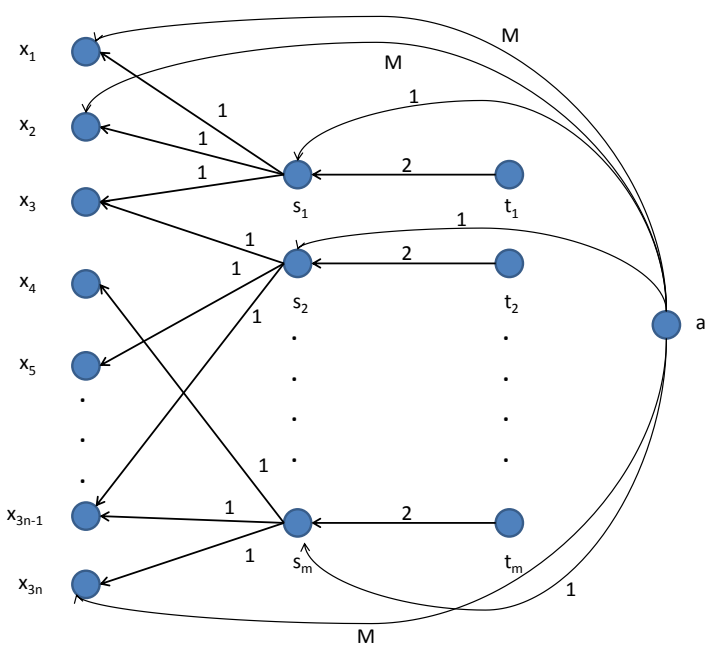

Figure A.1
The graph constructed given an input X of size $3n$ and S of size m. Each $s \in S$ is a set of size 3. In this example, $s_1 = \{x_1, x_2, x_3\}, s_2 = \{x_3, x_5, x_{3n-1}\}, s_m = \{x_4, x_{3n-1}, x_{3n}\}$.

To complete the construction of the Max-Sub-FRG problem, we define $z = 4m - 2n$. We next prove that S has an exact 3-cover of $X \Leftrightarrow$ there is a transitive FRG subgraph of G such that the sum of weights deleted is no more than $4m - 2n$.

$\Rightarrow$: Assume there is an exact 3-cover of X by S. The FRG subgraph will consist of: n edges (a, s_i) for the n vertices s_i that cover X, the $3n$ edges (s_i, x_j), and $m - n$ edges (t_f, s_f), for the $m - n$ vertices s_f that do not cover X. The transitive closure contains all edges (a, x_i), and the rest of the edges are deleted: for all s_f's that are not part of the cover, the three edges (s_f, x_j), and the edge (a, s_f) are deleted. In addition, the weight 2 edges (t_i, s_i) for the s_i's that cover X are deleted. The total weight deleted is thus $3(m - n) + m - n + 2n = 4m - 2n$. It is easy to verify that the subgraph is a transitive FRG - there are no connected components of size > 1, and in the transitive reduction of G there is no node with more than one parent.

$\Leftarrow$: Assume there is no exact 3-cover of X by S, we will show that any FRG subgraph must delete more than $4m - 2n$ weight. We cannot omit any edge (a, x_i), as the weight of each such edge is too large. Thus all these edges are in the FRG and are either deleted during transitive reduction or not.

Assume first that all these edges are deleted in the transitive reduction, that is, for every x_j there exists an s_i such that (a, s_i) and (s_i, x_j) are in the forest. Since there is no collection of n subsets S_i that cover X, there must be at least $k > n$ such s_i's. Consequently, for these s_i's, the forest must not contain the edges (t_i, s_i) (otherwise, s_i would have two parents and violate both the forest and the transitivity properties). For the $m - k$ nodes with no edge (s_f, x_j) we can either add an edge (a, s_f) or (t_f, s_f), but not both (otherwise, s_f would have more than one parent). Since $w(t_f, s_f) > w(a, s_f)$ it is better to delete the (a, s_f) edges. Hence, the total weight of deleted edges is $3m - 3n$ for the edges between s_i's and x_j's, $2k$ in the edges (t_i, s_i), for s_i's that cover the x_j's, and $m - k$ for the edges (a, s_f). Total weight deleted is $4m - 3n + k > 4m - 2n$ since $k > n$.

Assume now that $r > 0$ edges (a, x_j) are not deleted in the transitive reduction. This means that for these x_j's there is no edge (s_i, x_j) for any i (otherwise, x_j will have more than one parent after transitive reduction). This means that $3n - r$ of the x_j's are covered by s_i's. To cover x_j's we need at least $k \geq \lceil n - \frac{r}{3} \rceil$ s_i's. As before, for these s_i's we also have the edges (a, s_i) and we delete the edges (t_i, s_i), and for the $m - k$ nodes s_f that do not cover any x_j it is best to add the edges (t_f, s_f) and to delete the edges (a, s_f). So the weight deleted is $3m - (3n - r)$ for edges between s_i and x_j, $2k$ in the edges (t_i, s_i), and $m - k$ for the edges (a, s_f). Thus, the weight deleted is $4m - 3n + k + r \geq 4m - 3n + \lceil n - \frac{r}{3} \rceil + r \geq 4m - 2n + r - \lfloor \frac{r}{3} \rfloor > 4m - 2n$. ■

Because it is known that X3C is NP-hard, this polynomial reduction shows that Max-Trans-Forest is also NP-hard.

References

Aho, Alfred V., Michael R. Garey, and Jeffrey D. Ullman. 1972. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137.

Angeli, Gabor and Christopher D. Manning. 2014. Naturalli: Natural logic inference for common sense reasoning. In *Proceedings of EMNLP*, pages 534–545, Doha.

Banko, Michele, Michael Cafarella, Stephen Soderland, Matt Broadhead, and Oren Etzioni. 2007. Open information extraction from the web. In *Proceedings of IJCAI*, pages 68–74, Hyderabad.

Ben Aharon, Roni, Idan Szpektor, and Ido Dagan. 2010. Generating entailment rules from FrameNet. In *Proceedings of ACL*, pages 241–246, Uppsala.

Berant, Jonathan, Ido Dagan, Meni Adler, and Jacob Goldberger. 2012. Efficient tree-based approximation for entailment

graph learning. In *Proceedings of ACL*, pages 117–125, Jeju Island.
Berant, Jonathan, Ido Dagan, and Jacob Goldberger. 2010. Global learning of focused entailment graphs. In *Proceedings of ACL*, pages 1,220–1,229, Uppsala.
Berant, Jonathan, Ido Dagan, and Jacob Goldberger. 2011. Global learning of typed entailment rules. In *Proceedings of ACL*, pages 610–619, Portland, OR.
Berant, Jonathan, Ido Dagan, and Jacob Goldberger. 2012. Learning entailment relations by global graph structure optimization. *Journal of Computational Linguistics*, 38(1):73–111.
Berant, Jonathan and Percy Liang. 2014. Semantic parsing via paraphrasing. In *Proceedings of ACL*, pages 1,415–1,425, Baltimore, MD.
Bhagat, Rahul, Patrick Pantel, and Eduard Hovy. 2007. LEDIR: An unsupervised algorithm for learning directionality of inference rules. In *Proceedings of EMNLP-CoNLL*, pages 161–170, Prague.
Blei, David M., Andrew Y. Ng, and Michael I. Jordan. 2003. Latent Dirichlet allocation. *Journal of Machine Learning Research*, 3:993–1022.
Budanitsky, Alexander and Graeme Hirst. 2006. Evaluating WordNet-based measures of lexical semantic relatedness. *Computational Linguistics*, 32(1):13–47.
Chambers, Nathanael and Dan Jurafsky. 2011. Template-based information extraction without the templates. In *Proceedings of ACL*, pages 976–986, Portland, OR.
Chang, Yin-Wen and Michael Collins. 2011. Exact decoding of phrase-based translation models through Lagrangian relaxation. In *Proceedings of EMNLP*, pages 26–37, Edinburgh.
Chklovski, Timothy and Patrick Pantel. 2004. Verb ocean: Mining the web for fine-grained semantic verb relations. In *Proceedings of EMNLP*, pages 33–40, Barcelona.
Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2002. *Introduction to Algorithms*. The MIT Press.
Coyne, Bob and Owen Rambow. 2009. Lexpar: A freely available English paraphrase lexicon automatically extracted from FrameNet. In *Proceedings of IEEE International Conference on Semantic Computing*, pages 53–58, Berkeley, CA.
Dagan, Ido, Dan Roth, Mark Sammons, and Fabio Massimo Zanzotto. 2013. *Recognizing Textual Entailment: Models and Applications*. Morgan and Claypool Publishers.
Dinu, Georgiana and Mirella Lapata. 2010. Topic models for meaning similarity in context. In *Proceedings of COLING*, pages 250–258, Beijing.
Do, Quang and Dan Roth. 2010. Constraints based taxonomic relation classification. In *Proceedings of EMNLP*, pages 1,099–1,109, Cambridge, MA.
Dzeroski, Saso and Ivan Brakto. 1992. Handling noise in inductive logic programming. In *Proceedings of the International Workshop on Inductive Logic Programming*, pages 48–64, Tokyo.
Fader, Anthony, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Proceedings of EMNLP*, pages 1,535–1,545, Portland, OR.
Fellbaum, Christiane, editor. 1998. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press.
Ganitkevitch, Juri, Benjamin Van Durme, and Chris Callison-Burch. 2013. PPDB: The paraphrase database. In *Proceedings of HLT-NAACL*, pages 758–764, Atlanta, GA.
Garey, Michael R. and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman.
Habash, Nizar and Bonnie Dorr. 2003. A categorial variation database for English. In *Proceedings NAACL-HLT*, pages 17–23, Edmonton.
Harris, Zellig. 1954. Distributional structure. *Word*, 10(23):146–162.
Hearst, Marti. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of COLING*, pages 23–28, Nantes.
Joachims, Thorsten. 2005. A support vector method for multivariate performance measures. In *Proceedings of ICML*, pages 377–384, Bonn.
Kelley, J. E. 1960. The cutting-plane method for solving convex programs. *Journal of the Society for Industrial Applied Mathematics*, 8:703–712.
Kipper, Karin, Hoa T. Dang, and Martha Palmer. 2000. Class-based construction of a verb lexicon. In *Proceedings of AAAI*, pages 691–696, Austin, TX.
Lin, Dekang. 1998. Automatic retrieval and clustering of similar words. In *Proceedings of COLING-ACL*, pages 768–774, Montreal.
Lin, Dekang and Patrick Pantel. 2001. Discovery of inference rules for question

answering. *Natural Language Engineering*, 7(4):343–360.
Martins, Andre, Noah Smith, and Eric Xing. 2009. Concise integer linear programming formulations for dependency parsing. In *Proceedings of ACL*, pages 342–350, Singapore.
McCreath, Eric and Arun Sharma. 1997. ILP with noise and fixed example size: a Bayesian approach. In *Proceedings of IJCAI*, pages 1,310–1,315, Nagoya.
Melamud, Oren, Jonathan Berant, Ido Dagan, Jacob Goldberger, and Idan Szpektor. 2013. A two level model for context sensitive inference rules. In *Proceedings of ACL*, pages 1,331–1,340, Sofia.
Meyers, Adam, Ruth Reeves, Catherine Macleod, Rachel Szekeley, Veronika Zielinska, and Brian Young. 2004. The cross-breeding of dictionaries. In *Proceedings of LREC*, pages 1,095–1,098, Lisbon.
Nakashole, Ndapandula, Gerhard Weikum, and Fabian M. Suchanek. 2012. Patty: A taxonomy of relational patterns with semantic types. In *Proceedings of EMNLP-CoNLL*, pages 1,135–1,145, Jeju Island.
ÓSéaghdha, Diarmuid. 2010. Latent variable models of selectional preference. In *Proceedings of ACL*, pages 435–444, Uppsala.
Pantel, Patrick, Rahul Bhagat, Bonaventura Coppola, Timothy Chklovski, and Eduard H. Hovy. 2007. ISP: Learning inferential selectional preferences. In *Proceedings of NAACL-HLT*, pages 564–571, Rochester, NY.
Pekar, Viktor. 2008. Discovery of event entailment knowledge from text corpora. *Computer Speech & Language*, 22(1):1–16.
Ravichandran, Deepak and Eduard H. Hovy. 2002. Learning surface text patterns for a question answering system. In *Proceedings of ACL*, pages 41–47, Philadelphia, PA.
Reichart, Roi and Regina Barzilay. 2012. Multi-event extraction guided by global constraints. In *Proceedings of NAACL-HLT*, pages 70–79, Montreal.
Richens, Tom. 2008. Anomalies in the WordNet verb hierarchy. In *Proceedings of COLING*, pages 729–736, Manchester.
Riedel, Sebastian and James Clarke. 2006. Incremental integer linear programming for non-projective dependency parsing. In *Proceedings of EMNLP*, pages 129–137, Sydney.
Ritter, Alan, Mausam, and Oren Etzioni. 2010. A latent Dirichlet allocation method for selectional preferences. In *Proceedings of ACL*, pages 424–434, Uppsala.
Roth, Michael and Anette Frank. 2012. Aligning predicates across monolingual comparable texts using graph-based clustering. In *Proceedings of EMNLP-CoNLL*, pages 171–182, Jeju Island.
Rush, Alexander M., Roi Reichart, Michael Collins, and Amir Globerson. 2012. Improved parsing and POS tagging using inter-sentence consistency constraints. In *Proceedings of EMNLP*, pages 1434–1444, Jeju Island.
Schoenmackers, Stefan, Jesse Davis, Oren Etzioni, and Daniel S. Weld. 2010. Learning first-order horn clauses from web text. In *Proceedings of EMNLP*, pages 1,088–1,098, Cambridge, MA.
Sekine, Satoshi. 2005. Automatic paraphrase discovery based on context and keywords between NE pairs. In *Proceedings of IWP*, pages 80–87, Jeju Island.
Shinyama, Yusuke and Satoshi Sekine. 2006. Preemptive information extraction using unrestricted relation discovery. In *HLT-NAACL*, pages 304–311, New-York, NY.
Snow, Rion, Daniel Jurafsky, and Andrew Y. Ng. 2006. Semantic taxonomy induction from heterogenous evidence. In *Proceedings of ACL*, pages 801–808, Sydney.
Sontag, David, Amir Globerson, and Tommi Jaakkola. 2011. Introduction to dual decomposition for inference. In Suvrit Sra, Sebastian Nowozin, and Stephen J. Wright, editors, *Optimization for Machine Learning*. MIT Press.
Stern, Asher and Ido Dagan. 2012. BIUTEE: A modular open-source system for recognizing textual entailment. In *Proceedings of ACL System Demonstrations*, pages 73–78. Association for Computational Linguistics, Jeju Island.
Szpektor, Idan and Ido Dagan. 2008. Learning entailment rules for unary templates. In *Proceedings of COLING*, pages 849–856, Manchester.
Szpektor, Idan and Ido Dagan. 2009. Augmenting WordNet-based inference with argument mapping. In *Proceedings of TextInfer*, pages 27–35, Singapore.
Szpektor, Idan, Hristo Tanev, Ido Dagan, and Bonaventura Coppola. 2004. Scaling web-based acquisition of entailment relations. In *Proceedings of EMNLP*, pages 41–48, Barcelona.

Wang, Mengqiu, Wanxiang Che, and Christopher D. Manning. 2013. Effective bilingual constraints for semi-supervised learning of named entity recognizers. In *Proceedings of AAAI*, pages 919–925, Bellevue, WA.

Weeds, Julie and David Weir. 2003. A general framework for distributional similarity. In *Proceedings of EMNLP*, pages 81–88, Sapporo.

Weisman, Hila, Jonathan Berant, Idan Szpektor, and Ido Dagan. 2012. Learning verb inference rules from linguistically-motivated evidence. In *Proceedings of EMNLP*, pages 194–204, Jeju Island.

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

Zeichner, Naomi, Jonathan Berant, and Ido Dagan. 2012. Crowdsourcing inference-rule evaluation. In *Proceedings of ACL (short papers)*, pages 156–160, Jeju Island.