Global Learning of Textual Entailment Graphs

Jonathan Berant

The Blavatnik School of Computer Science, Raymond and Beverly Sackler Faculty of Exact Sciences, Tel Aviv University

This work was carried out under the supervision of Prof. Eytan Ruppin, Prof. Ido Dagan, and Prof. Shimon Edelman

Submitted to the Senate of Tel Aviv University
Thesis for the degree of

Doctor of Philosophy

August 2012

To my parents, Ety and Michael, and my wife, Einav, with all my love.

Acknowledgements

Looking back at my time as a PhD student brings a smile to my face, and there are many people I would like to thank for that. First, I would like to express my deep gratitude to my advisor Ido Dagan. Ido has the rare quality of being a visionary leader, while still keeping track of every detail. Somehow he manages to never lose sight of neither the forest nor the trees. But what is truly great about Ido is that he deeply cares about each one of his students, and for that I will always be thankful. Second, I would like to thank Jacob Goldberger, who has an equal contribution to the work presented in this dissertation. A short consultation with Jacob evolved into almost four years of close and enjoyable collaboration. Jacob is the kind of person who always distills the pure essence of every problem, and helps you focus on what really counts. He challenges you to be creative and think deeply about your problem, and that is what ultimately brings out new ideas.

I would also like to thank Eytan Ruppin and Shimon Edelman, for giving me the opportunity to do research and guiding me in my first steps in this world. Eytan taught me the invaluable lessons that I should always work on problems that are important, and always have fun at what I am doing. Shimon has been an inspiration to me – a truly interdisciplinary scholar, who is fluent in almost every field and always sees the commonalities in different research areas. I would also like to thank Shimon for hosting me at Cornell University, making me feel at home (and I still miss the bike rides).

I spent four exciting months as a visiting graduate student at the University of Washington, and I would like to deeply thank Oren Etzioni for inviting me. I am grateful for having that opportunity, which led to fascinating

research that went on long after I left UW. I would also like to thank Anthony Fader and Stefan Schoenmackers for their ideas and collaboration, and Yoav Artzi, Alan Ritter and Robert Gens for great talks and sweet memories.

I was part of the ITCH project, and I would like thank Oliviero Stock for running the project and for his continuing hospitality. I would also like to deeply thank Claudio Giuliano, who has been a wonderful colleague and a true friend throughout the years of the project.

I am grateful to the many lab members at Bar-Ilan University and Tel-Aviv University I closely worked with: Shachar Mirkin, Idan Szpektor, Eyal Shnarch, Roy Bar-Haim, Meni Adler, Hila Weissman, Naomi Zeichner, Oren Melmoud, Amnon Lotan, and Ben Sandbank. I would also like to thank all of the lab members at both universities for their friendship: Liron Levkovitz, Anat Kreimer, Oron Vanunu, Shaul Karni, Asher Stern, Lili Kotlerman, Chaya Lebeskind, Roni Ben-Aharon, Ofer Bronstein, Eden Erez and Erel Segal. You are the reason why going to work every day was so much fun.

My parents, Michael and Ety, and my family, Roni, Orit, Dafna, Ariel, Tamar and Eytan, are really the ones who made this work possible through their endless love and support. You walked every step of this long and winding road with me and I am fortunate to always have you by my side. Last, I would like to give my thanks and love to my wife, Einav. It is by no means a coincidence that my work got on the right track shortly after we met. Your love and faith give me strength and follow me wherever I go.

Abstract

Two decades after the introduction of the World Wide Web, the world is experiencing an "information explosion", that is, a rapid increase in the amount of available textual information (55). Consequently, the longstanding dream of developing systems that perform semantic inference over natural language text has become ever more pressing. Such systems are invaluable for many natural language understanding applications such as Question Answering, Information Extraction, Search, and Automatic Summarization. The main challenge in developing such methods is overcoming the problem of language variability: the fact that each target meaning can be expressed in natural language in a myriad of ways, which computer programs must recognize.

Textual Entailment is a recent paradigm that reduces language understanding to the following task: given a text and a textual hypothesis, determine automatically whether a human reading the text would infer that the meaning of the hypothesis is true. This definition of textual entailment succinctly captures the type of inferences over text that we would like systems to perform. One of the crucial components in Textual Entailment systems is acquiring resources of "rules" that describe entailment relations between atomic units of language such as words and phrases. For example, the rule '*cause an increase* $\Rightarrow$ *affect*' is useful since the text "salt causes an increase in blood pressure" textually entails "salt affects blood pressure". The absence of such large and precise knowledge resources of entailment rules is currently the main stumbling block for semantic inference systems in general and Textual Entailment systems in particular. In this dissertation, we address this problem by developing methods for automatically learning entailment rules between verbal and predicative phrases.

A substantial body of work on learning entailment rules has been carried out in the last decade. However, most prior work considered each rule in isolation, although entailment rules strongly interact with one another. In this dissertation, we propose models that improve rule learning by exploiting such interactions, most notably the fact that entailment rules are transitive, *i.e.,* if '$A \Rightarrow B$' and '$B \Rightarrow C$', then '$A \Rightarrow C$'. We suggest the problem of learning *entailment graphs*, where graph nodes correspond to predicative phrases and edges represent entailment rules. Given the graph nodes, our goal is to find the optimal set of entailment edges respecting the constraint of transitivity. We show that this optimization problem is NP-hard and suggest to find an exact solution by formulating the problem as an Integer Linear Program (ILP) and applying a standard ILP solver. We empirically test our method over a set of manually-annotated entailment graphs and demonstrate that learning graphs that respect the constraint of transitivity significantly improves the quality of learned rules.

Since ILP solvers do not scale well, it is imperative to develop methods for learning large entailment graphs that contain many entailment rules. A major contribution of this dissertation is in suggesting several such methods. Our first insight is that entailment graphs tend to be sparse, that is, most predicates do not entail one another. We show that sparseness allows us to decompose large graphs into several smaller components, and to exactly solve each component independently of the others. We empirically demonstrate that this approach substantially improves scalability over a large set of entailment graphs. A second improvement in scalability is achieved by identifying a novel structural property of entailment graphs, namely, that they tend to have a "tree-like" structure. We formally define this notion and prove that by assuming that entailment graphs are "tree-like", an efficient polynomial approximation algorithm for learning entailment graphs can be devised, which empirically scales to graphs containing tens of thousands of nodes.

We apply our methods over an open domain web-scale data set and generate new knowledge resources containing hundreds of thousands and even mil-

lions of predicative entailment rules. We show that our resources compare favorably to current state-of-the-art knowledge-bases in several evaluations, and publicly release these resources for the benefit of the scientific community. Last, we present a novel application for text exploration that is based on learning entailment graphs. We propose that entailment graphs can aid text exploration by allowing users to navigate from one document to another based on entailment relations between propositions in the documents. As a case study, we implement a text exploration system in the health-care domain.

To conclude, this dissertation demonstrates that by exploiting the structural properties of entailment graphs one can achieve more precise and scalable methods for entailment rule learning. This opens the door for further research on the structure of entailment graphs and on the dependencies between textual entailment and other semantic relations. We believe that taking this global view on the problem of learning semantic relations can lead to algorithms that learn large and coherent knowledge resources, which are so essential for progress in the field of semantic inference.

Contents

1

Introduction

1.1 Semantic Inference

Semantic inference is the task of performing inferences over natural language representations. This task has been a longstanding goal in Artificial Intelligence (AI) ever since the inception of the field in the 1950s, and goes back even to Alan Turing's seminal paper (177) from 1950, in which he suggested to determine whether a machine is intelligent by conversing with it in natural language. As an example for the type of challenges inherent to semantic inference, consider the following pair of sentences:

(1.1) (a) *Lindsay Lohan was convicted of driving under the influence of alcohol.*

(b) *The police arrested Ms. Lohan for drunk driving.*

A semantic inference system is expected to be able to automatically determine that sentence 1.1b can be inferred from sentence 1.1a.

Performing semantic inference is at the core of many Natural Language Processing (NLP) applications. In Question Answering (QA), systems are required to detect texts from which the expected answer can be inferred. For example, a system would have to identify that the sentences above contain a valid answer for the question *'What was Lindsay Lohan accused of?'*. Information Extraction (IE) systems should identify certain events that are expressed in text and their participants. For instance, a system aiming to extract *'trial'* events would have to recognize that sentence (a) above implies a *'trial'* event in which the defendant is 'Lindsay Lohan' and the felony is 'driving under the influence of alcohol'. Summarization systems should not include sentences

that can already be inferred by other sentences in the summary, and similar analogies can also be derived for applications such as Information Retrieval (IR) and Machine Translation (MT) evaluation.

Naturally, semantic inference is quite a difficult task. One of the prominent reasons for that is the *variability* of natural language, that is, the fact that the same information can be expressed in a myriad of different ways. In the aforementioned example, the expressions *'drunk-driving'* and *'driving under the influence of alcohol'* are *paraphrases*, that is, they are different ways of expressing an equivalent meaning. Moreover, much of the variability in language stems from assumptions we make on the knowledge that humans possess. Such knowledge includes, for instance, the fact that an event of *'conviction'* occurs after an event of *'arrest'*, and that the organization responsible for arrests is *'the police'*. A system that intends to perform semantic inference over natural language must confront such hurdles.

Although different semantic inference applications face similar challenges, as outlined above, research in the various fields of semantic inference proceeded for many years in parallel. In the last decade, a unifying framework termed *Textual Entailment (TE)* has been suggested, which focuses on the common need of all applications to capture inference relations between textual "units".

1.2 Textual Entailment

The textual entailment framework, suggested by Dagan and Glickman (49, 50) and Dagan et al. (48), is a generic paradigm that aims to reduce the needs of many semantic inference applications to a single task. In this framework, a system is given a certain *text*, T (typically a sentence, a paragraph or a document), and a textual assertion termed the *hypothesis*, H (typically a short assertion), and is required to determine whether a human reading T is most likely to infer that H is true. In this case, we say that T textually entails H and denote this by '$T \Rightarrow H$'. For example, if sentence 1.1a is that text T and sentence 1.1b is the hypothesis H, then in this case '$T \Rightarrow H$'. For brevity, we will employ the word *'entails'* instead of the phrase *'textually entails'* in the remainder of this dissertation.

The definition of textual entailment focuses on what "humans are likely to infer". This is in contrast to formal semantics (38) where the common definition is that T

entails H if in any possible world in which T is true, H must also be true. Dagan et al. (48) explain that the reason for choosing this more relaxed definition is to allow for the types of inferences that are typically expected from NLP *applications*. Consider the following two examples adapted from the first *Recognising Textual Entailment (RTE1)* dataset (50):

(1.2) **Text:** *iTunes software has seen strong sales in Europe.*
Hypothesis: *Strong sales for iTunes in Europe.*

(1.3) **Text:** *The U.S government reported that two out of three Americans are fat.*
Hypothesis: *More than half of U.S citizens are fat.*

In Example 1.2 the text entails the hypothesis according to the definition of both textual entailment as well as formal semantics. However, In Example 1.3 it is conceivable to imagine a world where the U.S government would like to deceive its citizens, and reports that 66% of Americans are fat although less than half are actually over-weight. Thus, the definition of formal semantics implies that in this example T does not entail H. Nevertheless, we expect an inference system to determine that T entails H since cases where this inference does not hold seem to be marginal.

Note that textual entailment is a directional relation. For instance, in Example 1.1, it is safe to assume that if Lindsay Lohan was convicted for driving under the influence of alcohol, then prior to that she had been arrested. However, if we only know that she had been arrested by the police, it would be premature to infer that she was also convicted. The task of determining whether both '$T \Rightarrow H$' and '$H \Rightarrow T$' is also common in NLP and is generally known as *paraphrasing* (112). For instance in Example 1.2 the text and the hypothesis are paraphrases of each other.

Reducing semantic tasks to textual entailment is generally quite natural. In QA, the answer passage can be cast as a text T that entails the question after appropriately transforming it into a hypothesis H with a variable. For example, to answer the above mentioned question *'What was Linsday Lohan accused of?'* we should look for texts entailing the hypothesis with a variable *'Lindsay Lohan was accused of X'*. To find *'trial'* events in an IE setting we should find texts that entail the hypothesis with variables *'X was tried for Y'*. In Summarization, we can omit sentences that are entailed by other sentences that are already in the summary; In Machine Translation evaluation,

we should check whether an automatically-generated translation is a paraphrase of a reference gold-standard translation, etc. Practically, TE has already been integrated into various semantic applications such as QA (76, 88, 124), IE (147), Summarization (77), Machine Translation evaluation (127) and more.

Since 2005, seven annual *Recognising Textual Entailment (RTE)* challenges were held, allowing developers to evaluate their success in confronting this generic task. The classical RTE challenge is composed of a set of pairs of T and H, and systems need to determine for each pair whether '$T \Rightarrow H$'. Systems participating in RTE challenges vary in their approach considerably. Few systems take a traditional approach and try to convert the text and hypothesis to logical formulas and apply a theorem prover (25), but most systems operate over linguistic representations such as parse trees.

One prevalent line of work in modeling textual entailment is to construct an *alignment* between words and phrases in T and H, and then to determine entailment by estimating the quality of the alignment (30, 109, 128, 193). A second major approach is to try and "prove" H by transforming T in a sequence of steps into a structure that is as "close" to H as possible (9, 78, 167). Regardless of the specifics of the approach, every textual entailment system depends on knowledge that will allow it to handle the problem of language variability and will bridge in some manner the gap between the linguistic content of the text and the linguistic content of the hypothesis (*e.g.,* the fact that *'Americans'* and *'U.S citizens'* are equivalent in Example 1.3). This knowledge can be generally described in what is often called *inference rules* or *entailment rules*.

1.3 Entailment Rules

Entailment rules describe an inference relation between two "atomic" textual units. An entailment rule '$L \Rightarrow R$' denotes that the meaning of the left-hand-side (LHS) of the rule, L, entails the meaning of the right-hand-side of the rule, R (at least in some contexts). For instance, the rule '*Lindsay Lohan* $\Rightarrow$ *actress*' represents the fact that a reference in text to *'Lindsay Lohan'* is likely to be also a reference to an *'actress'* (119). As mentioned, entailment rules are employed in alignment-based entailment systems to align non-identical phrases, in transformation-based entailment systems to perform modifications to the text, and in other systems in some analogous manners.

Entailment rules can be categorized in a multitude of ways. One way is with respect to the representation or content of the LHS and RHS. This categorization is mostly relevant for the type of mechanism that will be required by an entailment system that wishes to utilize the rules. If both the LHS and RHS of a rule contain only lexical items, that is actual words (as in '*Lindsay Lohan* $\Rightarrow$ *actress*'), then we refer to this as a *lexical rule*. Another type of rule are *template rules*, in which the LHS and RHS contain both lexical items but also one or more shared variables, *e.g.,* '*X defeat Y* $\Rightarrow$ *Y lose to X*'. Often, these rules also specify the syntactic relation between the variables and the lexical items. For instance, when using dependency trees the representation of the previous rule is '$X \xleftarrow{subj}$ *defeat* $\xrightarrow{obj} Y \Rightarrow Y \xleftarrow{subj}$ *lose* $\xrightarrow{mod}$ *to* $\xrightarrow{p-comp} Y$' (assuming dependency labels induced by the Minipar parser (101)).

Employing variables enables more precise inferences since we can restrict the type of words that match the rule in any manner, *e.g.,* we can have a rule where the X is specified to be a noun, another rule where it is specified to be a geographic entity, etc. Adding syntactic information also improves inference precision since the rule only applies in appropriate syntactic contexts. On the other hand, template rules require entailment systems to perform more complex procedures than lexical rules, *e.g.,* matching the LHS or RHS of a template rule to a text is a more involved process.

A more abstract type of entailment rule are *syntactic rules*, in which no content words occur at the LHS and RHS of the rule. These rules capture general syntactic phenomena in languages such as transformations from passive to active in English (see Figure 1.1) (10).

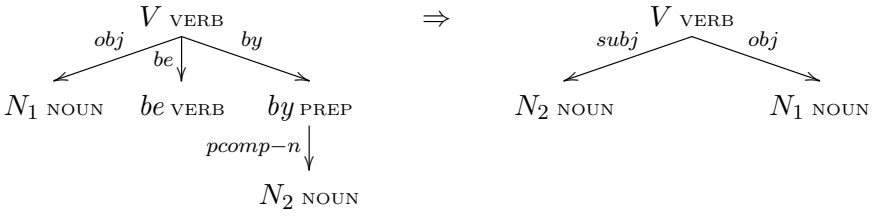

Figure 1.1: Passive to active transformation taken from Bar-Haim et al. (10). N_1 and N_2 are noun variables and V is a verb variable. Dependency labels are based on the MINIPAR dependency parser (101).

Another way to categorize entailment rules is according to the type of knowledge

that they represent. Sammons et al. (151) and Yates and LoBue (107) attempted to survey the wide spectrum of common sense knowledge required for performing the complex task of recognising textual entailment. This categorization is important for automatic learning of entailment rules, since different types of knowledge are amenable to different types of rule acquisition methods. The following entailment rules provide a few examples for these types of knowledge:

(1.4) (a) '*dog* ⇒ *mammal*'

(b) '*Steering wheel* ⇒ *motor vehicle*'

(c) '*Sydney* ⇒ *Australia*'

(d) '*X is before Y* ⇒ *Y is after X*'

(e) '*X reduce Y* ⇒ *X affect Y*'

(f) '*X snore* ⇒ *X sleep*'

(g) '*X convicted of Y* ⇒ *X arrested for Y*'

(h) '*X admitted into Y* ⇒ *X belong to Y*'

Many types of common-sense knowledge can be captured by entailment rules. Examples 1.6a-c focus on objects and entities, while Examples 1.6d-h highlight actions and events. Example 1.6a provides taxonomic knowledge that a *'dog'* is a type of *'mammal'*; the rule in Example 1.6b is due to the fact that steering wheels are almost invariably a part of a motor vehicle (*meronymy*); Example 1.6c is an instance of geographical knowledge, and Example 1.6d is concerned with temporal reasoning. Automatically learning each type of rule can benefit from different sources of information and different learning algorithms.

Though entailment rules encompass a wide range of phenomena, there are some types of knowledge that are not easily captured by entailment rules. A classic example for that is arithmetic knowledge. Consider the following sentences:

(1.5) (a) *Eight passengers and three crew members lost their lives on a plane crash.*

(b) *Eleven people died on a plane crash.*

Clearly, encoding in a succinct form arithmetic knowledge such as 8+3=11 would be more natural in some formal language rather than by entailment rules.

In this dissertation we will focus on automatically learning an important type of entailment rules, illustrated by Examples 1.6e-h, namely, entailment rules between *predicates*.

1.3.1 Predicative entailment rules

One of the most basic types of textual utterances are *propositions*. Propositions are simple natural language expressions that comprise a single *predicate* and one or more *arguments*. The arguments correspond to semantic *concepts* while the predicate describes a property of a concept or a semantic relation between multiple concepts. Consider the following three propositions:

(1.6) (a) *Ice melts.*

(b) *Alcohol affects blood pressure.*

(c) *Facebook bought Instagram for one billion dollars.*

In the first proposition, the argument is *'ice'* and the predicate *'melt'* describes a property of ice. In the second proposition, the arguments are *'alcohol'* and *'blood pressure'* and the predicate *'affect'* describes a semantic relation between the two arguments. Similarly, in the third proposition the predicate *'buy'* describes a relation between the arguments *'Facebook'*, *'Instagram'* and *'one billion dollars'*. A proposition where one or more of the arguments are replaced by variables is termed *propositional template* or *predicative template*. For example, *'X affect blood pressure'* and *'X buy Y for Z'* are predicative templates. The main focus of this dissertation is on automatically learning entailment rules between predicative templates, such as '*X increase blood pressure* $\Rightarrow$ *X affect blood pressure*' and '*X buy Y from Z* $\Rightarrow$ *X own Y*'. For brevity, we will term these rules *predicative entailment rules* and whenever the distinction is immaterial refer to predicative templates simply as predicates.

Natural language uses predicates to express actions, events and states. Consequently, facts and knowledge almost invariably involve predicates and so predicative entailment rules are central for the task of textual entailment. This has led to active research on broad-scale acquisition of such rules (104, 153, 156, 172, 191).

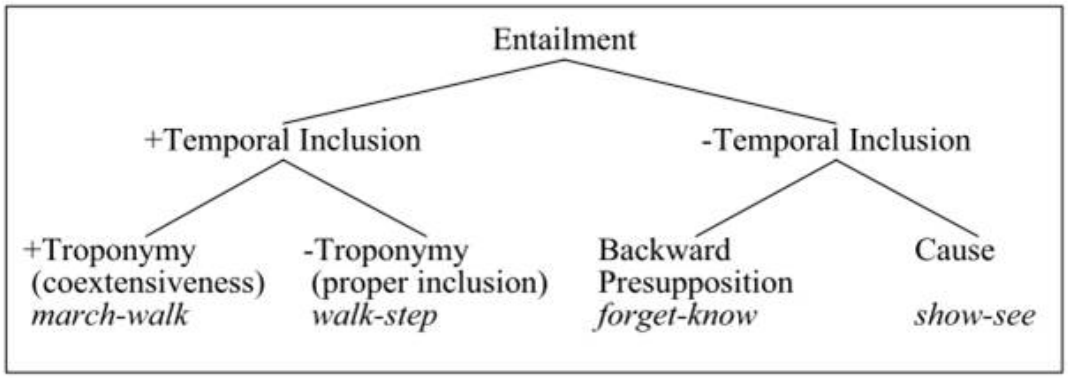

Figure 1.2: Classification of verb entailment to subtypes taken from Fellbaum (59).

Another valuable property of propositional templates is that they essentially represent abstract propositions and thus can be assigned a truth value. This makes rule evaluation much simpler, which is extremely important for developers of entailment rule resources. For example, the rule '*X increase blood pressure* ⇒ *X affect blood pressure*' is correct since according to the classic definition of entailment if something increases blood pressure, then it must also affect blood pressure. Compare this to entailment rules between entities: it is less trivial to determine the correctness of the rule '*Abbey Road* ⇒ *The Beatles*' – Mirkin et al. (119) explain that this is a valid rule since there are non-anecdotal natural language texts in which a *reference* to *'Abbey Road'* implies a reference to *'The Beatles'*. Of course this definition deviates from the original formulation of the textual entailment relation.

It is also interesting to notice that predicative entailment rules can be further divided into subtypes. Fellbaum (59) proposed a hierarchical classification of the entailment relation between verbs (which correspond loosely to predicates) according to the temporal relation between the events denoted by the verbs (see Figure 1.2). The first division depends on whether the predicates denote events or states that co-occur in time. If the events or states co-occur in any time, then in case they start and end at the same time this is called *troponymy*, which is analogous to *hyponymy* in nouns. In other words, one predicate is more specific than the other, as in Example 1.6e. If one event or state occurs during another event and is properly included in it, then this

is termed *temporal inclusion*, as in Example 1.6f. If the predicates denote events or states that do not co-occur in time then again there are two cases. In the first case, the occurrence of one event presupposes an occurrence of the other in a previous point in time. This is known as *backward presuppostion* and is illustrated in Example 1.6g. The last case is when one event or state necessarily causes after it another event or state. This is known as *cause-effect* and is illustrated in Example 1.6h.

There have been few attempts to focus on learning subtypes of predicative entailment (176, 194). In this dissertation we will focus on the general predicative entailment relation, but will discuss the potential of examining entailment subtypes in Chapter 7.

Next, we turn to the topic of learning entailment rules in general and in particular automated learning of predicative entailment rules.

1.4 Learning Predicative Entailment Rules

The construction and acquisition of knowledge resources in general has been a fundamental task long before the foundation of textual entailment. Broad-coverage semantic resources such as WordNet (59) and Cyc (99) have been manually constructed at great cost, describing various semantic relations between textual units and concepts. Although these resources have been successfully employed in a wide variety of applications, they suffer from limited coverage. Furthermore, the amount of textual information is increasing in such a rapid pace that it has become virtually impossible to manually create and annotate all the knowledge necessary for semantic inference.

The other side of the coin is that the massive amounts of available text provide an unprecedented opportunity for automated corpus-based learning methods. A plethora of methods have been employed including pattern-based methods (20, 27, 81), distributional similarity methods (102, 105, 153), graph walks (86) and many more. In Section 2 we will provide an extensive survey of works that are most relevant for the understanding of this dissertation.

Since predicative entailment rules are fundamental in many semantic applications there have been numerous attempts to acquire such rules automatically in the last decade (21, 104, 142, 153, 156, 158, 172, 175). However, most previous work focused on learning such rules in isolation, while ignoring the interaction between rules. In particular, given a pair of predicative templates x and y, most methods tried to determine

whether '$x \Rightarrow y$' by computing various statistics about x and y: whether they appear in similar contexts, whether they co-occur often in the same local scope, whether they are related to one another in manually-constructed ontologies, etc. We term this type of learning *local learning* as it involves only information about the two templates of a single rule. However, it is clear that the decision whether '$x \Rightarrow y$' should be influenced by similar decisions regarding other predicates. For example, if we know that the predicate x is a synonym of the predicate z and that '$z \Rightarrow y$', then '$x \Rightarrow y$' must also be true. More precisely, one of the prominent phenomena of the textual entailment relation is that it is *transitive*, that is the rules '$x \Rightarrow y$' and '$y \Rightarrow z$' imply the rule '$x \Rightarrow z$'. Leveraging this transitivity is in fact at the core of transformation-based entailment systems.

The main contribution of this dissertation is in what we term *global learning*, that is, methods for learning predicative entailment rules that take into account the interaction between different rules. In our setting, we model the problem as a graph learning problem where the input is a set of predicates $\mathcal{X}$, which are the graph nodes, and the goal is to learn simultaneously all of the graph edges, representing rules '$x \Rightarrow y$', where $x, y \in \mathcal{X}$. This allows us to incorporate information about the global structure of the graph into the learning algorithm. The main structural property we utilize is indeed *transitivity*, but we also investigate other interesting properties, such as the tendency of predicative entailment rules to form "tree-like" structures (Chpater 4).

At this point we should note that the property of transitivity does not necessarily always hold. The most common reason for that is the problem of *context*. The classic example is the following: the rule '*X buy Y* $\Rightarrow$ *X acquire Y*' is correct in the context of companies and purchases. The rule '*X acquire Y* $\Rightarrow$ *X learn Y*' is correct in the context of skills or knowledge. However, '*X buy Y* $\not\Rightarrow$ *X learn Y*'. This violation of transitivity is caused by the fact that the word *'acquire'* has different meanings in different contexts, or in other words the one-to-many mapping from form to meaning in natural language, known as *ambiguity*. In this dissertation we will first sidestep this issue by working in settings where the the context problem is greatly reduced (Chapters 3 and 4). Then, we will empirically examine and analyze its effect in settings where it is likely to pose a real problem (Chapter 5).

1.5 Contributions and Outline

In this dissertation we demonstrate that modeling entailment rule learning as a graph learning problem and applying constraints on permissible graph structures can substantially improve the quality of learned knowledge-resources. Constraining the graph structure results in an optimization problem that is computationally hard and thus we suggest algorithms that scale to graphs containing tens of thousands of nodes. We apply our algorithms over web-scale data and publicly release a state-of-the-art resource of predicative entailment rules. Finally, we propose to use graphs that contain predicative entailment rules as the foundation of a novel application for text exploration, and implement this application in the health-care domain.

In Chapter 2 we provide background on prior work in the field of learning predicative entailment rules. We aim to both provide the necessary background for the understanding of this dissertation but also highlight the relations and connections between the field of entailment rule learning and adjacent research area that share common characteristics.

We present our basic learning model in Chapter 3. We first describe a structure termed **entailment graph** that models entailment relations between propositional templates and then present an algorithm that uses a global approach to learn the entailment relations. This is performed by defining an objective function and looking for the graph that maximizes that function and satisfies a global transitivity constraint. The optimization problem is shown to be NP-hard, and then formulated as an *Integer Linear Program (ILP)* and exactly solved by an ILP solver. We empirically demonstrate that taking advantage of global information such as transitivity significantly improves performance relative to local learning, over a manually annotated data set.

In Chapter 4 we focus on solving efficiently the aforementioned optimization problem, since ILP solvers do not scale well to large data. We propose two approaches. The first approach takes advantage of a structural property of entailment graphs, that is, the fact that they tend to be sparse. We propose an exact algorithm for learning the edges of sparse entailment graphs that is based on a decomposition of the original graph into smaller components and show that it substantially improves the scalability of our model. The second approach complements the first and utilizes another structural property of entailment graphs, namely, entailment graphs tend to have a "tree-like"

structure. We show that by assuming that entailment graphs are "tree-like" we can tailor an iterative optimization procedure to our problem . This procedure is polynomial, converges to a local maximum, is empirically fast, and obtains performance that is close to that given by the optimal solution.

Chapter 5 describes the creation of a large resource containing millions of predicative entailment rules, which was generated over web-scale data. In this chapter we apply local and global methods described in earlier chapters over sets of $\sim 10^4 - 10^5$ predicates and demonstrate empirically that we are able to learn large knowledge bases that outperform previous state-of-the-art. Moreover, we make the learned knowledge bases publicly available for the benefit of the NLP community. In addition, we investigate in this chapter the effects of applying transitivity constraints over open domain and context-sensitive data.

Chapter 6 proposes an application that directly benefits form our learned entailment graphs. We suggest that entailment graphs can aid text exploration, by allowing users to navigate through collections of documents according to entailment relations that exist between propositions found in these documents. As a case study, we implement a text exploration system over a large corpus in the health-care domain. A demo of the system is also made publicly available.

In the last chapter of this dissertation, we discuss our results and suggest various directions for future research.

1.6 Publications Related to this Dissertation

Most of the contributions described in this dissertation have first appeared in other publications. These are the publications related to each chapter:

- Chapter 3:
    1. Jonathan Berant, Ido Dagan and Jacob Goldberger. 2010. *Global Learning of Focused Entailment Graphs.* Proceedings of ACL (17).
    2. Jonathan Berant, Ido Dagan and Jacob Goldberger. 2012. *Learning Entailment Relations by Global Graph Structure Optimization.* Computational Linguistics (19).
- Chapter 4:

1. Jonathan Berant, Ido Dagan and Jacob Goldberger. 2011. *Global Learning of Typed Entailment Rules*. Proceedings of ACL (18).
2. Jonathan Berant, Ido Dagan, Meni Adler, and Jacob Goldberger. 2012. *Efficient Tree-based Approximation for Entailment Graph Learning*. Proceedings of ACL (16).

- Chapter 5:

1. Jonathan Berant, Ido Dagan and Jacob Goldberger. In preparation. *A Large-scale Resource of Predicative Entailment Rules*. Language Resources and Evaluation.
2. Naomi Zeichner, Jonathan Berant and Ido Dagan. 2012. *Crowdsourcing Inference-Rule Evaluation*. Proceedings of ACL (short paper) (195).

- Chapter 6: Meni Adler, Jonathan Berant and Ido Dagan. 2012. *Entailment-based Text Exploration with Application to the Health-care Domain*. Proceedings of ACL demo session (1).

2

Background

The task of acquiring predicative entailment rules is tightly bound to other tasks that involve learning of semantic relations. Adjacent fields inspire and influence one another and it is illuminating to point out the various similarities and differences between them. Different methods relate to one another in various aspects - in the target semantic relation chosen (entailment, paraphrasing, hyponymy, meronymy), in the type of data used (lexicographic resources, monolingual corpora, bilingual corpora), in the source of information (distributional similarity, pattern-based methods), in the learning paradigm (supervised, unsupervised), in the linguistic representation, etc.

In this Chapter we will survey the main approaches for learning predicative entailment rules and other semantic relations, and highlight the inter-connections between the various works according to the above mentioned aspects.

2.1 Type of Data

The two main types of data useful for learning semantic relations are lexicographic resources and natural language corpora.

2.1.1 Lexicographic resources

Lexicographic resources are manually-constructed knowledge-bases that describe lexicalized items in some manner. Extracting semantic relations from lexicographic resources is sometimes trivial, for example the *hypernymy* relation is explicitly annotated

in WordNet, but in other cases can be more involved, for instance semantic relatedness can be estimated by performing random walks over lexicographic resources such as Wikipedia (86, 192). Providing an overview of the awesome number of methods (28, 62, 160, 180) that use lexicographic resources to extract semantic relations is naturally beyond the scope of this dissertation and so we will focus on works more directly related to predicative entailment rules.

WordNet (59), by far the most widely used resource, specifies relations between lexical items such as *hyponymy*, *synonymy* and *derivation*, which are related to the *textual entailment* relation. For example, if WordNet specifies that *'reduce'* is a hyponym of *'affect'*, then one can infer that '*reduce* $\Rightarrow$ *affect*'.

A drawback of WordNet is that it specifies semantic relations for words and terms but not for more complex expressions. For example, WordNet does not cover a complex predicate such as *'cause a reduction in'*. Another drawback is that it only supplies semantic relations between lexical items, but does not provide any information on how to map arguments of predicates. For example, WordNet specifies that there is an entailment relation between the predicates '*pay*' and '*buy*', but does not describe the way in which arguments are mapped: '*X pay Y for Z* $\Rightarrow$ *X buy Z from Y*'. Thus, using WordNet directly to derive predicative entailment rules is possible only for semantic relations such as hyponymy and synonymy, where arguments typically preserve their syntactic positions on both sides of the rule.

Some knowledge bases try to overcome this difficulty: Nomlex (110) is a dictionary that provides the mapping of arguments between verbs and their nominalizations (for example, '*X's treatment of Y* $\Rightarrow$ *X treat Y*') and has been utilized to derive predicative entailment rules (118, 173). FrameNet (6) is a lexicographic resource that is arranged around "frames": each frame corresponds to an event type and includes information on the predicates and arguments relevant for that specific event supplemented with annotated examples that specify argument positions. For instance, FrameNet contains an *'attack'* frame, and specifies that *'attack'* events include an *'assailant'*, a *'victim'*, a *'weapon'*, etc. In addition, Framenet provides a list of lexical items that belong to the *'attack'* frame, such as *'attack'*, *'bomb'*, *'charge'*, *'invade'*, and more. Consequently, FrameNet was also used to derive predicative entailment rules (14, 47).

Other relevant lexicographic resources include (a) CatVar (72): a database specifying sets of derivationally-related lexical items with their part-of-speech in English (*e.g.,*

'trick::noun', *'trick::verb'*, *'trickery::noun'*, *'trickery::adjective'*). (b) VerbNet (93): a verb lexicon augmented with syntactic and semantic information derived from Levin's (100) verb classes. (c) ProbBank (92): a project that bears similarities to FrameNet and contains a corpus of sentences annotated with verbal predicates and their arguments.

2.1.2 Corpus-based methods

Corpus-based methods are used to learn broad-scale resources, since lexicographic resources tend to have limited coverage. Madnani and Dorr (112) presented a comprehensive overview of corpus-based methods for phrasal and sentential *paraphrasing*, which as mentioned largely corresponds to bi-directional textual entailment. They organize the methods according to the type of corpus used: a single monolingual corpus, monolingual comparable corpora, monolingual parallel corpora, and bilingual parallel corpora.

Single monolingual corpus In our work, the main type of data utilized is a single monolingual corpus (combined with information from lexicographic resources). Learning inference relations from a monolingual corpus usually employs the "distributional hypothesis" (80) that semantically similar words occur in similar contexts. We elaborate on learning from a monolingual corpus in Section 2.2.

Monolingual Parallel corpora Monolingual parallel corpora are created when the same text is translated into a target language by several translators. Clearly, the advantage of such a corpus is that we automatically obtain pairs of sentences that are semantically equivalent. This allows to extract paraphrases by directly aligning words and phrases from one sentence to the other, in a fashion similar to Statistical Machine Translation (13, 87, 129, 140). However, monolingual parallel corpora are quite scarce, and so the amount of paraphrases that can be derived from such methods is rather limited. In addition, alignment-based methods fit more naturally to the *paraphrasing* relation rather than to the directional *entailment* relation.

Monolingual comparable corpora A monolingual comparable corpus is composed documents in the same language that overlap in the information they convey, for instance, stories about the same events from different press agencies (11). Monolingual

comparable corpora are much more common than monolingual parallel corpora and so potentially can yield more entailment rules. On the other hand, parallelism between sentences is replaced by topical overlap at the level of documents. Thus, the task of aligning words and phrases from one document to the other becomes much more difficult. Consequently, methods that take advantage of comparable corpora either utilize matching Named Entities (NEs) in the pair of documents as anchors for discovering paraphrase candidates (159), or develop more sophisticated coarse-grained alignment methods and leverage this alignment (12, 157)

Bilingual parallel corpora A bilingual parallel corpus contains a text alongside its translation into another language. As globalization spreads throughout the world, the availability of bilingual parallel corpora is increasing. Similar to monolingual parallel corpora, the advantage of bilingual parallel corpora is that sentences with equivalent semantics are aligned to one another. Paraphrase extraction from bilingual parallel corpora was proposed by Bannard and Callison-Burch (8). They generated a bilingual phrase table between a source and a target language using standard Statistical Machine Translation (SMT) techniques, and then obtained paraphrases in the source language by *pivoting*, that is, looking for different phrases in the source language aligned in the phrase table to the same target phrase. Subsequent research extended this idea and included various types of syntactic information (31, 44, 111, 196), extracted syntactic paraphrases (64) with Synchronous Context Free Grammars (SCFGs) (3), and employed more than just a pair of languages (94). The main drawback of bilingual parallel corpora is that methods rely on an often noisy automatic alignment step.

To the best of our knowledge there have been few attempts to combine the information from the four types of data presented in this Section. However, Chan et al. (36) and also recently Ganitkevitch et al. (65) attempted to re-rank paraphrases extracted from a parallel bilingual corpus using distributional similarity computed over a monolingual corpus. This method combines orthogonal signals and thus in our opinion has potential to improve current state-of-the-art techniques.

Next, we describe methods for learning or extracting entailment rules from a single monolingual corpus. We stress again that this is the type of data used most often to learn directional entailment rules (rather than paraphrases only).

2.2 Single Monolingual Corpus Learning

Most methods proposed in the past for learning predicative entailment rules utilized *local learning*, as we termed in Section 1.4. We first review local methods (Section 2.2.1) and then turn to *global* approaches (Section 2.2.2).

2.2.1 Local learning

2.2.1.1 Distributional similarity

Distributional similarity is the most popular method for learning semantic relations between entities, and is based on the idea that semantically similar entities occur in large corpora in relatively similar contexts. Distributional similarity algorithms generally define *"elements"* that are compared by applying a similarity measure over *feature vectors* that represent the elements' contexts. In some algorithms the elements are lexical, that is, they do not contain variables and are unparsed. Lin (102) proposed an unsupervised information-theoretic symmetric similarity measure where elements are words and context features are syntactic, *i.e.,* based on dependency relations. Pasca and Dienes (132) extracted word and phrase paraphrases from an unparsed web snapshot in an unsupervised manner, where context features are n-grams. Bhagat and Ravichandran (22) presented an unsupervised method for extracting paraphrases from a large 150GB pos-tagged corpus, where elements are POS-tagged phrases and features are nouns or noun-noun compounds.

When learning entailment rules between predicates, the elements are the predicates and usually contain some syntactic information, while the features are the arguments. Lin and Pantel (104) proposed the *DIRT* algorithm that is based on the mentioned *Lin* similarity measure. The predicates are represented by *binary propositional templates*, which are dependency paths in a parsed sentence between two arguments of a predicate, where the arguments are replaced by variables. Note that in a dependency tree, a path between two arguments must pass through their common predicate. Also note that if a predicate has more than two arguments, then it is represented by more than one binary template, where each template corresponds to a different aspect of the predicate. For example, the proposition *'I bought a gift for her'* contains a predicate and three arguments, and therefore is represented by the following three templates: *'X* $\xleftarrow{subj}$ *buy* $\xrightarrow{obj}$ *Y'*, *'X* $\xleftarrow{obj}$ *buy* $\xrightarrow{prep}$ *for* $\xrightarrow{pcomp-n}$ *Y'* and *'X* $\xleftarrow{subj}$ *buy* $\xrightarrow{prep}$ *for* $\xrightarrow{pcomp-n}$ *Y'*.

For each template Lin and Pantel computed two sets of features F_x and F_y, which are the nouns that instantiate the arguments X and Y respectively in a large corpus. Given a template t and its feature set for the X variable F_x^t, every $f_x \in F_x^t$ is weighted by the pointwise mutual information between the template and the feature: $w_x^t(f_x) = \log \frac{Pr(f_x|t)}{Pr(f_x)}$, where the probabilities are computed using maximum likelihood over the corpus. Given two templates u and v, the *Lin* measure (102) is computed for the X variable:

$$Lin_x(u,v) = \frac{\sum_{f \in F_x^u \cap F_x^v} [w_x^u(f) + w_x^v(f)]}{\sum_{f \in F_x^u} w_x^u(f) + \sum_{f \in F_x^v} w_x^v(f)} \tag{2.1}$$

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:

$$DIRT(u,v) = \sqrt{Lin_x(u,v) \cdot Lin_y(u,v)} \tag{2.2}$$

If $DIRT(u,v)$ is high, this means that the templates u and v share many "informative" arguments and so the predicates are semantically similar.

Szpektor et al. (175) suggested *TEASE*, a web-based method for paraphrase recognition that is based on the idea of *bootstrapping* – given a seed predicative template, queries to the web are used to find argument fillers for the template, which are then used in turn to find other templates that hold an entailment relation (in any direction) with the original seed template. Generally, the new templates can then be used as seeds to further find other paraphrases, but this was avoided due to the problem of semantic drift (116).

All distributional similarity algorithms mentioned so far use a symmetric similarity measure, which is more appropriate for paraphrasing than for entailment. However, directional similarity measures that employ distributional information can also be devised. Almost all directional distributional similarity approaches are based on the intuition that semantically-general predicates occur in more contexts than semantically-specific predicates. Thus, if the contexts of a predicate u are properly included in the contexts of a predicate v, then this might imply that $u \Rightarrow v$. Geffet and Dagan (67) suggested a concrete implementation of this idea, focusing on entailment between lexical items. Bhagat et al. designed the LEDIR algorithm (21), which first utilizes a

symmetric similarity measure, but then attempts to recognize the true directionality of each predicative entailment rule based on the number of contexts with which the LHS and RHS of the rule occur, assuming that more general predicates occur in more contexts. They use the Same element representation as in the DIRT algorithm, but features are based on the *semantic classes* of the arguments.

Szpektor and Dagan (172) also proposed a directional distributional similarity measure for predicates, but also modified the predicate representation. Instead of using binary propositional templates as elements, Szpektor and Dagan represented predicates with *unary propositional templates*, which contain a predicate and a single argument, such as: *'X* $\xleftarrow{subj}$ *buy'*. Szpektor and Dagan explained that unary templates are more expressive than binary templates, and that some predicates, *e.g.,* intransitive verbs, can only be encoded using unary templates. They implemented a directional similarity measure proposed by Weeds and Weir (183), again assuming that if for two templates $u \Rightarrow v$, then relatively many of the features (noun arguments in this case) of u should be covered by the features of v:

$$Cover(u,v) = \frac{\sum_{f \in F^u \cap F^v} w^u(f)}{\sum_{f \in F^u} w^u(f)} \tag{2.3}$$

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

$$BInc(u,v) = \sqrt{Lin(u,v) \cdot Cover(u,v)} \tag{2.4}$$

This average is performed since employing the *Cover* similarity measure alone promotes rules in which the LHS is very rare. Kotlerman et al. (96) recently suggested another directional similarity measure, termed *BAP*, and demonstrated it outperforms previously suggested directional measures.

Last, Schoenmackers et al. (153, 154) presented an approach for learning predicative entailment rules using a directional measure that is fundamentally different in several ways. First, the syntactic representation of propositions is much more shallow – no parsing is performed and binary propositions are simply represented as tuples of strings *(*argument$_1$,predicate,argument$_2$), or *pred(arg$_1$,arg$_2$)* for short. However, arguments are typed, that is, variables may be restricted to be, for example, some type of *country*, *disease*, *profession*, etc. Hence, a propositional template is a *typed predicate* described as *pred(X_{Type_1}, Y_{Type_2})*.

Second, rule representation is inspired by ideas from Inductive Logic Programming (122, 139). In this framework, the LHS of the rule may be composed of a conjunction of propositional templates (also known as *horn clauses*. A rule for example might state that if a company is headquartered in a city, and the city is located in some state, then this implies that the company is based in that state. Such a rule can be denoted by *'IsHeadquarteredIn($X_{Company}$,Y_{City}) ∧ IsLocatedIn(Y_{City},Z_{State}) ⇒ IsBasedIn($X_{Company}$,Z_{State})'*. A similar representation have also been recently proposed in the NELL project (33).

Last, the feature vector representation of Schoenmackers et al. differs from the vectors of *DIRT* and *BInc*. A feature in their work is a *pair* of arguments (*e.g., (Microsoft,Redmond)*), as opposed to most prior work where a separate similarity score is computed for each argument, effectively decoupling the arguments from one another. Although this decoupling alleviates sparsity problems, it disregards an important piece of information, namely the co-occurrence of arguments. For example, if one looks at the following propositions: *'coffee increases blood pressure'*, *'coffee decreases fatigue'*, *'wine decreases blood pressure'*, *'wine increases fatigue'*, one can notice that the predicates occur with similar arguments and might mistakenly infer that *'decrease ⇒ increase'*. However, looking at pairs of arguments reveals that the predicates do not share a single pair of arguments. Schoenmackers et al. prefer to use pairs of arguments as features since the data they work with is a large web-based corpus. We note that this type of feature representation is also shared by Szpektor et al.'s web-based method TEASE (175), which uses argument pairs, as well as the LEDIR algorithm, which uses pairs of semantic classes.

Table 2.1 summarizes the characteristics of most of the distributional similarity methods presented so far. The table illustrates that various methods for both paraphrasing as well as directional entailment have been suggested, that many possible representations are possible for both the elements and the features, and that most distributional similarity methods fall under the unsupervised learning paradigm.

2.2.1.2 Co-occurrence methods

Despite the effort put into developing *directional* distributional similarity methods, still often many rules are learned where the direction of entailment is erroneous or an entirely different semantic relation holds between the elements. Co-occurrence methods try to

Method	Task	Element rep.	Feature rep.	Learning
Lin 98'	⇔	words	syntactic	unsupervised
Pasca 05'	⇔	lexical phrases	lexical phrases	unsupervised
Bhagat 08'	⇔	pos-tagged phrases	noun arguments	unsupervised
DIRT 01'	⇔	syntactic	noun arguments	unsupervised
TEASE 04'	⇔	syntactic	pairs of noun phrases	bootstrapping
Geffet 05'	⇒	words	syntactic	unsupervised
LEDIR 07'	⇒	syntactic	semantic classes / pairs of classes	unsupervised
BInc 08'	⇒	syntactic	noun arguments	unsupervised
BAP 10'	⇒	words	syntactic	unsupervised
Schoenmackers 10'	⇒	typed predicate	pairs of noun arguments	unsupervised

Table 2.1: Summary of characteristics of distributional similarity methods for learning paraphrasing and entailment. The task of *paraphrasing* is denoted by '⇔' and the task of *entailment* is denoted by '⇒'. The details on each of the methods are given in the body of the section.

complement distributional similarity by tapping onto a different source of information – instead of comparing the typical contexts in which a pair of elements appear in a large corpus, co-occurrence methods focus on the co-occurrence of the pair of elements in a local scope such as a sentence or a document. For example, from the sentence *'He scared and even startled me'* one might infer that *'startle'* is semantically stronger than *'scare'* and thus '*startle* $\Rightarrow$ *scare*'. Naturally, the flip side of the same coin is that co-occurrence methods often suffer from low coverage.

Learning semantic relations using co-occurrence was first articulated in Hearst's seminal paper (81) on automatic acquisition of the *hyponym* or *is-a* relation between nouns from large corpora. The key insight was that the semantic relation is manifested in template patterns, for example the patterns *'NP_y such as NP_x'* or *'NP_x or other NP_y'* often imply that NP_x is a kind of NP_y. Note that the information conveyed in such patterns is clearly *directional*. Since Hearst's work many researchers expanded her work in various ways to automatically acquire the hyponymy relation (32, 40, 97, 185), but also other relations such as *meronymy* (20, 68) and even bio-medical relations between genes and proteins (63).

Snow et al. (164) proposed an important extension to Hearst's work on learning hyponyms. Instead of using a few dozens of manually-constructed patterns, they took advantage of tens of thousands of patterns and learned each pattern's importance with a linear supervised classifier. For every pair of nouns, they extracted all sentences in which the nouns co-occur, and then used the syntactic patterns linking the nouns as features (represented as dependency paths). Hence, a pair of nouns is represented by a feature vector in which each entry counts the number of times some syntactic pattern links the pair of nouns. To generate a training set for the classifier they employed "distant supervision", that is, positive and negative examples were generated automatically using some lexicographic resource, in their case WordNet. A pair of nouns is considered a positive example if they are hyponyms in WordNet, and a negative example if both nouns exist in WordNet and neither one is the ancestor of the other in the hyponymy hierarchy. A classifier is trained on the training set and then used to classify new unseen pairs of nouns. In our work, we will also represent pairs of predicates as feature vectors and employ distant supervision.

Co-occurrence based methods are more suited for nouns than for verbs, since nouns tend to co-occur in sentences more often than verbs. Nevertheless, Chklovsky and

Semantic relation	Example pattern
similarity	V_1 i.e. V_2
strength	V_2 and even V_1
enablement	to V_2 by V_1ing the
antonymy	either V_1 or V_2
happens-before	to V_1 and subsequently V_2

Table 2.2: The five pattern groups suggested by Chklovsky and Pantel and an example pattern for each group.

Pantel (39) used lexical patterns to discover semantic relations between *verbs* in their system *VerbOcean*. Similar to Hearst, they manually constructed 33 patterns divided into five pattern groups, where each group corresponds to a different semantic relation (some relations are symmetric and some directional): *similarity*, *strength*, *antonymy*, *enablement* and *happens-before*. Table 2.2 specifies the five pattern groups and provides one example pattern for each group. Pairs of verbs were classified to semantic relations according to their frequency of occurrence in each of the 33 patterns. Since verbs rarely instantiate lexical patterns in a sentence, they used the web as their corpus.

An alternative for pattern-based methods that improves coverage is to adopt a more "relaxed" notion of co-occurrence. Concretely, instead of looking for events where a pair of verbs instantiates a lexical pattern, one can take advantage of any event of co-occurrence in the same sentence and encode information about the relation between the verbs with more loose features. Tremper (176) suggested a supervised method for learning *presupposition* (a subtype of entailment) with features such as the distance between verbs in a sentence, the part-of-speech tags preceding and following the verbs, the tense and aspect of the verb, etc.

An even more loose notion of co-occurrence is to utilize the co-occurrence of a pair of verbs in the same *document*. Since the fact that a verb occurs often with another verb in a document does not say much about the semantic relation between them, usually similarity is computed by checking whether the pair of verbs share arguments. In other words, this approach can be viewed as a blend of both distributional similarity and co-occurrence since we check whether a pair of verbs co-occurs in a document

Method	Task	Elements	Information	Learning
Hearst 92'	hyponymy	nouns	constructed patterns	unsupervised
Berland 99'	meronymy	nouns	constructed patterns	unsupervised
Snow 05'	hyponymy	nouns	learned patterns	distant supervision
VerbOcean 04'	verb relations	verbs	constructed patterns	unsupervised
Tremper 10'	presupposition	verbs	sentence-level features	supervised
Pekar 06'	event entailment	predicates	document-level	unsupervised
Chambers 08'	narrative schemes	predicates	document-level	unsupervised
Weisman 12'	entailment	verbs	sentence- and document-level	supervised

Table 2.3: Summary of characteristics of co-occurrence methods for learning semantic relations. The column 'Elements' specifies the elements for which the relation is learned, and the column 'Information' specifies the type of information used to deduce the semantic relation. Details are given in the body of the section.

and appears with similar arguments. Pekar (134) suggested an unsupervised approach along these lines for detecting entailment, while focusing on co-occurrence in the same paragraph. Chambers and Jurafsky (34, 35) took advantage of co-occurrence at the level of documents to automatically construct narrative chains such as *'A search B* $\Rightarrow$ *A arrest B* $\Rightarrow$ *B is convicted* $\Rightarrow$ *B is sentenced'*.

Last, recently Weisman et al. (184) combined co-occurrence cues at both the sentence level and document level with distributional similarity information and trained a supervised classifier that determines verb entailment.

Table 2.3 summarizes the characteristics of most of the presented co-occurrence methods. The Table illustrates that co-occurrence methods have been used for learning a myriad of semantic relations between many types of elements, and that in recent years researchers have moved from manually constructing patterns to more sophisticated ways

of exploiting co-occurrence information.

2.2.1.3 Combinations methods

.

Naturally, combining information from lexicographic resources, distributional similarity, and co-occurrence-based methods is an appealing research direction, as these information sources are orthogonal and may complement one another.

Mirkin et al. (120) extracted entailment rules between nouns based on both distributional similarity and Hearst patterns. The two sources of information were combined by encoding them as features in a supervised classification scheme. Pennacchiotti and Pantel (135) expanded Mirkin et al.'s work and augmented their features with additional features gathered from a large web crawl and query logs for the task of Entity Extraction. Hagiwara et al. (74) trained a supervised classifier for the task of paraphrasing, combining thousands of both distributional similarity and co-occurrence-based features.

Szpektor and Dagan (173) combined lexicographic resources with distributional similarity in an unsupervised setting. They constructed a resource, termed *Argument-mapped WordNet (AmWN)*, which combines information from WordNet with information on mapping of arguments provided by Nomlex. Rules generated in the resource were verified using distributional similarity methods.

Yates and Etzioni (191) identified synonymous predicates by combining distributional similarity information with *string similarity* in an unsupervised probabilistic framework. To compare strings that represent predicates they utilized the Levenshtein edit-distance metric (43).

In this work we develop a component that takes advantage of *local* information. As we present in subsequent chapters, we combine various distributional similarity methods, lexicographic resources, and also string similarity metrics.

2.2.2 Global learning

As explained in Section 1.4, all methods reviewed so far focus on *local learning*, that is, given a pair of "element" (x, y) and a target semantic relation R the task is to determine whether $(x, y) \in R$. In global learning the input is a set of elements $\mathcal{X}$ and the target semantic relation R, and the task is to find all pairs for which the relation holds, that is, the set $\{(x, y)|x, y \in \mathcal{X}, (x, y) \in R\}$. Global learning allows joint

learning of semantic relations, namely deciding that $(x, y) \in R$ may have an effect on the decision whether some $(w, z) \in R$. In this dissertation the elements we focus on are predicative templates (or propositional templates) and the relation in question is textual entailment. The main (but not single) structural property that we will explore and exploit is transitivity, that is, the fact that '$x \Rightarrow y$' and '$y \Rightarrow z$' implies '$x \Rightarrow z$'.

The structural property of transitivity has been investigated in adjacent research area in NLP. For example, *hyponymy* is a directed transitive relation where the elements are *nouns* or *noun phrases*. Snow et al. (165) proposed an algorithm for learning hyponymy taxonomies that takes transitivity into account. Finkel and Manning (60) explored the co-reference relation, which is an undirected transitive relation between noun phrases. Yates and Etzioni (191) learned the undirected transitive *synonymy* relation between nouns and predicates. Poon and Domingos (136, 137) utilized transitivity in the context of semantic parsing, in which they learned the transitive synonymy, hyponymy and troponymy relations. Last, ordering a sequence of events through time is also a directed transitive relation in which the elements are the events, and was explored by Ling and Weld (106). Importantly, it is quite natural to describe all of the above relations as a graph in which the nodes represent the elements and the edges represent the target semantic relation. As mentioned, in the case of entailment we term such graphs *entailment graphs*.

Using a graph for global learning of predicative entailment rules rather then local learning has attracted relatively little attention. One exception is the work of Szpektor and Dagan (173), who presented the resource *Argument-mapped WordNet*, providing entailment relations for predicates in WordNet, as explained in Section 2.2.1.3. Their resource makes simple heuristic use of WordNet's global graph structure: new rules are suggested by transitively chaining graph edges, and then verified using distributional similarity measures. Effectively, this is equivalent to using the intersection of the set of rules derived by this transitive chaining and the set of rules in a distributional similarity knowledge base.

Incorporating global knowledge about transitivity incurs a great toll of efficiency. Specifically, we show in Chapter 3 that finding the optimal set of edges (representing a semantic relation) that respect transitivity in both directed and undirected graphs is NP-hard. Next, we provide some details on the main approaches suggested in prior work for tackling the efficiency problem.

Snow et al. (165) learned a semantic taxonomy that describes the hyponymy relation. They formulated an optimization problem and devised a greedy polynomial hill-climbing optimization procedure that enforces the transitivity constraint. As the problem is NP-hard, the taxonomy produced by the algorithm is not guaranteed to be optimal. We describe their method briefly here since we implement it and compare to it in Chapter 3.

Snow et al. define a taxonomy T to be a set of pairs of words, expressing the hyponymy relation. The notation $H_{uv} \in T$ means that noun u is a hyponym of the noun v in T. They define D to be the set of observed data over all pairs of words, and define $D_{uv} \in D$ to be the observed evidence we have in the data for the event $H_{uv} \in T$. They also assume a model exists for inferring $P(H_{uv} \in T|D_{uv})$: the posterior probability of the event $H_{uv} \in T$, given the data. Their goal is to find the taxonomy that maximizes the likelihood of the data, that is, to find:

$$\hat{T} = \underset{T}{\operatorname{argmax}}\, P(D|T) \tag{2.5}$$

Using some independence assumptions and Bayes rule, the likelihood $P(D|T)$ is expressed:

$$P(D|T) = \prod_{H_{uv} \in T} \frac{P(H_{uv} \in T|D_{uv})P(D_{uv})}{P(H_{uv} \in T)} \cdot \prod_{H_{uv} \notin T} \frac{P(H_{uv} \notin T|D_u v)P(D_{uv})}{P(H_{uv} \notin T)} \tag{2.6}$$

Crucially, they demand that the taxonomy learned respects the constraint that hyponymy is a transitive relation. To ensure that, they propose the following greedy algorithm: at each step they go over all pairs of words (u, v) that are not in the taxonomy, and try to add a single hyponymy relation H_{uv}. Then, they calculate the set of relations S_{uv} that H_{uv} will add to the taxonomy due to the transitivity constraint. Last, they choose to add that set of relations S_{uv} that maximizes $P(D|T)$ out of all the possible S_{uv} candidates (corresponding to all (u, v) candidates for adding to the taxonomy). This iterative process stops when $P(D|T)$ starts dropping. Their implementation of the algorithm uses a hyponym classifier presented in an earlier work (164) as a model for $P(H_{uv} \in T|D_{uv})$ and a single *sparsity* parameter $k = \frac{P(H_{uv} \notin T)}{P(H_{uv} \in T)}$.

We note that although the algorithm is polynomial, it is not very efficient. Let n denote the number of nouns in the taxonomy. In each iteration we go through $O(n^2)$

candidate edges and for each one compute the set of edges that will be added due to transitivity. The number of such iterations in worst case is $O(n^2)$, which results in bad performance. Practically, graphs are usually sparse and the stopping criterion is met relatively quickly, but still this approach of finding the best edge to add at each iteration limits scalability.

Snow et al. tailored an optimization procedure for their problem. A similar approach was taken by Yates and Etzioni, who focused on the symmetric *synonymy* relation. It is important to note that learning an undirected (symmetric) transitive relation is equivalent to *clustering*. Indeed, Yates and Etzioni propose their own variant of greedy agglomerative clustering whose main benefit is its scalability to very large data. Again, as the problem is NP-hard the solution they provide is not guaranteed to be optimal.

2.2.2.1 Integer Linear Programming

Instead of tailoring an algorithm for every problem, a different approach is to formulate the problem in some standard framework, since this allows the use of off-the-shelf optimization packages and other general standard techniques. One such framework, which we ourselves employ in Chapter 3, is *Integer Linear Programming (ILP)*.

A *Linear Program (LP)* is an optimization problem, where a linear objective function is minimized (or maximized) under linear constraints.

$$\min_{x \in \mathbb{R}^d} c^\top x \quad \text{s.t.} \quad Ax \leq b \tag{2.7}$$

where $c \in \mathbb{R}^d$ is a coefficient vector, and $A \in \mathbb{R}^n \times \mathbb{R}^d$ and $b \in \mathbb{R}^n$ specify the constraints. In short, we wish to find the optimal assignment for the d variables in the vector x, such that all n linear constraints specified by the matrix A and the vector b are satisfied by this assignment. If the variables are forced to be integers, the problem is termed an *Integer Linear Program (ILP)*. The ILP formulation has attracted considerable attention recently in several fields of NLP, such as semantic role labeling, summarization and parsing (4, 42, 52, 60, 113, 144, 148).

In the context of transitivity, finding the best set of transitive relations can be performed by formulating some desired linear objective function and expressing transitivity constraints as integer linear constraints. This was performed by Finkel and Manning (60) in the context of co-reference resolution. The advantages of such an approach is

that formulating problems as an ILP is relatively simple, and as ILP is well understood nowadays, state-of-the-art packages such as Gurobi[1] and CPLEX[2] implement branch-and-bound methods, which solve exactly (rather than approximately) the problem and perform well on small inputs

The main disadvantage of using an ILP solver to find an exact solution is that this is a classical NP-hard problem and so scalability is inherently limited. In fact, Smith (163) explains that ILPs are a good way to declaratively formulate a problem, but often directly utilizing an ILP solver is not feasible. Consequently, there has been a lot of work in the field of optimization on finding efficient good-quality approximate solutions for ILP. Two relevant approaches that have been successfully applied recently in NLP are *LP relaxation* and *dual decomposition*.

In LP relaxation, the constraint that variables must be integers is dropped, transforming the problem from an ILP to an LP, which is polynomial. An LP solver is then applied to the problem, and variables that are assigned a fractional value are rounded to their nearest integer. An advantage of LP relaxation is that if all variables are assigned an integer value by the LP solver, then this is guaranteed to be the optimal solution. A disadvantage is that when rounding fractional values to the nearest integer constraint violations can easily occur. Martins et al. (113) successfully applied LP relaxation in the context of dependency parsing. In Chapter 4 we will empirically investigate an application of LP relaxation to our problem of learning entailment graphs.

Dual decomposition, also known as *Lagrangian relaxation* (150), is a framework for deriving algorithms that solve complex inference problems, which is strongly connected to LP relaxation. In dual decomposition a complex problem is decomposed into simple sub-problems, these sub-problems are efficiently solved and then combined to form a global solution. The main two advantages of dual decomposition are first, that the approximation algorithm provides a *certificate* whenever it is able to reach the optimal solution, and second, that it has been shown empirically that for a variety of NLP tasks it reaches the optimal solution in more than 90% of the inputs. Dual decomposition has been successfully applied in parsing (95), machine translation (37, 149), bio-medical information extraction (145), and more.

[1] `http://www.gurobi.com/`

[2] `http://www-01.ibm.com/software/integration/optimization/cplex-optimizer/`

In our work we formulate the problem of learning entailment graphs as an ILP (Chapter 3). In Chapters 3 and 4 we discuss the usage of standard off-the-shelf ILP solvers and also suggest algorithms that improve efficiency and scalability by taking advantage of the specific structural characteristics of entailment graphs.

A possible alternative for Integer Linear Programming is *Markov Logic Networks (MLN)* (53). While in ILP integer constraints are *hard*, that is, they are not allowed to be violated, the basic idea of Markov logic is to allow for soft or probabilistic constraints, which are common in real life[1]. This is performed by having non-negative weights assigned to each constraint, where a high weight means that violating the corresponding constraint will incur a high "penalty" and a low weight corresponds to a low "penalty". An MLN provides a probability distribution over all possible "worlds", or solutions, and the goal is to find the best one.

Another fundamental difference between MLN and ILP is that representation of constraints in MLNs is based on first-order logic. This is important since first-order logic is more expressive than propositional logic, which is known to be equivalent to ILP (85), and hence using first-order logic increases the expressiveness of the framework. Specifically, first-order logic allows for reasoning about entire classes of objects at once, also known as *lifted reasoning*. The advantage is both representational – it is simpler to state once that some property should hold for an entire class rather than to specify it separately for each and every class member, but in addition reducing the number of constraints can potentially improve efficiency (71). Nevertheless, learning the weights of MLNs and performing inference is a computationally hard problem and scalability remains an open research issue.

Expressing transitivity constraints in Markov Logic Networks was performed in the tasks of unsupervised semantic parsing (136, 137) as well as temporal information extraction (106).

Last, we note that on top of ILPs and MLNs, other general formalisms exist that can model transitivity. For example, the problem of learning entailment rules under transitivity can be formulated as a Markov Random Field (MRF) and standard approximation algorithms such as loopy belief propagation (133) can be applied to solve

[1]It is possible to have soft constraints in ILP (148), however MLNs are more naturally geared towards probabilistic constraints.

the problem . However, we are unaware of any prior attempts to model transitivity in such a manner in the field of NLP.

2.3 Context-sensitive entailment rules

All of the methods presented so far disregard a fundamental property of natural language – the fact that it is ambiguous and context-sensitive. For example, the rule '*acquire* ⇒ *learn*' is true in the context of *languages* but incorrect in the context of *purchases*. As mentioned in Section 1.4, this issue is mostly sidestepped in our work, however, recently interest in the topic of modeling context in natural language has substantially increased (51, 121, 126, 130, 143, 146, 155, 174, 178), and therefore we provide a brief overview.

One major challenge for inference systems is to determine given an entailment rule applied in some context whether that rule is appropriate in that particular context. Szpektor et al. (174) suggested a comprehensive framework that models this scenario. They argue that generally inference systems are faced with three types of objects: a text T, a hypothesis H, and an inference rule r. Each of these objects must have some representation expressing its typical contexts, and for inference to hold it is important that the contexts of T and H match, and also that a rule r applied over T to infer H will match this context[1]. Szpektor et al. presented an implementation of their framework that is based on unsupervised context models, and demonstrated that by considering context during rule application they are able to improve performance in an information extraction task. Later, Mirkin et al. (121) presented another implementation of their framework based on a classification-based model and showed it leads to improved performance in the task of text classification.

Other works also modeled the context of inference rules in order to utilize them more precisely in applications. Pantel et al. (130) considered predicative entailment rules, such as '*X charge Y* ⇒ *X accuses Y*', and learned for each rule a representation describing the arguments that may instantiate the variables. A rule is applied only in contexts where the arguments match that representation. For instance, in the above example *'Y'* should be a person rather than an electrical appliance. Following, Ritter et al. (146) constructed a probabilistic representation for arguments that is based

[1]Szpektor et al. explain that context matching is directional, but for simplicity we skip the details.

on *Latent Dirichlet Allocation (LDA)* (23) and demonstrated that this representation outperforms Pantel et al.'s approach in the task of filtering erroneous rule applications.

The issue of proper context representation for words and phrases has been explored in the field of computational semantics in general. Many context models have been recently developed, and such models can be adapted for semantic inference applications. Dinu and Lapata (51) and also later Van de Cruys et al. (178) suggested to represent the meaning of words as a probability distribution over latent "senses" that are learned in an unsupervised manner using either LDA or *Non-negative Matrix Factorization (NMF)* (98), and showed this representation is useful in the task of assessing the similarity of words in context. Reisinger and Mooney represented the meaning of words as a set of vectors, where each vector corresponds to a different sense or context. Ó Séaghdha (126) developed a model that is highly similar to Ritter et al.'s but rather than evaluating in the framework of textual inference, utilized the model to judge the plausibility of propositions.

Nevertheless, one important aspect of context-sensitive entailment rules has been neglected so far, namely, consideration of context during entailment rule learning. In the methods above, the first step is to learn inference rules that are accompanied by some score using distributional similarity, and subsequently to construct a context representation for each rule. However, distributional similarity algorithms learn predicative entailment rules by comparing the arguments that occur with the different predicative templates. Thus, if a template has multiple senses then the arguments it occurs with are a mix corresponding to its various senses, and consequently the score learned will be some biased "average" over the senses. This raises the interesting idea of *context-sensitive rule learning*, and we provide some more details on this potential future direction in Chapter 7.

2.4 Relation extraction

Relation extraction is the task of identifying and extracting instances of a certain relation in running text. For instance, given the *'acquire'* relation we would like to find sentences that describe acquisition events and specify the "acquirer", "acquiree", and the amount of money paid. As we saw in Chapter 1, relation extraction can be reduced to textual entailment by looking for sentences that entail templates such

as *'$X_{company}$ acquire $Y_{company}$ for Z_{money}'*. Therefore, learning to perform relation extraction can be viewed as learning predicative entailment rules where the RHS of the rule expresses the target relation. For instance, in the example above relevant rules would have as their RHS the templates *'$X_{company}$ acquire $Y_{company}$'* or *'acquire $Y_{company}$ for Z_{money}'*. The two conceptual differences from methods hitherto presented are that first, when learning predicative entailment rules we are interested in all valid rules without specifying the RHS whereas in Relation Extraction we are interested only in rules whose RHS expresses a certain semantic relation, and second, relation extraction is more focused on annotating instances of the relation in text. However, progress in research in recent years has made the seemingly different fields of relation extraction and entailment rule learning closer to one another. Therefore, we dedicate this section to a short overview of dominant relation extraction methods.

Supervised relation extraction In supervised approaches, we are given a set of positive and negative examples. A positive example is usually a sentence annotated with a pair of phrases that participate in the target relation. A negative example is a sentence annotated with a pair of phrases that do not participate in the target relation. In some approaches, sentences are pre-processed and then each example is converted into a feature vector that includes manually-engineered features such as the lemmas and part-of-speech tags between and around the annotated phrases, the semantic types of the phrases, the distance between the phrases, etc. Subsequently, a supervised classifier is trained and used at test time on unseen sentences (91, 197).

Other researchers chose the kernel approach, in which features are not engineered explicitly, and instead only kernel functions $K(x, y)$ are defined and combined. The goal of a kernel function is to estimate the similarity of the objects x and y, which may be the word sequence between (or to the left and right of) the pair of relation phrases, the dependency path between the relation phrases, etc. Once this function is defined it implicitly defines a feature vector in high dimensional space for each of the training examples and then an SVM classifier can be utilized for training and classification (29, 69). Empirically, kernel approaches outperformed feature approaches, but the computational complexity is higher.

An important disadvantage of supervised approaches is that for each target relation we would like to extract we must manually annotate example sentences. Also note that

supervised approaches to not explicitly identify the predicate in the sentence evoking the target relation, whereas in textual entailment the predicate needs to be matched against the LHS of a relevant entailment rule.

Semi-supervised relation extraction Semi-supervised learning aims to reduce the burden of manual annotation for each relation by requiring only a few seed examples as input. This is accomplished by the general framework of *bootstrapping*, as described in Algorithm 1. The input to bootstrapping algorithms is merely a few pairs of seed phrases (assuming the relation is binary) for which the target relation holds. For example, if the target relation is *'(author,title)'*, then an input seed might be *'(Arthur Conan Doyle, The Adventures of Sherlock Holmes')*. At the first step, we look for sentences that contain both phrases, as in Examples 2.8a and 2.8b. In the second step, the labeled examples are generalized by some pattern. For instance, the pattern *'(Sir,#1,wrote,#2,in)'* generalizes Example 2.8 and specifies that if a sentence has the word *'Sir'* followed by a phrase, followed by the word *'wrote'*, followed by a second phrase, followed by the word *'in'*, then the target relation should hold between the pair of phrases. Next, sentences that match the pattern can be used to find new pairs of phrases participating in the target relation (*e.g.,* the above pattern will match the sentence *'Sir William Scott wrote Ivanhoe in 1820'*), and the new pairs of phrases can be used to label new examples, and so on. The main challenge in bootstrapping is to generalize the patterns in a way that acquires new seed pairs of phrases on the one hand, but on the other hand does not introduce too much noise, since noise tends to propagate and increase from one iteration to another.

(2.8) (a) *... know that Sir Arthur Conan Doyle wrote The Adventures of Sherlock Holmes in 1892.*

(b) *... when Sir Arthur Conan Doyle wrote The Adventures of Sherlock Holmes in March 1892, he was...*

Bootstrapping algorithms have existed for quite a while now. The DIPRE algorithm (27) was suggested as a general algorithm for relation extraction and applied to extract the *'(author,title)'* relation. The Snowball algorithm (2) was employed to extract the binary *'located-in'* relation, and made use of the semantic classes *'LOCATION'* and *'ORGANIZATION'* to improve precision and reduce the number of seeds required.

Algorithm 1 A general description of bootstrapping for binary relation extraction.

Input: A target relation R, a seed set of pairs of phrases $(e_1, e_2) \in R$, a corpus C

1: Use seed examples to label sentences in C.
2: Induce generalized patterns from the labeled data
3: Apply the patterns on unlabeled sentences in C
4: Find new examples $(e_1^*, e_2^*) \in R$ and add them to the seed set.
5: Return to step 1 until some convergence criterion is met.

Espresso (131) further extended previous algorithms by suggesting better weighting schemes for assessing the confidence of extracted patterns and phrases. Bootstrapping algorithms such as KnowItAll (56) and Kozareva et al.'s (97) were also used to extract the taxonomic *hyponymy* relation, which is strongly related to textual entailment, and as mentioned in Section 2.2.1.1, the TEASE algorithm (Szpektor et al. (147, 175)) used bootstrapping over automatically-generated seeds to extract paraphrases.

Similar to supervised relation extraction, in bootstrapping the target relation is defined by example pairs of phrases and not by an actual predicate. However, in contrast to supervised relation extraction, the extracted patterns usually include the predicate that evokes the target relation. Thus, each pair of patterns may be viewed as a bi-directional entailment rule or a paraphrase rule.

Distantly-supervised relation extraction .

A disadvantage of both supervised methods as well as bootstrapping algorithms is that they focus on a rather small set of target relations, since for each relation some form of manual intervention is required. A possible solution for that is to take advantage of existing manually-constructed knowledge-bases to automatically generate training examples that will be later employed by a supervised learner (this is referred to often as *distant supervision* or *weak supervision*). For example, many Wikipedia pages contain tables, termed *infoboxes*, that summarize some aspects of the article. In many cases, if the article is about a person, the infobox will contain a field *'born'* specifying the date of birth of that person. For instance, the infobox in the page about William Shakespeare specifies that he was born in 1564. Subsequently, one can look for sentences containing the phrases *'William Shakespeare'* and *'1564'* and automatically generate

training examples for the relation *'born-in'*. Since infoboxes contain thousands of fields that are repeated in many Wikipedia pages, one can automatically train thousands of classifiers for thousands of relations.

The approach of using knowledge-bases to automatically label examples has gained popularity in recent years. Both Wikipedia and Freebase[1] have been used to label examples in systems such as Kylin (187), Luchs (83) and WOE (188). The information generated from these knowledge-bases was later integrated into sophisticated graphical models that allow to handle the uncertainty in the labeling process and relax assumptions about the independence between different relations (82, 170, 190). These developments draw the fields of learning predicative entailment rules and relation extraction nearer to one another. In other words, a possible alternative for learning predicative entailment rules, in which the LHS and RHS of the rules are unknown, is to automatically learn thousands of relation extractors, where each one corresponds to an entailment rule in which the RHS is predetermined. Nevertheless, we are unaware of any attempts to explicitly use state-of-the-art relation extraction techniques to generate a large scale resource of predicative entailment rules.

"Open" relation extraction One last form of relation extraction, in which the task is slightly different, is *open information extraction*. In open information extraction no target relations are defined. Instead, the task is, given a large domain-independent corpus of sentences (often web-based), to extract as many triples of the form *'(arg$_1$,predicate,arg$_2$)'* as possible. However, an open information extraction system does not know when two different predicates that appear in different triples are semantically-related. Three well-known open information extraction systems are TextRunner (7), REVERB (57) and OLLIE(114). A massive triple database that is the output of open information extraction systems can be used as input for systems that learn entailment relations between the extracted predicates and arguments (146, 153, 191)[2]. In Chapter 5, we will present a method for learning a large-scale predicative entailment rules knowledge-base from such triples.

[1] `http://www.freebase.com/`

[2] These works are often also considered to be part of the open information extraction effort.

3

Global Graph Model

In this Chapter we present a global model for learning predicative entailment rules, which utilizes the transitivity property of entailment rules (discussed in Chapters 1 and 2). First, We define a graph structure over propositional templates that represents entailment relations as directed edges. Then, we use a global transitivity constraint on the graph to learn the optimal set of edges, formulating the optimization problem as an Integer Linear Program. The algorithm is applied in a setting where given a target concept, the algorithm learns on-the-fly all entailment rules between predicates that co-occur with this concept. Focusing on a target concept substantially reduced the problem of predicate ambiguity, and results show that our global algorithm improves performance over local and global baseline algorithms by more than 10%.

3.1 Entailment Graph

We now formally define a structure termed **entailment graph** that describes the entailment relations between propositional templates (Section 3.1.1),and a specific type of entailment graph, termed **focused entailment graph**, that concentrates on entailment relations that are relevant for some pre-defined target concept (Section 3.1.2).

3.1.1 Entailment graph: definition and properties

The nodes of an entailment graph are **propositional templates**. A propositional template in our work is similar to the binary templates of DIRT, that is, a dependency

path between two arguments that passes through the predicate[1]. However, while in DIRT the template contains two variable X and Y, we allow in our model one of the arguments to be instantiated. In addition, we assume that the sense of the predicate is specified (according to some sense inventory, such as WordNet) and so each sense of a polysemous predicate corresponds to a separate template (and a separate graph node). For example, *'X* $\xleftarrow{subj}$ *treat#1* $\xrightarrow{obj}$ *Y'* and *'X* $\xleftarrow{subj}$ *treat#2* $\xrightarrow{obj}$ *nausea'* are propositional templates for the first and second sense of the predicate *treat*, respectively. An edge (u, v) in the graph represents the fact that template u entails template v. Note that the entailment relation transcends beyond hyponymy/troponomy. For example, the template *'X is diagnosed with asthma'* entails the template *'X suffers from asthma'*, although one is not a hyponym of the other. An example for an entailment graph is given in Figure 3.1.

Since entailment is a transitive relation, an entailment graph is **transitive**, that is, if the edges (u, v) and (v, w) are in the graph, so is the edge (u, w). As explained in Chapter 1, the property of transitivity does not hold when the senses of the predicates are not specified. For example, *'X buy Y* $\Rightarrow$ *X acquire Y'* and *'X acquire Y* $\Rightarrow$ *X learn Y'*, but *'X buy Y* $\not\Rightarrow$ *X learn Y'*. This violation occurs since the predicate *'acquire'* has two distinct senses in the two templates, but this distinction is lost when senses are not specified.

Transitivity implies that in each strongly connected component[2] of the graph all nodes entail each other. For example, in Figure 3.1 the nodes *'X-related-to-nausea'* and *'X-associated-with-nausea'* form a strongly connected component. Moreover, if we merge every strongly connected component to a single node, the graph becomes a Directed Acyclic Graph (DAG), and a hierarchy of predicates can be obtained.

3.1.2 Focused entailment graphs

In this Chapter we concentrate on learning a type of entailment graph, termed *focused entailment graph*. Given a target concept, such as *'nausea'*, a focused entailment graph describes the entailment relations between propositional templates for which the target concept is one of the arguments (see Figure 3.1). Learning such entailment rules in

[1] We restrict our discussion to templates with two arguments, but generalization is simple.

[2] A strongly connected component is a subset of nodes in the graph where there is a path from any node to any other node in the subset.

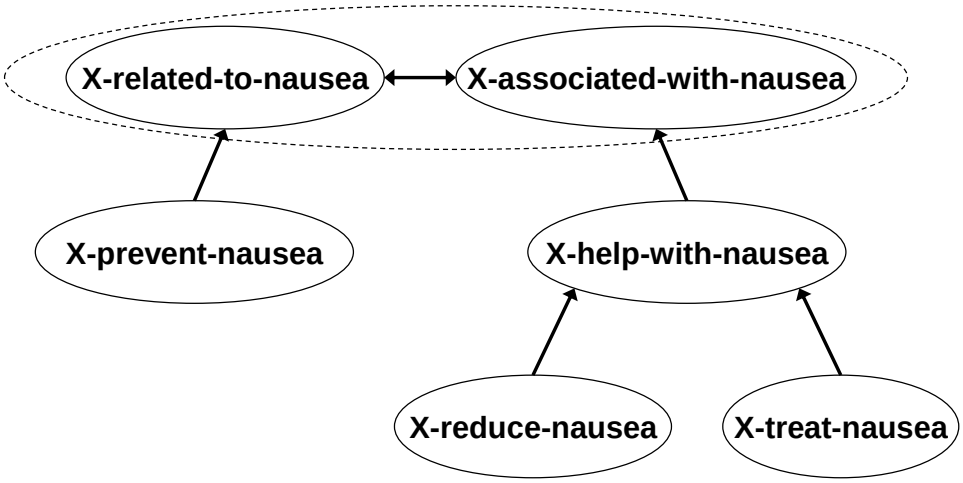

Figure 3.1: A focused entailment graph: For clarity, edges that can be inferred by transitivity are omitted. The single strongly connected component is surrounded by a dashed line.

real time for a target concept is useful in scenarios such as Information Retrieval and Question Answering, where a user specifies a query about the target concept. The need for such rules has been also motivated by Clark et al. (41), who investigated what types of knowledge are needed to identify entailment in the context of the RTE challenge, and found that often rules that are specific to a certain concept are required. Another example for a semantic inference algorithm that is utilized in real time is provided by Do and Roth (52), who recently described a system that given two terms determines the taxonomic relation between them on-the-fly. Last, in Chapter 6 we present an application that uses focused entailment graphs for *textual exploration*, that is, to present information about a target concept according to a hierarchy that is based on entailment.

The benefit of learning focused entailment graphs is three-fold. First, the target concept that instantiates the propositional template usually disambiguates the predicate and hence the problem of predicate ambiguity is greatly reduced. Thus, we do not employ any form of disambiguation in this chapter, but assume that every node in a focused entailment graph has a single sense (we further discuss this assumption

when describing the experimental setting in Section 3.4.1), which allows us to utilize transitivity constraints.

An additional (albeit rare) reason that might also cause violations of transitivity constraints is the notion of *probabilistic entailment*. While troponomy rules (58) such as *'X walk ⇒ X move'* can be perceived as being almost always correct, rules such as *'X cough ⇒ X is sick'* (this was termed *cause-effect* entailment in Section 1.3.1) might only be true with some probability. Consequently, chaining a few probabilistic rules such as $A \Rightarrow B$, $B \Rightarrow C$, and $C \Rightarrow D$ might not guarantee the correctness of $A \Rightarrow D$. Since in focused entailment graphs the number of nodes and diameter[1] are quite small (for example, in the data set we present in Section 3.4 the maximal number of nodes is 26, the average number of nodes is 22.04, the maximal diameter is 5, and the average diameter is 2.44), we do not find this to be a problem in our experiments in practice.

Last, the optimization problem that we formulate is NP-hard (as we show in Section 3.2.2). Since the number of nodes in focused entailment graphs is rather small, a standard ILP solver is able to quickly reach the optimal solution.

To conclude, the algorithm we suggest next is applied in our experiments on focused entailment graphs. However, we believe that it is suitable for any entailment graph whose properties are similar to those of focused entailment graphs. For brevity, the term *entailment graph* will stand for *focused entailment graph* in this chapter.

3.2 Learning Entailment Graph Edges

In this section we present an algorithm that given a set of propositional templates, constituting the nodes of an entailment graph, learns its edges, that is, the entailment relations between all pairs of nodes. The algorithm comprises two steps (described in Sections 3.2.1 and 3.2.2): in the first step we use a large corpus and a lexicographic resource (WordNet) to train a generic *local entailment classifier* that given any pair of propositional templates estimates the likelihood that one template entails the other. This generic step is performed only once, and is independent of the specific nodes of the target entailment graph whose edges we want to learn. In the second step we learn on-the-fly the edges of a specific target graph: given the graph nodes, we employ

[1]The distance between two nodes in a graph is the number of edges in a shortest path connecting them. The diameter of a graph is the maximal distance between any two nodes in the graph.

a global optimization approach that determines the set of edges that maximizes the probability (or score) of the entire graph. The global graph decision is determined by the given edge probabilities (or scores) supplied by the entailment classifier and by the graph constraints (transitivity and others).

3.2.1 Training an entailment classifier

We describe a procedure for learning a generic local entailment classifier, which can be used to estimate the entailment likelihood for any given pair of templates. The classifier is constructed based on a corpus and a lexicographic resource (WordNet) using the following four steps:

(a) Extract a large set of propositional templates from the corpus.

(b) Use WordNet to automatically generate a training set of pairs of templates — both positive and negative examples.

(c) Represent each training set example with a feature vector of various distributional similarity scores.

(d) Train a classifier over the training set.

(a) Template extraction We parse the corpus with the Minipar dependency parser (103) and use the Minipar representation to extract all binary templates from every parse tree, employing the procedure described by Lin and Pantel (104), which considers all dependency paths between every pair of nouns in the parse tree. We also apply over the extracted paths the syntactic normalization procedure described by Szpektor and Dagan (171), which includes transforming passive forms into active forms and removal of conjunctions, appositions and abbreviations. In addition, we use a simple heuristic to filter out templates that probably do not include a predicate: we omit "uni-directional" templates where the root of template has a single child, such as *therapy* $\xrightarrow{prep}$ *in* $\xrightarrow{p-comp}$ *patient* $\xrightarrow{nn}$ *cancer*, unless one of the edges is labeled with a passive relation, such as in the template *nausea* $\xleftarrow{vrel}$ *characterized* $\xleftarrow{subj}$ *poisoning*, which contains the Minipar passive label *'vrel'*[1]. Last, the arguments are replaced by variables,

[1]This passive construction is not handled by the normalization scheme employed by Szpektor and Dagan (171).

Positive examples	Negative examples
($X \xleftarrow{subj}$ *desire* $\xrightarrow{obj}$ *Y*, $X \xleftarrow{subj}$ *want* $\xrightarrow{obj}$ *Y*)	($X \xleftarrow{subj}$ *push* $\xrightarrow{obj}$ *Y*,$X \xleftarrow{subj}$ *blow* $\xrightarrow{obj}$ *Y*)
($X \xleftarrow{subj}$ *cause* $\xleftarrow{vrel}$ *Y*, $X \xleftarrow{subj}$ *create* $\xleftarrow{vrel}$ *Y*)	($X \xleftarrow{subj}$ *issue* $\xleftarrow{vrel}$ *Y*,$X \xleftarrow{subj}$ *sign* $\xleftarrow{vrel}$ *Y*)

Table 3.1: Positive and negative examples for entailment in the training set. The direction of entailment is from the left template to the right template.

resulting in propositional templates such as $X \xleftarrow{subj}$ *affect* $\xrightarrow{obj} Y$. The lexical items that remain in the template after replacing the arguments by variables are termed **predicate words**.

(b) Training set generation WordNet is used to automatically generate a training set of positive (entailing) and negative (non-entailing) template pairs. Let T be the set of propositional templates extracted from the corpus. For each $t_i \in T$ with two variables and a single predicate word w, we extract from WordNet the set H of direct hypernyms (distance of one in WordNet) and synonyms of w. For every $h \in H$, we generate a new template t_j from t_i by replacing w with h. If $t_j \in T$, we consider (t_i, t_j) to be a positive example. Negative examples are generated analogously, only considering direct co-hyponyms of w, which are direct hyponyms of direct hypernyms of w that are not synonymous to w. It has been shown in past work that in most cases co-hyponym terms do not entail one another (120). A few examples for positive and negative training examples are given in Table 3.1.

As we saw in Chapter 2, this generation method falls into the framework of "distant supervision", and is similar to the method proposed by Snow et al. (164) for training a noun hypernym classifier. However, it differs in some important aspects: First, Snow, Jurafsky, and Ng consider a positive example to be any Wordnet hypernym, irrespective of the distance, while we look only at direct hypernyms. This is since predicates are mainly verbs and precision drops quickly when looking at verb hypernyms in WordNet at a longer distance. Second, Snow, Jurafsky, and Ng generate negative examples by looking at any two nouns where one is not the hypernym of the other. In the spirit of "contrastive estimation" (162), we prefer to generate negative examples that are "hard", that is, negative examples that while not entailing are still semantically similar to positive examples and thus focus the classifier's attention on determining

the boundary of the entailment class. Last, we use a balanced number of positive and negative examples, since classifiers tend to perform poorly on the minority class when trained on imbalanced data (125, 179).

(c) Distributional similarity representation We aim to train a classifier that for an input template pair (t_1, t_2) determines whether t_1 entails t_2. Our approach is to represent a template pair by a feature vector where each coordinate is a different distributional similarity score for the pair of templates. The different distributional similarity scores are obtained by utilizing various distributional similarity algorithms that differ in one or more of their characteristics. In this way we hope to combine the various methods proposed in the past for measuring distributional similarity. The distributional similarity algorithms we employ vary in one or more of the following dimensions: the way the predicate is represented, the way the features are represented, and the function used to measure similarity between the feature representations of the two templates.

Predicate representation As mentioned, we represent predicates over dependency tree structures. However, some distributional similarity algorithms measure similarity between binary templates directly, (21, 104, 175, 191), while some decompose binary templates into two unary templates, estimate similarity between two pairs of unary templates, and combine the two scores into a single score (172).

Feature representation The features of a template are some function of the terms that instantiated the argument variables in a corpus. Two representations that are used in our experiments are derived from an ontology that maps natural language phrases to semantic identifiers (see Section 3.4). Another variant occurs when using binary templates: a template may be represented by a pair of feature vectors, one for each variable like in the DIRT algorithm (104), or by a single vector, where features represent pairs of instantiations (175, 191). As explained in Section 2.2.1.1, the former variant reduces sparsity problems, while Yates and Etzioni showed that the latter is more informative and performs favorably on their data.

Similarity function We consider two similarity functions: The symmetric *Lin* (104) similarity measure, and the directional *BInc* (172) similarity measure, reviewed in Section 2.2.1.1. Thus, information about the direction of entailment is provided by the *BInc* measure.

We compute for any pair of templates (t_1, t_2) twelve distributional similarity scores using all possible combinations of the aforementioned dimensions. These scores are then used as 12 features representing the pair (t_1, t_2) (A full description of the features is given in Section 3.4). This is reminiscent of Connor and Roth (45), who used the output of unsupervised classifiers as features for a supervised classifier in a verb disambiguation task.

(d) Training a classifier Two types of classifiers may be trained in our scheme over the training set: margin classifiers (such as SVM) and probabilistic classifiers. Given a pair of templates (i, j) and their feature vector F_{ij}, we denote by an indicator variable x_{ij} the event that i entails j. A margin classifier estimates a score s_{ij} for the event $x_{ij} = 1$, which indicates the positive or negative distance of the feature vector F_{ij} from the separating hyperplane. A probabilistic classifier provides the posterior probability $P_{uv} = P(x_{ij} = 1|F_{ij})$.

3.2.2 Global learning of edges

In this step we get a set of propositional templates as input, and we would like to learn all of the entailment relations between these propositional templates. For every pair of templates we can compute the distributional similarity features and get a score from the trained entailment classifier. Once all the scores are calculated we try to find the optimal graph, that is, the best set of edges over the propositional templates. Thus, in this scenario the input is the nodes of the graph and the output are the edges.

To learn edges we consider global constraints, which allow only certain graph topologies. Since we seek a global solution under transitivity and other constraints, Integer Linear Programming is a natural choice, enabling the use of state of the art ILP optimization packages. Given a set of nodes V and a weighting function $w : V \times V \rightarrow \mathbb{R}$ (derived from the entailment classifier in our case), we want to learn the directed graph $G = (V, E)$, where $E = \{(i, j)|\ x_{ij} = 1\}$, by solving the following Integer Linear Program over the variables x_{ij}:

$$\hat{G} = \underset{G}{\operatorname{argmax}} \sum_{i \neq j} w_{ij} \cdot x_{ij} \tag{3.1}$$

$$\text{s.t. } \forall_{i,j,k \in V} \; x_{ij} + x_{jk} - x_{ik} \leq 1 \tag{3.2}$$

$$\forall_{i,j \in A_{yes}} \; x_{ij} = 1 \tag{3.3}$$

$$\forall_{i,j \in A_{no}} \; x_{ij} = 0 \tag{3.4}$$

$$\forall_{i \neq j} \; x_{ij} \in \{0, 1\} \tag{3.5}$$

The objective function in eq. 3.1 is simply a sum over the weights of the graph edges. The global constraint is given in eq. 3.2 and states that the graph must respect transitivity. This constraint is equivalent to the one suggested by Finkel and Manning (60) in a coreference resolution task, except that the edges of our graph are directed. The constraints in eq. 3.3 and 3.4 state that for a few node pairs, defined by the sets A_{yes} and A_{no} respectively, we have prior knowledge that one node does or does not entail the other node. Note that if $(i, j) \in A_{no}$, then due to transitivity there must be no path in the graph from i to j, which rules out additional edge combinations. We elaborate on how the sets A_{yes} and A_{no} are computed in our experiments in Section 3.4. Altogether, this Integer Linear Program contains $O(|V|^2)$ variables and $O(|V|^3)$ constraints, and can be solved using state of the art optimization packages.

A theoretical aspect of this optimization problem is that it is NP-hard. We can phrase it as a decision problem in the following manner: given V, w and a threshold k, we wish to know if there is a set of edges E that respects transitivity and $\sum_{(i,j) \in E} wij \geq k$. Yannakakis (189) has shown that the simpler problem of finding in a graph $G' = (V', E')$ a subset of edges $A \subseteq E'$ that respects transitivity and $|A| \geq k$ is NP-hard. Thus, we can conclude that our optimization problem is also NP-hard by the trivial polynomial reduction defining the function w that assigns the score 0 for node pairs $(i, j) \notin E'$ and the score 1 for node pairs $(i, j) \in E'$. Since the decision problem is NP-hard, it is clear that the corresponding maximization problem is also NP-hard. Thus, obtaining a solution using ILP is quite reasonable and in our experiments also proves to be efficient (Section 3.4).

Next, we describe two ways of obtaining the weighting function w, depending on the type of entailment classifier we prefer to train.

3.2.2.1 Score-based weighting function

In this case, we assume that we choose to train a margin entailment classifier estimating the score s_{ij} (A positive score if the classifier predicts entailment, and a negative score otherwise) and define $w_{score}(i,j) = s_{ij} - \lambda$. This gives rise to the following objective function:

$$\hat{G}_{score} = \underset{G}{\text{argmax}} \sum_{i \neq j} (s_{ij} - \lambda) \cdot x_{ij} = \underset{G}{\text{argmax}} \left(\sum_{i \neq j} s_{ij} \cdot x_{ij} \right) - \lambda \cdot |E| \tag{3.6}$$

The term $\lambda \cdot |E|$ is a regularization term reflecting the fact that edges are sparse. Intuitively, this means that we would like to insert into the graph only edges with a score $s_{ij} > \lambda$, or in other words to "push" the separating hyperplane towards the positive half space by λ. Note that the constant λ is a parameter that needs to be estimated and we discuss ways of estimating it in Section 3.4.2.

3.2.2.2 Probabilistic weighting function

In this case, we assume that we choose to train a probabilistic entailment classifier. Recall that x_{ij} is an indicator variable denoting whether i entails j, that F_{ij} is the feature vector for the pair of templates i and j, and define F to be the set of feature vectors for all pairs of templates in the graph. The classifier estimates the posterior probability of an edge given its features: $P_{ij} = P(x_{ij} = 1|F_{ij})$, and we would like to look for the graph G that maximizes the posterior probability $P(G|F)$. In Section 3.3 we specify some simplifying independence assumptions under which we prove that this graph maximizes the following linear objective function:

$$\hat{G}_{prob} = \underset{G}{\text{argmax}} \sum_{i \neq j} (\log \frac{P_{ij}}{1 - P_{ij}} + \log \eta) \cdot x_{ij} = \underset{G}{\text{argmax}} \sum_{i \neq j} \log \frac{P_{ij}}{1 - P_{ij}} \cdot x_{ij} + \log \eta \cdot |E| \tag{3.7}$$

where $\eta = \frac{P(x_{ij}=1)}{P(x_{ij}=0)}$ is the prior odds ratio for an edge in the graph, which needs to be estimated in some manner. Thus, the weighting function is defined by $w_{prob}(i,j) = \log \frac{P_{ij}}{1-P_{ij}} + \log \eta$.

Both the score-based and the probabilistic objective functions obtained are quite similar: both contain a weighted sum over the edges and a regularization component reflecting the sparsity of the graph. Next, we show that we can provide a probabilistic interpretation for our score-based function (under certain conditions), which will allow us to use a margin classifier and interpret its output probabilistically.

3.2.2.3 Probabilistic interpretation of score-based function

We would like to use the score s_{ij}, which is bounded in $(\infty, -\infty)$ and derive from it a probability P_{ij}. To that end we project s_{ij} onto $(0, 1)$ using the sigmoid function, and define P_{ij} in the following manner:

$$P_{ij} = \frac{1}{1 + \exp(-s_{ij})} \tag{3.8}$$

Note that under this definition the log probability ratio is equal to the inverse of the sigmoid function:

$$\log \frac{P_{ij}}{1 - P_{ij}} = \log \frac{\frac{1}{1+\exp(-s_{ij})}}{\frac{\exp(-s_{ij})}{1+\exp(-s_{ij})}} = \log \frac{1}{\exp(-s_{ij})} = s_{ij} \tag{3.9}$$

Therefore, when we derive P_{ij} from s_{ij} with the sigmoid function, we can re-write $\hat{G}_{prob}$ as:

$$\hat{G}_{prob} = \underset{G}{\operatorname{argmax}} \sum_{i \neq j} s_{ij} \cdot x_{ij} + \log \eta \cdot |E| = \hat{G}_{score} \tag{3.10}$$

where we see that in this scenario the two objective functions are identical and the regularization term λ is related to the edge prior odds ratio by: $\lambda = -\log \eta$.

Moreover, assume that the score s_{ij} is computed as a linear combination over n features (such as a linear-kernel SVM), that is, $s_{ij} = \sum_{k=1}^{n} s_{ij}^k \cdot \alpha_k$, where s_{ij}^k denotes feature values and α_k denotes feature weights. In this case, the projected probability acquires the standard form of a logistic classifier:

$$P_{ij} = \frac{1}{1 + \exp(-\sum_{k=1}^{n} s_{ij}^k \cdot \alpha_k)} \tag{3.11}$$

Hence, we can train the weights α_k using a margin classifier and interpret the output of the classifier probabilistically, as we do with a logistic classifier. In our experiments in Section 3.4 we indeed use a linear-kernel SVM to train the weights α_k and then we can interchangeably interpret the resulting Integer Linear Program as either score-based or probabilistic optimization.

3.2.2.4 Comparison to Snow et al.

Our work resembles Snow et al's work (165) in that both try to learn graph edges given a transitivity constraint. However, there are two key differences in the model and in the optimization algorithm. First, they employ a greedy optimization algorithm that incrementally adds hyponyms to a large taxonomy (WordNet), while we simultaneously learn all edges using a global optimization method, which is more sound and powerful theoretically, and leads to the optimal solution. Second, Snow et al.'s model attempts to determine the graph that maximizes the likelihood $P(F|G)$ and not the posterior $P(G|F)$. If we cast their objective function as an Integer Linear Program we get a formulation that is almost identical to ours, only containing the *inverse* prior odds ratio $\log \frac{1}{\eta} = -\log \eta$ rather than the prior odds ratio as the regularization term (see Section 2.2.2):

$$\hat{G}_{Snow} = \underset{G}{\operatorname{argmax}} \sum_{i \neq j} \log \frac{P_{ij}}{(1 - P_{ij})} \cdot x_{ij} - \log \eta \cdot |E| \tag{3.12}$$

This difference is insignificant when $\eta \sim 1$, or when η is tuned empirically for optimal performance on a development set. However, if η is statistically estimated, this might cause unwarranted results: Their model will favor dense graphs when the prior odds ratio is low ($\eta < 1$ or $P(x_{ij} = 1) < 0.5$), and sparse graphs when the prior odds is high ($\eta > 1$ or $P(x_{ij} = 1) > 0.5$), which is counterintuitive. Our model does not suffer from this shortcoming because it optimizes the posterior rather than the likelihood. In Section 3.4 we show that our algorithm significantly outperforms the algorithm presented by Snow et al.

3.3 Derivation of the Probabilistic Objective Function

In this section we provide a full derivation for the probabilistic objective function given in section 3.2.2.2. Given two nodes i and j from a set of nodes V, we denote by $x_{ij} = 1$ the event that i entails j, by F_{ij} the feature vector representing the ordered pair (i, j), and by F the set of feature vectors over all ordered pairs of nodes, that is $F = \cup_{i \neq j} F_{ij}$. We wish to learn a set of edges E, such that the posterior probability $P(G|F)$ is maximized, where $G = (V, E)$. We assume that we have a "local" model estimating the edge posterior probability $P_{ij} = P(x_{ij} = 1|F_{ij})$. Since this model was trained over a balanced training set, the prior for the event that i entails j under the model is uniform: $P(x_{ij} = 1) = P(x_{ij} = 0) = \frac{1}{2}$. Using Bayes rule we get:

$$P(x_{ij} = 1|F_{ij}) = \frac{P(x_{ij} = 1)}{P(F_{ij})} \cdot P(F_{ij}|x_{ij} = 1) = a \cdot P(F_{ij}|x_{ij} = 1) \tag{3.13}$$

$$P(x_{ij} = 0|F_{ij}) = \frac{P(x_{ij} = 0)}{P(F_{ij})} \cdot P(F_{ij}|x_{ij} = 0) = a \cdot P(F_{ij}|x_{ij} = 0) \tag{3.14}$$

where $a = \frac{1}{2 \cdot P(F_{ij})}$ is a constant with respect to any graph. Thus, we conclude that $P(x_{ij}|F_{ij}) = a \cdot P(F_{ij}|x_{ij})$. Next, we make three independence assumptions (the first two are following Snow et al. (165)):

$$P(F|G) = \prod_{i \neq j} P(F_{ij}|G) \tag{3.15}$$

$$P(F_{ij}|G) = P(F_{ij}|x_{ij}) \tag{3.16}$$

$$P(G) = \prod_{i \neq j} P(x_{ij}) \tag{3.17}$$

Assumption 3.15 states that each feature vector is independent from other feature vectors given the graph. Assumption 3.16 states that the features F_{ij} for the pair (i, j) are generated by a distribution depending only on whether entailment holds for (i, j). Last, assumption 3.17 states that edges are independent and the prior probability of a graph is a product of the prior probabilities of the edges. Using these assumptions and equations 3.13 and 3.14, we can now express the posterior $P(G|F)$:

$$P(G|F) \propto P(G) \cdot P(F|G) \tag{3.18}$$
$$= \prod_{i \neq j} [P(x_{ij}) \cdot P(F_{ij}|x_{ij})] \tag{3.19}$$
$$= \prod_{i \neq j} P(x_{ij}) \cdot \frac{P(x_{ij}|F_{ij})}{a} \tag{3.20}$$
$$\propto \prod_{i \neq j} P(x_{ij}) \cdot P_{ij} \tag{3.21}$$
$$= \prod_{(i,j) \in E} P(x_{ij} = 1) \cdot P_{ij} \cdot \prod_{(i,j) \notin E} P(x_{ij} = 0) \cdot (1 - P_{ij}) \tag{3.22}$$

Note that under the "local model" the prior for an edge in the graph was uniform, since the model was trained over a balanced training set. However, generally, this is not the case, and thus we introduce an edge prior into the model when formulating the global objective function. Now, we can formulate $P(G|F)$ as a linear function:

$$\hat{G} = \underset{G}{\operatorname{argmax}} \prod_{(i,j) \in E} P(x_{ij} = 1) \cdot P_{ij} \cdot \prod_{(i,j) \notin E} P(x_{ij} = 0) \cdot (1 - P_{ij}) \tag{3.23}$$
$$= \underset{G}{\operatorname{argmax}} \sum_{(i,j) \in E} \log(P_{ij} \cdot P(x_{ij} = 1)) + \sum_{(i,j) \notin E} \log[(1 - P_{ij}) \cdot P(x_{ij} = 0)] \tag{3.24}$$
$$= \underset{G}{\operatorname{argmax}} \sum_{i \neq j} (x_{ij} \cdot \log(P_{ij} \cdot P(x_{ij} = 1)) + (1 - x_{ij}) \cdot \log[(1 - P_{ij}) \cdot P(x_{ij} = 0)]) \tag{3.25}$$
$$= \underset{G}{\operatorname{argmax}} \sum_{i \neq j} \left(\log \frac{P_{ij} \cdot P(x_{ij} = 1)}{(1 - P_{ij}) \cdot P(x_{ij} = 0)} \cdot x_{ij} + (1 - P_{ij}) \cdot P(x_{ij} = 0) \right) \tag{3.26}$$
$$= \underset{G}{\operatorname{argmax}} \sum_{i \neq j} \log \frac{P_{ij}}{(1 - P_{ij})} \cdot x_{ij} + \log \eta \cdot |E| \tag{3.27}$$

In the last transition we omit $\sum_{i \neq j} (1 - P_{ij}) \cdot P(x_{ij} = 0)$, which is a constant with respect to the graph and denote the prior odds ratio by $\eta = \frac{P(x_{ij}=1)}{P(x_{ij}=0)}$. This leads to the final formulation described in section 3.2.2.2.

3.4 Experimental Evaluation

This section presents an evaluation and analysis of our algorithm.

3.4.1 Experimental setting

A health-care corpus of 632MB was harvested from the web and parsed using the Minipar parser (103). The corpus contains 2,307,585 sentences and almost 50 million word tokens. We used the Unified Medical Language System (UMLS)[1] to annotate medical concepts in the corpus. The UMLS is a database that maps natural language phrases to over one million *concept identifiers* in the health-care domain (termed *CUIs*). We annotated all nouns and noun phrases that are in the UMLS with their (possibly multiple) CUIs. We now provide the details of training an entailment classifier as explained in Section 3.2.1.

We extracted all templates from the corpus where both argument instantiations are medical concepts, that is, annotated with a CUI (∼50,000 templates). This was done to increase the likelihood that the extracted templates are related to the health-care domain and reduce problems of ambiguity.

As explained in Section 3.2.1, a pair of templates constitutes an input example for the entailment classifier, and should be represented by a set of features. The features we used were different distributional similarity scores for the pair of templates, as summarized in Table 3.2. Twelve distributional similarity measures were computed over the health-care corpus using the aforementioned variations (Section 3.2.1), where two feature representations were considered: in the UMLS each natural language phrase may be mapped not to a *single* CUI, but to a *tuple* of CUIs. Therefore, in the first representation, each feature vector coordinate counts the number of times a *tuple* of CUIs was mapped to the term instantiating the template argument, and in the second representation it counts the number of times each single CUI was *one* of the CUIs mapped to the term instantiating the template argument. In addition, we obtained the original template similarity lists learned by Lin and Pantel (104), and had available three distributional similarity measures learned by Szpektor and Dagan (172), over the RCV1 corpus[2], as detailed in Table 3.2. Thus, each pair of templates is represented by a total of 16 distributional similarity scores.

We automatically generated a balanced training set of 20,144 examples using WordNet and the procedure described in Section 3.2.1, and trained the entailment classifier with SVMperf (89). We use the trained classifier to obtain estimates for P_{ij} and s_{ij},

[1] http://www.nlm.nih.gov/research/umls

[2] http://trec.nist.gov/data/reuters/reuters.html

#	Corpus	Template	Similarity measure	Feature representation
1	health-care	binary	BInc	pair of CUI tuples
2	health-care	binary	BInc	pair of CUIs
3	health-care	binary	BInc	CUI tuple
4	health-care	binary	BInc	CUI
5	health-care	binary	Lin	pair of CUI tuples
6	health-care	binary	Lin	pair of CUIs
7	health-care	binary	Lin	CUI tuple
8	health-care	binary	Lin	CUI
9	health-care	unary	BInc	CUI tuple
10	health-care	unary	BInc	CUI
11	health-care	unary	Lin	CUI tuple
12	health-care	unary	Lin	CUI
13	RCV1	binary	Lin	lexical items
14	RCV1	unary	Lin	lexical items
15	RCV1	unary	BInc	lexical items
16	Lin & Pantel	binary	Lin	lexical items

Table 3.2: The similarity score features used to represent pairs of templates. The columns specify the corpus over which the similarity score was computed, the template representation, the similarity measure employed and the feature representation (as described in Section 3.2.1).

given that the score-based and probabilistic scoring functions are equivalent (cf. Section 3.2.2.3).

To evaluate the performance of our algorithm, we manually constructed gold standard entailment graphs. First, 23 medical target concepts, representing typical topics of interest in the medical domain, were manually selected from a (longer) list of the most frequent concepts in the health-care corpus. The 23 target concepts are: *alcohol*, *asthma*, *biopsy*, *brain*, *cancer*, *CDC*, *chemotherapy*, *chest*, *cough*, *diarrhea*, *FDA*, *headache*, *HIV*, *HPV*, *lungs*, *mouth*, *muscle*, *nausea*, *OSHA*, *salmonella*, *seizure*, *smoking* and *x-ray*. For each concept, we wish to learn a focused entailment graph (cf. Figure 3.1). Thus, the nodes of each graph were defined by extracting all propositional templates in which the corresponding target concept instantiated an argument at least $K(= 3)$ times in the health-care corpus (average number of graph nodes=22.04, std=3.66, max=26, min=13).

Ten medical students were given the nodes of each graph (propositional templates) and constructed the gold standard of graph edges using a web interface. We gave an oral explanation on the annotation process to each student, and the first two graphs annotated by every student were considered part of the annotator training phase and were discarded. The annotators were able to click on every propositional template and observe all of the instantiations of that template in our health-care corpus. For example, clicking on the template *'X help with nausea'* might show the propositions *'relaxation help with nausea'*, *'acupuncture help with nausea'* and *'Nabilone help with nausea'*. The concept of *entailment* was explained under the framework of Textual Entailment (48), that is, the template t_1 entails the template t_2 if given that the instantiation of t_1 with some concept is true then the instantiation of t_2 with the same concept is most likely true.

As explained in Section 3.1.2, we did not perform any disambiguation since a target concept disambiguates the propositional templates in focused entailment graphs. In practice, cases of ambiguity were very rare, except for a single scenario where in templates such as *'X treat asthma'*, annotators were unclear whether *'X'* is a type of doctor or a type of drug. The annotators were instructed in such cases to click on the template, read the instantiations of the template in the corpus and choose the sense that is most prevalent in the corpus. This instruction was applicable to all cases of ambiguity.

Each concept graph was annotated by two students. Following the current RTE practice (15), after initial annotation the two students met for a reconciliation phase. They worked to reach an agreement on differences and corrected their graphs. Inter-annotator agreement was calculated using the Kappa statistic (161) both before ($\kappa = 0.59$) and after ($\kappa = 0.9$) reconciliation. Each learned graph was evaluated against the two reconciliated gold standard graphs.

Summing the number of possible edges over all 23 concept graphs we get 10,364 possible edges, of which 882 on average were included by the annotators (averaging over the two gold standard annotations for each graph). The concept graphs were randomly split into a development set (11 concepts) and a test set (12 concepts).

We used the *lpsolve*[1] package to learn the edges of the graphs. This package efficiently solves the model without imposing integer restrictions and then uses the branch-and-bound method to find an optimal integer solution. We note that in the experiments reported in this article the optimal solution without integer restrictions was already integer. Thus, although in general our optimization problem is NP-hard, in our experiments we were able to reach an optimal solution for the input graphs very efficiently (we note that in some scenarios not reported in this chapter the optimal solution was not integer and so an integer solution is not guaranteed a-priori).

As mentioned in Section 3.2.2, we added a few constraints in cases where there is strong evidence that edges are not in the graph. This is done in the following scenarios (examples given in Table 3.3): (1) When two templates i and j are identical except for a pair of words w_i and w_j, and w_i is an antonym of w_j, or a hypernym of w_j at distance ≥ 2 in WordNet. (2) When two nodes i and j are transitive "opposites", that is, if $i = X \xleftarrow{subj} w \xrightarrow{obj} Y$ and $j = X \xleftarrow{obj} w \xrightarrow{subj} Y$, for any word w. We note that there are some transitive verbs that express a reciprocal activity, such as "X marry Y", but usually reciprocal events are not expressed using a transitive verb structure.

In addition, in some cases we have strong evidence that edges do exist in the graph. This is done in a single scenario (see Table 3.3), which is specific to the output of Minipar: when two templates differ by a single edge and the first is of the type $X \xrightarrow{obj} Y$ while the other is of the type $X \xleftarrow{vrel} Y$, which expresses a passive verb modifier of nouns. Altogether, these initializations took place in less than 1% of the node pairs in the graphs. We note that we tried to use WordNet relations such as hypernym and

[1] http://lpsolve.sourceforge.net/5.5/

Scenario	Example	Init.
antonym	$(X \xleftarrow{subj} decrease \xrightarrow{obj} Y, X \xleftarrow{subj} increase \xrightarrow{obj} Y)$	$x_{ij} = 0$
hypernym ≥ 2	$(X \xleftarrow{subj} affect \xrightarrow{obj} Y, X \xleftarrow{subj} irritate \xrightarrow{obj} Y)$	$x_{ij} = 0$
transitive opposite	$(X \xleftarrow{subj} cause \xrightarrow{obj} Y, Y \xleftarrow{subj} cause \xrightarrow{obj} X)$	$x_{ij} = 0$
syntactic variation	$(X \xleftarrow{subj} follow \xrightarrow{obj} Y, Y \xleftarrow{subj} follow \xleftarrow{vrel} X)$	$x_{ij} = 1$

Table 3.3: Scenarios in which we add hard constraints to the ILP.

synonym as "positive" hard constraints (using the constraint $x_{ij} = 1$), but this turned out to reduce performance because the precision of WordNet was not high enough.

The graphs learned by our algorithm were evaluated by two measures. The first measure evaluates the graph edges directly, and the second measure is motivated by semantic inference applications that utilize the rules in the graph. The first measure is simply the F_1 of the set of learned edges compared to the set of gold standard edges. In the second measure we take the set of learned rules and infer new propositions by applying the rules over all propositions extracted from the health-care corpus. We apply the rules iteratively over all propositions until no new propositions are inferred. For example, given the corpus proposition *'relaxation reduces nausea'* and the edges *'X reduce nausea ⇒ X help with nausea'* and *'X help with nausea ⇒ X related to nausea'*, we evaluate the set {*'relaxation reduces nausea', 'relaxation helps with nausea', 'relaxation related to nausea'*}. For each graph we measure the F_1 of the set of propositions inferred by the learned graphs when compared to the set of propositions inferred by the gold standard graphs. For both measures the final score of an algorithm is a macro-average F_1 over the 24 gold standard test set graphs (two gold standard graphs for each of the 12 test concepts).

Learning the edges of a graph given an input concept takes about 1-2 seconds on a standard desktop.

3.4.2 Evaluated algorithms

First, we describe some baselines that do not utilize the entailment classifier or the ILP solver. For each of the 16 distributional similarity measures (Table 3.2) and for

each template t, we computed a list of templates most similar to t (or entailing t for directional measures). Then, for each measure we learned graphs by inserting an edge (i, j), when i is in the top K templates most similar to j. The parameter K can be optimized either on the automatically generated training set (from WordNet) or on the manually-annotated development set. We also learned graphs using WordNet: we inserted an edge (i, j) when i and j differ by a single word w_i and w_j, respectively, and w_i is a direct hyponym or synonym of w_j. Next, we describe algorithms that utilize the entailment classifier.

Our algorithm, termed *ILP-Global*, utilizes global information and an ILP formulation to find maximum a-posteriori graphs. Therefore, we compare it to the following three variants: (1) *ILP-Local*: An algorithm that uses only local information. This is done by omitting the global transitivity constraints, and results in an algorithm that inserts an edge (i, j) if and only if $(s_{ij} - \lambda) > 0$ (2) *Greedy-Global*: An algorithm that looks for the maximum a-posteriori graphs but only employs the greedy optimization procedure as described by Snow et al. (3) *ILP-Global-Likelihood*: An ILP formulation where we look for the maximum likelihood graphs, as described by Snow et al .(cf. Section 3.2.2).

We evaluate these algorithms in three settings which differ in the method by which the edge prior odds ratio, η (or λ), is estimated: (1) $\eta = 1$ ($\lambda = 0$), which means that no prior is used. (2) Tuning η and using the value that maximizes performance over the development set. (3) Estimating η using maximum likelihood over the development set, which results in $\eta \sim 0.1$ ($\lambda \sim 2.3$), corresponding to the edge density $P(x_{ij} = 1) \sim 0.09$.

For all local algorithms whose output does not respect transitivity constraints, we added all edges inferred by transitivity. This was done since we assume that the rules learned are to be used in the context of an inference or entailment system. Since such systems usually perform chaining of entailment rules (10, 79, 141), we conduct this chaining as well. Nevertheless, we also measured performance when edges inferred by transitivity are not added: we once again chose the edge prior value that maximizes F_1 over the development set and obtained macro-average recall/precision/F_1 of 51.5/34.9/38.3. This is comparable to the macro-average recall/precision/F_1 of 44.5/45.3/38.1 we report next in Table 3.4.

	Edges			Propositions		
	Recall	Precision	F_1	Recall	Precision	F_1
ILP-global ($\lambda = 0.45$)	46.0	50.1	**43.8**	67.3	69.6	**66.2**
Greedy-global ($\lambda = 0.3$)	45.7	37.1	36.6	64.2	57.2	56.3
ILP-Local ($\lambda = 1.5$)	44.5	45.3	38.1	65.2	61.0	58.6
Local_1 ($K = 10$)	53.5	34.9	37.5	73.5	50.6	56.1
Local_2 ($K = 55$)	52.5	31.6	37.7	69.8	50.0	57.1

Table 3.4: Results when tuning for performance over the development set.

	Edges			Propositions		
	Recall	Precision	F_1	Recall	Precision	F_1
ILP-global	58.0	28.5	**35.9**	76.0	46.0	**54.6**
Greedy-global	60.8	25.6	33.5	77.8	41.3	50.9
ILP-Local	69.3	19.7	26.8	82.7	33.3	42.6
Local_1 ($K = 100$)	92.6	11.3	20.0	95.3	18.9	31.1
Local_2 ($K = 100$)	63.1	25.5	34.0	77.7	39.9	50.9
WordNet	10.8	44.1	13.2	39.9	72.4	47.3

Table 3.5: Results when the development set is not used to estimate λ and K.

3.4.3 Experimental results and analysis

In this section we present experimental results and analysis that show that the ILP-Global algorithm improves performance over baselines, specifically in terms of precision.

Tables 3.4-3.7 and Figure 3.2 summarize the performance of the algorithms. Table 3.4 shows our main result when the parameters λ and K are optimized to maximize performance over the development set. Notice that the algorithm ILP-Global-likelihood is omitted, since when optimizing λ over the development set it conflates with ILP-Global. The rows *Local*$_1$ and *Local*$_2$ present the best algorithms that use a single distributional similarity resource. Local_1 and Local_2 correspond to the configurations described in Table 3.2 by features no. 5 and no. 1 respectively (see also Table 3.8). ILP-Global improves performance by at least 13%, and significantly outperforms all local methods, as well as the greedy optimization algorithm both on the edges F_1 measure

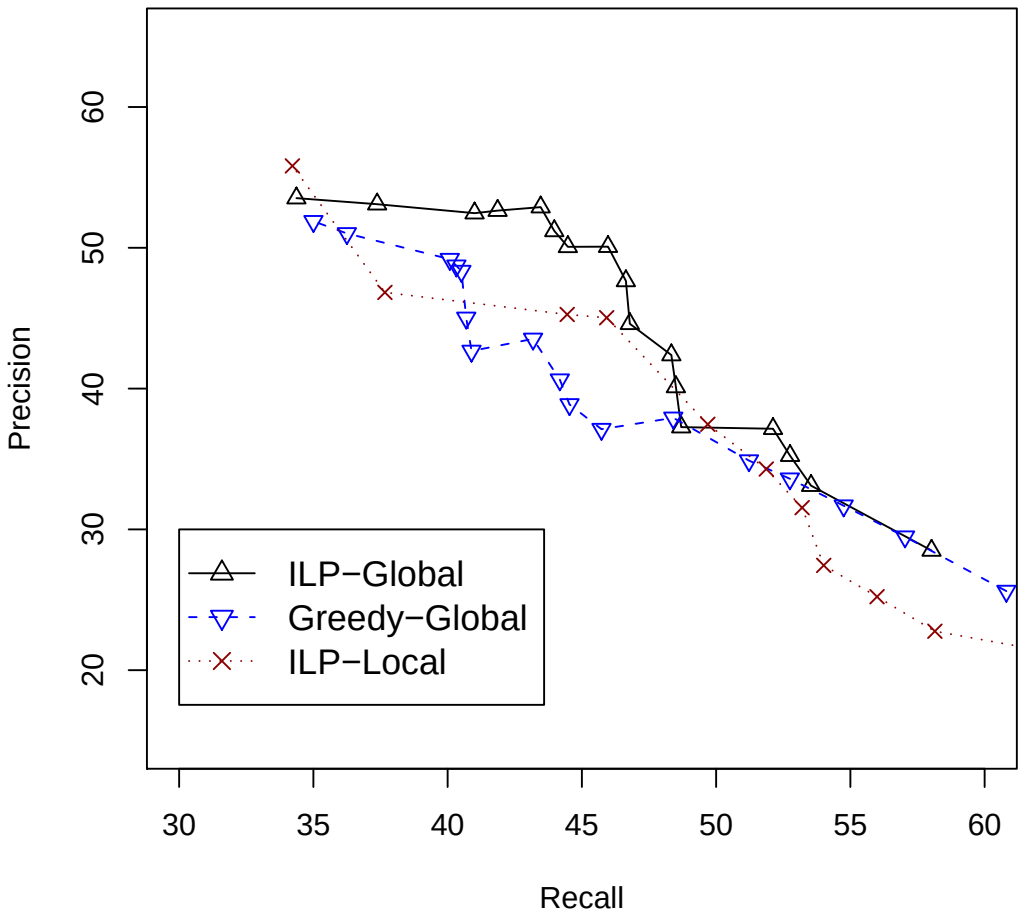

Figure 3.2: Recall-precision curve comparing ILP-Global with Greedy-Global and ILP-Local.

	Edges			Propositions		
	Recall	Precision	F_1	Recall	Precision	F_1
ILP-Global	16.8	67.1	**24.4**	43.9	86.8	**56.3**
ILP-Global-Likelihood	91.8	9.8	17.5	94.0	16.7	28.0
Greedy-Global	14.7	62.9	21.2	43.5	86.6	56.2
Greedy-Global-Likelihood	100.0	9.3	16.8	100.0	15.5	26.5

Table 3.6: Results with prior estimated on the development set, that is $\eta = 0.1$, which is equivalent to $\lambda = 2.3$.

($p<0.05$) and on the propositions F_1 measure ($p<0.01$)[1].

Table 3.5 describes the results when the development set is not used to estimate the parameters λ and K: a uniform prior ($P_{uv} = 0.5$) is assumed for algorithms that use the entailment classifier, and the automatically generated training set is employed to estimate K. Again ILP-Global-Likelihood is omitted in the absence of a prior. ILP-Global outperforms all other methods in this scenario as well, although by a smaller margin for a few of the baselines. Comparing table 3.4 to table 3.5 reveals that excluding the sparse prior indeed increases recall at a price of a sharp decrease in precision. However, note that local algorithms are more vulnerable to this phenomenon. This makes sense since in local algorithms eliminating the prior adds edges that in turn add more edges due to the constraint of transitivity and so recall dramatically raises at the expense of precision. Global algorithms are not as prone to this effect because they refrain from adding edges that eventually lead to the addition of many unwarranted edges.

Table 3.5 also shows that WordNet, a manually constructed resource, has notably the highest precision and lowest recall. The low recall exemplifies how the entailment relations given by the gold standard annotators transcend much beyond simple lexical relations that appear in WordNet: many of the gold standard entailment relations are missing from WordNet or involve multi-word phrases that do not appear in WordNet at all.

[1]We tested significance using the two-sided Wilcoxon rank test (186).

Note that although the precision of WordNet is the highest in Table 5, its absolute value (44.1%) is far from perfect. This illustrates that hierarchies of predicates are quite ambiguous and thus using WordNet directly yields relatively low precision. WordNet is vulnerable for such ambiguity since it is a generic domain-independent resource, while our algorithm learns from a domain-specific corpus. For example, The words *'have'* and *'cause'* are synonyms according to one of the senses in WordNet and so the erroneous rule *'X have asthma* $\Leftrightarrow$ *X cause asthma'* is learned using WordNet. Another example is the rule *'X follow chemotherapy* $\Rightarrow$ *X take chemotherapy'*, which is incorrectly inferred since *'follow'* is a hyponym of *'take'* according to one of WordNet's senses (*'she followed the feminist movement'*). Due to these mistakes made by WordNet, the precision achieved by our automatically trained ILP-Global algorithm when tuning parameters on the development set (Table 3.4) is higher than that of WordNet.

Table 3.6 shows the results when the prior η is estimated using maximum likelihood over the development set (by computing the edge density over all the development set graphs), and not tuned empirically with grid search. This allows for a comparison between our algorithm that maximizes the a-posteriori probability and Snow et al.'s algorithm that maximizes the likelihood. The gold standard graphs are quite sparse ($\eta \sim 0.1$), therefore, as explained in Section 3.2.2.4, the effect of the prior is substantial. ILP-Global and Greedy-Global learn sparse graphs with high precision and low recall, while ILP-Global-Likelihood and Greedy-Global-Likelihood learn dense graphs with high recall but very low precision. Overall, optimizing the a-posteriori probability is substantially better than optimizing likelihood, but still leads to a large degradation in performance. This can be explained since our algorithm is not purely probabilistic: the learned graphs are the product of mixing a probabilistic objective function with non-probabilistic constraints. Thus, plugging the estimated prior into this model results in performance that is far from optimal.

Figure 3.2 shows a recall-precision curve for ILP-Global, Greedy-Global and ILP-Local, obtained by varying the prior parameter, λ. The figure clearly demonstrates the advantage of using global information and ILP. ILP-Global is better than Greedy-Global and ILP-Local in almost every point of the recall–precision curve, regardless of the exact value of the prior parameter. Last, we present for completeness in Table 3.7 the results of ILP-Global for all concepts of the test set.

Concept	R	P	F_1
Smoking	58.1	81.8	67.9
Seizure	64.7	51.2	57.1
Headache	60.9	50.0	54.9
Lungs	50.0	56.5	53.1
Diarrhea	42.1	60.0	49.5
Chemotherapy	44.7	52.5	48.3
HPV	35.2	76.0	48.1
Salmonella	27.3	80.0	40.7
X-ray	75.0	23.1	35.3
Asthma	23.1	30.6	26.3
Mouth	17.7	35.5	23.7
FDA	53.3	15.1	23.5

Table 3.7: Results per concept for ILP-Global.

In Table 3.8 we present the results obtained for all 16 distributional similarity measures. The main conclusion we can derive from this table is that the best distributional similarity measures are those that represent templates using *pairs* of argument instantiations rather than each argument separately. A similar result was found by Yates and Etzioni (191), who described the RESOLVER paraphrase learning system and have shown that it outperforms DIRT. In their analysis, they attribute this result to their representation that utilizes pairs of arguments comparing to DIRT, which computes a separate score for each argument.

In the next two sections we perform a more thorough qualitative and quantitative comparison trying to analyze the importance of using global information in graph learning (Section 3.4.3.1), as well as the contribution of using ILP rather than a greedy optimization procedure (Section 3.4.3.2). We note that the analysis presented in both sections is for the results obtained when optimizing parameters over the development set.

	Edges			Propositions		
Dist. sim. measure	Recall	Precision	F_1	Recall	Precision	F_1
h-b-B-pCt	52.5	31.6	**37.7**	69.8	50.0	**57.1**
h-b-B-pC	50.5	26.5	30.7	67.1	43.5	50.1
h-b-B-Ct	10.4	44.5	15.4	39.1	78.9	51.6
h-b-B-C	7.6	42.9	11.1	37.9	79.8	50.7
h-b-L-pCt	53.4	34.9	37.5	73.5	50.6	56.1
h-b-L-pC	47.2	35.2	35.6	68.6	52.9	56.2
h-b-L-Ct	47.0	26.6	30.2	64.9	47.4	49.6
h-b-L-C	34.6	22.9	22.5	57.2	52.6	47.6
h-u-B-Ct	5.1	37.4	8.5	35.1	91.0	49.7
h-u-B-C	7.2	42.4	11.5	36.1	90.3	50.1
h-u-L-Ct	22.8	22.0	18.3	49.7	49.2	44.5
h-u-L-C	16.7	26.3	17.8	47.0	56.8	48.1
R-b-L-l	49.4	21.8	25.2	72.4	39.0	45.5
R-u-L-l	24.1	30.0	16.8	47.1	55.2	42.1
R-u-B-l	9.5	57.1	14.1	37.2	84.0	49.5
Lin & Pantel	37.1	32.2	25.1	58.9	54.6	48.6

Table 3.8: Results of all distributional similarity measures when tuning K over the development set. We encode the description of the measures presented in Table 3.2 in the following manner — *h: health-care corpus*, *R: RCV1 corpus*, *b: binary templates*, *u: unary templates*, *L: Lin similarity measure*, *B: BInc similarity measure*, *pCt: pair of CUI tuples representation*, *pC: pair of CUIs representation*, *Ct: CUI tuple representation*, *C: CUI representation*, *Lin and Pantel: Similarity lists learned by Lin and Pantel.*

3.4.3.1 Global vs. local information

We looked at all edges in the test set graphs where ILP-Global and ILP-Local disagree and checked which algorithm was correct. Table 3.9 presents the result. The main advantage of using ILP-Global is that it avoids inserting wrong edges into the graph. This is since ILP-Local adds any edge (i, j) such that P_{ij} crosses a certain threshold, disregarding edges that will be consequently added due to transitivity (recall that for local algorithms we add edges inferred by transitivity, cf. Section 3.4.2). ILP-Global will avoid such edges of high probability if it results in inserting many low probability edges. This results in an improvement in precision, as exhibited by Table 3.4.

Figures 3.3 and 3.4 show fragments of the graphs learned by ILP-Global and ILP-Local (prior to adding transitive edges) for the test set concepts *diarrhea* and *seizure*, and illustrate qualitatively how global considerations improve precision. In Figure 3.3, we witness that the single erroneous edge *'X result in diarrhea ⇒ X prevent diarrhea'* inserted by the local algorithm because P_{ij} is high, effectively bridges two strongly connected components and induces a total of 12 wrong edges (All edges from the upper component to the lower component), while ILP-Global refrains from inserting this edge. Figure 3.4 depicts an even more complex scenario. First, ILP-Local induces a strongly connected component of five nodes for the predicates *'control'*, *'treat'*, *'stop'*, *'reduce'* and *'prevent'*, while ILP-Global splits this strongly connected component into two, which while not perfect, is more compatible with the gold standard graphs. In addition, ILP-Local inserts four erroneous edges that connect two components of size 4 and 5, which results in adding eventually 30 wrong edges. On the other hand, ILP-Global is aware of the consequences of adding these four seemingly good edges, and prefers to omit them from the learned graph, leading to much higher precision.

Although the main contribution of ILP-Global, in terms of F_1, is in an increase in precision, we also notice an increase in recall in Table 3.4. This is since the optimal prior is $\lambda = 0.45$ in ILP-Global but $\lambda = 1.5$ in ILP-Local. Thus, any edge (i, j) such that $0.45 < s_{ij} < 1.5$ will have positive weight in ILP-Global and might be inserted into the graph, but will have negative weight in ILP-Local and will be rejected. The reason is that in a local setting, reducing false positives is handled only by applying a large penalty for every wrong edge, while in a global setting wrong edges can be rejected since they induce more "bad" edges. Overall, this leads to an improved recall

	Global=True/Local=False	Global=False/Local=True
Gold standard=true	48	42
Gold standard=false	78	494

Table 3.9: Comparing disagreements between ILP-Global and ILP-Local against the gold standard graphs.

	ILP=True/Greedy=False	ILP=False/Greedy=True
Gold standard=true	66	56
Gold standard=false	44	480

Table 3.10: Comparing disagreements between ILP-Global and Greedy-Global against the gold standard graphs.

in ILP-Global. This also explains why ILP-Local is severely harmed when no prior is used at all, as shown in Table 3.5.

Last, we note that across the 12 test set graphs, ILP-Global achieves better F_1 over the edges in 7 graphs with an average advantage of 11.7 points, ILP-Local achieves better F_1 over the edges in 4 graphs with an average advantage of 3.0 points, and once performance is equal.

3.4.3.2 Greedy vs. non-greedy optimization

We would like to understand how using an ILP solver improves performance comparing to a greedy optimization procedure. Table 3.4 demonstrates that ILP-Global and Greedy-Global reach a similar level of recall, however ILP-Global achieves far better precision. Again, we investigated edges for which the two algorithms disagree and checked which one was correct. Table 3.10 demonstrates that the higher precision is because ILP-Global avoids inserting wrong edges into the graph.

Figure 3.5 illustrates some of the reasons ILP-Global performs better than Greedy-Global. Parts A1-A3 show the progression of Greedy-Global, which is an incremental algorithm, for a fragment of the *'headache'* graph. In part A1 the learning algorithm still separates the nodes *'X prevent headache'* and *'X reduce headache'* from the nodes *'X cause headache'* and *'X result in headache'* (nodes surrounded by a bold oval shape

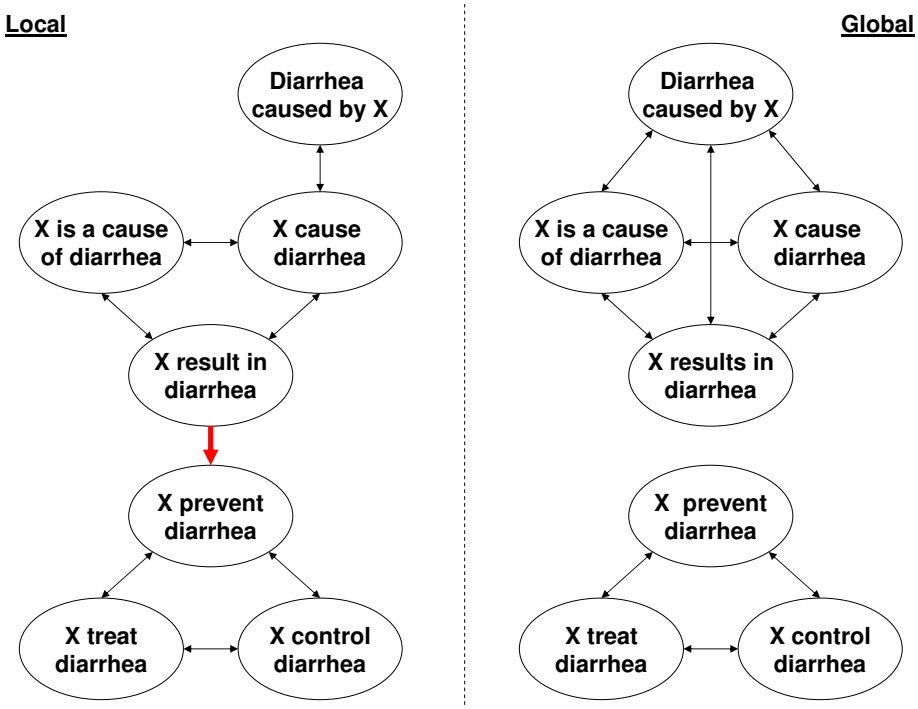

Figure 3.3: A comparison between ILP-Global and ILP-local for two fragments of the test set concept *diarrhea*.

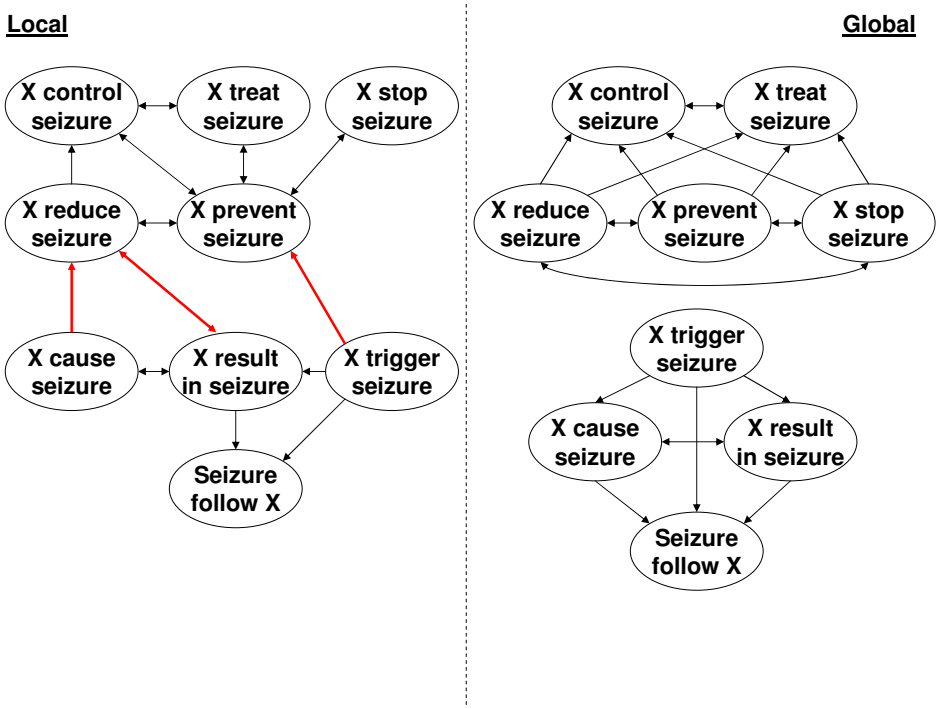

Figure 3.4: A comparison between ILP-Global and ILP-local for two fragments of the test set concept *seizure*.

constitute a strongly connected component). However, after two iterations the four nodes are joined into a single strongly connected component, which is an error in principle, but at this point seems to be the best decision to increase the posterior probability of the graph. This greedy decision has two negative ramifications. First, the strongly connected component can not be untied anymore. Thus, in A3 we observe that in future iterations the strongly connected component expands further and many more wrong edges are inserted into the graph. On the other hand, in B we see that ILP-Global takes into consideration the global interaction between the four nodes and other nodes of the graph, and decides to split this strongly connected component in two, which improves the precision of ILP-Global. Second, note that in A3 the nodes *'Associate X with headache'* and *'Associate headache with X'* are erroneously isolated. This is since connecting them to the strongly connected component that contains six nodes will add many edges with low probability and so this is avoided by Greedy-Global. Because in ILP-Global the strongly connected component was split in two, it is possible to connect these two nodes to some of the other nodes and raise the recall of ILP-Global. Thus, we see that greedy optimization might get stuck in local maxima and consequently suffer in terms of both precision and recall.

Last, we note that across the 12 test set graphs, ILP-Global achieves better F_1 over the edges in 9 graphs with an average advantage of 10.0 points, Greedy-Global achieves better F_1 over the edges in 2 graphs with an average advantage of 1.5 points, and once performance is equal.

3.4.4 Error Analysis

In this section, we compare the results of ILP-Global with the gold standard graphs and perform error analysis. Error analysis was performed by comparing the 12 graphs learned by ILP-Global to the corresponding 12 gold standard graphs (randomly sampling from the two available gold standard graphs), and manually examining all edges for which the two disagree. We found that the number of false positives and false negatives is almost equal: 282 edges were learned by ILP-Global but are not in the gold standard graphs (false positive) and 287 edges were in the gold standard graphs but were not learned by ILP-Global (false negatives).

Table 3.11 presents the results of our manual error analysis. Most evident is the fact that the majority of mistakes are misclassifications of the entailment classifier. For

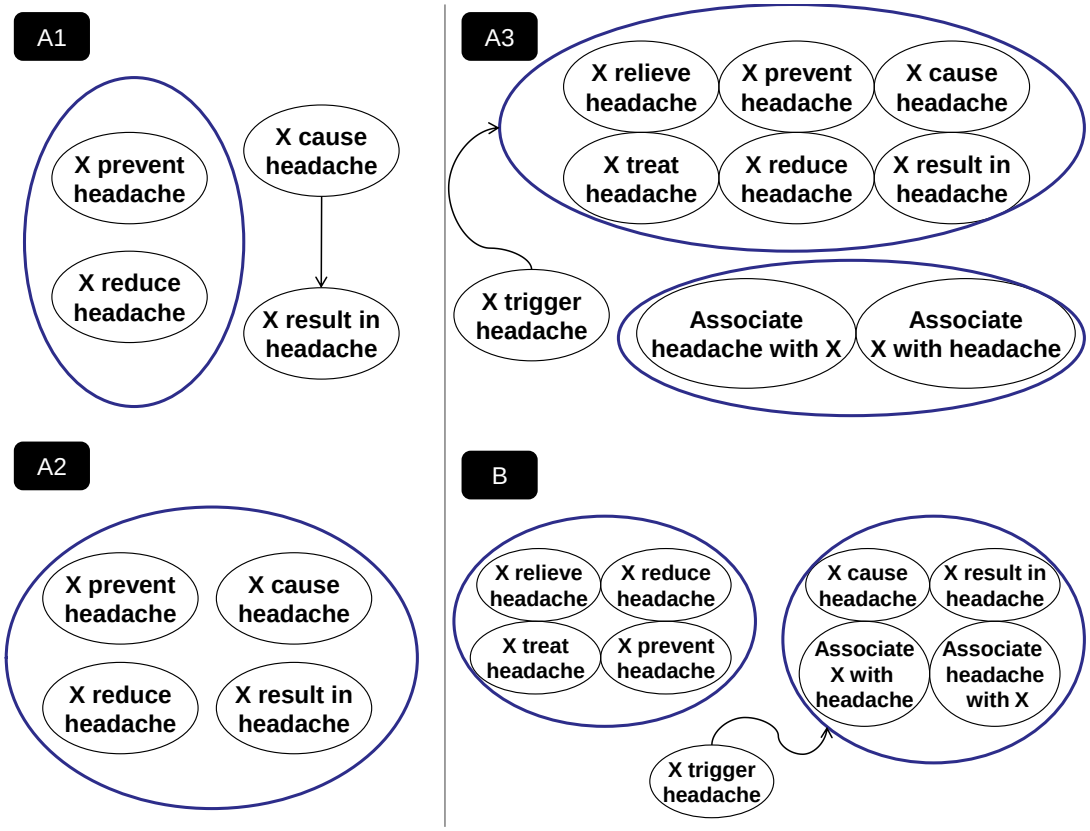

Figure 3.5: A comparison between ILP-Global and Greedy-Global. Parts A1-A3 depict the incremental progress of Greedy Global for a fragment of the *headache* graph. Part B depicts the corresponding fragment in ILP-Global. Nodes surrounded by a bold oval shape are strongly connected components.

False Positives		**False Negatives**	
Total count	282	Total count	287
Classifier error	84.8%	Classifier error	73.5%
Co-hyponym error	18.0%	Long-predicate error	36.2%
Direction error	15.1%	Generality error	26.8%
		String overlap error	20.9%

Table 3.11: Error analysis for false positives and false negatives.

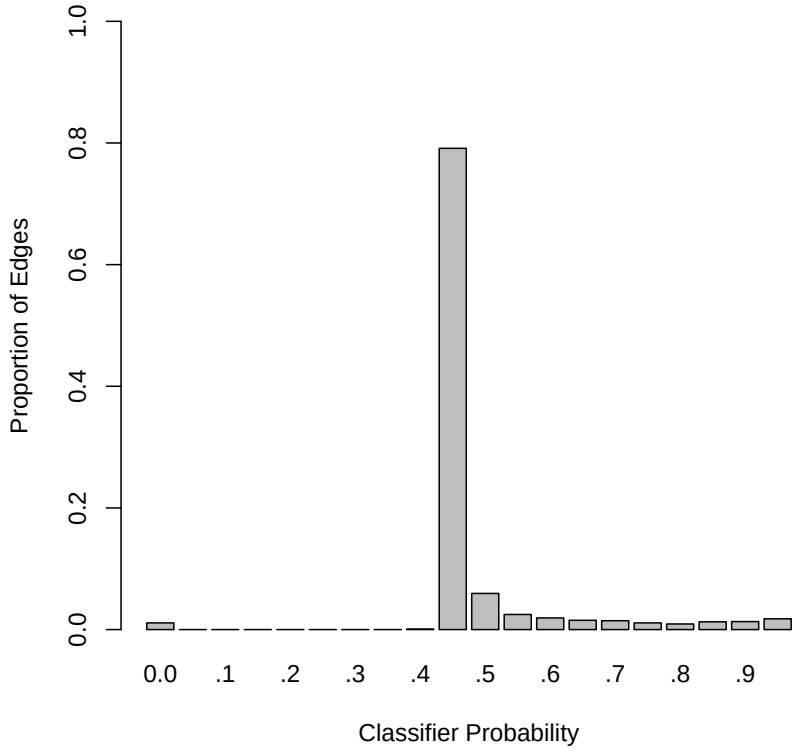

Figure 3.6: Distribution of probabilities given by the classifier over all node pairs of the test set graphs.

73.5% of the false negatives the classifier's probability was $P_{ij} < 0.5$ and for 84.8% of the false positives the classifier's probability was $P_{ij} > 0.5$. This shows that our current classifier struggles to distinguish between positive and negative examples. Figure 3.6 illustrates some of this difficulty by showing the distribution of the classifier's probability, P_{ij}, over all node pairs in the 12 test set graphs. Close to 80% of the scores are in the range 0.45-0.5, most of which are simply node pairs for which all distributional similarity features are zero. Although in the grand majority of such node pairs (t_1, t_2) indeed t_1 does not entail t_2, there are also some cases where t_1 does entail t_2. This implies that the current feature representation is not rich enough, and in the next section and in Chapter 5 we explore a larger feature set.

Table 3.11 also shows some other reasons found for false positives. Many false positives are pairs of predicates that are semantically related, that is, 18% of false positives are templates that are hyponyms of a common predicate (co-hyponym error), and 15.1% of false positives are pairs where we err in the direction of entailment (direction error). For example ILP-Global learns that *'place X in mouth ⇒ remove X from mouth'*, which is a co-hyponym error, and also that *'X affect lungs ⇒ X damage lungs'*, which is a direction error since entailment holds in the other direction. This illustrates the infamous difficulty of distributional similarity features to discern the type of semantic relation between two predicates.

Table 3.11 also shows additional reasons for false negatives. We found that in 36.2% of false negatives one of the two templates contained a "long" predicate, that is a predicate composed of more than one content word, such as *'Ingestion of X causes injury to Y'*. This might indicate that the size of the health-care corpus is too small to collect sufficient statistics for complex predicates. In addition, 26.8% of false negatives were manually analyzed as "generality errors". An example is the edge *'HPV strain cause X ⇒ associate HPV with X'* that is in the gold standard graph but missed by ILP-Global. Indeed this edge falls within the definition of textual entailment and is correct: for example, if some HPV strain causes cervical cancer then cervical cancer is associated with HPV. However, because the entailed template is much more general than the entailing template, they are not instantiated by similar arguments in the corpus and distributional similarity features fail to capture their semantic similarity. Last, we note that in 20.9% of the false negatives, there was some string overlap between the entailing and entailed templates, for example in *'X control asthma symptoms ⇒ X control asthma'*. In the next section and in Chapter 5 we experiment with a feature that is based on string similarity.

Tables 3.9 and 3.10 show that there are cases where ILP-Global makes a mistake, while ILP-Local or Greedy-Global are correct. An illustrating example for such a case is shown in Figure 3.7. Looking at ILP-Local we see that the entailment classifier classifies correctly the edges *'X trigger asthma ⇒ X cause asthma'* and *'X cause asthma ⇒ Associate X with asthma'*, but misclassifies *'X trigger asthma ⇒ Associate X with asthma'*. Since this configuration violates a transitivity constraint, ILP-Global must make a global decision whether to add the edge *'X trigger asthma ⇒ Associate X with asthma'* or to omit one of the correct edges. The optimal global decision in this

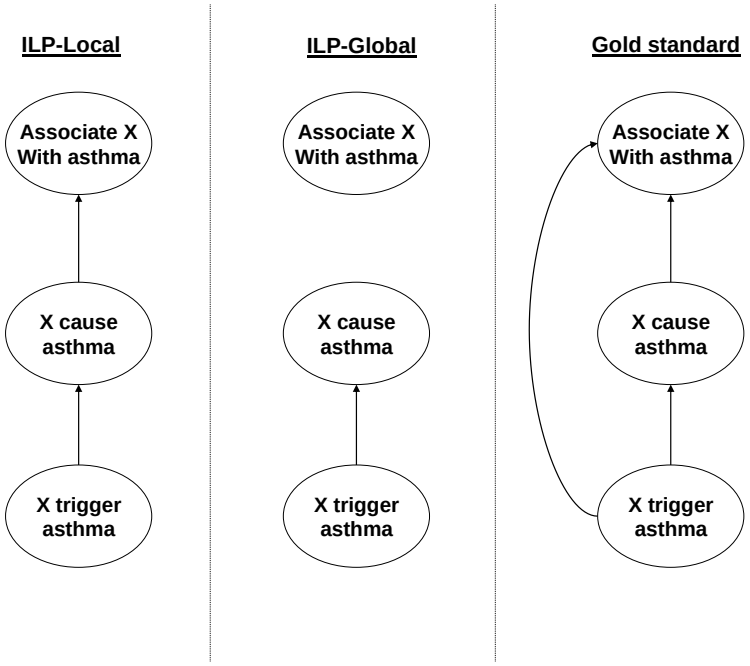

Figure 3.7: A scenario where ILP-Global makes a mistake, but ILP-Local is correct.

case causes a mistake with respect to the gold standard. More generally, a common phenomenon of ILP-Global is that it splits components that are connected in ILP-Local, for example in Figures 3.3 and 3.4. ILP-Global splits the components in a way that is optimal according to the scores of the local entailment classifier, but these are not always accurate according to the gold standard.

Figure 3.5 exemplifies a scenario where ILP-Global errs, but Greedy-Global is (partly) correct. ILP-Global mistakenly learns entailment rules from the templates *'Associate X with headache'* and *'Associate headache with X'* to the templates *'X cause headache'* and *'X result in headache'*, while Greedy-Global isolates the templates *'Associate X with headache'* and *'Associate headache with X'* in a separate component. This happens because of the greedy nature of Greedy-Global. Notice that in step A2 the templates *'X cause headache'* and *'X result in headache'* are already included (erroneously) in a connected component with the templates *'X prevent headache'* and *'X reduce headache'*. Thus, adding the rules from *'Associate X with headache'* and *'Associate headache with X'* to *'X cause headache'* and *'X result in headache'* would also

add the rules to *'X reduce headache'* and *'X prevent headache'* and the Greedy-Global avoids that. ILP-Global does not have that problem: it simply chooses the optimal choice according to the entailment classifier, which splits the connected component presented in A2. Thus, once again we see that mistakes made by ILP-Global are often due to the inaccuracies of the scores given by the local entailment classifier.

3.5 Local Classifier Extensions

The error analysis in Section 3.4.4 exemplified that most errors are the result of misclassifications made by the local entailment classifier. In this Section, we investigate the local entailment classifier component, focusing on the set of features used for classification. We first present an experimental setting in which we consider a wider set of features, then we present the results of the experiment, and last we perform feature analysis and draw conclusions.

3.5.1 Feature set and experimental setting

In previous sections we employed a distant supervision framework: we generated training examples automatically with WordNet, and represented each example with distributional similarity features. However, distant supervision comes with a price — it prevents us from utilizing all sources of information. For example, looking at the pair of gold standard templates *'X manage asthma'* and *'X improve asthma management'*, one can exploit the fact that *'management'* is a derivation of *'manage'* to improve the estimation of entailment. However, the automatically generated training set was generated by looking at WordNet's hypernym, synonym, and co-hyponyms relations, and hence no such examples appear in the training set, rendering this type of feature useless. Moreover, one can not use WordNet's hypernym, synonym and co-hyponym relations as features since the generated training set is highly biased — all positive training examples are either hypernyms or synonyms and all negative examples are co-hyponyms.

In this section we would like to examine the utility of various features, while avoiding the biases that occur due to distant supervision. Therefore, we use the 23 manually annotated gold standard graphs for both training and testing, in a cross-validation setting. Although this reduces the size of the training set it allows us to estimate the

utility of various features in a setting where the training set and test set are sampled from the same underlying distribution, without the aforementioned biases.

We would like to extract features that express information that is diverse and orthogonal to the one given by distributional similarity. Therefore, we turn to existing knowledge resources that were created using both manual and automatic methods, expressing various types of linguistic and statistical information that is relevant for entailment prediction:

1. WordNet: contains manually annotated relations such as hypernymy, synonymy, antonymy, derivation and entailment.

2. VerbOcean [1] (39): contains verb relations such as *'stronger-than'* and *'similar'* that were learned with pattern-based methods.

3. CATVAR [2] (73): contains word derivations such as *'develop–development'*.

4. FRED [3] (14): contains entailment rules between templates learned automatically from FrameNet.

5. NOMLEX [4] (110): contains English nominalizations including their argument mapping to the corresponding verbal form.

6. BAP [5] (96): contains directional distributional similarity scores between lexical terms (rather than propositional templates) calculated with the BAP similarity scoring function.

Table 3.12 describes the 16 new features that were generated for each of the gold standard examples (resulting in a total of 32 features). The first 15 features were generated by the aforementioned knowledge-bases. The last feature measures the edit distance between templates: given a pair of templates (t_1, t_2), we concatenate the words in each template and derive a pair of strings (s_1, s_2). Then we compute the Levenshtein string edit-distance (43) between s_1 and s_2 and divide the score by $|s_1| + |s_2|$ for normalization.

[1] http://demo.patrickpantel.com/demos/verbocean/
[2] http://clipdemos.umiacs.umd.edu/catvar/
[3] http://u.cs.biu.ac.il/~nlp/downloads/FRED.html
[4] http://nlp.cs.nyu.edu/nomlex/index.html
[5] http://u.cs.biu.ac.il/~nlp/downloads/DIRECT.html

Table 3.12 also describes for each feature the number and percentage of examples for which the feature value is non-zero (out of the examples generated from the 23 gold standard graphs). A salient property of many of the new features is that they are sparse: the four antonymy features as well as the Derivation, Entailment, Nomlex and FRED features occur in very few examples in our data set, which might make training with these features difficult.

After generating the new features we employ a leave-one-graph-out strategy to maximally exploit the manually annotated gold standard for training. For each of the *test-set* graphs, we train over all development and test set graphs except for the one that is left out[1], after tuning the algorithm's parameters and test. Parameter tuning is done by cross-validation over the development set, tuning to maximize the F_1 of the set of learned edges (the development and test set are described in Section 3.4). Graphs are always learned with the ILP-Global algorithm.

Our main goal is to check whether the added features improve performance, and therefore we run the experiment both with and without the new features. In addition, we would like to test whether using different classifiers affects performance. Therefore, we run the experiments with a linear-kernel SVM, a square-kernel SVM, a Gaussian-kernel SVM, logistic regression and nave Bayes. We use the SVMPerf package (89) to train the SVM classifiers and the Weka package (75) for logistic regression and naive Bayes.

3.5.2 Experiment Results

Table 3.13 describes the macro-average recall, precision and F_1 of all classifiers both with and without the new features on the development set and test set. Using all features is denoted by X_{all}, while using the original features is denoted by X_{old}.

Examining the results it does not appear that the new features improve performance. While on the development set the new features add 1.2-1.5 F_1 points for all SVM classifiers, on the test set using the new features decreases performance for the linear and square classifiers. This shows that even if there is some slight increase in

[1] As described in Section 3.4, we train with a balanced number of positive and negative examples. Since the number of positive examples in the gold standard is smaller than the number of negative examples, we use all positives and randomly sample the same number of negatives, resulting in ~ 1500 training examples.

performance when using SVM on the development set, it is masked by the variance added in the process of parameter tuning. In general, including the new features does not yield substantial differences in performance.

Secondly, the SVM classifiers perform better than the logistic and nave Bayes classifiers. However, using the more complex square and Gaussian kernels does not seem justified as the differences between the various kernels are negligible. Therefore, in the following analysis we will use a linear kernel SVM classifier.

Last, we note that although we use supervised learning rather than distant supervision, the results we get are slightly lower than those presented in Section 3.4. This is probably due to the fact that our manually annotated data set is rather small. Nevertheless, this shows that the quality of the distant supervision training set generated automatically from WordNet is reasonable.

Next, we perform analysis of the different features of the classifier to better understand the reasons for the negative result obtained.

3.5.3 Feature analysis

We saw that the new features slightly improved performance for SVM classifiers on the development set, while no clear improvement was witnessed on the test set. To further check whether the new features carry useful information we measured the training set accuracy for each of the 12 training sets (leaving out each time one test set graph). Using the new features improved the average training set accuracy from 71.6 to 72.3. More importantly, it improved performance consistently in all 12 training sets by 0.4-1.2 points. This strengthens our belief that the new features do carry a certain amount of information, but this information is too sparse to affect the overall performance of the algorithm. In addition, notice that the absolute accuracy on the training set is low — 72.3. This shows that separating entailment from non-entailment using the current set of features is challenging.

Next, we would like to perform analysis on each of the features. First, we perform an ablation test over the features by omitting each one of them and re-training the classifier Linear$_{all}$. In table 3.14, the columns *ablation* F_1 and Δ show the F_1 obtained and the difference in performance from the Linear$_{all}$ classifier, which scored 40.3 F_1 points. Results show that there is no "bad" feature that deteriorates performance. For almost all features ablation causes a decrease in performance, however this decrease is

relatively small. There are only 4 features for which ablation decreases performance by more than one point: three distributional similarity features, but also the new hypernym feature.

The next three columns in the table describe the precision and recall of the new boolean features. The column *Fea. type* indicates whether we expect a feature to indicate entailment or non-entailment and the columns *Prec.* and *Recall* specify the precision and recall of that feature . For example, the feature *FRED* is a *positive* feature that we expect to support entailment, and indeed 77.8% of the gold standard examples for which it is turned on are positive examples. However, it is turned on only in 0.8% of the positive examples. Similarly, *VO ant.* is a *negative* feature that we expect to support non-entailment, and indeed 96% of the gold standard examples for which it is on are negative examples, but it is turned on in only 0.2% of the negative examples. The precision results are quite reasonable: for most positive features the precision is well over the proportion of positive examples in the gold standard, which is about 10% (except for the *Entailment* feature whose precision is only 15%). For the negative features it seems that the precision of VerbOcean features is very high (though they are sparse), while the precision of WordNet antonyms and co-hyponyms is lower. Looking at the recall we can see that the coverage of the boolean features is low.

The last column in the table describes results of training the classifier with a single feature. For each feature we train a linear kernel SVM, tune the sparsity parameter on the development set and measure F_1 over the test set. Naturally, classifiers that are trained on sparse features yield low performance.

This column allows us once again (cf. Table 3.8) to examine the original distributional similarity features. There are 3 distributional similarity features that achieve F_1 of more than 30 points, and all three represent features using *pairs* of argument instantiations rather than treat each argument separately, as we have already witnessed in Section 3.4.

Note also that the feature *h-b-L-pCt*, which uses binary templates, the *Lin* similarity measure and features that are pairs of CUI tuples is the best feature both in terms of the ablation test and when it is used as a single feature for the classifier. The result obtained by this feature is only 3.3 points lower than the one when using the entire feature set. We believe this is the result of two reasons: First, the 16 distributional similarity features are correlated with one another and thus using all of them does

not boost performance substantially. For example, the Pearson correlation coefficients between the features *h-b-B-pCt, h-b-B-Ct, h-b-L-pCt, h-b-L-Ct, h-u-B-Ct, h-u-L-Ct* (all utilize CUI tuples) and *h-b-B-pC, h-b-B-C, h-b-L-pC, h-b-L-C, h-u-B-C, h-u-L-C* (all use CUIs) respectively are over 0.9. The second reason for gaining only 3.3 points by the remaining features is that, as discussed, the new set of features is relatively sparse.

To sum up, we suggest several hypotheses that explain our results and analysis:

- The new features are too sparse to substantially improve the performance of the local entailment classifier in our data set. This perhaps can be attributed to the nature of our domain-specific health-care corpus. In Chapter 5 we will employ similar features in a domain-general setting and demonstrate that in this setting the new features contribute to performance.

- Looking at the training set accuracy, ablations and precision of the new features, it seems that the behavior of most of them is reasonable. Thus, it is possible that in a a different learning scheme that does not use the resources as features the information they provide may become beneficial. For example, in a simple "back-off" approach one can use rules from precise resources to determine entailment, and apply a classifier only when no precise resource contains a relevant rule.

- In our corpus representing distributional similarity features with pairs of argument instantiations is better than treating each argument independently .

- Given the current training set accuracy and the sparsity of the new features, it is important to develop methods that gather large scale information that is orthogonal to distributional similarity. In our opinion,the most promising direction for acquiring such rich information is co-occurrence-based methods (see Section 2.2.1.2).

3.6 Conclusions

This chapter presented a global optimization algorithm for learning predicative entailment rules. Most previous work on learning such rules focused on local learning methods, which consider each pair of predicates in isolation. We modeled the problem as a graph learning problem, and searched for the best graph under a global transitivity

constraint. Two objective functions were defined for the optimization procedure, one score-based and the other probabilistic, and we have shown that under certain conditions the score-based function can be interpreted probabilistically. This allowed us to use both margin as well as probabilistic classifiers for the underlying entailment classifier. We solved the optimization problem using Integer Linear Programming, which provides an optimal solution (compared to the greedy algorithm suggested by Snow et al.), and demonstrated empirically that this method outperforms local algorithms as well as Snow et al.'s greedy optimization algorithm on the graph learning task. We also analyzed quantitatively an qualitatively the reasons for the improved performance of our global algorithm and performed detailed error analysis. Last, we experimented with various entailment classifiers that utilize different sets of features from many knowledge bases.

An important next step for our work is apply our algorithm to graphs that are much larger than the focused entailment graphs dealt with in this chapter. This will introduce a challenge to our current optimization algorithm due to complexity issues, as our ILP contains $O(|V|^3)$ constraints. In addition, this will raise the issue of predicate ambiguity, which interferes with the transitivity of entailment. The focus of Chapter 4 is to develop more scalable algorithms that are able to learn large transitive graph, and we will show how to do this efficiently, while still improving over local learning methods. We will also show how to deal with the problem of ambiguity by *typing* of the predicate's arguments. The methods developed in Chapter 4 will also be applied in large scale in Chapter 5.

The experiments and analysis performed in this chapter indicate that the current performance of the local entailment classifier needs to improved. In Chapter 5 we will train a local entailment classifier over both lexicographic as well as distributional similarity features and demonstrate that this richer set of features improves performance when trained over a large domain-general corpus. Nevertheless, we believe that a promising future direction for substantially improving the local classifier is to exploit co-occurrence-based information, since this is an orthogonal source of information that excels at identifying specific semantic relations.

Last, in Section 3.1.1 we mentioned that by merging strongly connected components in entailment graphs hierarchies of predicates can be generated (recall Figure

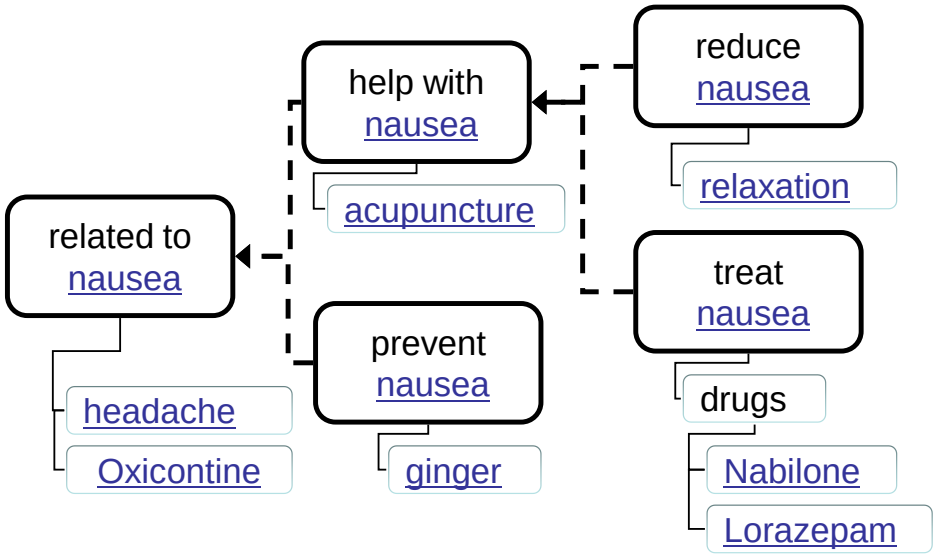

Figure 3.8: A hierarchical summary of propositions involving *nausea* as an argument, such as *headache is related to nausea*, *acupuncture helps with nausea*, and *Lorazepam treats nausea*.

3.1). We believe that these hierarchies can be useful not only in the context of semantic inference applications, but also in the field of faceted search and hierarchical text exploration (169). Figure 3.8 exemplifies how a set of propositions can be presented to a user according to the hierarchy of predicates shown in Figure 3.1. In the field of faceted search, information is presented using a number of hierarchies, corresponding to different facets or dimensions of the data. One can easily use the hierarchy of predicates learned by our algorithm as an additional facet in the context of a text exploration application. In Chapter 6 we will present a concrete implementation of this application over the health-care corpus utilized in this chapter.

Name	Type	Description	#	%
Hyper.	boolean	Whether (t_1, t_2) are identical except for a pair of words (w_1, w_2) such that w_2 is a hypernym (distance ≤ 2) of w_1 in WordNet.	120	1.1
Syno.	boolean	Whether (t_1, t_2) are identical except for a pair of words (w_1, w_2) such that w_2 is a synonym of w_1 in WordNet.	94	0.9
Co-hypo.	boolean	Whether (t_1, t_2) are identical except for a pair of words (w_1, w_2) such that w_2 is a co-hyponym of w_1 in WordNet.	302	2.8
WN Ant.	boolean	Whether (t_1, t_2) are identical except for a pair of words (w_1, w_2) such that w_2 is an antonym of w_1 in WordNet.	6	0.06
VO Ant.	boolean	Whether (t_1, t_2) are identical except for a pair of words (w_1, w_2) such that w_2 is an antonym of w_1 in VerbOcean.	25	0.2
WN Ant. 2	boolean	Whether there exists in (t_1, t_2) a pair of words (w_1, w_2) such that w_2 is an antonym of w_1 in WordNet.	22	0.2
VO Ant. 2	boolean	Whether there exists in (t_1, t_2) a pair of words (w_1, w_2) such that w_2 is an antonym of w_1 in VerbOcean.	73	0.7
Derivation	boolean	Whether there exists in (t_1, t_2) a pair of words (w_1, w_2) such that w_2 is a derivation of w_1 in WordNet or CATVAR.	78	0.7
Entailment	boolean	Whether there exists in (t_1, t_2) a pair of words (w_1, w_2) such that w_2 is entailed by w_1 in WordNet.	20	0.2
FRED	boolean	Whether t_1 entails t_2 in FRED.	9	0.08
Nomlex	boolean	Whether t_1 entails t_2 in Nomlex.	8	0.07
VO strong	boolean	whether (t_1, t_2) are identical except for a pair of words (w_1, w_2) such that w_2 is *stronger than* w_1 in VerbOcean.	104	1
VO simil.	boolean	Whether (t_1, t_2) are identical except for a pair of words (w_1, w_2) such that w_2 is *similar to* w_1 in VerbOcean.	191	1.8
Positive	boolean	Disjunction of the features Hypernym, Synonym, Nomlex and VO stronger.	289	2.7
BAP	real	$max_{w_1 \in t_1, w_2 \in t_2} BAP(w_1, w_2)$	506	4.7
Edit	real	normalized edit-distance		100

Table 3.12: The set of new features. The last two columns denote the number and percentage of examples for which the value of the feature is non-zero in examples generated from the 23 gold standard graphs.

	Development set			Test set		
Algorithm	**Recall**	**Precision**	F_1	**Recall**	**Precision**	F_1
Linear_{all}	48.1	31.9	**36.3**	51.7	37.7	40.3
Linear_{old}	40.3	33.3	34.8	47.2	42.2	41.1
Gaussian_{all}	41.8	32.4	35.1	48.0	41.1	40.7
Gaussian_{old}	41.1	31.2	33.9	50.3	39.7	40.5
Square_{all}	39.9	32.0	34.1	43.7	39.8	38.9
Square_{old}	38.0	31.6	32.9	50.2	41.0	**41.3**
Logistic_{all}	34.4	27.6	29.1	39.8	41.7	37.8
Logistic_{old}	39.3	31.2	33.5	45.4	40.9	39.9
Bayes_{all}	20.8	33.2	24.5	27.4	46.0	31.7
Bayes_{old}	20.3	34.9	24.6	26.4	45.4	30.9

Table 3.13: macro-average recall, precision and F_1 on the development set and test set using the parameters that maximize F_1 of the learned edges over the development set.

Feature name	%	Ablation F_1	Δ	Fea. type	Prec.	Recall	Classification F_1
h-b-B-pCt	8.2	39.3	-1				14.9
h-b-B-pC	6.9	39.5	-0.8				33.2
h-b-B-Ct	1.6	40.3	0				15.4
h-b-B-C	1.6	40.5	0.2				11.2
h-b-L-pCt	23.6	38.3	**-2.0**				**37.0**
h-b-L-pC	21.4	39.4	-0.9				35.2
h-b-L-Ct	9.7	40.1	-0.2				27.3
h-b-L-C	8.1	39.7	-0.6				14.1
h-u-B-Ct	1.0	39.4	-0.9				10.9
h-u-B-C	1.1	39.8	-0.5				12.6
h-u-L-Ct	6.1	39.8	-0.5				18.5
h-u-L-C	6.3	39.2	-1.1				19.3
R-b-L-l	22.5	40.1	-0.2				26.7
R-u-L-l	8.3	39.4	-0.9				23.2
R-u-B-l	1.9	39.8	-0.5				16.7
Lin & Pantel	8.8	38.7	-1.6				23.0
Hyper.	1.1	38.7	-1.6	+	37.1	4.9	9.7
Syno.	0.9	40.3	0	+	43.1	4.5	15.8
Co-hypo.	2.8	40.1	-0.2	-	82.0	2.5	17.9
WN ant.	0.06	39.8	-0.5	-	75.0	0.05	1.2
VO ant.	0.2	40.1	-0.2	-	96.0	0.2	2.2
WN ant. 2.	0.2	39.4	-0.9	-	59.1	0.1	2.7
VO ant. 2	0.7	40.2	-0.1	-	98.6	0.7	2.2
Derivation	0.7	39.5	-0.8	+	47.4	4.1	10.2
Entailment	0.2	39.7	-0.6	+	15.0	0.3	1.2
FRED	0.08	39.7	-0.6	+	77.8	0.8	3.2
NOMLEX	0.07	39.8	-0.5	+	75.0	0.7	3.3
VO strong.	1	39.4	-0.9	+	34.6	4	6.9
VO simil.	1.8	39.4	-0.9	+	28.8	6.1	12.5
Positive	2.7	39.8	-0.5	+	36.7	11.8	
BAP	4.7	40.1	-0.2				13.3
Edit	100	39.9	-0.4				15.5

Table 3.14: Results of feature analysis. The second column denotes the proportion of manually annotated examples for which the feature value is non-zero. A detailed explanation of the other columns is provided in the body of the chapter.

4

Optimization Algorithms

In Chapter 3 we modeled our problem as an ILP and employed standard ILP solvers to reach the optimal solution. Since ILP is NP-hard this is not scalable. In this chapter, we present two algorithms that take advantage of structural properties of entailment graphs to improve scalability.

The first algorithm is based on the structural assumption that entailment graphs are relatively sparse, that is, most predicates do not entail one another. This leads to an exact algorithm that first decomposes the graph into smaller components and then applies an ILP solver on each of the components to reach an optimal solution.

In the second algorithm, we first identify that entailment graphs exhibit a tree-like property and are very similar to a novel type of graph termed *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 set of edges in forest-reducible graphs is NP-hard.

We apply both algorithms over a large data set of extracted propositions, from which a resource of *typed* entailment rules has been recently released (153). Our results show that using global transitivity information substantially improves performance over this resource and several baselines, and that our methods improve scalability and allow us to increase the scope of global learning of entailment-rule graphs.

4.1 An Exact Algorithm for Learning Typed Entailment Graphs

As previously discussed, there are two main challenges in *global* learning. The first is the issue of scalability, which we will consider in Section 4.1.2. The second is the issue of ambiguity, that is, the fact that transitivity does not always hold when predicates are ambiguous. In the previous chapter, we circumvented the problem of ambiguity by learning rules where one of the predicate's arguments is instantiated (e.g., *'X reduce nausea ⇒ X affect nausea'*), which was useful for learning small graphs on-the-fly, given a target concept such as *'nausea'*. While rules may be effectively learned when needed, their scope is narrow and they are not useful as a generic knowledge resource. In this chapter, we overcome the problem of ambiguity by adopting a rule representation suggested by Schoenmackers et al. (153).

Schoenmackers et al. (153) proposed a *local learning* algorithm for learning horn clauses (see Section 2.2.1.1). They used distributional similarity to learn rules between *typed predicates*, i.e., predicates where the argument types (e.g., *'city'* or *'drug'*) are specified, and allowed the left-hand-side of the rule to contain more than a single predicate. In their work, they used Hearst-patterns (81) to extract a set of 29 million *'(argument, type)'* pairs from a large web crawl. Then, they employed several filtering methods to clean this set and automatically produced a mapping of 1.1 million arguments into 156 types. Examples for *'(argument, type)'* pairs are *'(EXODUS, book)'*, *'(CHINA, country)'* and *'(ASTHMA, disease)'*. Then, they utilized the types, the mapped arguments and tuples from TextRunner (7) to generate 10,672 typed predicates (such as *'conquer($X_{country}$, Y_{city})'* and *'common in($X_{disease}$, Y_{place})'*), and learned 30,000 rules between these predicates[1].

Typing predicates was important for Schoenmackers et al. since they were dealing with noisy and ambiguous web text. Typing substantially reduces ambiguity and helps filtering of noise, while still maintaining rules of wide-applicability. In this chapter we take global learning one step further – we adopt Schoenmackers et al.'s representation to avoid ambiguity, but we do it in the context of *global learning* rather than *local*

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

learning. This allows us to create a more precise knowledge-base of rules that is useful for inference systems.

We term the rules learned in this chapter *typed entailment rules*, and apply our learning algorithm in a domain-general setting. We first show how to construct a structure termed *typed entailment graph*, where the nodes are typed predicates and the edges represent entailment rules. We then suggest scaling techniques that allow to optimally learn such graphs over a large set of typed predicates by first *decomposing* nodes into components and then applying *incremental ILP* (144). Using these techniques, the obtained algorithm is guaranteed to return an optimal solution. We ran our algorithm over the data set of Schoenmackers et al. and released a resource of 30,000 rules[1] that achieves substantially higher recall without harming precision. To the best of our knowledge, this is the first resource of that scale to use global optimization for learning predicative entailment rules. Our evaluation shows that global transitivity improves the F_1 score of rule learning by 27% over several baselines and that our exact method allows dealing with larger graphs, resulting in improved coverage.

4.1.1 Typed entailment graphs

Given a set of typed predicates, entailment rules can only exist between predicates that share the same (unordered) pair of types (such as *'place'* and *'country'*), 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 (Figure 4.1). Next, we show how to represent entailment rules between typed predicates in a structure termed *typed entailment graph*, which will be the learning goal of our algorithm.

A typed entailment graph is a directed graph where the nodes are *typed predicates*. A typed predicate is a triple (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 [2].

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

[2] Denoting the typed variables and the types themselves in the same way is an abuse of notation. However, notation meaning will always be clear from context, and this will keep the amount of notation minimal.

Condition	Consequence
$(i \xrightarrow{d} j) \wedge (j \xrightarrow{d} k)$	$i \xrightarrow{d} k$
$(i \xrightarrow{d} j) \wedge (j \xrightarrow{r} k)$	$i \xrightarrow{r} k$
$(i \xrightarrow{r} j) \wedge (j \xrightarrow{d} k)$	$i \xrightarrow{r} k$
$(i \xrightarrow{r} j) \wedge (j \xrightarrow{r} k)$	$i \xrightarrow{d} k$

Table 4.1: Transitivity calculus for single-type entailment graphs.

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 $a_2 \in t_2$ are termed *arguments*. For example, *'be common in(ASTHMA,AUSTRALIA)'* is an instance of *'be common in(disease,place)'*. For brevity, we refer to typed entailment graphs and typed predicates as *entailment graphs* and *predicates* respectively.

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

Since entailment is a transitive relation, typed-entailment graphs are transitive: if the edges (i, j) and (j, k) are in the graph so is the edge (i, k). Note that in single-type entailment graphs one needs to consider whether mapping of edges is direct or reversed: if mapping of both (i, j) and (j, k) is either direct or reversed, mapping of (i, k) is direct, otherwise it is reversed (see Table 4.1).

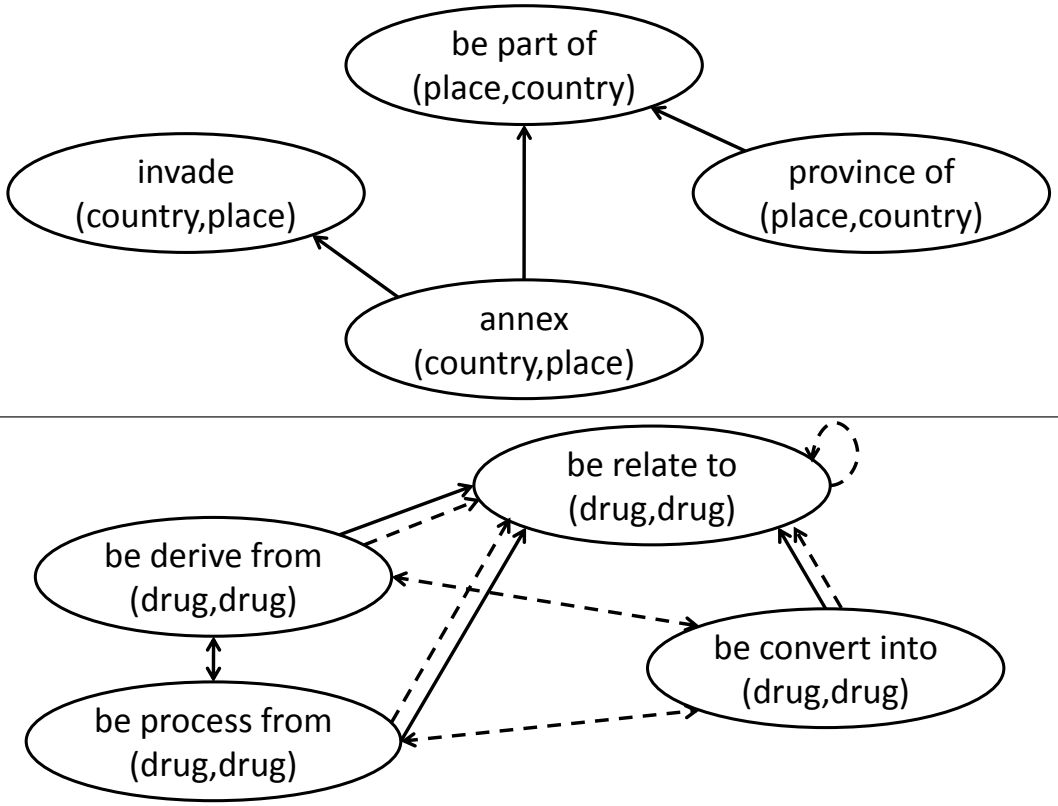

Figure 4.1: *Top:* A fragment of a two-types entailment graph. *bottom:* A fragment of a single-type entailment graph. Mapping of solid edges is direct and of dashed edges is reversed.

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

4.1.2 Learning typed entailment graphs

Our learning algorithm follows the same procedure described in Chapter 3: (1) Given a set of typed predicates and their instances extracted from a corpus, we train a *local entailment classifier* that estimates for every pair of predicates whether one entails the other. (2) Using the classifier scores we perform global optimization, i.e., learn the set of edges over the nodes that maximizes the global score of the graph under transitivity

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)* $\nRightarrow$ *bomb(country,city)*
hyponym (distance=2)	*defeat(city,city)* $\nRightarrow$ *eliminate(city,city)*
random	*hold(place,event)* $\nRightarrow$ *win(place,event)*

Table 4.2: Automatically generated training set examples

and background-knowledge constraints.

Section 4.1.2.1 briefly describes the local classifier training procedure. Section 4.1.2.2 gives an ILP formulation for the optimization problem. Sections 4.1.2.4 and 4.1.2.5 propose scaling techniques that exploit graph sparsity to optimally solve larger graphs.

4.1.2.1 Training a local entailment classifier

Training of the local classifier is performed as described in Section 3.2.1. We will now outline the slight modifications made to adapt the method to the new corpus. The input is a lexicographic resource (WordNet) and a set of typed propositions, and the following steps are performed:

1. **Training set generation** Positive examples are again generated using WordNet synonyms and hypernyms. Negative pairs are generated again using WordNet direct co-hyponyms (sister terms), but we also utilize Word hyponyms at distance 2. In addiiton, we generate negative examples by randomly sampling pairs of typed predicates that share the same types. Our goal here is to generate a more diverse set of negative examples. Table 4.2 provides an example for each type of automatically generated example.

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

We compute 11 distributional similarity scores for each pair of binary typed predicates based on the extracted arguments. The first 6 scores are computed by trying all combinations of two similarity functions *Lin* and *BInc* with three types of feature representations: (1) a feature is a pair of arguments (*Tease* representation). (2) predicate representation is binary, and each typed predicate has two feature vectors, one for the X slot and one for the Y slot, which are combined by a geometric average (*DIRT* representation). (3) binary typed predicates are decomposed into two unary typed predicates, and similarity is computed separately for each unary predicate, and then combined by a geometric average.

The other 5 scores were provided by Schoenmackers et al. (153) and include *SR* (153), *LIME* (115), *M-estimate* (54), the standard *G-test* and a simple implementation of *Cover* (183). Overall, the rationale behind this representation is that combining various scores will yield a better classifier than each single measure [1].

3. **Training** We again train over an equal number of positive and negative examples.

4.1.2.2 ILP formulation

Once the classifier is trained, we would like to learn all edges (entailment rules) of each typed entailment graph. Given a set of predicates V and an entailment score function $w : V \times V \to \mathbb{R}$ derived from the classifier, we want to find a graph $G = (V, E)$ that respects transitivity and maximizes the sum of edge weights $\sum_{(i,j)\in E} w(i, j)$.

For two-types entailment graphs the formulation is identical to the one given in Equations 3.1-3.5 (Section 3.2.2). Recall that for a graph with n nodes we get $n(n-1)$ variables and $n(n-1)(n-2)$ transitivity constraints.

The simplest way to expand this formulation to single-type graphs is to duplicate each predicate node, with one node for each order of the types, and then the ILP is unchanged. However, this is inefficient as it results in an ILP with $2n(2n-1)$ variables and $2n(2n-1)(2n-2)$ transitivity constraints. Since our main goal is to scale the use of ILP, we modify it a little. We denote a direct-mapping edge (i, j) by the indicator

[1] To compute similarity, we represent each predicate with the *expected* count of its instances and not the count, because a typed instance may belong to more than one typed predicate due to the fact that an argument may belong to more than one type. To compute expectations, we use a distribution of types for each argument provided by (153).

x_{ij} and a reversed-mapping edge (i, j) by y_{ij}. The functions w_d and w_r provide scores for direct and reversed mappings respectively. Then, the following ILP is formulated:

$$\hat{G} = \text{argmax} \sum_{i \neq j} w_d(i,j) \cdot x_{ij} + \sum_{i,j} w_r(i,j) \cdot y_{ij} \tag{4.1}$$

$$\text{s.t. } \forall_{i,j,k \in V} \; x_{ij} + x_{jk} - x_{ik} \leq 1 \tag{4.2}$$

$$\forall_{i,j,k \in V} \; x_{ij} + y_{jk} - y_{ik} \leq 1 \tag{4.3}$$

$$\forall_{i,j,k \in V} \; y_{ij} + x_{jk} - y_{ik} \leq 1 \tag{4.4}$$

$$\forall_{i,j,k \in V} \; y_{ij} + y_{vw} - x_{ik} \leq 1 \tag{4.5}$$

$$\forall_{x_{ij} \in A_{yes}} \; x_{ij} = 1 \tag{4.6}$$

$$\forall_{y_{ij} \in A_{yes}} \; y_{ij} = 1 \tag{4.7}$$

$$\forall_{x_{ij} \in A_{no}} \; x_{ij} = 0 \tag{4.8}$$

$$\forall_{y_{ij} \in A_{no}} \; y_{ij} = 0 \tag{4.9}$$

$$\forall_{i \neq j} \; x_{ij}, y_{ij} \in \{0, 1\} \tag{4.10}$$

The new objective function sums the weights over both direct-mapping edges and reversed-mapping edges (note that reversed mapping loops are allowed). The transitivity constraint is replaced by four transitivity constraints that mirror the calculus we presented in Table 4.1. This results in $2n^2 - n$ variables and about $4n^3$ transitivity constraints, cutting the ILP size in half.

To obtain the function w, we use the derivation presented in Section 3.3[1]:

$$\hat{G} = \underset{G}{\text{argmax}} \log P(G|F) = \text{argmax} \sum_{i \neq j} (\log \frac{P_{ij}}{1 - P_{ij}}) x_{ij} + \log \eta \cdot |E| \tag{4.11}$$

where $\eta = \frac{P(x_{ij}=1)}{P(x_{ij}=0)}$ and $w(i,k) = \log \frac{P_{ij} \cdot P(x_{ij}=1)}{(1-P_{ij})P(x_{ij}=0)}$. Note that w is composed of a likelihood component and an *edge prior* expressed by $P(x_{ij} = 1)$. In the next section, we present two approaches for modeling this edge prior.

[1]We focus on two-types graphs but extending to single-type graphs is trivial.

4.1.2.3 Modeling the edge prior

A simple approach for modeling the prior is to assume that it is *constant*. This is reasonable in the small graphs presented in the previous chapter. However, if this prior is constant, then this means that in a directed graph with n nodes and m edges, the expected *density* $\frac{m}{n(n-1)}$ is also constant. If density is a constant C, then $E(m) = Cn(n-1)$. If we denote the average node degree by D, then by standard graph theory $D = \frac{2m}{n}$ and so $E(D) = 2C(n-1)$. Thus, the expected average degree grows *linearly* with the graph, which seems counterintuitive: it is reasonable that there is some bound on the number of predicates entailing or entailed by a predicate regardless of the graph size.

An alternative approach would be to assume that the expected average degree of a node d is constant. This seems more intuitive, and by a similar derivation we can estimate $P(x_{ij} = 1) = \frac{m}{n(n-1)} = \frac{d}{2(n-1)}$[1]. This would indeed mean that sparsity decreases as the number of nodes grows.

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

4.1.2.4 Graph decomposition

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

Let V be a set of nodes and I, J a partitioning of V into two disjoint non-empty subsets. We term any edge (i, j) or (j, i), such that $i \in I$ and $j \in J$ a *crossing edge*.

Proposition 4.1.1. *If we can partition a set of nodes V into disjoint sets I, J such that for any crossing edge (i, j) (and (j, i)) between them, $w(i, j) < 0$ (and $w(j, i) < 0$), then the optimal set of edges E_{opt} does not contain any crossing edge.*

[1] For single-type graphs $P(x_{ij} = 1) = \frac{d}{2(2n-1)}$

Algorithm 2 Decomposed-ILP

Input: A set V and a function $w : V \times V \to \mathbb{R}$

Output: An optimal set of directed edges E^*

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

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

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

This proposition suggests a simple exact algorithm (see Algorithm 2): Add to the graph an undirected edge for any node pair with a positive score, then find the connected components, and apply an ILP solver over the nodes in each component. The edges returned by the solver provide an optimal (not approximate) solution to the optimization problem.

Finding the undirected edges (Line 1) and computing connected components (Line 2) can be performed in $O(V^2)$. Thus, the efficiency of the algorithm is dominated by the application of an ILP solver (Line 4). Consequently, efficiency depends on whether the graph is sparse enough to be decomposed into small enough components. Note that the edge prior plays an important role: low values make the graph sparser and easier to solve. In Section 4.1.3 we empirically test how typed entailment graphs benefit from decomposition given different prior values. It is also interesting to note that the algorithm can be easily parallelized by solving each component on a different core.

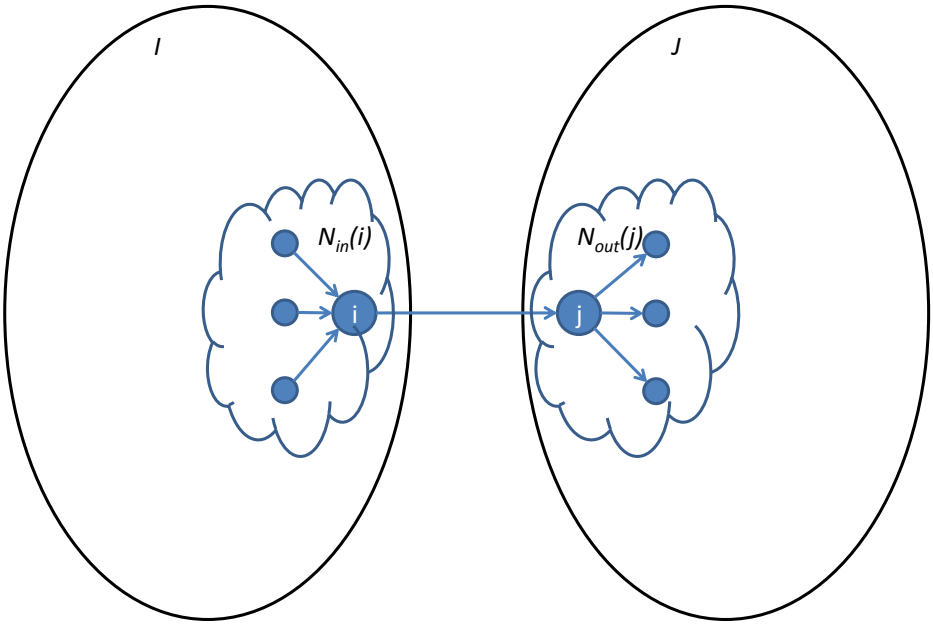

Figure 4.2: Two components I and J for which there is a single pair of nodes (i, j) such that $w(i, j) > 0$.

Algorithm 2 is able to reduce size of an ILP when the graph decomposes into small components, and we will empirically employ Algorithm 2 in Section 4.1.3. However, it is possible to generalize this algorithm and reduce the size of the ILP when the graph components are connected by a single edge. Given a graph $G = (V, E)$ and the nodes i, j we denote by $N_{in}(i)$ the set of nodes with an outgoing edge into i including i and by $N_{out}(j)$ the set of nodes with an incoming edge from j including j (see Figure 4.2). For any subset of nodes $U \subseteq V$, we can define an optimal set of edges E^U_{opt} with respect to U by narrowing the scope of the weighting function to $w^* : U \times U \to \mathbb{R}$. Given two subsets of nodes $U, W \subseteq V$, we say that E^U_{opt} *agrees with* E^W_{opt} if for any pair of nodes $i, j \in U \cap W$ either $(i, j) \in E^U_{opt}$ and $(i, j) \in E^W_{opt}$, or $(i, j) \notin E^U_{opt}$ and $(i, j) \notin E^W_{opt}$.

Proposition 4.1.2. *Assume we can partition a set of nodes V into disjoint sets I, J such that the weight of all crossing edges is negative except for a single edge $(i,j), i \in I, j \in J$ for which $w(i,j) > 0$. Let E^I_{opt}, E^J_{opt}, and $E^{IJ}_{opt} = E^{N_{in}(i) \cup N_{out}(j)}_{opt}$ be the optimal sets of edges with respect to their corresponding subsets of nodes. If both E^I_{opt} and E^J_{opt}* agree with *E^{IJ}_{opt}, then the optimal set of edges is $E_{opt} = E^I_{opt} \cup E^J_{opt} \cup E^{IJ}_{opt}$.*

Proof We first claim that E_{opt} does not violate any transitivity constraints. Clearly, $E^I_{opt} \cup E^J_{opt}$ does not violate transitivity, as both E^I_{opt} and E^J_{opt} respect transitivity and I and J are disjoint. Since E^{IJ}_{opt} agrees with E^I_{opt} and E^J_{opt}, then $E^I_{opt} \cup E^J_{opt} \cup E^{IJ}_{opt}$ simply adds to $E^I_{opt} \cup E^J_{opt}$ some crossing edges. Thus, violations of transitivity constraints are due to some crossing edge. Note also that since there is just a single crossing edge with positive weight, all crossing edges must be from I to J. Assume by contradiction that there is a crossing edge (u,v) that participates in a transitivity violation. Then, without loss of generality, E_{opt} contains an edge (v,w) and does not contain the edge (u,w). Clearly, $u, v \in N_{in}(i) \cup N_{out}(j)$ since (u,v) was added by E^{IJ}_{opt}. In addition, w has an incoming edge from v and so $w \in N_{out}(v)$. This means that u, v, w, are all in $N_{in}(i) \cup N_{out}(j)$, and so E^{IJ}_{opt} violates transitivity, Contradiction.

It is easy to verify that E_{opt} is the optimal solution. Given a set of nodes U and a set of edges E, let $S_E(U) = \sum_{\{i,j \in U : (i,j) \in E\}} w(i,j)$. Clearly, for any $U \subseteq V$, $S_{E^U_{opt}}(U) \geq S_{E^V_{opt}}(U)$. This is because the optimal solution in the subset of nodes is less constrained. Since in our case E_{opt} agrees with its two disjoint subsets E^I_{opt} and E^J_{opt}, then there can not be any changes inside I and J that will improve the objective function. Thus, the only edge that can improve the objective function is (i,j) and by considering E^{IJ}_{opt} (in case it agrees with E^I_{opt} and E^J_{opt}) we determine whether (i,j) should be added or not. □

Proposition 4.1.2 suggests another optimization algorithm. Given a set of nodes V and the weighting function w, we can look for the minimal cut in edges of V. If the minimal cut contains no edges we can apply an ILP solver on the two components. If the minimal cut contains a single edge (i,j), we can apply an ILP solver on the two components, compute $N_{in}(i)$ and $N_{out}(j)$, apply an ILP solver on $N_{in}(i) \cup N_{out}(j)$, and if E^I_{opt} and E^J_{opt} agree with E^{IJ}_{opt}, then $E_{opt} = E^I_{opt} \cup E^J_{opt} \cup E^{IJ}_{opt}$. The algorithm can also be applied iteratively on the two components. Another generalization can be formulated when the two components are connected by a small number of edges, but all pointing in the same direction (either from I to J or from J to I). These are all

Algorithm 3 Incremental-ILP

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

interesting directions for theoretical and empirical research, but we will not discuss them further in this dissertation.

4.1.2.5 Incremental ILP

Another solution for scaling ILP is to employ *incremental ILP*, also known as *cutting-plane method*, which has been used in dependency parsing (144). The idea is that even if we omit the transitivity constraints, we still expect most transitivity constraints to be satisfied, given a good local entailment classifier. Thus, it makes sense to avoid specifying the constraints ahead of time, but rather add them when they are violated. This is formalized in Algorithm 3.

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

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

Combining *Incremental-ILP* and *Decomposed-ILP* is easy: We decompose any large graph into its components and apply Incremental ILP on each component. We applied

this algorithm on our evaluation data set (Section 4.1.3) and found that it converges in at most 6 iterations and that the maximal number of active constraints in large graphs drops from $\sim 10^6$ to $\sim 10^3 - 10^4$.

4.1.3 Experimental evaluation

In this section we empirically answer the following questions: (1) Does transitivity improve rule learning over typed predicates? (Section 4.1.3.1) (2) Do *Decomposed-ILP* and *Incremental-ILP* improve scalability? (Section 4.1.3.2)

4.1.3.1 Experiment 1

A data set of 1 million TextRunner tuples (7), mapped to 10,672 distinct typed predicates over 156 types was provided by Schoenmackers et al. (153). Readers are referred to their paper for details on mapping of tuples to typed predicates. Since entailment only occurs between predicates that share the same types, we decomposed predicates by their types (e.g., all predicates with the types *'place'* and *'disease'*) into 2,303 *typed entailment graphs*. The largest graph contains 118 nodes and the total number of potential rules is 263,756.

We generated a training set by applying the procedure described in Section 4.1.2.1, yielding 2,644 examples. We used SVMperf (89) to train a Gaussian kernel classifier and computed P_{ij} by projecting the classifier output score, S_{ij}, with the sigmoid function: $P_{ij} = \frac{1}{1+exp(-S_{ij})}$ (see Section 3.2.2.3). We tuned two SVM parameters using 5-fold cross validation and a development set of two typed entailment graphs.

Next, we used our algorithm to learn rules, using the *lpsolve* package. As mentioned in Section 4.1.2.2, we integrate background knowledge using the sets A_{yes} and A_{no} that contain predicate pairs for which we know whether entailment holds. A_{yes} was constructed with syntactic rules: We normalized each predicate by omitting the first word if it is a modal and turning passives to actives. If two normalized predicates are equal they are synonymous and inserted into A_{yes}. A_{no} was constructed from 3 sources (1) Predicates differing by a single pair of words that are WordNet antonyms (2) Predicates differing by a single word of negation (3) Predicates $p(t_1, t_2)$ and $p(t_2, t_1)$ where p is a transitive verb (e.g., *beat)* in VerbNet (93). In addition, we experimented with two priors – one where the expected graph density is constant and the other where the expected average degree is constant (see Section 4.1.2.3). Performance was

comparable with a slight advantage for constant density, and so this is the option that we will report.

We compared our algorithm (termed ILP_{scale}) to the following baselines. First, to 10,000 rules released by Schoenmackers et al. (153) (*Sherlock*), where the LHS contains a single predicate (Schoenmackers et al. released 30,000 rules but 20,000 of those have more than one predicate on the LHS, see Section 4.1), as we learn rules over the same data set. Second, to distributional similarity algorithms: (a) *SR:* the score used by Schoenmackers et al. as part of the *Sherlock* system. (b) *DIRT* (104) (c) *BInc* (172). Third, we compared to the entailment classifier with no transitivity constraints (*clsf*) to see if combining distributional similarity scores improves performance over single measures. Last, we added to all baselines background knowledge with A_{yes} and A_{no} (adding the subscript X_k to their name).

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

Figure 4.3 presents the precision-recall curve of the algorithms. In all algorithms adding background knowledge improved performance so we only present results for algorithms that are supplied background knowledge. The curve is formed by varying a score threshold in the baselines and varying the edge prior in ILP_{scale}[1]. For figure clarity, we omit *DIRT* and *SR*, since *BInc* outperforms them.

Table 4.3 shows micro-recall, precision and F_1 at the point of maximal F_1, and the Area Under the Curve (AUC) for recall in the range of 0.08-0.45 for all algorithms[2]. The table also shows results for the rules from $Sherlock_k$.

Results show that using global transitivity information substantially improves performance. ILP_{scale} is better than all other algorithms by a large margin starting from recall .2, and improves AUC by 29% and the maximal F_1 by 27%. Moreover, ILP_{scale}

[1] we stop raising the prior when run time over the graphs exceeds 2 hours.

[2] We start at recall 0.08, since background knowledge alone provides recall of 0.08 with perfect precision.

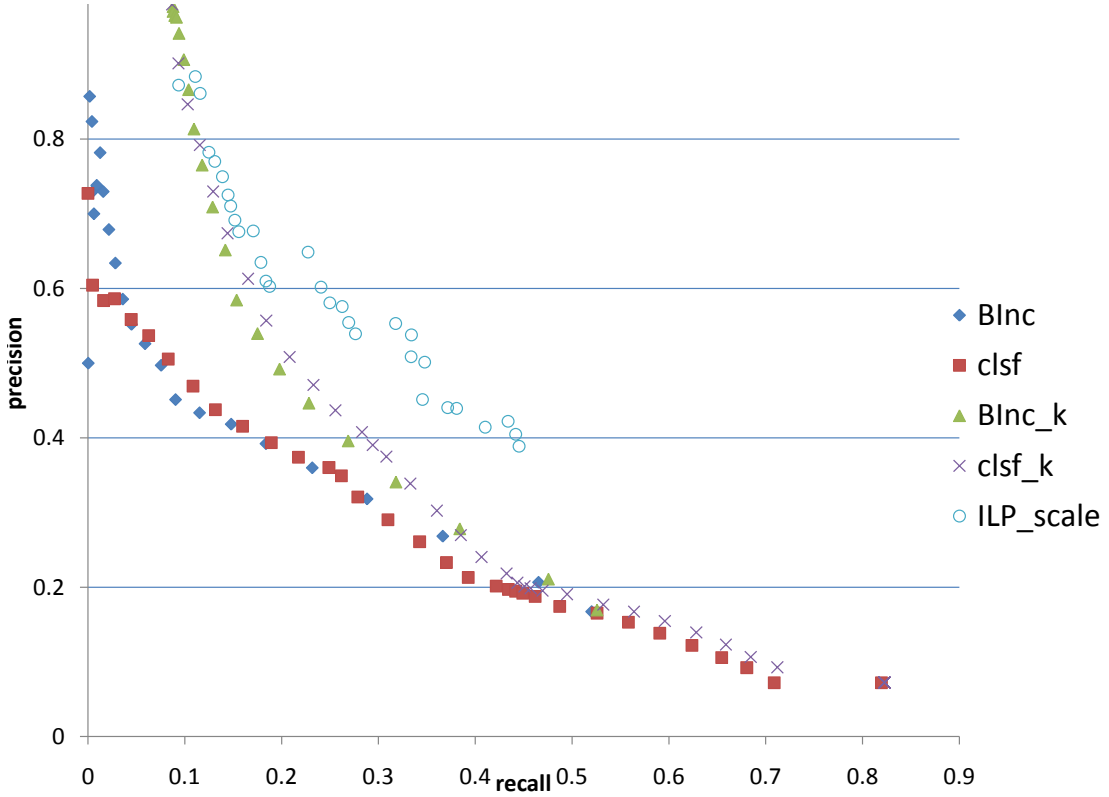

Figure 4.3: Precision-recall curve for the algorithms.

doubles recall comparing to the rules from the *Sherlock* resource, while maintaining comparable precision.

Results also show that the entailment classifier improved very little over the best distributional similarity algorithm, possibly because distributional scores are too correlated to one another to boost performance and orthogonal features are required to improve performance.

4.1.3.2 Experiment 2

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

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

Table 4.3: micro-average F_1 and AUC for the algorithms.

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

Table 4.4: Impact of scaling techinques (ILP^{-}/ILP_{scale}).

We used the same data set as in Experiment 1 and learned edges for all 2,303 entailment graphs in the data set. If the ILP solver was unable to hold the ILP in memory or took more than 2 hours for some graph, we did not attempt to learn its edges. We ran ILP_{scale} and ILP^{-} in three density modes to examine the behavior of the algorithms for different graph densities: (a) $\log \eta = -0.6$: the configuration that achieved the best recall/precision/F_1 of 43.4/42.2/42.8. (b) $\log \eta = -1$ with recall/precision/F_1 of 31.8/55.3/40.4. (c) $\log \eta = -1.75$: A high precision configuration with recall/precision/F_1 of 0.15/0.75/0.23. Experiments were run on an Intel i5 CPU with 1.5GB of virtual memory.

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

Table 4.4 shows the results. Column # *unlearned* and # *rules* describe the number of unlearned graphs and the number of learned rules. Column △ shows relative increase in the number of rules learned and column *Reduction* shows the average $1 - r$.

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

Also note that as density increases, the number of rules learned grows and the effectiveness of decomposition decreases. This shows how *Decomposed-ILP* is especially useful for sparse graphs. As mentioned, we released the 29,732 rules learned by the configuration $\log \eta = -0.6$ as a resource.

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

4.1.4 Conclusions

This section proposed two contributions over Chapter 3 and Schoenmackers et al.'s work: Chapter 3 presented a *global* optimization procedure to learn entailment rules between predicates using transitivity, and applied this algorithm over focused entailment graphs, that is, small graphs where all predicates have one argument instantiated by a target concept. Consequently, the rules learned are of limited applicability. Conversely, Schoenmackers et al. learned rules of wider applicability by using *typed predicates*, but utilized a *local* approach.

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

In the next section, we scale the algorithm further and introduce a polynomial approximation algorithm for learning entailment graphs. This is achieved by taking advantage of some more structural properties of entailment graphs.

4.2 Efficient Tree-based Approximation Algorithm

Despite the progress presented in the previous section, finding the exact solution to our optimization problem is still fundamentally NP-hard – recall that we were unable to solve some of the graphs in Section 4.1.3. The method proposed works insofar as the graph decomposes into small components, and thus coverage is still limited. Therefore, scaling to data sets with tens of thousands of predicates (e.g., the extractions of Fader et al. (57)) remains a challenge.

In this section we present a novel method for learning the edges of entailment graphs. This method computes much more efficiently an approximate solution that is almost as good as the exact solution on the data set presented in Section 4.1.3.

To that end, we first (Section 4.2.2) conjecture and empirically show that entailment graphs exhibit a "tree-like" property, i.e., that 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 (Section 4.2.3. Then, we present in Section 4.2.4 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 linear algorithm for node re-attachment. Section 4.2.5 shows empirically that this algorithm is by orders of magnitude faster than the state-of-the-art exact algorithm, and that though an optimal solution is not guaranteed, the area under the precision-recall curve drops by merely a point.

To sum up, the contribution of this section is two-fold: First, we define a novel modeling assumption about the tree-like structure of entailment graphs and 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.

4.2.1 Preliminaries

The method that we present in this section requires two modifications to the setting described in Section 4.1. Next, we describe these two modifications.

The first distinction from the previous section is that we consider only two-types entailment graphs, which are simple directed graphs, and get rid of single-type entailment graphs, which have both direct-mapping edges and reversed-mapping edges. This is done in the manner already hinted at in Section 4.1.1: a typed predicate such as '*beat(team,team)*' is split into two typed predicates *'beat(X_{team},Y_{team})'* and *'beat(Y_{team},X_{team})'*. Then, a direct-mapping edge '*beat(team,team)* $\xrightarrow{d}$ *defeat(team,team)*' is replaced by two equivalent edges: '*beat(X_{team},Y_{team})* $\Rightarrow$ *defeat(X_{team},Y_{team})*' and '*beat(Y_{team},X_{team})* $\Rightarrow$ *defeat(Y_{team},X_{team})*'. In a similar way, a reversed-mapping edge '*beat(team,team)* $\xrightarrow{r}$ *lose to(team,team)*' is replaced by two equivalent edges '*beat(X_{team},Y_{team})* $\Rightarrow$ *lose to(Y_{team},X_{team})*' and '*beat(Y_{team},X_{team})* $\Rightarrow$ *lose to(X_{team},Y_{team})*'. Indeed, transforming single-type entailment graphs into two-types entailment graphs doubles the number of variables and constraints. However, developing our method is made much simpler, and since the algorithm is polynomial the penalty in efficiency is not too large.

Ridding single-type entailment graphs means that the optimization problem we discuss is once again the one described in Section 3.2.2 (Equations 3.1-3.5). In this formulation, the constraints in Equations 3.3-3.4 reflect prior knowledge about some of the candidate edges of the graph. These constraints were easy to encode and use when we employed an ILP solver. However, in this section we present an algorithm that does not utilize an optimization package and thus our second moficiation is to discard them from the formulation.

Removing these constraints is simple. Instead of using constraints to encode prior knowledge, we can take advantage of the weighting function $w : V \times V \rightarrow \mathbb{R}$. For pairs of predicates i, j for which we have prior knowledge that i entails j (termed *positive local constraints*), we set $w_{ij} = \infty$. For pairs of predicates i, j for which we have prior knowledge that i does not entail j (termed *negative local constraints*), we set $w_{ij} = -\infty$[1]. This will force our algorithm to always insert edges in positive local constraints, and to always avoid adding edges in negative local constraints. Note that

[1]Naturally, in practice we have to choose very large positive/negative numbers that are effectively equivalent to ∞/-∞.

we assume here that the positive and negative local constraints do not create violations of transitivity.

With the modified weighting function w, we can reformulate our problem (which we term in this section *Max-Trans-Graph*) more succinctly. Recall that x_{ij} is a binary variable indicating the existence of an edge $i \Rightarrow j$ in E. Then, $\mathcal{X} = \{x_{ij} : i \neq j\}$ are the variables of the following ILP for Max-Trans-Graph:

$$\begin{aligned} &\underset{\mathcal{X}}{\operatorname{argmax}} \sum_{i \neq j} w_{ij} \cdot x_{ij} \\ &\text{s.t. } \forall_{i,j,k \in V} \; x_{ij} + x_{jk} - x_{ik} \leq 1 \\ &\forall_{i,j \in V} \; x_{ij} \in \{0,1\} \end{aligned} \tag{4.12}$$

The method presented in this section provides an approximation for the optimal solution. We remind the reader of two other approaches for ILP approximation recently proposed in the field of NLP.

Do and Roth (52) suggested a method for the related task of learning taxonomic relations between terms. Given a pair of terms, a small graph is constructed and constraints are imposed on the graph structure. Their work, however, is geared towards scenarios where relations are determined on-the-fly for a given pair of terms and no global knowledge base is explicitly constructed. Thus, their method easily produces solutions where global constraints, such as transitivity, are violated.

Another approximation method that violates transitivity constraints is LP relaxation (113) (see Section 2.2.2). In LP relaxation, the constraint $x_{ij} \in \{0,1\}$ is replaced by $0 \leq x_{ij} \leq 1$, transforming the problem from an ILP to a Linear Program (LP), which is polynomial. An LP solver is then applied on the problem, and variables x_{ij} that are assigned a fractional value are rounded to their nearest integer and so many violations of transitivity easily occur. The solution when applying LP relaxation is not a transitive graph, but nevertheless we show for comparison in Section 4.2.5 that our method is much faster.

4.2.2 Forest-reducible graph

The entailment relation, described by 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 4.2.4.

For a directed edge $i \Rightarrow j$ in a directed acyclic graphs (DAG), we term the node i a *child* of node j, and j a *parent* of i[1]. A *directed forest* is a DAG where all nodes have no more than one parent.

The entailment graph in Figure 4.4a (subgraph from the data set described in Section 4.1.3) 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 3 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 (a set of semantically-equivalent predicates, in our graphs) 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 (46), and if $\mathcal{G}$ is transitive then the SCC graph is also transitive. The graph in Figure 4.4b is the SCC graph of the one in Figure 4.4a, but is still not a directed forest since 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 (3). 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 4.4c is the reduced graph of the one in Figure 4.4a and is a directed forest. We say a graph is a *forest-reducible graph (FRG)* if all nodes in its reduced form have no more than one parent.

We now hypothesize that entailment graphs are FRGs. The intuition behind this assumption is that the predicate on the left-hand-side of a uni-directional entailment rule has a more specific meaning than the one on the right-hand-side. For instance, in Figure 4.4a *'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 *'X frequent in Y'*, which are equivalent, while

[1] 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 '⇒'

'X occur in Y' is even more general. Accordingly, the reduced graph in Figure 4.4c is an FRG. We note that this is not always the case: for example, the entailment graph in Figure 4.5 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 (4.12) to an FRG. We term this problem *Max-Trans-Forest*.

To test whether our hypothesis holds empirically we performed the following analysis. We sampled 7 gold standard entailment graphs from the data set described in Section 4.1.3, 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 an entailment graph into an FRG. Further support for the practical validity of this hypothesis is obtained from our experiments in Section 4.2.5. In these experiments we show that exactly solving Max-Trans-Graph and Max-Trans-Forest (with an ILP solver) results in nearly identical performance.

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). Therefore, adding a new type of constraint (Line 4.15) to the ILP given in (4.12) results in a formulation for Max-Trans-Forest:

$$\underset{\mathrm{x}}{\operatorname{argmax}} \sum_{i \neq j} w_{ij} \cdot x_{ij} \tag{4.13}$$

$$\text{s.t. } \forall_{i,j,k \in V} \; x_{ij} + x_{jk} - x_{ik} \leq 1 \tag{4.14}$$

$$\forall_{i,j,k \in V} \; x_{ij} + x_{ik} + (1 - x_{jk}) + (1 - x_{kj}) \leq 3 \tag{4.15}$$

$$\forall_{i,j \in V} \; x_{ij} \in \{0, 1\} \tag{4.16}$$

Next, we prove that Max-Trans-Forest is an NP-hard problem by a polynomial reduction from the X3C problem (66).

4.2.3 FRGs are NP-hard

This section is the result of a discussion with Noga Alon.

4.2.3.1 Problem Definition

We are interested in showing that the following decision problem is NP-hard:

Max-Trans-Forest: Given a set of nodes V, a function $w : V \times V \to \mathbb{R}$ and a real number k, is there an FRG $G = (V, E)$ such that $\sum_{e \in E} w(e) \geq k$.

We show this by two polynomial reductions: first we perform a simple polynomial reduction to Max-Trans-Forest from a variant called Max-Sub-FRG

Max-Sub-FRG: Given a directed graph $G = (V, E)$, a function $w : E \to \mathbb{Z}_+$ and a positive integer z, is there a forest-reducible subgraph $G' = (V', E')$ of G such that $\sum_{e \in E'} w(e) \geq z$.

Then we show a polynomial reduction from the classical Exact Cover by 3-sets (X3C) problem to Max-Sub-FRG.

Exact Cover by 3-sets (X3C) 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.

Since it is known the X3C is NP-hard, the reductions show that Max-Trans-Forest is also NP-hard.

4.2.3.2 Max-Sub-FRG $\leq_p$ Max-Trans-Forest

Given an instance $(G = (V, E), w, z)$ of Max-Sub-FRG we construct the instance (V', w', k) of Max-Trans-Forest:

1. $V' = V$
2. $k = z$
3. $w'(u, v) = w(u, v)$ if $(u, v) \in E$ and $-\infty$ otherwise.

We need show that $(G = (V, E), w, z) \in$ Max-Sub-FRG iff $(V', w', k) \in$ Max-Trans-Forest. This is trivial: if there is a forest-reducible subgraph of G whose sum of edges $\geq z$, then choosing the same edges E' for $G' = (V', E')$ will yield an FRG whose sum of edges $\geq k$. Similarly, any FRG over $G' = (V', E')$ whose sum of edges $\geq k$ can not 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 $\geq z$.

4.2.3.3 X3C $\leq_p$ Max-Sub-FRG

Note: 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 and so from now on z will denote the sum of weights of the edges deleted from the graph.

Given an instance (X, S) of X3C, we construct an instance $(G = (V, E), w, z)$ as follows (An illustration of the construction is given in Figure 4.6). 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. We define $M = 4(n + m)$.

Next we construct the edges E and the weight function $w : E \to \mathbb{Z}_+$.

- For all $1 \leq i \leq m$, an edge (t_i, s_i) of weight 2.
- For all $1 \leq i \leq m$, an edge (a, s_i) of weight 1.
- For all $1 \leq j \leq 3n$, an edge (a, x_j) of weight M.
- For each s_i $(1 \leq i \leq m)$, 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).

Last, we define $z = 4m - 2n$. We need to show that S has an exact 3-cover of X $\Leftrightarrow$ there is a forest-reducible subgraph of G such that the sum of weights deleted is no more than z.

$\Rightarrow$: Assume there is an exact 3-cover of X by S. The forest-reducible 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 3 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 an FRG - there are no connected components of size > 1 and so $SCC(G) = G$, and in the transitive reduction of G there are no violation of transitivity and no node with more than one parent.

$\Leftarrow$: Assume there is no exact 3-cover of X by S, we will show that any forest-reducible 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 $4m - 3n + k + r \geq 4m - 3n + \lceil n - \frac{r}{3} \rceil + r \geq 4m - 2n + r - \lfloor \frac{r}{3} \rfloor > 4m - 2n$. Clearly, the reduction is polynomial and correct, which concludes our proof that Max-Trans-Forest is NP-hard. □

4.2.4 Optimization algorithm

In this section we present *Tree-Node-Fix*, an efficient approximation algorithm for Max-Trans-Forest, as well as *Graph-Node-Fix*, an approximation for Max-Trans-Graph.

4.2.4.1 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 cannot be improved anymore by re-attaching a node.

Re-attaching a node v is performed by removing v from the graph 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 remains fixed. Formally, let $S_{v-in} = \sum_{i \neq v} w_{iv} \cdot x_{iv}$ be the sum of scores over v's incoming edges and $S_{v-out} = \sum_{k \neq v} w_{vk} \cdot x_{vk}$ be the sum of scores over v's outgoing edges. Re-attachment amount to optimizing a linear objective:

$$\underset{\mathcal{X}_v}{\operatorname{argmax}}(S_{v\text{-}in} + S_{v\text{-}out}) \tag{4.17}$$

where the variables $\mathcal{X}_v \subseteq \mathcal{X}$ are indicators for all pairs of nodes involving v. We approximate a solution for (4.12) by iteratively optimizing the simpler objective (4.17). Clearly, at each re-attachment the value of the objective function cannot decrease, since 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 connected component in $\mathcal{G}$. Let $v_c \in c$ be an arbitrary representative node in c. 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:

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

Note that $\{x_{v_c k}, x_{k v_c}\}$ are edge indicators in $\mathcal{G}$ and not $\mathcal{G}_{red}$. There are two possibilities for re-attaching v – either it is inserted into an existing component $c \in V_{red}$ (Figure 4.7a), or it forms a new component. In the latter, there are also two cases: either v is inserted as a child of a component c (Figure 4.7b), or not and then it becomes a root in $\mathcal{G}_{red}$ (Figure 4.7c). We describe the details of these 3 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 (4.17) in this case is

$$s_1(c) \triangleq S_{v\text{-}in}(c) + S_{v\text{-}out}(c) \tag{4.18}$$

Case 2: 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 can not become a child of v, since 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 4.7b), since for any other descendant of c, v would become a second parent, and $\mathcal{G}_{red}$ will no longer be a directed forest (Figure 4.7b'). Thus, this case requires adding in $\mathcal{G}$ edges from v to all nodes in c and their ancestors, and also 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 (4.17) in this case is:

$$s_2(c) \triangleq S_{v\text{-}out}(c) + \sum_{d \in child(c)} \max(0, S_{v\text{-}in}(d)) \tag{4.19}$$

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

Case 3: Inserting v as a new root in $\mathcal{G}_{red}$. Similar to case 2, 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 (4.17) of re-attaching v is:

$$s_3 \triangleq \sum_{r} \max(0, S_{v\text{-}in}(r)) \tag{4.20}$$

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

Algorithm 4 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. 4.21)
3: for all $c \in V_{red}$ in pre-order compute $S_{v\text{-}out}(c)$ (Eq. 4.22)
4: case 1: $s_1 = \max_{c \in V_{red}} s_1(c)$ (Eq. 4.18)
5: case 2: $s_2 = \max_{c \in V_{red}} s_2(c)$ (Eq. 4.19)
6: case 3: compute s_3 (Eq. 4.20)
7: re-attach v according to $\max(s_1, s_2, s_3)$.

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

$$S_{v\text{-}in}(c) = \sum_{i \in c} w_{iv} + \sum_{d \in child(c)} S_{v\text{-}in}(d), \ c \in V_{red} \tag{4.21}$$

$$S_{v\text{-}out}(c) = \sum_{i \in c} w_{vi} + S_{v\text{-}out}(p), \ c \in V_{red} \tag{4.22}$$

where p is the parent of c in $\mathcal{G}_{red}$. These recursive definitions allow to compute 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 4. Computing an SCC graph is linear (46) 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 3 are trivially linear and in case 2 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.

Since 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. In Section 4.2.5

we ran TNF until convergence and the maximal number of iterations over graph nodes was 8.

4.2.4.2 Graph-node-fix

Next, we show Graph-Node-Fix (GNF), a similar approximation that employs the same re-attachment strategy but does not assume the graph is an FRG. Thus, re-attachment of a node v is done with an ILP solver. Nevertheless, the ILP in GNF is simpler than (4.12), since we consider only candidate edges involving v. Figure 4.8 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 (4.23) below, and the middle and right depict two violations when the edge $(i,k) \in E$, expressed by the constraints in (4.24). Thus, the ILP is formulated by adding the following constraints to the objective function (4.17):

$$\forall_{i,k \in V \setminus \{v\}} \text{ if } (i,k) \notin E,\ x_{iv} + x_{vk} \leq 1 \tag{4.23}$$

$$\text{if } (i,k) \in E,\ x_{vi} \leq x_{vk},\ x_{kv} \leq x_{iv} \tag{4.24}$$

$$x_{iv}, x_{vk} \in \{0,1\} \tag{4.25}$$

Complexity is exponential due to the ILP solver; however, the ILP size is reduced by an order of magnitude to $O(|V|)$ variables and $O(|V|^2)$ constraints.

4.2.5 Experimental evaluation

In this section we empirically demonstrate that TNF is more efficient than other baselines and its output quality is close to that given by the optimal solution.

4.2.5.1 Experimental setting

We use the same experimental setting described in Section 4.1.3. However, we transform the three single-type entailment graphs into two-types entailment graphs by applying the procedure described in Section 4.2.1.

We remind that we trained a local entailment classifier that provides for every pair of predicates i, j in every graph a *local score* s_{ij}, where a positive s_{ij} indicates that the classifier believes $i \Rightarrow j$. The weighting function w is defined as $w_{ij} = s_{ij} - \lambda$, where

λ is a single parameter controlling graph sparseness: as λ increases, w_{ij} decreases and becomes negative for more pairs of predicates, rendering the graph more sparse. In addition, we mention again that the weighting function was modified to represent both positive and negative local constraints (Section 4.2.1).

We implemented the following algorithms for learning graph edges, where in all of them the graph is first decomposed into components as described i Section 4.1.

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

Exact-graph The method described in Section 4.1.

Exact-forest Solving Max-Trans-Forest exactly by applying an ILP solver (see Lines 4.13-4.16).

LP-relax Solving Max-Trans-Graph approximately by applying LP-relaxation on each graph component. We apply the LP solver within the same cutting-plane procedure (incremental ILP) as Exact-graph to allow for a direct comparison. This also keeps memory consumption manageable, as otherwise all $|V|^3$ constraints must be explicitly encoded into the LP. As mentioned, our goal is to present a method for learning transitive graphs, while LP-relax produces solutions that violate transitivity. However, we run it on our data set to obtain empirical results, and to compare run-times against TNF.

Graph-Node-Fix (GNF) Initialization of each component is performed in the following way: if the graph is very sparse, i.e. $\lambda \geq C$ 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, i.e., $\lambda = C$.

Tree-Node-Fix (TNF) Initialization is done 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 note that the Gurobi optimization package[1] was used as our ILP solver in all experiments. In addition, the experiments were run on a multi-core 2.5GHz server with 32GB of virtual memory.

[1] `www.gurobi.com`

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 run-times.

4.2.5.2 Results

We first focus on run-times and show that TNF is efficient and has potential to scale to large data sets.

Figure 4.9 compares run-times of Exact-graph, GNF, TNF, and LP-relax as $-\lambda$ increases and the graph becomes denser. Note that the y-axis is in logarithmic scale. Clearly, Exact-graph is extremely slow and run-time increases quickly. For $\lambda = 0.3$ run-time was already 12 hours and we were unable to obtain results for $\lambda < 0.3$, while 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, run-time for GNF and TNF increases much more slowly than for Exact-graph.

Run-time of LP-relax is also bad comparing to TNF and GNF. Run-time increases more slowly than Exact-graph, but still very fast comparing to 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 since it is an approximation of Exact-graph we omit if from the figure for clarity.

As for the quality of learned graphs, Figure 4.10 provides a precision-recall curve for Exact-graph, TNF and No-trans (GNF and LP-relax are omitted from the figure and described below to improve readability). We observe that both Exact-graph and TNF substantially outperform No-trans and that TNF's garph 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 to Exact-graph, TNF reduces AUC by merely a point and the maximal F_1 score by 2 points only.

GNF results are almost identical to those of TNF (maximal F_1=0.41, AUC: 0.31), and in fact for all λ configurations TNF outperforms GNF by no more than one F_1 point. 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. Last, we note that the results of Exact-forest are almost identical to Exact-graph (maximal F_1: 0.43), illustrating that assuming that entailment graphs are FRGs (Section 4.2.2) is reasonable in this data set.

To conclude, TNF learns transitive entailment graphs of good quality much faster than Exact-graph. Our experiment in this section utilized the data set of Schoenmackers et al., but we expect TNF to scale to much larger data sets, where other baselines would be impractical. Such a data set if presented and investigated empirically in Chapter 5.

4.2.6 Conclusion

In this section we have presented two main contributions. The first was a novel modeling assumption that entailment graphs are very similar to FRGs, which was analyzed and validated empirically. The second contribution is an efficient polynomial approximation algorithm for learning entailment rules, which is based on this assumption. We demonstrated empirically that our method is by orders of magnitude faster than the state-of-the-art exact algorithm, but still produces an output that is almost as good as the optimal solution.

Overall in this chapter we have presented methods that allow to scale the model presented in Chapter 3 to large graphs. In the next chapter we apply these methods on a data set containing $10^5 - 10^6$ predicative templates. This is a domain-general data set and we experiment both with training a local entailment classifier over a rich set of features and with exploiting transitivity to improve over the local classifier. We work with untyped predicates, which raises the problem of ambiguity, and we investigate how this problems interferes with our structural assumptions that the graph is transitive and forest-reducible. Most importantly, we release a state-of-the-art knowledge resource containing millions of predicative entailment rules for the benefit of the scientific community.

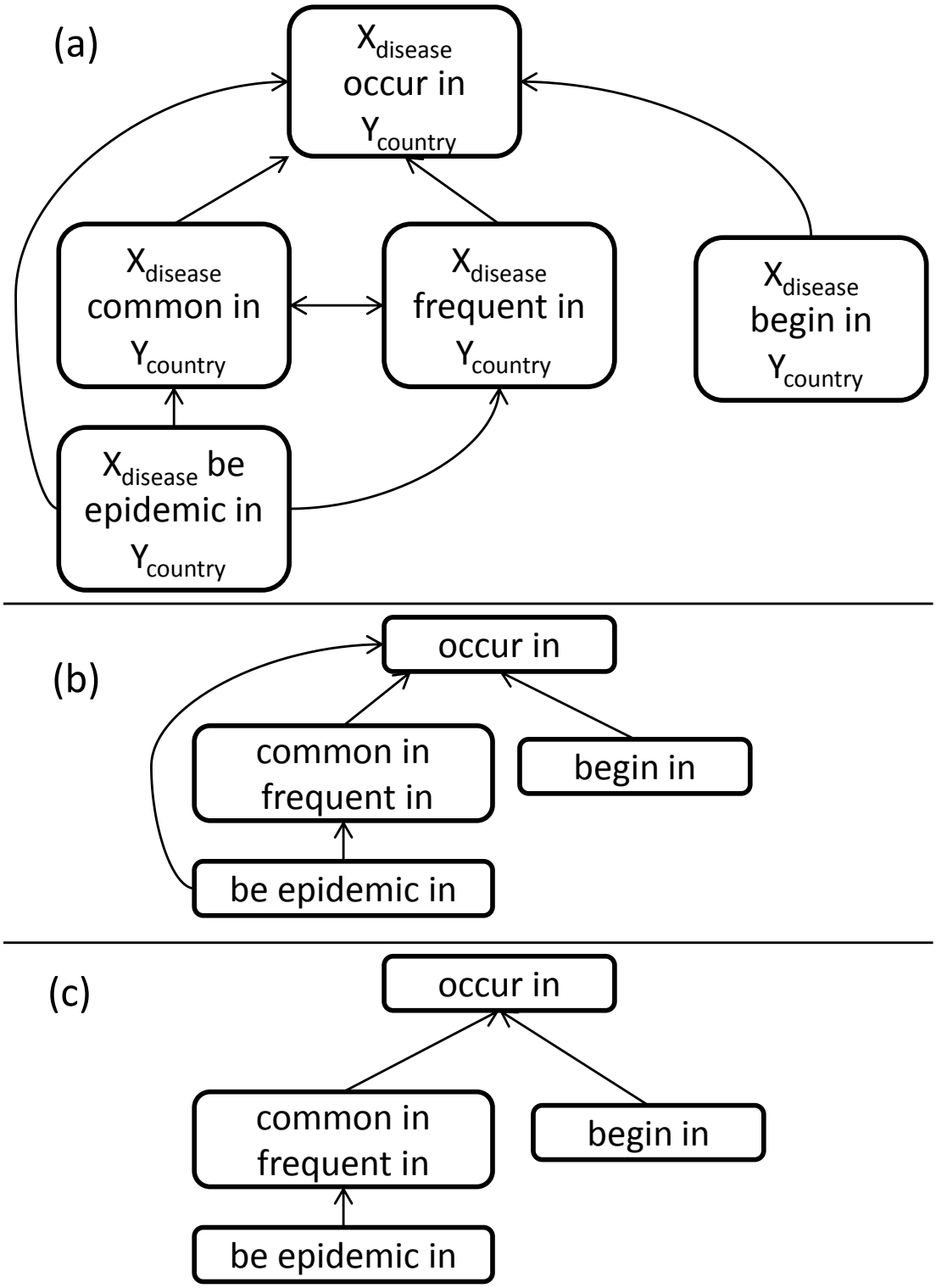

Figure 4.4: A fragment of an entailment graph (a), its SCC graph (b) and its reduced graph (c). Nodes are predicates with typed variables, which are omitted in (b) and (c) for compactness.

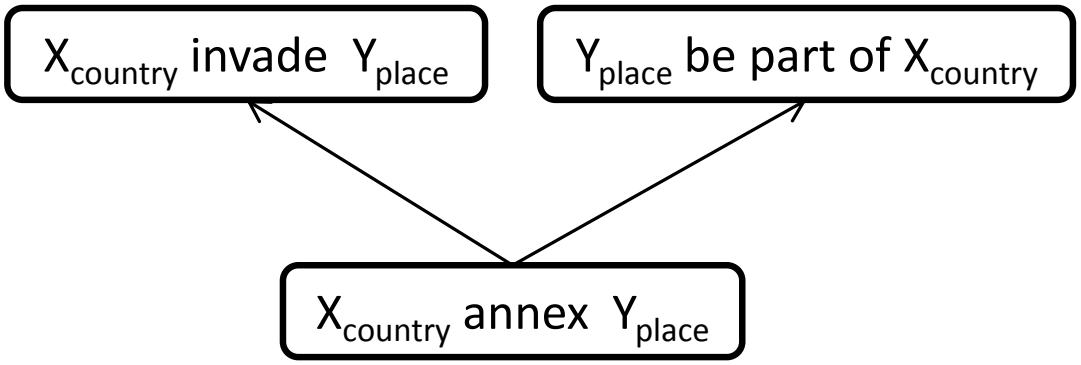

Figure 4.5: A fragment of an entailment graph that is not an FRG.

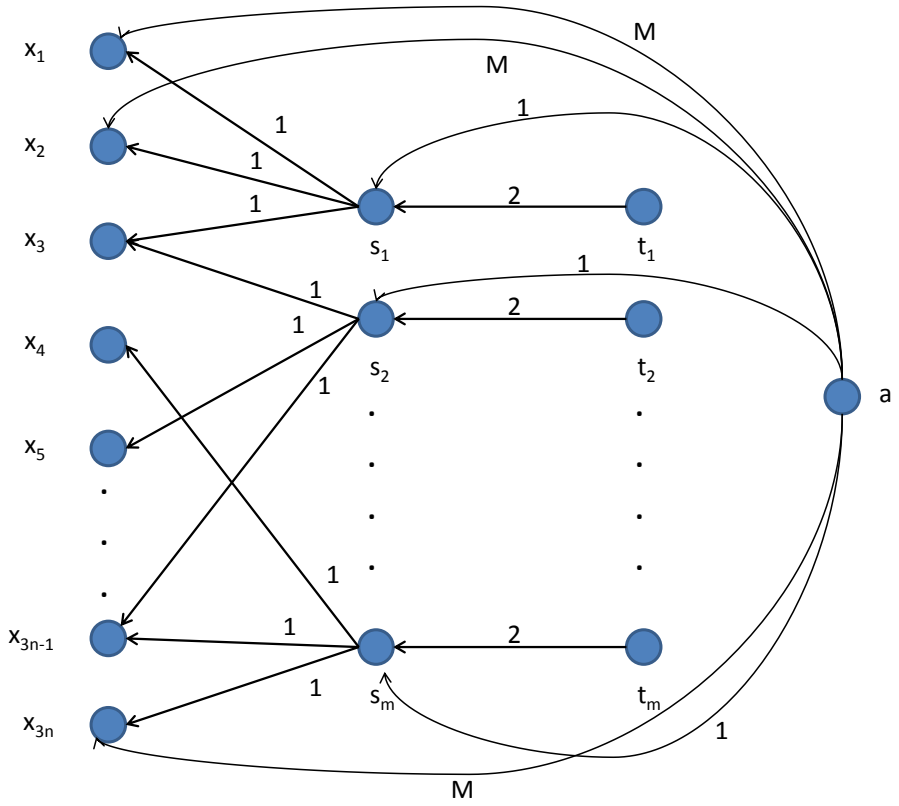

Figure 4.6: The graph constructed given an input X of size $3n$ and S of size m. Each $s \in S$ is a set of size 3. In this example $s_1 = \{x_1, x_2, x_3\}, s_2 = \{x_3, x_5, x_{3n-1}\}, s_m = \{x_4, x_{3n-1}, x_{3n}\}$.

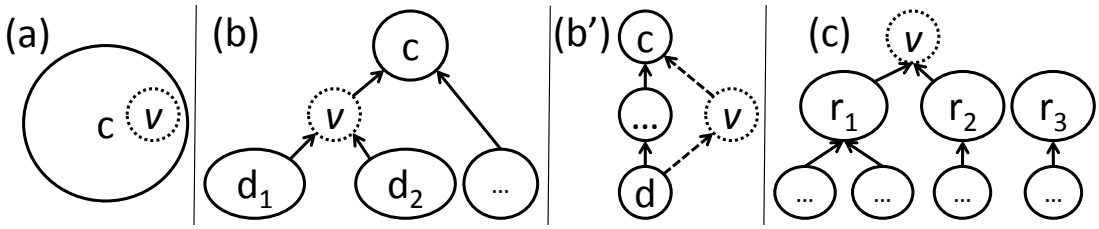

Figure 4.7: (a) Inserting v into a component $c \in V_{red}$. (b) Inserting v as a child of c and a parent of a subset of c's children in $\mathcal{G}_{red}$. (b') A node d that is a descendant but not a child of c can not choose v as a parent, as v becomes its second parent. (c) Inserting v as a new root.

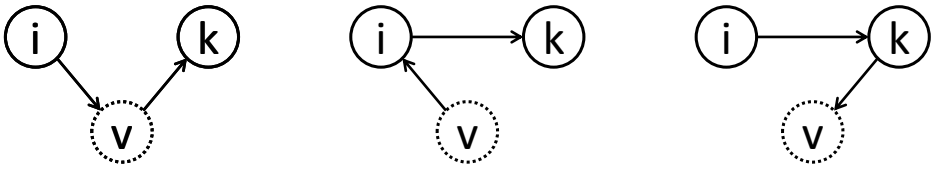

Figure 4.8: Three types of transitivity constraint violations.

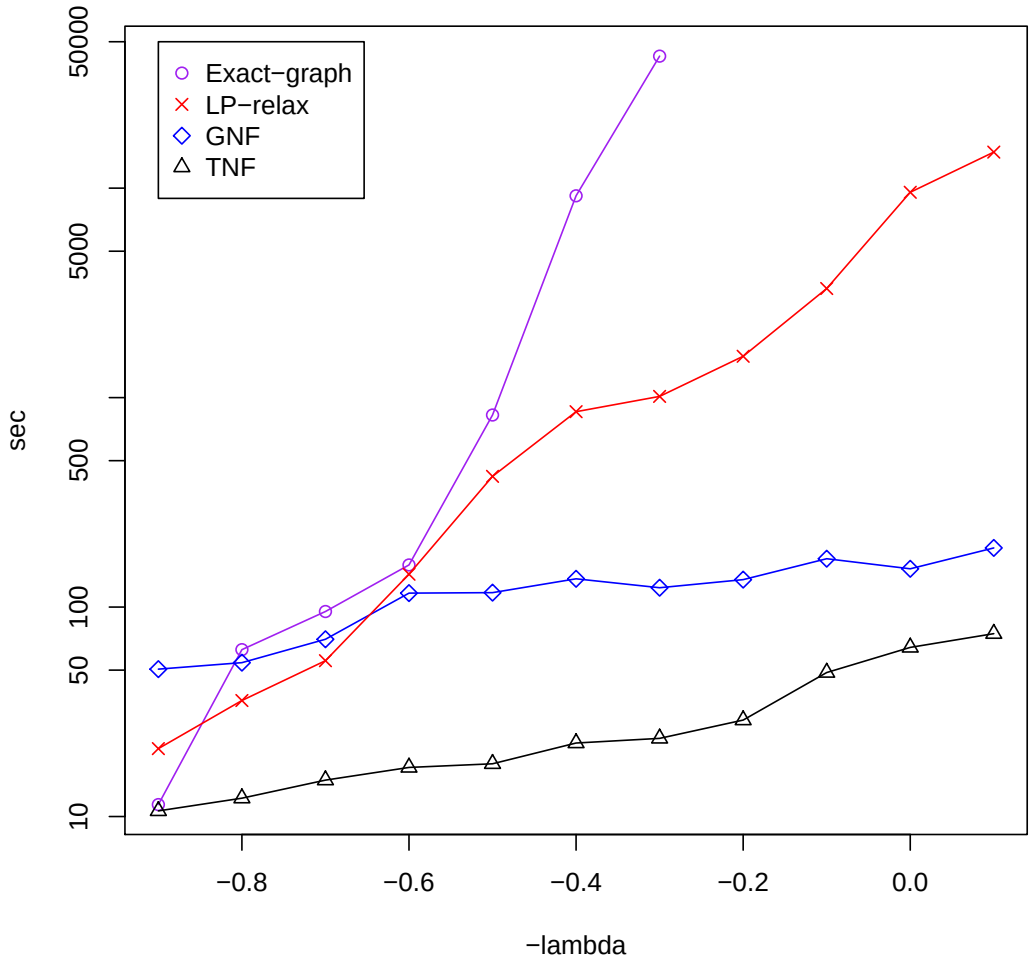

Figure 4.9: Run-time in seconds for various $-\lambda$ values.

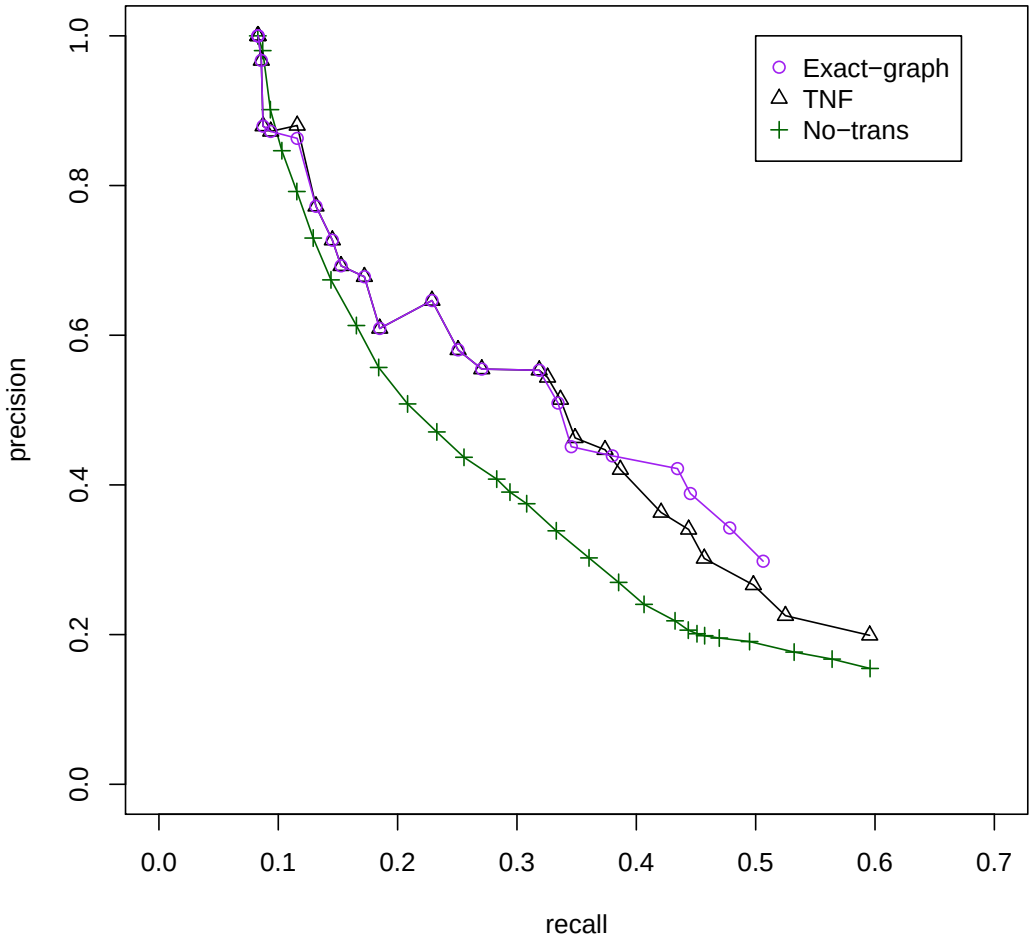

Figure 4.10: Precision (y-axis) vs. recall (x-axis) curve. Maximal F_1 on the curve is .43 for Exact-graph, .41 for TNF, and .34 for No-trans. AUC in the recall range 0-0.5 is .32 for Exact-graph, .31 for TNF, and .26 for No-trans.

5

Large-scale Entailment Rules Resource

Despite the plethora of recent works on learning predicative entailment rules, many works did not release an entailment rule resource. Therefore, most semantic applications still make use of the knowledge-base learned by the *DIRT* algorithm (104) more than a decade ago. In this Chapter we describe the creation of a resource[1] that contains millions of predicative entailment rules, and demonstrate that it outperforms DIRT and can also be combined with it.

The main source of information for our resource is *REVERB*, a recently created huge domain-general data set of tuples that were extracted from the web ($\sim 10^9$ tuples), where each tuple contains a predicate and its pair of arguments (*'pred(arg$_1$,arg$_2$)'*). The resource contains three independently learned knowledge-bases. The first, containing millions of predicative entailment rules, was learned over a set of more than 100,000 predicates by a *local entailment classifier* (Section 3.2.1) trained over a rich set of features. The other two knowledge-bases were learned using *global learning* algorithms over a graph of 10,000 predicates, which is much larger than the graphs presented in Chapter 4. We show that global learning can still improve precision compared to local learning methods, even in a domain-general setting.

In Section 5.1 we describe the REVERB data set. Then, we specify the steps necessary for constructing the resource (Section 5.2), including both the preprocess-

[1]The resource can be freely downloaded from the downloads page of the NLP lab at Bar-Ilan University: `http://u.cs.biu.ac.il/~nlp/downloads/index.htm`

ing performed (Section 5.2.1), and the local and global learning methods applied to the data (Section 5.2.2). We also describe several expansions to the global learning algorithms presented in Chapter 4, which are of practical importance when working over large graphs. As the resource is learned over a shallow syntactic representation, we also specify how we generated a "deep-syntax" version that can be utilized by applications working over dependency trees (Section 5.2.3). In Section 5.3 we evaluate the resource: first we present a novel method for evaluating entailment rules using a gold standard that was generated by many non-expert annotators in a crowdsourcing framework (Section 5.3.1), and then we present an evaluation over several RTE data sets (Section 5.3.2). Last, we perform an analysis of the results – as mentioned, we have presented methods for learning large graphs, but did not suggest a solution for the problem of predicate ambiguity when working over domain-general data. We investigate the specifics of how ambiguity interferes with our assumptions about the structure of entailment graphs (Section 5.4).

5.1 The REVERB data set

Arg$_1$	**Predicate**	**Arg$_2$**
Cabbage	also contains significant amounts of	Vitamin A
Our leaders	continue to make	contacts
Anthony	is a professor of	law
Boston market	is a registered trademark of	Mcdonald's corporation
Niacin	may also give	false-positive reactions
The question	applies to	everyone

Table 5.1: Examples of REVERB extractions.

The REVERB open information extraction system (see Section 2.4) was presented recently by Fader et al. (57). This system extracts tuples of the form *'(arg$_1$,predicate,arg$_2$)'* (see Table 5.1), given a POS-tagged and chunked sentence, where the predicate always contains a *verb*. Fader et al. noticed that prior state-of-the-art open information extraction systems tend to extract many *incoherent* as well as *uninformative* tuples (see examples in Tables 5.2 and 5.3). Therefore, they introduced two simple syntactic and lexical constraints on the extracted tuples, which resulted in a dramatic increase in

Sentence	Incoherent predicate
The guide *contains* dead links and *omits* sites.	contains omits
The Mark 14 *was central* to the *torpedo* scandal of the fleet.	was central torpedo
They *recalled* that Nungesser *began* his career as a precinct leader.	recall began

Table 5.2: Examples taken from Fader et al. (57) of *incoherent* predicates extracted by state-of-the-art open information extraction systems.

Uninformative predicate	Informative predicate
is	is an album by, is the author of, is a city in
has	has a population of, has a Ph.D. in, has a cameo in
made	made a deal with, made a promise to
took	took place in, took control over, took advantage of
gave	gave birth to, gave a talk at, gave new meaning to
got	got tickets to, got a deal on, got funding from

Table 5.3: Examples taken from Fader et al. (57) of *uninformative* predicates extracted by state-of-the-art open information extraction systems, and their informative original counterparts.

extractor performance. Moreover, they manually annotated 1,000 extractions as either correct or incorrect, and then used them as a training set for a supervised classifier that assigns each extraction a confidence value of 0 to 1. The set of features they utilized is easily computable from the sentence and extraction and includes features such as whether the extraction covers the entire sentence, the length of the sentence, whether there are NPs to the right of the extraction, etc.

Fader et al. ran their extractor on the large ClueWeb09 data set, which contains more than 1 billion web pages, and graciously provided us with an output of 2 billion extraction tuples. Each provided extraction contains the following fields:

1. Arg_1
2. Predicate

3. Arg_2
4. Normalized arg_1
5. Normalized predicate
6. Normalized arg_2
7. POS-tags of arg_1
8. POS-tags of predicate
9. POS-tags of arg_2
10. confidence value

Normalization was performed word-by-word, where different normalization procedures were employed for arguments and predicates. We describe an extension to their normalization procedure in Section 5.2.1. The tag list employed was the Penn Treebank POS-tag list[1].

Since the REVERB algorithm tries to avoid predicates that are uninformative, one of the characteristics of this data set is that predicates are relatively "semantically-rich", that is, they often carry a rather specific semantic meaning. An example for that is a predicate such as *'is a registered trademark of'*, shown in Table 5.1. Also note that some of the predicates in the table can be further normalized, for example we would like to omit the words *'may also'* from the predicate *'may also give'*. This data set is the input to our algorithm for learning predicative entailment rules, and allows us to employ distributional similarity methods since predicates can be compared according to the arguments they co-occur with.

For more information on the REVERB algorithm, code and data set, see the REVERB webpage[2].

[1] `http://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html`
[2] `http://reverb.cs.washington.edu/`

5.2 Resource Construction

In this section we present in detail the process of creating our novel resource of predicative entailment rules. We start with describing the preprocessing steps performed over the input REVERB data set.

5.2.1 Preprocessing

There are several goals to preprocessing: (a) Omitting noisy extractions that are likely to be erroneous. (b) Reducing sparseness by normalizing predicates and arguments that contain immaterial content. (c) Defining a reasonable set of predicates for which we would like to learn entailment rules. The outline of the preprocessing procedure is given in Algorithm 5.

Algorithm 5 Outline of preprocessing steps.

Input: A data set of REVERB extractions

1: Remove extractions with a low confidence value.
2: Normalize predicates.
3: Normalize arguments.
4: Remove extractions containing predicates that do not occur with many arguments.

As mentioned, each extraction is accompanied by a confidence value. To reduce the amount of noisy extractions we simply discard any extraction for which the confidence value is smaller than 0.5. This reduces the size of the data set by half, leaving about 1 billion extractions.

Next, we normalize the predicates. As we saw in Table 5.1, predicates may contain unnecessary content such as modals and adverbs. This fragments the statistics in the data, and so it is preferable to normalize the predicates so long that this does not compromise their semantics. We employ a rule-based approach which takes as input the predicate, its normalization provided by REVERB, and its POS-tags, and using a few dozens of rules outputs a normalized form that will be subsequently used in our learning algorithm. During normalization we delete extractions containing invalid predicates (*e.g.,* predicates whose first character is a number or some other invalid character), omit from the predicates adverbs, modals and quasi-modals (*'ought to'*, *'be able to'*, etc.), and also canonize various grammatical constructions such as present

perfect (*'have eaten'*), passive (*'was raised'*), and more. Our complete normalization procedure is now part of the public REVERB project and can thus be reviewd in detail and downloaded from the REVERB repository[1].

We also normalize the arguments in the extractions. We replace the pronouns *'I'*, *'we'*, *'you'*, *'he'*, *'she'*, and *'they'* by the single token 'PRONOUN'. We also replace all occurrences of *'a'* and *'the'* by *'DET'*. Last, we run the BIU number normalizer[2] and replace all numbers larger than 1 by the token *'NUM'*. After normalizing the predicates and arguments we are left with a total of 960 million extractions of which 291 million are distinct.

Last, we focus on predicates that occur frequently and with many arguments. The intuition is that interesting predicates should be frequent in the data set, but more than that should appear with quite many different arguments. A predicate that is very frequent in the data set but always appears with the same arg_1 or arg_2 might be the result of a faulty extraction that is very common on the web. However, if a predicate occurs with many different arguments then it is more likely that it is indeed correct and meaningful. Based on a preliminary analysis, we decided to keep extractions only for predicates that occur with at least 25 distinct arguments of type arg_1, 25 distinct arguments of type arg_2, and 225 distinct pairs (arg_1, arg_2). This results in the final preprocessed data set, containing 878 million extractions of which 217 million are distinct.

5.2.2 Learning

We now provide the details of the learning process. First, we describe the construction of a *local* resource over a large set of more than 100,000 predicates, and then the construction of a *global* resource over an entailment graph of 10,000 predicates.

5.2.2.1 Local resource

An illustration of the overall flow for the construction of the local resource (including preprocessing) is given in Figure 5.1. Next, we describe this flow step by step.

[1] `https://github.com/knowitall/reverb-core/blob/master/core/src/main/java/edu/washington/cs/knowitall/normalization/RelationString.java`

[2] `http://u.cs.biu.ac.il/~nlp/downloads/normalizer.html`

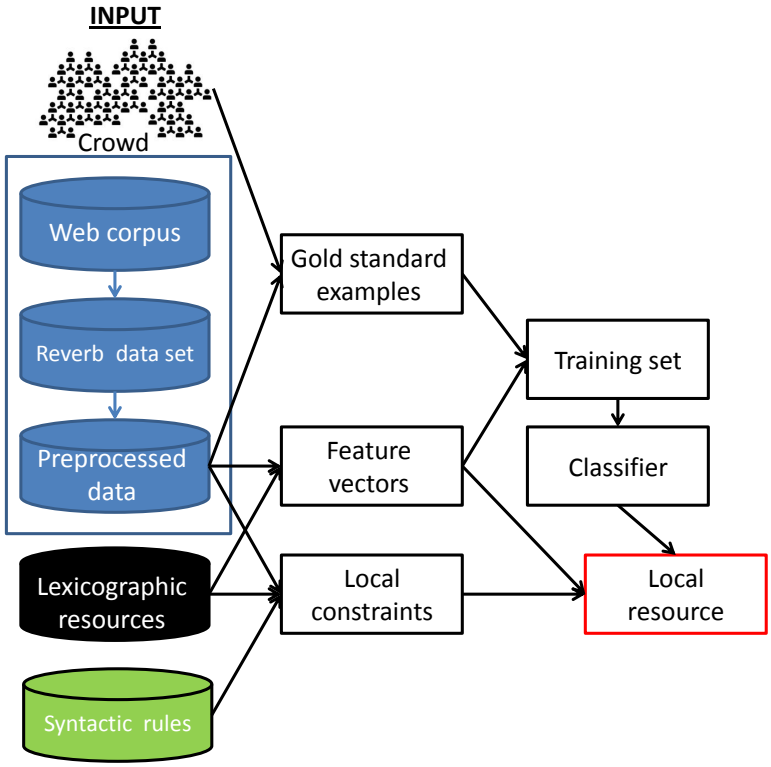

Figure 5.1: A diagram illustrating the process of constructing the local resource.

Feature vector construction The preprocessed REVERB data set contains 103,315 distinct predicates, which as previously mentioned appear with a large number of distinct arguments and pairs of arguments. Every pair of predicates is represented by a feature vector, but since the number of pairs of predicates is higher than 10^{10} this is performed as follows. First, we split every predicate into two predicates, as described in Section 4.2.1. For example, the predicate *'defeat'* is split into *'X defeat Y'* and *'Y defeat X'*. This is necessary for learning rules where the order of arguments is reversed, such as '*X defeat Y* $\Rightarrow$ *Y lose to X*'. Recall that this split means that each rule is learned twice, for instance the previous rule is equivalent to the rule '*Y defeat X* $\Rightarrow$ *X lose to Y*'. Thus, the total number of predicates is doubled to 206,630.

Next, we compute for each predicate p_1 the top-100 predicates p_2 that obtained the highest similarity score (p_1, p_2) according to some distributional similarity measure. This is repeated six times with six different distributional similarity measures. Naturally, the top-100 lists of different similarity measures often overlap and so the number of (p_1, p_2) pairs for each predicate p_1 is bounded between 100 and 600. Practically, the number of pairs (p_1, p_2) for each predicate p_1 is usually about 200-300. The first three measures employ the approach suggested by DIRT (104) (see Section 2.2.1.1): each predicate is represented by two feature vectors, and the similarity of a pair of predicates is computed by comparing the arg_1 vectors, the arg_2 vectors, and then calculating their geometric average. The last three measures employ the approach suggested by TEASE (175) and Schoenmackers et al. (153), where each predicate is represented by a single vector of pairs of arguments. The difference between the three measures (both the first and the last) is in the similarity function used: *Lin* (104), *BInc* (172), or *Cover* (183) (see Section 2.2.1.1).

The size of the union of candidate rules (p_1, p_2) described above across all predicates p_1 is about 47 million rules. We treat each distributional similarity measure as a feature and hence each one of the 47 million pairs of predicates has at least one non-zero feature. The features for all other pairs of predicates are all zero.

Next, we enrich each one of the 47 million feature vectors with lexicographic features computed from WordNet (59), VerbOcean (39) and Catvar (72) (see Chapter 2), and also with string-similarity features. This set of 21 features is inspired by the features presented in Section 3.5.1, and Table 5.4 provides their exact details. A feature is computed for a pair of propositional predicates (p_1, p_2) where each propositional 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.

The four string-similarity features *'remove word'*, *'add word'*, *'remove adj.'*, and *'add adj.'* are generated since the predicates of REVERB are often multi-word expressions such as *'have big plan for'*. We expect that '*have big plan for* $\Rightarrow$ *have plan for*', but '*have plan for* $\not\Rightarrow$ *have big plan for*'. Therefore, the features *'remove word'* and *'remove adj.'* indicate that the RHS is more general than the LHS and are expected to support entailment, while the features *'add word'* and *'add adj.'* indicate that the RHS is more specific than the LHS and are expected to be negative entailment signals. Note that if the value of the binary feature *'add adj.'* is 1 then the value of the binary feature *'add word'* must be 1, and similarly if the value of the binary feature *'remove adj.'* is 1 then the value of the binary feature *'remove word'* must be 1. In addition, the feature *'Edit'* represents normalized edit-distance as explained in Section 3.5.1, except that if $rev_1 \neq rev_2$, the feature value is 1. Overall, each pair of predicates is represented by 27 features.

Training a local classifier After obtaining a feature representation for every pair of predicates, we turn to training a local entailment classifier. We take advantage of crowdsourcing services to generate a large gold standard, as will be described in detail in Section 5.3.1, where each gold standard example is an annotated pair of predicative templates (*e.g.,* '*X unable to pay Y* $\Rightarrow$ *X owe Y*' and '*X own Y* $\not\Rightarrow$ *Y be sold to X*'). We use half of the gold standard as a training set, yielding 1,224 positive examples and 2,060 negative examples. We utilize a Gaussian kernel SVM classifier that optimizes F_1 over the training set (SVM-perf implementation (89)), 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 47 million pairs of predicates for which there was at least one non-zero feature. Additionally, we assume that the feature vector for all other pairs of predicates is all zeros and compute their classification score s_{zero} by applying the classifier over an all-zero feature vector.

Local constraints As in previous chapters, we automatically generate local constraints for pairs of predicates for which we know with high certainty whether the first

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.
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 $rev_1 = rev_2$, then normalized edit-distance between $pred_1$ and $pred_2$ (see Section 3.5.1), otherwise 1.
Reverse	binary	String	$rev_1 = rev_2$

Table 5.4: Definition of features added to the distributional similarity features. We note 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. In WordNet we take advantage of the annotated relations *synonymy*, *hyponymy*, *hypernymy*, *co-hyponymy*, and *verb-entailment*. In VerbOcean we use the relation *stronger-than*.

entails the second or not. We take advantage of both lexicographic resources as well as syntactic knowledge to create these constraints.

We define and compute constraints over pairs of predicative templates (p_1, p_2), where a predicative template p is once again a pair $(pred, rev)$. We begin with negative constraints, that is, pairs of predicates for which we believe '$p_1 \nRightarrow p_2$' (Examples for each type of constraint are given in Table 5.5). Note that some of the constraints are symmetric (*i.e.,* if '$p_1 \nRightarrow p_2$', then '$p_2 \nRightarrow p_1$') and some are directional:

- *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.

- *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.

- *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.

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

- *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 *"n't"*) 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 $pred_2$.

- *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.

Name	⇒/⇔	**Example**	#
cousin	⇔	'*X beat Y* ⇏ *X fix Y*'	5,673,388
indirect hypernym	⇒	'*X give all kind of* ⇏ *X sell all kind of Y*'	284,824
antonym	⇔	'*X announce death of* ⇏ *X announce birth of Y*'	20,832
negative implication	⇒	'*X want to analyze Y* ⇏ *X analyze Y*'	2,770
negation	⇔	'*X do no harm to Y* ⇏ *X do harm to Y*'	2,188
transitive opposite	⇔	'*X abandon Y* ⇏ *Y abandon X*'	5,832

Table 5.5: Examples for negative local constraints. The column '⇒/⇔' indicates whether the constraint is directional or symmetric. The column '#' specifies the number of constraints generated over the predicate set.

Name	⇒/⇔	**Example**	#
determiner	⇔	'*X be the answer to Y* ⇒ *X be answer to Y*'	12,620
positive implication	⇒	'*X start to doubt Y* ⇒ *X doubt Y*'	3,254
passive-active	⇔	'*X command Y* ⇒ *Y be commanded by X*'	7,388

Table 5.6: Examples for positive local constraints. The column '⇒/⇔' indicates whether the constraint is directional or symmetric. The column '#' specifies the number of constraints generated over the predicate set.

Next, we define the following positive constraints (Examples provided in Table 5.6). Again some of the constraints are symmetric and some directional.

- *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 $pred_2$.
- *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 $pred_1$.
- *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.

Last, some of the negative constraints generated automatically are not safe enough. This happens when the pair of predicates differ by one word, which has a very loose semantic meaning. Therefore, we omit negative constraints (p_1, p_2) when $rev_1 = rev_2$ and $pred_1$ is identical to $pred_2$ except for a single pair of words (w_1, w_2), and either w_1 or w_2 belong to a small set of verbs with a loose semantic meaning (*"get"*, *"make'*, *"seem"*, *"have"*, *"be"* or *"do"*). After applying this filter, the total number of negative constraints is 5,806,802 and the total number of positive constraints is 23,262.

Once we compute all local constraints, we can combine these constraints with the scores s_{ij} we obtained earlier. For any positive local constraint (i, j) we modify the score $s_{ij} = \infty$ and conversely for any negative local constraint (i, j) we modify the score $s_{ij} = -\infty$. This completes the creation of the local resource as illustrated by Figure 5.1. A user of the resource needs to define a threshold λ and use only rules that cross this threshold. For example, a default threshold of $\lambda = 0$, which corresponds to all candidate rules that were classified as *'entailing'* by the local classifier, results in 15 million predicative entailment rules.

5.2.2.2 Global resource

Once we obtain the local scores s_{ij} for any pair of predicative templates, we can use the methods presented in Chapters 3 and 4 to learn a global resource. Applying global methods over a set of more than 100,000 predicates is still computationally quite expensive, and therefore we restrict the predicate set to the 10,000 predicates that appear with the highest number of pairs of arguments (resulting in 20,000 graph nodes since as described above each predicate $'p'$ is split into two predicative templates *'X p Y'* and *'Y p X'*). Our goal is to solve the global optimization problem presented in Equation 4.12 (Section 4.2.1), where again the weight is defined by the local score and a sparseness prior: $w_{ij} = s_{ij} - \lambda$. We remind the reader that our local and global algorithms optimize exactly the same objective function from Equation 4.12, which depends on λ, except that the local algorithm omits the transitivity constraints. Therefore, we refer to the notation λ in both algorithms, although in the local setting it is often interpreted as a "decision threshold" for determining whether a rule is correct or not, while in the global setting λ is interpreted as an edge cost or a sparseness prior.

Many of the methods presented so far are not practical when working over such a large set of predicates. Applying an ILP solver becomes intractable, unless the graph

decomposes into quite small components. However, we empirically demonstrate in Section 5.3.1 that when working over a large set of untyped ambiguous predicates, a "giant" component is formed quite quickly. Hence, computing the exact solution and even applying Graph-node-fix is not feasible. Even Snow et al.'s method (165), which is polynomial, considers at each iteration all possible candidate edges. Since the number of candidate edges at each iteration is $O(|V|^2)$ and $|V| = 20,000$, a straightforward implementation is also far too slow.

The Tree-node-fix algorithm (Section 4.2.4.1) enables scaling to graphs of 20,000 nodes. Since applying decomposition is very efficient, we first decompose the graph into components and then apply Tree-node-fix over each component. However, recall that in Tree-node-fix each component must be initialized in some valid way, that is, an initial forest-reducible graph must be constructed for every component. A trivial solution is to initialize every component with an empty graph (which is of course an FRG), but this leads to solutions of low quality, especially when attempting to reach high recall values. In the experiment described in Section 4.2.5 we initialized each component by applying an ILP solver in a sparse configuration where it was still tractable after graph decomposition. However, in a very large graph this is not a viable solution since applying an ILP solver is intractable even for low recall values. Therefore, we now present two initialization procedures that are efficient variants of Snow et al.'s method.

Initialization Our goal is to design a quick initialization procedure that provides a good approximation and allows to subsequently apply Tree-node-fix, which will improve over the initial approximation. As explained, Snow et al. suggested a greedy optimization that searches for the best possible edge to be added to the graph at each iteration, but this leads to iterations that are too slow. Algorithm 6 suggests a method where we first determine an ordering for the edges, and then scan them one by one and decide for each one whether to insert it into the graph or not.

In this algorithm, 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 (Line 2). This number of pairs of predicates depends on λ but is manageable (in the order of millions or a few tens of millions). We perform this computationally expensive step only once. Then, we go over the predicate pairs one-by-one and compute for each such candidate edge (i, j) its *transitive closure* T_{ij}. The transitive closure describes node pairs that must be edges

Algorithm 6 The high-to-low algorithm

Input: A predicate set V and a weighting function $w : V \times V \rightarrow \mathbb{R}$

Output: A set of directed edges E

1: $E = \phi$.
2: $S \leftarrow$ a list of pairs of predicates (i, j) for which $w_{ij} > 0$ sorted by w.
3: **repeat**
4: **for** $(i, j) \in S$ according to sorting order **do**
5: $T_{ij} \leftarrow$ transitive closure of (i, j)
6: $C \leftarrow \sum_{(k,l) \in Tij \setminus E} w_{kl}$
7: **if** $C > 0$ **then**
8: $E \leftarrow E \cup T_{ij}$
9: $S \leftarrow S \setminus T_{ij}$
10: **end if**
11: **end for**
12: **until** objective function can not be improved by inserting T_{ij} for any $(i, j) \in S$

in the graph in case we insert (i, j) as an edge, due to transitivity. More formally, let I denote the node set containing i and all of its predecessors, *i.e.,* all nodes l such that (l, i) is an edge in the graph. Let J denote the node set containing j and all of its successors, *i.e.,* all nodes m such that (j, m) is an edge in the graph. Then the transitive closure T_{ij} equals to the cartesian product $I \times J$ (Line 5).

Next, we compute the change in the value of the objective function if we were to insert T_{ij} into the graph (Line 6). 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). Note that the way we score a candidate edge (i, j) is identical to the one proposed by Snow et al. (165) (Section 2.2.2), nevertheless we describe it here for the sake of readability. We keep going over the candidate edges from highest score to lowest until the objective function can not be improved anymore by inserting a candidate edge. Practically, we witness that going over the candidate edges more than once almost does not change the graph. Assuming that the degree of nodes in the graph is bounded, complexity is dominated by the sorting operation, and so if the number of candidate edges $|S| = K$, complexity is $K \log K$. We term this algorithm *high-to-low*

(HTL), since we scan edges from high-score to low-score.

The HTL algorithm computes a transitive graph, but it is not guaranteed to be an FRG – nodes may have more than one parent in the reduced graph. Therefore, we need to make a small modification to HTL: before computing the transitive closure of an edge (i, j) (Line 5), we must verify that inserting (i, j) will not create a violation of the FRG constraint. This is done by going over all edges $(i, k) \in E$ and checking the relation between j and k as illustrated by Figure 5.2 – if for every k either $(j, k) \in E$ or $(k, j) \in E$, 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, which is a violation of the FRG constraint. If we assume that the degree of nodes is bound by a constant, then checking the validity of (i, j) takes constant time. The modified algorithm is termed *HTL-tree*.

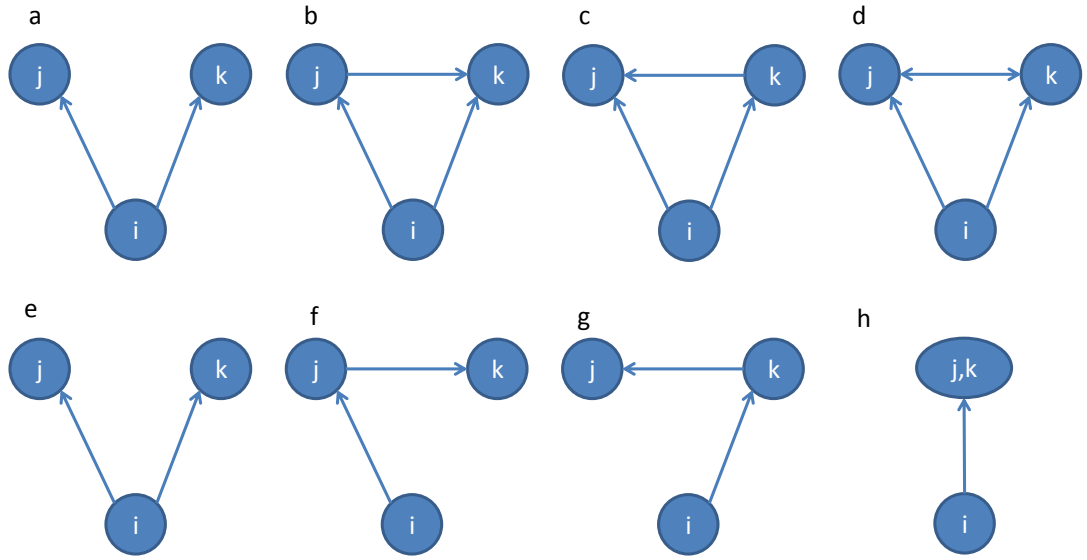

Figure 5.2: HTL-tree initialization. Top row shows all four configurations of a pair of nodes (j, k) when inserting a candidate edge (i, j) into a graph that contains the edge (i, k). Bottom row shows the corresponding reduced graphs. Cases (a) and (e) exemplify that if both edges (j, k) and (k, j) are not in the graph, then inserting (i, j) will cause an FRG violation. In all other cases there is no FRG violation in the reduced graphs.

Tree-node-and-component-fix After applying HTL-tree as an initialization procedure, we can use Tree-node-fix (TNF) to improve the value of the objective function. Assuming that entailment graphs are FRGs allows us to employ the *node re-attachment*

operation in linear time (Section 4.2.4.1). However, this assumption also enables to perform other graph operations efficiently. In this dissertation we do not thoroughly investigate this set of graph operations, but we do suggest here a natural extension to the TNF algorithm that will allow us to better explore 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 5.3. 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 employs one possible variant. In TNCF we first perform node re-attachment until convergence as usual (after initialization), but then compute the SCC graph and perform component re-attachment until convergence. Component re-attachment is practically identical to node re-attachment, except that the reduced graph is guaranteed to be a forest. There is one difference however between node re-attachment and component re-attachment. Recall from Section 4.2.4.1 that there are 3 cases we consider when re-attaching a node: (1) inserting the node into a component (2) inserting the node as a child of a component (3) inserting the node as a root. For simplicity, when re-attaching a component we only consider cases 2 and 3 and do not allow case 1, in which components should be merged. After performing component re-attachment until convergence we again perform node re-attachments and then component re-attachments and so on, until the entire process converges.

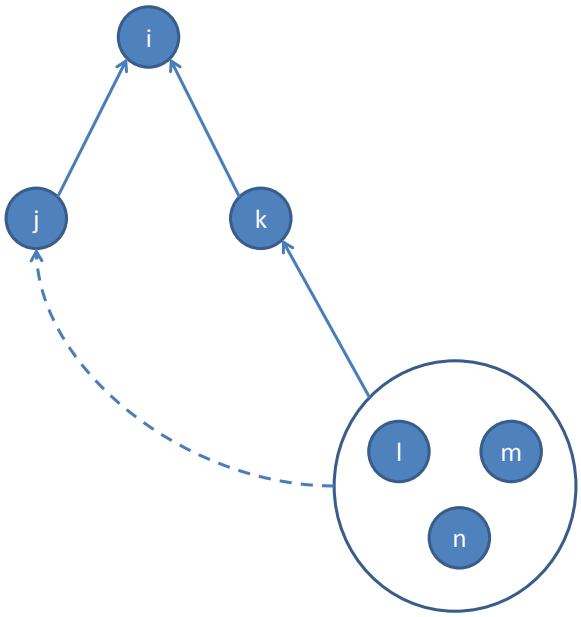

Figure 5.3: An example graph where single node re-attachment is a disadvantage.

In summary, our global learning algorithm decomposes the graph into weak connectivity components (Section 4.1) and then applies an approximation algorithm over each weak connectivity component. HTL-tree is employed for initialization, and then TNCF is used to improve the graph, where we perform node re-attachments and component re-attachments iteratively until convergence. Note also that one can simply utilize HTL and HTL-tree as simple and quick approximation algorithms. The size of the learned global resource depends on the value of the sparseness parameter λ. In Section 5.3.1 we empirically test the algorithms HTL, HTL-tree, TNF and TNCF with various values of λ.

5.2.3 Syntactic representation

The representation of rules in the resources we have hitherto learned is relatively shallow: it contains the lexical realization of the predicates and the argument ordering, for

instance, '*X have an impact on Y* $\Rightarrow$ *Y be affected by X*'. However, many semantic inference applications work over deeper syntactic representations, most commonly dependency parse trees. For example, the syntactic representation for the aforementioned rule is '$X \xleftarrow{nsubj}$ *have* $\xrightarrow{dobj}$ *impact* $\xrightarrow{prep}$ *on* $\xrightarrow{pobj}$ $Y \Rightarrow Y \xleftarrow{nsubjpass}$ *affect* $\xrightarrow{prep}$ *by* $\xrightarrow{pobj}$ X'. In this section we describe a general method that given a shallow representation for a binary predicative template (where both arguments are variables), outputs a syntactic representation. We restrict ourselves to the most popular syntactic representation, which is the dependency paths suggested by DIRT (see Section 2.2.1.1). The general idea is to first find an instantiation for each shallow predicative template in the REVERB data set, parse the instantiation, and then generate a dependency-path for each predicative template from its parsed instantiation. We next provide the details of this process.

Extracting corpus instantiations In this step we find for each of the predicative templates in the preprocessed REVERB data set an instantiation to parse. The main problem is that parsers assume that their input is a sentence, while we would like to provide the parser with a REVERB tuple (concatenation of arg_1, the predicate, and arg_2). The parser is often not robust enough to handle such an input and will provide a wrong parse tree.

To confront this problem we make use of the confidence scores provided with every REVERB extraction. For each of the input predicative templates we find the extraction with the highest confidence score, assuming that high confidence extractions will be easier to parse. This assumption is backed by the fact that the highest scoring feature in the classifier that assigns extractions with a confidence score is the feature that specifies whether the concatenation of arg_1, the predicate, and arg_2 covers the entire original sentence. Naturally, we use the unnormalized extraction since the parser is very sensitive to morphological changes in the original sentence. For instance, the instantiation found for the predicative template *'X have an impact on Y'* is *'Zinc has an impact on our immune system.'*. We parsed the resulting 103,315 instantiations using the EasyFirst parser (70).

Generating dependency-based representation In this step the input is (a) a shallow-representation of a predicative template, *e.g., 'X have an impact on Y'* (b) a

parsed instantiation of that template, *e.g.,* the parse tree given in Figure 5.4a for the instantiation *'Zinc has an impact on our immune system'*. The output is a dependency-based representation if one can be recovered. The first step is to replace in the parse tree arg_1 with a variable node X and arg_2 with a variable node Y (argument nodes are colored yellow in Figure 5.4). This is done by validating that all words in arg_1 (or arg_2) and only words in arg_1 (or arg_2) are dominated by one of the argument words – namely, it's head. If this is the case then the argument subtree is replaced by the corresponding variable node, otherwise, no dependency-based representation is recovered.

Next, the unique dependency path from the X variable to the Y variable is traversed and extracted. For instance, given the predicative template *'have an impact on'* and the instantiation *'Zinc has an impact on our immune system.'*, the dependency path will be *'X* $\xleftarrow{nsubj}$ *have* $\xrightarrow{dobj}$ *impact* $\xrightarrow{prep}$ *on* $\xrightarrow{pobj}$ *Y'*. Some of the words in the predicative template might not occur on the path, for example the word *'an'* (colored green in Figure 5.4a), which is a determiner of the noun *'impact'*. Since we support only dependency paths in our syntactic representation, we need to decide whether omission of words that are not on the path from the predicative template modifies its semantic meaning, and in this case the dependency path is discarded, or whether meaning is preserved and in this case omission is allowed. This is determined by a few manually-constructed rules – we alow omission of determiners, punctuation marks, and pronouns (such as *'my'*, *'your'*, etc.) since they do not bear much semantic content. We also allow omission of some other adjectives and adverbs such as *'very'*, *'just'*, etc. For example, the determiner *'an'* was omitted in the above example , which is legal. However, if the predicative template was *'X have a direct impact on'*, then the same process would lead to a dependency path that does not contain the adjective *'direct'* (colored red in Figure 5.4b) and the resulting dependency-based representation would be discarded, since this adjective modifies the meaning of the predicative template.

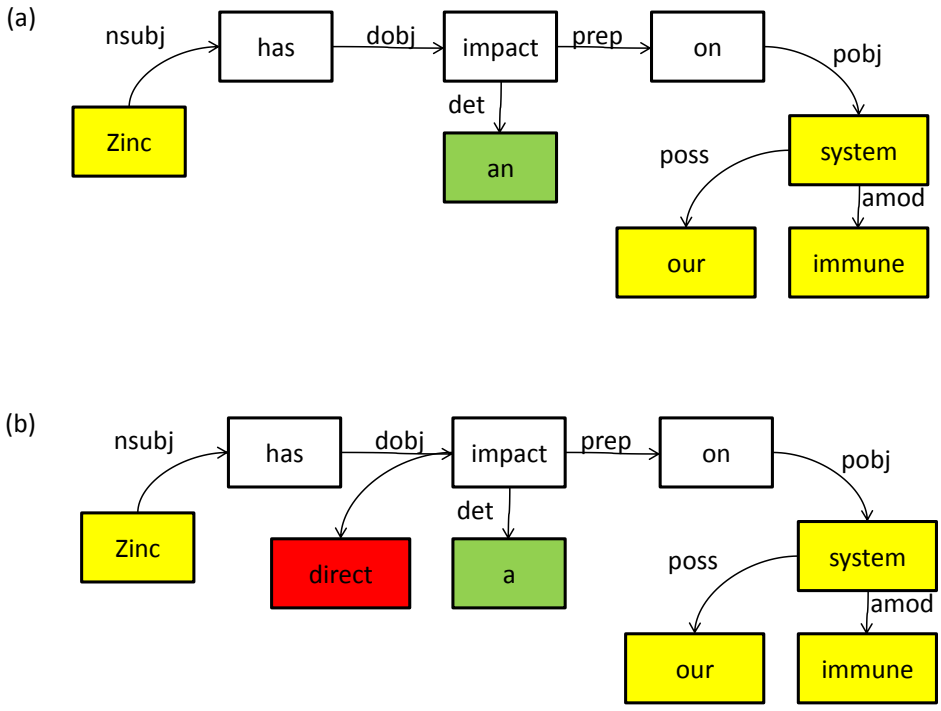

Figure 5.4: Dependency parse trees for the instantiations *'Zinc has an impact on our immune system'* (a) and *'Zinc has a direct impact on our immune system'*. Nodes colored yellow correspond to the arguments of the predicative template. Nodes colored green (red) correspond to nodes that are (are not) allowed to be omitted from the syntactic representation.

The process above allowed us to extract a dependency-based representation for 79,599 out of the 103,315 predicative templates (77%). The final dependency-based rule-base is composed of all rules for which we were able to map both the LHS and the RHS to a dependency-based representation. For example, out of about 15 million rules in the local resource for which the score $s_{ij} > 0$, we generated almost 10 million distinct dependency-based rules.

5.3 Experimental Evaluation

In this section our goal is to evaluate both the local and global resources and demonstrate that they outperform previous state-of-the-art baselines. Unfortunately, evaluating entailment rules is not trivial, and there is currently no common evaluation standard.

One option for evaluating entailment rule resources is to measure their impact on an end task, as that is what ultimately interests an inference system developer. However, a disadvantage of this approach is that performance on an end task often depends on multiple factors within the overall system and thus it is hard to assess the effect of a single resource. An example is the Recognizing Textual Entailment (RTE) framework (48) where it has been shown that resources' impact can vary considerably from one system to another. These issues have also been noted by Sammons et al. (152), who discussed the need for more detailed annotations for RTE components, and similarly by LoBue and Yates (108). Despite these disadvantages we evaluate our learned resources in the RTE framework (Section 5.3.2).

Another option is to let annotators judge rule correctness *directly*, that is by asking them to judge the correctness of a given rule (156, 159). However, Szpektor et al. (171) observed that directly judging rules out of context often results in low inter-annotator agreement. To remedy that, Szpektor et al. and Bhagat et al. (21) proposed *"instance-based evaluation"*, in which annotators are presented with an *application* of a rule in a particular context and need to judge whether it results in a valid inference. This simulates the utility of rules in an application and yields high inter-annotator agreement. Unfortunately, their method requires lengthy guidelines and substantial annotator training effort, which are time consuming and costly. Thus, a simple, robust and replicable evaluation method is needed.

Recently, crowdsourcing services such as Amazon Mechanical Turk (AMT) and CrowdFlower (CF)[1] have been employed for semantic inference annotation (117, 123, 166, 182). In Section 5.3.1 we propose a novel instance-based evaluation framework for entailment rules that takes advantage of crowdsourcing. This procedure substantially simplifies annotation of rule applications and avoids annotator training completely. The novelty in this evaluation framework is two-fold: (1) We simplify instance-based

[1] https://www.mturk.com and http://crowdflower.com

evaluation from a complex decision scenario to two independent binary decisions. (2) We apply methodological principles that efficiently communicate the definition of the "entailment" relation to untrained crowdsourcing workers (*Turkers*).

5.3.1 Crowdsourcing-based entailment rule evaluation

We first describe our general crowdsourcing-based evaluation framework (Section 5.3.1.1) and then present experiments and results (Sections 5.3.1.2-5.3.1.3).

5.3.1.1 Evaluation framework

As mentioned, in instance-based evaluation individual rule applications are judged rather than rules in isolation, and the quality of a rule-resource is then evaluated by the validity of a sample of applications of its rules. Rule application is performed by finding an instantiation of the rule LHS in a corpus (termed *LHS extraction*) and then applying the rule on the extraction to produce an instantiation of the rule RHS (termed *RHS instantiation*). For example, the rule *'X observe Y⇒X celebrate Y'* can be applied on the LHS extraction *'they observe holidays'* to produce the RHS instantiation *'they celebrate holidays'*.

The target of evaluation is to judge whether each rule application is valid or not according to the standard textual entailment definition. In the aforementioned example, the annotator is expected to judge that *'they observe holidays'* entails *'they celebrate holidays'*. In addition to this straightforward case, two more subtle situations may arise. The first is that the LHS extraction is meaningless. We regard a proposition as meaningful if a human can easily understand its meaning (despite some simple grammatical errors). A meaningless LHS extraction usually occurs due to a faulty extraction process (e.g., Table 5.7, Example 2). Such rule applications can either be removed from the generated examples so that the rule-base is not penalized (since the problem is in the extraction procedure), or can be used as examples of non-entailment, if we are interested in overall performance. A second situation is a meaningless RHS instantiation, usually caused by rule application in a wrong context. This case is tagged as non-entailment (for example, applying the rule *'X observe Y⇒X celebrate Y'* in the context of the extraction *'companies observe dress code'*).

Phrase	Meaningful	Comments
1) Doctors be treat Mary	Yes	Annotators are instructed to ignore simple inflectional errors
2) A player deposit an	No	Bad extraction for the rule LHS *'X deposit Y'*
3) humans bring in bed	No	Wrong context, result of applying *'X turn in Y→X bring in Y'* on *'humans turn in bed'*

Table 5.7: Examples of phrase "meaningfulness" (Note that the comments are not presented to Turkers).

Each rule application therefore requires an answer to the following three questions: 1) Is the LHS extraction meaningful? 2) Is the RHS instantiation meaningful? 3) If both are meaningful, does the LHS extraction entail the RHS instantiation?

Crowdsourcing Previous works using crowdsourcing noted some principles to help get the most out of the service(181). In keeping with these findings we employ the following principles: **(a) Simple tasks**. The global task is split into simple sub-tasks, each dealing with a single aspect of the problem. **(b) Do not assume linguistic knowledge by annotators**. Task descriptions avoid linguistic terms such as "tense", which confuse workers. **(c) Gold standard validation**. Using CF's built-in methodology, gold standard (GS) examples are combined with actual annotations to continuously validate annotator reliability.

We split the annotation process into two tasks, the first to judge meaningfulness (Questions 1 and 2 above) and the second to judge entailment (Question 3 above). In Task 1, the LHS extractions and RHS instantiations of all rule applications are separated and presented to different Turkers independently of one another. This task is simple, quick and cheap and allows Turkers to focus on the single aspect of judging phrase meaningfulness. Rule applications for which both the LHS extraction and RHS instantiation are judged as meaningful are passed to Task 2, where Turkers need to decide whether a given rule application is valid. If not for Task 1, Turkers would need

to distinguish in Task 2 between non-entailment due to (1) an incorrect rule (2) a meaningless RHS instantiation (3) a meaningless LHS extraction. Thanks to Task 1, Turkers are presented in Task 2 with two meaningful phrases and need to decide only whether one entails the other.

To ensure high quality output, each example is evaluated by three Turkers. Similarly to Mehdad et al. (117) we only use results for which the confidence value provided by CF is greater than 70%.

We now describe the details of both tasks. Our simplification contrasts with Szpektor et al. (171), whose judgments for each rule application are similar to ours, but had to be performed simultaneously by annotators, which required substantial training.

Task 1: *Is the phrase meaningful?*
In keeping with the second principle above, the task description is made up of a short verbal explanation followed by positive and negative examples. The definition of "meaningfulness" is conveyed via examples pointing to properties of the automatic phrase extraction process, as seen in Table 5.7.

Task 2: *Judge if one phrase is true given another.*
As mentioned, rule applications for which both sides were judged as meaningful are evaluated for entailment. The challenge is to communicate the definition of "entailment" to Turkers. To that end the task description begins with a short explanation followed by "easy" and "hard" examples with explanations, covering a variety of positive and negative entailment "types" (Table 5.8).

Defining "entailment" is quite difficult when dealing with expert annotators and still more with non-experts, as was noted by Negri et al. (123). We therefore employ several additional mechanisms to get the definition of entailment across to Turkers and increase agreement with the GS. We run an initial small test run and use its output to improve annotation in two ways: First, we take examples that were "confusing" for Turkers and add them to the GS with explanatory feedback presented when a Turker answers incorrectly. (E.g., the pair *('The owner be happy to help drivers', 'The owner assist drivers')* was judged as entailing in the test run but only achieved a confidence value of 0.53). This is similar in spirit to boosting approaches in Machine Learning (61), that use misclassified instances to improve subsequent training. Second, we add examples that were annotated unanimously by Turkers to the GS to increase its size, allowing CF to better estimate Turker's reliability (following CF recommendations, we

Example	⇒	Explanation given to Turkers
LHS: The lawyer sign the contract RHS: The lawyer read the contract	Yes	As the lawyer signed the contract, he most likely have read it.
LHS: John be related to Jerry RHS: John be a close relative of Jerry	No	The LHS can be understood from the RHS, but not the other way around as the LHS is more general.
LHS: They have mixed feeling about their council RHS: They be disappointed with their council	No	It is possible to have mixed feelings about the council like "hope" or "worry" without being disappointed.
LHS: Women be at increased risk of cancer RHS: Women die of cancer	No	Although the RHS is correct, it cannot be understood from the LHS.

Table 5.8: Examples given in the description of Task 2.

aim to have around 10% GS examples in every run). In Section 5.3.1.2 we show that these mechanisms improved annotation quality. We now turn to an application of our evaluation framework over the REVERB data set.

5.3.1.2 Experimental setting

Gold standard generation We employ our crowdsourcing-based framework to generate annotated gold standard examples with which we will evaluate our local and global resources. We first sample 5,000 extractions from our preprocessed REVERB data set. Next, we create a broad rule-base that represents the union of rules learned from all the resources that we wish to evaluate, in the following manner: for each of the 5,000 sampled extractions we take the extraction predicate p and find the top-K (K=20) predicates p' that obtained the highest score (p, p') according to the six distributional similarity measures we previously computed. This results in a rule-base of 156,133 rules of the form '$p \Rightarrow p'$', which are relevant for our sampled extractions. We then go over each extraction and apply all rules that are relevant for that extraction. For example, given the extraction *'Anne attend Smith College'* and the rule '*attend* ⇒ *be*

admitted to' we generate the rule application '*Anne attend Smith College* ⇒ *Anne be admitted to Smith College*'. This results in 220,878 rule applications. Last, we randomly sample 20,000 rule instantiations and annotate their applications using our crowdsourcing-based methodology.

In Task 1, 281 rule applications were annotated as meaningless LHS extraction and were consequently discarded. 1,012 rule application were annotated as meaningful LHS extraction but meaningless RHS instantiation and so considered as non-entailments. Of the remaining rule applications 10,443 were discarded due to low CF confidence, and 8,264 rule applications were passed on to Task 2, as both sides were judged meaningful. In Task 2, 5,555 rule applications were judged with a high confidence and supplied as output, 2,447 of them as positive entailment and 3,108 as negative. Overall, 6,567 rule applications (2,447 positives and 4,120 negatives) were annotated for a total cost of $1000. The annotation process took about one week. We used half of the annotated gold standard as a training set and the other half as a test set.

In tests run during development we experimented with Task 2 wording and GS examples, seeking to make the definition of entailment as clear as possible. To do so we randomly sampled and manually annotated 200 rule applications (from the initial 20,000), and had Turkers judge them. In our initial test, Turkers tended to answer "yes" more often than compared to our own annotation, with 0.79 agreement between their annotation and ours, corresponding to a kappa score of 0.54. After applying the mechanisms described in Section 5.3.1.1, false-positive rate was reduced from 18% to 6% while false-negative rate only increased from 4% to 5%, corresponding to a high agreement of 0.9 and kappa of 0.79.

In our test, 63% of the 200 rule applications were annotated unanimously by the Turkers. Importantly, all these examples were in perfect agreement with our own annotation, reflecting their high reliability. For the purpose of evaluating the resources learned by the algorithms we used annotations with CF confidence ≥ 0.7, for which kappa is 0.99.

Evaluated algorithms We would like to evaluate both the local resource and the global resource. The local resource was constructed using a local entailment classifier trained over a rich set of features (Section 5.2.2.1). Therefore, we would like to examine whether these features improve performance compared to a classifier trained over

only distributional similarity features. Recall that in the evaluation of Section 3.5 over small graphs augmenting lexicographic and string-similarity features to the distributional similarity features did not improve the local classifier performance. We wish to investigate whether in the more domain-general setting of this chapter these features can yield better results. We term the full local entailment classifier $clsf_{all}$ and the distributional similarity classifier $clsf_{ds}$. In addition, we compare the two classifiers to the six individual distributional similarity measures that we computed. Recall (Section 3.5) that these measures were computed by combining two feature vector representations (termed in this section *DIRT* and *TEASE*) with three similarity measures (*Lin*, *BInc* and *Cover*). Last, we add the prior knowledge expressed by local constraints (see Section 5.2.2.1) to all algorithms, and report only results that include this knowledge.

All local algorithms provide a score for each candidate pair of predicative templates. Thus, they require to set a threshold parameter that determines whether entailment holds. As in previous chapters, we avoid setting this parameter by computing recall, precision and F_1 for various thresholds, which results in a precision-recall curve for which we can compute the area under the curve (AUC).

Our global resource is computed over a more restricted set of 10,000 frequent predicates, yielding 20,000 graph nodes. Therefore, the size of the test set relevant for the global resource is also smaller – out of 3,283 examples in the original test set we can utilize 1,734 annotated examples (649 positives and 1,085 negatives). Notice that since we chose frequent predicates and the test set was sampled by predicate frequency, 53% of the test set is usable although the number of predicates is 10 times smaller. We compare our local algorithm with various global methods: HTL, HTL-tree, TNF and TNCF. We generate various recall-precision points by varying the sparseness parameter λ. All experiments are run over a server with 8 cores and 20GB of virtual memory.

5.3.1.3 Results

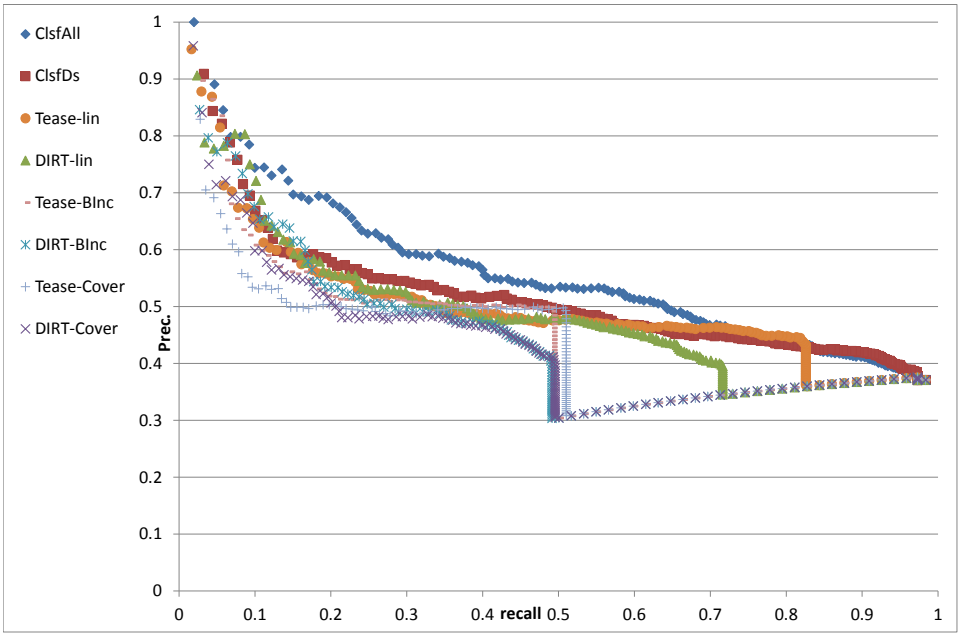

Figure 5.5: Precision-recall curve for all local methods.

Method	AUC
clsf_{all}	**56.1**
clsf_{ds}	52.1
TEASE-Lin	50.3
DIRT-Lin	49.2
TEASE-BInc	45.6
DIRT-BInc	45.4
TEASE-Cover	44.3
DIRT-Cover	44.3

Table 5.9: Area under the precision-recall curve for all local methods.

Figure 5.5 and Table 5.9 present the results of our local resource evaluation. The figure presents the precision-recall curve and it is clear that employing a rich set of

λ	**recall**	**precision**	$\mathbf{F_1}$
0	0.66	0.49	0.56
0.05	0.48	0.53	0.5
0.1	0.39	0.57	0.47
0.15	0.32	0.59	0.42
0.2	0.29	0.6	0.39

Table 5.10: Recall, precision, and F_1 of $clsf_{all}$ for various values of λ. The column λ indicates that only pairs of predicates for which $s_{ij} > \lambda$ are classified as entailing.

features improves performance in this domain-general setting comparing to using only distributional similarity features. In addition, the distributional similarity classifier performs better than each of the distributional similarity measures on their own.

This conclusion is re-affirmed quantitatively in Table 5.9, where the AUC is specified. The relative improvement in AUC of our local resource $clsf_{all}$ over $clsf_{ds}$ is 8%, and $clsf_{ds}$ in turn has higher AUC than the best individual distributional similarity measure, which is TEASE-Lin. It is also notable that the *'Lin'* similarity measure outperforms *'Cover'* and *'BInc*, especially when recall is higher than 0.5. In addition, the *'TEASE'* representation performs better than the *'DIRT'* representation, most probably since the problem of sparseness in our data is relatively minor.

Last, we present for completeness the recall, precision and F_1 obtained by $clsf_{all}$ for various values of λ in Table 5.10. For instance, when $\lambda = 0$ we obtain the classification threshold of the local classifier, where all pairs of predicates for which $s_{ij} > 0$ are classified as entailing. Naturally, as λ grows, precision increases and recall drops. Also note that the classifier was trained to maximize F_1 and indeed when $\lambda = 0$ we obtain the best F_1.

Next, we would like to evaluate the global algorithms. First, notice that our methods allow us to employ global constraints in a large graph containing 20,000 nodes. Figure 5.6 presents the precision-recall curve of all global algorithms comparing to the local classifier for $0.05 \leq \lambda \leq 2.5$. The most evident property of this curve is that $clsf_{all}$ is able to reach much higher recall values than all global algorithms. This means that adding a global transitivity constraint prevents the algorithm from adding many correct

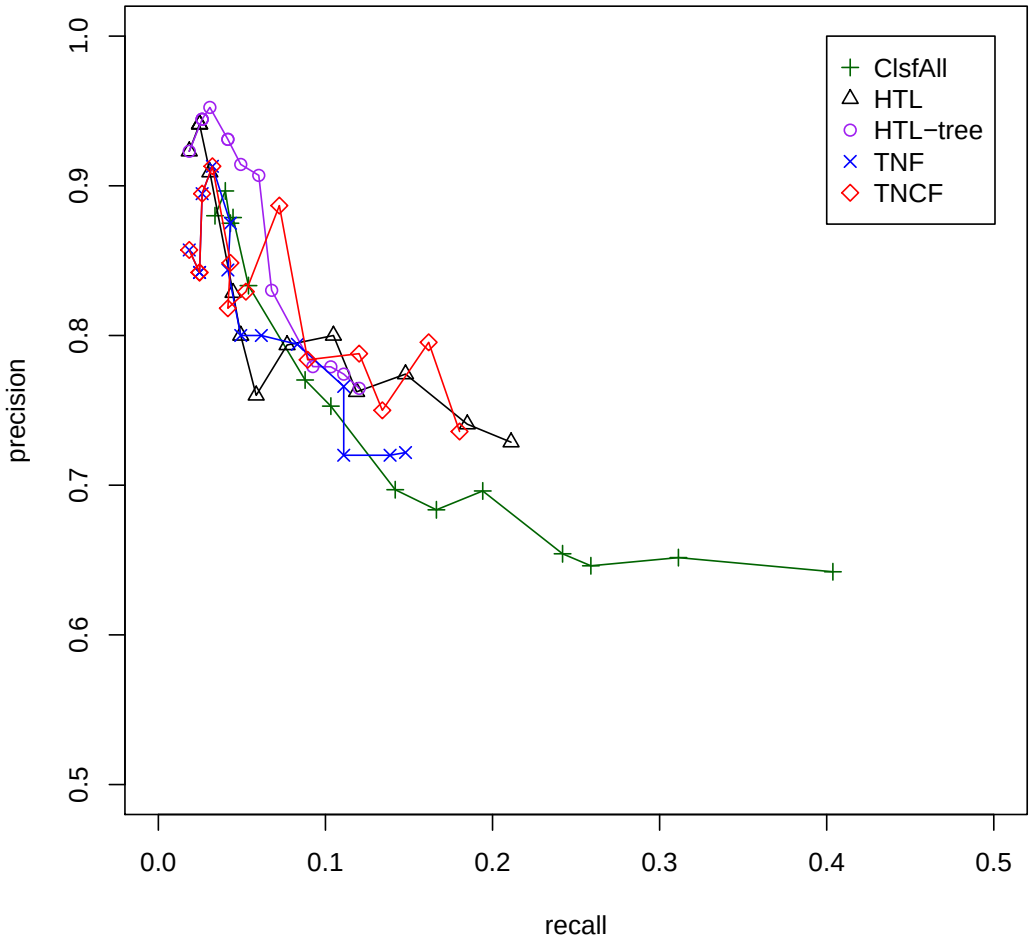

Figure 5.6: Precision-recall curve comparing global algorithms to the local algorithm.

edges that have positive weight, since this would cause the addition of many other edges (to preserve transitivity) that have negative weight. We have witnessed this behavior in Section 3.4.4, but a plausible reason for the strong manifestation in our case might be the fact that predicates are ambiguous. Consequently, positive-weight edges connect connectivity components through a predicate that has more than one meaning. This means that a natural direction for future research is to develop global algorithms that identify predicate ambiguity and learn the correct rules in various contexts. We will return to this point in our analysis in Section 5.4 and suggest possible research directions for combining disambiguation mechanisms with global learning in Section 7.1.

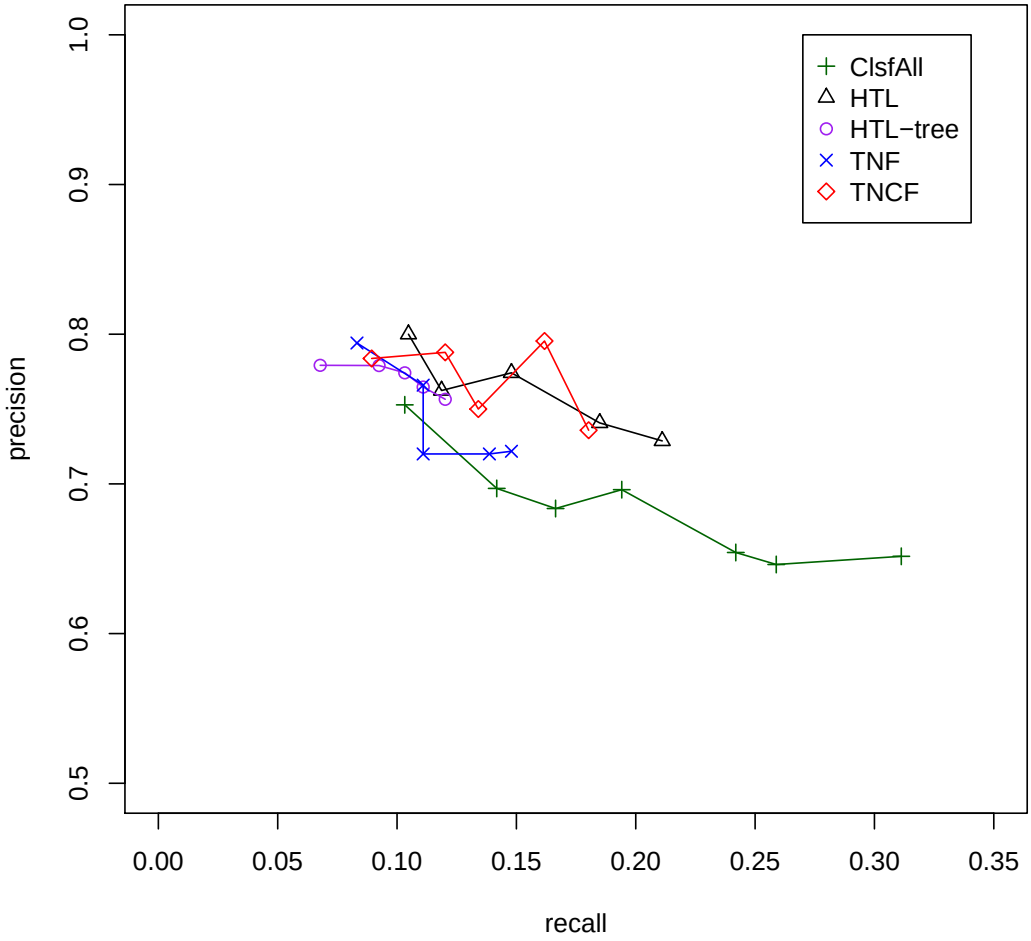

Figure 5.7: Precision-recall curve comparing global algorithms to the local algorithm in the recall range 0.1-0.3.

method	λ	**recall**	**precision**	$\mathbf{F_1}$	**# of rules**
clsf_{all}	1	0.1	0.75	0.18	39,872
clsf_{all}	0.8	0.14	0.7	0.24	57,276
clsf_{all}	0.6	0.17	0.68	0.27	65,232
clsf_{all}	0.4	0.19	0.7	0.3	77,936
clsf_{all}	0.2	0.24	0.65	0.35	116,832
HTL	0.2	0.12	0.76	0.21	88,338
HTL	0.15	0.15	0.77	0.25	111,812
HTL	0.1	0.18	0.74	0.3	153,820
HTL	0.05	0.21	0.73	0.33	275,766
HTL-tree	0.2	0.09	0.78	0.17	48,986
HTL-tree	0.15	0.1	0.78	0.18	55,756
HTL-tree	0.1	0.11	0.77	0.19	65,802
HTL-tree	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

Table 5.11: Recall, precision, F_1, and the number of learned rules for clsf_{all} and several global algorithms over several values of the parameter λ.

Although the global algorithms are limited in their recall, for recall values of 0.1-0.2 they substantially improve precision over the local clsf_{all}. To better observe this result, Figure 5.7 and Table 5.11 present the results for the recall range 0.1-0.3. Comparing TNCF and clsf_{all} shows that the TNCF algorithm improves over clsf_{all} by 5-10 precision points: In the recall range of 0.1-0.2 the precision of clsf_{all} is 0.68-0.75, while 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-tree as an initialization method for TNCF, which on its own is not a very good approximation algorithm. Comparing HTL-tree to clsf_{all}, we observe that it

is unable to reach high recall values and it only marginally improves precision comparing to clsf_{all}. Applying TNF over HTL-tree also provides disappointing results – it increases recall comparing to HTL-tree, but precision is not better than clsf_{all}. 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 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-tree.

A surprising finding is the good results obtained by HTL, whose performance is comparable to TNCF. Recall that this is a simple algorithm that sorts the candidate edges by weight and tries to insert them one-by-one while maintaining a transitivity constraint. Comparing HTL to HTL-tree also shows that the FRG constraint added by HTL-tree substantially restricts the set of edges that can be inserted into the graph, since the recall obtained by HTL-tree is much lower than the one obtained by HTL. This is an indication that in our large graph, which contains ambiguous predicates, the FRG property might not hold, and thus developing global algorithms that handle ambiguity is highly important.

Last, Figure 5.8 presents the results of graph decomposition as described in Section 4.1.2. 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 component is 90% of the total number of graph nodes. Thus, contrary to the typed graphs presented in Section 4.1.3, untyped graphs that contain ambiguous predicates do not decompose well into small components.

To summarize the results from this section, it is clear that the local entailment classifier we have trained, which makes use of a rich set of features, performs better than a state-of-the-art classifier trained over only distributional similarity features, which we presented in previous work (17, 19). As for our global learning algorithms, we demonstrate that they allow us to scale computationally to large graphs and observe that even when predicates are ambiguous we are able to improve precision for low values of recall. Nevertheless, there are many indications that our modeling assumptions are violated when applied over a large graph with ambiguous predicates: both the

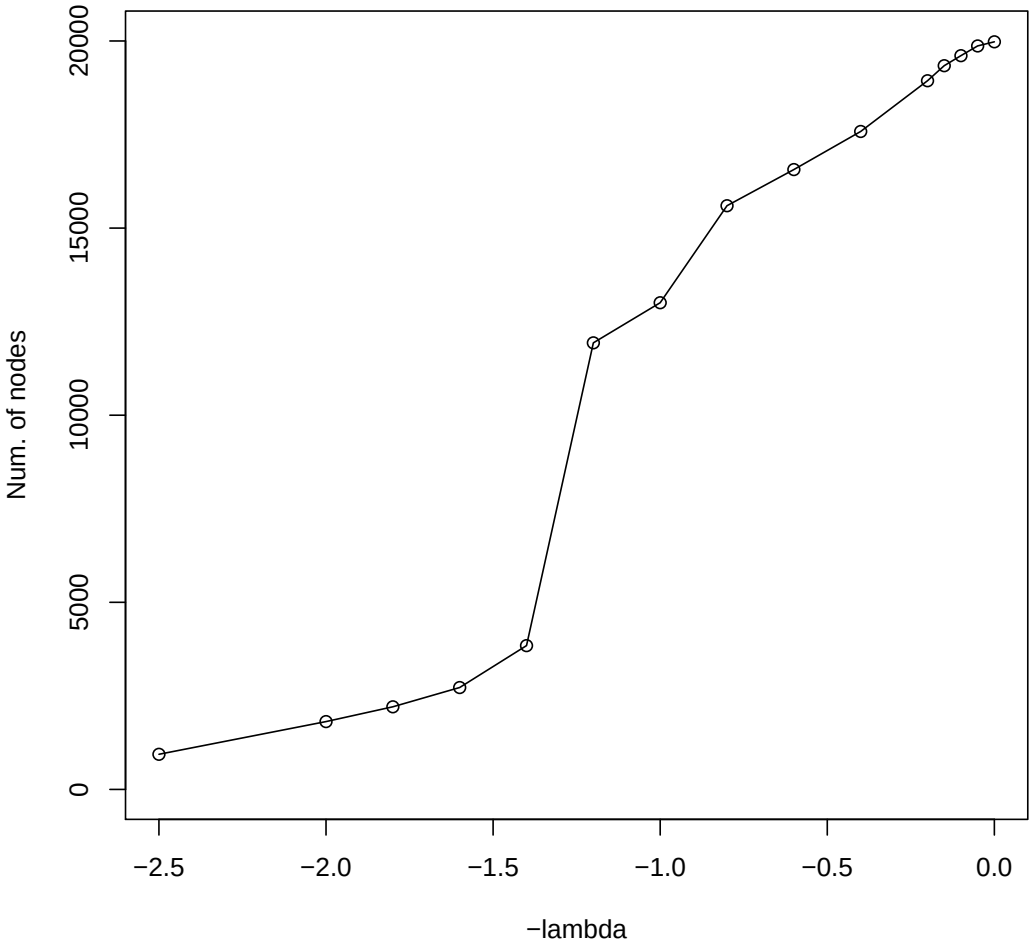

Figure 5.8: The number of nodes in the largest component as a function of the sparseness prior $-\lambda$.

transitivity as well as the FRG constraints limit the recall value that we are able to reach, and the graph does not decompose very well into components as with typed entailment graphs. We will further analyze these findings in Section 5.4 and discuss possible solutions in Section 7.1.

5.3.2 RTE evaluation

A natural way to evaluate knowledge resources is to test their utility in a real-world scenario such as recognising textual entailment (RTE, see Section 1.2). Therefore, we test our learned knowledge resources by using them as part of a textual entailment system over benchmark RTE data sets. For a textual entailment engine we choose BIUTEE[1] (5, 167, 168), a transformation-based natural language inference system. Given a text and a hypothesis, BIUTEE "proves" the hypothesis by performing a sequence of transformations over the text until it is identical to the hypothesis. Then, it estimates the cost of the proof and subsequently determines whether the proof is valid or not. Importantly, transformations are mainly based on entailment rules that are extracted from knowledge resources. Therefore, knowledge resources may be compared by running BIUTEE with various resources over RTE data sets and measuring which knowledge resources lead to better performance.

We run BIUTEE over several RTE data sets: RTE-3, RTE-5, RTE-6, RTE-7. Each data set contains a training set and a test set where each example comprises a text-hypothesis pair[2]. In each run, BIUTEE is trained over the training set and evaluated against the test set. Training determines the cost associated with the application of each transformation. Table 5.12 details the number of positive and negative examples in each of the RTE data sets.

A system is evaluated by comparing the classifications of text-hypothesis pairs provided by the system with a gold standard annotation. In RTE-3 and RTE-5 the data set is balanced, that is, the number of positive examples is equal to the number of negative examples and so the official RTE evaluation measure is simply *accuracy* – the number of correct classifications out of the entire data set. In RTE-6 and RTE-7 the data set reflects a more natural distribution of examples, where most examples are negative. Therefore, the official RTE evaluation measure is F_1.

[1] `http://u.cs.biu.ac.il/~nlp/downloads/biutee/protected-biutee.html`

[2] We do not test on RTE-4 since it does not contain a training set.

Data Set	Train-Pos.	Train-Neg.	Test-Pos.	Test-Neg.
RTE-3	400	400	400	400
RTE-5	300	300	300	300
RTE-6	897	15,057	945	19,027
RTE-7	1,136	20,284	1,308	21,118

Table 5.12: Number of positive and negative examples in RTE data sets.

We evaluate three of our learned resources. Note that BIUTEE works over EasyFirst parse trees and therefore we must utilize a syntactic representation for the rules (see Section 5.2.3).

- clsf_{all} – Resource constructed by our local algorithm containing 15 million entailment rules, of which 10 million were transformed into a syntactic representation.
- HTL – Resource constructed by the HTL algorithm containing 275,766 rules ($\lambda = 0.05$). We managed to generate a syntactic representation for 191,389 rules. This is the configuration that led to highest recall among our global algorithms.
- TNCF - Resource constructed by the TNCF algorithm containing 102,565 rules ($\lambda = 0.1$). We managed to generate a syntactic representation for 70,468 rules. This is a high-precision configuration.

Our first and main goal is to compare clsf_{all} and DIRT (104), which is the most popular resource of predicative entailment rules to date, since the number of rules in both DIRT and clsf_{all} is similar – about 10 million rules. According to ablation tests reported on the ACL wiki[1], the effect of DIRT on accuracy in RTE-5 was positive for some systems and negative for others. In both cases, accuracy changed by no more than 1.5 points. In RTE-6, DIRT also had a mixed effect on different systems, where in one system F_1 dropped by more that 1.5 points, and in another system F_1 improved by almost 4 points. We also run BIUTEE with both clsf_{all} and DIRT in order to investigate whether they are complementary and thus it is beneficial to combine them. Additionally, we add WordNet as another resource since WordNet is a high-precision manually-constructed knowledge-base that can complement predicative entailment rules by supplying information on lexical items such as nouns. WordNet is

[1] `http://aclweb.org/aclwiki/index.php?title=RTE_Knowledge_Resources#Ablation_tests`

also the most popular resource used by entailment systems in past RTEs, as is evident from the ablations tests reported on the ACL wiki.

BIUTEE requires specifying a top-K parameter when utilizing resources of predicative entailment rules. This parameter controls the number of rules that share the same RHS that can be used. For example, if $K = 10$ then for every RHS we will consider only the 10 rules with highest score that contain that RHS. Naturally, when K is small the engine has access to less rules. Our local algorithms provide a classification score for every learned rule. However, we prefer to avoid tuning of the parameter K since we noticed that the optimal K varies considerably from one case to the other. Thus, we run BIUTEE in four configurations: $K = 20$, $K = 40$, $K = 60$ and $K = 1000$ (which is equivalent to using all rules). For each RTE and each resource combination we report the configuration of K that gave best results. Note that this does not provide any resource combination an advantage over another. We believe that since the value of K may change from one application to the other, it is the responsibility of the resource user to tune K on some small development set.

Data Set	**RTE-3 (acc)**	**RTE-5 (acc)**	**RTE-6 (F_1)**	**RTE-7 (F_1)**
No resource	64.0	64.0	48.38	40.39
DIRT	64.0	64.0	48.9	41.24
clsf$_{all}$	64.17	**64.17**	49.47	40.86
DIRT+clsf$_{all}$	64	63.5	49.59	41.05
WordNet	65.75	64	48.73	40.9
WordNet+DIRT	66.13	63.67	49.49	41.14
WordNet+clsf$_{all}$	**66.5**	64	49.61	41.44
WordNet+DIRT+clsf$_{all}$	66.38	63.83	**49.62**	**41.45**

Table 5.13: RTE results comparing clsf$_{all}$ and DIRT.

Table 5.13 reports the results of all resource configurations on all RTE data sets. In line with past RTE experiments, it is easy to see that the effect of different resources is relatively small. This is since, as explained in Section 5.3, RTE is a complex task that requires many types of knowledge and thus the effect of a single resource is usually limited.

However, we still observe that the best result over each of the RTE data sets always includes clsf$_{all}$. In RTE-3 the best configuration includes WordNet and clsf$_{all}$, in RTE-

5 it includes clsf_{all} only, and in RTE-6 and RTE-7 the best configuration includes WordNet, DIRT, and clsf_{all}. In RTE-3, RTE-6 and RTE-7 using resources seems helpful - in RTE-3 accuracy increases by 2.5 points and in RTE-6 and RTE-7 F_1 increases by more than a point, which is quite reasonable. Unfortunately, resources do not seem to improve performance in RTE-5 – all resource combinations seem to have little effect on performance comparing to a configuration that uses no resources at all. Additionally, past RTE campaigns have shown that adding resources can often lead to a decline in performance of systems (as reported in the ablation tests on the ACL wiki). Therefore, the fact that employing clsf_{all} consistently improves performance across RTEs (both with and without WordNet) is encouraging.

Comparing clsf_{all} and DIRT shows that their performance is comparable with a slight advantage in favor of clsf_{all}. Out of the 8 runs (four RTE data sets either with or without WordNet), comparing clsf_{all} and DIRT, clsf_{all} outperforms DIRT in 7 runs. However, improvement is quite minor – no more than 0.5 points. The best resource combinations is the one that uses all resources – WordNet, DIRT and clsf_{all} – this combination is the best in RTE-6 and RTE-7 and second best in RTE-3. This is a good indication that the resources are of complementary nature. To conclude, our evaluation demonstrates that clsf_{all} is slightly better than DIRT and that it is beneficial to combine the two resources with each other and also with WordNet.

Next, we would like to evaluate our global algorithms HTL and TNCF. As we have shown in Section 5.3.1.3, global algorithms can improve precision comparing to clsf_{all}. However, this comes at a price in recall – the size of the learned resources is smaller than both clsf_{all} and DIRT by two orders of magnitude. Nevertheless, we test our global algorithms within BIUTEE to examine whether these relatively small resources can have a positive effect on RTE evaluations. We run both algorithms with and without WordNet over RTE-6 and RTE-7. Recall that HTL and TNCF get as input a set of predicates and a weighting function and output binary decisions regarding the graph edges. Therefore, there is no ordering on the edges and using the top-K parameter is arbitrary. Thus, we allow BIUTEE to access all of the entailment rules in the resource (by setting $K = 1000$).

Data Set	RTE-6 (F_1)	RTE-7 (F_1)
No resource	48.38	40.39
DIRT	48.9	41.24
$clsf_{all}$	49.47	40.86
HTL	48.34	40.9
TNCF	49.05	40.9
WordNet	48.73	40.9
WordNet+DIRT	49.49	41.14
WordNet+$clsf_{all}$	**49.61**	**41.44**
WordNet+HTL	49.26	41.02
WordNet+TNCF	48.56	40.77

Table 5.14: RTE results comparing HTL, TNCF, $clsf_{all}$ and DIRT.

Table 5.14 presents the results of our global algorithms HTL and TNCF comparing to $clsf_{all}$ and DIRT. Evidently, the resources HTL and TNCF do not perform as well as $clsf_{all}$ and DIRT due to their much smaller size. On the bright side, despite their relatively small size using these resources seems to provide some benefit comparing to a configuration that uses no resources at all. The best configuration for global methods combines HTL with WordNet, leading to an improvement of almost one point in both RTE-6 and RTE-7 compared to *'No resource'*. Nevertheless, the improvement we witnessed in precision does not justify the limited coverage and overall we recommend utilizing the much larger local resource $clsf_{all}$.

Data Set	RTE-6	RTE-7
DIRT	391	246
$clsf_{all}$	201	181
HTL	74	33
TNCF	48	16

Table 5.15: Number of rule applications performed by BIUTEE for the resources DIRT, $clsf_{all}$, HTL and TNCF in RTE-6 and RTE-7.

Last, to further illustrate the advantage of a large resource we compare the number of rule applications from each resource that were performed by BIUTEE in the process of proving the hypotheses from the corresponding texts. Table 5.15 shows the number

of rule applications when BIUTEE used DIRT, clsf$_{all}$, HTL and TNCF. Clearly, the number of rule applications performed by BIUTEE when employing HTL and TNCF is smaller than when employing DIRT and clsf$_{all}$. However, while the number of rules in the global resources is smaller by two orders of magnitude, the number of rule applications is smaller by only one order of magnitude approximately. This is most probably since the 10,000 predicates on which our global algorithms were trained are the most frequent ones and thus they are much more likely to actually be applied in real text.

5.4 Analysis

In this chapter we employed and evaluated local and global methods for learning entailment graphs over a large set of untyped predicates. Our results revealed several interesting issues that we would like to further investigate: (a) Why do methods that assume that entailment graphs are transitive and forest-reducible have limited recall comparing to clsf$_{all}$? (b) Does performing node re-attachments substantially affect graph structure? (c) Are scores provided by the local classifier symmetric, that is, $s_{ij} \approx s_{ji}$?

Next, we try to answer these questions in order to draw conclusions regarding future research directions.

Transitivity and FRG assumptions As we saw in Table 5.10, applying the transitivity and FRG assumptions considerably limits recall. This is reminiscent of the results obtained in Section 3.4.3, where we noticed that the main effect of transitivity is to omit "unsafe" edges that connect disparate connectivity components. Our hypothesis is that since the predicates in our graph are untyped and consequently ambiguous, the transitivity and FRG assumptions do not necessarily hold any more, which results in a recall reduction.

To get a first intuition, we would like to understand the general structure of graphs learned by local and global algorithms. We construct a graph over our 20,000 nodes (cf. Section 5.2.2.2) by inserting as edges all pairs of predicates annotated by turkers as entailing, and deleting all "isolated nodes" (nodes with zero in-degree and out-degree). This results in a graph with 1,548 nodes and almost 1,300 edges. Then, we add to this

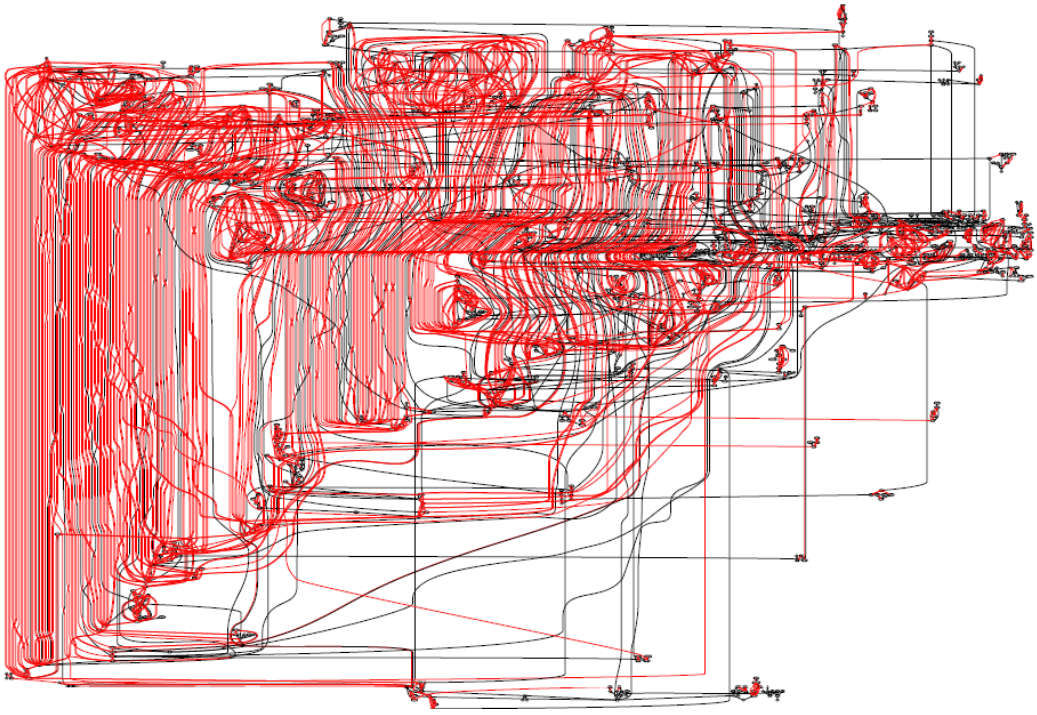

Figure 5.9: A graph over 1,548 nodes with gold standard edges in black and Clsf_{ds} edges in red.

graph all edges learned by the local classifer Clsf_{ds}, setting the sparseness parameter $\lambda = 0.2$ (*i.e.*, all edges (i, j) such that $s_{ij} > 0.2$). We create another graph by repeating the same process only using the global TNF algorithm (over the Clsf_{ds} classifer).

Figure 5.9 shows the first graph with gold standard edges in black and Clsf_{ds} edges in red. Similarly, Figure 5.10 shows the second graph with gold standard edges in black and TNF edges in red[1]. Of course, the details in the graphs are undiscernible. However, the general intuition is that there are many more "long-range" edges inserted by the local classifier comparing to TNF. This might be an indication that many of these edges connect connectivity components that are distant from one another, and thus generate FRG and transitivity violations. Looking at Figure 5.10 it seems that most of the red edges connect nodes that are relatively "close" to one another.

[1]The graphs are generated by the dot layout algorithm, which is part of the Graphviz package: (`http://www.graphviz.org/`).

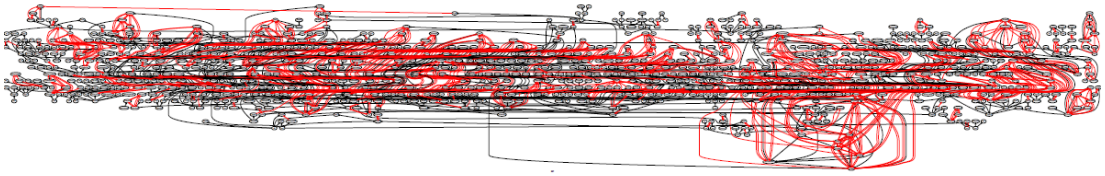

Figure 5.10: A graph over 1,548 nodes with gold standard edges in black and TNF edges in red.

Next, we wish to taker a closer look at the gold standard edges. We construct a graph with all gold standard edges and manually search for transitivity and FRG violations caused by ambiguous predicates. Our goal is to find cases where our structural assumptions are violated, and thus global methods would unnecessarily remove edges from the graph.

One example is the pair of rules '*seek* $\Rightarrow$ *apply for*' and '*apply for* $\Rightarrow$ *file for*'. The first rule was annotated in the instantiation '*Others seek medical care* $\Rightarrow$ *Others apply for medical care*'. The second rule was annotated in the instantiation '*Students apply for more than one award* $\Rightarrow$ *Student file for more than one award*'. This causes a transitivity violation since '*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 instantiations '*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, while 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 instantiations: '*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*', and so this is

	Global=True/Local=False	Global=False/Local=True
Gold standard=true	5.5%	49.2%
Gold standard=false	5.5%	39.8%

Table 5.16: Comparing disagreements between TNCF and Clsf_{all} against gold standard edges.

an FRG violation. Indeed the three instantiations 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 since '*be committed to* $\not\Rightarrow$ *contain information on*' and '*contain information on* $\not\Rightarrow$ *be committed to*'. This violation occurs due to the ambiguity of the predicate *'be dedicated to'*, which can be resolved by knowing whether the subject is human or an object that carries information (*e.g., 'The web site is dedicated to the life of lizards'*).

Now, we would like to directly compare cases where global and local methods disagree with one another. We hypothesize that since predicates are not disambiguated, then there might be cases where a global algorithm omits a correct edge due to an ambiguous predicate. We set the sparseness parameter $\lambda = 0.2$ and compare the set of edges learned by our best local algorithm Clsf_{all} and the set of edges learned by our best global algorithm TNCF.

Table 5.16 shows the disagreements of TNCF and Clsf_{all} against the test set gold standard. As expected, cases where Clsf_{all} inserts an edge while TNCF does not account for almost 90% of the disagreements. We divide such cases into two sub-cases in the following manner: we compute the weakly connected components of the graph learned by TNCF, and then for each edge inserted by Clsf_{all} and omitted by TNCF we check whether it connects two weakly-connected components or not. We hypothesize that TNCF often omits edges that connect weakly-connected components, since such edges are likely to add many paths from one component to another causing many transitivity violations. Indeed, out of all edges in the test set where Clsf_{all} inserts an edge and TNCF does not, 76.4% connect weakly-connected components in the global graph.

#	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 5.17: Correct edges inserted by Clsf_{all} and omitted by TNCF that connect weakly-connected components in TNCF. An explanation for each column in provided in the body of the section.

We now focus on this set of edges that connect weakly-connected components and are omitted by TNCF. We randomly sample five cases where TNCF was wrong and Clsf_{all} was correct – these are cases where ambiguity is likely to occur. For comparison, we also sample five cases where TNCF was right and Clsf_{all} erred. Table 5.17 shows the samples where Clsf_{all} is correct. The first column describes the rule and the second column specifies the score s_{ij} provided by the local classifier Clsf_{all}. 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 5.18 is equivalent and shows the samples where TNCF is correct.

Example 1 in Table 5.17 already demonstrates the problem of ambiguity. The LHS extraction for this rule in the crowdsourcing experiment was *'your parents turn off comments'* and indeed in this case the RHS instantiation *'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, while the RHS component deals with a more explicit and physical meaning of *cutting*. This is since 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 extraction is *'This site*

#	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 realise 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 5.18: Erroneous edges inserted by Clsf_{all} and omitted by TNCF that connect weakly-connected components in TNCF. An explanation for each column in provided in the body of the section.

was put by fans', which in this context entails the RHS instantiation *'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, while the LHS component has a more abstract sense of *availability*. The LHS extraction in this case was *'The services are available to museums'*.

Table 5.18 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 as we have already seen in Chapter 3. In Example 1 although $s_{ij} = 0.83$, which is higher than $\lambda = 0.2$, the edge is not inserted since 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 many edges. Although many times this is desirable, it also often prevents correct rules from being discovered due to problems of ambiguity. This

in our opinion is the main reason for the limited recall of global algorithms run over untyped open-domain predicates. Thus, developing a context-sensitive model (as discussed in Sections 2.3 and 7.1) is a highly important direction for future research.

Node re-attachment operation We would like to understand the effect of applying Tree-Node-Fix over the entailment graph, or more precisely check whether Tree-Node-Fix substantially alters the set of graph edges comparing to an initialization with HTL-tree. First, we run both HTL-tree and TNF with sparseness parameter $\lambda = 0.2$ (over the Clsf_{all} local classifier) and compute the proportion of edges that is in their intersection. HTL-tree learns 48,986 edges while 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-tree were removed (13,350 edges) and in addition 25,550 edges were added into the graph. Clearly, this indicates that Tree-Node-Fix dramatically changes the learned graph.

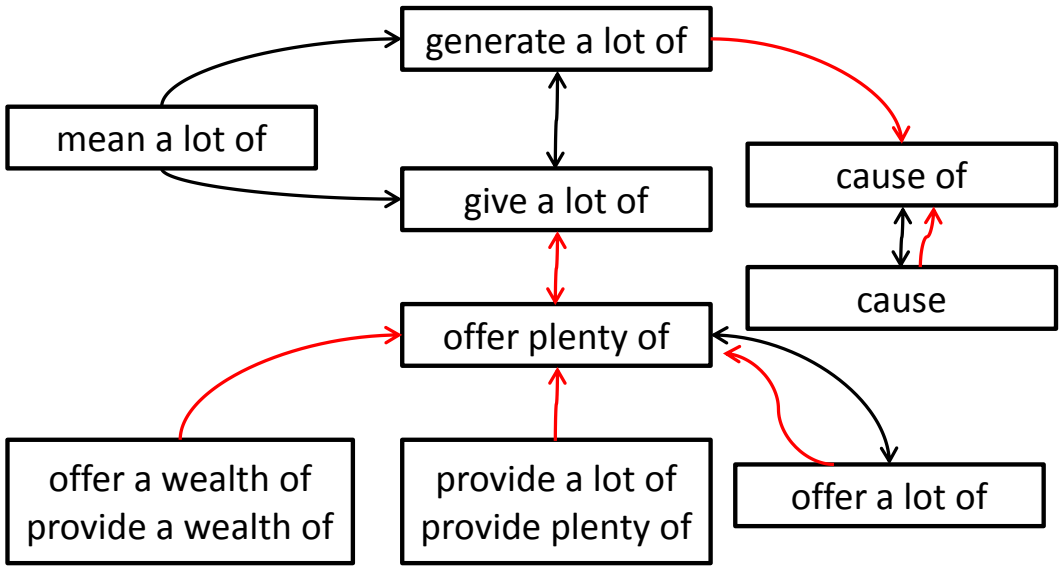

Figure 5.11: A fragment from the graphs learned by HTL-tree (black edges) and TNF (red edges).

We exemplify this with a few fragments of graphs learned by HTL-tree and TNF. Figure 5.11 shows 11 predicates[1], where the edges colored black correspond to HTL-

[1]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.

tree edges, while edges colored red correspond to TNF edges. Nodes that contain more than a single predicate represent an agreement between TNF and HTL-tree that all predicates in the node entail one another. Clearly, TNF substantially changes the initialization generated by HTL-tree. HTL-tree 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 – *e.g.,* HTL-tree decided that *'cause'* and *'cause of'* are equivalent, but TNF deleted the correct entailment rule '*cause of* ⇒ *cause*'.

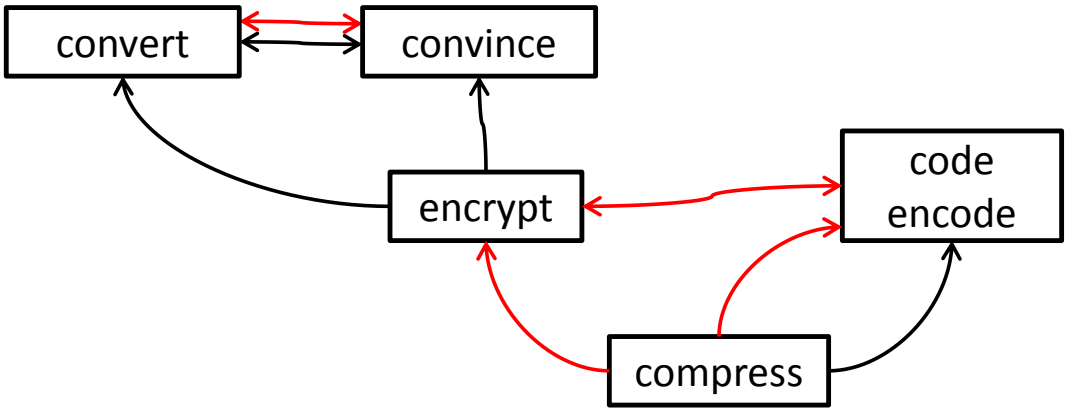

Figure 5.12: A fragment from the graphs learned by HTL-tree (black edges) and TNF (red edges).

Figure 5.12 provides another interesting example for an improvement in the graph due to the application of Tree-Node-Fix. In the initialization, HTL-tree determined that the predicate *'encrypt'* entails the predicates *'convert'* and *'convince'* rather than the predicates *'code'* and *'encode'*. This is since 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-tree inserts the wrong edge '*encrypt* ⇒ *convert*' and then it is unable to insert the correct rule '*encrypt* ⇒ *encode*' due to the FRG assumption. After HTL-tree terminates, Tree-Node-Fix 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-tree and Tree-Node-Fix are able to identify in this case the correct directionality of the rule '*compress* ⇒ *encode*'. The local score given by the classifier for *('compress','encode')* is 0.65, while the local score for *('encode','compress')* is -0.11.

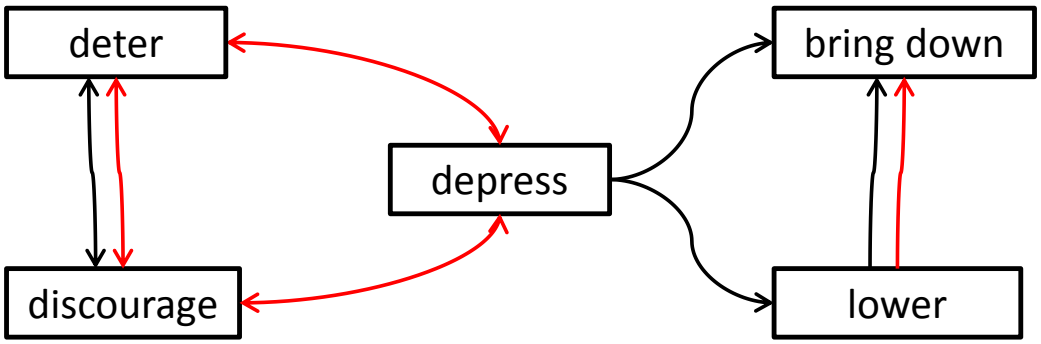

Figure 5.13: A fragment from the graphs learned by HTL-tree (black edges) and TNF (red edges).

As a last example, Figure 5.13 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-tree 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'* (since this improves the objective function), resulting in a set of edges that corresponds to the first meaning of *'depress'* mentioned above. The methods we employ in this dissertation do not allow to learn the rules for both contexts since this would result in violations of the transitivity and FRG assumptions. This once again (as discussed in Section 5.3.1.3) raises the need for algorithms that identify predicate ambiguity, learn the correct rules for various contexts, and have a mechanism for applying a rule only in its appropriate context.

Symmetry of local classifier scores In Section 3.4.4 we saw that one of the limitations of our local classifier is its inability to correctly determine the direction of entailment. As explained, distributional similarity features do not discriminate entailment directionality well, despite the fact that some of the similarity functions are directional. In this chapter we have presented two local classifiers – the first (clsf_{ds}) trained over distributional similarity features only, and the second (clsf_{all}) also employs lexicographic and string-similarity features, which contain more directional information. We would like to examine whether the features added to clsf_{all} result in a more "directional" local classifier.

To this end, we go over all pairs of predicates (i, j) in our original set of 103,315 predicates such that $s_{ij} > \lambda$. We choose $\lambda = 0.2$, which results in almost 2.5 million predicate pairs. For each pair of predicates (i, j) we compute the difference in score between s_{ij} and s_{ji}, *i.e.*, $\Delta_{ij} = |s_{ij} - s_{ji}|$. We place each Δ_{ij} in a bin at logarithmic scale, that is, we count the proportion of node pairs for which $\Delta_{ij} < 10^{-10}$, the proportion of predicate pairs for which $\Delta_{ij} < 10^{-9}$, etc. We expect that an improvement in the detection of directionality should manifest itself in an overall increase in the values of Δ_{ij}.

Figure 5.14 presents the results of this analysis. We observe that the main difference between the two classifiers is in the range $10^{-3} \leq \Delta_{ij} \leq 10^{1}$. Clearly, in this range the distribution of Δ_{ij} for clsf_{all} is skewed towards the higher values comparing to the matching distribution for clsf_{ds} This illustrates that the local classifier clsf_{all} is more directional than clsf_{ds} and hopefully better determines entailment directionality.

However, for both clsf_{all} and clsf_{ds} we witness that in about 46% of the cases Δ_{ij} is close to zero. Hence, for many pairs of predicates the local score for (i, j) and (j, i) is practically identical. This is a desirable property for equivalent predicates, however it also happens for many pairs of predicates that are not synonymous. Thus, it seems that incorporating more high-coverage directional features is still an important direction for future research.

5.5 Conclusion

In this chapter, we described the creation of a large resource of predicative entailment rules, which we publicly release for the benefit of the research community. The resource

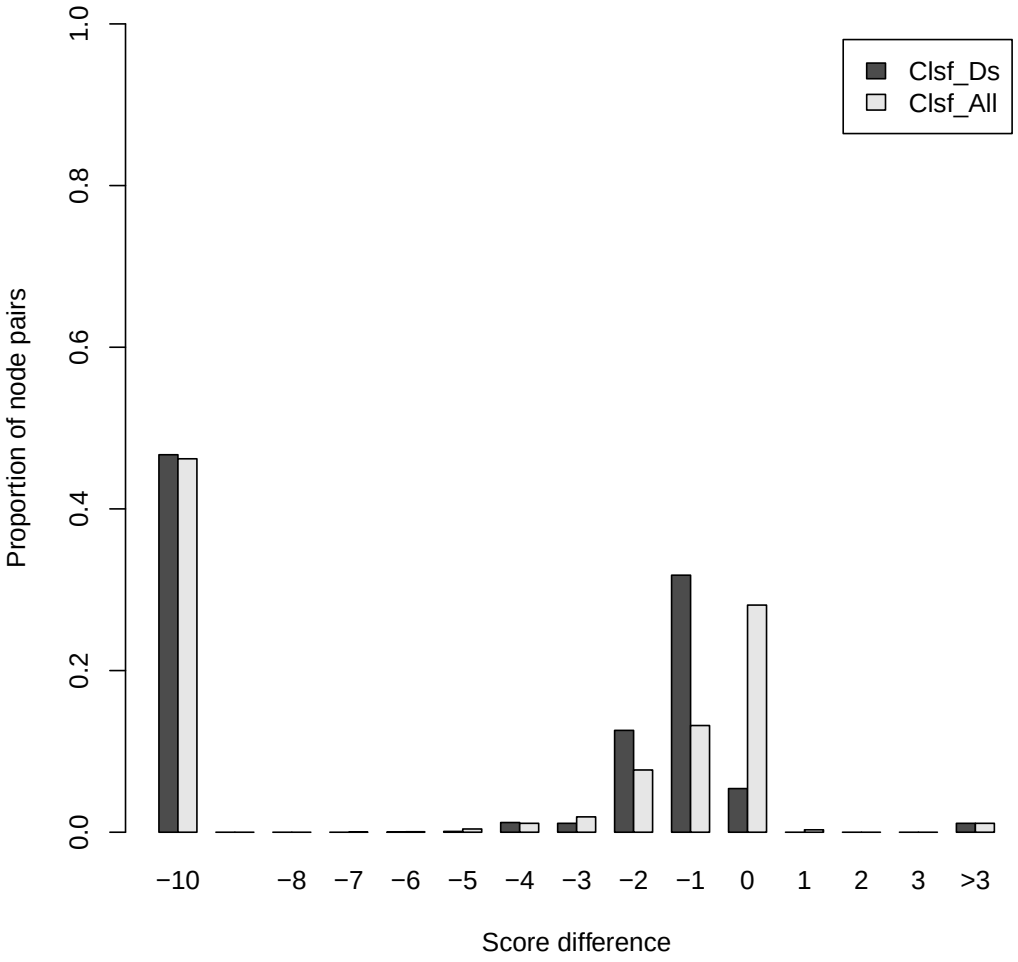

Figure 5.14: Distribution of Δij for the local classifiers clsf_{all} and clsf_{ds}. The x-axis is in logarithmic scale, *e.g.,* the column -1 shows the proportion of Δ_{ij} such that $10^{-2} \leq \Delta_{ij} \leq 10^{-1}$.

was learned over the web-based domain-general REVERB data set and it is larger than the popular DIRT resource. We demonstrated that our resource outperforms DIRT in an instance-based evaluation of learned entailment rules and that it is of comparable quality and complementary to DIRT in an RTE evaluation.

We have also shown that the global algorithms presented in previous chapters scale to graphs containing tens of thousands of nodes, and that global learning improves precision for low values of recall even when predicates are untyped and ambiguous. Nevertheless, our experiments and thorough analysis reveal that the transitivity and FRG assumptions are violated in large untyped graphs, and thus developing global algorithms that handle predicate ambiguity holds great potential for improving the quality and usability of resources of predicative entailment rules.

6

Text Exploration System

Learning entailment rules is important for various semantic inference applications, and in this dissertation we have presented various methods for learning such rules. In this chapter we suggest a novel text exploration application that is based on learning resources of predicative entailment rules. Our novel text exploration model extends the scope of state-of-the-art exploration technologies by moving from standard *concept*-based exploration to *statement*-based exploration. A user of our system can explore the result space of a query by drilling down/up from one statement to another, according to entailment relations specified by an entailment graph and an optional concept taxonomy. As a prominent use case, we apply our exploration system and illustrate its benefit on the health-care domain. To the best of our knowledge this is the first implementation of an exploration system at the statement level that is based on the textual entailment relation.

6.1 Introduction

Finding information in a large body of text is becoming increasingly more difficult. Standard search engines output a set of documents for a given query, but do not allow any exploration of the thematic structure in the retrieved information. Thus, the need for tools that allow to effectively sift through a target set of documents is becoming ever more important.

Faceted search (90, 169) supports a better understanding of a target domain, by allowing exploration of data according to multiple views or *facets*. For example, given a

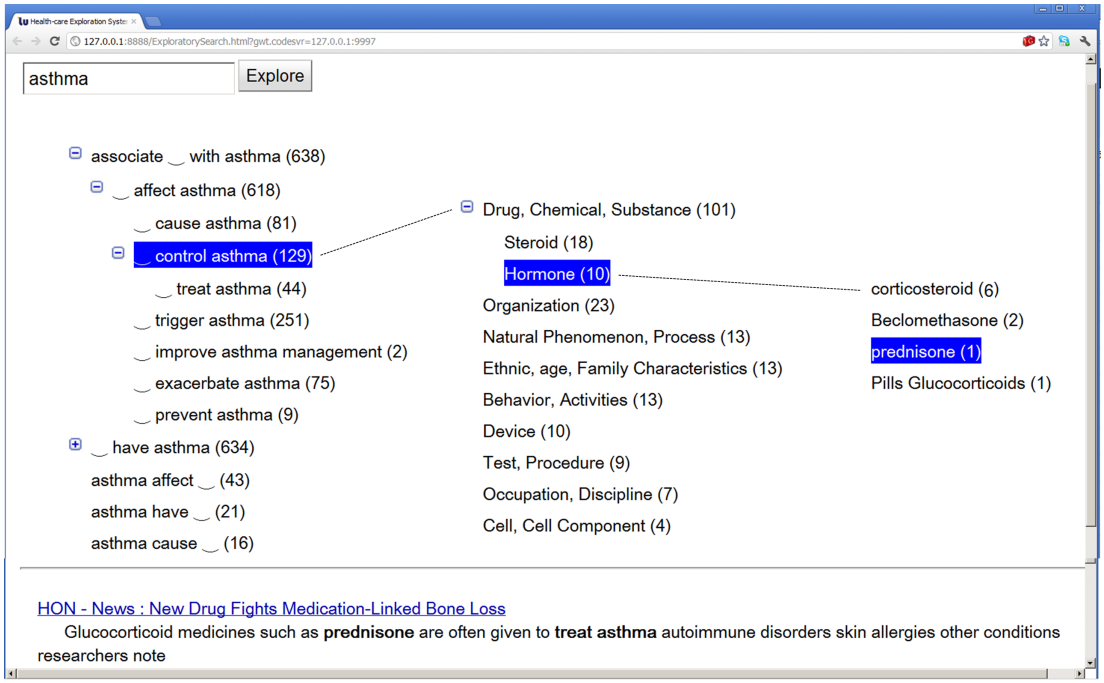

Figure 6.1: Exploring *asthma* results.

set of documents on Nobel Prize laureates we might have different facets corresponding to the laureate's nationality, the year when the prize was awarded, the field in which it was awarded, etc. However, this type of exploration is still severely limited insofar that it only allows exploration by *topic* rather than *content*. Put differently, we can only explore according to *what a document is about* rather than *what a document actually says*. For instance, the facets for the query *'asthma'* in the faceted search engine Yippy[1] include the concepts *allergy* and *children*, but do not specify what are the exact *relations* between these concepts and the query (*e.g.*, allergy *causes* asthma, and children *suffer from* asthma).

In this chapter we propose an exploration scheme that focuses on relations between concepts, which are derived from a graph describing textual entailment relations between *propositions* or *propositional templates*, *e.g.*, *'X control asthma'* (see Chapter 2). An entailment graph over propositions or propositional templates can help a user, who

[1] `http://search.yippy.com/`

is browsing documents dealing with substances that affect asthma, drill down and explore only substances that control asthma. This type of exploration can be viewed as an extension of faceted search, where the new facet concentrates on the actual statements expressed in the texts.

We implement this idea and present a novel entailment-based text exploration system, which we applied to the health-care domain. A user of this system can explore the result space of her query, by drilling down/up from one propositional template to another, according to a set of entailment relations described by an *entailment graph*. In Figure 6.1, for example, the user looks for 'things' that affect asthma. She invokes an *'asthma'* query and starts drilling down the entailment graph to *'X control asthma'* (left column). In order to examine the arguments of a selected propositional template, the user may drill down/up a concept taxonomy that classifies *terms* that occur as arguments. The user in Figure 6.1, for instance, drills down the concept taxonomy (middle column), in order to focus on *Hormones* that control asthma, such as *'prednisone'* (right column). Each drill down/up induces a subset of the documents that correspond to the aforementioned selections. The retrieved document in Figure 6.1 (bottom) is highlighted by the relevant proposition, which clearly states that prednisone is often given to treat asthma (and indeed in the entailment graph *'X treat asthma'* entails *'X control asthma'*).

Our system is built over a corpus of documents, a set of propositions extracted from the documents, an entailment graph describing entailment relations between propositional templates, and, optionally, a concept hierarchy. The system implementation for the health-care domain, for instance, is based on a web-crawled health-care corpus, propositions that were automatically extracted from the corpus, the manually-annotated health-care entailment graphs described in Section 3.4.1, and the UMLS[1] taxonomy. To the best of our knowledge this is the first implementation of an exploration system, at the proposition level, based on the textual entailment relation.

6.2 Background

Exploratory search addresses the need of users to quickly identify the important pieces of information in a target set of documents. In exploratory search, users are presented

[1] `http://www.nlm.nih.gov/research/umls/`

with a result set and a set of exploratory facets, which are proposals for refinements of the query that can lead to more focused sets of documents. Each facet corresponds to a clustering of the current result set, focused on a more specific topic than the current query. The user proceeds in the exploration of the document set by selecting specific documents (to read them) or by selecting specific facets, to refine the result set.

Early exploration technologies were based on a single hierarchical conceptual clustering of information (84), enabling the user to drill up and down the concept hierarchies. Hierarchical faceted meta-data (169), or *faceted search*, proposed more sophisticated exploration possibilities by providing multiple facets and a hierarchy per facet or dimension of the domain. These types of exploration techniques were found to be useful for effective access of information (90).

In this work, we suggest proposition-based exploration as an extension to concept-based exploration. Our intuition is that text exploration can profit greatly from representing information not only at the level of individual concepts, but also at the propositional level, where the relations that link concepts to one another are represented effectively in a hierarchical entailment graph.

6.3 Exploration Model

In this section we extend the scope of state-of-the-art exploration technologies by moving from standard concept-based exploration to proposition-based exploration, or equivalently, statement-based exploration. In our model, it is the entailment relation between propositional templates which determines the granularity of the viewed information space. We first describe the inputs to the system and then detail our proposed exploration scheme.

6.3.1 System Inputs

Corpus A collection of *documents*, which form the search space of the system.

Extracted Propositions A set of propositions, extracted from the corpus document. The propositions are usually produced by an *extraction method*, such as TextRunner (7) or ReVerb (57). In order to support the exploration process, the documents are indexed by the propositional templates and argument terms of the extracted propositions. For

example, the index for a document containing the sentence "'Corticosteroid controls symptoms of asthma"', will include the propositional template *X control symptoms of Y* and the arguments *asthma* and *corticosteroid* as keys.

Entailment graph for predicates The nodes of the entailment graph are propositional templates, where edges as usual indicate entailment relations between templates. In order to avoid circularity in the exploration process, the graph is transformed into a DAG by computing its SCC graph (see Section 4.2.2), and then for simplicity and clarity transitive reduction is applied over the SCC graph. Figure 6.2 illustrates the result of applying this procedure to a fragment of the entailment graph for *'asthma'*.

Taxonomy for arguments The optional concept taxonomy maps *terms* to one or more pre-defined concepts, arranged in a hierarchical structure. These terms may appear in the corpus as arguments of predicates. Figure 6.3, for instance, illustrates a simple medical taxonomy, composed of three concepts (medical, diseases, drugs) and four terms (cancer, asthma, aspirin, flexeril).

6.3.2 Exploration Scheme

The objective of the exploration scheme is to support querying and offer facets for result exploration, in a visual manner. The following components cover the various aspects of this objective, given the above system inputs:

Querying The user enters a search term as a query, *e.g.*, *'asthma'*. The given term induces a subgraph of the entailment graph that contains all propositional templates (graph nodes) with which this term appears as an argument in the extracted propositions (see Figure 6.2). This subgraph is represented as a DAG, as explained in Section 6.3.1, where all nodes that have no parent are defined as the *roots* of the DAG. As a starting point, only the roots of the DAG are displayed to the user. Figure 6.4 shows the five roots for the *'asthma'* query.

Exploration process The user selects one of the entailment graph nodes (*e.g.*, *'associate X with asthma'*). At each exploration step, the user can drill down to a more

specific template or drill up to a more general template, by moving along the entailment hierarchy. For example, the user in Figure 6.5, expands the root *'associate X with asthma'*, in order to drill down through *'X affect asthma'* to *'X control Asthma'*.

Selecting a propositional template (Figure 6.1, left column) displays a concept taxonomy for the arguments that correspond to the variable in the selected template (Figure 6.1, middle column). The user can explore these argument concepts by drilling up and down the concept taxonomy. For example, in Figure 6.1 the user, who selected *'X control Asthma'*, explores the arguments of this template by drilling down the taxonomy to the concept *'Hormone'*.

Selecting a concept opens a third column, which lists the terms mapped to this concept that occurred as arguments of the selected template. For example, in Figure 6.1, the user is examining the list of arguments for the template *'X control Asthma'*, which are mapped to the concept *'Hormone'*, focusing on the argument *'prednisone'*.

Document retrieval At any stage, the list of documents induced by the current selected template, concept and argument is presented to the user, where in each document snippet the relevant proposition components are highlighted. Figure 6.1 (bottom) shows such a retrieved document. The highlighted extraction in the snippet, *'prednisone treat asthma'*, entails the proposition selected during exploration, *'prednisone control asthma'*.

6.4 System Architecture

In this section we briefly describe system components, as illustrated in the block diagram (Figure 6.6).

The *search service* implements full-text and faceted search, and document indexing. The *data service* handles data (*e.g.*, documents) replication for clients. The *entailment service* handles the logic of the entailment relations (for both the entailment graph and the taxonomy).

The *index server* applies periodic indexing of new texts, and the *exploration server* serves the exploration application on querying, exploration, and data access. The *exploration application* is the front-end user application for the whole exploration process described above (Section 6.3.2).

6.5 Application to the Health-care Domain

As a prominent use case, we applied our exploration system to the health-care domain. With the advent of the internet and social media, patients now have access to new sources of medical information: consumer health articles, forums, and social networks (26). A typical health information searcher is uncertain about her exact questions and is unfamiliar with medical terminology. Exploring relevant information about a given medical issue can be essential and time-critical.

System implementation For the search service, we used SolR servlet[1], where the data service is built over FTP. The exploration application is implemented as a web application.

Input resources Our setting is directly derived from the experiment described in Section 3.4.1. We used the same health-care corpus which contains more than 2M sentences and deals with various aspects of the health care domain: answers to questions, surveys on diseases, articles on life-style, etc. We also employed the propositions extracted from that corpus and the 23 entailment graphs that were manually-annotated by medical students[2]. For the argument taxonomy we employed UMLS – a database that maps natural language phrases to over one million unique concept identifiers (CUIs) in the health-care domain. The CUIs are also mapped in UMLS to a concept taxonomy for the health-care domain.

The web application of our system is available at: `http://132.70.6.148:8080/exploration`

6.6 Conclusion and Future Work

We presented a novel exploration model, which extends the scope of state-of-the-art exploration technologies by moving from standard concept-based exploration to proposition-based exploration. Our model combines the textual entailment paradigm within the exploration process, with application to the health-care domain. According

[1] `http://lucene.apache.org/solr/`

[2] `http://www.cs.tau.ac.il/~jonatha6/homepage_files/resources/HealthcareGraphs.rar`

to our model, it is the entailment relation between propositions, encoded by the entailment graph and the taxonomy, which leads the user between more specific and more general statements throughout the search result space. We believe that employing the entailment relation between propositions, which focuses on the statements expressed in the documents, can contribute to the exploration field and improve information access.

Our current application to the health-care domain relies on a small set of entailment graphs for 23 medical concepts. As a next step, we would like to learn a larger entailment graph for the health-care domain. Another important direction for future research is to investigate methods for evaluating the exploration process (24). As noted by Qu and Furnas (138), the success of an exploratory search system does not depend simply on how many relevant documents will be retrieved for a given query, but more broadly on how well the system helps the user with the exploratory process.

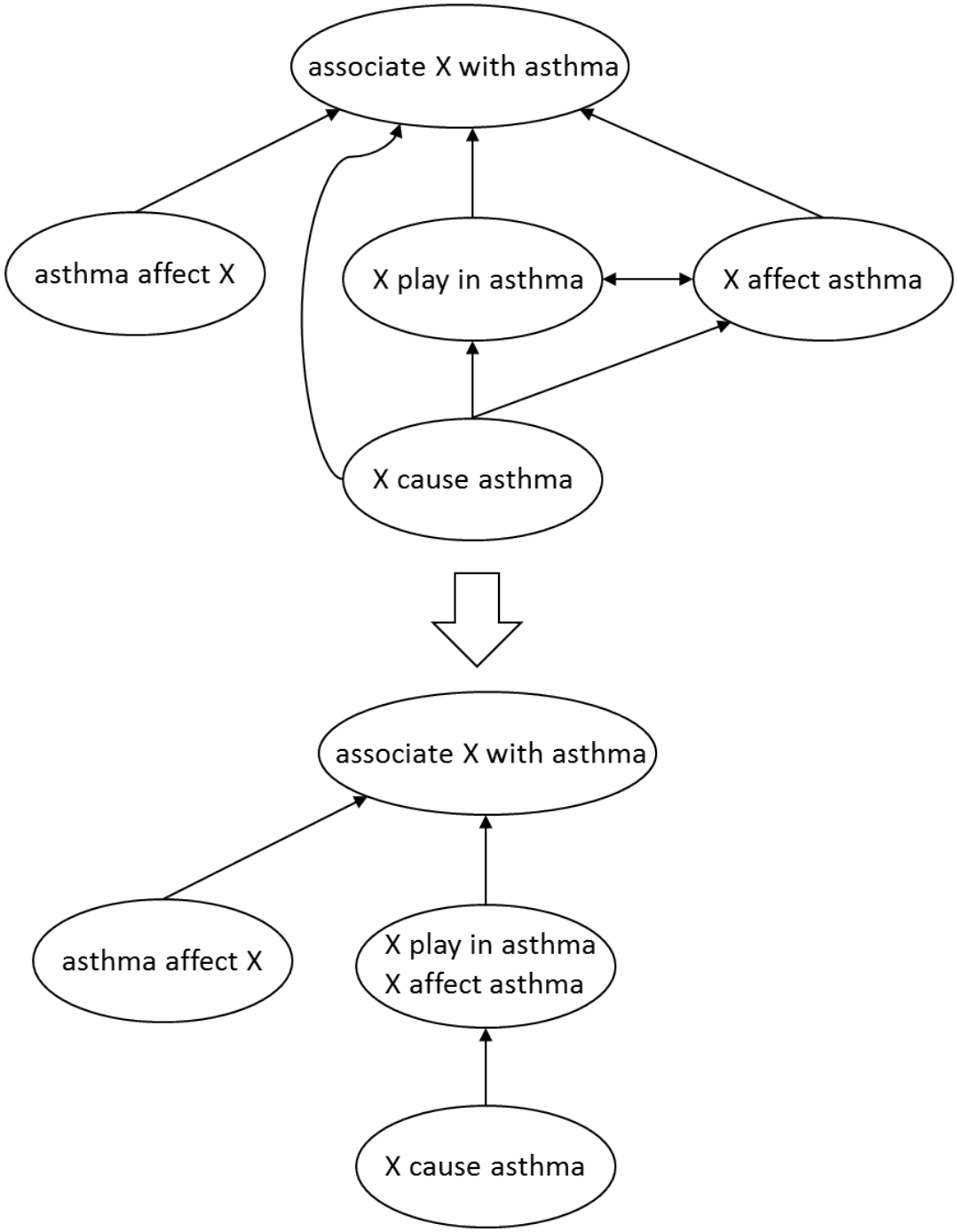

Figure 6.2: Fragment of the entailment graph for *'asthma'* (top), and its conversion to the reduced SCC graph (bottom).

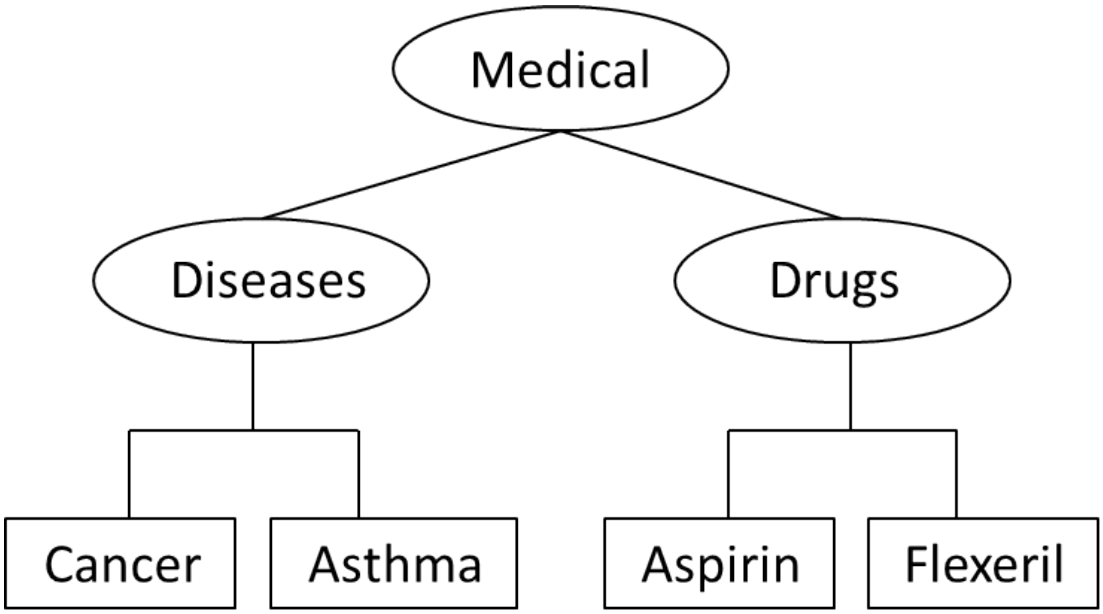

Figure 6.3: Partial medical taxonomy. Ellipses denote *concepts*, while rectangles denote *terms*.

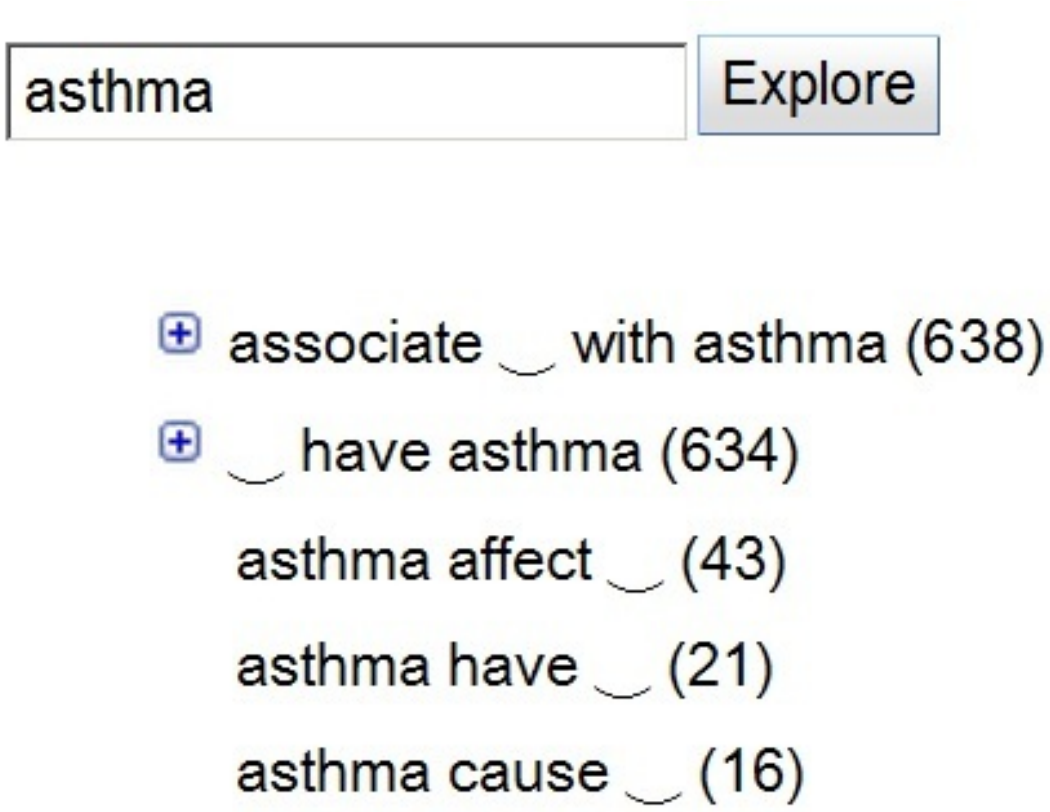

Figure 6.4: The roots of the entailment graph for the *'asthma'* query.

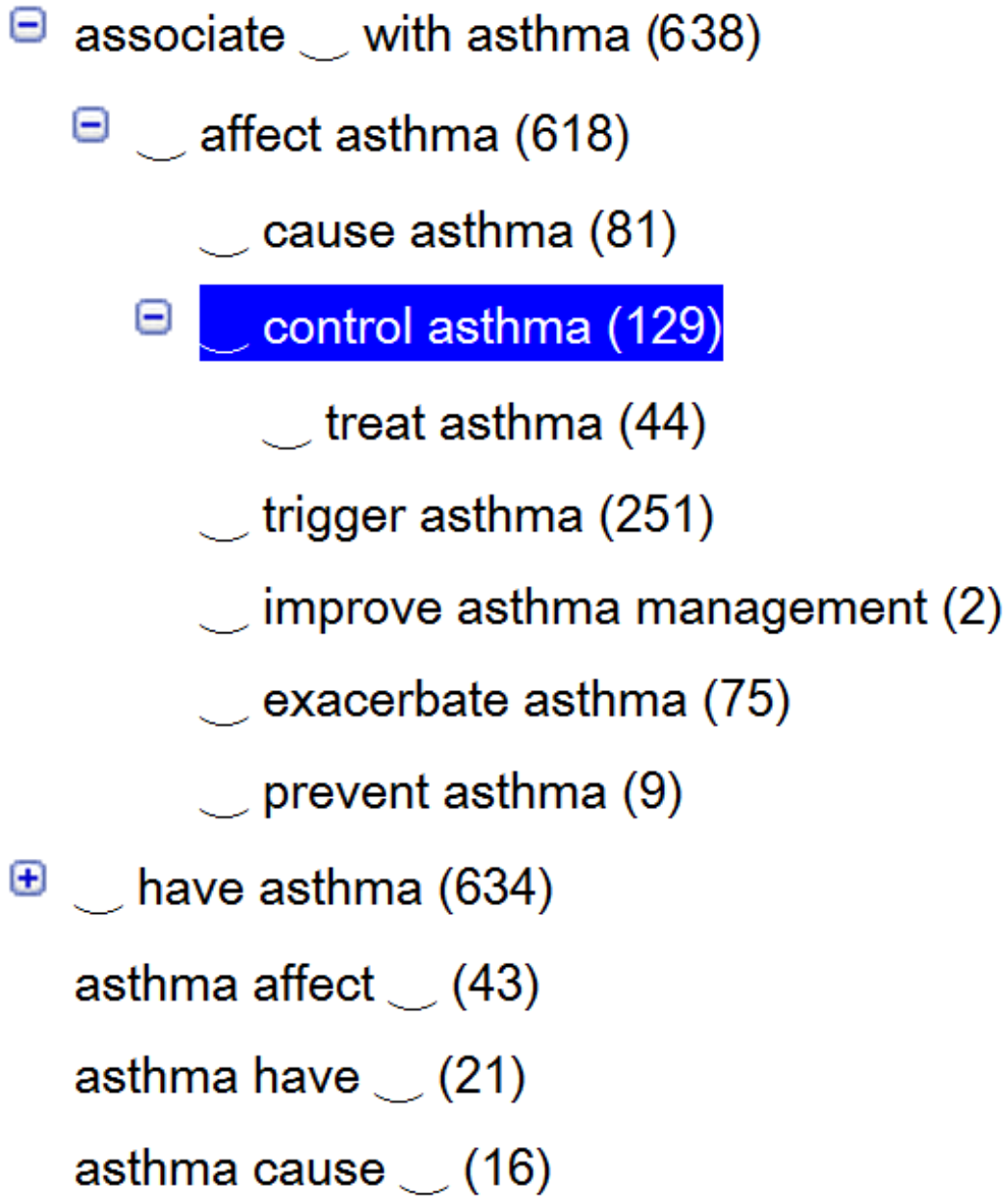

Figure 6.5: Part of the entailment graph for the *'asthma'* query, after two exploration steps. This corresponds to the left column in Figure 6.1.

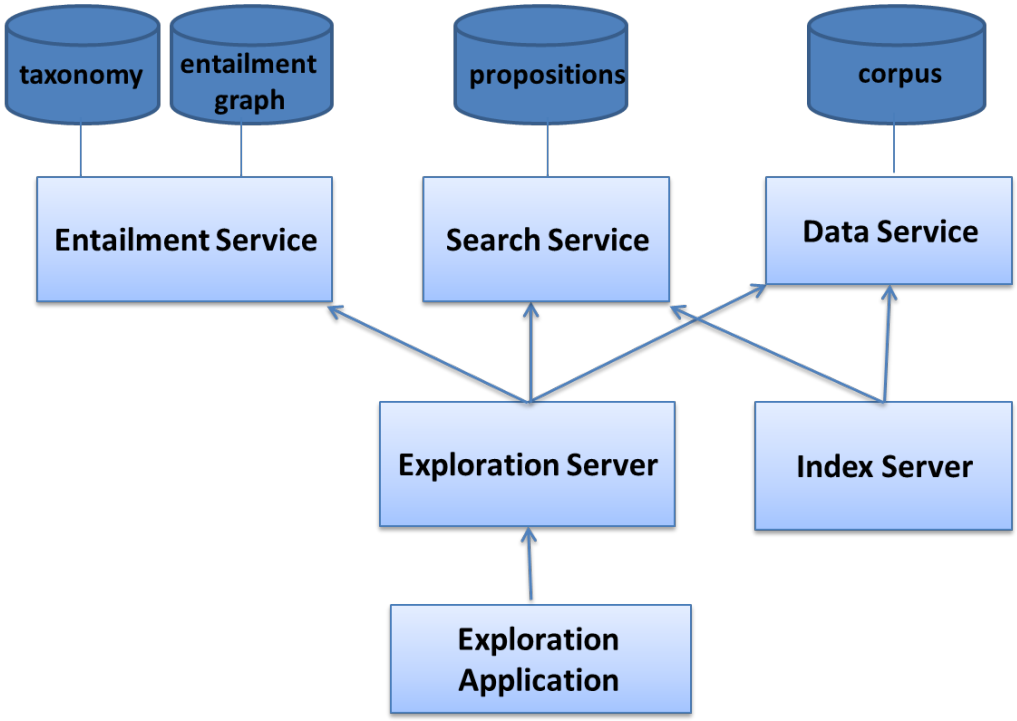

Figure 6.6: Block diagram of the exploration system.

7

Discussion

In this dissertation we attempted to tackle one of the most important challenges in the field of semantic inference, namely, learning of predicative entailment rules. We defined the *entailment graph* structure, and focused on the exploitation of its global structural properties to improve the quality of learned resources. The main structural properties that we utilized were transitivity and the fact that entailment graphs are approximately forest-reducible, and we demonstrated that using these properties we are able to improve performance comparing to local state-of-the-art algorithms and also importantly scale to graphs of tens of thousands of nodes. We release for the benefit of the research community both a local resource containing millions of entailment rules, and two more precise global resources, containing hundreds of thousands of rules. Last, we have suggested and implemented a novel text exploration application that is based on the learning of entailment graphs. Nevertheless, there is still much room for improvement in the field of learning predicative entailment rules, and in this chapter we highlight the most promising directions for future research, in our humble opinion.

7.1 Context-sensitive Rule Learning

As described in Section 2.3, modeling context has become nowadays an extremely active field in NLP. Specifically in the field of learning predicative entailment rules it is clear that entailment rules should be accompanied with some mechanism that allows to decide in which contexts they should be applied. Appropriate modeling of context is an important challenge that we have sidestepped in this dissertation. In Chapters 3

and 4 we have worked over focused entailment graphs and typed entailment graphs, in which nodes are disambiguated either by an argument or by argument types, and in Chapter 5 we ignored context altogether and witnessed that although some benefit can be gained by using global learning, it is much more limited.

One of the promising directions for modeling the context of entailment rules is through topic models, as suggested by Ritter et al. (146). In this framework the context of predicates, entailment rules, and sentences can be represented by distributions over a set of topics that is learned at the corpus-level. For example, the topic distribution for the *'Y'* variable in the predicate *'X acquire Y'* might indicate that *'Y'* is usually some commodity or company, but sometimes a skill. On the other hand, the analogous topic distribution for the rule '*X acquire Y* $\Rightarrow$ *X learn Y*' would be more concentrated on skills. This can help us during rule applications since the topic distribution for the sentence *'Facebook acquired Instagram'* would be concentrated on companies rather than skills, and so the inappropriate rule '*X acquire Y* $\Rightarrow$ *X learn Y*' would not be applied.

Although a theoretical framework for modeling context already exists, there are currently no public resources that are accompanied by a context model (including a context representation for entailment rules). Consequently, many knowledge-bases are under-utilized by semantic inference applications. This is since applying rules in wrong contexts may cause sometimes more harm than good, and thus applications simply avoid using these knowledge-bases altogether. In our opinion, developing a large-scale context-sensitive knowledge-base with an integrated mechanism for testing rule applications in context is an important direction for future research.

Moreover, as mentioned in Section 2.3, we are unaware of any work so far that has attempted to consider context during rule learning. Distributional similarity methods provide an "entailment score" for any pair of predicates by comparing their arguments, but many of the arguments are irrelevant for the context of the rule. For example, we would not want the score of the rule '*X acquire Y* $\Rightarrow$ *X learn Y*' to be influenced by the occurrences of *'X acquire Y'* in the sense of *'buying'* or *'purchasing'*. We believe that topic models, such as LDA, suggested by Ritter et al. can provide a good framework for integrating knowledge about context during rule learning.

Let u and v be a pair of predicates, and $A(u)$ and $A(v)$ be the arguments of the predicate slot X. The classical *Lin* distributional similarity measure computes similarity as

follows:

$$Lin(u,v) = \frac{\sum_{a \in A(u) \cap A(u)} [pmi(u,a) + pmi(v,a)]}{\sum_{a \in A(u) \cup A(u)} [pmi(u,a) + pmi(v,a)]} \tag{7.1}$$

Now, assume that we train a topic model, such as LDA, where latent variables represent topics such as sports or finance. Such a topic model provides us with a posterior probability $p(a|t)$ for any argument a and any topic t. We can define some representation C_{uv} for the candidate rule '$u \Rightarrow v$', for instance $C_{uv} = A(u) \cap A(v)$. Using a trained topic model it is easy to compute $p(a|C_{uv}) = \sum_t P(a|t)P(t|C_{uv})$ since we have $p(a|t)$ and $p(t|C_{uv})$ can be calculated by standard LDA inference. Thus, we can define a context-sensitive distributional similarity measure Lin^*:

$$Lin^*(u,v) = \frac{\sum_{a \in A(u) \cap A(u)} [pmi(u,a) + pmi(v,a)] \cdot p(a|C_{uv})}{\sum_{a \in A(u) \cup A(u)} [pmi(u,a) + pmi(v,a)] \cdot p(a|C_{uv})} \tag{7.2}$$

This measure puts more weight on arguments that are related to the context relevant for the candidate entailment rule, and hopefully will softly filter out arguments that are not in the "right" context.

Employing context representations is also important for handling ambiguity when applying transitivity constraints. Topic models allow us to obtain a topic distribution $p_T(C_{uv})$ over a set of topics T for a context C_{uv}. Thus, given the predicates u, v, w we can enforce a transitivity constraint only if $p_T(C_{uv})$ and $p_T(C_{vw})$ are similar, that is, the rules are related to the same topics. For example, we expect the topic distribution of $p_T(C_{\text{`}buy'\text{,`}acquire'})$ to be quite different from the topic distribution of $p_T(C_{\text{`}acquire'\text{,`}learn'})$, and hence we will not enforce a transitivity constraint in this case.

To conclude, although there has been much work on context modeling recently, marrying this work with current state-of-the-art entailment rule learning methods has not yet been achieved. In our humble opinion, modeling context during rule learning, combined with the construction of context-sensitive rule-bases that are accompanied by mechanisms to determine when rules should be applied, can drastically improve the utilization of resources in semantic inference applications, and more importantly improve the performance of these applications.

7.2 Soft Transitivity Constraints

The model presented in this dissertation performs a hard prediction for every pair of predicates (i, j) by assigning a value for every binary variable x_{ij}. However, there are

reasons to prefer a model that provides a "soft prediction", which assigns a real value $x_{ij} \in [0, 1]$ for every pair of predicates.

First, soft decisions are a more appropriate model for some subtypes of entailment. In entailment subtypes such as *troponymy* and *backward presupposition* hard decisions are suitable – *walk* is a troponym of *move* and indeed if one walks, then it is certain that one moves. Similarity, *win* presupposes *play* and thus if a sports team wins a game then it must have played it. However, there are cases of entailment that are more probabilistic in nature. For instance, if a person is *sneezing*, then he *has a cold* with probability p_1. In addition, if a person *has a cold*, then he *stays home* with probability p_2. An interesting direction is to model soft transitivity and to estimate the probability that one will stay home if one sneezes. A second reason to prefer soft decisions is that inference engines and applications that use entailment rules often require that rules be accompanied by a measure of confidence (167). It would be beneficial if we could use the global interaction between rules to improve the estimation of "confidence" that we have for each rule.

Note that above we have described two types of probabilities that can be associated with a rule (i, j) – one is related to *probabilistic entailment*, that is, if i is true what is the probability that j is also true (as in the rule '*X sneeze* $\Rightarrow$ *X has a cold*'). The second is the confidence of the learning algorithm that the rule is correct. It is important to understand that these two probabilities are different from one another and estimating each one requires different sources of information.

Allowing soft predictions in our framework requires some modifications to the model. Let's assume that the input to the model is a set of nodes V and a local entailment probability function $p : V \times V \rightarrow [0, 1]$, specifying the probability associated with each pair of predicates. We would like the model to learn a modified entailment probability function $\hat{p}$. To capture the interaction between rules the entailment probability function $\hat{p}$ will respect some form of "triangle inequality", requiring that for any nodes i, j, k, $p_{ij} \cdot p_{jk} \leq p_{ik}$ (another sensible variant might be $p_{ij} \cdot p_{jk} \approx p_{ik}$), or equivalently $\log p_{ij} + \log p_{jk} \leq \log p_{ik}$. This corresponds to the intuition that given a rule chain $i \rightarrow j \rightarrow k$, the probability p_{ik} should be the multiplication $p_{ij} \cdot p_{jk}$, if the chain rules are independent, or higher if they are positively correlated. At the extreme case this means that if $p_{ij} = p_{jk} = 1$, then $p_{ik} = 1$, which is exactly transitivity. The goal is to find a modified score function $\hat{p}$ that is as similar as possible to the input

entailment probability function p, while respecting the triangle inequality. We suggest here one possible way of formalizing this intuition:

$$\hat{p} = \underset{\bar{p}}{\operatorname{argmin}} \sum_{i \neq j} (\bar{p_{ij}} - p_{ij})^2$$
$$s.t.\ \forall i,j,k \in V\ \ \log \bar{p}_{ij} + \log \bar{p}_{jk} \leq \log \bar{p}_{ik}$$
$$\forall_{i \neq j}\ \bar{p}_{ij} \in [0,1]$$

To summarize, we believe that allowing the model to make soft predictions can increase the scope of the learned rules, and additionally allow for better use of rules in semantic inference systems.

7.3 Joint Modeling of Semantic Relations

In this dissertation entailment graphs contained a single type of edge that represented the *entailment* relation. But, there are other semantic relations that are related to entailment, and learning these related semantic relations can improve learning of entailment. For example, given the sentence *"cats, dogs, and other animals"* we can infer that *'cat'* and *'dog'* are both hyponyms of *'animal'* or in other words *co-hyponyms* or *sister-terms* in the noun taxonomy. Clearly, if *'cat'* and *'dog'* are co-hyponyms, then '*cat* $\not\Rightarrow$ *dog*' and '*dog* $\not\Rightarrow$ *cat*'. Thus, it is possible to use different sources of information to learn different semantic relations and encode the inter-dependencies between the different semantic relations in some way. This exact idea was suggested by Do and Roth (52), however they manually encoded all permissible edge configurations over very small graphs containing only 3 nodes. Next, we outline a more comprehensive methodology to perform joint learning over the semantic relations *synonymy*, *entailment*, and *co-hyponymy*.

Assume we have three classifiers that provide a probability for every pair of predicats $i \neq j$:

- p_{ij} : probability indicating whether i entails j.
- q_{ij} : probability indicating whether i and j are synonyms.
- r_{ij} : probability indicating whether i and j are co-hyponyms, that is they have a common ancestor but none of the two is an ancestor of the other.

We can define binary indicators e_{ij}, s_{ij}, c_{ij} that correspond to whether i entails j, i and j are synonyms, and i and j are co-hyponyms. Then, by assuming that different semantic relations are independent of one another it is easy to expand the linear objective function defined in Section 3.3 to include all three types of relations. Importantly, global properties of the semantic relations (such as transitivity) as well as dependencies between the different semantic relations are encoded using linear constraints:

1. Entailment is transitive:

$$\forall_{i \neq j \neq k} \; e_{ij} + e_{jk} - e_{ik} \leq 1 \tag{7.3}$$

2. Synonymy is transitive:

$$\forall_{i \neq j \neq k} \; s_{ij} + s_{jk} - s_{ik} \leq 1 \tag{7.4}$$

3. if i and j are co-hyponyms, and also j and k are co-hyponyms then either i and k are co-hyponyms, or i is an ancestor of k, or k is an ancestor of i

$$\forall_{i \neq j \neq k} \; c_{ij} + c_{jk} - c_{ik} - e_{ik} - e_{ki} \leq 1 \tag{7.5}$$

4. i and j are synonyms if and only if i entails j and j entails i:

$$\forall_{i \neq j} \; e_{ij} + e_{ji} - s_{ij} \leq 1 \tag{7.6}$$

$$\forall_{i \neq j} \; 2s_{ij} - e_{ij} - e_{ji} \leq 0 \tag{7.7}$$

5. if i and j both entail k but they don't entail one another, then they are co-hyponyms:

$$\forall_{i \neq j \neq k} \; e_{ik} + e_{jk} + (1 - e_{ij}) + (1 - e_{ji}) - c_{ij} \leq 3 \tag{7.8}$$

6. If i and j are co-hyponyms, then they don't entail one another and are not synonyms:

$$\forall_{i \neq j} \; c_{ij} - (1 - e_{ij}) - (1 - e_{ji}) \leq 0 \tag{7.9}$$

Joint modeling of different semantic relations allows us to take advantage of complementary sources of information for learning each semantic relation, and then this information can be combined using constraints. This type of modeling can help us combat one of the famous problems of distributional similarity methods, namely, that they often confuse *entailment* with *co-hyponymy*.

7.4 Learning Sub-types of Entailment

In this dissertation we treated entailment as a single relation. However, as explained in Section 1.3, entailment is composed of several subtypes such as *troponymy*, *temporal inclusion*, *backward presupposition*, etc. An interesting open question is whether learning at the level of subtypes would be more accurate than learning at the level of entailment itself. For instance, it is quite intuitive that the co-occurrence patterns of troponymy should be different than the co-occurrence patterns of temporal inclusion. In the case of distributional similarity, it is still unclear what types of entailment are exactly captured by distributional similarity methods.

Some work in this direction has been carried out already. Tremper (176) trained a multi-class classifier over various entailment subtypes, aiming to learn the *backward presupposition* relation. Recently, Weisman et al. (184) trained a classifier for verb entailment over linguistically motivated features, where different features correspond to different subtypes of entailment. A natural research direction is to combine these two ideas and train a multi-class classifier hat classifies every pair of predicates (i, j) to one of the subtypes of entailment or non-entailment (such as *antonymy* or *co-hyponymy*). Then, we can extrapolate from this classification whether '$i \Rightarrow j$'. If this results in better performance than using a 2-way *entailment* classifier, then this means that categorizing examples according to subtypes of entailment divides the feature space in such a way that makes it easier for the classifier to discriminate between the different classes.

7.5 Conclusion

This dissertation is concerned with learning of predicative entailment rules. Our main contribution is in suggesting methods for using global structural properties (especially

transitivity) to improve rule learning. Additionally, we have released a large state-of-the-art resource of entailment rules that is freely available to developers of semantic inference applications, researchers in the field of semantic relation learning, and the entire NLP community.

There is still a long way to go – in this chapter we have outlined the main directions for future research, which we believe can improve the quality and usefulness of entailment rule resources. Primarily, we believe that the subject of context-sensitive rule learning and rule application is the most important direction, and that progress and effort in this direction can help make entailment rule resources a standard and inseparable part of any semantic inference application.

References

[1] Meni Adler, Jonathan Berant, and Ido Dagan. **Entailment-based Text Exploration with Application to the Health-care Domain**. In *Proceedings of ACL*, 2012. 13

[2] Eugene Agichtein and Luis Gravano. *Snowball*: **extracting relations from large plain-text collections**. In *ACM DL*, pages 85–94, 2000. 36

[3] Alfred V. Aho, Michael R. Garey, and Jeffrey D. Ullman. **The Transitive Reduction of a Directed Graph**. *SIAM Journal on Computing*, **1**(2):131–137, 1972. 18, 106

[4] Ernst Althaus, Nikiforos Karamanis, and Alexander Koller. **Computing Locally Coherent Discourses**. In *Proceedings of ACL*, pages 399–406, 2004. 30

[5] Shachar Mirkin Eyal Shnarch Lili Kotlerman Jonathan Berant Asher Stern, Amnon Lotan and Ido Dagan. **Knowledge and Tree-Edits in Learnable Entailment Proofs**. In *Proceedings of TAC*, 2011. 159

[6] Collin. F. Baker, Charles J. Fillmore, and John B. Lowe. **The Berkeley FrameNet project**. In *Proceedings of COLING-ACL*, pages 86–90, 1998. 16

[7] Michele Banko, Michael Cafarella, Stephen Soderland, Matt Broadhead, and Oren etzioni. **Open Information Extraction from the Web**. In *Proceedings of IJCAI*, 2007. 38, 86, 98, 180

[8] Colin J. Bannard and Chris Callison-Burch. **Paraphrasing with Bilingual Parallel Corpora**. In *ACL*, 2005. 18

[9] Roy Bar-Haim, Jonathan Berant, and Ido Dagan. **A compact forest for scalable inference over entailment and paraphrase rules**. In *Proceedings of EMNLP*, 2009. 4

[10] Roy Bar-Haim, Ido Dagan, Iddo Greental, and Eyal Shnarch. **Semantic Inference at the Lexical-Syntactic Level**. In *Proceedings of AAAI*, pages 871–876, 2007. 5, 58

[11] Regina Barzilay and Noemie Elhadad. **Sentence Alignment for Monolingual Comparable Corpora**. In *Proceedings of EMNLP*, 2003. 17

[12] Regina Barzilay and Lillian Lee. **Learning to paraphrase: An unsupervised approach using multiple-sequence alignment**. In *Proceedings of HLT-NAACL*, 2003. 18

[13] Regina Barzilay and Kathleen McKeown. **Extracting Paraphrases from a Parallel Corpus**. In *ACL*, pages 50–57, 2001. 17

[14] Roni Ben Aharon, Idan Szpektor, and Ido Dagan. **Generating Entailment Rules from FrameNet**. In *Proceedings of ACL*, pages 241–246, 2010. 16, 75

[15] Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernarde Magnini. **The Fifth Pascal Recognizing Textual Entailment Challenge**. In *Proceedings of TAC-09*, pages 14–24, 2009. 56

[16] Jonathan Berant, Ido Dagan, Meni Adler, and Jacob Goldberger. **Efficient Tree-based Approximation for Entailment Graph Learning**. In *Proceedings of ACL*, 2012. 13

[17] Jonathan Berant, Ido Dagan, and Jacob Goldberger. **Global Learning of Focused Entailment Graphs**. In *Proceedings of ACL*, pages 1220–1229, 2010. 12, 157

[18] Jonathan Berant, Ido Dagan, and Jacob Goldberger. **Global Learning of Typed Entailment Rules**. In *Proceedings of ACL*, pages 610–619, 2011. 13

[19] Jonathan Berant, Ido Dagan, and Jacob Goldberger. **Learning Entailment Relations by Global Graph Structure Optimization**. *Computational Linguistics*, **38**(1):73–111, 2012. 12, 157

[20] Matthew Berland and Eugene Charniak. **Finding Parts in Very Large Corpora**. In *Proceedings of ACL*, pages 57–64, College Park, Maryland, USA, 1999. 9, 24

[21] Rahul Bhagat, Patrick Pantel, and Eduard Hovy. **LEDIR: An Unsupervised Algorithm for Learning Directionality of Inference Rules**. In *Proceedings of EMNLP-CoNLL*, pages 161–170, 2007. 9, 20, 45, 144

[22] Rahul Bhagat and Deepak Ravichandran. **Large Scale Acquisition of Paraphrases for Learning Surface Patterns**. In *Proceedings of ACL-08: HLT*, pages 674–682, Columbus, Ohio, June 2008. Association for Computational Linguistics. Available from: `http://www.aclweb.org/anthology/P/P08/P08-1077`. 19

[23] David M. Blei, Andrew Y. Ng, and Michael I. Jordan. **Latent Dirichlet Allocation**. *Journal of Machine Learning Research*, **3**:993–1022, 2003. 34

[24] Pia Borlund and Peter Ingwersen. **The development of a method for the evaluation of interactive information retrieval systems**. *Journal of Documentation*, **53**:225–250, 1997. 184

[25] Johan Bos and Katja Markert. **Recognising Textual Entailment with Logical Inference**. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 628–635, Vancouver, British Columbia, Canada, 2005. 4

[26] Maged N. Kamel Boulos and Steve Wheeler. **The emerging Web 2.0 social software: an enabling suite of sociable technologies in health and health care education**. *Health Information & Libraries Journal*, **24**(1):2–23, 2007. Available from: `http://dx.doi.org/10.1111/j.1471-1842.2007.00701.x`. 183

[27] Sergey Brin. **Extracting Patterns and Relations from the World Wide Web**. In *Proceedings of WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT98*, pages 172–183, Valencia, Spain, 1998. 9, 36

[28] Alexander Budanitsky and Graeme Hirst. **Evaluating WordNet-based Measures of Lexical Semantic Relatedness**. *Computational Linguistics*, **32**(1):13–47, 2006. 16

[29] Razvan C. Bunescu and Raymond J. Mooney. **A Shortest Path Dependency Kernel for Relation Extraction**. In *HLT/EMNLP*, 2005. 35

[30] Aljoscha Burchardt, Marco Pennacchiotti, Stefan Thater, and Manfred Pinkal. **Assessing the Impact of Frame Semantics on Textual Entailment**. *Journal of Natural Language Engineering*, **15**(4):527–550, 2009. 4

[31] Chris Callison-Burch. **Syntactic Constraints on Paraphrases Extracted from Parallel Corpora**. In *EMNLP*, pages 196–205, 2008. 18

[32] Sharon A. Caraballo. **Automatic construction of a hypernym-labeled noun hierarchy from text**. In *ACL*, 1999. 24

[33] Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R. Hruschka Jr., and Tom M. Mitchell. **Toward an Architecture for Never-Ending Language Learning**. In *Proceedings of AAAI*, 2010. 22

[34] Nathanael Chambers and Dan Jurafsky. **Unsupervised Learning of Narrative Schemas and their Participants**. In *ACL/AFNLP*, pages 602–610, 2009. 26

[35] Nathanael Chambers and Daniel Jurafsky. **Unsupervised Learning of Narrative Event Chains**. In *ACL*, pages 789–797, 2008. 26

[36] Tsz Ping Chan, Chris Callison-Burch, and Benjamin Van Durme. **Reranking Bilingually Extracted Paraphrases Using Monolingual Distributional Similarity**. In *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics*, pages 33–42, Edinburgh, UK,

July 2011. Association for Computational Linguistics. Available from: `http://www.aclweb.org/anthology/W11-2504`. 18

[37] Yin-Wen Chang and Michael Collins. **Exact Decoding of Phrase-Based Translation Models through Lagrangian Relaxation**. In *EMNLP*, pages 26–37, 2011. 31

[38] Gennaro Chierchia and Sally McConnell-Ginet. *Meaning and Grammar (2nd ed.): an Introduction to Semantics*. MIT Press, Cambridge, Massachusetts, USA, 2000. 2

[39] Timothy Chklovski and Patrick Pantel. **VERB OCEAN: Mining the web for fine-grained semantic verb relations**. In *Proceedings of EMNLP*, pages 33–40, 2004. 25, 75, 130

[40] Philipp Cimiano, Andreas Hotho, and Steffen Staab. **Comparing Conceptual, Divise and Agglomerative Clustering for Learning Taxonomies from Text**. In *ECAI*, pages 435–439, 2004. 24

[41] Peter Clark, William Murray, John Thompson, Phil Harrison, Jerry Hobbs, and Christiane Fellbaum. **On the Role of Lexical and World Knowledge in RTE3**. In *Proceedings of the Workshop on Textual Entailment and Paraphrasing*, pages 54–59, 2007. 41

[42] James Clarke and Mirella Lapata. **Global Inference for Sentence Compression: An Integer Linear Programming Approach**. *Journal of Artificial Intelligence Research*, **31**:273–381, 2008. 30

[43] William Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. **A comparison of string distance metrics for name-matching tasks**. In *Proceedings of IIWeb*, pages 73–78, 2003. 27, 75

[44] Trevor Cohn and Mirella Lapata. **Sentence compression beyond word deletion**. In *Proceedings of COLING*, 2008. 18

[45] Michael Connor and Dan Roth. **Context Sensitive Paraphrasing with a Single Unsupervised Classifier**. In *Proceedings of ECML*, pages 104–115, 2007. 46

[46] Thomas H. Cormen, Charles E. leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2002. 106, 113

[47] Bob Coyne and Owen Rambow. **LexPar: A Freely Available English Paraphrase Lexicon Automatically Extracted from FrameNet**. In *Proceedings of IEEE International Conference on Semantic Computing*, pages 53–58, 2009. 16

[48] Ido Dagan, Bill Dolan, Bernardo Magnini, and Dan Roth. **Recognizing textual entailment: Rational, evaluation and approaches**. *Natural Language Engineering*, **15**(4):1–17, 2009. 2, 3, 55, 144

[49] Ido Dagan and Oren Glickman. **Probabilistic Textual Entailment: Generic Applied Modeling of Language Variability**. In *PASCAL Workshop on Learning Methods for Text Understanding and Mining*, Grenoble, France, 2004. 2

[50] Ido Dagan and Oren Glickman. **The PASCAL Recognising Textual Entailment Challenge**. In *Proceedings of the PASCAL Challenges Workshop on Recognising Textual Entailment*, Southampton, UK, 2005. 2, 3

[51] Georgiana Dinu and Mirella Lapata. **Measuring Distributional Similarity in Context**. In *EMNLP*, pages 1162–1172, 2010. 33, 34

[52] Quang Do and Dan Roth. **Constraints Based Taxonomic Relation Classification**. In *Proceedings of EMNLP*, pages 1099–1109, 2010. 30, 41, 105, 195

[53] P. Domingos and D. Lowd. *Markov Logic: An Interface Layer for Artificial Intelligence*. Morgan & Claypool, 2009. 32

[54] Saso Dzeroski and Ivan Brakto. **Handling Noise in Inductive Logic Programming**. In *Proceedings of the International Workshop on Inductive Logic Programming*, 1992. 91

[55] Oren Etzioni. **Search needs a shake-up**. *Nature*, **476**:25–26, 2011. iv

[56] Oren Etzioni, Michael J. Cafarella, Doug Downey, Stanley Kok, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S.

Weld, and Alexander Yates. **Web-scale information extraction in knowitall: (preliminary results)**. In *WWW*, pages 100–110, 2004. 37

[57] Anthony Fader, Stephen Soderland, and Oren Etzioni. **Identifying Relations for Open Information Extraction**. In *Proceedings of Empirical Methods in Natural Language Processing*, 2011. 38, 103, 124, 125, 180

[58] Christiane Fellbaum. **A Semantic Network of English: The Mother of All WordNets**. *Natural Language Engineering*, **32**:209–220, 1998. 42

[59] Christiane Fellbaum, editor. *WordNet: An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press, 1998. 8, 9, 16, 130

[60] Jenny R. Finkel and Christopher D. Manning. **Enforcing Transitivity in Coreference Resolution**. In *Proceedings of ACL-08: HLT, Short Papers*, pages 45–48, 2008. 28, 30, 47

[61] Yoav Freund and Robert E. Schapire. **A Decision-Theoretic Generalization of on-Line Learning and an Application to Boosting**. In *Proceedings of the Second European Conference on Computational Learning Theory (EuroCOLT '95)*, 1995. 147

[62] Atsushi Fujita, Kentaro Furihata, Kentaro Inui, Yuji Matsumoto, and Koichi Takeuchi. **Paraphrasing of Japanese light-verb constructions based on lexical conceptual structure**. In *Proceedings of ACL Workshop on Multiword Expressions: Integrating Processing*, 2004. 16

[63] Katrin Fundel, Robert Küffner, and Ralf Zimmer. **RelEx - Relation extraction using dependency parse trees**. *Bioinformatics*, **23**(3):365–371, 2007. 24

[64] Juri Ganitkevitch, Chris Callison-Burch, Courtney Napoles, and Benjamin Van Durme. **Learning Sentential Paraphrases from Bilingual Parallel Corpora for Text-to-Text Generation**. In *EMNLP*, pages 1168–1179, 2011. 18

[65] Juri Ganitkevitch, Benjamin Van Durme, and Chris Callison-Burch. **Monolingual Distributional Similarity for Text-to-Text Generation**. In *Proceedings of *SEM*, 2012. 18

[66] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979. 107

[67] Maayan Geffet and Ido Dagan. **The Distributional Inclusion Hypotheses and Lexical Entailment**. In *Proceedings of ACL*, 2005. 20

[68] Roxana Girju, Adriana Badulescu, and Dan I. Moldovan. **Automatic Discovery of Part-Whole Relations**. *Computational Linguistics*, **32**(1):83–135, 2006. 24

[69] Claudio Giuliano, Alberto Lavelli, and Lorenza Romano. **Exploiting Shallow Linguistic Information for Relation Extraction from Biomedical Literature**. In *EACL*, 2006. 35

[70] Yoav Goldberg and Michael Elhadad. **An Efficient Algorithm for Easy-First Non-Directional Dependency Parsing**. In *HLT-NAACL*, pages 742–750, 2010. 141

[71] Geoffrey J. Gordon, Sue Ann Hong, and Miroslav Dudík. **First-Order Mixed Integer Linear Programming**. In *UAI*, pages 213–222, 2009. 32

[72] Nizar Habash and Bonnie Dorr. **A Categorial Variation Database For English**. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology*, NAACL '03, pages 17–23, Edmonton, Canada, 2003. 16, 130

[73] Nizar Habash and Bonnie Dorr. **A Categorial Variation Database for English**. In *Proceedings of the NAACL*, pages 17–23, 2003. 75

[74] Masato Hagiwara, Yasuhiro Ogawa, and Katsuhiko Toyama. **Supervised Synonym Acquisition Using Distributional Features and Syntactic Patterns**. *Journal of Natural Language Processing*, **16**(2):59–83, 2009. 27

[75] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. **The WEKA Data Mining Software: An Update**. *SIGKDD Explorations*, **11**(1):10–18, 2009. 76

[76] S. Harabagiu and A. Hickl. **Methods for Using Textual Entailment in Open-domain Question Answering**. In *Proceedings of ACL*, 2006. 4

[77] Sanda Harabagiu, Andrew Hickl, and Finley Lacatusu. **Satisfying Information Needs with Multi-document Summaries**. *Inf. Process. Manage.*, **43**:1619–1642, November 2007. 4

[78] Stefan Harmeling. **Inferring Textual Entailment with a Probabilistically Sound Calculus**. *Journal of Natural Language Engineering*, pages 459–477, 2009. 4

[79] Stefan Harmeling. **Inferring textual entailment with a probabilistically sound calculus**. *Natural Language Engineering*, **15**(4):459–477, 2009. 58

[80] Zellig Harris. **Distributional structure**. *Word*, **10(23)**:146–162, 1954. 17

[81] Marti Hearst. **Automatic Acquisition of Hyponyms from Large Text Corpora**. In *Proceedings of COLING*, 1992. 9, 24, 86

[82] Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S. Weld. **Knowledge-Based Weak Supervision for Information Extraction of Overlapping Relations**. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 541–550, Portland, Oregon, USA, June 2011. Association for Computational Linguistics. Available from: `http://www.aclweb.org/anthology/P11-1055`. 38

[83] Raphael Hoffmann, Congle Zhang, and Daniel S. Weld. **Learning 5000 Relational Extractors**. In *ACL*, pages 286–295, 2010. 38

[84] Thomas Hofmann. **The cluster-abstraction model: Unsupervised learning of topic hierarchies from text data**. In *Proceedings of IJCAI*, 1999. 180

[85] John N. Hooker and María Auxilio Osorio Lama. **Mixed Logical-linear Programming**. *Discrete Applied Mathematics*, **96-97**:395–442, 1999. 32

[86] Thad Hughes and Daniel Ramage. **Lexical Semantic Relatedness with Random Graph Walks.** In *Proceedings of EMNLP-CoNLL*, pages 581–589, Prague, Czech Republic, 2007. Proceedings of ACL. 9, 16

[87] Ali Ibrahim, Boris Katz, and Jimmy Lin. **Extracting structural paraphrases from aligned monolingual corpora**. In *Proceedings of IWP*, 2003. 17

[88] Adrian Iftene and Alexandra Balahur-Dobrescu. **Hypothesis Transformation and Semantic Variability Rules Used in Recognizing Textual Entailment**. In *Proceedings of the ACL-PASCAL Workshop on Textual Entailment and Paraphrasing*, pages 125–130, Prague, Czech Republic, June 2007. 4

[89] Thorsten Joachims. **A Support Vector Method for Multivariate Performance Measures**. In *Proceedings of ICML*, pages 377–384, 2005. 53, 76, 98, 131

[90] Mika Kaki. **Findex: Search Results Categories Help Users when Document Ranking Fails**. In *Proceedings of CHI*, 2005. 177, 180

[91] Nanda Kambhatla. **Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations**. In *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions*, ACLdemo '04, Stroudsburg, PA, USA, 2004. Association for Computational Linguistics. Available from: `http://dx.doi.org/10.3115/1219044.1219066`. 35

[92] Paul Kingsbury, Martha Palmer, and Mitch Marcus. **Adding semantic annotation to the Penn TreeBank**. In *Proceedings of HLT*, pages 252–256, 2002. 17

[93] Karin. Kipper, Hoa T. Dang, and Martha Palmer. **Class-based construction of a verb lexicon**. In *Proceedings of AAAI*, pages 691–696, 2000. 17, 98

[94] Stanley Kok and Chris Brockett. **Hitting the right paraphrases in good time**. In *Proceedings of the GEMS 2011 Workshop on GEometrical Models of Natural Language Semantics*. Proceedings of NAACL, 2010. 18

[95] Terry Koo, Alexander M. Rush, Michael Collins, Tommi Jaakkola, and David Sontag. **Dual Decomposition for Parsing with Non-Projective Head Automata**. In *EMNLP*, pages 1288–1298, 2010. 31

[96] Lili Kotlerman, Ido Dagan, Idan Szpektor, and Maayan Zhitomirsky-Geffet. **Directional Distributional Similarity for Lexical Inference**. *Natural Language Engineering*, **16**:359–389, 2010. 21, 75

[97] Zornitsa Kozareva, Ellen Riloff, and Eduard H. Hovy. **Semantic Class Learning from the Web with Hyponym Pattern Linkage Graphs**. In *ACL*, pages 1048–1056, 2008. 24, 37

[98] Daniel D. Lee and H. Sebastian Seung. **Algorithms for Non-negative Matrix Factorization**. In *NIPS*, pages 556–562, 2000. 34

[99] Douglas B. Lenat. **CYC: A Large-Scale Investment in Knowledge Infrastructure**. *Communications of the ACM*, **38**(11):32–38, 1995. 9

[100] Beth Levin. *English verb classes and alternations : a preliminary investigation.* 1993. 17

[101] Dekang Lin. **Dependency-based evaluation of MINIPAR**. In *Proceedings of the Workshop on Evaluation of Parsing Systems at LREC 1998*, Granada, Spain, 1998. 5

[102] Dekang Lin. **Automatic Retrieval and Clustering of Similar Words**. In *Proceedings of COLING-ACL*, pages 768–774, 1998a. 9, 19, 20

[103] Dekang Lin. **Dependency-based evaluation of Minipar**. In *Proceedings of the Workshop on Evaluation of Parsing Systems at LREC*, pages 317–329, 1998b. 43, 53

[104] Dekang Lin and Patrick Pantel. **Discovery of inference rules for question answering**. *Natural Language Engineering*, **7**(4):343–360, 2001. 7, 9, 19, 43, 45, 53, 99, 123, 130, 160

[105] Krister Linden and Jussi Piitulainen. **Discovering Synonyms and Other Related Words**. In *COLING 2004 CompuTerm 2004: 3rd International Workshop on Computational Terminology*, pages 63–70, Geneva, Switzerland, 2004. 9

[106] Xiao Ling and Daniel S. Weld. **Temporal Information Extraction**. In *Proceedings of AAAI*, 2010. 28, 32

[107] Peter LoBue and Alexander Yates. **Types of Common-Sense Knowledge Needed for Recognizing Textual Entailment**. In *Proccedings of ACL (Short Papers)*, pages 329–334, 2011. 6

[108] Peter LoBue and Alexander Yates. **Types of common-sense knowledge needed for recognizing textual entailment**. In *Proceedings of ACL-HLT*, 2011. 144

[109] Bill MacCartney, Michel Galley, and Christopher D. Manning. **A Phrase-based Alignment Model for Natural Language Inference**. In *Proceedings of EMNLP*, pages 802–811, Honolulu, Hawaii, 2008. 4

[110] Catherine Macleod, Ralph Grishman, Adam Meyers, Leslie Barrett, and Ruth Reeves. **NOMLEX: A Lexicon of Nominalizations**. In *In Proceedings of Euralex*, pages 187–193, 1998. 16, 75

[111] Nitin Madnani, Necip Fazil Ayan, Philip Resnik, and Bonnie Dorr. **Using paraphrases for parameter tuning in statistical machine translation**. In *Proceedings of WMT*, 2007. 18

[112] Nitin Madnani and Bonnie J. Dorr. **Generating Phrasal and Sentential Paraphrases: A Survey of Data-driven Methods**. *Computational Linguistics*, **36**, 2010. 3, 17

[113] Andre Martins, Noah Smith, and Eric Xing. **Concise Integer Linear Programming Formulations for Dependency Parsing**. In *Proceedings of ACL*, pages 342–350, 2009. 30, 31, 105

[114] Mausam, Michael Schmitz, Stephen Soderland, Robert Bart, and Oren Etzioni. **Open Language Learning for Information Extraction**. In *EMNLP-CoNLL*, pages 523–534, 2012. 38

[115] Eric McCreath and Arun Sharma. **ILP with noise and fixed example size: a Bayesian approach**. In *Proceedings of the Fifteenth international joint conference on artificial intelligence - Volume 2*, 1997. 91

[116] Tara McIntosh and James R. Curran. **Challenges for automatically extracting molecular interactions from full-text articles**. *BMC Bioinformatics*, **10**:311, 2009. 20

[117] Yashar Mehdad, Matteo Negri, and Marcello Federico. **Towards Cross-Lingual Textual Entailment**. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics (ACL)*, 2010. 144, 147

[118] Adam Meyers, Ruth Reeves, Catherine Macleod, Rachel Szekeley, Veronika Zielinska, and Brian Young. **The Cross-Breeding of Dictionaries**. In *Proceedings of LREC*, 2004. 16

[119] S. Mirkin, L. Specia, N. Cancedda, I. Dagan, M. Dymetman, and I. Szpektor. **Source-Language Entailment Modeling for Translating Unknown Terms**. In *Proceedings of ACL*, 2009. 4, 8

[120] Shachar Mirkin, Ido Dagan, and Maayan Gefet. **Integrating Pattern-based and Distributional Similarity methods for Lexical Entailment Acquisition**. In *Proceedings of COLING-ACL*, pages 579–586, 2006. 27, 44

[121] Shachar Mirkin, Ido Dagan, Lili Kotlerman, and Idan Szpektor. **Classification-based Contextual Preferences**. In *Proceedings of the 2011 Workshop on Applied Textual Inference (TextInfer)*, Edinburgh, Scotland, UK, August 2011. 33

[122] Stephen Muggleton. **Inverse Entailment and Progol**. *New Generation Comput.*, **13**(3&4):245–286, 1995. 22

[123] Matteo Negri, Luisa Bentivogli, Yashar Mehdad, Danilo Giampiccolo, and Alessandro Marchetti. **Divide and Conquer: Crowdsourcing the Creation of Cross-Lingual Textual Entailment Corpora**. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11)*, 2011. 144, 147

[124] Matteo Negri, Milen Kouylekov, and Bernardo Magnini. **Detecting Expected Answer Relations through Textual Entailment**. In *Proceedings of the Conference on Intelligent Text Processing and Computational Linguistics (CICLing)*, pages 532–543, 2008. 4

[125] Vladimir Nikulin. **Classification of Imbalanced Data with Random sets and Mean-Variance Filtering**. *IJDWM*, **4**(2):63–78, 2008. 45

[126] Diarmuid Ó Séaghdha. **Latent Variable Models of Selectional Preference**. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 435–444, Uppsala, Sweden, July 2010. Association for Computational Linguistics. Available from: `http://www.aclweb.org/anthology/P10-1045`. 33, 34

[127] Sebastian Padó, Daniel Cer, Michel Galley, Daniel Jurafsky, and Christopher D. Manning. **Measuring Machine Translation Quality as Semantic Equivalence: A Metric based on Entailment Features**. *Machine Translation*, **23**(2–3):181–193, 2009. 4

[128] Sebastian Padó, Marie-Catherine de Marneffe, Bill MacCartney, Anna N. Rafferty, Eric Yeh, and Christopher D. Manning. **Deciding Entailment and Contradiction with Stochastic and Edit Distance-based Alignment**. In *Proceedings of the Text Analysis Conference (TAC)*, Gaithersburg, Maryland, USA, 2008. 4

[129] Bo Pang, Kevin Knight, and Daniel Marcu. **syntax-based alignment of multiple translations: Extracting paraphrases and generating new sentences**. In *Proceedings of HLT-NAACL*, 2003. 17

[130] Patrick Pantel, Rahul Bhagat, Bonaventura Coppola, Timothy Chklovski, and Eduard H. Hovy. **ISP: Learning Inferential Selectional Preferences**. In *HLT-NAACL*, pages 564–571, 2007. 33

[131] Patrick Pantel and Marco Pennacchiotti. **Espresso: Leveraging Generic Patterns for Automatically Harvesting Semantic Relations**. In *ACL*, 2006. 37

[132] Marius Pasca and Péter Dienes. **Aligning Needles in a Haystack: Paraphrase Acquisition Across the Web**. In *Prceedings of IJCNLP*, pages 119–130, 2005. 19

[133] Judea Pearl. *Probabilistic reasoning in intelligent systems - networks of plausible inference*. Morgan Kaufmann series in representation and reasoning. Morgan Kaufmann, 1989. 32

[134] Viktor Pekar. **Discovery of event entailment knowledge from text corpora**. *Computer Speech & Language*, **22**(1):1–16, 2008. 26

[135] Marco Pennacchiotti and Patrick Pantel. **Entity Extraction via Ensemble Semantics**. In *EMNLP*, pages 238–247, 2009. 27

[136] Hoifung Poon and Pedro Domingos. **Unsupervised Semantic Parsing**. In *Proceedings of EMNLP*, 2009. 28, 32

[137] Hoifung Poon and Pedro Domingos. **Unsupervised Ontology Induction from Text**. In *Proceedings of ACL*, 2010. 28, 32

[138] Yan Qu and George W. Furnas. **Model-driven formative evaluation of exploratory search: A study under a sensemaking framework**. *Inf. Process. Manage.*, **44**(2):534–555, 2008. 184

[139] J. Ross Quinlan. **Learning Logical Definitions from Relations**. *Machine Learning*, **5**:239–266, 1990. 22

[140] Chris Quirk, Chris Brockett, and William Dolan. **Monolingual machine translation for paraphrase generation**. In *Proceedings of EMNLP*, 2004. 17

[141] Rajat Raina, Andrew Ng, and Christopher Manning. **Robust textual inference via learning and abductive reasoning**. In *Proceedings of AAAI*, pages 1099–1105, 2005. 58

[142] Deepak Ravichandran and Eduard Hovy. **Learning Surface Text Patterns for a Question Answering System**. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, 2002. 9

[143] Joseph Reisinger and Raymond J. Mooney. **Multi-Prototype Vector-Space Models of Word Meaning**. In *HLT-NAACL*, pages 109–117, 2010. 33

[144] Sebastian Riedel and James Clarke. **Incremental Integer Linear Programming for Non-projective Dependency Parsing**. In *Proceedings of EMNLP*, pages 129–137, 2006. 30, 87, 97

[145] Sebastian Riedel and Andrew McCallum. **Fast and Robust Joint Models for Biomedical Event Extraction**. In *EMNLP*, pages 1–12, 2011. 31

[146] Alan Ritter, Mausam, and Oren Etzioni. **A Latent Dirichlet Allocation Method for Selectional Preferences**. In *ACL*, pages 424–434, 2010. 33, 38, 192

[147] L. Romano, M. Kouylekov, I. Szpektor, I. Dagan, and A. Lavelli. **Investigating a Generic Paraphrase-based Approach for Relation Extraction**. In *Proceedings of EACL*, 2006. 4, 37

[148] Dan Roth and Wen-tau Yih. **A Linear Programming Formulation for Global Inference in Natural Language Tasks**. In *Proceedings of CoNLL*, pages 1–8, 2004. 30, 32

[149] Alexander M. Rush and Michael Collins. **Exact Decoding of Syntactic Translation Models through Lagrangian Relaxation**. In *ACL*, pages 72–82, 2011. 31

[150] Alexander M. Rush, David Sontag, Michael Collins, and Tommi Jaakkola. **On Dual Decomposition and Linear Programming Relaxations for Natural Language Processing**. In *EMNLP*, pages 1–11, 2010. 31

[151] Mark Sammons, V. G. Vinod Vydiswaran, and Dan Roth. **"Ask Not What Textual Entailment Can Do for You..."**. In *Proceedings of ACL*, 2010. 6

[152] Mark Sammons, V. G. Vinod Vydiswaran, and Dan Roth. **"Ask not what textual entailment can do for you..."**. In *Proceedings of ACL*, 2010. 144

[153] Stefan Schoenmackers, Jesse Davis, Oren Etzioni, and Daniel S. Weld. **Learning First-Order Horn Clauses from Web Text**. In *Proceedings of EMNLP*, pages 1088–1098, 2010. 7, 9, 21, 38, 85, 86, 91, 98, 99, 130

[154] Stefan Schoenmackers, Oren Etzioni, and Daniel S. Weld. **Scaling Textual Inference to the Web**. In *EMNLP*, pages 79–88, 2008. 21

[155] Diarmuid Ó Séaghdha and Anna Korhonen. **Probabilistic models of similarity in syntactic context**. In *EMNLP*, pages 1047–1057, 2011. 33

[156] Satoshi Sekine. **Automatic Paraphrase Discovery Based on Context and Keywords between NE Pairs**. In *Proceedings of IWP*, pages 80–87, 2005. 7, 9, 144

[157] Siwei Shen, Dragomir R. Radev, Agam Patel, and Gunes Erkan. **Adding syntax to dynamic programming for aligning comparable texts for the generation of paraphrases**. In *Proceedings of ACL-COLING*, 2006. 18

[158] Yusuke Shinyama and Satoshi Sekine. **Preemptive Information Extraction using Unrestricted Relation Discovery**. In *Proceedings of the Human Language Technology Conference of the NAACL, Main Conference*, 2006. 9

[159] Yusuke Shinyama, Satoshi Sekine, and Kiyoshi Sudo. **Automatic paraphrase acquisition from news articles**. In *Proceedings of HLT*, 2002. 18, 144

[160] Eyal Shnarch, Libby Barak, and Ido Dagan. **Extracting Lexical Reference Rules from Wikipedia**. In *Proceedings of ACL*, 2009. 16

[161] Sideny Siegel and N. John Castellan. *Non-parametric Statistics for the Behavioral Sciences*. McGraw-Hill, New-York, 1988. 56

[162] Noah Smith and Jason Eisner. **Contrastive Estimation: Training Log-Linear Models on Unlabeled Data**. In *Proceedings of ACL*, pages 354–362, 2005. 44

[163] Noah A. Smith. *Linguistic Structure Prediction*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool Publishers, 2011. 31

[164] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. **Learning syntactic patterns for automatic hypernym discovery**. In *Proceedings of NIPS*, pages 1297–1304, 2004. 24, 29, 44

[165] Rion Snow, Daniel Jurafsky, and Andrew Y. Ng. **Semantic Taxonomy Induction from Heterogenous Evidence**. In *Proceedings of ACL*, pages 801–808, 2006. 28, 29, 50, 51, 136, 137

[166] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y. Ng. **Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks**. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '08)*, 2008. 144

[167] Asher Stern and Ido Dagan. **A Conidence Model for Syntactically-Motivated Entailment Proofs**. In *Proceedings of RANLP*, 2011. 4, 159, 194

[168] Asher Stern, Roni Stern, Ido Dagan, and Ariel Felner. **Efficient Search for Transformation-based Inference**. In *ACL (1)*, pages 283–291, 2012. 159

[169] Emilia Stoica, Marti Hearst, and Megan Richardson. **Automating Creation of Hierarchical Faceted Metadata Structures**. In *Proceedings of NAACL-HLT*, pages 244–251, 2007. 81, 177, 180

[170] Mihai Surdeanu, Julie Tibshirani, Ramesh Nallapati, and Christopher D. Manning. **Multi-instance Multi-label Learning for Relation Extraction**. In *EMNLP-CoNLL*, pages 455–465, 2012. 38

[171] Idan Szpektor and Ido Dagan. **Learning Canonical Forms of Entailment Rules**. In *Proceedings of RANLP*, pages 1–8, 2007. 43, 144, 147

[172] Idan Szpektor and Ido Dagan. **Learning Entailment Rules for Unary Templates**. In *Proceedings of COLING*, pages 849–856, 2008. 7, 9, 21, 45, 53, 99, 130

[173] Idan Szpektor and Ido Dagan. **Augmenting WordNet-based Inference with Argument Mapping**. In *Proceedings of TextInfer*, pages 27–35, 2009. 16, 27, 28

[174] Idan Szpektor, Ido Dagan, Roy Bar-Haim, and Jacob Goldberger. **Contextual Preferences**. In *ACL*, pages 683–691, 2008. 33

[175] Idan Szpektor, Hristo Tanev, Ido Dagan, and Bonaventura Coppola. **Scaling Web-based Acquisition of Entailment Relations**. In *Proceedings of EMNLP*, pages 41–48, 2004. 9, 20, 22, 37, 45, 130

[176] Galina Tremper. **Weakly Supervised Learning of Presupposition Relations between Verbs**. In *Proceedings of ACL (Student Research Workshop)*, pages 97–102, 2010. 9, 25, 197

[177] Alan M. Turing. **Computing machinery and intelligence**. *Mind*, **59**:433–460, 1950. 1

[178] Tim Van de Cruys, Thierry Poibeau, and Anna Korhonen. **Latent Vector Weighting for Word Meaning in Context**. In *EMNLP*, pages 1012–1022, 2011. 33, 34

[179] Jason Van Hulse, Taghi Khoshgoftaar, and Amri Napolitano. **Experimental perspectives on learning from imbalanced data**. In *Proceedings of ICML*, pages 935–942, 2007. 45

[180] Peter Wallis. **Information retrieval based on paraphrase**. In *Proceedings of PACLING*, 1993. 16

[181] Aobo Wang, Cong Duy Vu Hoang, and Min-Yen Kan. **Perspectives on Crowdsourcing Annotations for Natural Language Processing**. *Journal of Language Resources and Evaluation)*, 2012. 146

[182] Rui Wang and Chris Callison-Burch. **Cheap facts and counter-facts**. In *Proceedings of the NAACL HLT 2010 Workshop on Creating Speech and Language Data with Amazon's Mechanical Turk*, 2010. 144

[183] Julie Weeds and David Weir. **A General Framework for Distributional Similarity**. In *Proceedings of EMNLP*, pages 81–88, 2003. 21, 91, 130

[184] Hila Weisman, Jonathan Berant, Idan Szpektor, and Ido Dagan. **Learning Verb Entailment from Diverse Linguistic Evidence**. In *Proceedings of EMNLP*, 2012. 26, 197

[185] Dominic Widdows and Scott Cederberg. **Monolingual and Bilingual Concept Visualization from Corpora**. In *HLT-NAACL*, 2003. 24

[186] Frank Wilcoxon. **Individual comparisons by ranking methods**. *Biometrics Bulletin*, **1**:80–83, 1945. 61

[187] Fei Wu, Raphael Hoffmann, and Daniel S. Weld. **Information extraction from Wikipedia: moving down the long tail**. In *KDD*, pages 731–739, 2008. 38

[188] Fei Wu and Daniel S. Weld. **Open Information Extraction Using Wikipedia**. In *ACL*, pages 118–127, 2010. 38

[189] Mihalis Yannakakis. **Node-and edge-deletion NP-complete problems**. In *STOC '78: Proceedings of the tenth annual ACM symposium on Theory of computing*, pages 253–264, New York, NY, USA, 1978. ACM. 47

[190] Limin Yao, Sebastian Riedel, and Andrew McCallum. **Collective Cross-Document Relation Extraction Without Labelled Data**. In *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pages 1013–1023, Cambridge, MA, October 2010. Association for Computational Linguistics. Available from: `http://www.aclweb.org/anthology/D10-1099`. 38

[191] Alexander Yates and Oren Etzioni. **Unsupervised Methods for Determining Object and Relation Synonyms on the Web**. *Journal of Artificial Intelligence Research*, **34**:255–296, 2009. 7, 27, 28, 38, 45, 63

[192] Eric Yeh, Daniel Ramage, Christopher D. Manning, Eneko Agirre, and Aitor Soroa. **WikiWalk: Random walks on Wikipedia for Semantic Relatedness**. In *Proceedings of the 2009 Workshop on Graph-based Methods for Natural Language Processing (TextGraphs-4)*, pages 41–49, Suntec, Singapore, August 2009. Association for Computational Linguistics. Available from: `http://www.aclweb.org/anthology/W/W09/W09-3206`. 16

[193] Fabio Massimo Zanzotto, Marco Pennacchiotti, and Alessandro Moschitti. **A Machine Learning Approach to Textual Entailment Recognition**. *Journal of Natural Language Engineering*, **15**(4):551–582, 2009. 4

[194] Fabio Massimo Zanzotto, Marco Pennacchiotti, and Maria Teresa Pazienza. **Discovering Asymmetric Entailment Relations between Verbs Using Selectional Preferences**. In *Proceedings of ACL*, 2006. 9

[195] Naomi Zeichner, Jonathan Berant, and Ido Dagan. **Crowdsourcing Inference-Rule Evaluation**. In *Proceedings of ACL*, 2012. 13

[196] Shiqi Zhao, Haifeng Wang, Ting Liu, and Sheng Li. **Pivot Approach for Extracting Paraphrase Patterns from Bilingual Corpora**. In *Proceedings of ACL*, pages 780–788, 2008. 18

[197] Guodong Zhou, Jian Su, Jie Zhang, and Min Zhang. **Exploring Various Knowledge in Relation Extraction**. In *ACL*, 2005. 35