Joint Models for Information and Knowledge Extraction

A dissertation submitted towards the degree
Doctor of Natural Science/Engineering
of the Faculty of Mathematics and Computer Science of
Saarland University

by

Dat Ba Nguyen

Saarbrücken
2017

Colloquium

Date	:	01.12.2017
Place	:	Saarbrücken
Dean	:	Prof. Frank-Olaf Schreyer

Examination Board

Supervisor and Reviewer	:	Prof. Gerhard Weikum
Supervisor and Reviewer	:	Prof. Martin Theobald
Reviewer	:	Prof. Klaus Berberich
Chairman	:	Prof. Dietrich Klakow
Scentific Assitant	:	Dr. Simon Razniewski

Abstract

Information and knowledge extraction from natural language text is a key asset for question answering, semantic search, automatic summarization, and other machine reading applications. There are many sub-tasks involved such as named entity recognition, named entity disambiguation, co-reference resolution, relation extraction, event detection, discourse parsing, and others. Solving these tasks is challenging as natural language text is unstructured, noisy, and ambiguous. Key challenges, which focus on identifying and linking named entities, as well as discovering relations between them, include:

- *High NERD Quality.* Named entity recognition and disambiguation, NERD for short, are preformed first in the extraction pipeline. Their results may affect other downstream tasks.
- *Coverage vs. Quality of Relation Extraction.* Model-based information extraction methods achieve high extraction quality at low coverage, whereas open information extraction methods capture relational phrases between entities. However, the latter degrades in quality by non-canonicalized and noisy output. These limitations need to be overcome.
- *On-the-fly Knowledge Acquisition.* Real-world applications such as question answering, monitoring content streams, etc. demand on-the-fly knowledge acquisition. Building such an end-to-end system is challenging because it requires high throughput, high extraction quality, and high coverage.

This dissertation addresses the above challenges, developing new methods to advance the state of the art. The first contribution is a robust model for joint inference between entity recognition and disambiguation. The second contribution is a novel model for relation extraction and entity disambiguation on Wikipedia-style text. The third contribution is an end-to-end system for constructing query-driven, on-the-fly knowledge bases.

Kurzfassung

Informations- und Wissensextraktion aus natürlichsprachlichen Texten sind Schlüsselthemen vieler wissensbassierter Anwendungen. Darunter fallen zum Beispiel Frage-Antwort-Systeme, semantische Suchmaschinen, oder Applikationen zur automatischen Zusammenfassung und zum maschinellem Lesen von Texten. Zur Lösung dieser Aufgaben müssen u.a. Teilaufgaben, wie die Erkennung und Disambiguierung benannter Entitäten, Koreferenzresolution, Relationsextraktion, Ereigniserkennung, oder Diskursparsen, durchgeführt werden. Solche Aufgaben stellen eine Herausforderung dar, da Texte natürlicher Sprache in der Regel unstrukturiert, verrauscht und mehrdeutig sind. Folgende zentrale Herausforderungen adressieren sowohl die Identifizierung und das Verknüpfen benannter Entitäten als auch das Erkennen von Beziehungen zwischen diesen Entitäten:

- *Hohe NERD Qualität. Die Erkennung und Disambiguierung benannter Entitäten* (engl. "Named Entity Recognition and Disambiguation", kurz "NERD") wird in Extraktionspipelines in der Regel zuerst ausgeführt. Die Ergebnisse beeinflussen andere nachgelagerte Aufgaben.
- *Abdeckung und Qualität der Relationsextraktion.* Modellbasierte Informationsextraktionsmethoden erzielen eine hohe Extraktionsqualität, bei allerdings niedriger Abdeckung. Offene Informationsextraktionsmethoden erfassen relationale Phrasen zwischen Entitäten. Allerdings leiden diese Methoden an niedriger Qualität durch mehrdeutige Entitäten und verrauschte Ausgaben. Diese Einschränkungen müssen überwunden werden.
- *On-the-Fly Wissensakquisition.* Reale Anwendungen wie Frage-Antwort-Systeme, die Überwachung von Inhaltsströmen usw. erfordern On-the-Fly Wissensakquise. Die Entwicklung solcher ganzheitlichen Systeme stellt eine hohe Herausforderung dar, da ein hoher Durchsatz, eine hohe Extraktionsqualität sowie eine hohe Abdeckung erforderlich sind.

Diese Arbeit adressiert diese Probleme und stellt neue Methoden vor, um den aktuellen Stand der Forschung zu erweitern. Diese sind:

- Ein robustes Modell zur integrierten Inferenz zur gemeinschaftlichen Erkennung und Disambiguierung von Entitäten.
- Ein neues Modell zur Relationsextraktion und Disambiguierung von Wikipedia-ähnlichen Texten.

- Ein ganzheitliches System zur Erstellung Anfrage-getriebener On-the-Fly Wissensbanken.

Acknowledgements

I would take this moment to thank my advisor Prof. Martin Theobald and my supervisor Prof. Gerhard Weikum for their invaluable guidance throughout my doctoral studies. I especially enjoyed the freedom they gave me to pursue my research interests. They have not only assisted me in completing this work, but they have also helped me to broaden my attitude towards research, and to develop my personality.

I would like to thank my co-authors Johannes Hoffart, Abdalghani Abujabal and Nam Khanh Tran for their great team work and insightful discussions. Many thanks to my officemates Amy Siu, Sairam Gurajada, Xuan-Cuong Chu, and to all my colleagues and staff at D5 group for making the workplace an exciting atmosphere.

A special note of thanks to Quan Nguyen and his family for their great help to my life. Thanks are sent to Uncle Duy Ta, Aunt Hong Le, Aunt Nhung Le, Aunt Nga Le, and to my close friends Quoc-Dai Nguyen, Hai-Dang Tran, Hoang-Vu Nguyen, Duc-Duy Nguyen for their encouragement.

Finally, I would like to thank my parents and my parents-in-law. Without their support I would have not done this work.

To my little family Thanh Hoa, Minh Vu and Hai Phong

Contents

Chapter 1

Introduction

1.1 Scope and Goals

The most natural form of storing information in the human history is text such as books, news articles, web pages, and more. This massive information source keeps increasing day by day. In the last decade, computer scientists have put a lot of effort into automatically extracting, representing, and organizing meaningful information from natural language text. *Information and knowledge extraction* is the process of deriving *high-quality* information or knowledge from natural language text (Hearst, 1999). Consider the following example:

> *"In 1905, Einstein published a paper advancing an explanation of the photoelectric effect, which awarded him a Nobel Prize later."*

The goal is to automatically extract potentially valuable information and lift it into formal representation, for example:

- fact F_1 : $\langle$`Albert_Einstein`, `work_on_in`, `Photoelectric_effect`, 1905$\rangle$,
- fact F_2 : $\langle$`Albert_Einstein`, `win`, `Nobel_Prize_in_Physics`$\rangle$,
- the temporal and causality relationships between F_1 and F_2.

There are many sub-tasks involved such as named entity recognition, named entity disambiguation, co-reference resolution, relation extraction, event detection, discourse parsing, and more.

Named Entity Recognition, NER for short, is the task that deals with the identification of entity mentions in natural language text and their classification into coarse-grained semantic types such as person, location, organization, misc, and more (Grishman and Sundheim, 1996; Tjong Kim Sang and De Meulder, 2003; Finkel et al., 2005). For example, state-of-the-art NER tools annotate the above example sentence with a `PERSON` mention "*Einstein*" and a `MISC` mention "*Nobel Prize*". Recently, the strict requirement that a mention must refer to an individual named entity as opposed to a general concept is relaxed, allowing for other *informative* noun-phrases to be recognized as well. This is because the recognition of such noun-phrases may contribute to the end results of the extraction process.

For instance, the TIME mention "*1905*" and the MISC mention "*photoelectric effect*" should be annotated as well.

Named Entity Disambiguation, NED for short, is the task that involves the disambiguation of entity mentions by mapping them to proper entities in a knowledge base (Bunescu and Pasca, 2006; Hoffart et al., 2011a; Cucerzan, 2014). For example, NED tools link the mention "*Einstein*" to the physicist Albert_Einstein, the mention "*Nobel Prize*" to Nobel_Prize_in_Physics, etc.

Co-reference Resolution, CR for short, is the task that aims to identify all linguistic expressions referring to the same entity within a text (Hirschman and Chinchor, 1998; Doddington et al., 2004). For example, CR tools determine that "*Einstein*" and "*him*" both refer to the same entity, but are different from the mention "*Nobel Prize*".

Relation Extraction is the task that aims to detect and classify the semantic relations between entities, and thus can perform fact extraction from natural language text (Surdeanu and Ciaramita, 2007; Mintz et al., 2009; Suchanek et al., 2009; Riedel et al., 2013). For example, the relation win between two entities Albert_Einstein and Nobel_Prize_in_Physics can be inferred based on the textual context (i.e., "*awarded*") in the example sentence.

Event Detection is the task that aims to extract events from text, each consisting of a fact with a given point of time and/or place (Ling and Weld, 2010; Kuzey and Weikum, 2012, 2014). For example, the fact that Albert Einstein won the Nobel Prize in Physics at a specific point of time (i.e., 1921) is an event.

Discourse Parsing is the task that aims to detect and categorize discourse relations between discourse segments in the text (Hernault et al., 2010; Feng and Hirst, 2012; Xue et al., 2015). For example, the example sentence expresses the temporal and causality relationships between the work of Einstein on photoelectric effect and his achievement (i.e., a Nobel Prize) later on.

Despite the fact that computer science and computational linguistics scientists have been working on those tasks for years, information and knowledge extraction is still not a solved task as natural language text is unstructured, noisy, and ambiguous. There is a multitude of issues that need to be dealt with. Some of these –which focus on identifying and linking named entities, as well as discovering relations between them– are addressed in this work.

1.2 Challenges

Challenge C1: NERD Quality. Named entity recognition and disambiguation, NERD for short, are preformed first in the knowledge extraction pipeline. Their results may affect other downstream tasks. Therefore, controlling the quality of this process is extremely important. State of the art methods such as Finkel et al. (2005), Ratinov and Roth (2009) for NER, and Cucerzan (2014); Hoffart et al.

(2011a); Ratinov et al. (2011) for NED are still far from human understanding capabilities. Moreover, NERD typically proceed in two stages, which may produce inconsistent results. For example, there are cases when NER classifies a mention (e.g., "*Germany*") into type `LOCATION` while NED links it to an `ORGANIZATION` entity (e.g., `Germany_national_football_team`).

Challenge C2: Coverage vs. Quality of Relation Extraction. Early approaches (Brin, 1998; Mintz et al., 2009; Suchanek et al., 2009; Singh et al., 2013) put great emphasis on the extraction quality by focusing on a set of predefined relations like those present in a knowledge base. However, they are inherently limited in the coverage of what happens in the real world. Other methods (Banko et al., 2007; Etzioni et al., 2011; Mausam et al., 2012) extract relational surface phrases, and thus achieve much higher recall. They can potentially find any relation that holds between entities. However, the relational phrases are not canonicalized, and they are often noisy. These limitations of both directions need to be overcome.

Challenge C3: On-the-fly Knowledge Acquisition. On-the-fly information extraction in general –and knowledge base construction in particular– are in high demand. They facilitate real-world applications such as question answering (Berant et al., 2013; Bast and Haussmann, 2015; Xu et al., 2016), monitoring digital content streams (e.g., when an analyst or journalist becomes interested in a particular person, organization or event), and more. These types of systems require:

- *efficiency* as they have to process a collection of text on-the-fly,
- *high coverage* as they have to capture new entities, facts, and events,
- *high quality* as the output will serve as input for downstream applications.

Building such an end-to-end system is challenging.

1.3 Contributions

This work addresses the above challenges, developing new methods to advance the state of the art:

J-NERD. We present J-NERD (Nguyen et al., 2014, 2016), a novel kind of probabilistic graphical model for the joint recognition and disambiguation of named-entity mentions in natural-language text. J-NERD is based on a supervised, non-linear graphical model that combines multiple per-sentence *tree-shaped* models into an entity-coherence-aware global model. The global model detects mention spans, tags them with coarse-grained types, and maps them to entities in a single joint-inference step based on the Viterbi algorithm (for exact inference) or Gibbs sampling (for approximate inference). J-NERD additionally considers richer features, including domain-oriented feature and syntactic dependency feature (i.e., harnessing dependency parse trees and verbal patterns that indicate mention types as part of their ***subject*** or ***object*** arguments). J-NERD improves the quality over the existing state of the art named entity recognition and disambiguation systems (**Challenge C1**).

J-REED. We present J-REED (Nguyen et al., 2017), a novel joint model for relation extraction and entity disambiguation on Wikipedia-style text. J-REED is based on graphical models that capture interdependencies between entity disambiguation and the relational phrases. By considering which lexical types of entities are compatible with the type signature of which relation, we can boost the accuracy of both sub-tasks. Entity names are mapped to the entities registered in a knowledge base, while relation patterns are extracted as crisp as possible (**Challenge C2**). Additionally, without particular assumptions about the target relations, J-REED can capture many interesting relations which are absent from all recent knowledge bases. J-REED improves the quality over pipelined combinations of the state-of-the-art Open Information Extraction (Open IE) systems and NED systems.

QKBfly. We present QKBfly (Nguyen et al., 2018), a novel end-to-end system for constructing query-driven, on-the-fly KBs. QKBfly takes as input an entity-centric query or a natural-language question, automatically retrieves relevant source documents (via Wikipedia and news sources), runs a novel form of knowledge extraction on the sources, and builds a high-coverage knowledge base that is focused on the entities of interest (**Challenge C3**). At the heart of QKBfly is a semantic-graph representation of sentences that captures per-sentence clauses, noun-phrases, pronouns, as well as their syntactic and semantic dependencies. Based on this graph, we devise an efficient inference technique that performs three key information extraction tasks, namely named-entity disambiguation, coreference resolution and relation extraction, in a light-weight and integrated manner. Because of the clause-based representation of sentences, QKBfly is not limited to binary predicates but can also extract ternary (or higher-arity) predicates. Compared to mainstream KBs, we acquire facts for a much larger set of predicates. Compared to Open IE methods, arguments of facts are canonicalized, thus referring to unique entities with semantically typed predicates derived from precomputed clusters of phrases (**Challenge C2**). In addition to supporting analytical queries, QKBfly thus also facilitates the application of question-answering (QA) frameworks.

1.4 Publications

This work includes material published in top-tier conferences and journals:

- D. B. Nguyen, J. Hoffart, M. Theobald, and G. Weikum. (2014). AIDA-light: High-Throughput Named-Entity Disambiguation. In *Proceedings of the Workshop on Linked Data on the Web co-located with the 23rd International World Wide Web Conference (WWW '14)*.
- D. B. Nguyen, M. Theobald, and G. Weikum. (2016). J-NERD: Joint Named Entity Recognition and Disambiguation with Rich Linguistic Features. *Transactions of the Association of Computational Linguistics (TACL, vol. 4)*.

- D. B. Nguyen, M. Theobald, and G. Weikum. (2017). J-REED: Joint Relation Extraction and Entity Disambiguation. In *Proceedings of the 26th Conference on Information and Knowledge Management (CIKM '17)*.
- D. B. Nguyen, A. Abujabal, N. K. Tran, M. Theobald, and G. Weikum. (2018). Query-Driven On-The-Fly Knowledge Base Construction. In *Proceedings of the 44th International Conference on Very Large Databases (VLDB '18)*.

1.5 Organization

The remainder of this dissertation is organized as follows. **Chapter** 2 introduces the data model, notations, and the feature space for named entity disambiguation - the core task in our work. This is followed by three Chapters which describe our methods for named entity recognition and disambiguation, relation extraction, and on-the-fly knowledge base construction. **Chapter** 3 presents a joint model for entity recognition and disambiguation. **Chapter** 4 presents a joint model for relation extraction and entity disambiguation for Wikipedia and other kinds of entity-centric documents. **Chapter** 5 presents an end-to-end system for on-the-fly KB construction that is triggered by an entity-centric user query or a natural-language question. Finally, **Chapter** 6 gives conclusions and possible directions for future work.

Chapter 2

Preliminaries

2.1 Data Model

In this section, we introduce our data model and establish necessary notations.

Pre-processing. Text from an input document (D) is pre-processed through a standard NLP pipeline, including tokenization, part-of-speech (POS) tagging, noun-phrase chunking, and dependency parsing. Thus, we obtain:

- A sequence of tokens **tok** = $\langle tok_1, tok_2, \ldots, tok_l \rangle$. We consider words, punctuations, and other special characters like "$", "€", etc. as tokens.
- A sequence of POS tags **pos** = $\langle pos_1, pos_2, \ldots, pos_l \rangle$. Every token has a POS tag, for example *NNP* for proper nouns, *VBD* for past tense verbs, etc. (Santorini, 1990).
- A sequence of noun-phrases **np** = $\langle np_1, np_2, \ldots, np_n \rangle$. A noun-phrase is a phrase (i.e., formed by adjacent tokens) which has a noun or an indefinite pronoun as its head token. In our models, overlaps between noun-phrases are not allowed.

For example, performing the NLP pipeline on the sentence "*Einstein was awarded a Nobel Prize.*" results in sequences:

- **tok** = ⟨*"Einstein"*, *"was"*, *"awarded"*, *"a"*, *"Nobel"*, *"Prize"*, *"."*⟩,
- **pos** = ⟨*NNP*, *VBD*, *VBN*, *DT*, *NNP*, *NNP*, .⟩,
- **np** = ⟨*"Einstein"*, *"a Nobel Prize"*⟩.

These sequences and the *dependency types* (de Marneffe et al., 2006) linked between tokens in **tok** such as ***nsubj*** (e.g., between *"awarded"* and *"Einstein"*), ***dobj*** (e.g., between *"awarded"* and *"Prize"*), ***prep_in***, ***prep_for***, etc. serve as input to the feature functions in our work.

Named Entity Recognition. NER for short, identifies entity mentions **m** = $\langle m_1, m_2, \ldots, m_t \rangle$, each with a coarse-grained type such as `PERSON`, `ORGANIZATION`, `LOCATION`, etc. For example, there are two mentions including m_1 = *"Einstein"* with type `PERSON`, and m_2 = *"Nobel Prize"* with type `MISC` in the above sentence. A mention is usually a sub-sequence of a noun-phrase.

Named Entity Recognition. NED for short, links each mention m_i to an entity e_i in an entity repository (E) extracted from Wikipedia or from a knowledge base. Thus, we obtain e_1 = `Albert_Einstein`, and e_2 = `Nobel_Prize_in_Physics`.

Co-reference Resolution. CR for short, determines whether two noun-phrases, or a noun-phrase and a pronoun (defined by the POS tags) refer to the same entity.

Relation Extraction. Finally, relation extraction (or knowledge extraction) detects and classifies the semantic relationships between entities. We consider different kinds of relations. Chapter 4 focuses on binary relations between two entities while Chapter 5 explores all possible relations in a group of entities including higher-arity ones.

2.2 Feature Space for Named Entity Disambiguation

In this section, we present a feature space used for named entity disambiguation (NED) – the core task in our work. After the pre-processing steps through a standard NLP pipeline, we extract some different kinds of *contexts* (e.g., token context, domain theme, and dependency context) for our features. Some of the features are fairly standard, whereas others are novel.

- Standard features include prior probability of mention-entity mapping, and similarity measures between mention strings and entity names. Additionally, mention-entity token context similarity and entity-entity token coherence are important features for NED – not exactly a standard feature, but used in some prior works (Cucerzan, 2014; Hoffart et al., 2011a; Ratinov et al., 2011).
- Novel features about the topical domain of an input text (e.g., politics, sports, football, etc.) are obtained by a classifier based on "*easy mentions*": those mentions for which the NED decision can be made easily with very high confidence without advanced features (Nguyen et al., 2014).
- The third feature group captures syntactic dependencies from the sentence parsing (Nguyen et al., 2016). To our knowledge, these have not been used in prior works.

2.2.1 Backgrounds

Named entity disambiguation is the task of linking a named entity mention to an instance in a knowledge base such as DBpedia (Auer et al., 2007), Yago (Suchanek et al., 2007), Wikidata (Vrandečić and Krötzsch, 2014), Freebase (Bollacker et al., 2008), and more. Consider the following sentence:

"David played for manu, real, and la galaxy."

with the four mentions "*David*", "*manu*", "*real*", and "*la galaxy*" that have already been marked up, NED aims to link them to the football player `David_Beckham` and the three football clubs including `Manchester_United_F.C.`, `Real_Madrid_C.F.`,

and LA_Galaxy, respectively. State-of-the-art systems includes the Wikipedia Miner Wikifier (Milne and Witten, 2013), the Illinois Wikifier (Ratinov et al., 2011), Spotlight (Mendes et al., 2011), Semanticizer (Meij et al., 2012), TagMe (Ferragina and Scaiella, 2010; Cornolti et al., 2014), and AIDA (Hoffart et al., 2011a). Most of these prior works combine contextual similarity measures with some form of consideration for the contextual coherence among a selected set of candidate entities. Popular joint models are graph algorithms (Hoffart et al., 2011a), integer linear programming (Ratinov et al., 2011), or probabilistic graphical models (Kulkarni et al., 2009). To mark up mentions, these methods use the Stanford NER Tagger (Finkel et al., 2005) or dictionary-based matching.

2.2.2 Entity Repository and Name-Entity Dictionary

We harness a knowledge base, namely Yago (Suchanek et al., 2007; Hoffart et al., 2011a, 2013), as an entity repository and as a dictionary of name-to-entity pairs (i.e., aliases and paraphrases). We import the Yago *means* and *hasName* relations, a total of more than 6 Million name-entity pairs (for more than 3 Million distinct entities). Additionally, we enrich this dictionary by including the first and last names of PERSON entities in Yago.

Dictionary Augmentation via Locality Sensitive Hashing. In order to improve recall over a variety of input texts, including Web pages where many out-of-dictionary mentions occur, we apply Locality Sensitive Hashing (LSH) in combination with Min-Hash (Gionis et al., 1999; Indyk and Motwani, 1998; Broder et al., 1998) to cover more spelling variations among these mentions. That is, once an out-of-dictionary mention occurs, we first attempt to find *similar names* in our name-entity dictionary via LSH. Second, all possible entity candidates of these similar names are assigned to this mention as candidate entities. For example, the mention "*Northwest Fur Company*" does not exist in the name-entity dictionary, and thus there is no entity candidate for it. However, via LSH we are able to detect that "*Northwest Fur Company*" and the dictionary entry "*North West Fur Company*" are highly similar, such that we are able to identify the entity North_West_Company as a candidate entity for this mention. KORE (Hoffart et al., 2012) is the first work that integrated LSH into an NED system to cluster key-phrases for an efficient form of similarity computation.

2.2.3 Standard Features

2.2.3.1 Context-Independent Features

Mention-Entity Prior. Feature $f_1(m_i, e_i)$ captures a prior probability of mention m_i mapping to entity e_i. These probabilities are estimated from co-occurrence frequencies of name-to-entity pairs in the background corpus, thus harnessing link-anchor texts in Wikipedia. For example, on the one hand, we may have a prior of f_1("*Beckham*", David_Beckham) = 0.7, as David Beckham is more popular (today)

than his wife Victoria. On the other hand, f_1("*David*", `David_Beckham`) may be lower than f_1("*David*", `David_Bowie`), for example, as this still active pop star is more frequently and prominently mentioned than the retired football player.

Mention-Entity n-Gram Similarity. Feature $f_2(m_i, e_i)$ measures the Jaccard similarity of character-level n-grams of mention name m_i and the primary (i.e., full and most frequently used) name of entity e_i. For example, for $n = 2$ the value of f_2("*Becks*", `David_Beckham`) is $\frac{3}{11}$. In our experiments, we set $n = 3$.

2.2.3.2 Token-Context-Based Features

We first explain how we maintain the *token contexts* of both mentions and candidate entities which form the basis for our features.

Mention Token Context. The *token context* of mention m_i, denoted by $cnt(m_i)$, is extracted from the input tokens. Specifically, we form $cnt(m_i)$ by taking all tokens with *tf-idf* scores except for stop-words.

Entity Token Context. Similarly, the token context of entity e_i, denoted by $cnt(e_i)$, consists of tokens (with *tf-idf* scores) which are obtained by simplifying the key-phrases provided by AIDA (Hoffart et al., 2011b) for e_i. For example, the context of the entity `David_Beckham` with two key-phrases {"*M.U. player*", "*M.U. midfielder*"} in AIDA is reduced to the vector of tokens {"*M.U.*", "*player*", "*midfielder*"}.

Mention-Entity Token Context Similarity. Feature $f_3(m_i, e_i)$ measures the similarity between the token contexts of mention m_i and entity e_i. For example, the mention "*David*" in the context of "*midfielder*", "*M.U.*", "*football*", etc. is more coherent with entity `David_Beckham` than with entity `David_Bowie`.

As similarity measure, we employ the weighted overlap coefficient between two vectors of weighted elements $cnt(x) = \langle v_1, v_2, \dots \rangle$ and $cnt(y) = \langle v'_1, v'_2, \dots \rangle$:

$$sim\big(cxt(x), cxt(y)\big) = \frac{\sum_k \min(v_k, v'_k)}{\min\big(\sum_k v_k, \sum_k v'_k\big)} \tag{2.1}$$

Entity-Entity Token Context Similarity. In analogy to mention-entity token context similarity feature, $f_4(e_i, e_j)$ measures the coherence between the two token contexts of two entity candidates e_i and e_j. This feature allows us to establish cross-dependencies among labels in our graphical model. For example, the two entities `David_Beckham` and `Manchester_United_F.C.` are highly coherent as they share many tokens in their contexts, such as "*champions*", "*league*", "*premier*", "*cup*", etc. Thus, they should mapped jointly.

2.2.4 Domain-Oriented Feature

We use WordNet *domains*, created by Miller (1995), Magnini and Cavaglià (2000), and Bentivogli et al. (2004), to construct a taxonomy of domains, including *Sports* (e.g., football, basketball, etc.), *Science* (e.g., computer science, mathematics, etc.),

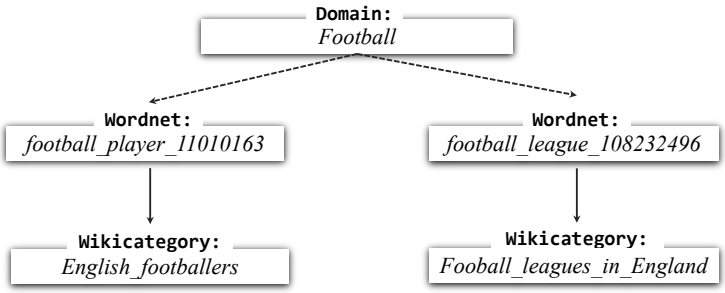

Figure 2.1: A hierarchy for the domain *Football*.

and more. We combine the domains with semantic *types* (classes of entities) provided by Yago, by assigning them to their respective domains. This is based on the manual assignment of WordNet synsets to domains, introduced by Magnini and Cavaglià (2000), and Bentivogli et al. (2004), and extends to additional types in Yago. For example, `SINGER` is assigned to *Music*, and `FOOTBALL-PLAYER` to *Football*, etc. In total, the 46 domain hierarchies[1] are enhanced with ca. 350,000 types imported from Yago. Figure 2.1 shows a domain hierarchy for *Football*.

We classify input texts into domains by means of "*easy mentions*". An easy mention is a match in the name-to-entity dictionary for which there exist at most three candidate entities. Although the mention boundaries may not be explicitly provided as input, we still can extract these easy mentions from the entirety of all mention candidates. In the following, let C^* be the set of candidate entities for the easy mentions in the input text. For each domain d in the 46 domains, we compute the *coherence* of the easy mentions $M^* = \{m_1, m_2, \dots\}$:

$$coh(M^*) = \frac{|C^* \cap C^d|}{|C^*|} \tag{2.2}$$

where C^d is the set of all entities under domain d. We classify the document into the domain with the highest coherence score. Although the mentions and their entities may be inferred incorrectly, the domain classification still tends to work very reliably as it aggregates over all easy mention candidates.

Domain-Entity Coherence. Feature $f_5(m_i, e_i)$ captures the coherence between mention m_i and entity candidate e_i with the respect to the domain d which the input text is classified into. That is, $f_5(m_i, e_i) = 1$ if domain d contains entity e_i; otherwise, the feature value is 0. For example, in domain *Football*, the mention "*David*" is more coherent with entity `David_Beckham` than with entity `David_Bowie`.

[1] The 46 domains: *badminton, baseball, basketball, cricket, football, golf, table tennis, rugby, soccer, tennis, volleyball, cycling, skating, skiing, hockey, mountaineering, rowing, swimming, sub, diving, racing, athletics, wrestling, boxing, fencing, archery, fishing, hunting, music, agriculture, alimentation, architecture, computer science, engineering, medicine, veterinary, astronomy, biology, chemistry, earth, mathematics, physics, economy, fashion, industry, politics.*

Entity-Entity Type Coherence. Feature $f_6(e_i, e_j)$ computes the relatedness between the Wikipedia categories of two candidate entities e_i and e_j.

$$f_6(e_i, e_j) = \max_{\substack{c_u \in cat(e_i) \\ c_v \in cat(e_j)}} tc(c_u, c_v) \tag{2.3}$$

where the *type coherence* function $tc(c_u, c_v)$ computes the reciprocal length of the shortest path between categories c_u, c_v in the domain taxonomy; and *cat* function retrieves all Wikipedia categories related to an entity.

2.2.5 Syntactic Dependency Feature

Mention Dependency Context. The *dependency context* of mention m_i, denoted by *dep-cxt*(m_i), contains all *dependency patterns* where (1) a pattern consists of a *dependency types* (de Marneffe et al., 2006), like ***nsubj***, ***dobj***, ***prep_in***, ***prep_for***, etc., with two *arguments*, and (2) one of the two argument is m_i. We assign a *tf-idf* score for each pattern in the dependency context.

Entity Dependency Context. Similarly, the dependency context of entity e_i, denoted by *dep-cxt*(e_i), contains all dependency patterns (with *tf-idf* scores) which are obtained from the background corpus (i.e., Wikipedia). For example, the dependency context of entity `Manchester_United_F.C.` includes ***prep_for****[play, *]*, ***nsubj****[*, win]*, etc.

Mention-Entity Dependency Context Similarity. Feature $f_7(m_i, e_i)$ measures the weighted overlap coefficient between the dependency contexts of mention m_i and candidate entity e_i. For example, the mention "*manu*" in the context of pattern ***prep_for****[play, *]* is more coherent with entity `Manchester_United_F.C.` than with entity `Manu_Chao` as the pattern "`FOOTBALLER` *play for* `FOOTBALL-CLUB`" is more frequent than any patterns of something or somebody "*play for* `SINGER`".

Chapter 3

Joint Model for Named Entity Recognition and Disambiguation

3.1 Introduction

Motivation. Methods for named entity recognition and disambiguation, NERD for short, typically proceed in two stages:

- At the NER stage, text spans of entity mentions are detected and tagged with coarse-grained types like `PERSON`, `ORGANIZATION`, `LOCATION`, etc. This is typically performed by a trained Conditional Random Field (CRF) over word sequences (e.g., Finkel et al. (2005)).
- At the NED stage, mentions are mapped to entities in a knowledge base (KB) based on contextual similarity measures and the semantic coherence of the selected entities (e.g., Cucerzan (2014); Hoffart et al. (2011a); Ratinov et al. (2011)).

This two-stage approach has limitations. First, NER may produce false positives that can misguide NED. Second, NER may miss out on some entity mentions, and NED has no chance to compensate for these false negatives. Third, NED is not able to help NER, for example, by disambiguating "*easy mentions*" (e.g., of prominent entities with more or less unique names), and then using the entities and knowledge about them as enriched features for NER.

Example. Consider the following sentences:

"David played for manu, real, and la galaxy.
His wife posh performed with the spice girls."

This is difficult for NER because of the absence of upper-case spelling, which is not untypical in social media, for example. Most NER methods will miss out on multi-word mentions or words that are also common nouns ("*spice*") or adjectives ("*posh*", "*real*"). Typically, NER would pass only the mentions "*David*", "*manu*", and "*la*" to the NED stage, which then is prone to many errors like mapping the first two mentions to any prominent people with first names David and Manu, and

mapping the third one to the city of Los Angeles. With NER and NED performed jointly, the possible disambiguation of "*la galaxy*" to the soccer club can guide NER to tag the right mentions with the right types (e.g., recognizing that "*manu*" could be a short name for a soccer team), which in turn helps NED to map "*David*" to the right entity `David_Beckham`.

Contribution. This chapter presents a novel kind of probabilistic graphical model for the joint recognition and disambiguation of named-entity mentions in natural-language text. With this integrated approach to NERD, we aim to overcome the limitations of the two-stage NER/NED methods. Our method, called $\mathcal{J}$-NERD (Nguyen et al., 2016), is based on a supervised, non-linear graphical model that combines multiple per-sentence models into an entity-coherence-aware global model. The global model detects mention spans, tags them with coarse-grained types, and maps them to entities in a single joint-inference step based on the Viterbi algorithm (for exact inference) or Gibbs sampling (for approximate inference). J-NERD comprises the following novel contributions:

- a tree-shaped model for each sentence, whose structure is derived from the dependency parse tree and thus captures linguistic context in a deeper way compared to prior work with CRF's for NER and NED;
- richer features not considered in prior work, including domain-oriented feature and syntactic dependency feature (i.e., harnessing dependency parse trees and verbal patterns that indicate mention types as part of their ***nsubj*** or ***dobj*** arguments);
- an inference method that maintains the uncertainty of both mention candidates (i.e., token spans) and entity candidates for competing mention candidates, and makes joint decisions, as opposed to fixing mentions before reasoning on their disambiguation.

We present experiments with three major datasets: the CoNLL'03 collection of newswire articles, the ACE'05 corpus of news and blogs, and the ClueWeb'09-FACC1 corpus of web pages. Baselines that we compare J-NERD with include Spotlight (Daiber et al., 2013), TagMe (Ferragina and Scaiella, 2010), and the recent joint NERD method of Durrett and Klein (2014). J-NERD consistently outperforms these competitors in terms of both precision and recall.

3.2 Related Work

NER. Detecting the boundaries of text spans that denote named entities has been mostly addressed by supervised CRF's over word sequences (McCallum and Li, 2003; Finkel et al., 2005). The work of Ratinov and Roth (2009) improved these techniques by additional features from context aggregation and external lexical sources (gazetteers, etc.). Passos et al. (2014) harnessed skip-gram features and external dictionaries for further improvement. An alternative line of NER techniques is based on dictionaries of name-entity pairs, including nicknames, shorthand names, and paraphrases (e.g., "*the first man on the moon*"). The work of

Ferragina and Scaiella (2010) and Mendes et al. (2011) are examples of dictionary-based NER. The work of Spitkovsky and Chang (2012) is an example of a large-scale dictionary that can be harnessed by such methods.

An additional output of the CRF's are type tags for the recognized word spans, typically limited to coarse-grained types like `PERSON`, `ORGANIZATION`, `LOCATION`, and `MISC`. The most widely used tool of this kind is the Stanford NER Tagger (Finkel et al., 2005). Many NED tools use the Stanford NER Tagger in their first stage of detecting mentions.

Mention Typing. The specific NER task of inferring semantic types has been further refined and extended by various works on fine-grained typing (e.g., `MUSICIAN`, `POLITICIAN`, `SINGER`, `GUITARIST`, etc.) for entity mentions and general noun phrases (Fleischman and Hovy, 2002; Rahman and Ng, 2010; Ling and Weld, 2012; Yosef et al., 2012; Nakashole et al., 2013). Most of these works are based on supervised classification, using linguistic features from mentions and their surrounding text. One exception is the work of Nakashole et al. (2013) which is based on text patterns that connect entities of specific types, acquired by sequence mining from the Wikipedia full-text corpus. In contrast to our work, those are simple surface patterns, and the task addressed here is limited to typing noun phrases that likely denote emerging entities that are not yet registered in a KB.

NED. Methods and tools for NED go back to the seminal work of Dill et al. (2003), Bunescu and Pasca (2006), Cucerzan (2007), and Milne and Witten (2008). More recent advances led to open-source tools like the Wikipedia Miner Wikifier (Milne and Witten, 2013), the Illinois Wikifier (Ratinov et al., 2011), Spotlight (Mendes et al., 2011), Semanticizer (Meij et al., 2012), TagMe (Ferragina and Scaiella, 2010; Cornolti et al., 2014), and AIDA (Hoffart et al., 2011a). We choose some, namely Spotlight and TagMe, as baselines for our experiments. These are the best-performing, publicly available systems for news and web texts. Most of these methods combine contextual similarity measures with some form of consideration for the coherence among a selected set of candidate entities for disambiguation. The latter aspect can be cast into a variety of computational models, like graph algorithms (Hoffart et al., 2011a), integer linear programming (Ratinov et al., 2011), or probabilistic graphical models (Kulkarni et al., 2009). All these methods use the Stanford NER Tagger or dictionary-based matching for their NER stages. Kulkarni et al. (2009) uses an ILP or LP solver (with rounding) for the NED inference, which is computationally expensive. Note that some of the NED tools aim to link not only named entities but also general concepts (e.g. "*world peace*") for which Wikipedia has articles. In our work, we solely focus on proper entities.

Joint NERD. There is little prior work on performing NER and NED jointly. Sil and Yates (2013), and Durrett and Klein (2014) are the most notable methods. Sil and Yates (2013) first compile a liberal set of mention and entity candidates, and then perform joint ranking of the candidates. Durrett and Klein (2014) present a CRF model for coreference resolution, mention typing, and mention disambiguation. Our model is also based on CRF's, but distinguishes itself from prior work

in three ways:

- tree-shaped CRF's derived from dependency parse trees, as opposed to merely having connections among mentions and entity candidates;
- syntactic dependency features about verbal phrases from parse trees;
- the maintaining of candidates for both mentions and entities and jointly reasoning on their uncertainty.

Our experiments include comparisons with the method of Durrett and Klein (2014). There are also benchmarking efforts on measuring the performance for end-to-end NERD (Cornolti et al., 2013; Carmel et al., 2014; Usbeck et al., 2015), as opposed to assessing NER and NED separately. However, to the best of our knowledge, none of the participants in these competitions considered integrating NER and NED.

3.3 System Overview

We formalize the task as a sequence labeling problem which aims to label a sequence of *input tokens* $\mathbf{tok} = \langle tok_1, \ldots, tok_l \rangle$ with a sequence of *output labels* $\mathbf{y} = \langle y_1, \ldots, y_l \rangle$, consisting of NER types and NED entities (i.e., Yago entities). We employ a family of linear-chain and tree-shaped *probabilistic graphical models* (Koller et al., 2007) to compactly encode a multivariate probability distribution over *random variables* $\mathcal{X} \cup \mathcal{Y}$, where $\mathcal{X}$ denotes the set of variables x_i we may observe, and $\mathcal{Y}$ denotes the set of output labels y_i. By writing $\mathbf{x}$, we denote an *assignment of observed variables* to $\mathcal{X}$, while by writing $\mathbf{y}$, we denote an *assignment of labels* to $\mathcal{Y}$. In our running example:

"David played for manu, real, and la galaxy.",

"*David*" is the first observed variable x_1 with the desired label y_1 = `PERSON:David_Beckham` where `PERSON` denotes the NER type and `David_Beckham` is the entity of interest. Consecutive and identical labels are considered to be entity mentions. For example, for x_5 = "*la*" and x_6 = "*galaxy*", the output would ideally be y_5 = `ORGANIZATION:Los_Angeles_Galaxy` and y_6 = `ORGANIZATION:Los_Angeles_Galaxy`, denoting the soccer club. Upfront these are merely candidate labels, though. Our method may alternatively consider the different labels y_5 = `LOCATION:Los_Angeles` and y_6 = `MISC:Samsung_Galaxy`. This would yield incorrect output with two single-token mentions and improper entities.

Given an input text, J-NERD works in two stages. First, J-NERD harnesses a joint feature model for NER and NED. Some of the features are standard, while others are novel (see Section 3.4). Second, J-NERD constructs a factor graph model (i.e., *linear-chain model* or *tree-shaped model*) from input sentences. Subsequently, it jointly performs inference for both NER and NED (see Section 3.5).

Stage 1: Feature Model. J-NERD first processes the text through a standard NLP pipeline, including sentence detection, tokenization, POS tagging, lemmatization,

dependency parsing, and noun-phrase chunking. Consequently, it extracts a variety of features, each take the possible assignments $\mathbf{x}$, $\mathbf{y}$ of observed variables and labels, respectively, as input and give a binary value or real number as output. Binary values denote the presence or absence of a feature; real-valued ones typically denote frequencies of observed features.

Stage 2: Factor Graph Model. Next, J-NERD constructs a factor graph model (i.e., *linear-chain model* or *tree-shaped model*) for each input sentence. State-of-the-art NER methods, such as the Stanford NER Tagger, employ *linear-chain factor graph*, known as Conditional Random Fields (CRF's) (Sutton and McCallum, 2012). We also devise more sophisticated *tree-shaped factor graphs* whose structure is obtained from the dependency parse trees of the input sentences. These per-sentence models are optionally combined into a *global factor graph* by adding also cross-sentence dependencies (Finkel et al., 2005). These cross-sentence dependencies connects similar tokens which are potential co-references (e.g., "*David*" and "*David Beckham*"), or tokens in adjacent mentions in the same sentence (e.g., "*David*" and "*manu*"). Finally, J-NERD performs a joint inference by variants of the Viterbi algorithm or Gibbs sampling.

For tractability, probabilistic graphical models are typically constrained by making conditional independence assumptions, thus imposing structure and locality on $\mathcal{X} \cup \mathcal{Y}$. In our models, we postulate that the following conditional independence assumptions hold:

$$p(y_i \mid \mathbf{x}, \mathbf{y}) = p(y_i \mid \mathbf{x}, y_{prev(i)}) \tag{3.1}$$

That is, the label y_i for the i^{th} variable directly depends only on the label $y_{prev(i)}$ of some previous variables at position $prev(i)$ and potentially on all input variables. The case where $prev(i) = i-1$ is the standard setting for a linear-chain CRF, where the label of a variable depends only on the label of its preceding variable. We generalize this approach to considering $prev(i)$ based on the edges of a dependency parse tree and from co-references in preceding sentences.

By the Hammersley-Clifford Theorem, such a graphical model can be factorized into a product form where each factor captures a subset $A \subseteq \mathcal{X} \cup \mathcal{Y}$ of the random variables. Typically, each factor considers only those $\mathcal{X}$ and $\mathcal{Y}$ variables that are coupled by a conditional (in-)dependence assumptions, with overlapping A sets of different factors. The probability distribution encoded by the graphical model can then be expressed as follows:

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \prod_{A} \mathcal{F}_A(\mathbf{x}_A, \mathbf{y}_A) \tag{3.2}$$

Here, $\mathcal{F}_A(\mathbf{x}_A, \mathbf{y}_A)$ denotes the *factors* of the model, each of which is of the following form:

$$\mathcal{F}_A(\mathbf{x}_A, \mathbf{y}_A) = \exp\left\{ \sum_k \lambda_k f_{A,k}(\mathbf{x}_A, \mathbf{y}_A) \right\} \tag{3.3}$$

The normalization constant $Z = \sum_{\mathbf{x},\mathbf{y}} \prod_A \mathcal{F}_A(\mathbf{x}_A, \mathbf{y}_A)$ ensures that this distribution sums up to 1, while λ_k are the parameters of the model, which we aim to learn from various annotated background corpora.

Our inference objective then is to find the *most probable sequence of labels* $\mathbf{y}^*$ when given the token sequence $\mathbf{x}$ as evidence:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} p(\mathbf{y} \mid \mathbf{x}) \tag{3.4}$$

That is, in our setting, we fix $\mathbf{x}$ to the observed token sequence $\mathbf{tok} = \langle tok_1, \ldots, tok_l \rangle$, while $\mathbf{y} = \langle y_1, \ldots, y_l \rangle$ ranges over all possible sequences of associated labels. In our approach, which we hence coined $\mathcal{J}$-NERD, each y_i label represents a combination of NER type and NED entity.

To reduce the dimensionality of the generated feature space and to make the factor-graph inference tractable, we use pruning techniques based on the knowledge base and the dictionaries. To determine if a token can be a mention or part of a mention, we first perform lookups of all sub-sequences against the name-entity dictionary (see Section 2.2.2). As an option (and by default), this can be limited to sub-sequences that are tagged as noun phrases by the Stanford parser. For higher recall, we then add partial-match lookups when a token sub-sequence matches only some but not all tokens of an entity name in the dictionary. For example, for the sentence

"*David played for manu, real and la galaxy.*",

we obtain "*David*", "*manu*", "*real*", "*la galaxy*", "*la*", and "*galaxy*" as candidate mentions. For each such candidate mention, we look up the knowledge base for entities and consider only the best n (using $n = 20$ in our experiments) highest ranked candidate entities. The ranking is based on the string similarity between the mention and the entity name, the prior popularity of the entity, and the local context similarity.

The NER types that we consider are the standard types including `PERSON`, `ORGANIZATION` and `LOCATION`. All other types that, for example, the Stanford NER Tagger would mark, are collapsed into a type `MISC` for miscellaneous. These include labels like date and money (which are not genuine entities anyway) and also entity types like events and creative works such as movies, songs, etc. (which are disregarded by the Stanford NER Tagger). We add two dedicated tags for tokens to express the case when no meaningful NER type or NED entity can be assigned. For tokens that should not be labeled as a named entity at all (e.g., "*played*" in our example), we use the tag `OTHER`. For tokens with a valid NER type, we add the virtual entity `out-of-KB` (for "*out of knowledge base*") to its entity candidates, to prepare for the possible situation where the token (and its surrounding tokens) actually denotes an emerging or long-tail entity that is not contained in the background knowledge base (i.e., Yago). Once the output labels are determined, the actual boundaries of the mentions, i.e., their token spans, are trivially derived by combining adjacent tokens with the same label (and disregarding all tokens with the tag `OTHER`).

3.4 Feature Model

First, we employ the Stanford CoreNLP tool suite (Manning et al., 2014) for processing input documents. This includes sentence detection, tokenization, POS tagging, lemmatization, dependency parsing, and noun-phrase chunking. All of these provide features for our graphical model. Second, we define features for detecting the combined NER/NED labels of token that denote or are part of an entity mention. In the following, we introduce the complete set of features which includes the 7 NED features $f_{1..7}$ (defined in Chapter 2, Section 2.2) and 10 feature templates $f_{8..17}$ for the joint NERD inference. Templates are instantiated based on the observed input and the candidate space of possible labels for this input, and guided by distant resources like knowledge bases and dictionaries. The generated feature values depend on the assignment of input tokens to variables $x_i \in \mathcal{X}$. In addition, our graphical models often consider only a specific subset of *candidate labels* as assignments to the output variables $y_i \in \mathcal{Y}$. Therefore, we formulate the feature-generation process as a set of *feature functions* that depend on both (per-factor subsets of) $\mathcal{X}$ and $\mathcal{Y}$. Table 3.1 illustrates the feature generation by the set of active feature functions for the token "*manu*" in our running example, using three different candidate labels.

Table 3.1: Positive features for the token "*manu*" (x_3) with candidate labels `ORGANIZATION:Manchester_United_F.C.` (y_3), `PERSON:Manu_Chao` (y_3') and `OTHER` (y_3''). Domain is *Football*, and dependency pattern is *prep_for[played, *]*.

Feature	y_3	y_3'	y_3''
f_1: Mention-Entity Prior	✓	✓	
f_2: Mention-Entity n-Gram Similarity	✓	✓	
f_3: Mention-Entity Token Context Similarity	✓	✓	
f_4: Entity-Entity Token Context Similarity	✓	✓	
f_5: Domain-Entity Coherence	✓		
f_6: Entity-Entity Type Coherence	✓	✓	
f_7: Mention-Entity Dependency Context Similarity	✓		
f_8: Token-Type Prior	✓	✓	✓
f_9: Current POS	✓	✓	✓
f_{10}: In-Dictionary	✓	✓	
f_{11}: Uppercase			
f_{12}: Surrounding Tokens	✓	✓	✓
f_{13}: Surrounding POS	✓	✓	✓
f_{14}: Surrounding In-Dictionary	✓	✓	✓
f_{15}: Typed-Dependency	✓		✓
f_{16}: Typed-Dependency/POS	✓		✓
f_{17}: Typed-Dependency/In-Dictionary	✓		

3.4.1 NED Features

We harness NED features $f_{1..7}$ which include:

- standard features, namely Mention-Entity Prior, Mention-Entity n-Gram Similarity, Mention-Entity Token Context Similarity, and Entity-Entity Token Context Similarity,
- domain-oriented features, namely Domain-Entity Coherence and Entity-Entity Type Coherence,
- a syntactic dependency feature, namely Mention-Entity Dependency Context Similarity.

When computing actual values for these features at token level, we take the maximum values over all mention candidates of the current token.

3.4.2 NER Features

For the following definitions of the NER feature templates, let dic_i denote the NER tag from the dictionary lookup of tok_i (with POS tag pos_i), and dep_i denote the parsing dependency that connects tok_i with another token. Further, we write $sur_i = \langle tok_{i-1}, tok_i, tok_{i+1} \rangle$ to refer to the sequence of tokens surrounding tok_i. As for the possible labels, let $type_i$ denote an NER type for the current token tok_i.

Token-Type Prior. Feature $f_8(tok_i, type_i)$ captures a prior probability for tok_i being of NER type $type_i$. These probabilities are estimated from an NER-annotated training corpus (i.e., CoNLL in our experiments). For example, we may thus obtain a prior of $f_8(\text{“}Ltd.\text{”}, \texttt{ORGANIZATION}) = 0.8$.

Current POS. Template $f_9(tok_i, type_i)$ generates a binary feature function if token tok_i occurs in the training corpus with POS tag pos_i and NER label $type_i$. For example, $f_9(\text{“}David\text{”}, \texttt{PERSON}) = 1$ if the current token “*David*” has occurred with POS tag *NNP* and NER label `PERSON` in the training corpus. For combinations of tokens with POS tags and NER types that do not occur in the training corpus, no actual feature function is generated from the template (i.e., the value of function would be 0). For the rest of this section, we assume that all binary feature functions are generated from their feature templates in an analogous manner.

In-Dictionary. Template $f_{10}(tok_i, type_i)$ generates a binary feature function if the current token tok_i occurs in the name-entity dictionary for some entity of NER label $type_i$.

Uppercase. Template $f_{11}(tok_i, type_i)$ generates a binary feature function if the current token tok_i appears in upper-case form and additionally has the NER label $type_i$ in the training corpus.

Surrounding Tokens. Template $f_{12}(tok_i, type_i)$ generates a binary feature function if the current token tok_i has NER label $type_i$, given that tok_i also appears with surrounding tokens sur_i in the training corpus. When instantiated, this template could possibly lead to a huge number of feature functions. For tractability, we thus ignore sequences that occur only once in the training corpus.

Surrounding POS. Template $f_{13}(tok_i, type_i)$ generates a binary feature function if the current token tok_i and the POS sequence of its surrounding tokens sur_i both appear in the training corpus, where tok_i also has the NER label $type_i$.

Surrounding In-Dictionary. We derive additional NER-type-specific phrase dictionaries from supporting phrases of GATE (Cunningham et al., 2011), e.g., "*Mr.*", "*Mrs.*", "*Dr.*", "*President*", etc. for the type `PERSON`; "*city*", "*river*", "*park*", etc. for the type `LOCATION`; "*company*", "*institute*", "*Inc.*", "*Ltd.*", etc. for the type `ORGANIZATION`. Template $f_{14}(tok_i, type_i)$ performs dictionary lookups for surrounding tokens in sur_i. It generates a binary feature function if the current token tok_i and the dictionary lookups of its surrounding tokens sur_i appear in the training corpus, where tok_i also has NER label $type_i$.

The above $f_{8..14}$ features are widely used in prior NER systems. In the following, we introduce three advanced features based on syntactic dependency parsing. Specifically, they harness dependency types between noun-phrases such as ***nsubj***, ***dobj***, ***prep_in***, ***prep_for***, etc. to refine NER labels.

Typed-Dependency. Template $f_{15}(tok_i, type_i)$ generates a binary feature function if the background corpus contains the pattern $dep_i = deptype(arg1, arg2)$ where the current token tok_i is either *arg1* or *arg2*, and tok_i is labeled with NER label $type_i$.

Typed-Dependency/POS. Template $f_{16}(tok_i, type_i)$ captures linguistic patterns that combine parsing dependencies (like in f_{15}) and POS tags (like in f_9) learned from an annotated training corpus. It generates binary features if the current token tok_i appears in the dependency pattern dep_i with POS tag pos_i and this combination also occurs in the training data under NER label $type_i$.

Typed-Dependency/In-Dictionary. Template $f_{17}(tok_i, type_i)$ captures linguistic patterns that combine parsing dependencies (like in f_{15}) and dictionary lookups (like in f_{10}) learned from an annotated training corpus. It generates a binary feature function if the current token tok_i appears in the dependency pattern dep_i and has an entry dic_i in the name-entity dictionary for an entity with NER label $type_i$.

3.5 J-NERD Factor Graph Model

3.5.1 Linear-Chain Model

In the local models, J-NERD works on each sentence S: $\mathbf{tok} = \langle tok_1, \ldots, tok_l \rangle$ separately. We construct a linear-chain CRF (see Figure 3.1) by introducing an observed variable x_i for each token tok_i that represents a proper word. For each x_i, we additionally introduce a variable y_i that represents the combined NERD label. As in any CRF, the x_i, y_i and y_i, y_{i+1} pairs are connected via factors $\mathcal{F}(\mathbf{x}, \mathbf{y})$, whose weights we obtain from the feature functions described in Section 3.4.

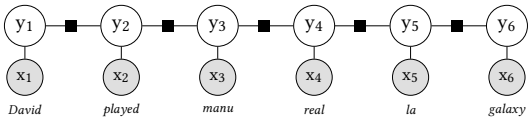

Figure 3.1: Linear-chain model (CRF).

3.5.2 Tree Model

The factor graph for the tree-shaped model is constructed in a similar way. However, here we add a factor that links a pair of labels y_i, y_j if their respective tokens tok_i, tok_j are connected via a typed dependency which we obtain from the Stanford parser. Figure 3.2 shows an example of such a tree model. Thus, while the linear-chain model adds factors between labels of adjacent tokens only based on their positions in the sentence, the tree model adds factors based on the dependency parse tree to enhance the coherence of labels across tokens.

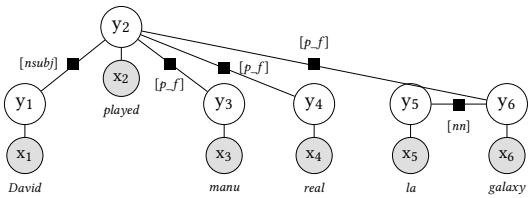

Figure 3.2: Tree model ([p_f] is [*prep_for*]).

3.5.3 Global Models

For global models, we consider an entire input text consisting of multiple sentences $S_1, \ldots, S_n$: **tok** = $\langle tok_1, \ldots, tok_l \rangle$, for augmenting either one of the *linear-chain model* or *tree-shaped model*. As shown in Figure 3.3, cross-sentence edges among pairs of labels y_i, y_j are introduced for the same observed tokens, such as the two tokens "*David*". Additionally, we introduce factors for all pairs of tokens in adjacent mentions within the same sentence, such as "*David*" and "*manu*".

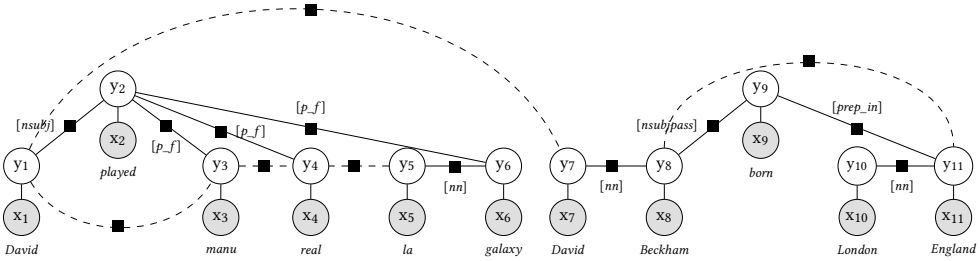

Figure 3.3: Global model, linking two tree models ([p_f] is [*prep_for*]).

3.5.4 Inference & Learning

Our *inference objective* is to find the most probable sequence of NERD labels:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \; p(\mathbf{y} \mid \mathbf{x}) \tag{3.5}$$

according to the objective function we defined in Section 3.3. Instead of considering the actual distribution:

$$p(\mathbf{x}, \mathbf{y}) = \frac{1}{Z} \prod_{A} \exp \left\{ \sum_{k} \lambda_k f_{A,k}(\mathbf{x}_A, \mathbf{y}_A) \right\} \tag{3.6}$$

for this purpose, we aim to maximize an equivalent objective function as follows. Each factor A in our model couples a label variable y_t with a variable $y_{prev(t)}$: either its immediately preceding token in the same sentence, or a parsing-dependency-linked token in the same sentence, or a co-reference-linked token in a different sentence. Each of these factors has its feature functions, and we can regroup these features on a per-token basis given the log-linear nature of the objective function. This leads to the following optimization problem which has its maximum for the same label assignment as the original problem:

$$\mathbf{y}^* = \arg\max_{y_1 \ldots y_l} \quad \exp \left(\sum_{t=1}^{l} \sum_{k=1}^{K} \lambda_k \, feature_k(y_t, y_{prev(t)}, x_1 \ldots x_l) \right) \tag{3.7}$$

where:

- $prev(t)$ is the index of label y_j on which y_t depends,
- $feature_{1..K}$ are the feature functions generated from templates f_1–f_{17} described in Section 3.4,
- and λ_k are the feature weights, i.e., the model parameters to be learned.

The actual number of generated features, K, depends on the training corpus and the choice of the graphical model. For example, the tree models have $K = 1,767$ parameters in our experiments. Given a trained model, exact inference with respect to the above objective function can be efficiently performed by variants of the Viterbi algorithm (Sutton and McCallum, 2012) for the local models, both in the linear-chain and tree-shaped cases. For the global models, however, exact solutions are computationally intractable. Therefore, we employ Gibbs sampling (Finkel et al., 2005) to approximate the solution.

As for the model parameters, J-NERD learns the feature weights λ_k from the training data by maximizing a respective conditional likelihood function (Sutton and McCallum, 2012), using a variant of the L-BFGS optimization algorithm (Liu and Nocedal, 1989). We do this for each local model (linear-chain and tree models), and apply the same learned weights to the corresponding global models. Our implementation uses the RISO toolkit[1] for belief networks.

[1] http://riso.sourceforge.net/

3.6 Experiments

3.6.1 Setup

Data Collections. Our evaluation is mainly based on the *CoNLL-YAGO2* corpus of newswire articles. Additionally, we report on experiments with an extended version of the *ACE-2005* corpus and a large sample of the entity-annotated *ClueWeb'09-FACC1* Web crawl.

CoNLL-YAGO2 is derived from the CoNLL-YAGO corpus (Hoffart et al., 2011a)[2] by removing tables where mentions in table cells do not have linguistic context; a typical example is sports results. The resulting corpus contains 1,244 documents with 20,924 mentions including 4,774 `out-of-KB` entities. Ground-truth entities in Yago are provided by Hoffart et al. (2011a). For a consistent ground-truth set, we derived the NER types from the NED ground-truth entities, fixing some errors in the original annotations related to metonymy (e.g., labeling the mentions in "*India beats Pakistan 2:1*" incorrectly as `LOCATION`, whereas the entities are the sports teams of type `ORGANIZATION`). This makes the dataset not only cleaner but also more demanding, as metonymous mentions are among the most difficult cases. For our evaluation, we use the "testb" subset of CoNLL-YAGO, which – after the removal of tables – has 199 documents with 3,054 mentions including 717 `out-of-KB` entities. The other 1,045 documents with a total of 17,870 mentions (including 4,057 `out-of-KB` mentions) are used for training.

ACE is an extended variant of the ACE 2005 corpus[3], with additional NED labels by Bentivogli et al. (2010). We consider only proper entities (i.e., the intersection of Wikipedia articles and Yago entities) and exclude mentions of general concepts such as "*revenue*", "*world economy*", "*financial crisis*", etc., as they do not correspond to individual entities in a knowledge base. This reduces the number of mentions, but gives the task a crisp focus. We disallow overlapping mention spans and consider only maximum-length mentions, following the rationale of the ERD Challenge 2014. The test set contains 117 documents with 2,958 mentions.

ClueWeb contains two randomly sampled subsets of the ClueWeb'09-FACC1[4] corpus with Freebase annotations:

- *ClueWeb*: 1,000 documents (24,289 mentions) each with at least 5 entities.
- $ClueWeb_{long-tail}$: 1,000 documents (49,604 mentions) each with at least 3 long-tail entities. We consider an entity to be "*long-tail*" if it has at most 10 incoming links in the English Wikipedia.

These Web documents are very different in style from the news-centric articles in CoNLL and ACE. Also note that the entity markup is automatically generated, but with emphasis on high precision. So the data captures only a small subset of the potential entity mentions, and it may contain a small fraction of false entities.

[2] https://www.mpi-inf.mpg.de/yago-naga/yago/
[3] http://projects.ldc.upenn.edu/ace/
[4] http://lemurproject.org/clueweb09/FACC1/

Methods under Comparison. We compare J-NERD in its four variants (*linear* vs. *tree* and *local* vs. *global*) to various state-of-the-art NER/NED methods.

For NER (i.e., mention boundaries and types) we use the recent version 3.4.1 of the Stanford NER Tagger[5] (Finkel et al., 2005) and the recent version 2.8.4 of the Illinois Tagger[6] (Ratinov and Roth, 2009) as baselines. These systems have NER benchmark results on CoNLL'03 that are as good as the result reported in Passos et al. (2014). We *retrained* this model by using the same corpus-specific training data that we use for J-NERD .

For NED, we compared J-NERD against the following methods for which we obtained open-source software or could call a Web service:

- Berkeley-entity (Durrett and Klein, 2014) uses a joint model for coreference resolution, NER and NED with linkage to Wikipedia.
- TagMe (Ferragina and Scaiella, 2010) is a Wikifier that maps mentions to entities or concepts in Wikipedia. It uses a Wikipedia-derived dictionary for NER.
- Spotlight (Mendes et al., 2011) links mentions to entities in DBpedia. It uses the LingPipe dictionary-based chunker for NER.

Some systems use confidence thresholds to decide on when to map a mention to `out-of-KB` entity. For each dataset, we used withheld data to tune these system-specific thresholds. Figure 3.4 illustrates the sensitivity of the thresholds for the CoNLL-YAGO2 dataset.

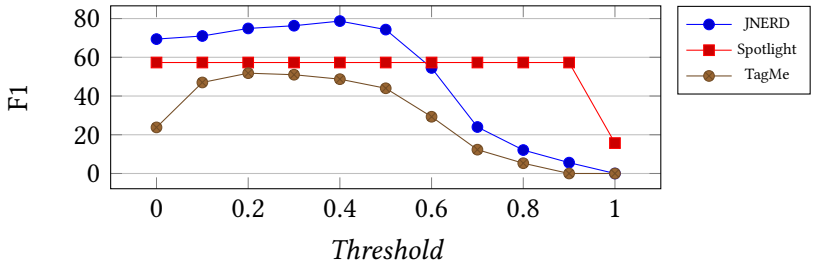

Figure 3.4: F1 for varying confidence thresholds.

Evaluation Measures. We evaluate the output quality at the NER level alone and for the end-to-end NERD task. We do not evaluate NED alone, as this would require giving a ground-truth set of mentions to the systems to rule out that NER errors affect NED. Most competitors do not have interfaces for such a controlled NED-only evaluation. Each test collection has ground-truth annotations (G) consisting of text spans for mentions, NER types of the mentions, and mapping mentions to entities in Yago or to `out-of-KB`. Recall that the `out-of-KB` case captures entities that are not in the KB at all. Let X be the output of system X: detected mentions, NER types, NED mappings. Following the ERD 2014 Challenge (Carmel et al., 2014), we define precision and recall of X for end-to-end NERD as:

[5] `nlp.stanford.edu/software/CRF-NER.shtml`
[6] `http://cogcomp.cs.illinois.edu/`

$$Precision(X) = |X \text{ agrees with } G|/|X| \tag{3.8}$$

$$Recall(X) = |X \text{ agrees with } G|/|G| \tag{3.9}$$

where agreement means that X and G overlap in the text spans (i.e., have at least one token in common) for a mention, have the same NER type, and have the same mapping to an entity or `out-of-KB`. The F1 score of X is the harmonic mean of precision and recall. For evaluating the mention-boundary detection alone, we consider only the overlap of text spans; for evaluating NER completely, we consider both mention overlap and agreement based on the assigned NER types.

3.6.2 Results for CoNLL-YAGO2

Our first experiment on CoNLL-YAGO2 is comparing the four CRF variants of J-NERD for three tasks: mention boundary detection, NER typing and end-to-end NERD. Then, the best model of J-NERD is compared against various baselines and a pipelined configuration of our method. Finally, we test the influence of different features groups.

3.6.2.1 Experiments on CRF Variants

Table 3.2 compares the different CRF variants. All CRFs have the same features, but differ in their factors. Therefore, some features are not effective for the linear model and the tree model. For the linear CRF, the parsing-based linguistic features and the cross-sentence features do not contribute; for the tree CRF, the cross-sentence features are not effective.

Table 3.2: Experiments on CRF Variants.

Perspective	Variants	Precision	Recall	F1
Mention Boundary Detection	$\mathcal{J}$-NERD$_{linear\text{-}local}$	94.2	89.6	91.8
	$\mathcal{J}$-NERD$_{tree\text{-}local}$	94.4	89.4	91.8
	$\mathcal{J}$-NERD$_{linear\text{-}global}$	95.1	90.3	92.6
	$\mathcal{J}$-NERD$_{tree\text{-}global}$	95.8	90.6	93.1
NER Typing	$\mathcal{J}$-NERD$_{linear\text{-}local}$	87.8	83.0	85.3
	$\mathcal{J}$-NERD$_{tree\text{-}local}$	89.5	82.2	85.6
	$\mathcal{J}$-NERD$_{linear\text{-}global}$	88.6	83.4	85.9
	$\mathcal{J}$-NERD$_{tree\text{-}global}$	90.4	83.8	86.9
End-to-End NERD	$\mathcal{J}$-NERD$_{linear\text{-}local}$	71.8	74.9	73.3
	$\mathcal{J}$-NERD$_{tree\text{-}local}$	75.1	74.5	74.7
	$\mathcal{J}$-NERD$_{linear\text{-}global}$	77.6	74.8	76.1
	$\mathcal{J}$-NERD$_{tree\text{-}global}$	81.9	75.8	78.7

We see that all variants perform very well on boundary detection and NER typing, with small differences only. For end-to-end NERD, however, $\mathcal{J}$-NERD$_{tree\text{-}global}$ outperforms all other variants by a large margin. This results in achieving the best F_1 score of 78.7%, which is 2.6% higher than $\mathcal{J}$-NERD$_{linear\text{-}global}$. We performed a paired t-test between these two variants, and obtained a p-value of 0.01. The local variants of $\mathcal{J}$-NERD lose around 4% of F1 because they do not capture the coherence among mentions in different sentences. In the rest of our experiments, we focus on $\mathcal{J}$-NERD$_{tree\text{-}global}$ and the task of end-to-end NERD.

3.6.2.2 Comparison of Joint vs. Pipelined Models and Baselines

In this subsection, we demonstrate the benefits of joint models against pipelined models including state-of-the-art baselines. Additionally, we add a pipelined configuration of J-NERD , coined *P*-NERD. That is, we first run J-NERD in NER mode (thus only considering NER features $f_{8..17}$. The best sequence of NER labels is then given to J-NERD to run in NED mode (only considering NED features $f_{1..7}$).

Table 3.3: Comparison between joint models and pipelined models on end-to-end NERD.

Method	Precision	Recall	F1
P-NERD	80.1	75.1	77.5
$\mathcal{J}$-NERD	81.9	75.8	78.7
TagMe	64.6	43.2	51.8
SpotLight	71.1	47.9	57.3

The results are shown in Table 3.3. $\mathcal{J}$-NERD achieves the highest precision of 81.9% for end-to-end NERD, outperforming all competitors by a significant margin. This results in achieving the best F1 score of 78.7%, which is 1.2% higher than *P*-NERD TagMe and Spotlight are clearly inferior on this dataset (more than 20% lower in F_1 than $\mathcal{J}$-NERD).These systems are more geared towards efficiency and coping with popular and thus frequent entities, whereas the CoNLL-YAGO2 dataset contains very difficult test cases. For the best F1 score of $\mathcal{J}$-NERD, we performed a paired t-test against the other methods' F_1 values and obtained a p-value of 0.075.

We also compared the NER performance of J-NERD against the state-of-the-art method for NER alone, the Stanford NER Tagger version 3.4.1 and the Illinois Tagger 2.8.4 (Table 3.4). For mention boundary detection, J-NERD achieved an F_1 score of 93.1% versus 93.4% by Stanford NER, 93.3% by Illinois Tagger, and 92.9% by *P*-NERD. For NER typing, J-NERD achieved an F_1 score of 86.9% versus 86.8% by Stanford NER, 85.3% by Illinois Tagger, and 86.3% by *P*-NERD. So we could not outperform the best prior method for NER alone, but achieved very competitive results. Here, we do not really leverage any form of joint inference (combining CRF's across sentences is used in Stanford NER, too), but harness rich features on domains, entity candidates, and linguistic dependencies.

Table 3.4: Experiments on NER against state-of-the-art NER systems.

Perspective	Variants	Precision	Recall	F1
Mention Boundary Detection	$\mathcal{P}$-NERD	95.6	90.5	92.9
	$\mathcal{J}$-NERD	95.8	90.6	93.1
	Stanford NER	95.6	91.3	93.4
	Illinois Tagger	95.5	91.2	93.3
NER Typing	$\mathcal{P}$-NERD	89.6	83.4	86.3
	$\mathcal{J}$-NERD	90.4	83.8	86.9
	Stanford NER	89.3	84.5	86.8
	Illinois Tagger	87.5	83.2	85.3

3.6.2.3 Influence of Features

To analyze the influence of the features, we performed an additional ablation study on the global J-NERD tree model, which is the best variant of J-NERD , as follows:

- *Standard features* excludes the domain features (i.e., f_5 and f_6) and the dependency features (i.e., f_7, f_{15}, f_{16}, and f_{17}).
- *Standard and domain features* excludes the syntactic dependency features (i.e., f_7, f_{15}, f_{16}, and f_{17}).
- *Standard and dependency features* excludes the domain-oriented features (i.e., f_5 and f_6).
- *All features* is the full-fledged J-NERD$_{tree\text{-}global}$ model.

Table 3.5: Feature Influence on CoNLL-YAGO2.

Perspective	Setting	F1
NER Typing	Standard features	85.1
	Standard and domain features	85.7
	Standard and linguistic features	86.4
	All features	86.9
End-to-End NERD	Standard features	74.3
	Standard and domain features	76.4
	Standard and linguistic features	76.6
	All features	78.7

Table 3.5 shows the results, demonstrating that linguistic features are crucial for both NER and NERD. For example, in the sentence "*Woolmer played 19 tests for England*", the mention "*England*"refers to an organization (the English cricket team), not to a location. The dependency-type feature ***prep_for****[play, *]* is a decisive cue to handle such cases properly. Domain features help in NED to eliminate, for example, football teams when the domain is *Cricket*.

3.6.3 End-to-End NERD on ACE

For comparison to the recently developed Berkeley-entity system (Durrett and Klein, 2014), the authors of that system provided us with detailed results for the entity-annotated ACE'2005 corpus, which allowed us to discount non-entity (so-called "*NOM-type*") mappings. All other systems, including the best J-NERD method, were run on the corpus under the same conditions.

Table 3.6: NERD results on ACE.

Method	Precision	Recall	F1
P-NERD	68.2	60.8	64.2
$\mathcal{J}$-NERD	69.1	62.3	65.5
Berkeley-entity	65.6	61.8	63.7
TagMe	60.6	43.5	50.7
SpotLight	68.7	29.6	41.4

J-NERD outperforms *P*-NERD and Berkeley-entity: F1 scores are 1.3% and 1.8% better, respectively, with a t-test p-value of 0.05 (Table 3.6). The others show substantially inferior performance. The performance gains that J-NERD achieves over Berkeley-entity can be attributed to two factors. First, the rich linguistic features of J-NERD help to correctly cope with more of the difficult cases, e.g., when common nouns are actually names of people. Second, the coherence features of global J-NERD help to properly couple decisions on related entity mentions.

3.6.4 End-to-End NERD on ClueWeb

The results for ClueWeb are shown in Table 3.7. J-NERD outperforms all other systems with a t-test p-value of 0.05. The differences between J-NERD and fast NED systems such as TagMe or SpotLight become smaller as the number of prominent entities (i.e., prominent people, organizations and locations) is higher on ClueWeb than on CoNLL-YAGO2.

Table 3.7: NERD results on ClueWeb.

Dataset	Method	Precision	Recall	F1
ClueWeb	*P*-NERD	80.9	67.1	73.3
	$\mathcal{J}$-NERD	81.5	67.5	73.8
	TagMe	78.4	60.5	68.3
	SpotLight	79.7	57.1	66.5
$\text{ClueWeb}_{long-tail}$	*P*-NERD	81.2	64.4	71.8
	$\mathcal{J}$-NERD	81.4	65.1	72.3
	TagMe	78.4	58.3	66.9
	SpotLight	81.2	56.3	66.5

3.7 Summary

In this chapter, we have shown that coupling the tasks of NER and NED in a joint CRF-like model is beneficial. Our method outperforms strong baselines on a variety of test datasets. The strength of J-NERD comes from three novel assets. First, our tree-shaped models capture the structure of dependency parse trees, and we couple multiple such tree models across sentences. Second, we harness non-standard features about domains and novel features based on dependency patterns derived from parsing. Third, our joint inference maintains uncertain candidates for both mentions and entities and makes decisions as late as possible.

Chapter 4

Joint Model for Relation Extraction and Entity Disambiguation

4.1 Introduction and Related Work

Motivation. Information extraction (IE) aims to distill relational triples, each consisting of an entity pair (or an entity and a literal value) plus a connecting relation, from natural-language text. This goal has been pursued by two major approaches. *Model-based IE* (Brin, 1998; Craven et al., 2000; Mintz et al., 2009; Suchanek et al., 2009; Carlson et al., 2010; Singh et al., 2013) focuses on a set of pre-specified relations, like those present in a knowledge base (KB) such as DBpedia (Auer et al., 2007), NELL (Carlson et al., 2010) or Yago (Suchanek et al., 2007). Entity names recognized in the input text are disambiguated by mapping them to proper entities in the KB. The relations have fine-grained type signatures that need to be matched by the assigned entities. Model-based IE techniques leverage this strategy to achieve high precision, but they are inherently limited in recall by the relatively small amount of given relations. *Open IE* (Banko et al., 2007; Etzioni et al., 2011; Mausam et al., 2012), on the other hand, extracts triples of surface phrases and thus achieves higher recall. It can potentially find any relation that holds between two arguments. However, the arguments are not canonicalized, and the resulting relations are often noisy.

Example. Consider the following sentences:

1) *Amy received the Oscar for the best documentary.*
2) *Amy received the Grammy for the best new artist.*
3) *Amy received her degree in neurobiology from Harvard.*
4) *Simone received two honorary degrees in music and humanities, from the University of Massachusetts Amherst and Malcolm X College.*

Sentences (1), (2) and (3) refer to different entities—the movie, the singer and the movie character—, which are all named "*Amy*". The sentences provide cues for identifying the lexical types, but no existing IE method can robustly handle such cases. Moreover, (3) and (4) express the same relation (i.e., "`receive_degree_from`") by different phrases, but Open IE would treat them as separate relations.

State-of-the-Art & Limitations. Recent work aimed to reconcile Model-based IE and Open IE. Hoffmann et al. (2010) present a distantly supervised IE system which can handle thousands of relations by clustering the relational paraphrases based on their arguments and type signatures (e.g., using 1,282 such clusters in an experiment). However, this approach is highly customized to using Wikipedia infoboxes as input. Galárraga et al. (2014) canonicalize Open IE triples by clustering noun phrases into arguments and verbal phrases into relations. This approach is limited, though, to a few hundred relation clusters by using Freebase (Bollacker et al., 2008) as backend for the relations. Li and Ji (2014), on the other hand, jointly extract entity and relation names, but refrain from disambiguating these.

None of the prior works considers *joint inference* to extract relations and to disambiguate the entities in one step. Instead, all of the prior works use pipelined architectures and, thus, cannot fully harness the coupling of lexical types for entities with the type signatures for relations. For example, to properly distinguish the entities *Amy_Winehouse* and *Amy_Farrah_Fowler* in the above examples (2) and (3), understanding the different type signatures of the relations "`receive_prize`" (`SINGER` × `MUSIC AWARD`) and "`receive_degree_from`" (`PERSON` × `UNIVERSITY`) is crucial. The approach proposed in this chapter can leverage these kinds of interdependencies.

Contributions. This chapter presents J-REED (Nguyen et al., 2017), a joint model for entity disambiguation and relation extraction for Wikipedia-style input texts. J-REED is based on *probabilistic graphical models* that captures the interdependencies between entities and relations. Specifically, by considering which lexical types of entities are compatible with the type signature of which relation, we can boost the accuracy of both sub-tasks. Entity names are mapped to the entities registered in a background knowledge base (using DBpedia in our experiments), while relation patterns are extracted as crisp as possible. We performed large-scale experiments with 1.2 million Wikipedia pages about entities of type `PERSON` and obtained 9.5 million triples with ca. 80% accuracy. J-REED consistently outperforms pipelined combinations of OLLIE (Mausam et al., 2012), a state-of-the-art Open IE system, and recent NED systems such as Babelfy (Moro et al., 2014) and Spotlight (Daiber et al., 2013).

4.2 System Overview

J-REED processes a text corpus in several steps. We first pass all documents through a standard NLP pipeline, including tokenization, POS tagging, dependency parsing, NER tagging, and a customized noun-phrase chunker. Specifically, we employ the Stanford CoreNLP tool suite for all of our text processing steps except for dependency parsing. For the latter, we use the MaltParser (Nivre and Hall, 2005) which is more efficient than the Stanford parser. Mention names in the text are primarily recognized by the Stanford NER (Finkel et al., 2005) tagger.

In addition, we implement a custom noun-phrase chunker (using a set of regular expressions over the POS tags) to also extract noun-phrases that do not overlap with any names extracted by the NER tagger. We remove noise among the obtained noun-phrases by keeping only those phrases that contain at least one *informative noun*, which we define to consist of the top 5% most frequent nouns in the current document that is processed. For example, within the Wikipedia article of `Amy_Winehouse`, the most informative nouns contain "*album*", "*alcohol*", etc. Thus, the noun phrase "*alcohol poisoning*" (i.e., the cause of her death) is considered as a mention even if the NER tagger missed this phrase.

By applying these preprocessing steps to a dedicated *development corpus* (which is disjoint from the collection of test documents used in our experiments), we also compute various (co-)occurrence statistics among nouns, verbs, prepositions and entities. These statistics are later used to mine the *relation patterns* (Section 4.3), and they further serve as input to the feature functions used for *relation pattern labeling* (Section 4.4), and *joint relation extraction and entity disambiguation* (Section 4.5).

4.3 Relation Pattern Mining

J-REED considers four types of relation patterns: verb (e.g., *marry*), verb-noun (e.g., *win prize*), verb-preposition (e.g. *play for*) and verb-noun-preposition (e.g., *win prize for*). Nouns, prepositions and verbs in *active voice* are considered in their lemmatized forms (e.g., *marry*). Verbs in *passive voice* are represented by their past participle (e.g., *married to*) to capture the inverse direction of the relationship. J-REED considers only *frequent relation patterns* occurring at least τ times in the development corpus. In our experiments, we set $\tau = 100$ and obtained 9,248 frequent patterns (out of 320,143 distinct ones). By only considering those patterns as relational candidates for the graphical models in the next sections, J-REED can extract concise relation patterns. For example, J-REED extracts the relation pattern "*receive degree from*" instead of the surface phrase "*received honorary degrees in music and humanities from*" which is considered by many other Open IE methods. This is useful for further applications such as KB construction, question answering, and others.

4.4 Relation Pattern Labeling

To extract a relation pattern from a sentence in the test corpus, we consider a dependency path between two mentions as a *sequence of tokens* $\mathbf{tok} = \langle tok_1, \ldots, tok_l \rangle$. Thus, extracting relations from a sentence may be seen as a sequence labeling task in which we aim to find the best *sequence of labels* $\mathbf{y} = \langle y_1, \ldots, y_l \rangle$ by using four labels:

- `N` for a chosen noun,
- `V` for a chosen verb,

- `P` for a chosen preposition,
- `O` for "other".

Each relation pattern must contain one `V` label with at most one `N` label and with at most one `P` label. To improve recall, J-REED considers two further heuristics.

- If only one `N` label is returned under the text pattern *NAME-LEFT's noun NAME-RIGHT* (e.g. "*Mary's son Bill Gates*"), we add "*have*" with label `V` to the beginning of the sequence (e.g., "*have son*").
- If no `V` label is returned after an apposition, we add "*be*" with label `V` to the beginning of the sequence (e.g., "*be daughter of*").

A linear-chain CRF model is built to obtain the labels, which we can then map to the relation patterns mined from the development corpus. The feature set consists of:

- token string tok_i,
- part-of-speech tag pos_i,
- upper/lower case,
- being the first verb in the sequence **tok**,
- being the last preposition in the sequence **tok**,
- following the preposition "*to*",
- following a verb,
- following any token with POS tag "*NNP*",
- previous label y_{i-1},

These features have been widely used in prior work on Open IE (Banko et al., 2007; Mausam et al., 2012). The CRF is trained by maximizing a conditional likelihood function (Sutton and McCallum, 2012) via L-BFGS optimization. Exact inference is feasible by a variant of the Viterbi dynamic-programming algorithm (Bellman, 2003). We remark that if we only considered the most likely relation candidate, our method would resemble a traditional Open IE task. However, in our joint model described in the next section, all relation pattern candidates (together with their weights) are considered as features.

4.5 Joint Model

For the joint relation pattern extraction and entity disambiguation, J-REED considers an input triple consisting of a *pair of mentions* $\mathbf{m} = \langle m_1, m_2 \rangle$ together with the *sequence of tokens* $\mathbf{tok} = \langle tok_1, \ldots, tok_l \rangle$ along the dependency path that connects these mentions. The output of J-REED consists of a *pair of entities* $e_1 \in \mathbf{e_1}$, $e_2 \in \mathbf{e_2}$ and a *relation* $r \in \mathbf{r}$ that represents the semantic relationship between e_1 and e_2. Relations in $\mathbf{r}$ are obtained from the output of the CRF model described in Section 4.4, while entity candidates in $\mathbf{e_1}$ and $\mathbf{e_2}$ are obtained from the name-entity dictionary which is based on surface forms and hyperlinks in Wikipedia. To improve recall, we enrich this dictionary by including also the first and last names of all person entities in Wikipedia.

Model. J-REED considers five features: one feature for relation extraction (namely, *relation prior*), three features for entity disambiguation (namely, f_1: *mention-entity prior*, f_3: *mention-entity token context similarity*, and f_4: *entity-entity token context similarity* described in Section 2.2) and one feature for the joint inference (namely, *type signature*). We remark that some NED features such as the domain-oriented feature and the syntactic dependency feature are not considered by J-REED as the former is less effective in Wikipedia-style texts, and the latter is embedded in type signature. The *type signature* feature captures type dependencies between entities (i.e., arguments) and relation patterns among those entities. This feature is used in some prior joint models of NED and Information Extraction which focus on a small set of pre-defined relations. Here, we further extend this technique by harnessing the type signatures for thousands of relations. Specifically, we define the *type signature* as the relative frequency at which the semantic types of two entities occur under a relation pattern. We obtain these frequency statistics from the development corpus. In addition to the four general NER types `PERSON`, `ORGANIZATION`, `LOCATION`, and `MISC`, we use frequent infobox types[1] that have at least 1,000 articles in Wikipedia. From the resulting 167 types, we also manually derive a subsumption hierarchy (e.g. `FOOTBALLER` $\subseteq$ `SPORTS-PERSON` $\subseteq$ `PERSON`).

These features are combined into the following *objective function*

$$\langle e_1, e_2, r\rangle^* = \arg\max_{\langle e_1,e_2,r\rangle} \sum_{i=1}^{5} \alpha_i f_i(\mathbf{m}, \mathbf{tok}, e_1, e_2, r) \tag{4.1}$$

where the f_i are the features described above and the α_i are the parameters of our joint model.

Training & Inference. The α_i parameters are learned by maximizing the probability of the ground-truth annotations under L-BFGS optimization. At extraction time, exact inference is performed by dynamic programming (Bellman, 2003).

4.6 Experiments

4.6.1 Corpora

J-REED is applicable to any corpus with entity-centric documents, such as homepages of people, companies, etc. In our experiments we focused on Wikipedia articles about people. We remark that this does not restrict J-REED to `PERSON` entities. J-REED extracts various types of entities (e.g., `PERSON`, `MOVIE`, `AWARD`, etc.), and various types of relations (e.g., "*receive award*": `PERSON` x `AWARD`).

Development & Training Corpus. For the various stages of training, tuning and gathering statistics, we considered a collection of 1,215,956 Wikipedia articles about `PERSON` entities (based on types in Yago (Hoffart et al., 2011a)) from the 01/2015 English Wikipedia dump. 80% of these articles with 19,287,432 triples

[1] https://en.wikipedia.org/wiki/Wikipedia:List_of_infoboxes

from 8,312,439 sentences, called "*Wikipedia-develop*", were used to develop J-REED (i.e., to compute various (co-)occurrence statistics for nouns, verbs, prepositions and entities). In addition, we manually annotated 162 sentences from 5 of these Wikipedia articles, referred to as "*J-REED-train*", about prominent person entities, namely, Andrew Ng, Angela Merkel, David Beckham, Larry Page and Paris Hilton (covering scientists, politicians, sports stars, business people and actors/singers/ celebrities). These annotations comprise 203 triples, each consisting of two DBpedia entities and a sequence of `N`, `V`, `P`, `O` labels along the dependency path that connects the two entities. *J-REED-train* serves to train the CRF model (Section 4.4) and the joint model (Section 4.5).

Test Corpus. The remaining 20% of the 1,215,956 articles, called "*Wikipedia-test*", were used for evaluating our methods. We extracted 5,964,464 triples from 2,226,433 sentences from these articles.

Assessment. We had two judges who assessed our experimental results independently. They were shown sampled facts (i.e., two entities and a relation) from the output of different methods, along with the source sentence from which the fact was extracted. The judges were asked to label the fact as *true* only if all three components were correct, otherwise the fact was labeled as *false*. We observed strong inter-judge agreement. Cohen's kappa (Carletta, 1996) was 0.7.

4.6.2 Systems under Comparison

We ran experiments with several J-REED variants to compare against the state-of-the-art Open IE approach (i.e., OLLIE (Mausam et al., 2012)), and named entity disambiguation (NED) tools (i.e., Babelfy (Moro et al., 2014) and Spotlight (Daiber et al., 2013)). We chose Spotlight and Babelfy due to their focus on DBpedia entities and their ability to process the output of an Open IE system. Other NED tools, such as TagMe (Ferragina and Scaiella, 2010), only work on plain text as input. Specifically, we compared the following *end-to-end* IE methods:

- J-REED is the joint model described in Section 4.5. We heuristically resolve pronouns (i.e., "*he*" and "*she*") by assuming that these always refer to the main entity of the article. This assumption is based on the common Wikipedia writing style (and carries over to other kinds of entity-centric documents such as people's homepages).
- J-REED-pipeline considers only the most likely relation label based on the CRF model described in Section 4.4. In other words, the relation is fixed before the entities are disambiguated. Pronoun resolution is considered as described above.
- OLLIE-Spotlight is a pipelined approach combining OLLIE and Spotlight.
- OLLIE-Babelfy is a pipelined approach combining OLLIE and Babelfy.

We also perform an ablation test with three settings:

- J-REED-nopronoun performs joint entity-relation disambiguation but omits pronoun resolution.

- J-REED-notype omits the *type signature* feature.
- J-REED-noprior omits the relation *prior* feature.

4.6.3 Experiments on Relation Extraction

We evaluated the *precision* on two randomly sampled subsets of extracted triples, by manually assessing the 100 most confident results and 100 randomly sampled results for Wikipedia-test , respectively. Precision for each setting is reported as the mean of a Wald interval at 95% confidence level. Moreover, we measured *recall* as the absolute number of relational triples extracted. This value implicitly reflects relative recall which is generally hard to estimate for a large-scale experiment. For both precision and recall, we considered only extractions that consisted of up to 6 tokens. As shown in Table 4.1, J-REED outperforms both baselines in terms of both quality and number of extractions. J-REED-pipeline loses about 2% in precision compared to J-REED . Many mistakes of OLLIE originate from ignoring type constraints of relations. We remark that, although the authors of OLLIE address this problem in their work (Mausam et al., 2012), this feature is apparently turned off in their prototype software to increase recall.

Table 4.1: Experiments on Relational Triple Extraction (confidence at 95%).

Method	Precision		#Extracts
	Top-100	Random-100	
J-REED	0.91 ± 0.05	0.90 ± 0.05	1,931,462
J-REED-pipeline	0.89 ± 0.06	0.86 ± 0.06	1,931,462
OLLIE	0.86 ± 0.06	0.80 ± 0.07	1,646,231

4.6.4 Experiments on Entity Disambiguation

In a similar manner as for the relational fact extraction, we also evaluated the precision and recall of the entity disambiguation. For this, we considered 1,931,462 triples as output from J-REED (see Subsection 4.6.3). As the other NED systems – Spotlight and Babelfy – do not map pronouns to entities, we ignored extractions containing pronouns. An extraction is considered to be correct only if both entities are disambiguated correctly. Table 4.2 shows the results. J-REED outperforms other methods. The overall difference in precision among the systems is not that large, however. Particularly among the top 100 most confident results, all systems achieve precision values of about 90%, as these results often come from full name mentions which are rather straightforward to disambiguate. The numbers of disambiguated entity pairs are different for the systems, since we do not consider *null* entities in the final results.

Table 4.2: Experiments on Entity Disambiguation (confidence at 95%).

Method	Precision		#Extracts
	Top-100	Random-100	
J-REED	0.94 ± 0.04	0.85 ± 0.06	1,931,462
J-REED-pipeline	0.93 ± 0.05	0.82 ± 0.07	1,931,462
`Spotlight`	0.92 ± 0.05	0.83 ± 0.07	1,036,319
`Babelfy`	0.89 ± 0.06	0.81 ± 0.07	854,159

4.6.5 End-to-End Experiments

For this setting, we evaluated the correctness of entire facts (i.e., relation triples). In analogy to the above experiments, a fact extraction was considered correct only if all of its three components (i.e., the relation and the two entities) were correctly disambiguated. Here, we distinguished three types of assessments:

- *True* for a correct result,
- *False* for a wrong result (either the entities or the relation was wrong),
- *Ignored* for OLLIE-Spotlight and OLLIE-Babelfy when OLLIE returns a relation that consists of more than 6 tokens (i.e., normally noise).

Ignored results are not considered for precision and recall.

Ablation Test. As shown in Table 4.3, the type signature feature and relation prior features are crucial for the precision of J-REED. Disabling them results in a drop of precision by up to 6%. Disabling pronoun resolution penalizes the coverage by 40%. Additionally, we conduct experiments on the two heuristics for resolving possessive forms and appositions from Section 4.4. Disabling each of them penalizes the coverage by 6% and 12%, respectively.

Table 4.3: Ablation Test on Fact Extraction (confidence at 95%).

Method	Precision		#Extracts
	Top-100	Random-100	
J-REED	0.86 ± 0.06	0.78 ± 0.08	1,931,462
J-REED-notype	0.83 ± 0.07	0.72 ± 0.08	1,931,462
J-REED-noprior	0.84 ± 0.07	0.75 ± 0.08	1,931,462
J-REED-nopronoun	0.86 ± 0.06	0.80 ± 0.07	1,237,352

Comparison to Baselines. As shown in Table 4.4, the J-REED variants clearly outperform OLLIE-Spotlight and OLLIE-Babelfy. J-REED achieves around 2% higher precision than J-REED-pipeline . J-REED processes all 1.2 Million Wikipedia pages in about five hours. Our competitors require more than a day to process the same data.

Table 4.4: Experiments on Fact Extraction (confidence at 95%).

Method	Precision		#Extracts
	Top-100	Random-100	
J-REED	0.86 ± 0.06	0.78 ± 0.08	1,931,462
J-REED-pipeline	0.84 ± 0.07	0.73 ± 0.08	1,931,462
OLLIE-Spotlight	0.80 ± 0.07	0.68 ± 0.09	690,409
OLLIE-Babelfy	0.80 ± 0.07	0.67 ± 0.09	547,031

4.7 Summary

In this chapter, we present J-REED, a large-scale and high-quality information extraction (IE) system for Wikipedia-style text. Its unique strength is that it jointly runs inference for two core IE tasks: relation extraction and named-entity disambiguation by leveraging semantic types. Extractions by J-REED are more informative than by Open IE, as we canonicalize entities to a knowledge base. Running J-REED on 1,215,956 Wikipedia `PERSON` pages yields 9,577,301 facts with a precision of around 80%.

Chapter **5**

On-the-Fly Knowledge Base Construction

5.1 Introduction and Related Work

Motivation & Problem Setting. Knowledge bases, KBs for short, contain subject-predicate-object triples about entities and their properties. Popular KBs include DBpedia (Auer et al., 2007), Yago (Suchanek et al., 2007), Wikidata (Vrandečić and Krötzsch, 2014) and Freebase (Bollacker et al., 2008). Their commercial counterparts at Google, Microsoft, Baidu, and others provide back-end support for search engines, online recommendations, and various knowledge-centric services. They cover many millions of entities with billions of triples for thousands of predicates. However, despite this impressive size, no KB is ever complete. In fact, even the largest KBs miss out on many interesting predicates and emerging entities such as brand-new events or unknown people who suddenly become notable. As an example, consider what KBs provide about Brad Pitt: his birthplace, his movies, wives, children, etc. However, they do not point out which children have been adopted, nor that Angelina Jolie recently filed for divorce from him. There is even interesting information about his movies that is absent from all KBs. An example is that he played the mountaineer Heinrich Harrer in Seven Years in Tibet, which would ideally be captured as a quadruple ⟨`Brad_Pitt`, `play_role_in`, `Heinrich_Harrer`, `Seven_Years_in_Tibet`⟩. These gaps cannot be easily filled, as many predicates are completely missing; and even for known predicates, it is hard to keep up with the pace at which new facts appear in the real world. This calls for a more open-ended and dynamic KB construction.

The goal of dynamic and broader construction of KBs has received substantial attention in the database research community recently. The DeepDive project (Niu et al., 2012; Shin et al., 2015; Zhang et al., 2016) has developed a highly versatile tool suite for information extraction (IE) and KB population (KBP), based on Markov Logic (Domingos and Lowd, 2009) and further techniques, including a variety of optimizations. Another ground-breaking project in this space is SystemT (Chiticariu et al., 2010, 2013; Reiss et al., 2008), which uses declarative

rules for IE in a wide range of applications, including enterprise content analytics. However, these prior works still require a specification of which predicates are of interest to the IE/KBP process. Unless predicates like `has_adopted_child`, `filed_divorce_from` or `plays_role_in` are made explicit by the application architect (or "knowledge engineer"), they will not be discovered automatically. This may not be a problem for most use cases in enterprises or data science, but it does limit the ability of these approaches to extract knowledge without any prior setup phase.

State-of-the-Art & Limitations. The field of Open IE (Banko et al., 2007; Mausam et al., 2012) partially addresses the task of on-the-fly KB construction. In Open IE, however, the subject-predicate-object arguments of the extracted triples are usually not canonicalized. For example, triples with subjects "Brad Pitt", "Bradley Pitt", "Oscar winner Pitt", etc. will all be present even if their statements are equivalent. The DEFIE (Bovi et al., 2015) system, for example, thus adds a post-processing stage to disambiguate entity names, but still leaves predicates unresolved. Predicates like `wins_prize` and `receives_award` will co-exist, although they are synonymous. Recently, Galárraga et al. (2014) and Riedel et al. (2013) further canonicalized Open IE output by clustering noun phrases as subjects and objects, while verbal phrases are clustered into relations. However, these approaches have high overhead and are not geared for dynamic knowledge acquisition. Declarative approaches to IE and KBP, like DeepDive (Niu et al., 2012; Shin et al., 2015; Zhang et al., 2016) and SystemT (Chiticariu et al., 2010, 2013; Reiss et al., 2008), require specifications of predicates and rules. Thus, also they cannot be used in a spontaneous "on-the-fly" manner. Query-time IE, as pursued in our work, resembles the notion of query-time inference over probabilistic databases (Dylla et al., 2014; Suciu et al., 2011). These methods operate on uncertain relational data as well as uncertain rules (i.e., views), and support flexible forms of top-k queries (Dylla et al., 2013) and general inference (Gatterbauer and Suciu, 2017; Gribkoff and Suciu, 2016; Li et al., 2017). However, all of these approaches require a relational schema that underlies the KB.

Approach & Contributions. This chapter presents QKBfly (Nguyen et al., 2018), a novel system for constructing query-driven, on-the-fly KBs. Based on our experience with various IE tasks (Hoffart et al., 2012; Nakashole et al., 2012; Nguyen et al., 2014, 2016; Yosef et al., 2011), our focus in this work is to develop an end-to-end system for KB construction, which may be triggered by an ad-hoc user query (e.g., when an analyst or journalist becomes interested in a particular person, organization or event). The system takes as input an entity-centric query or a natural-language question, automatically retrieves relevant source documents (via Wikipedia and news sources), runs a novel form of knowledge extraction on the sources, and builds a high-coverage KB that is focused on the entities of interest. Compared to mainstream KBs, we acquire facts for a much larger set of predicates. Compared to Open IE methods, arguments of facts are canonicalized, thus referring to unique entities with semantically typed predicates derived from

precomputed clusters of phrases. Besides supporting analytical queries, QKBfly thus also facilitates the application of current question-answering (QA) frameworks (Berant et al., 2013; Bast and Haussmann, 2015), which increasingly rely on structured knowledge backends, to currently popular events and queries.

At the heart of QKBfly is a *semantic-graph* representation of sentences that captures per-sentence clauses, noun-phrases, pronouns, as well as their syntactic and semantic dependencies. Based on this graph, we devise an efficient inference technique that performs three key IE tasks, namely *named-entity disambiguation*, *co-reference resolution* and *relation extraction*, in a light-weight and integrated manner. Because of the clause-based representation of sentences, QKBfly is not limited to binary predicates but can also extract ternary (or higher-arity) predicates. To conclude our motivation for this work, we summarize the novel contributions of QKBfly as follows:

- we present an end-to-end system for on-the-fly KB construction that is triggered by an entity-centric user query or a natural-language question;
- QKBfly employs a novel graph-based approach for cleaning, canonicalizing and organizing noisy extractions from Open IE into a crisp KB (including ternary and higher-arity predicates);
- we conduct extensive experiments that demonstrate the viability of our approach under various IE and QA settings.

In our experiments, we evaluate QKBfly's capability of building on-the-fly KBs against the state-of-the-art baselines DEFIE (Bovi et al., 2015) and DeepDive (Shin et al., 2015). As an extrinsic use case, we employ QKBfly for ad-hoc QA on emerging topics derived from Google Trends.

5.2 System Overview

5.2.1 Design Space and Choices

KB construction generally faces an inherent *trade-off* between precision (i.e., fraction of correct tuples among the acquired ones) and recall (i.e., fraction of correct tuples among the ones that could possibly be acquired from the input). In traditional KB construction the priority is usually precision, as large KBs (e.g., commercial knowledge graphs, Yago, Wikidata, etc.) are an infrastructure asset meant to support a wide variety of applications. In contrast, on-the-fly KB building is intended to support analysts in ad-hoc exploration and querying. Therefore, recall is the primary priority, and good precision is a secondary goal within this regime.

This overriding design decision has consequences on the system architecture. While holistic methods with joint inference on all steps and sub-goals (e.g., probabilistic graphical models, constraint-based reasoners, etc.) are often attractive, they are much harder to control in their behavior towards separately tunable recall and precision. Moreover, tools like Alchemy (Domingos and Lowd, 2009), DeepDive (Niu et al., 2012; Shin et al., 2015), Thebeast (Riedel, 2008) or Sofie

(Suchanek et al., 2009) require sophisticated modeling, training and configuration upfront, which is all but straightforward in our open-domain on-the-fly setting. These considerations are the rationale for splitting our approach to on-the-fly KB construction into two phases: a recall-oriented extraction phase followed by a precision-oriented cleaning phase. This separation gives us best control on the trade-off.

Extraction. Since on-the-fly KB construction aims for high recall, we adopt the OpenIE paradigm (Banko et al., 2007) for this phase. To cope with the high diversity of input documents that a query-driven approach comes with, we employ a judiciously designed set of linguistic pre-processsing steps. For these, we use standard tools that are state-of-the-art in NLP. One of these is ClausIE (Del Corro and Gemulla, 2013), which decomposes sentences into a set of clauses. For efficiency, we modified ClausIE, to use the MaltParser (Nivre and Hall, 2005) instead of its original reliance on the Stanford parser (Klein and Manning, 2003). In our experiments, we compare QKBfly against a variety of best-practice OpenIE tools.

Cleaning. The extraction phase produces a large set of – still noisy – candidates for triples and tuples. To remove false positives and reconcile semantic redundancy, we perform two major cleaning steps: one resolving entity mentions and co-references (see Section 5.4), and one for canonicalizing relational predicates (see Section 5.5). Both are potentially expensive tasks. Since the case for on-the-fly KB construction is ad-hoc information needs that should support analysts in a same-day manner, we decided to devise light-weight algorithms for both tasks (see below), based on a graph model, but avoiding the heavy-duty joint inference that probabilistic graphical models (PGMs) usually incur. Our experiments provide some comparison points for precision and run-time of our method against the joint-inference paradigm. As MAP inference (maximum a posteriori) in PGMs is equivalent to solving a weighted MaxSat problem which in turn is a form of constraint programming, we picked the Integer Linear Programming (ILP) solver Gurobi[1] for comparison. This is a mature and highly optimized tool, performing very well on a wide range of constraint-based reasoning tasks. Note that some PGMs actually use ILP for efficient MAP inference (e.g., (Riedel, 2008)).

5.2.2 QKBfly Overview

Figure 5.1 depicts the architecture of QKBfly. Given a set of *input documents* (D), retrieved in response to a *user query* (Q), QKBfly works in three stages. First, it builds a *semantic graph* (G) from clauses and initial co-references extracted from each of the sentences contained in (D). Second, it refines each semantic graph by jointly performing named-entity disambiguation and co-reference resolution via a *graph-densification algorithm* (A). Third, it canonicalizes the *on-the-fly KB* (K) by merging co-reference nodes and by mapping relational paraphrases to a canonical set of entities and relations.

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

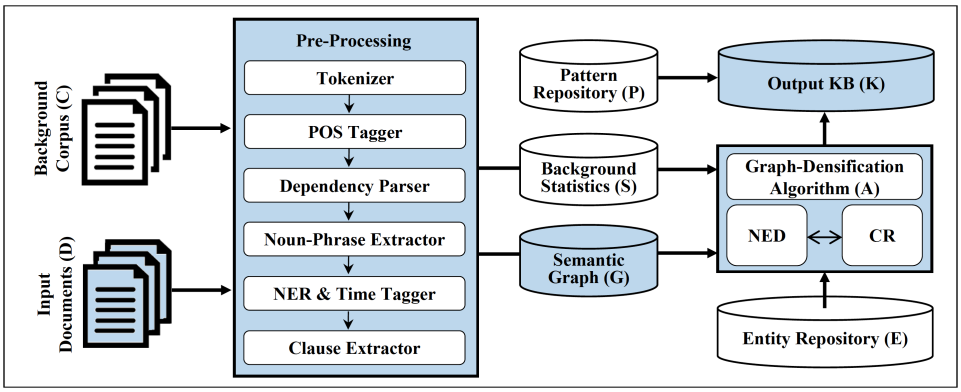

Figure 5.1: System overview. Blue components are processed on-the-fly.

Background Repositories. As static input, QKBfly employs three types of repositories, namely a *background corpus* (C), a *pattern repository* (P) and an *entity repository* (E). From (C), we also extract *statistics* (S) used later by QKBfly. Each of these background repositories is exchangeable. For the experimental setup described in Section 5.7, we fixed them as follows: we employ a Wikipedia[2] full-text dump from Sept. 1, 2015 as (C), PATTY[3] (Nakashole et al., 2012) consisting of 127,811 relational paraphrases as (P), and Yago[4] (Suchanek et al., 2007) with 3,420,126 entities as (E). As for Yago, we merely harness its knowledge about alias names of entities together with their gender attributes (for better pronoun resolution), while none of the actual KB facts are used in QKBfly. In particular, we do not require all entities we recognize during KB construction to be present in the given entity repository.

Statistics. Both the background corpus (C) and the input documents (D) are pre-processed by pipeline of linguistic tools, consisting of tokenization, part-of-speech (POS) tagging, noun-phrase chunking and named-entity recognition (NER), all of which are performed by the Stanford CoreNLP toolkit (Manning et al., 2014). In addition, we employ time tagging (Chang and Manning, 2012) and the Open IE tool ClausIE (Del Corro and Gemulla, 2013) to extract clause structures in which all arguments are annotated either as names or time expressions. As for the Wikipedia-based background corpus (C), we also map clause components to Wikipedia entities by their *href* links if the NER type of the clause component matches the type of Wikipedia entity. Based on the resulting clause structures, we compute (co-)occurrence statistics for clause-argument-types and the relationships among them. These serve as input to the feature functions described in Section 5.4.

Stage 1: Semantic Graph. From the pre-processed input documents (D), we build a semantic graph for each sentence in (D) based on the clause structure

[2] https://dumps.wikimedia.org/enwiki/
[3] https://d5gate.ag5.mpi-sb.mpg.de/pattyweb/
[4] https://www.yago-knowledge.org/

detected by ClausIE. A leaf node in this graph represents an occurrence of an entity, while an edge among two leaf nodes represents a relation pattern in our on-the-fly KB (K). The per-sentence graphs are linked via an initial set of possible co-reference edges (based on the technique of Bamman et al. (2014)), thus connecting pairs of leaf nodes that potentially refer to the same entity (see Section 5.3).

Stage 2: Graph Algorithm. Next, the graph-densification algorithm refines each of the connected semantic graphs. It thereby jointly performs entity disambiguation and co-reference resolution based on an efficient algorithm for densifying the semantic graphs. The method is inspired by and generalizes the dense-subgraph algorithm introduced in Hoffart et al. (2011b), which we judiciously chose as a basis due to its good runtime performance and accuracy. The algorithm employs a greedy heuristic for pruning edges to obtain a dense subgraph under a set of given constraints. The remaining edges in the densified subgraph link mentions to unique entities in the resulting on-the-fly KB (see Section 5.4).

Stage 3: On-the-fly KB Canonicalization. In the final stage, QKBfly constructs the on-the-fly KB (K) by combining and canonicalizing the remaining nodes and edges in the semantic graph. Entities are either linked to the entity repository (E) or are identified as a cluster of new names (for emerging entities) which are connected by co-reference edges. At this stage, QKBfly uses the pattern repository (P) to map relational paraphrases to a canonicalized set of relations. Similarly to the entity repository, new relational paraphrases not contained in (P) are considered as new relations. Moreover, by considering the per-sentence clause structure, QKBfly is able to acquire triples as well as higher-arity facts (see Section 5.5).

5.3 Semantic Graph

When given a set of natural-language sentences (i.e., in a web page, a Wikipedia article, etc.) as input, QKBfly first builds one semantic graph for each input sentence. It then connects the per-sentence graphs by co-reference links among nodes that potentially refer to the same entity. QKBfly primarily employs ClausIE to construct these per-sentence graphs. To improve the efficiency of the extraction process, we use the Malt parser (Nivre and Hall, 2005) in our implementation instead of the Stanford parser (Klein and Manning, 2003) used in the original version of ClausIE. Besides dependency parsing, ClausIE exploits further linguistic features such as POS tagging and noun-phrase chunking to detect so-called *clauses*. Following Quirk et al. (1985), a clause is a coherent piece of information within a sentence that consists of one *subject* (S), one *verb* (V), an optional (either direct or indirect) *object* (O), an optional *complement* (C), and a variable amount of *adverbials* (A). A main observation of Quirk et al. (1985) is that only seven combinations of the above constituents, namely SV, SVA, SVC, SVO, SVOO, SVOA and SVOC, actually occur in the English language, which is a key for relation extraction in ClausIE. That is, one clause confirms to exactly one n-ary fact

with these constituents as arguments. In addition, we use the Stanford NER tagger (Finkel et al., 2005) and SUTime (Chang and Manning, 2012) to detect named entities within the clauses.

Nodes. A node in our semantic graph is a container for *clauses*, *nouns*, *pronouns* and *entities* occurring in an input sentence and the entity repository, respectively. Specifically, we distinguish the following four types of nodes:

- A `clause` node is generated for each clause detected by ClausIE. A clause may be connected to multiple dependent clauses in the same sentence. Their dependency structure is also detected by ClausIE.
- A `noun-phrase` node is generated for each noun phrase detected by the noun-phrase chunker and for each named entity detected by the NER tagger (both using Manning et al. (2014)). Additional time expressions are detected by the time tagger (using Chang and Manning (2012)).
- A `pronoun` node is generated for a pronoun (such as "*he*", "*she*", "*they*", etc.). Noun-phrases and pronouns together form the leaves of the subtree built for each sentence.
- An `entity` node is generated for each candidate (e.g., `Brad_Pitt`) of a `noun-phrase` node that matches a known alias name in the entity repository.

Edges. Edges represent the syntactic and semantic *dependencies* among nodes in the semantic graph. Here, we distinguish the following four types of edges:

- A `depends` edge links two dependent clauses. It additionally links a `clause` node with the `noun-phrase` and `pronoun` nodes it contains.
- A `relation` edge represents a relation pattern (i.e., the lemmatized verb (V) constituent of the clause with an optional preposition such as "*to*", "*in*", etc.) that connects two `noun-phrase` or `pronoun` nodes in a clause.
- A `sameAs` edge links two `noun-phrase` or `pronoun` nodes which likely refer to the same entity (by following Bamman et al. (2014) for co-reference and pronoun resolution).
- A `means` edge links a `noun-phrase` or `pronoun` node with an `entity` node based on matching alias names in the entity repository.

We follow the method of Bamman et al. (2014) to initialize the `sameAs` edges between `noun-phrase` nodes and `pronoun` nodes, respectively. The `sameAs` edges among two `noun-phrase` nodes with the same NER label (e.g., `PERSON`) are determined by string matching (e.g., between "*Brad Pitt*" and "*Pitt*"). Additionally, `sameAs` edges are created between `pronoun` nodes and all `noun-phrase` nodes that precede the pronoun by at most five backward sentences. Our graph algorithm (see Section 5.4) will later remove all but the most likely `sameAs` edge between a `pronoun` node and its linked `noun-phrase` nodes. In addition to the verb (V) constituents detected by ClausIE, we apply one more heuristic to label `relation` edges. That is, for text patterns of the form "*'s* ⟨*noun*⟩" (e.g., "*Pitt's ex-wife Angelina Jolie*"), we consider the middle noun (i.e., "*ex-wife*") as the relation candidate between the two `noun-phrase` nodes.

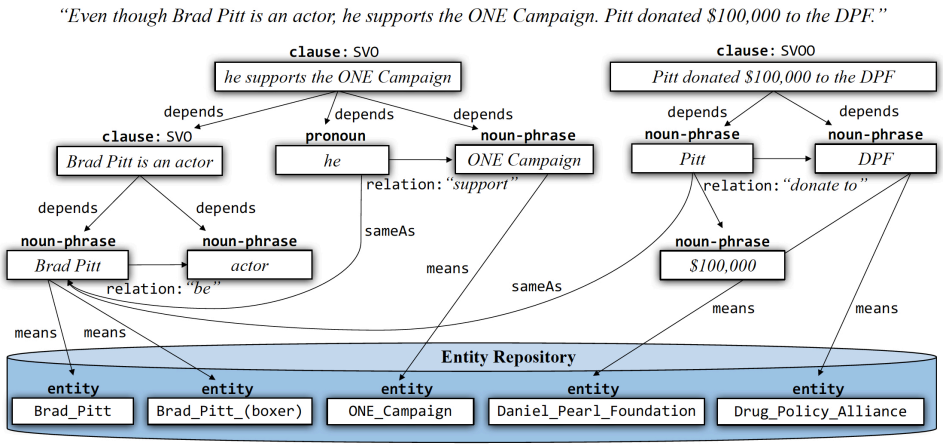

Figure 5.2: Semantic graph example.

Figure 5.2 depicts a semantic graph built from the two input sentences shown on top of the figure. The first sentence contains a SVC clause and a SVO clause, namely *"Brad Pitt is an actor"* and *"he supports the ONE Campaign"*, which results in two triples ⟨*"Brad Pitt"*, *"be"*, *"actor"*⟩ and ⟨*"he"*, *"support"*, *"ONE Campaign"*⟩ whose arguments are not yet canonicalized. Similarly, the quadruple ⟨*"Pitt"*, *"donate to"*, *"$100,000"*, *"Daniel Pearl Foundation"*⟩ is extracted from the SVOO clause of the second sentence. Notice that the noun phrases *"actor"* and *"$100,000"* could not be linked to any entity in the entity repository. These will remain string literals in the respective arguments of the former two facts.

5.4 Graph Algorithm

We henceforth refer to the semantic graph built according to the previous section as $\mathcal{G} = (\mathcal{N}, \mathcal{R})$, where $\mathcal{N}$ denotes the set of nodes, and $\mathcal{R}$ denotes the set of edges (i.e., "relationships") among nodes in $\mathcal{G}$. The goal of our graph-densification algorithm then is to remove false-positive `means` and `sameAs` edges from $\mathcal{R}$ by solving a *constraint-based optimization* problem. In doing so, we perform a form of *joint inference* for the two key IE tasks of *named-entity disambiguation* and *co-reference resolution*.

Edge Weights. For a subgraph $\mathcal{S} = \langle \mathcal{N}' \subseteq \mathcal{N}, \mathcal{R}' \subseteq \mathcal{R} \rangle$ of $\mathcal{G}$, we first distinguish the following dependencies among nodes.

- For each `noun-phrase` node n_i, let $ent(n_i, \mathcal{S})$ be the set of all `entity` nodes associated with n_i by `means` edges in $\mathcal{R}'$.
- For each `pronoun` node p_i, let $np(p_i, \mathcal{S})$ be the set of all `noun- phrase` nodes associated with p_i by `sameAs` edges in $\mathcal{R}'$.
- Further, let $ent(p_i, \mathcal{S})$ denote the union of all $ent(n_t, \mathcal{S})$ sets, where $n_t \in np(p_i, \mathcal{S})$.

Next, we define the edge weights to establish our densest-subgraph objective as follows.

(1) The weight of a `means` edge between a `noun-phrase` node n_i and an `entity` node $e_{i_j} \in ent(n_i, \mathcal{S})$ for a subgraph $\mathcal{S}$ is computed as

$$\begin{aligned} w(n_i, e_{i_j}) = \; & \alpha_1 \cdot f_1(n_i, e_{i_j}) + \\ & \alpha_2 \cdot f_3(n_i, e_{i_j}) \end{aligned} \tag{5.1}$$

where α_1 and α_2 are hyper-parameters and:

- $f_1(n_i, e_{i_j})$ captures the *mention-entity prior* between n_i and e_{i_j}.
- $f_3(n_i, e_{i_j})$ captures the *mention-entity token context similarity* between node n_i and candidate e_{i_j}.

(2) The weight of a `relation` edge between two `noun-phrase` or `pronoun` nodes n_i, n_t for a subgraph $\mathcal{S}$ is computed as

$$\begin{aligned} w(n_i, n_t, \mathcal{S}) = \; & \alpha_3 \cdot \sum_{\substack{e_{i_j} \in ent(n_i, \mathcal{S}) \\ e_{t_k} \in ent(n_t, \mathcal{S})}} f_4(e_{i_j}, e_{t_k}) + \\ & \alpha_4 \cdot \sum_{\substack{e_{i_j} \in ent(n_i, \mathcal{S}) \\ e_{t_k} \in ent(n_t, \mathcal{S})}} ts(e_{i_j}, e_{t_k}, r_{i,t}) \end{aligned} \tag{5.2}$$

where α_3 and α_4 are hyper-parameters and:

- $f_4(e_{i_j}, e_{t_k})$ captures the *entity-entity token context similarity* between two entity candidates e_{i_j} and e_{t_k}.
- $ts(e_{i_j}, e_{t_k}, r_{i,t})$ is the relative frequency under which the semantic types of e_{i_j}, e_{t_k} occur under the relation pattern $r_{i,t}$ in the clauses detected by ClausIE. Since an entity can have several types (e.g., `ACTOR` and `PERSON`) in our type system (described in Chapter 4, Section 4.5), we take the sum over all type combinations for the given entity pair.

We remark that in order to speed up the system, we only consider four key features including f_1, f_3, f_4, and f_7 in the list of seven features described in Chapter 2, Section 2.1. Particularly, f_7 is embedded in the type signature feature. Additionally, we do not compute weights for `sameAs` and `depends` edges. While `sameAs` edges are used as constraints in the optimization model, `depends` edges only contribute to the final KB construction, particularly for determining the fact boundaries as described in Section 5.5.

Optimization Objective. After assigning the edge weights, we next aim to find the densest subgraph $\mathcal{S}^* = \langle \mathcal{N}^* \subseteq \mathcal{N}, \mathcal{R}^* \subseteq \mathcal{R} \rangle$ that maximizes the following objective function.

- The sum of all edge weights in $\mathcal{S}^*$, denoted as $W(\mathcal{S}^*)$, is maximized,

which is subject to the following constraints:

1. each `noun-phrase` node is connected to at most one `entity` node by a `means` edge in $\mathcal{R}^*$;
2. each `pronoun` node is connected to at most one `noun-phrase` node by a `sameAs` edge in $\mathcal{R}^*$;

3. all `noun-phrase` or `pronoun` nodes that are mutually linked by `sameAs` edges, are connected to the same entity;
4. each `pronoun` node connected to a `noun-phrase` node that is connected to an `entity` node of type `PERSON` for which the background KB provides gender information, must match that gender.

Data: Semantic graph $\mathcal{G} = \langle \mathcal{N}, \mathcal{R} \rangle$ from the input text.
Result: The densest subgraph $\mathcal{S}^* = \langle \mathcal{N}^* \subseteq \mathcal{N}, \mathcal{R}^* \subseteq \mathcal{R} \rangle$ which satisfies the four constraints (1), (2), (3) and (4) (Chapter 5, Section 5.4).

for *subgraph* $\mathcal{S} = \langle \mathcal{N}' \subseteq \mathcal{N}, \mathcal{R}' \subseteq \mathcal{R} \rangle$ **do**
  for *node* $x \in \mathcal{N}'$ **do**
    $np(x) \leftarrow$ all `noun-phrase` nodes linked to x by `sameAs` edges in $\mathcal{R}'$;
  for `noun-phrase` *node* $n_i \in \mathcal{N}'$ **do**
    $ent(n_i) \leftarrow$ all `entity` nodes linked to n_i by `means` edges in $\mathcal{R}'$;
    if $ent(n_i) = \emptyset$ **then**
      report a *new* entity;
  for `noun-phrase` *node* $n_i \in \mathcal{N}'$ **do**
    for `noun-phrase` *node* $n_t \in np(n_i)$ **do**
      $ent(n_i) \leftarrow ent(n_i) \cap ent(n_t)$;
  for `pronoun` *node* $p_i \in \mathcal{N}'$ **do**
    $ent(p_i) \leftarrow \cup_{n_t \in np(p_i)} ent(n_t)$;
    for *entity* $e \in ent(p_i)$ **do**
      if *doesn't satisfy gender constraint (4)* **then**
        remove e from $ent(p_i)$;
  $W(\mathcal{S}) \leftarrow$ sum over all edge weights in $\mathcal{S}$;
$\mathcal{S} \leftarrow \mathcal{G}$;
while *all constraints are satisfied* **do**
  for `means` *or* `sameAs` *edge of two node* $(x, y) \in \mathcal{R}'$ **do**
    $\mathcal{S}' \leftarrow$ subgraph of $\mathcal{S}$ by removing (x, y);
    $c(x, y, \mathcal{S}) = W(\mathcal{S}) - W(\mathcal{S}')$;
  remove the `means` or `sameAs` edge between two node (x, y) with the smallest contribution $c(x, y, \mathcal{S})$ and no constraint will be violated;
  update $\mathcal{S}$;
  if *no edge is removed* **then**
    return $\mathcal{S}$;

Algorithm 1: Densest-Subgraph Algorithm.

Approximation Algorithm. Sozio and Gionis (2010) shows that the above formulation of a densest-subgraph problem with constraints is NP-hard. We thus resort to approximating our optimization objective by the following greedy al-

gorithm. We first restrict the entity candidates for each `noun-phrase` node to the ones contained in the dictionary of the background KB, but allow unlinked `noun-phrase` nodes in the final subgraph for *out-of-KB* entities. For all `noun-phrase` nodes that are mutually connected via `sameAs` edges, the entity candidate sets are intersected. The approximation algorithm then greedily iterates on the graph's edges as long as no constraint is violated. In each round, the algorithm removes the `means` or `sameAs` edge between two nodes (x, y) with the smallest contribution to the objective function of the current subgraph $\mathcal{S}$. This contribution to the objective function is defined as

$$c(x, y, \mathcal{S}) = W(\mathcal{S}) - W(\mathcal{S}') \tag{5.3}$$

where $\mathcal{S}'$ is the subgraph of $\mathcal{S}$ we obtain by removing edge (x, y) from $\mathcal{S}$. If removing an edge leaves an entity candidate isolated, then that entity node is removed as well. After each edge removal, the weights of all remaining edges need to be recomputed, as the cutting of `sameAs` edges modifies the influence of `relation` edges attached to `noun-phrase` or `pronoun` nodes. Our implementation performs this recomputation step in a selective and incremental way. Algorithm 1 shows pseudocode for our algorithm.

Confidence Scores. As an additional filtering step, we assign a normalized confidence score to each `noun-phrase` or `pronoun` node n_i that is disambiguated to an entity e_{i_j} as

$$score(n_i, e_{i_j}, \mathcal{S}^*) = \frac{c(n_i, e_{i_j}, \mathcal{S}^*)}{\sum_{e_{i_t} \in ent(n_i, \mathcal{G})} c(n_i, e_{i_t}, \mathcal{S}_t)} \tag{5.4}$$

where the denominator sums up over all subgraphs $\mathcal{S}_t$ constructed from $\mathcal{S}^*$ by replacing e_{i_j} (and its associated `means` edge) with one of the original candidates e_{i_t} (and its `means` edge). For the confidence of a triple (or higher-arity fact), we choose the minimum of the confidence scores of all disambiguated entities that occur as the arguments of the fact. In our experiments, we use a score threshold $\tau = 0.5$ to distill high-quality facts.

Hyper-Parameter Tuning. We manually annotated 162 sentences from 5 Wikipedia articles, about prominent person entities, including Andrew Ng, Angela Merkel, David Beckham, Larry Page and Paris Hilton (thus covering scientists, politicians, sports stars, business people and models). These annotations comprise 203 facts, each consisting of a pair of Yago entities and a relation pattern (e.g., ⟨`Larry_Page`, "*born in*", `Michigan`⟩).

By independently constructing a graph $\mathcal{G}$ with two `noun-phrase` nodes n_i and n_t for each triple fact, we define the probability of choosing `entity` node $e_{i_j} \in ent(n_i, \mathcal{G})$ and `entity` node $e_{t_k} \in ent(n_t, \mathcal{G})$ as

$$prob(n_i, e_{i_j}, n_t, e_{t_k}, \mathcal{G}) = \frac{W(\mathcal{S})}{W(\mathcal{G})} \tag{5.5}$$

where the subgraph $\mathcal{S}$ is constructed from $\mathcal{G}$ by removing all `entity` nodes except e_{i_j} and e_{t_k}. The parameters $\alpha_{1..4}$ are learned by maximizing the probability of the ground-truth annotations using L-BFGS optimization (Liu and Nocedal, 1989), which implements a memory-efficient variant of stochastic gradient descent.

5.5 On-the-fly Knowledge Base Construction

In the final stage, QKBfly processes the output of the graph densification to perform our final IE task, *relation extraction*, to populate our on-the-fly KB with facts. Noun-phrases and pronouns at this time are either linked to unique entities in the entity repository or are identified by a group of `noun-phrase` nodes connected via `sameAs` edges. Specifically, a new entity is introduced for each group of `sameAs` nodes that consists only of out-of-repository names. We additionally consider all groups of `noun-phrase` nodes that link to entities in the background repository with very low confidence scores as new entities. These are added to the on-the-fly KB as emerging entities—an important asset for up-to-date knowledge.

Relation edges carry surface-form labels: patterns that denote predicates. To canonicalize also these patterns, we harness the pattern repository, specifically the PATTY dictionary of relational paraphrases (Nakashole et al., 2012). All node-edge-node triples that have the same node labels and have edge labels that belong to the same synset in PATTY are combined into a single triple, thereby clustering the relation patterns together. For example, relation edges with labels "*play in*", "*act in*" and "*star in*" that connect the same actor-movie pair are combined. This way, we are not limited to the relations that are registered in an existing KB (such as the ca. 100 predicates in Yago or the ca. 6,000 property labels in DBpedia), but can also capture many interesting relations on-the-fly. Unlike all of the major KBs, we also construct facts for ternary or higher-arity relations. This is where the extraction of clauses pays off. Whenever `noun-phrase` or `pronoun` nodes are linked to the same `clause` node via `depends` edges, we merge those nodes into a single fact. For example, QKBfly can construct ternary facts from SVOO or SVOA clauses such as ⟨`Brad_Pitt`, `play_in`, `Heinrich_Harrer`, `Seven_Years_In_Tibet`⟩, ⟨`Brad_Pitt`, `adopt_in`, `Pax_Thien_ Jolie-Pitt`, *"2008"*⟩, etc. Those higher-arity facts provide more complete information than triple facts, which is useful for many extrinsic use cases such as QA on complex questions (e.g., "*Who plays Achilles in Troy?*").

Table 5.1 shows sample results (entities and their mentions, relational patterns and facts) of the on-the-fly KB constructed by QKBfly from the `Brad_Pitt` Wikipedia page. QKBfly captures long-tail entities, such as Brad Pitt's father `William_Alvin_Pitt` who is missing in most KBs. QKBfly captures binary facts (triples) such as ⟨`Brad_Pitt`, `born_to`, `William_Alvin_Pitt`⟩, as well as ternary facts such as ⟨`Brad_Pitt`, `play_in`, `Achilles`, `Troy_(film)`⟩.

As a demonstration of QKBfly's ability to compile KBs in a query-driven manner on-the-fly, we ran it for a set of different queries with news articles returned

by the Google search engine. Table 5.2 shows sample results extracted from top-ranked news about some celebrities. QKBfly compiles *up-to-date* knowledge like the Pitt and Jolie divorce, the Nobel prize for Bob Dylan, and the death of Thailand's king. Also, QKBfly captures emerging (out-of-traditional-KB) entities such as two women who accuse `Donald_Trump` of sexual abuse: `Jessica_Leeds` and `Natasha_Stoynoff`.

Table 5.1: Excerpt of QKBfly output from Google news articles. Out-of-Yago entities have an asterisk; relations are in italics.

Entities & Mentions
`Brad_Pitt` → *"William Bradley Pitt"*, *"Brad Pitt"*, *"Pitt"*, etc.
`William_Alvin_Pitt`* → *"William Alvin Pitt"*
`Achilles` → *"Achilles"*, *"warrior Achilles"*, etc.
`Troy_(film)` → *"Troy"*
Relations & Patterns
`born_to` → *"born to"*, *"father"*, etc.
`play` → *"play"*, *"act"*, etc.
`play_in` → *"act in"*, *"star in"*, *"play in"*, *"have role in"*, etc.
Facts
⟨`Brad_Pitt`, `born_to`, `William_Alvin_Pitt`*⟩
⟨`Brad_Pitt`, `play_in`, `Achilles`, `Troy_(film)`⟩

Table 5.2: Excerpt of QKBfly output from Google news articles. Out-of-Yago entities have an asterisk; relations are in italics.

Query	**Fact**
Brad Pitt	⟨`Brad_Pitt`, `divorce_from`, `Angelina_Jolie`⟩
	⟨`Angelina_Jolie`, `file_for_on`, *"divorce"*, *"Sep. 19"*⟩
	⟨`Brad_Pitt`, `reunite_with_on`, *"his kids"*, *"Oct. 8"*⟩
Bob Dylan	⟨`Bob_Dylan`, `win_for`, `Nobel_Prize_in_Literature`, *"having created new poetic expressions within the American song tradition"*⟩
Bhumibol Adulyadej	⟨`Bhumibol_Adulyadej`, `die_on`, *"13 Oct. 2016"*⟩
	⟨`Vajiralongkorn`, `become`, *"new king"*⟩
Donald Trump	⟨`Jessica_Leeds`*, `accuse_of`, `Donald_Trump`, *"groping her on an airplane in the 1980s"*⟩
	⟨`Natasha_Stoynoff`*, `accuse_of`, `Donald_Trump`, *"making sexual advances during an interview"*⟩

5.6 QKBfly at Work

We developed a user interface to demonstrate the ability of QKBfly to construct KBs on-the-fly. Given a set of input documents, such as a Wikipedia page or a collection of news articles, QKBfly can build a KB with hundreds (or thousands) of facts within a minute.

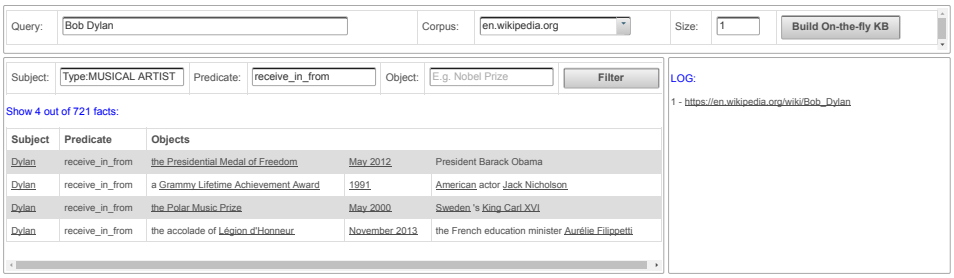

Figure 5.3: Sample of higher-arity facts by QKBfly.

Figure 5.4: Sample of up-to-date facts from news by QKBfly.

System Implementation. Figure 5.3 and Figure 5.4 show two screenshots of QKBfly in a browser.

- For the input, we let the user choose a query (e.g., "*Bob Dylan*"), the input source (Wikipedia or news articles) and the desired number of input documents. QKBfly then processes relevant documents by restricting the search to `en.wikipedia.org` for Wikipedia, and to `bbc.com` for news.
- For the output, we show all facts from the on-the-fly KB. Prominent entities may be linked to entities in the entity repository. As the number of facts can be huge, we offer a string search on subjects, predicates or objects. QKBfly also supports *type* search if the user specifies the prefix `Type:` to the queried category. Figure 5.3 shows the result of 4 out of 721 facts in total by searching for `Type:MUSICAL_ARTIST` as the subject and `receive_in_from` as the predicate in the on-the-fly KB constructed from the Wikipedia page of Bob Dylan.

Figure 5.4 also illustrates the ability of QKBfly to capture up-to-date facts from news. For example, there is the fact that Patti Smith forgot the lyrics when performing Bob_Dylan's song at the Nobel Prize ceremony, which is extracted from 10 news articles. The predicate `forget` would not be covered by many of the existing KBs. Also, obtaining this kind of knowledge is not possible with state-of-the-art tools for IE and knowledge base population. Methods, like DeepDive or SystemT, would require substantial setup work by a knowledge engineer to obtain these results.

5.7 Experiments

In our experiments, we first compare QKBfly's capability of building on-the-fly KBs against the state-of-the-art Open IE baseline DEFIE (Bovi et al., 2015). Second, we compare our greedy approximation algorithm for the joint inference of named entity disambiguation (NED) and co-reference resolution (CR) against an integer linear programming (ILP) algorithm. Third, we compare QKBfly's capability of performing mentions of spouses extraction (i.e., relation "`married_to`") against DeepDive (Zhang et al., 2016). Finally, as an extrinsic use case, we dynamically construct ad-hoc KBs for question answering.

5.7.1 Experiments on KB Construction

Benchmarks. We use two datasets:

- DEFIE-Wikipedia dataset[5] (Bovi et al., 2015), consisting of 14,072 randomly chosen Wikipedia pages with 225,867 sentences. We use this dataset to run experiments on end-to-end KB construction.
- Reverb dataset[6] (Fader et al., 2011), consisting of 500 sentences which have been obtained by the random-link service of Yahoo. This is used to run experiments on Open IE components.

Methods under Comparison. We compare several configurations of QKBfly against DEFIE:

- QKBfly. This jointly performs fact extraction, NED and CR.
- QKBfly-pipeline. This is a pipeline architecture with three separate stages for fact extraction, NED and CR. The type signature feature (for fact extraction and NED) is omitted.
- QKBfly-noun. This performs fact extraction and NED. CR is omitted.
- DEFIE. This is a pipeline architecture with two stages for Open IE and NED, using Babelfy (Moro et al., 2014) for NED.

Environment. To have a fair comparison about runtime among systems, all experiments are run single-threaded on an Intel Xeon X5650 server with 64GB RAM.

[5] provided by the authors
[6] http://reverb.cs.washington.edu/

Assessment. We asked 2 human assessors to evaluate the correctness of 200 randomly sampled extractions. Inter-assessor agreement was high, with Cohen's kappa being $\kappa = 0.77$. Precision values are reported with Wald confidence intervals at 95%. Table 5.3 depicts a number of anecdotal examples.

Table 5.3: Sample extractions from the sentence "*His form attracted Bob Paisley, and McGarvey signed for Liverpool in May 1979.*". Relations are in italic style.

Method	Fact	Assessment
QKBfly & QKBfly-noun	⟨"*His form*", `attract`, `Bob_Paisley`⟩	true
	⟨`Frank_McGarvey`, `sign_for_in`, `Liverpool_F.C.`, "*May 1979*"⟩	true
QKBfly-pipeline	⟨"*His form*", `attract`, `Bob_Paisley`⟩	true
	⟨`Frank_McGarvey`, `sign_for_in`, `Liverpool`, "*May 1979*"⟩	false

5.7.1.1 Results on Fact Extraction

Table 5.10 shows experimental results for fact extraction on the DEFIE-Wikipedia dataset. QKBfly-noun achieves the highest precision: 73% for triple facts and 68% for quadruple facts. QKBfly significantly increases the number of extractions, with a relatively small loss in precision. This suggests that our method works fairly well for co-references. Compared to the pipeline architecture, the joint model of QKBfly increases precision by 5%. All QKBfly variants significantly outperform DEFIE in both precision and coverage. DEFIE has been optimized for short sentences (i.e., definitions) and loses effectiveness when processing complex texts with subordinate clauses and co-references. Also, DEFIE only yields triples, whereas QKBfly returns a large number of higher-arity facts with good precision.

In terms of run-time efficiency, all QKBfly variants perform similarly, less than a second for processing one document. Almost half of the run-time is for pre-processing via the Stanford CoreNLP pipeline and the MaltParser. Thus, all approaches, including the joint models, are efficient and scale to processing large input corpora on the fly. Note that the runtimes for DEFIE are not known; all DEFIE numbers in Table 5.10 are from Bovi et al. (2015).

5.7.1.2 Results on Entity Disambiguation

We compare QKBfly variants to DEFIE/Babelfy on the NED sub-task: linking entities to the KB (Yago and BabelNet, cross-linked via Wikipedia). We remark that Babelfy (Moro et al., 2014) also is a graph-based approach to NED. It performs word sense disambiguation based on a loose identification of candidate meanings.

This is coupled with a densest subgraph heuristic which selects high-coherence semantic interpretations. Since Babelfy does not consider pronouns, we omit the pronoun resolution.

Table 5.4: Experimental results on linking entities to Yago (95% confidence intervals).

Method	Precision	#Extractions
$\text{DEFIE}_{\text{Babelfy}}$	0.82 ± 0.05	39,684
QKBfly	0.86 ± 0.04	50,026
QKBfly-pipeline	0.80 ± 0.05	50,026

As shown in Table 5.4, QKBfly gains 4% while QKBfly-pipeline loses 2% against Babelfy. We observe subtle errors of QKBfly-pipeline and Babelfy coming from the missing type signature feature (e.g., for `Liverpool` the city versus `Liverpool_F.C.` the soccer club, as shown in Table 5.3).

5.7.1.3 Results on Initial Extraction

We compare the Open IE component of QKBfly against state-of-the-art methods including the original work of ClausIE, Reverb (Fader et al., 2011), Ollie (Mausam et al., 2012) and Open IE 4.2[7].

Table 5.5: Experiments on Open IE component. Average runtime (ms/sentence) at confidence of 95%.

Method	Precision	#Extractions	Avg. Run-time (ms)
ClausIE	0.62	1,707	374 ± 127
QKBfly	0.57	1,308	36 ± 11
Reverb	0.53	727	8 ± 2
Ollie	0.44	1,242	24 ± 9
Open IE 4.2	0.56	1,153	59 ± 14

Table 5.5 shows experimental results on the Reverb dataset. ClausIE performs best in terms of precision and the number of extractions. However, it does not provide any canoncalized output and is much slower than the other methods including QKBfly, Ollie, and Open IE 4.2 that benefit from using the MaltParser instead of the Stanford Parser. The purely pattern-based Reverb, which does not use any dependency parsing, is the fastest one. QKBfly shows decent performance in all regards.

[7] https://github.com/knowitall/openie

5.7.2 Experiment on Joint NED and CR

Benchmark. In addition to DEFIE-Wikipedia dataset, we run experiments on two new benchmarks:

- News dataset, consisting of 100 sport news articles with 3,751 sentences extracted from more than 20 news websites such as bbc.com, nytimes.com, telegraph.co.uk, and more on 3rd June 2017.
- Wikia dataset, consisting of 10 Wikia pages with 880 sentences about Game of Thrones, Season 1[8]. Each page consists of narrative text describing an episode of the series.

Methods under Comparison. We compare two configurations of QKBfly with different graph algorithms.

- QKBfly, which performs NED and CR by the greedy approximation algorithm (described in Section 5.4).
- QKBfly-ilp, which performs NED and CR by an Integer Linear Programming (ILP) approach (described in Appendix A.1).

Table 5.6: Experiments on graph algorithms (at 95% confidence intervals).

DEFIE-Wikipedia dataset			
Method	**Precision**	**#Extractions**	**Avg. Run-time (s)**
QKBfly	0.65 ± 0.06	69,630	0.88 ± 0.03
QKBfly-ilp	0.66 ± 0.06	69,630	46.59 ± 16.41
News dataset			
Method	**Precision**	**#Extractions**	**Avg. Run-time (s)**
QKBfly	0.65 ± 0.06	2,096	1.43 ± 0.07
QKBfly-ilp	0.67 ± 0.06	2,096	71.18 ± 25.76
Wikia dataset			
Method	**Precision**	**#Extractions**	**Avg. Run-time (s)**
QKBfly	0.54 ± 0.06	917	4.29 ± 0.11
QKBfly-ilp	0.55 ± 0.06	917	542.36 ± 61.72

Results. As shown in Table 5.6, even though QKBfly-ilp gains 1%-2% in precision, which we also found to be significant under a t-test with a p-value of 0.01 on the DEFIE-Wikipedia dataset, it is much slower than QKBfly, especially when processing long documents in the Wikia dataset. This is because QKBfly-ilp has to handle a very large number of variables in the ILP translation of the graph problem, which makes it less suitable for on-the-fly KB construction. The QKBfly vari-

[8] http://gameofthrones.wikia.com/wiki/Season_1

ants generally lose around 10% in precision when working on the Wikia dataset in comparison to the News and DEFIE-Wikipedia datasets, since the former contains many emerging ("out-of-Yago") entities such as movie characters. We observe that 71% the of entities extracted from the Wikia dataset are out-of-Yago, while only 24% of entities extracted from News dataset and 13% of entities extracted from DEFIE-Wikipedia dataset are new.

5.7.3 Experiments on Information Extraction

To understand how well our method works on a more traditional IE task, we compare QKBfly also against DeepDive (Zhang et al., 2016) by extracting instances of the `spouse` relation from the entire DEFIE-Wikipedia dataset. The DeepDive tutorials[9] specifically provide a pre-configured extraction model for this particular relation, which we additionally retrained by feeding all instances of married couples in DBpedia as positive examples into the DeepDive learner. As in traditional IE, the priority is precision, we use a high confidence threshold $\tau = 0.9$ for both systems.

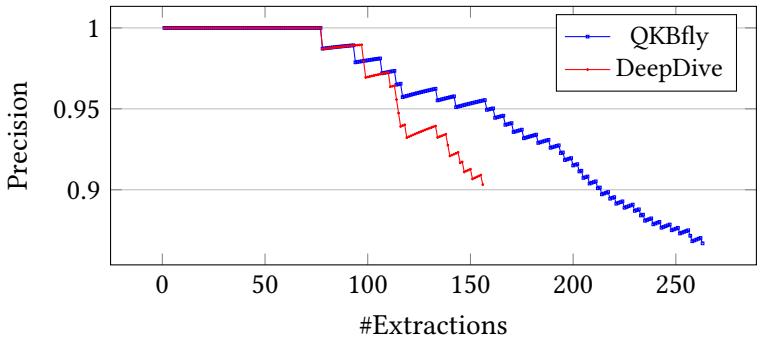

Figure 5.5: Precision-recall curves on the DEFIE-Wikipedia.

Results. Figure 5.5 shows the precision-recall curves of QKBfly and DeepDive over the results, which we ranked by the confidence scores that each of the two systems assigns to its extracted facts. Table 5.7 also depicts the precision values at various recall levels (measured in terms of the number of extractions). Both systems perform very well in terms of precision at the lower recall levels, while QKBfly tends to outperform DeepDive at the higher recall levels. We observe that many extractions of QKBfly come from co-reference resolution for pronouns, which is not part of the extraction model of DeepDive. On the downside, QKBfly is substantially slower than DeepDive in this setting. However, one should notice that QKBfly always performs extractions for all relations – not just for the spouse relation. Considering that DeepDive needs a separate (manually crafted) extraction model for each individual target relation, we believe that this is an excellent result for QKBfly.

[9] http://deepdive.stanford.edu/example-spouse

Table 5.7: Experiments on extracting instances of spouses.

Method	Precision	#Extractions	Run-time (minutes)
QKBfly	1.0	50	206
	0.95	150	
	0.87	250	
DeepDive	1.0	50	117
	0.91	150	
	—	250	

5.7.4 Use Case: Question Answering

As a final use-case, we run experiments on a newly designed QA benchmark, which we coin "GoogleTrendQuestions". First, we used the popular Google Trend service to identify 50 recent events of wider interest between January 2015 and October 2016. Second, we asked students to formulate meaningful questions about these events, and also provide the gold-standard answers. This resulted in 100 questions in total. An example question is: "*Which band was playing during the Paris attacks?*"

Evaluation Metric. We report the macro-averaged precision, recall and F1 score across all test questions. That is, given a set of questions $q_1 \dots q_n$, their gold answers $g_1 \dots g_n$ and the answer sets $a_1 \dots a_n$ returned by our system, where each answer set a_i can consist of a single value or a list of values, we compute the macro-average precision, recall and F1 score as

$$\begin{aligned} \text{avg. precision} &= \frac{1}{n}\sum_{i=1}^{n} \mathit{precision}(g_i, a_i) \\ \text{avg. recall} &= \frac{1}{n}\sum_{i=1}^{n} \mathit{recall}(g_i, a_i) \\ \text{avg. F1} &= \frac{1}{n}\sum_{i=1}^{n} F1(g_i, a_i) \end{aligned} \tag{5.6}$$

where each of $\mathit{precision}(g_i, a_i)$, $\mathit{recall}(g_i, a_i)$ and $F1(g_i, a_i)$ is computed in the regular way (Perry et al., 1955).

Methods under Comparison. We compare several configurations of QKBfly as follows:

- QKBfly answers questions from an on-the-fly KB dynamically constructed by QKBfly. The KB contains triples as well as higher-arity facts.
- QKBfly-triples is an variant where the on-the-fly KB is limited to *subject-predicate-object* (SPO) triples.

- Sentence-Answers is a text-centric baseline where QKBfly is used to retrieve relevant sentences, but does not perform any fact extraction. Entities in these sentences then become answer candidates.
- QA-Freebase is baseline with a static KB where we apply the same QA method on the huge fact collection of Freebase.

A detailed description of how we implement QA in QKBfly is described in Appendix A.2. The Sentence-Answers baseline differs from the others in the step of collecting answer candidates (Step 3 in the Appendix). Here, all entities that co-occur with one of the question entities in the same sentence are considered as candidate answers. Additionally, the candidate features are the tokens in the sentences (rather than facts) where the candidate occurs. This is in the spirit of a traditional passage-retrieval-based QA method. It uses the same on-the-fly corpus as QKBfly , but does not perform explicit knowledge extraction.

5.7.4.1 Results

Table 5.8 shows the results on the GoogleTrendQuestions benchmark. Here, QKBfly achieves an F1 score of 34.1%, and QKBfly-triples reaches 30.7%. Sentence-Answers and QA-Freebase perform far inferior. Particularly, QA-Freebase returns empty results in most cases due to the lack of facts about recent events. QKBfly performs better than QKBfly-triples in questions which require ternary facts, such as "*Who plays Han Solo in 'Star Wars: The Force Awakens'?*" (see Table 5.11).

Table 5.8: Results on GoogleTrendQuestions.

Method	Precision	Recall	F1
QKBfly	0.330	0.383	0.341
QKBfly-triples	0.294	0.363	0.307
Sentence-Answers	0.173	0.199	0.179
QA-Freebase	0.095	0.100	0.096

To separate the effects of the two kinds of input documents we considered – Wikipedia and Google News –, we also ran QKBfly by using only Wikipedia articles and only the top-10 news articles, respectively. When using only Wikipedia articles to construct the on-the-fly KB, QKBfly achieves 32.4% in the F1 measure. Restricting it to using only Google news, on the other hand, leads to an F1 score of 33.2%. For an end-to-end experiment, we compare QKBfly against the state-of-the-art KB-QA system AQQU (Bast and Haussmann, 2015). AQQU achieves an F1 score of 10%. To be fair, we emphasize that this system has not been designed for a huge but static KB, namely, Freebase, and cannot utilize any on-the-fly knowledge. Table 5.9 shows anecdotal samples for illustration.

Table 5.9: Sample results for GoogleTrendQuestions.

Question & Answers
Where was Pope Francis born?
Gold Answers: [Buenos_Aires]
AQQU: [Buenos_Aires]
QKBfly: [Buenos_Aires]
Who shot Keith Lamont Scott?
Gold Answer: [Brentley_Vinson]
AQQU: []
QKBfly: [a black officer, Brentley_Vinson]
When was the Iran Nuclear Deal signed?
Gold Answer: [14 July 2015]
AQQU: [ir]
QKBfly: [14 July 2015, 17 July 2015, 2015]

5.8 Summary

In this chapter, we present QKBfly, a novel approach to build on-the-fly knowledge bases in a query-driven manner. We acquire facts for a much larger set of predicates than those in mainstream KBs. In contrast to the output of Open IE, arguments of facts are canonicalized, so that they refer to unique entities with semantically typed predicates derived from clusters of phrases. Moreover, QKBfly is not limited to binary facts and comprises also higher-arity facts. Use cases for QKBfly include ad-hoc question answering, summarization and other kinds of machine-reading applications.

Table 5.10: Experimental results on fact extraction (95% confidence intervals).

Method	Triple Facts		Higher-arity Facts		Avg. Run-time (s)
	Precision	#Extractions	Precision	#Extractions	per Document
DEFIE	0.62 ± 0.06	39,684	—	—	unknown
QKBfly	0.67 ± 0.06	44,605	0.63 ± 0.06	25,025	0.88 ± 0.03
QKBfly-pipeline	0.62 ± 0.06	44,605	0.58 ± 0.06	25,025	0.85 ± 0.03
QKBfly-noun	0.73 ± 0.06	33,400	0.68 ± 0.06	16,626	0.76 ± 0.02

Table 5.11: Sample questions and relevant facts extracted by QKBfly. Final answers are marked with an asterisk.

Question	Fact
Who plays Han Solo in 'Star Wars: The Force Awakens'?	⟨Harrison_Ford*, act_in, Han_Solo, Star_Wars:_The_Force_Awakens⟩ ⟨Harrison_Ford*, return_in_as, *"Star Wars film series"*, Han_Solo⟩
Which band was playing during the Paris attacks?	⟨Eagles_of_Death_Metal*, play_in, *"a concert"*, Paris⟩ ⟨Eagles_of_Death_Metal*, injured_in_in, *"the attack"*, Paris⟩
Who killed Cecil the lion?	⟨Walter_Palmer*, kill, *"Cecil the lion"*⟩ ⟨Walter_Palmer*, admit, *"to killing Cecil the lion in July"*⟩
Where was Pope Francis born?	⟨Pope_Francis, born_in_on, Buenos_Aires*, *"17 December 1936"*⟩ ⟨Pope_Francis, birth_place, Buenos_Aires*⟩
Who shot Keith Lamont Scott?	⟨Keith_Lamont_Scott, shot_by, *"a black officer"**⟩ ⟨Brently_Vinson*, shoot, *"Mr Scott"*⟩

Chapter 6

Conclusions

6.1 Contributions

This dissertation addresses important and challenging tasks in the field of information and knowledge extraction. Specifically, we focus on three challenges: *high quality for entity recognition and disambiguation*, *trade-off between coverage and quality of relation extraction*, and *on-the-fly knowledge acquisition.* To this end, we develop new methods to advance the state of the art.

The first contribution, J-NERD, presents a novel probabilistic graphical model for the joint recognition and disambiguation of named-entity mentions in natural-language text. J-NERD considers a rich feature model, including domain-oriented feature and syntactic dependency feature about verbal patterns from dependency parse trees. The salience of J-NERD comes from a supervised, non-linear graphical model that combines multiple per-sentence *tree-shaped* models into an entity-coherence-aware global model. J-NERD detects mention spans, tags them with coarse-grained types, and maps them to entities in a single joint-inference step, which experimentally verifies improvements in quality over the existing named entity recognition and disambiguation systems.

The second contribution, J-REED, presents a novel joint model for relation extraction and entity disambiguation on Wikipedia-style text. J-REED is based on graphical models that capture interdependencies between entity disambiguation and the relational phrases, in particular by considering which lexical types of entities are compatible with the type signature of which relation. Thereby, J-REED improves the extraction quality over pipelined combinations of the state-of-the-art Open IE systems and entity disambiguation systems. J-REED extracts high quality information by mapping entity mentions to the entities in a background knowledge base, and by making relation patterns as crisp as possible. Additionally, without particular assumptions about the target relations, J-REED can capture thousands of interesting relations.

The third contribution, QKBfly, presents a novel end-to-end system for constructing query-driven, on-the-fly KBs. QKBfly takes as input an entity-centric query or a natural-language question, automatically retrieves relevant source doc-

uments (via Wikipedia and news sources), runs a novel form of knowledge extraction on the sources, and builds a high-coverage knowledge base that is focused on the entities of interest. At the heart of QKBfly is a semantic-graph representation of sentences that captures per-sentence clauses, noun-phrases, pronouns, as well as their syntactic and semantic dependencies. Therefore, QKBfly is not limited to binary predicates but can also extract ternary (or higher-arity) predicates. Compared to mainstream KBs, we acquire facts for a much larger set of predicates. Compared to Open IE methods, arguments of facts are canonicalized, thus referring to unique entities with semantically typed predicates derived from precomputed clusters of phrases.

6.2 Outlook

While a number of key problems have been addressed in this dissertation, there are many future possibilities which can be extended with our work.

6.2.1 Joint Inference at Feature-Level for Relation Extraction and Entity Disambiguation

J-REED presents a join model for relation extraction and entity disambiguation based on graphical models. The first model, particularly a CRF model, extracts the relation pattern candidates (with weights). Consequently, all relation candidates and entity candidates serve as input to the join inference in the second graphical model. However, it would be even better if the two models are fused at a lower level, for example, by modeling one CRF model that directly uses all relation extraction features and entity disambiguation features. As for training data, we can make use of existing fact collections (e.g., in a knowledge base) to generate a large training corpus (i.e., *distant supervision*).

6.2.2 Higher-arity Relation Extraction and Entity Disambiguation

J-REED focuses on binary relations between two entities. While QKBfly has already tackled the higher-arity relation extraction problem, –specifically by employing a rule-based system, namely ClausIE, to detect clauses–, further improvements are needed. For example, a machine learning based model to perform joint inference between higher-arity relation extraction and entity disambiguation would be a nice research direction.

6.2.3 On-the-Fly Relation Paraphrase Mining

QKBfly presents an end-to-end system for knowledge acquisition. In the knowledge base construction step, QKBfly harnesses the pattern repository, specifically the dictionaries of relational paraphrases from PATTY, to canonicalize relational

patterns. Although, these dictionaries contains a large number of relational paraphrases, they are far from complete. On-the-fly relation paraphrase mining, especially for unseen patterns, may bring great benefits.

Bibliography

Auer, S., Bizer, C., Kobilarov, G., Lehmann, J., and Ives, Z. (2007). DBpedia: A Nucleus for a Web of Open Data. In *Proceedings of the International Semantic Web Conference (ISWC '07)*, pages 11–15. Springer.

Bamman, D., Underwood, T., and Smith, N. (2014). A Bayesian Mixed Effects Model of Literary Character. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '14)*, pages 370–379. ACL.

Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., and Etzioni, O. (2007). Open Information Extraction from the Web. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '07)*, pages 2670–2676. Morgan Kaufmann Publishers Inc.

Bast, H. and Haussmann, E. (2015). More Accurate Question Answering on Freebase. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM '15)*, pages 1431–1440. ACM.

Bellman, R. E. (2003). *Dynamic Programming*. Dover Publications Inc.

Bentivogli, L., Forner, P., Giuliano, C., Marchetti, A., Pianta, E., and Tymoshenko, K. (2010). Extending English ACE 2005 Corpus Annotation with Ground-truth Links to Wikipedia. In *Proceedings of the International Conference on Computational Linguistics (CICLing '10)*, pages 19–27. COLING.

Bentivogli, L., Forner, P., Magnini, B., and Pianta, E. (2004). Revising the Wordnet Domains Hierarchy: Semantics, Coverage and Balancing. In *Proceedings of the Workshop on Multilingual Linguistic Ressources (MLR '04)*, pages 101–108. ACL.

Berant, J., Chou, A., Frostig, R., and Liang, P. (2013). Semantic Parsing on Freebase from Question-Answer Pairs. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '13)*, pages 1533–1544. ACL.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '08)*, pages 1247–1250. ACM.

Bovi, C. D., Telesca, L., and Navigli, R. (2015). Large-Scale Information Extraction from Textual Definitions through Deep Syntactic and Semantic Analysis. *Transactions of the Association for Computational Linguistics*, 3:529–543.

Brin, S. (1998). Extracting Patterns and Relations from the World Wide Web. In *Proceedings of the International Workshop on the Web and Databases (WebDB '98)*, pages 172–183.

Broder, A. Z., Charikar, M., Frieze, A. M., and Mitzenmacher, M. (1998). Min-wise Independent Permutations. *Journal of Computer and System Sciences*, pages 327–336.

Bunescu, R. and Pasca, M. (2006). Using Encyclopedic Knowledge for Named Entity Disambiguation. In *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics (EACL '06)*, pages 9–16. ACL.

Carletta, J. (1996). Assessing Agreement on Classification Tasks: The Kappa Statistic. *Computational Linguistics*, 22(2):249–254.

Carlson, A., Betteridge, J., Kisiel, B., Settles, B., Jr., E. R. H., and Mitchell, T. M. (2010). Toward an Architecture for Never-Ending Language Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '10)*, pages 1306–1313. AAAI Press.

Carmel, D., Chang, M.-W., Gabrilovich, E., Hsu, B.-J. P., and Wang, K. (2014). ERD'14: Entity Recognition and Disambiguation Challenge. *SIGIR Forum*, 48(2):63–77.

Chang, A. X. and Manning, C. (2012). SUTime: A Library for Recognizing and Normalizing Time Expressions. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC '12)*, pages 3735–3740. ELRA.

Chiticariu, L., Krishnamurthy, R., Li, Y., Raghavan, S., Reiss, F. R., and Vaithyanathan, S. (2010). SystemT: An Algebraic Approach to Declarative Information Extraction. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '10)*, pages 128–137. ACL.

Chiticariu, L., Li, Y., and Reiss, F. R. (2013). Rule-Based Information Extraction is Dead! Long Live Rule-Based Information Extraction Systems! In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '13)*, pages 827–832. ACL.

Cornolti, M., Ferragina, P., and Ciaramita, M. (2013). A Framework for Benchmarking Entity-Annotation Systems. In *Proceedings of the International World Wide Web Conference (WWW '13)*, pages 249–260. ACM.

Cornolti, M., Ferragina, P., Ciaramita, M., Schütze, H., and Rüd, S. (2014). The SMAPH System for Query Entity Recognition and Disambiguation. In *Proceedings of the International Workshop on Entity Recognition and Disambiguation (ERD '14)*, pages 25–30. ACM.

Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T. M., Nigam, K., and Slattery, S. (2000). Learning to Construct Knowledge Bases from the World Wide Web. *Artificial Intelligence*, 118(1-2):69–113.

Cucerzan, S. (2007). Large-Scale Named Entity Disambiguation Based on Wikipedia Data. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '07)*, pages 708–716.

Cucerzan, S. (2014). Name Entities Made Obvious: The Participation in the ERD 2014 Evaluation. In *Proceedings of the International Workshop on Entity Recognition and Disambiguation (ERD '14)*, pages 95–100. ACM.

Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Aswani, N., Roberts, I., Gorrell, G., Funk, A., Roberts, A., Damljanovic, D., Heitz, T., Greenwood, M. A., Saggion, H., Petrak, J., Li, Y., and Peters, W. (2011). *Text Processing with GATE*. University of Sheffield.

Daiber, J., Jakob, M., Hokamp, C., and Mendes, P. N. (2013). Improving Efficiency and Accuracy in Multilingual Entity Extraction. In *Proceedings of the International Conference on Semantic Systems (I-SEMANTICS '13)*, pages 121–124. ACM.

de Marneffe, M.-C., MacCartney, B., and Manning, C. D. (2006). Generating Typed Dependency Parses from Phrase Structure Parses. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC '06)*, pages 449–454. ELRA.

Del Corro, L. and Gemulla, R. (2013). ClausIE: Clause-based Open Information Extraction. In *Proceedings of the International World Wide Web Conference (WWW '13)*, pages 355–366. ACM.

Dill, S., Eiron, N., Gibson, D., Gruhl, D., Guha, R., Jhingran, A., Kanungo, T., Rajagopalan, S., Tomkins, A., Tomlin, J. A., and Zien, J. Y. (2003). SemTag and Seeker: Bootstrapping the Semantic Web via Automated Semantic Annotation. In *Proceedings of the International World Wide Web Conference (WWW '03)*, pages 178–186. ACM.

Doddington, G., Mitchell, A., Przybocki, M., Ramshaw, L., Strassel, S., and Weischedel, R. (2004). The Automatic Content Extraction (ACE) Program Tasks, Data, and Evaluation. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC '04)*, pages 837–840. ELRA.

Domingos, P. and Lowd, D. (2009). *Markov Logic: An Interface Layer for Artificial Intelligence.* Morgan and Claypool Publishers.

Durrett, G. and Klein, D. (2014). A Joint Model for Entity Analysis: Coreference, Typing, and Linking. *Transactions of the Association for Computational Linguistics*, 2:477–490.

Dylla, M., Miliaraki, I., and Theobald, M. (2013). Top-k Query Processing in Probabilistic Databases with Non-Materialized Views. In *Proceedings of the Annual IEEE International Conference on Data Engineering (ICDE '13)*. IEEE Computer Society.

Dylla, M., Theobald, M., and Miliaraki, I. (2014). Querying and Learning in Probabilistic Databases. In *Reasoning Web (RW '14)*, pages 313–368. Springer.

Etzioni, O., Fader, A., Christensen, J., Soderland, S., and Mausam, M. (2011). Open Information Extraction: The Second Generation. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI '11)*, pages 3–10. AAAI Press.

Fader, A., Soderland, S., and Etzioni, O. (2011). Identifying Relations for Open Information Extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11)*, pages 1535–1545. ACL.

Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., and Lin, C.-J. (2008). LIBLINEAR: A Library for Large Linear Classification. *Journal of Machine Learning Research*, 9:1871–1874.

Feng, V. W. and Hirst, G. (2012). Text-level Discourse Parsing with Rich Linguistic Features. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '12)*, pages 60–68. ACL.

Ferragina, P. and Scaiella, U. (2010). TAGME: On-the-fly Annotation of Short Text Fragments (by Wikipedia Entities). In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM '10)*, pages 1625–1628. ACM.

Finkel, J. R., Grenager, T., and Manning, C. (2005). Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '05)*, pages 363–370. ACL.

Fleischman, M. and Hovy, E. (2002). Fine Grained Classification of Named Entities. In *Proceedings of the International Conference on Computational Linguistics (COLING '02)*, pages 1–7. ACL.

Galárraga, L., Heitz, G., Murphy, K., and Suchanek, F. M. (2014). Canonicalizing Open Knowledge Bases. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM '14)*, pages 1679–1688. ACM.

Gatterbauer, W. and Suciu, D. (2017). Dissociation and Propagation for Approximate Lifted Inference with Standard Relational Database Management Systems. *VLDB Journal*, 26(1):5–30.

Gionis, A., Indyk, P., and Motwani, R. (1999). Similarity Search in High Dimensions via Hashing. In *Proceedings of the International Conference on Very Large Databases (VLDB '99)*, pages 518–529. VLDB Endowment.

Gribkoff, E. and Suciu, D. (2016). SlimShot: In-database Probabilistic Inference for Knowledge Bases. *Proceedings of the VLDB Endowment*, 9(7):552–563.

Grishman, R. and Sundheim, B. (1996). Message Understanding Conference-6: A Brief History. In *Proceedings of the International Conference on Computational Linguistics (COLING '96)*, pages 466–471. ACL.

Hearst, M. A. (1999). Untangling Text Data Mining. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '99)*, pages 3–10. ACL.

Hernault, H., Prendinger, H., duVerle, D. A., and Ishizuka, M. (2010). HILDA: A Discourse Parser Using Support Vector Machine Classification. *Dialogue and Discourse*, 1(3):1–33.

Hirschman, L. and Chinchor, N. (1998). Coreference Task Definition. In *Proceedings of the Message Understanding Conference (MUC '98)*. ACL.

Hoffart, J., Seufert, S., Nguyen, D. B., Theobald, M., and Weikum, G. (2012). KORE: Keyphrase Overlap Relatedness for Entity Disambiguation. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM '12)*, pages 545–554. ACM.

Hoffart, J., Suchanek, F. M., Berberich, K., Lewis-Kelham, E., de Melo, G., and Weikum, G. (2011a). YAGO2: Exploring and Querying World Knowledge in Time, Space, Context, and Many Languages. In *Proceedings of the International World Wide Web Conference (WWW '11)*, pages 229–232. ACM.

Hoffart, J., Suchanek, F. M., Berberich, K., and Weikum, G. (2013). YAGO2: A Spatially and Temporally Enhanced Knowledge Base from Wikipedia. *Artificial Intelligence*, 194:28–61.

Hoffart, J., Yosef, M. A., Bordino, I., Fürstenau, H., Pinkal, M., Spaniol, M., Taneva, B., Thater, S., and Weikum, G. (2011b). Robust Disambiguation of Named Entities in Text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP '11)*, pages 782–792. ACL.

Hoffmann, R., Zhang, C., and Weld, D. S. (2010). Learning 5000 Relational Extractors. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '10)*, pages 286–295. ACL.

Indyk, P. and Motwani, R. (1998). Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality. In *Proceedings of the ACM Symposium on Theory of Computing (STOC '98)*, pages 604–613. ACM.

Klein, D. and Manning, C. D. (2003). Accurate Unlexicalized Parsing. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '03)*, pages 423–430. ACL.

Koller, D., Friedman, N., Getoor, L., and Taskar, B. (2007). Graphical Models in a Nutshell. In *An Introduction to Statistical Relational Learning*. MIT Press.

Kulkarni, S., Singh, A., Ramakrishnan, G., and Chakrabarti, S. (2009). Collective Annotation of Wikipedia Entities in Web Text. In *Proceedings of the SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '09)*, pages 457–466. ACM.

Kuzey, E. and Weikum, G. (2012). Extraction of Temporal Facts and Events from Wikipedia. In *Proceedings of the Temporal Web Analytics Workshop (TempWeb '12)*, pages 25–32. ACM.

Kuzey, E. and Weikum, G. (2014). EVIN: Building a Knowledge Base of Events. In *Proceedings of the International World Wide Web Conference (WWW '14)*, pages 103–106. ACM.

Li, K., Zhou, X., Wang, D. Z., Grant, C., Dobra, A., and Dudley, C. (2017). In-database Batch and Query-time Inference over Probabilistic Graphical Models Using UDA-GIST. *VLDB Journal*, 26(2):177–201.

Li, Q. and Ji, H. (2014). Incremental Joint Extraction of Entity Mentions and Relations. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '14)*, pages 402–412. ACL.

Ling, X. and Weld, D. S. (2010). Temporal Information Extraction. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '10)*, pages 1385–1390. AAAI Press.

Ling, X. and Weld, D. S. (2012). Fine-grained Entity Recognition. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI '12)*. AAAI Press.

Liu, D. C. and Nocedal, J. (1989). On the Limited Memory BFGS Method for Large Scale Optimization. *Mathematical Programming*, 45(3):503–528.

Magnini, B. and Cavaglià, G. (2000). Integrating Subject Field Codes into Wordnet. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC '00)*, pages 1413–1418. ELRA.

Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J., Bethard, S. J., and McClosky, D. (2014). The Stanford CoreNLP Natural Language Processing Toolkit. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '14)*, pages 55–60. ACL.

Mausam, Schmitz, M., Bart, R., Soderland, S., and Etzioni, O. (2012). Open Language Learning for Information Extraction. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '12)*, pages 523–534. ACL.

McCallum, A. and Li, W. (2003). Early Results for Named Entity Recognition with Conditional Random Fields, Feature Induction and Web-enhanced Lexicons. In *Proceedings of the Conference on Natural Language Learning (CoNLL '03)*, pages 188–191. ACL.

Meij, E., Weerkamp, W., and de Rijke, M. (2012). Adding Semantics to Microblog Posts. In *Proceedings of the International Conference on Web Search and Data Mining (WSDM '12)*, pages 563–572. ACM.

Mendes, P. N., Jakob, M., García-Silva, A., and Bizer, C. (2011). DBpedia Spotlight: Shedding Light on the Web of Documents. In *Proceedings of the International Conference on Semantic Systems (I-SEMANTICS '11)*, pages 1–8. ACM.

Miller, G. A. (1995). WordNet: A Lexical Database for English. *Communications of the ACM*, 38(11):39–41.

Milne, D. and Witten, I. H. (2008). Learning to Link with Wikipedia. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM '08)*, pages 509–518. ACM.

Milne, D. and Witten, I. H. (2013). An Open-source Toolkit for Mining Wikipedia. *Artificial Intelligence*, 194:222–239.

Mintz, M., Bills, S., Snow, R., and Jurafsky, D. (2009). Distant Supervision for Relation Extraction without Labeled Data. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '09)*, pages 1003–1011. ACL.

Moro, A., Raganato, A., and Navigli, R. (2014). Entity Linking meets Word Sense Disambiguation: a Unified Approach. *Transactions of the Association for Computational Linguistics*, 2:231–244.

Nakashole, N., Tylenda, T., and Weikum, G. (2013). Fine-grained Semantic Typing of Emerging Entities. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL '13)*, pages 1488–1497. ACL.

Nakashole, N., Weikum, G., and Suchanek, F. (2012). PATTY: A Taxonomy of Relational Patterns with Semantic Types. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL '12)*, pages 1135–1145. ACL.

Nguyen, D. B., Abujabal, A., Tran, N. K., Theobald, M., and Weikum, G. (2018). Query-Driven On-The-Fly Knowledge Base Construction. *Proceedings of the VLDB Endowment*, 11.

Nguyen, D. B., Hoffart, J., Theobald, M., and Weikum, G. (2014). AIDA-light: High-Throughput Named-Entity Disambiguation. In *Proceedings of the Linked Data on the Web Workshop (LDOW '14)*. CEUR-WS.org.

Nguyen, D. B., Theobald, M., and Weikum, G. (2016). J-NERD: Joint Named Entity Recognition and Disambiguation with Rich Linguistic Features. *Transactions of the Association for Computational Linguistics*, 4:215–229.

Nguyen, D. B., Theobald, M., and Weikum, G. (2017). J-REED: Joint Relation Extraction and Entity Disambiguation. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM '17)*. ACM.

Niu, F., Zhang, C., Re, C., and Shavlik, J. W. (2012). DeepDive: Web-scale Knowledge-base Construction using Statistical Learning and Inference. In *Proceedings of the International Workshop on Searching and Integrating New Web Data Sources (VLDS '12)*, pages 25–28. CEUR-WS.org.

Nivre, J. and Hall, J. (2005). Maltparser: A Language-Independent System for Data-Driven Dependency Parsing. In *Proceedings of the International Workshop on Treebanks and Linguistic Theories (TLT '05)*, pages 95–135.

Passos, A., Kumar, V., and McCallum, A. (2014). Lexicon Infused Phrase Embeddings for Named Entity Resolution. In *Proceedings of the Conference on Natural Language Learning (CONLL '14)*, pages 78–86. ACL.

Perry, J. W., Kent, A., and Berry, M. M. (1955). Machine Literature Searching X. Machine Language; Factors Underlying its Design and Development. *American Documentation*, 6(4):242–254.

Quirk, R., Greenbaum, S., Leech, G., and Svartvik, J. (1985). *A Comprehensive Grammar of the English Language*. Longman.

Rahman, A. and Ng, V. (2010). Inducing Fine-grained Semantic Classes via Hierarchical and Collective Classification. In *Proceedings of the International Conference on Computational Linguistics (COLING '10)*, pages 931–939. ACL.

Ratinov, L. and Roth, D. (2009). Design Challenges and Misconceptions in Named Entity Recognition. In *Proceedings of the Conference on Natural Language Learning (CoNLL '09)*, pages 147–155. ACL.

Ratinov, L., Roth, D., Downey, D., and Anderson, M. (2011). Local and Global Algorithms for Disambiguation to Wikipedia. In *Proceedings of the Human Language Technologies (HLT '11)*, pages 1375–1384. ACL.

Reiss, F., Raghavan, S., Krishnamurthy, R., Zhu, H., and Vaithyanathan, S. (2008). An Algebraic Approach to Rule-Based Information Extraction. In *Proceedings of the Annual IEEE International Conference on Data Engineering (ICDE '08)*, pages 933–942. IEEE.

Riedel, S. (2008). Improving the Accuracy and Efficiency of MAP Inference for Markov Logic. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI '08)*, pages 468–475. AUAI Press.

Riedel, S., Yao, L., McCallum, A., and Marlin, B. M. (2013). Relation Extraction with Matrix Factorization and Universal Schemas. In *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (HLT-NAACL '13)*, pages 74–84. ACL.

Santorini, B. (1990). Part-Of-Speech Tagging Guidelines for the Penn Treebank Project. Technical report, Department of Linguistics, University of Pennsylvania.

Shin, J., Wu, S., Wang, F., De Sa, C., Zhang, C., and Ré, C. (2015). Incremental Knowledge Base Construction Using DeepDive. *Proceedings of the VLDB Endowment*, 8(11):1310–1321.

Sil, A. and Yates, A. (2013). Re-ranking for Joint Named-Entity Recognition and Linking. In *Proceedings of the ACM International Conference on Information and Knowledge Management (CIKM '13)*, pages 2369–2374. ACM.

Singh, S., Riedel, S., Martin, B., Zheng, J., and McCallum, A. (2013). Joint Inference of Entities, Relations, and Coreference. In *Proceedings of the Automated Knowledge Base Construction Workshop (AKBC '13)*, pages 1–6. ACM.

Sozio, M. and Gionis, A. (2010). The Community-search Problem and How to Plan a Successful Cocktail Party. In *Proceedings of the SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '10)*, pages 939–948. ACM.

Spitkovsky, V. I. and Chang, A. X. (2012). A Cross-Lingual Dictionary for English Wikipedia Concepts. In *Proceedings of the International Conference on Language Resources and Evaluation (LREC '12)*. ELRA.

Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: A Core of Semantic Knowledge. In *Proceedings of the International World Wide Web Conference (WWW '07)*, pages 697–706. ACM.

Suchanek, F. M., Sozio, M., and Weikum, G. (2009). SOFIE: A Self-organizing Framework for Information Extraction. In *Proceedings of the International World Wide Web Conference (WWW '09)*, pages 631–640. ACM.

Suciu, D., Olteanu, D., Christopher, R., and Koch, C. (2011). *Probabilistic Databases*. Morgan & Claypool Publishers.

Surdeanu, M. and Ciaramita, M. (2007). Robust Information Extraction with Perceptrons. In *Proceedings of the National Institute of Standards and Technology: Automatic Content Extraction Program (NIST '07)*.

Sutton, C. A. and McCallum, A. (2012). An Introduction to Conditional Random Fields. *Foundations and Trends in Machine Learning*, 4(4):267–373.

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the CoNLL-2003 Shared Task: Language-independent Named Entity Recognition. In *Proceedings of the Conference on Natural Language Learning (CONLL '03)*, pages 142–147. ACL.

Usbeck, R., Röder, M., Ngonga Ngomo, A.-C., Baron, C., Both, A., Brümmer, M., Ceccarelli, D., Cornolti, M., Cherix, D., Eickmann, B., Ferragina, P., Lemke, C., Moro, A., Navigli, R., Piccinno, F., Rizzo, G., Sack, H., Speck, R., Troncy, R., Waitelonis, J., and Wesemann, L. (2015). GERBIL – General Entity Annotation Benchmark Framework. In *Proceedings of the International World Wide Web Conference (WWW '15)*. ACM.

Vrandečić, D. and Krötzsch, M. (2014). Wikidata: A Free Collaborative Knowledgebase. *Communication of the ACM*, 57(10):78–85.

Xu, K., Reddy, S., Feng, Y., Huang, S., and Zhao, D. (2016). Question Answering on Freebase via Relation Extraction and Textual Evidence. In *ACL '16*. ACL.

Xue, N., Ng, H. T., Pradhan, S., Prasad, R., Bryant, C., and Rutherford, A. (2015). The CoNLL-2015 Shared Task on Shallow Discourse Parsing. In *Proceedings of the Conference on Natural Language Learning (CoNLL '15)*, pages 1–16. ACL.

Yosef, M. A., Bauer, S., Hoffart, J., Spaniol, M., and Weikum, G. (2012). HYENA: Hierarchical Type Classification for Entity Names. In *Proceedings of the International Conference on Computational Linguistics (COLING '12)*, pages 1361–1370. ACL.

Yosef, M. A., Hoffart, J., Bordino, I., Spaniol, M., and Weikum, G. (2011). AIDA: An Online Tool for Accurate Disambiguation of Named Entities in Text and Tables. *Proceedings of the VLDB Endowment*, 4(12):1450–1453.

Zhang, C., Shin, J., Ré, C., Cafarella, M., and Niu, F. (2016). Extracting Databases from Dark Data with DeepDive. In *Proceedings of the ACM SIGMOD International Conference on Management of Data (SIGMOD '16)*, pages 847–859. ACM.

Appendices

Chapter A

Additional Details

A.1 ILP Setup

We next describe how we translate the densest-subgraph problem $\mathcal{S}^*$ into an ILP. First, we introduce:

- A binary variable cnd_{i_j} for each `noun-phrase` or `pronoun` node n_i and each `entity` candidate $e_{i_j} \in ent(n_i, \mathcal{G})$. The variable $cnd_{i_j} = 1$ iff e_{i_j} is chosen for n_i in the densest subgraph $\mathcal{S}^*$.
- A binary variable $joint\text{-}rel_{i_j t_k}$ for each pair of `noun-phrase` or `pronoun` nodes n_i, n_t, which are connected by a `relation` edge $r_{i,t}$, and similarly for each pair of `entity` nodes $e_{i_j} \in ent(n_i, \mathcal{G})$, $e_{t_k} \in ent(n_t, \mathcal{G})$. Thus, the variable $joint\text{-}rel_{i_j t_k} = 1$ iff e_{i_j} is chosen for n_i and e_{t_k} is chosen for n_t in $\mathcal{S}^*$.

We consider the following constraints:

- $\sum_j cnd_{i_j} = 1 \ \ \forall i$,
- two `noun-phrase` or `pronoun` nodes n_i, n_t are linked by a `sameAs` edge in $\mathcal{S}^*$ iff $cnd_{i_j} = cnd_{t_j} \ \ \forall j$,
- $joint\text{-}rel_{i_j t_k} = 1$ iff $cnd_{i_j} = 1$ and $cnd_{t_k} = 1$.

We then aim to maximize

$$\begin{aligned} &\sum_{\substack{n_i \in \mathcal{G} \\ e_{i_j} \in ent(n_i, \mathcal{G})}} cnd_{i_j} \ \cdot \ w(n_i, e_{i_j}) + \\ &\sum_{n_i, n_t \in \mathcal{G}} joint\text{-}rel_{i_j t_k} \ \cdot \ w(n_i, n_t, \mathcal{S}^{i_j t_k}) \end{aligned} \tag{A.1}$$

where:

- $w(n_i, e_{i_j})$ is the `means` edge weight between n_i and e_{i_j},
- $w(n_i, n_t, \mathcal{S}^{i_j t_k})$ is the `relation` edge weight between n_i and n_t in a subgraph $\mathcal{S}^{i_j t_k}$ constructed from $\mathcal{G}$ by only considering e_{i_j} and e_{t_k} as the candidates for n_i and n_t, respectively.

A.2 QA Setup

Question answering over structured knowledge bases (KB-QA) (Berant et al., 2013; Bast and Haussmann, 2015; Xu et al., 2016) denotes the task of translating a natural language question into a structured query (e.g., using SPARQL for querying SPO triples), which is then executed over the underlying KB (e.g., Freebase (Bollacker et al., 2008)) to obtain answer entities. As an extrinsic use-case, we harness QKBfly for KB-QA. In contrast to mainstream works, we pursue the case where no fact repository is available upfront and all relevant facts need to be gathered on-the-fly, triggered by a natural-language question. Specifically, when given a question like "*who did vladimir lenin marry?*", QKBfly computes the answers in the following four steps.

Step 1. Entities in the question are detected and used to retrieve relevant documents in Wikipedia and Google News. For example, we use the Wikipedia article that has the id of `Vladimir_Lenin`, and we issue a Google News query with the full text of the input question. For each question, we retrieve the top-10 results from Google News.

Step 2. QKBfly processes the retrieved documents to build an on-the-fly KB of facts. No pre-existing fact repository is used.

Step 3. As answer candidates, our method fetches all entities (or string literals like dates) from its question-specific ad-hoc KB. A type filter is applied to ensure that candidates satisfy the expected answer type(s). For example, a question starting with *"Who"* can be answered only by entities of types `PERSON`, `CHARACTER` or `ORGANIZATION`. Here, we use our type system based on infoboxes for entities, combined with Stanford NER and SUTime tags. Note that this step is focused on recall, ensuring that we do not miss good answers. The following step ensures high precision by further filtering the candidates and ranking them.

Step 4. We run each answer candidate through a pre-trained binary classifier, using an SVM model. The model is trained on the WebQuestions training questions and their gold-standard answers (Berant et al., 2013). The positively labeled candidates are output as final answers. For single-answer factoid questions (if detectable), only the top-ranked answer is output.

Classifier Features. For each question, we extract all tokens: word-level lemmatized unigrams and entities. For example, the question "*who did vladimir lenin marry?*" contains "*who*", "*do*", "*marry*" and the entity `Vladimir_Lenin`. For each candidate answer, we similarly extract all tokens co-occurring in the same KB facts, again all word-level lemmatized unigrams and all entities. The feature set for a pair of a question and its candidate answer then are all token pairs (x, y) where x is a token occurring with the question and y is a token occurring with the candidate. For data sparseness and simplicity, we treat these as binary features.

Classifier Training. WebQuestions consists of 3,778 training questions, each of which is paired with its answer set. These were collected using the Google Suggest

API and further crowdsourcing. Answers (i.e., Freebase entities) are finally converted to Wikipedia pages by the Freebase API. We use the gold answers of the WebQuestions training corpus and Wikipedia-based question-specific KBs produced by QKBfly for training the SVM classifier. Facts extracted by QKBfly that contain correct or incorrect answers are used as positive or negative training samples, respectively. The model is constructed using the Liblinear SVM library (Fan et al., 2008) by using the default settings.

List of Figures

List of Tables

List of Algorithms