Information Extraction as a Filtering Task

Henning Wachsmuth
University of Paderborn, s-lab
Paderborn, Germany
hwachsmuth@s-lab.upb.de

Benno Stein
Bauhaus-Universität Weimar
Weimar, Germany
benno.stein@uni-weimar.de

Gregor Engels
University of Paderborn, s-lab
Paderborn, Germany
engels@upb.de

ABSTRACT

Information extraction is usually approached as an annotation task: Input texts run through several analysis steps of an extraction process in which different semantic concepts are annotated and matched against the slots of templates. We argue that such an approach lacks an efficient control of the input of the analysis steps. In this paper, we hence propose and evaluate a model and a formal approach that consistently put the *filtering view* in the focus: Before spending annotation effort, filter those portions of the input texts that may contain relevant information for filling a template and discard the others. We model all dependencies between the semantic concepts sought for with a truth maintenance system, which then efficiently infers the portions of text to be annotated in each analysis step. The filtering view enables an information extraction system (1) to annotate only relevant portions of input texts and (2) to easily trade its run-time efficiency for its recall. We provide our approach as an open-source extension of Apache UIMA and we show the potential of our approach in a number of experiments.

Categories and Subject Descriptors

I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*Text analysis*; F.2.3 [**Analysis of Algorithms and Problem Complexity**]: Tradeoffs among Complexity Measures; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*Nonmonotonic reasoning and belief revision*; H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*Information filtering*; B.2.1 [**Arithmetic and Logic Structures**]: Design Styles—*Pipeline*

Keywords

information extraction; filtering; run-time efficiency; truth maintenance; relevance

CIKM'13, Oct. 27–Nov. 1, 2013, San Francisco, CA, USA.

ACM 978-1-4503-2263-8/13/10 ...$15.00.
http://dx.doi.org/10.1145/2505515.2505557.

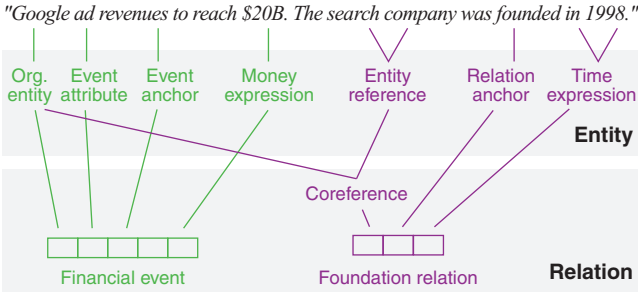

Figure 1: A sample text that contains information matching the slots of templates of a financial event (left) and of a foundation relation (right). The various shown semantic concepts can be abstracted into the two types *Entity* and *Relation*.

1. INTRODUCTION

Information extraction aims at identifying various semantic concepts in natural language texts, ranging from named entities, numeric expressions, and attributes over references to relations, events, and their anchors [24]. From an extraction viewpoint, these concepts can be unified into two abstract types of information, as illustrated in Figure 1: An *entity type*, whose instances are represented by spans of text, and a *relation type*, whose instances are expressed in a text, indicating relations between two or more entities. This unification reflects the view of information extraction as filling the (entity) slots of (relation) templates.

Usually, information extraction is approached as an *annotation task* where an input text is annotated in a scheduled process of extraction steps. Given an extraction problem, such an approach first performs certain lexical analyses (e.g. tokenization or part-of-speech tagging) followed by an annotation of all relevant entity types in the whole text. Next, it conducts syntactic analyses (e.g. dependency parsing) that are needed to extract relations between the annotated entities as well as to resolve coreferences [1].

In times of big data, the need for efficiency in information extraction is constantly growing because extraction problems of increasing complexity have to be tackled under strict time constraints on increasing numbers of input texts. While existing information extraction systems control the process of creating all output sought for, they often lack an efficient control of the processed input. In particular, much effort is spent for annotating portions of input texts that are not relevant for a given extraction problem at all. Assume, for in-

stance, that the financial event template in Figure 1 requires an appropriate time expression (which is missing). Then the effort of filling its other slots is wasted. In order to improve the run-time efficiency of extraction, it hence makes sense to discard irrelevant portions of text as early as possible, thereby filtering only the relevant portions.

The idea of filtering for improving efficiency is well-known in information extraction [8] as well as in areas that combine retrieval and extraction techniques such as question answering [9]. Until today, however, many information extraction systems either do not incorporate filtering, or they filter relevant portions of text only based on vague statistical models or hand-coded heuristics (cf. Section 2 for details).

In this paper, we propose to consistently address information extraction as a *filtering task*, which means to infer, annotate and filter only relevant portions of text in each extraction step. In case of the text in Figure 1, an approach that performs filtering on the sentence level might e.g. annotate time expressions first. Only the second sentence remains relevant then and, thus, needs to be filtered. Depending on the schedule of the extraction steps, that sentence sooner or later turns out not to contain a financial event, and so on.[1]

To enable filtering, we model the dependencies between all entity and relation types that are relevant for the extraction problem at hand (Section 3). In addition, we manually specify a *degree of filtering* for each relation type, i.e., the unit of text (e.g. a sentence or a paragraph) all dependent types of information must lie within. Small units favor extraction efficiency over recall, whereas efficiency can also be optimized without losing effectiveness by specifying degrees that match the actual analyses (e.g. most binary relation extractors analyze only entity pairs within sentence boundaries).

As outlined above, the relevance of a portion of text is non-monotonic within a process of extraction steps. To handle non-monotonicity, we efficiently analyze the modeled dependencies with a *truth maintenance system* [36] that formally determines in advance whether possible output annotations of an extraction step within a portion of text may contribute to fulfilling the extraction problem at hand. The actually created output annotations are then used to filter the relevant portions of text after each step (Section 4).

We implemented our approach as an open-source software framework on top of Apache UIMA[2] (Section 5). While the efficiency potential of filtering naturally depends on the amount of relevant information in the processed input texts, we evaluate the main parameters of filtering as well as the efficiency of our approach for different extraction problems on text corpora of different languages (Section 6). Our results demonstrate the main contributions of our research:

1. The efficient input control based on truth maintenance enables information extraction systems to formally infer and to annotate only relevant portions of text.

2. The proposed filtering view of information extraction provides an intuitive means to optimize the run-time efficiency of a process of extraction steps and to easily trade run-time efficiency for recall.

3. The realized software framework allows researchers to efficiently address arbitrary extraction problems as filtering tasks with minimum additional effort.

[1]The example indicates that the schedule of the extraction steps affects efficiency, as studied in related work (see Section 2). However, scheduling is not the focus of this paper.
[2]Apache UIMA, `http://uima.apache.org`

2. RELATED WORK

The common view of information extraction originates in the *Message Understanding Conferences*, which focused on the effective extraction of entities and relations from newswire texts in order to fill complex event templates [6]. While current approaches address the same problems of extracting relations [29] and events [28], information extraction is now on the verge of being exploited at web-scale [27] and gradually finds its way into search engines [33]. Still, many approaches fail computational efficiency.

In the last decade, the efficiency of information extraction is getting increasing attention in research and industry [13]. Applied techniques range from parallelization [20], specialized search indexes [4], and relation-oriented search queries [2] over the use of cheap algorithms for preprocessing [3] and extraction [32] to the efficient processing of input texts, e.g. based on string hashing [18] or filtering [1]. We focus on filtering, but we point out that our approach does not render any of the other approaches impossible.

Filtering is very common in tasks where information needs have to be addressed in real-time, such as question answering, which usually includes so called *passage retrieval* to filter relevant portions of input texts [26]. The authors of [5] compare the benefit of statistical and linguistic knowledge for filtering candidate answers, and a fuzzy matching of questions and possibly relevant portions of text is proposed in [9]. As in these examples filtering in question answering usually relies on heuristics and vague statistical models, whereas we approach filtering formally in information extraction.

Information extraction has adopted filtering techniques since the early times [8], mostly to improve efficiency. However, [34] observed that classifying the relevant portions of text before extraction can also improve effectiveness. [23] exploits this for template filling, and [47] filters trustworthy candidate relations in unsupervised relation extraction. Filtering approaches for efficiency purposes often focus on complete texts, e.g. using fast text categorization [41]. In [38], the author complains that techniques for efficiently finding the relevant portions of texts are still restricted to hand-coded heuristics. [30] stresses the importance of filtering for all kinds of extraction problems where relevant information is sparse. There, machine learning is used to *predict* relevant sentences. In contrast, we *infer* the relevant portions of text from the currently available information. Moreover, a restriction to sentences limits effectiveness [42], whereas we provide a means to specify the sizes of filtered portions, thereby trading efficiency for effectiveness.

In [46], we approach the creation of efficient information extraction systems without losing effectiveness, partly by including steps to filter relevant portions of text. Optimal efficiency then results from an optimal schedule of the filtering steps, as shown in [45]. Schedules are also optimized by SystemT [7], which follows the paradigms of *declarative information extraction*: user queries define extraction steps with logical constraints, while the system manages the workflow [12]. SystemT restricts some analyses to *scopes* of a text based on location conditions in a query [40]. We also use queries and scopes in Section 3, but only to determine relevance, which significantly reduces query complexity. While

SystemT is limited to rule-based extraction, we do not place constraints on the kinds of analyses to be performed.

The filtering view targets at the classic and dominant form of information extraction system, i.e., a *pipeline* [11]. A pipeline sequentially applies a set of extraction algorithms to its input. The main frameworks for natural language processing, UIMA [15] and GATE [10], focus on pipelines. In [44], we present a system that builds filtering pipelines automatically for given extraction problems. Without an input control, however, the system cannot tackle different problems at the same time. Because of filtering, we do not consider pipelines that employ more than one algorithm to annotate the same type of information [48], though filtering could be delayed in such cases. Accordingly, iterative pipelines [17] and probabilistic pipelines [22] are not covered in this paper.

Our approach profits from dividing an extraction process into small steps. This contradicts the idea of *joint information extraction* [35]. Still, joint extraction algorithms can be used in pipelines, as is e.g. common in entity recognition [16]. Particularly, our approach benefits complex extraction problems, such as those of the BioNLP task [25]. Nevertheless, it applies to all extraction processes with dependencies between the relevant types of information. To the best of our knowledge, we are the first to consider these dependencies for filtering. Moreover, since our approach performs filtering during the extraction process, it can be integrated with all existing approaches where filtering precedes extraction.

3. INFORMATION EXTRACTION AS A FILTERING TASK

In this section, we propose a filtering view of information extraction, which makes it possible to address every extraction problem as a *filtering task*, i.e, to analyze and filter only the relevant portions of each input text.

3.1 The Relevance of a Portion of Text

In order to infer the relevance of a portion of text u, we need a clear specification of the output sought for in an extraction problem. As motivated in Section 1, we distinguish two types of information: An *entity type*, denoted as E, can be regarded as atomic in that its instances are used to fill slots of templates. In contrast, a *relation type* R expresses a conjunction of types, i.e., a template. Extraction problems may target at different templates concurrently. We represent such a disjunction of conjunctions in the form of a *query*:

Query A query q specifies the relevant types of information in an extraction problem. Its abstract syntax is defined by the following grammar:

$$q ::= q \vee q \mid r$$
$$r ::= R(\, r\, \{,\, r\}^*\,) \mid E$$

A portion of text u is *relevant* at some point of an extraction process, if it may still contain all information needed to fulfill the associated query. As an example, consider the query q_1 that addresses a simple binary relation type:

$$q_1 = \textit{Founded(Organization, Time)}$$

Before extracting relations, only those portions of text are relevant that contain instances of both entity types. If organization entities are e.g. annotated first, then time entities need to be annotated only in portions of texts with organization entities, and vice versa. Hence, we can filter the relevant portions of a text based on the output of an extraction step (and discard the others). Figure 2 shows the portions of a sample text to be filtered for q_1 and opposes the slot-filling view to the filtering view of information extraction.

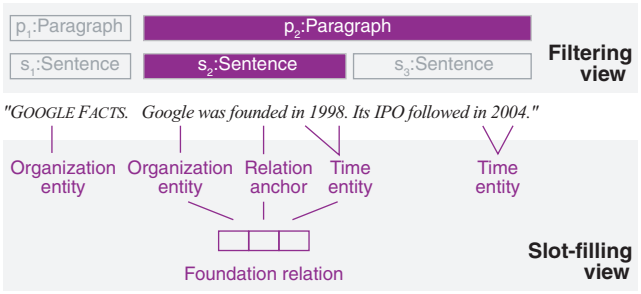

Figure 2: The slot-filling view of information extraction (bottom) and the proposed filtering view (top). For the query q_1 = *Founded(Organization, Time)*, the relevant portion of text is p_2 on the paragraph level and s_2 on the sentence level, respectively.

In general, it is reasonable to filter different portions of text for the different relation types in a query. The following two queries illustrate this. While the former targets at arbitrary forecasts (i.e., statements about the future) with time information, the latter asks for financial relations, which relate such forecasts to money entities:

$$q_2 = \textit{Forecast(Anchor, Time)}$$
$$q_3 = \textit{Financial(Money, } q_2\textit{)}$$

Assume that forecasts are extracted from single sentences while financial relations may span a whole paragraph. Then, a sentence without time entities is irrelevant for q_2, but since it might contain a money entity, its enclosing paragraph remains relevant for q_3 as a whole. In case of disjunctive queries, relevance is even independent for each conjunction:

$$q_4 = q_1 \vee q_3$$

For instance, a portion of text without financial relations can, of course, still contain foundation relations. Generally, every relation type in a query may entail a different set of relevant portions of text at each extraction step. For a given input text, we call such a set a *scope* of that text:

Scope A scope $S_R = (u_1, \ldots, u_n)$ is the ordered set of $n \geq 0$ portions of a text where instances of a relation type $R(r_1, \ldots, r_k)$ may occur.

3.2 Specification of Degrees of Filtering

The concept of scopes reveals that a query q alone does not suffice to perform filtering: While q enables us to automatically infer the relevance of a portion of text u, it does not define the size of the portions of text to be filtered, when given the output of an extraction step. We hence manually assign a *degree of filtering* to each relation type in q that binds instances of the relation type to units of the text:

Degree of Filtering A degree of filtering U is a type of lexical or syntactic text unit that defines the size of a portion of text, all information of an instance of a relation type $R(r_1, \ldots, r_k)$ must lie within, denoted as $U[R(r_1, \ldots, r_k)]$.

The definition accounts for the fact that all extraction algorithms operate on a certain text unit level. E.g., most

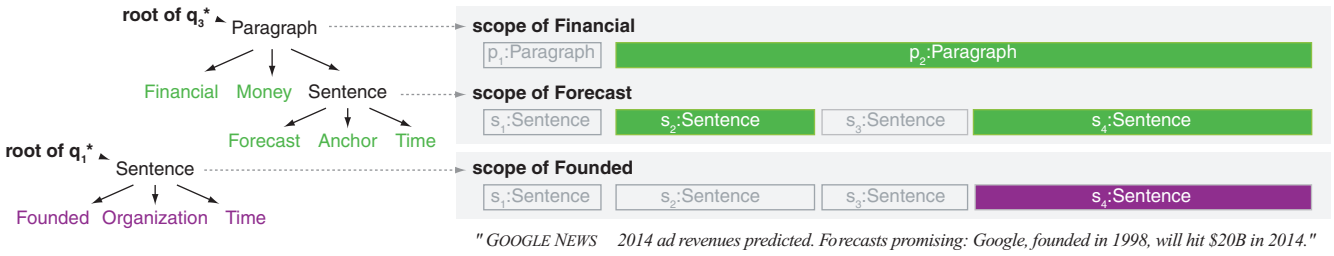

Figure 3: The dependency graph of the scoped query $q_4^* = q_1^* \vee q_3^*$ (left) and a mapping from the degrees of filtering in q_4^* to the associated scopes, which represent the relevant portions of a sample text for q_4^* (right).

binary relation extractors process one *sentence* at a time, taking as input only candidate entity pairs within that sentence. In contrast, coreference resolution algorithms rather analyze *paragraphs* or even the entire *text*.

Degrees of filtering provide a means to influence the trade-off between the efficiency and the effectiveness of extraction: small degrees allow for filtering small portions of text, which positively affects run-time efficiency. Larger degrees provide less room for filtering, but they allow for higher recall if relations exceed the boundaries of smaller text units. When the degrees match the text unit levels of the employed extraction algorithms, efficiency will be improved without losing recall.[3] Hence, the slot-filling view and filtering view of information extraction can be integrated without loss. We call a query with defined degrees of filtering a *scoped query*:

Scoped Query A scoped query q^* is a query q that has assigned a degree of filtering to each contained relation type $R(r_1, \ldots, r_k)$.

3.3 The Dependency Graph of a Query

Within an extraction process, the degrees of filtering in a scoped query q^* are associated to respective scopes of the current input text. These scopes may depend on each other, as the following scoped version of the query q_4 shows:

$$q_4^* = \textit{Sentence[Founded(Organization, Time)]} \\ \vee \textit{ Paragraph[Financial(Money, Sentence[}q_2\textit{])]}$$

According to this query, paragraphs without time entities will never span sentences with forecasts and, thus, will not yield financial relations. Similarly, if a paragraph contains no money entities, then there is no need for extracting forecasts from the paragraph's sentences. So, filtering one of the scopes of *Forecast* and *Financial* affects the other one.[4] As in this example, a query implies hierarchical dependencies between the relevant types of information that can be represented as a *dependency graph*.

Dependency Graph The dependency graph of a scoped query $q^* = q_1 \vee \ldots \vee q_m$ is a set of directed trees with one tree for each conjunction $q_i \in \{q_1, \ldots, q_m\}$. An inner node of q_i corresponds to a degree of filtering and a leaf to an entity type E or a relation type R. An edge from an inner node to a leaf means that the respective degree is assigned to the respective type, and edges between inner nodes imply that the associated scopes are dependent. The degree of filtering of q_i itself defines the root of the tree.

Figure 3 visualizes the dependency graph of q_4^* and the associated scopes of a sample text. Such a graph can be exploited to automatically maintain relevant portions of text.

[3]While there is no clear connection between degrees of filtering and *precision*, a higher precision seems easier to achieve if extraction is performed only on small portions of text.

[4]For complex relation types like *coreference*, the degree of filtering of an inner relation type possibly exceeds the degree of filtering of an outer relation type. In such a case, filtering with respect to the outer relation type affects the entities to be resolved, but not the entities to be used for resolution.

4. MAINTAINING THE RELEVANT PORTIONS OF INPUT TEXTS

We now show how to automatically determine and filter scopes of an input text in each step of an extraction process. This enables the employed extraction algorithms to analyze only portions of text their output is relevant for.

4.1 Input Control using Truth Maintenance

An extraction process can be regarded as non-monotonic in that knowledge about input texts (i.e., annotated entities and relations) changes in each step. In the beginning, usually no knowledge is given and, hence, each portion u of an input text must be assumed relevant. If u lacks any required knowledge in some step, it becomes irrelevant and can be excluded from further analyses. In artificial intelligence, such non-monotonicity is well-studied and it is tackled with an *assumption-based truth maintenance system (ATMS)*, which justifies and retracts assumptions expressed as propositional formulas based on a set of believed assumptions [36].

We adapt the ATMS concept to filter the scopes of input texts. For this purpose, we interpret all queries, entity types and relation types as propositional symbols. Given a scoped query q^*, we then model relevant portions of text as follows. For each degree of filtering U that is a root in the dependency graph of q^*, the relevance $q^{*(u)}$ of each portion of text u in the scope associated with U corresponds to the truth value of the relation type $R(r_1, \ldots, r_k)$, to which U is assigned:

$$\phi^{(u)}: \quad R^{(u)} \wedge r_1^{(u)} \wedge \ldots \wedge r_k^{(u)} \rightarrow q^{*(u)}$$

For all child nodes $r_i^{(u)}$ of a root node U of the form $U_i[R_i(r_{i1}, \ldots, r_{il})]$, we additionally model the relevance of a portion of text u' of the scope associated to U_i as:

$$\psi^{(u')}: \quad R_i^{(u')} \wedge r_{i1}^{(u')} \wedge \ldots \wedge r_{il}^{(u')} \rightarrow r_i^{(u)}$$

This modeling step is repeated recursively until each child node $r_{ij}^{(u')}$ in a new formula $\psi^{(u')}$ represents either an entity type or a relation type. Now, the set of assumptions about an input text is given by the set of all formulas $\phi^{(u)}$ and $\psi^{(u)}$ of that text. In case of the example in Figure 3, four assumptions are initially believed for the paragraph p_2:

Pseudocode 1 DETERMINE SCOPE(Output types **O**)

```
1: for each Output type O in O do
2:     if O is a degree of filtering in the scoped query q* then
3:         return the whole input text
4: Scopes S
5: for each Output type O in O do
6:     Scopes S_O ← all scopes O is relevant for according to q*
7:     S.addAll( S_O)
8: Scope S_∪
9: for each Scope S in S do
10:    for each Portion of text u in S do
11:        if not u intersects with S_∪ then S_∪.add(u)
12:        else S_∪.merge(u)
13: return S_∪
```

Pseudocode 2 FILTER(Scopes **S**, Output types **O**)

```
1: for each Scope S in S do
2:     Output types O' ← all types in O that are relevant for S
3:     for each Portion of text u in S do
4:         if not u contains an instance of any O ∈ O' then
5:             S.remove(u)
6:     Scope S_0 ← S.getRootScope()
7:     if S_0 ≠ S then
8:         for each Portion of text u in S_0 do
9:             if not u intersects with S then S_0.remove(u)
10:    Scopes S' ← S_0.getDescendantScopes()
11:    for each Scope S' ≠ S in S' do
12:        for each Portion of text u in S' do
13:            if not u intersects with S_0 then S'.remove(u)
```

$$
\begin{array}{lll}
\phi^{(p_2)}: & \mathit{Financial}^{(p_2)} \wedge \mathit{Money}^{(p_2)} \wedge q_2^{*(p_2)} & \rightarrow q_3^{*(p_2)} \\
\psi^{(s_2)}: & \mathit{Forecast}^{(s_2)} \wedge \mathit{Anchor}^{(s_2)} \wedge \mathit{Time}^{(s_2)} & \rightarrow q_2^{*(p_2)} \\
\psi^{(s_3)}: & \mathit{Forecast}^{(s_3)} \wedge \mathit{Anchor}^{(s_3)} \wedge \mathit{Time}^{(s_3)} & \rightarrow q_2^{*(p_2)} \\
\psi^{(s_4)}: & \mathit{Forecast}^{(s_4)} \wedge \mathit{Anchor}^{(s_4)} \wedge \mathit{Time}^{(s_4)} & \rightarrow q_2^{*(p_2)}
\end{array}
$$

However, the relevance of a portion of text u at a particular extraction step depends on the set of *currently* believed assumptions. This set follows from the output of all extraction algorithms applied so far. Instead of maintaining all assumptions, we filter the scopes of an input text according to the output of the last algorithm and maintain assumptions for the scopes only. For instance, if time entities are found only in the sentences s_2 and s_4 in Figure 3, then $\psi^{(s_3)}$ becomes false, whereas the other assumptions of p_2 are updated as follows:

$$
\begin{array}{lll}
\phi^{(p_2)}: & \mathit{Financial}^{(p_2)} \wedge \mathit{Money}^{(p_2)} \wedge q_2^{*(p_2)} & \rightarrow q_3^{*(p_2)} \\
\psi^{(s_2)}: & \mathit{Forecast}^{(s_2)} \wedge \mathit{Anchor}^{(s_2)} & \rightarrow q_2^{*(p_2)} \\
\psi^{(s_4)}: & \mathit{Forecast}^{(s_4)} \wedge \mathit{Anchor}^{(s_4)} & \rightarrow q_2^{*(p_2)}
\end{array}
$$

An algorithm employed in an extraction step then has to analyze all current scopes, its *output types* are relevant for:

Output Type An output type O of an extraction algorithm is an entity or relation type annotated by that algorithm.

Below, we describe how to determine and filter the scopes of an input text based on the set of output types **O** of an extraction algorithm. Initially, all scopes span the whole text. These scopes must be generated by *segmentation algorithms*, i.e., extraction algorithms whose output types include a degree of filtering (e.g. a sentence splitter). However, we do not explicitly distinguish algorithm types but simply create scopes when the according output is annotated.

4.2 Determining Relevant Portions of Text

Given a scoped query q^*, an employed extraction algorithm must be applied to each portion of text u, for which an assumption $\phi^{(u)}$ or $\psi^{(u)}$ exists that depends on an output type of the algorithm. E.g., the assumptions $\phi^{(p_2)}$, $\psi^{(s_2)}$, and $\psi^{(s_4)}$ imply that an algorithm with output type *Forecast* needs to analyze the sentences s_2 and s_4. In general, an algorithm with a set of output types **O** must be applied to the union $S_\cup$ of the set **S** of all scopes S that meet one of two conditions: (1) S is associated to a degree of filtering that is assigned to any $O \in \mathbf{O}$ in q^*. (2) S is associated to a degree of filtering assigned to an entity or relation type in q^*, which needs instances of any $O \in \mathbf{O}$ as input. E.g., part-of-speech tags are not specified in q_4^* from Section 3, but they might be necessary for extracting the type *Organization*.

The determination of $S_\cup$ based on a set of output types **O** is sketched in Pseudocode 1. Lines 1–3 identify segmentation algorithms, whereas the scopes to be unified are collected in lines 4–7. The union $S_\cup$ is then obtained by taking all of the scopes' non-overlapping portions of text and by merging overlapping portions (lines 8–12).[5]

4.3 Filtering Relevant Portions of Text

For each analyzed portion of text u, the believed assumptions $\phi^{(u)}$ and $\psi^{(u)}$ containing a type $O \in \mathbf{O}$ of the applied extraction algorithm can be updated based on the algorithm's output. This in turn leads to a recursive update of assumptions that contain the consequent of some $\psi^{(u)}$. In case all forecasts and their anchors have been detected in the sample text in Figure 3, $\psi^{(s_2)}$ and $\psi^{(s_4)}$ turn out to be true. Hence, $q_2^{*(p_2)}$ is true and $\phi^{(p_2)}$ remains in the following form:

$$\phi^{(p_2)}: \quad \mathit{Financial}^{(p_2)} \wedge \mathit{Money}^{(p_2)} \rightarrow q_3^{*(p_2)}$$

Thus, the output of an extraction algorithm is not only used to filter the scopes in the above-mentioned set **S** but also their dependent scopes. The set of dependent scopes of a scope S consists of the scope associated to the root of the node of S in the dependency graph of q^* as well as of all scopes of the root's descendant nodes. This, of course, includes all ancestor scopes of S.

Pseudocode 2 shows how to perform filtering. For each scope $S \in \mathbf{S}$, a portion of text u is maintained only if it contains an instance of one of the output types $\mathbf{O}' \subseteq \mathbf{O}$ that are relevant for S (lines 1–5). Afterwards, lines 6–9 remove all portions of text from the root scope S_0 of S that do not intersect with any portion of text in S.[6] Accordingly, only those portions of text in the set of descendant scopes $\mathbf{S}'$ of S_0 are retained that intersect with a portion in S_0 (lines 10–13).

5. IMPLEMENTATION

We realized all concepts that are needed to address extraction problems as filtering tasks in an efficient Java software framework as an extension of Apache UIMA. The source

[5]For a concise presentation, Pseudocode 1 contains nested loops. Actually, the unification can be realized in time linear in the number of the portions of text of all scopes by stepwise comparing portions according to their ordering in the text.

[6]Similar to a unification, an intersection can be achieved in time linear in the number of text units of all scopes.

code of this *filtering framework* has been designed with a focus on easy integration and minimal additional effort. It can be freely accessed at `http://www.arguana.com`, together with usage instructions and some sample applications.

5.1 Apache UIMA at a Glance

Apache UIMA is a software framework that allows developers to easily compose natural language processing applications while managing the applications' control flow and data flow [15]. For this purpose, algorithms are accompanied by descriptor files with metadata.

Throughout this section, we provide a simplified conceptual view of Apache UIMA. Its architecture is defined by the white classes and their associations on the left of the UML class diagram [31] in Figure 4. Applications input *texts* and analyze them with *aggregate analysis engines* (say, information extraction systems). An aggregate analysis engine controls a composition of *primitive analysis engines* (say, extraction algorithms), which access *common analysis structures* to process and to create *annotations*. Each annotation specifies a span of a text, thereby representing an entity. Besides, it may have features that store values or references to other annotations. Hence, also relations can be realized as annotations. In an application, subtypes of the Apache UIMA type *Annotation* specify the concrete types of information. They are defined in an application-specific type system.

5.2 The Filtering Framework

The implemented extension of the Apache UIMA framework by the filtering framework is illustrated on the right of Figure 4. It consists of four main classes:

Filtering Analysis Engine Extraction algorithms that analyze and filter only relevant portions of their input texts are represented by *filtering analysis engines*, which inherit from primitive analysis engines and, hence, can be composed in an aggregate analysis engine. Prior to analysis, a filtering analysis engine automatically determines the scope its output annotation types and features **O** are relevant for. After analysis, it triggers the generation or the filtering of scopes based on **O** and its created output.[7]

Scoped Query Each *scoped query* to be addressed by an aggregate analysis engine is defined from an application and then automatically parsed to derive the query's dependency graph (cf. Section 3.3) as well as the degrees of filtering of all associated scopes.

Scope We have realized a *scope* as a set of generic annotations in order not to require explicit scope types. In an application, a scope of an input text may have a text unit type assigned, e.g. *Sentence*. According to the derived dependency graph, a scope can have at most one root scope and an arbitrary number of descendant scopes.

Scope TMS To avoid modifications of the Apache UIMA framework, we maintain all scopes using a blackboard architecture [21].[8] In particular, filtering analysis engines determine and filter scopes via a globally accessible truth maintenance system. This *scope TMS* maintains the dependency graph of each scoped query, a mapping from the degrees of filtering to the respective scopes, and a mapping from the entity and relation types in a scoped query to their scopes. Dependencies between the output types of analysis engines are derived from their metadata. Given the output types of an analysis engine, the scope TMS determines scopes according to Pseudocode 1. If a type O is a degree of filtering, the scope TMS generates each associated scope S by adding all annotations of type O to S. Otherwise, it filters all concerned scopes as shown in Pseudocode 2.[9]

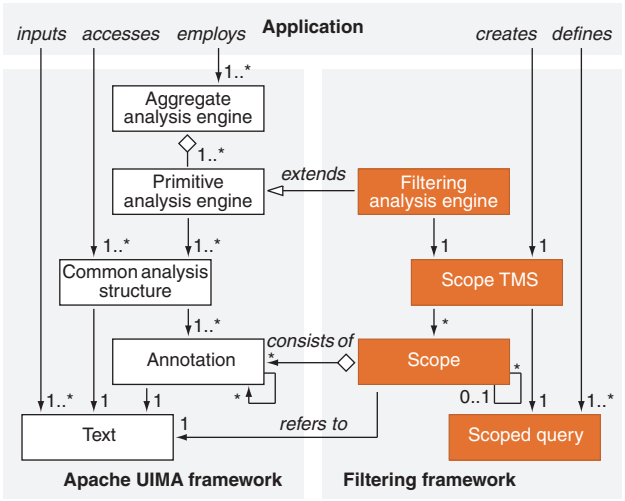

Figure 4: Architecture of the filtering framework extension of Apache UIMA. A filtering analysis engine is a primitive analysis engine, which analyzes only a scope of a text determined by the scope TMS.

Maintaining the scopes of an input text imposes computational costs linear in the number of portions of text of the scopes. In Section 6, we see that these additional costs only marginally affect the efficiency of an application.

6. EVALUATION

We now present experimental results on the efficiency and effectiveness of filtering. A comprehensive evaluation of filtering in information extraction seems infeasible since its potential depends on the amount of information in the given input texts that is relevant for the given extraction problem. We hence provide an appropriate proof-of-concept instead, based on the example queries from Section 3. Our goal is to reveal the impact of the main parameters intrinsic to filtering (i.e., the complexity of a query and the degree of filtering), as well as to demonstrate the efficiency of our approach.

6.1 Experimental Set-up

All experiments were conducted on an *2 GHz Intel Core 2 Duo MacBook* with 4 GB memory. The Java source code of this evaluation is attached to the filtering framework given at `http://www.arguana.com`.

[7] In Apache UIMA, the set **O** can be inferred from the result specification of an analysis engine, which in turn is automatically derived from the analysis engine's descriptor file.

[8] Future versions of Apache UIMA could integrate the scope TMS in the common analysis structure to allow for an optimized integration of extraction and truth maintenance.

[9] Notice that the Scope TMS never removes any annotations from the UIMA indexes, but it only deletes references of the annotations in order to exclude them from further analyses.

Table 1: The number of processed characters in millions with implied filter ratio, the run-time in seconds with standard deviation σ and implied time ratio, and the numbers of true positives (TP), false positives (FP), and positives (P) as well as the resulting precision of pipeline Π_1 for the query q_1 = *Founded(Organization, Time)* under different degrees of filtering on the English CoNLL'03 corpus and on the German Revenue corpus.

Corpus	Degree of filtering	Characters	Filter ratio	Run-time $\pm$ σ	Time ratio	TP	FP	P	Precision
CoNLL'03	No filtering	12.70 M	100.0%	75.4 s $\pm$ 0.3 s	100.0%	7	1	8	87.5%
	Paragraph level	10.35 M	81.5%	52.1 s $\pm$ 0.5 s	69.0%	7	1	8	87.5%
	Sentence level	5.16 M	40.6%	24.8 s $\pm$ 0.2 s	32.9%	5	0	5	100.0%
Revenue	No filtering	30.63 M	100.0%	157.8 s $\pm$ 0.3 s	100.0%	37	15	52	71.2%
	Paragraph level	24.95 M	81.4%	126.5 s $\pm$ 0.5 s	80.2%	27	11	38	71.1%
	Sentence level	14.67 M	47.9%	74.9 s $\pm$ 0.2 s	47.5%	14	5	19	73.7%

Text Corpora In the evaluation, we analyze filtering on two text corpora of different languages, which have been used for information extraction purposes in the last years: First, the complete English corpus of the *CoNLL'03 Shared Task* [37] with 1393 news stories. And second, the complete German *Revenue corpus* that we introduced in [43] and that consists of 1128 online business news articles.

Algorithms Our experiments refer to the example queries from Section 3, for which we employed eleven extraction algorithms that can be parameterized to work for both English and German. All algorithms have more or less comparable run-time that scales linear with the text length.[10]

Concretely, we relied on self-implemented rule-based algorithms for tokenization (TOK), paragraph splitting (PAR), and sentence splitting (SEN), while we used the *TreeTagger* [39] wrapper *tt4j*[11] for part-of-speech tagging (POS) and chunking (CHU). Organization entities (ORG) were extracted with *Stanford NER* [16] using the model from [14] for German texts. We employed the regex-based recognizers for time entities (TIM) and money entities (MON) as well as the SVM-based forecast event detector (FOR) presented in [46]. Finally, we developed lexicon-based relation extractors for *Founded* (FOU) and *Financial* (FIN) that qualify for arbitrary degrees of filtering. These relation extractors look for indicator words of the relation type in question and relate a pair of entities if it is close enough to such a word.

Approaches Each information extraction system that we evaluate is a pipeline Π, which sequentially applies a subset of the eleven extraction algorithms to its input. The concrete pipelines are given below; some perform filtering, some do not. We provide no comparison to existing filtering approaches (cf. Section 2), as these approaches do not compete with our approach, but rather can be integrated with it.

Measures We determined the *filter ratio* of each pipeline Π, which we define as the quotient of the number of characters processed by Π and the number of characters processed by a respective non-filtering pipeline. Similarly, we measured the run-time of each Π in seconds (averaged over ten runs) to compute the *time ratio* as the quotient of the run-time of Π and the run-time of a non-filtering pipeline.

[10] We explicitly avoided to include computationally expensive algorithms (e.g. a dependency parser). While such algorithms significantly increase the efficiency potential of filtering, they would make it difficult to distinguish between the effects of filtering and of the order of algorithm application.

[11] tt4j wrapper, `http://code.google.com/p/tt4j`

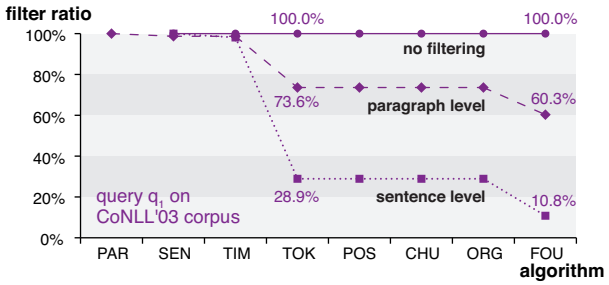

Figure 5: Interpolated curves of the filter ratios of the algorithms in pipeline Π_1 under three degrees of filtering for query q_1 = *Founded(Organization, Time)* on the CoNLL'03 corpus. The more computationally expensive the algorithms later in a pipeline are, the higher the efficiency impact of filtering is.

In terms of effectiveness, we counted the *positives*, i.e., the number of extracted relations, in order to roughly compare the recall of pipelines. An exact evaluation of recall is hardly feasible on the given corpora for lack of appropriate manual event and relation annotations. In Section 6.2, we show the *precision* of extracting foundation relations under different degrees of filtering. To this end, we decided for each found positive manually whether it is true or false. In particular, a relation was considered a true positive if and only if its anchor was brought into relation with the correct time entity while spanning the correct organization entity.[12]

6.2 Impact of the Degree of Filtering

In order to analyze the effects of filtering on the efficiency and the effectiveness of extraction, we considered the query q_1 = *Founded(Organization, Time)* from Section 3.1 under different degrees of filtering on both corpora. In particular, we separately assigned the degrees *Paragraph* and *Sentence* to q_1. Then, we ran the pipeline

$$\Pi_1 = (\text{PAR}, \text{SEN}, \text{TIM}, \text{TOK}, \text{POS}, \text{CHU}, \text{ORG}, \text{FOU})$$

to compare filtering for the according scoped queries to the application of Π_1 without filtering.

Figure 5 visualizes the filter ratios of all algorithms in Π_1 on the CoNLL'03 corpus depending on the degree of filtering. The first algorithm that discards significant portions

[12] The evaluation of precision is only fairly representative, as in practice many extractors do not take into account cross-sentence or even cross-paragraph relations at all. In such cases, precision remains unaffected by the degree of filtering.

Table 2: The filter ratio, the time ratio, and the number of positives (in terms of extracted relations) of pipeline Π_2 for q_2 = *Forecast(Anchor, Time)* under different degrees of filtering on the Revenue corpus.

Degree of filtering	Filter ratio	Time ratio	Positives
No filtering	100.0%	100.0%	3 622
Paragraph level	87.2%	83.1%	3 622
Sentence level	64.8%	53.9%	3 622

of irrelevant text is TIM, which filters 73.6% of its input on the paragraph level and 28.9% on the sentence level. These percentages further decrease after ORG to 60.3% and 10.8%, respectively. While the efficiency and effectiveness impact of filtering depends on the employed algorithms, the overall values of Π_1 on both corpora are listed in Table 1.

In case of the CoNLL'03 corpus, 81.5% of the 12.70 million characters that are processed without filtering are analyzed when performing filtering on the paragraph level. Thereby, about a third of the run-time of 75.4 seconds is saved. For both these degrees of filtering, the same eight relations were extracted with a precision of 87.5%. This implies that no relation was found, which exceeds paragraph boundaries. Filtering on the sentence level lowered the filter ratio to 40.6% and the time ratio to 32.9%, but also reduced the number of positives to 5. The fact that all found in-sentence relations are true positives might be coincidence, but it also indicates a tendency to achieve better precision, when the size of the filtered portions of texts remains small.

On the Revenue corpus, the filter and time ratios are higher than on the CoNLL'03 corpus due to a larger amount of time entities (which are extracted first in Π_1). Still, under the degree *Sentence*, Π_1 needs only 47.5% of the time of a non-filtering pipeline. Hence, the benefit of filtering is obvious even for simple binary relation types like *Founded* and even though we did not employ expensive algorithms like a dependency parser. Moreover, the numbers of positives in Table 1 (52 in total, 38 within paragraphs, 19 within sentences) suggest that degrees of filtering provide an intuitive means to adjust the trade-off between a pipeline's efficiency and its recall, whereas precision remains rather stable.

6.3 Optimization of Run-Time Efficiency

As discussed in Section 3.1, filtering can also be exploited to optimize the efficiency of a pipeline without compromising effectiveness. To demonstrate this, we assigned the same degrees of filtering as above to q_2 = *Forecast(Anchor, Time)* while knowing that the forecast event detector FOR operates only on the sentence level. For all three degrees, we then addressed q_2 on the Revenue corpus with the pipeline Π_2:

$$\Pi_2 = (\text{PAR, SEN, TIM, TOK, POS, FOR})$$

Table 2 underlines the implied efficiency optimization potential of filtering: Irrespective of the degree of filtering, the same 3622 sentences were recognized as forecast relations. At the same time, filtering reduces the fraction of analyzed characters to less than two third (64.8%), though more than every tenth sentence is classified relevant (3 622 of 33 364 sentences in the Revenue corpus). Such filtering forms the basis of our and other pipeline scheduling approaches [40, 45, 46]. Here, it improves the run-time of Π_2 by almost factor 2, as expressed by a time ratio of 53.9%.

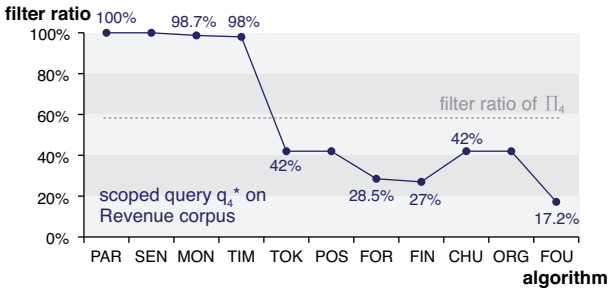

Figure 6: Interpolated curve of the filter ratios of the eleven algorithms in pipeline Π_4 for the disjunctive scoped query $q_4^* = q_1^* \vee q_3^*$ on the Revenue corpus.

6.4 Impact of the Complexity of the Query

In a last experiment, we analyzed the benefit and computational effort of filtering under increasing complexity of the addressed query on the Revenue corpus. For this purpose, we considered the following scoped versions of the queries q_1, q_3, and q_4 from Section 3:

$$q_1^* = \textit{Sentence[Founded(Organization, Time)]}$$

$$q_3^* = \textit{Paragraph[Financial(Money, Sentence[}q_2\textit{])]}$$

$$q_4^* = q_1^* \vee q_3^*$$

We applied pipeline Π_1 for q_1^* again and we used the following pipelines for q_3^* and q_4^*, respectively:

$$\Pi_3 = (\text{PAR, SEN, MON, TIM, TOK, POS, FOR, FIN})$$

$$\Pi_4 = (\text{PAR, SEN, MON, TIM, TOK, POS, FOR, FIN, CHU, ORG, FOU})$$

Table 3 lists the results. While the time ratios slightly increase from q_1^* to q_4^*, they reveal that all three pipelines took less than half of the run-time of their respective non-filtering pipelines. Not surprisingly, the longest pipeline Π_4 processed the largest number of characters (24.40 millions). However, the filter ratio of Π_4 (57.9%) seems more like a "weighted average" of the filter ratios of Π_1 and Π_3.

The reason behind can be inferred from Figure 6, which illustrates the filter ratios of all algorithms in Π_4. The values of the interpolated curve do not decline monotonously along the pipeline, but they depend on what portions of the input texts are relevant for which conjunction in the disjunctive scoped query q_4^*. E.g., the algorithm FOR analyzed only text units relevant for q_3^*, i.e., sentences that contain a time entity in paragraphs that contain a money entity. In contrast, CHU processed the 42% of the characters that belong to sentences with time entities. The same holds for POS, although the output of POS was needed for both q_3^* and q_4^*.

Table 3 also shows the efficiency of our approach by comparing the analysis times of each of the three pipelines (i.e., the time taken by their employed extraction algorithms) to the time required by the filtering framework. Only 1.0% (0.7 of 74.9 seconds) was spent for the generation, determination, and filtering of the scopes of q_1^*. This percentage grows only marginally under increasing query complexity, as the values for q_3^* (1.1%) and q_4^* (1.2%) suggest. We hence conclude that the filtering view of information extraction can be operationalized efficiently, though our implementation certainly leaves much room for optimization.

Table 3: The number of processed characters with implied filter ratio as well as the run-time with standard deviation and implied time ratio of pipelines under increasing query complexity on the Revenue corpus. Each run-time is broken down into the time spent for analysis and the time required by our filtering framework.

Query	Pipeline	Characters	Filter ratio	Run-time $\pm \sigma$	Time ratio	Analysis time	Framework time
q_1^*	Π_1	14.67 M	47.9%	74.9 s $\pm$ 0.2 s	47.5%	74.2 s (99.0%)	0.7 s (1.0%)
q_3^*	Π_3	17.86 M	58.3%	34.9 s $\pm$ 0.1 s	48.6%	34.5 s (98.9%)	0.4 s (1.1%)
q_4^*	Π_4	24.40 M	57.9%	91.2 s $\pm$ 0.5 s	48.8%	90.2 s (98.8%)	1.1 s (1.2%)

7. CONCLUSION

The need for run-time efficiency in information extraction is pressing in times of big data where extraction problems are tackled at large scale. To improve extraction efficiency, we propose to view and to consistently address information extraction as the task to filter the relevant portions of input texts. For this purpose, we introduce an input control that maintains the dependencies between all relevant types of entities and relations in order to analyze and filter only relevant portions of text in each step of an extraction process. Thereby, the efficiency of an extraction process can be optimized without losing effectiveness, and we can easily trade efficiency for effectiveness, in particular for recall. Still, other approaches to improve efficiency remain applicable.

We have implemented and evaluated the filtering view in an easy-to-use and open-source software framework on top of the Apache UIMA framework. While the exact efficiency potential of filtering naturally depends on the amount of information in the given input texts that is relevant for the extraction problem at hand, the results emphasize that our proposed approach significantly improves extraction efficiency without requiring notable additional time.

8. ACKNOWLEDGMENTS

This work has been partly funded by the research project *ArguAna* of the German Federal Ministry of Education and Research (BMBF) under contract number 01IS11016A.

9. REFERENCES

[1] E. Agichtein. Scaling Information Extraction to Large Document Collections. *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, 28:3–10, 2005.

[2] E. Agichtein and L. Gravano. Querying Text Databases for Efficient Information Extraction. In *Proceedings of the 19th International Conference on Data Engineering*, pages 113–124, 2003.

[3] R. Al-Rfou' and S. Skiena. SpeedRead: A Fast Named Entity Recognition Pipeline. In *Proceedings of the 24th International Conference on Computational Linguistics*, pages 51–66, 2012.

[4] M. J. Cafarella, D. Downey, S. Soderland, and O. Etzioni. KnowItNow: Fast, Scalable Information Extraction from the Web. In *Proceedings of the Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 563–570, 2005.

[5] C. Cardie, V. Ng, D. Pierce, and C. Buckley. Examining the role of statistical and linguistic knowledge sources in a general-knowledge question-answering system. In *Proceedings of the Sixth Applied Natural Language Processing Conference*, pages 180–187, 2000.

[6] N. Chinchor, D. D. Lewis, and L. Hirschman. Evaluating Message Understanding Systems: An Analysis of the Third Message Understanding Conference (MUC-3). *Computational Linguistics*, 19(3):409–449, 1993.

[7] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. R. Reiss, and S. Vaithyanathan. SystemT: An Algebraic Approach to Declarative Information Extraction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 128–137, 2010.

[8] J. Cowie and W. Lehnert. Information Extraction. *Communications of the ACM*, 39(1):80–91, 1996.

[9] H. Cui, R. Sun, K. Li, M.-Y. Kan, and T.-S. Chua. Question Answering Passage Retrieval using Dependency Relations. In *Proceedings of the 28th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 400–407, 2005.

[10] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, N. Aswani, I. Roberts, G. Gorrell, A. Funk, A. Roberts, D. Damljanovic, T. Heitz, M. A. Greenwood, H. Saggion, J. Petrak, Y. Li, and W. Peters. *Text Processing with GATE (Version 6)*. University of Sheffield, 2011.

[11] A. Das Sarma, A. Jain, and P. Bohannon. Building a Generic Debugger for Information Extraction Pipelines. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 2229–2232, 2011.

[12] A. Doan, J. F. Naughton, R. Ramakrishnan, A. Baid, X. Chai, F. Chen, T. Chen, E. Chu, P. DeRose, B. Gao, C. Gokhale, J. Huang, W. Shen, and B.-Q. Vuong. Information Extraction Challenges in Managing Unstructured Data. *SIGMOD Records*, 37(4):14–20, 2009.

[13] A. Doan, R. Ramakrishnan, and S. Vaithyanathan. Managing Information Extraction: State of the Art and Research Directions. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, pages 799–800, 2006.

[14] M. Faruqui and S. Padó. Training and Evaluating a German Named Entity Recognizer with Semantic Generalization. In *Proceedings of KONVENS 2010*, pages 129–133, 2010.

[15] D. Ferrucci and A. Lally. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, 10(3–4):327–348, 2004.

[16] J. R. Finkel, T. Grenager, and C. D. Manning. Incorporating Non-local Information into Information Extraction Systems by Gibbs Sampling. In *Proceedings of the 43nd Annual Meeting of the Association for Computational Linguistics*, pages 363–370, 2005.

[17] J. R. Finkel, C. D. Manning, and A. Y. Ng. Solving the Problem of Cascading Errors: Approximate Bayesian Inference for Linguistic Annotation Pipelines. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 618–626, 2006.

[18] G. Forman and E. Kirshenbaum. Extremely Fast Text Feature Extraction for Classification and Indexing. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 1221–1230, 2008.

[19] R. Grishman. Information Extraction: Techniques and Challenges. In *International Summer School on Information Extraction: A Multidisciplinary Approach to an Emerging Information Technology*, pages 10–27, 1997.

[20] D. Gruhl, L. Chavet, D. Gibson, J. Meyer, P. Pattanayak, A. Tomkins, and J. Y. Zien. How to Build a WebFountain: An Architecture for Very Large-scale Text Analytics. *IBM Systems Journal*, 43(1):64–77, 2004.

[21] B. Hayes-Roth. A Blackboard Architecture for Control. *Artificial Intelligence*, 26(3):251–321, 1985.

[22] K. Hollingshead and B. Roark. Pipeline Iteration. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, pages 952–959, 2007.

[23] L. Jean-Louis, R. Besançon, and O. Ferret. Text Segmentation and Graph-based Method for Template Filling in Information Extraction. In *Proceedings of the 5th International Joint Conference on Natural Language Processing*, pages 723–731, 2011.

[24] D. Jurafsky and J. H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. Prentice-Hall, 2nd edition, 2009.

[25] J.-D. Kim, S. Pyysalo, T. Ohta, R. Bossy, N. Nguyen, and J. Tsujii. Overview of BioNLP Shared Task 2011. In *Proceedings of the BioNLP Shared Task 2011 Workshop*, pages 1–6, 2011.

[26] E. Krikon, D. Carmel, and O. Kurland. Predicting the Performance of Passage Retrieval for Question Answering. In *Proceedings of the 21st ACM International Conference on Information and Knowledge management*, pages 2451–2454, 2012.

[27] G. Limaye, S. Sarawagi, and S. Chakrabarti. Annotating and Searching Web Tables using Entities, Types and Relationships. *Proceedings of the VLDB Endowment*, 3(1):1338–1347, 2010.

[28] W. Lu and D. Roth. Automatic Event Extraction with Structured Preference Modeling. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics*, pages 835–844, 2012.

[29] Mausam, M. Schmitz, R. Bart, S. Soderland, and O. Etzioni. Open Language Learning for Information Extraction. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 523–534, 2012.

[30] C. Nedellec, M. O. A. Vetah, and P. Bessières. Sentence Filtering for Information Extraction in Genomics, a Classification Problem. In *Proceedings of the 5th European Conference on Principles of Data Mining and Knowledge Discovery*, pages 326–337, 2001.

[31] OMG. *Unified Modeling Language (OMG UML) Superstructure, Version 2.4.1*. 2011.

[32] P. Pantel, D. Ravichandran, and E. Hovy. Towards Terascale Knowledge Acquisition. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 771–777, 2004.

[33] M. Pasca. Web-based Open-Domain Information Extraction. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, pages 2605–2606, 2011.

[34] S. Patwardhan and E. Riloff. Effective Information Extraction with Semantic Affinity Patterns and Relevant Regions. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 717–727, 2007.

[35] H. Poon and P. Domingos. Joint Inference in Information Extraction. In *Proceedings of the 22nd National Conference on Artificial Intelligence*, pages 913–918, 2007.

[36] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, 3rd edition, 2009.

[37] E. F. T. K. Sang and F. D. Meulder. Introduction to the CoNLL-2003 Shared Task: Language-independent Named Entity Recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147, 2003.

[38] S. Sarawagi. Information Extraction. *Foundations and Trends in Databases*, 1(3):261–377., 2008.

[39] H. Schmid. Improvements in Part-of-Speech Tagging with an Application to German. In *Proceedings of the ACL SIGDAT-Workshop*, pages 47–50, 1995.

[40] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative Information Extraction using Datalog with Embedded Extraction Predicates. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, pages 1033–1044, 2007.

[41] B. Stein, S. M. zu Eissen, G. Gräfe, and F. Wissbrock. Automating Market Forecast Summarization from Internet Data. In *Proceedings of the Fourth Conference on WWW/Internet*, pages 395–402, 2005.

[42] M. Stevenson. Fact Distribution in Information Extraction. *Language Resources and Evaluation*, 40(2):183–201, 2007.

[43] H. Wachsmuth, P. Prettenhofer, and B. Stein. Efficient Statement Identification for Automatic Market Forecasting. In *Proceedings of the 23rd International Conference on Computational Linguistics*, pages 1128–1136, 2010.

[44] H. Wachsmuth, M. Rose, and G. Engels. Automatic Pipeline Construction for Real-Time Annotation. In *Proceedings of the 14th International Conference on Intelligent Text Processing and Computational Linguistics*, pages 38–49, 2013.

[45] H. Wachsmuth and B. Stein. Optimal Scheduling of Information Extraction Algorithms. In *Proceedings of the 24th International Conference on Computational Linguistics: Posters*, pages 1281–1290, 2012.

[46] H. Wachsmuth, B. Stein, and G. Engels. Constructing Efficient Information Extraction Pipelines. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management*, pages 2237–2240, 2011.

[47] W. Wang, R. Besançon, O. Ferret, and B. Grau. Filtering and Clustering Relations for Unsupervised Information Extraction in Open Domain. In *Proceedings of the 20th ACM Conference on Information and Knowledge Management*, pages 1405–1414, 2011.

[48] C. Whitelaw, A. Kehlenbeck, N. Petrovic, and L. Ungar. Web-scale Named Entity Recognition. In *Proceedings of the 17th ACM Conference on Information*.