Information Extraction as a Filtering Task

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

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

Gregor Engels  
University of Paderborn, s-lab  
Paderborn, Germany  
egels@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 between 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

General Terms  
Algorithms, Performance, Design

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

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

ACM ‘13, October 27–November 1, 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 [23]. 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 instance, 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 [35] 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 UIMA² (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.

¹The 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.

²Apache 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 [28] and events [27], information extraction is now on the verge of being exploited at web-scale [26] and gradually finds its way into search engines [32]. 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 [19], specialized search indexes [4], and relation-oriented search queries [2] over the use of cheap algorithms for preprocessing [3] and extraction [31] 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 to be addressed in real-time, such as question answering, which usually includes so called passage retrieval to filter relevant portions of input texts [25]. 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, [33] observed that classifying the relevant portions of text before extraction can also improve effectiveness. [22] exploits this for template filling, and [46] filters trustworthy candidate relations in unsupervised relation extraction. Filtering approaches for efficiency purposes often focus on complete texts, e.g. using fast text categorization [40]. In [37], the author complaints that techniques for efficiently finding the relevant portions of texts are still restricted to hand-coded heuristics. [29] 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 [41], whereas we provide a means to specify the sizes of filtered portions, thereby trading efficiency for effectiveness.

In [45], 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 [44]. 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 [39]. 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 pipe-
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:

\[
q ::= q \lor q \mid r
\]

\[
r ::= R(\{, r\}^*) \mid E
\]

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:

\[
q_1 = \text{Founded} (\text{Organization}, \text{Time})
\]

Before extracting relations, only those portions of text that are relevant 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.

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:

\[
q_2 = \text{Forecast} (\text{Anchor}, \text{Time})
\]

\[
q_3 = \text{Financial} (\text{Money}, q_2)
\]

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:

\[
q_4 = q_1 \lor q_3
\]

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 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 tradeoff 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. 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.

### 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:

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

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. 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 \( q^* \) 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.

### 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 of an input text must be assumed relevant. If it 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.

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 \( \psi^{(u)} \) 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:

\[
\psi^{(u)} : R^{(u)} \land r_1^{(u)} \land \ldots \land r_k^{(u)} \rightarrow q^*(u)
\]

For all child nodes \( r_i^{(u)} \) of a root node \( U \) of the form \( U_i[R(r_1, \ldots, r_i)] \), we additionally model the relevance of a portion of text \( u' \) of the scope associated to \( U_i \) as:

\[
\psi^{(u')} : R_i^{(u')} \land r_1^{(u')} \land \ldots \land r_i^{(u')} \rightarrow r_i^{(u')}
\]

This modeling step is repeated recursively until each child node \( r_i^{(u')} \) 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 \( \psi^{(u)} \) and \( \psi^{(u')} \) of that text. In case of the example in Figure 3, four assumptions are initially believed for the paragraph \( p_2 \):

\[
\psi^{(p_2)} : \text{Financial}^{(p_2)} \land \text{Money}^{(p_2)} \land q_4^* \rightarrow q_4^{(p_2)}
\]

\[
\psi^{(s_2)} : \text{Forecast}^{(s_2)} \land \text{Anchor}^{(s_2)} \land \text{Time}^{(s_2)} \rightarrow q_2^{(s_2)}
\]

\[
\psi^{(s_3)} : \text{Forecast}^{(s_3)} \land \text{Anchor}^{(s_3)} \land \text{Time}^{(s_3)} \rightarrow q_3^{(s_3)}
\]

\[
\psi^{(s_4)} : \text{Forecast}^{(s_4)} \land \text{Anchor}^{(s_4)} \land \text{Time}^{(s_4)} \rightarrow q_2^{(s_4)}
\]
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 of an input text based on the set of output types \( O \). Thus, the output of an extraction algorithm is not only believed over the whole text, but also their dependent scopes. The set of dependent scopes \( S \) of a portion of text \( u \) of the root’s descendant nodes. This, of course, includes all ancestor scopes of \( S \).

### 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 \( \phi^{(p2)} \), \( \psi^{(s2)} \), and \( \psi^{(s4)} \) imply that an algorithm with output type \( \text{Forecast} \) needs to analyze the sentences \( s2 \) and \( s4 \). In general, an algorithm with a set of output types \( O \) must be applied to the union \( S_o \) 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 \( O \in O \) 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 \( O \in O \) as input.

The determination of \( S_o \) 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

<table>
<thead>
<tr>
<th>Pseudocode 1 DETERMINE SCOPE(Output types ( O ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: for each Output type ( O ) in ( O ) do</td>
</tr>
<tr>
<td>2: if ( O ) is a degree of filtering in the scoped query ( q^* ) then</td>
</tr>
<tr>
<td>3: return the whole input text</td>
</tr>
<tr>
<td>4: Scopes ( S )</td>
</tr>
<tr>
<td>5: for each Output type ( O ) in ( O ) do</td>
</tr>
<tr>
<td>6: Scopes ( S_o ) ← all scopes ( O ) is relevant for according to ( q^* )</td>
</tr>
<tr>
<td>7: ( S ).addAll( ( S_o ))</td>
</tr>
<tr>
<td>8: Scope ( S_o )</td>
</tr>
<tr>
<td>9: for each Scope ( S ) in ( S ) do</td>
</tr>
<tr>
<td>10: for each Portion of text ( u ) in ( S ) do</td>
</tr>
<tr>
<td>11: if not ( u ) intersects with ( S_o ) then ( S_o ).add(( u ))</td>
</tr>
<tr>
<td>12: else ( S_o ).merge(( u ))</td>
</tr>
<tr>
<td>13: return ( S_o )</td>
</tr>
</tbody>
</table>

The determined \( S_o \) is then obtained by taking all of the scopes’ non-overlapping portions of text and by merging overlapping portions (lines 8–12).

### 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 \( O \in O \) 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, \( \psi^{(p2)} \) and \( \psi^{(s4)} \) turn out to be true. Hence, \( q^*_{(p2)} \) is true and \( \phi^{(p2)} \) remains in the following form:

\[
\phi^{(p2)} : \text{Financial}^{p2} \land \text{Money}^{p2} \land q^*_{(p2)} \rightarrow q^*_{(p2)}
\]

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 \( S \in S \), a portion of text \( u \) is maintained only if it contains an instance of one of the output types \( O \subseteq O \) that are relevant for \( S \) (lines 1–5). Accordingly, only those portions of text in the set of descendant scopes \( S' \) of \( S \) that do not intersect with any portion of text in \( S \).

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 \( S \).

<table>
<thead>
<tr>
<th>Pseudocode 2 FILTER(Scopes ( S ), Output types ( O ))</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: for each Scope ( S ) in ( S ) do</td>
</tr>
<tr>
<td>2: Output types ( O' ) ← all types in ( O ) that are relevant for ( S )</td>
</tr>
<tr>
<td>3: for each Portion of text ( u ) in ( S ) do</td>
</tr>
<tr>
<td>4: if not ( u ) contains an instance of any ( O \in O' ) then</td>
</tr>
<tr>
<td>5: ( S ).remove(( u ))</td>
</tr>
<tr>
<td>6: Scope ( S_o ) ← ( S ).getRootScope()</td>
</tr>
<tr>
<td>7: if ( S_o \neq S ) then</td>
</tr>
<tr>
<td>8: for each Portion of text ( u ) in ( S_o ) do</td>
</tr>
<tr>
<td>9: if not ( u ) intersects with ( S ) then ( S_o ).remove(( u ))</td>
</tr>
<tr>
<td>10: Scopes ( S' ) ← ( S_o ).getDescendantScopes()</td>
</tr>
<tr>
<td>11: for each Scope ( S' \neq S ) in ( S' ) do</td>
</tr>
<tr>
<td>12: for each Portion of text ( u ) in ( S' ) do</td>
</tr>
<tr>
<td>13: if not ( u ) intersects with ( S ) then ( S' ).remove(( u ))</td>
</tr>
</tbody>
</table>

### 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 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](http://www.arguana.com), together with usage instructions and some sample applications.

---

5For 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.

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

---
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 [30] 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 [20]. 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

\(^7\)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.

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.

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.\(^9\)

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.

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 [36] with 1393 news stories. And second, the complete German Revenue corpus that we introduced in [42] and that consists of 1128 online business news articles.

\(^9\)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 $\sigma$ 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 $\Pi$ 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.

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

**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 [38] 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 [45]. 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 $\Pi$, 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 $\Pi$, which we define as the quotient of the number of characters processed by $\Pi$ and the number of characters processed by a respective non-filtering pipeline. Similarly, we measured the run-time of each $\Pi$ in seconds (averaged over ten runs) to compute the time ratio as the quotient of the run-time of $\Pi$ and the run-time of a non-filtering pipeline.

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

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

to compare filtering for the according scoped queries to the application of $\Pi_1$ without filtering.

Figure 5 visualizes the filter ratios of all algorithms in $\Pi_1$ on the CoNLL’03 corpus depending on the degree of filtering. The first algorithm that discards significant portions 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 $\Pi_1$ 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

\(^{10}\)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

\(^{12}\)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.
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 3622 sentences to each of the three pipelines for the disjunctive scoped query \( q_3^* \). The values in 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 (3622 of 33,364 sentences in the Revenue corpus). This filtering forms the basis of our other pipeline scheduling approaches [39, 44, 45]. Here, it improves the run-time of \( \Pi_2 \) by almost factor 2, as expressed by a time ratio of 53.9%.

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.

<table>
<thead>
<tr>
<th>Query</th>
<th>Pipeline</th>
<th>Characters</th>
<th>Filter ratio</th>
<th>Run-time ± σ</th>
<th>Time ratio</th>
<th>Analysis time</th>
<th>Framework time</th>
</tr>
</thead>
<tbody>
<tr>
<td>( q_1^* )</td>
<td>( \Pi_1 )</td>
<td>14.67 M</td>
<td>47.9%</td>
<td>74.9 s ± 0.2 s</td>
<td>47.5%</td>
<td>74.2 s (99.0%)</td>
<td>0.7 s (1.0%)</td>
</tr>
<tr>
<td>( q_2^* )</td>
<td>( \Pi_3 )</td>
<td>17.86 M</td>
<td>58.3%</td>
<td>34.9 s ± 0.1 s</td>
<td>48.6%</td>
<td>34.5 s (98.9%)</td>
<td>0.4 s (1.1%)</td>
</tr>
<tr>
<td>( q_3^* )</td>
<td>( \Pi_4 )</td>
<td>24.40 M</td>
<td>57.9%</td>
<td>91.2 s ± 0.5 s</td>
<td>48.8%</td>
<td>90.2 s (98.8%)</td>
<td>1.1 s (1.2%)</td>
</tr>
</tbody>
</table>

Table 2: The filter ratio, the time ratio, and the number of positives (in terms of extracted relations) of pipeline \( \Pi_2 \) for \( q_2 = \text{Forecast} (\text{Anchor}, \text{Time}) \) under different degrees of filtering on the Revenue corpus.

<table>
<thead>
<tr>
<th>Degree of filtering</th>
<th>Filter ratio</th>
<th>Time ratio</th>
<th>Positives</th>
</tr>
</thead>
<tbody>
<tr>
<td>No filtering</td>
<td>100.0%</td>
<td>100.0%</td>
<td>3622</td>
</tr>
<tr>
<td>Paragraph level</td>
<td>87.2%</td>
<td>83.1%</td>
<td>3622</td>
</tr>
<tr>
<td>Sentence level</td>
<td>64.8%</td>
<td>53.9%</td>
<td>3622</td>
</tr>
</tbody>
</table>

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 \( \Pi_1 \)). Still, under the degree \( \text{Sentence} \), \( \Pi_1 \) 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 \( \text{Founded} \) 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.

6.4 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:

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

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

\[
q_4^* = q_1^* \lor q_2^*
\]

We applied pipeline \( \Pi_3 \) for \( q_3^* \) again and we used the following pipelines for \( q_3^* \) and \( q_4^* \), respectively:

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

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

Table 3 also shows the efficiency of our approach by comparing the analysis times of each of the three pipelines (i.e.,

![Figure 6: Interpolated curve of the filter ratios of the eleven algorithms in pipeline \( \Pi_4 \) for the disjunctive scoped query \( q_3^* = q_1^* \lor q_2^* \) on the Revenue corpus.](image)
the time taken by their employed extraction algorithms) to
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∗. This percentage grows only
marginally under increasing query complexity, as the values
for q∗ (1.1%) and q* (1.2%) suggest. We hence conclude
that the filtering view of information extraction can be oper-
ationized efficiently, though our implementation certainly
leaves much room for optimization.

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
texts. For this purpose, we introduce an input control that
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 opti-
mized 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 ef-
ciency 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 ef-
fi ciency without requiring notable additional time.

8. ACKNOWLEDGMENTS

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

9. REFERENCES

[1] E. Agichtein. Scaling Information Extraction to Large
Society Technical Committee on Data Engineering,
Databases for Efficient Information Extraction. In
Proc. of the 19th International Conference on Data
Entity Recognition Pipeline. In Proc. of the 24th
International Conference on Computational
[4] M. J. Cafarella, D. Downey, S. Soderland, and
O. Etzioni. KnowItNow: Fast, Scalable Information
Extraction from the Web. In Proc. of the Conference
on Human Language Technology and Empirical
Methods in Natural Language Processing, pages
563–570, 2005.
Examining the Role of Statistical and Linguistic
Knowledge Sources in a General-Knowledge
Question-Answering System. In Proc. of the Sixth
Applied Natural Language Processing Conference,
Evaluating Message Understanding Systems: An
Analysis of the Third Message Understanding
Conference (MUC-3). Computational Linguistics,
[7] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan,
F. R. Reiss, and S. Vaithyanathan. SystemT: An
Algebraic Approach to Declarative Information
Extraction. In Proc. of the 18th Annual Meeting of the
Association for Computational Linguistics, pages
Question Answering Passage Retrieval using
Dependency Relations. In Proc. of the 28th Annual
International ACM SIGIR Conference on Research
and Development in Information Retrieval, pages
[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.
Generic Debugger for Information Extraction
Conference on Information and Knowledge
[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,
Managing Information Extraction: State of the Art
and Research Directions. In Proc. of the 2006 ACM
SIGMOD International Conference on Management of
German Named Entity Recognizer with Semantic
Approach to Unstructured Information Processing in
the Corporate Research Environment. Natural
Incorporating Non-local Information into Information
Extraction Systems by Gibbs Sampling. In Proc. of
the 43nd Annual Meeting of the Association for
the Problem of Cascading Errors: Approximate
Bayesian Inference for Linguistic Annotation
Pipelines. In Proc. of the 2006 Conference on
Empirical Methods in Natural Language Processing,


