Constructing Efficient Information Extraction Pipelines

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

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

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

ABSTRACT

Information Extraction (IE) pipelines analyze text through several stages. The pipeline's algorithms determine both its effectiveness and its run-time efficiency. In real-world tasks, however, IE pipelines often fail acceptable run-times because they analyze too much task-irrelevant text. This raises two interesting questions: 1) How much "efficiency potential" depends on the scheduling of a pipeline's algorithms? 2) Is it possible to devise a reliable method to construct efficient IE pipelines? Both questions are addressed in this paper. In particular, we show how to optimize the run-time efficiency of IE pipelines under a given set of algorithms. We evaluate pipelines for three algorithm sets on an industrially relevant task: the extraction of market forecasts from news articles. Using a system-independent measure, we demonstrate that efficiency gains of up to one order of magnitude are possible without compromising a pipeline's original effectiveness.

Categories and Subject Descriptors: I.2.7 [AI]: Natural Language Processing—*Text analysis* H.3.4 [Information Storage and Retrieval]: Systems and Software—*Performance evaluation (efficiency and effectiveness)*
General Terms: Algorithms, Performance
Keywords: information extraction, run-time efficiency

1. INTRODUCTION

Information Extraction (IE) will improve today's possibilities of knowledge and information acquisition. Yet, the number of industrial applications is still limited. Typical IE tasks require several analysis stages ranging from preprocessing over the extraction of entities, attributes, relations, and events to reference resolution and normalization. "Conventional" pipelines for these tasks execute all analyses on the whole input and tend to run into efficiency problems, i.e., they fail to meet some required response time. We use the term *efficiency* here only to address such run-time concerns. Accordingly, we use *effectiveness* to describe the quality of an algorithm, such as its accuracy. Existing approaches to increase efficiency focus on faster algorithms for single analyses within a pipeline. However, more effective algorithm sets normally lead to a decrease in efficiency as illustrated in Figure 1a. In contrast, we aim to increase efficiency without losing effectiveness by optimizing the pipeline itself: instead of changing the pipeline's algorithms we introduce filtering stages and rearrange the pipeline's schedule (Figure 1b).

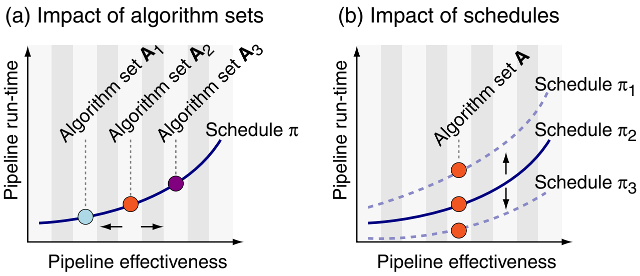

Figure 1: (a) Typical growth curve of the run-time of an IE pipeline; more effective sets of algorithms improve the pipeline's effectiveness but entail higher run-time. (b) A smart scheduling of a pipeline's algorithms can improve its efficiency without impairing its effectiveness.

Within an IE task, different analyses $\mathcal{A}_1, ..., \mathcal{A}_k$ take place, e.g. tokenization or person entity recognition, whereas each analysis $\mathcal{A}_i$ can be operationalized by alternative IE algorithms. We call an ordered subset $\pi = (\mathcal{A}_1, ..., \mathcal{A}_k)$ of analyses an *analysis schedule* and denote the employed set of algorithms as $\mathbf{A}$. The tuple $\Pi = \langle \mathbf{A}, \pi \rangle$ constitutes an *Information Extraction pipeline.* Each algorithm $A \in \mathbf{A}$ needs information of certain input types in order to work properly, and A provides information of certain output types. Consequently, only a subset of the $k!$ possible analysis schedules fulfill the input constraints for all algorithms in $\mathbf{A}$. While some algorithms will fail completely if their requirements are not met, others may degrade significantly in their effectiveness. We hence consider only pipelines whose schedules comply with all input constraints imposed by the employed algorithms. We call such pipelines *admissible.* If admissibility is maintained, the exchange of a pipeline's schedule does not impair the pipeline's effectiveness.

Contributions In this paper, we present a generic method to construct efficient IE pipelines based on four paradigms: 1) a *maximum decomposition* of the given IE task into single analyses, 2) *early filtering* of relevant information, 3) *lazy evaluation* of the applied algorithms, and 4) an *optimized scheduling* of the resulting filtering stages. We have evaluated our method on a subtask of the market forecast extraction task [13]. To make the efficiency of pipelines comparable beyond system bounds, we use a new measure of efficiency that relies on an internal comparison of the algorithms in a pipeline. The results indicate that our method can speed up conventional IE pipelines up to an order of magnitude, while maintaining effectiveness.

CIKM'11, October 24–28, 2011, Glasgow, Scotland, UK.

Related Work The limited industrial usage of IE has recently been discussed in [14]. The authors identify the lack of reusability as one major problem and introduce a component-based IE approach. We also exploit a decomposition into components, but we address another major problem of IE: efficiency. Run-time efficiency has always been a main aspect of algorithm research, but we observe that most rewarded research in IE and in Natural Language Processing (NLP) focuses on effectiveness as do the leading evaluation tracks, namely, MUC, ACE, and the CoNLL Shared Task. While effectiveness is the key to making information processable, we argue that the success of IE systems in the wild is decided by their response times.

Efficiency is by far not an unknown topic in IE and NLP. For instance, greedy decoding has been shown to be competitive with exact inference in entity recognition [10] and machine translation [7]. Other research uses hashing or heuristic search for feature extraction [6] and parsing [4, 8], while Petrov [9] studies efficiency under an iterative refinement of the complexity of NLP models. Like in these examples most research concentrates on *single analyses*. In contrast, we aim to develop efficient *analysis pipelines*. Pipelines that operate on large sources first need to efficiently filter candidate texts, which has been done with fast text categorization [12] or querying techniques trained on text with the relations of interest [1]. The KnowItNow system builds specialized index structures to speed up the IE process [5], and a schedule designed with efficiency in mind is given in [2]. In both cases, the authors focus on retrieval aspects. Instead, we investigate the core IE algorithms for text analysis.

2. PIPELINE OPTIMIZATION

For the purpose of this paper, we first introduce two definitions in order to quantify the efficiency of IE algorithms and pipelines. On this basis, we then present the essential steps to optimize the efficiency of IE pipelines.

Definition 1. Let A be an IE algorithm and D a text collection with n sentences. If A needs time $T_D(A)$ to process all sentences of D on a system Σ, then the *average sentence response time* of A on Σ is $t(A) = T_D(A)/n$.

Definition 1 can be adopted to $t(\Pi)$ for an IE pipeline Π. However, response times are not meaningful by themselves: usually, run-time complexities are defined in a relative manner, by measuring the run-time in unit cost steps that are accepted over system bounds, programming environments, and the like. In this regard we employ $t(A_0)$ of a tokenization algorithm A_0 on the given system Σ as unit cost step. Thereby, we normalize $t(A)$ with respect to IE tasks and cancel out many system-specific side effects.

Definition 2. Let τ be an IE task, Π a pipeline to tackle τ, $\mathcal{E}$ the effectiveness of Π on τ, and A_0 a tokenization algorithm. Then, the *averaged efficiency of* Π *to tackle* τ *at effectiveness* $\mathcal{E}$ is $\mathcal{F}@\mathcal{E} = t(\Pi)/t(A_0)$.

$\mathcal{F}@\mathcal{E}$ allows to estimate $t(\Pi)$ system-independently. In general, $t(\Pi)$ follows from the response times of the algorithms and the number of processed sentences. While $t(A)$ is inherent to an algorithm A, the number of processed sentences depends on a pipeline's schedule. To construct a preferably efficient pipeline, we now introduce four paradigms that yield a tailored scheduling of its algorithms, and that can be seen as operational steps of a method.

Paradigm 1: Maximum Decomposition The deeper a task τ is decomposed into single analyses, the better a pipeline for τ can be optimized. Some analyses have a predefined order, though. Also, some algorithms may be bundled in off-the-shelf tools, which can only be used in a black-box manner. In preprocessing, decomposition means to separate the analyses for sentence splitting, tokenization, the decoding of sequence information (e.g. part-of-speech-tagging), and the creation of syntactic tree structures (e.g. dependency parsing). Similarly, decomposing the recognition of different entity, attribute, and relation types supports optimization. Some of these are *required*, whereas others are *optional*. Finally, IE pipelines extract information on specific events, which relate to *statements* in the text, i.e., text windows of $w > 0$ sentences. Such a statement is only relevant if it comprises an anchor for an event of interest.

Paradigm 2: Early Filtering It is reasonable to focus only on information that probably satisfies a given information need. Thus, filters are inserted after each core IE analyses. In particular, statements without instances of a required entity, attribute, or relation type are discarded as well as statements that are classified not to refer to an event of interest. Also, statements are only kept if the required entities and attributes can be normalized. We call the combination of preprocessing, IE analysis, and filter a *filtering stage*. While filtering can drastically reduce the amount of data to be processed, we thereby miss statements where entities and attributes are spread across the text. This tradeoff can be influenced by adjusting the statement size w.

Paradigm 3: Lazy Evaluation Based on the filtering stages, a simple but effective paradigm can be applied: each analysis is delayed until its results is needed, and it is executed only on possibly relevant text windows. Of course, no analysis is performed more than once. The rationale is that, the more filtering takes place before the execution of an algorithm A, the less data A will process.

Paradigm 4: Optimized Scheduling Finally, the filtering stages are arranged in the most efficient way. Assume that we have implementations of two stages F_1 and F_2 with $t(F_1)$ and $t(F_2)$. If F_1 is applied to n sentences, it filters a number $m(F_1)$ of the n sentences on average, while F_2 filters $m(F_2)$ sentences. In order to minimize the response time of both stages in sequence, F_1 is then applied before F_2 iff

$$t(F_1) + m(F_1)/n \cdot t(F_2) < t(F_2) + m(F_2)/n \cdot t(F_1). \quad (1)$$

While the *filter ratios*, such as $m(F_1)/n$, can be estimated based on a training set, they may change after each applied filter. For simplicity, we approximate the optimized schedule in Section 3 by only pairwise computing the most efficient schedule of two filters using their initial filter ratios.

3. EVALUATION

The Task We evaluated the following subtask τ of the market forecast extraction task [13]: *Extract all related time and money entities that refer to a revenue forecast.* An example for such a relation is: "In 2009, market analysts expected touch screen revenues to reach $[\$9\text{B}]_{\text{money}}$ $[\text{by } 2015]_{\text{time}}$." While τ is quite new and non-standard, the implications of the evaluation can be transferred to other tasks as well.

The Data We used the *Revenue Corpus* introduced in [13] and processed its training set to develop and train algorithms, to measure their response times, and to estimate

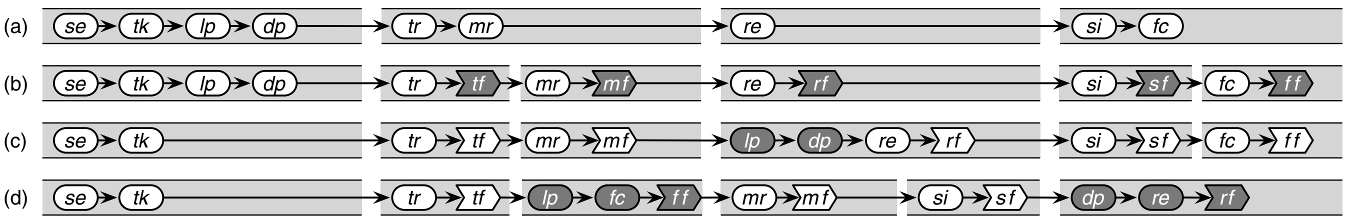

Figure 2: Construction of an optimized schedule following the four paradigms: (a) Maximum decomposition. (b) Early filtering. (c) Lazy evaluation. (d) Optimized scheduling (here, according to algorithm set $\mathbf{A}_2$).

the initial filter ratios of the filtering stages.[1] On the test set, we evaluated the efficiency and the effectiveness of all constructed pipelines. In this set, all 347 of the 6,038 sentences, in which both a time and a money entity relate to revenue, are manually annotated as statements on revenue, among them 104 forecasts. Hence, we set the statement size w (cf. Section 2) of all evaluated algorithms to 1.

Table 1: The evaluated algorithms A for analysis $\mathcal{A}$, the analyses each A depends on, its average sentence response time $t(A)$ in milliseconds, and its effectiveness $\mathcal{E}(A)$ if applied in isolation (accuracy Acc, labeled attachment score LAS, precision P, recall R).

$\mathcal{A}$	A	approach	depends on	$t(A)$	$\mathcal{E}(A)$
se:	se_1	rule-based	–	0.46	Acc 0.95
tk:	tk_1	rule-based	*se*	0.60	Acc 0.98
lp:	lp_1	*TreeTagger*	se, tk	0.34	Acc 0.97 [11]
	lp_2	*mate-tools*	se, tk	21.87	Acc 0.98 [3]
dp:	dp_1	*mate-tools*	se, tk, lp	54.61	LAS 0.87 [4]
	dp_2	*mate-tools*	se, tk, lp	166.14	LAS 0.88 [4]
tr:	tr_1	regex	se, tk	0.37	P 0.91, R 0.97
mr:	mr_1	regex	se, tk	0.70	P 0.99, R 0.95
re:	re_1	rule-based	se, tk, tr, mr	0.02	P 0.69, R 0.88
	re_2	SVM	se, tk, lp, dp, tr, mr	1.39	P 0.75, R 0.88
si:	si_1	lexicon	se, tk	0.02	P 0.86, R 0.93
	si_2	SVM	se, tk, tr, mr	1.43	P 0.87, R 0.93
fc:	fc_1	SVM	se, tk, lp, tr	0.27	Acc 0.93

Table 2: Precision (P), recall (R), and F_1-score (F_1) of all admissible pipelines $\Pi_{i,j} = \langle \mathbf{A}_i, \pi_j \rangle$.

π	P	R	F_1	P	R	F_1	P	R	F_1
	algorithm set $\mathbf{A}_1$			*algorithm set* $\mathbf{A}_2$			*algorithm set* $\mathbf{A}_3$		
π_a	0.65	0.56	0.60	0.72	0.58	0.65	0.75	0.61	0.67
π_b	0.65	0.56	0.60	0.72	0.58	0.65	0.75	0.61	0.67
π_c	0.65	0.56	0.60	0.72	0.58	0.65	0.75	0.61	0.67
π_1^*	0.65	0.56	0.60	–	–	–	–	–	–
π_2^*	0.65	0.56	0.60	0.71	0.58	0.64	0.75	0.61	0.67
π_3^*	0.65	0.56	0.60	0.71	0.58	0.64	0.75	0.61	0.67

Analyses and Algorithms Following paradigm 1, we decomposed τ into the following nine analyses (cf. Figure 2a):

- *se* Split the input text into sentences.
- *tk* Tokenize the sentences.
- *lp* Lemmatize the tokens and tag their part-of-speech.
- *dp* Parse dependencies of the tokens in the sentences.
- *tr* Recognize all time entities in the sentences.
- *mr* Recognize all money entities in the sentences.
- *re* Extract relations between time and money entities.
- *si* Identify statements on revenue.
- *fc* Classify whether a sentence is a forecast or not.

As we only needed lemmas and tags together, we did not decompose *lp*. Table 1 lists the algorithms that we evaluated for the nine analyses. We used the *TreeTagger*[2] [11] and the *mate-tools*[3] [3, 4] for lemmatization, tagging, and parsing. Where no reference is given, we measured effectiveness on the test set. Since we had more than one algorithm for *lp*, *dp*, *re*, and *si*, we investigated three algorithm sets:

$$\begin{aligned}
\mathbf{A}_1 &= \{\text{se}_1, \text{tk}_1, \mathbf{lp_1}, - \ \ , \text{tr}_1, \text{mr}_1, \mathbf{re_1}, \mathbf{si_1}, \text{fc}_1\} \\
\mathbf{A}_2 &= \{\text{se}_1, \text{tk}_1, \mathbf{lp_1}, \mathbf{dp_1}, \text{tr}_1, \text{mr}_1, \mathbf{re_2}, \mathbf{si_2}, \text{fc}_1\} \\
\mathbf{A}_3 &= \{\text{se}_1, \text{tk}_1, \mathbf{lp_2}, \mathbf{dp_2}, \text{tr}_1, \text{mr}_1, \mathbf{re_2}, \mathbf{si_2}, \text{fc}_1\}
\end{aligned}$$

Differences are marked in bold. Because of a very simple relation extractor, $\mathbf{A}_1$ needs no dependency parsing at all.

Filtering Stages For paradigm 2, we inserted a filter after each IE analysis (cf. Figure 2b). The results of *dp* are used for *re*, whereas *dp* itself requires lemmas and tags from *lp* as input. So, for lazy evaluation, we delayed *lp* and *dp* to execute them right before *re* (cf. Figure 2c).

Schedules and Pipelines The last step is to derive an optimized schedule, which we sketch for $\mathbf{A}_2$. As follows from the dependencies in Table 1 the filtering stages $F_t = (tr, tf)$ and $F_m = (mr, mf)$ have to precede $F_r = (lp, dp, re, rf)$ and $F_s = (si, sf)$ for admissibility, and F_t also has to precede $F_f = (fc, ff)$. We pairwise computed the most efficient schedule of two stages using inequation 1. As a result, we moved F_f before F_r and inserted *lp* before *fc*. Also, we postponed F_r after F_s as illustrated in Figure 2d. Altogether, we finished with the optimized schedule π_2^* for $\mathbf{A}_2$:

$$\pi_2^* = (se, tk, tr, tf, lp, fc, ff, mr, mf, si, sf, dp, re, rf)$$

For $\mathbf{A}_1$ and $\mathbf{A}_3$, we accordingly obtained π_1^* and π_3^*:

$$\begin{aligned}
\pi_1^* &= (se, tk, si, sf, tr, tf, mr, mf, lp, fc, ff, re, rf) \\
\pi_3^* &= (se, tk, tr, tf, mr, mf, si, sf, lp, fc, ff, dp, re, rf)
\end{aligned}$$

As baselines, we used all pipelines with one of the schedules π_a, π_b, and π_c from Figure 2a–c. In fact, π_c resembles the schedule from [2]. Except for π_1^*, we implemented a pipeline $\Pi_{i,j}$ in Java for each $\mathbf{A}_i$ and π_j. π_1^* applies *si* before *tr* and *mr*. Thus, a pipeline with π_1^* is only admissible for $\mathbf{A}_1$, which contains si_1 for *si*. We ran all pipelines five times on the test set using a *2 GHz Intel Core 2 Duo MacBook* with *4 GB RAM* and averaged over the response times.

Effectiveness Results The effectiveness of each pipeline is given in Table 2. Precision and recall increase from $\mathbf{A}_1$ to $\mathbf{A}_3$ significantly. Only in case of $\mathbf{A}_2$, the optimization impaired the effectiveness: the F_1-scores of both $\Pi_{2,2} = \langle \mathbf{A}_2, \pi_2^* \rangle$ and $\Pi_{2,3} = \langle \mathbf{A}_2, \pi_3^* \rangle$ are one point lower than for the baselines. However, this is noise from the TreeTagger, which operates

[1] Revenue Corpus, http://infexba.upb.de

[2] TreeTagger, http://code.google.com/p/tt4j

[3] mate-tools, http://code.google.com/p/mate-tools

Table 3: The average sentence response time $t(\Pi_{i,j})$ with standard deviation $\sigma_{i,j}$ of each admissible pipeline $\Pi_{i,j} = \langle \mathbf{A}_i, \pi_j \rangle$ measured in milliseconds.

π_j	$t(\Pi_{1,j})$	$\sigma_{1,j}$	$t(\Pi_{2,j})$	$\sigma_{2,j}$	$t(\Pi_{3,j})$	$\sigma_{3,j}$
	algorithm set $\mathbf{A}_1$		*algorithm set* $\mathbf{A}_2$		*algorithm set* $\mathbf{A}_3$	
π_a	3.23	0.07	51.05	0.40	168.40	0.57
π_b	2.86	0.09	49.66	0.28	167.85	0.70
π_c	2.54	0.08	15.54	0.23	45.16	0.53
π_1^*	**2.44**	0.03	–	–	–	–
π_2^*	2.47	0.15	**4.77**	0.06	16.25	0.15
π_3^*	2.62	0.05	4.95	0.09	**10.19**	0.05

on token-level. For such algorithms, filtering may influence the feature values of tokens near sentence boundaries.

Efficiency Results Table 3 lists $t(\Pi_{i,j})$ of each $\Pi_{i,j}$. For $\mathbf{A}_1$, our method led to a decrease from 3.23 ms to 2.44 ms. This improvement seems small, but note that se_1 and tk_1 already need over 1 ms (cf. Table 1). In case of $\mathbf{A}_2$, the response time was reduced to less than 1/10 of $t(\Pi_{2,a})$ and to less than 1/3 of $t(\Pi_{2,c})$. For $\mathbf{A}_3$, the pipeline $\Pi_{3,3}$ with $t(\Pi_{3,3}) = 10.19$ ms is over 4 times as fast as the most efficient baseline $\Pi_{3,c}$. Moreover, $\Pi_{3,3}$ clearly outperforms $\Pi_{3,2}$. This offers evidence for the benefit of paradigm 4, while especially the scheduling of $\mathbf{A}_1$ profits most from paradigm 1 and 2. Accordingly, looking at the standard deviations, we see that π_b is significantly faster than π_a only for $\mathbf{A}_1$.

Efficiency at Effectiveness Using the F_1-score as effectiveness $\mathcal{E}$, we plotted $\mathcal{F}@\mathcal{E}$ for the three algorithm sets and the six schedules in Figure 3. The interpolated curves of the schedules have the shape introduced in Section 1, and we observe only a moderate slope for the optimized schedules. Also, $\mathcal{F}@\mathcal{E}$ cancelled out effects of high memory load from applying complex algorithms, such as dp_2, to many sentences. For instance, this yielded $\mathcal{F}(\Pi_{3,a})/\mathcal{F}(\Pi_{3,3}) = 15.1$ as opposed to $t(\Pi_{3,a})/t(\Pi_{3,3}) = 16.5$.

Discussion The three algorithm sets mainly differ in the uniformity of the algorithms' response times (cf. Table 1), which gives rise to the potential of lazy evaluation and optimized scheduling. In general, the room for improvement depends on the density and distribution of task-relevant information in input texts. The more dense relevant information occurs, the less improvement through filtering can be expected, and, the more spread event-related information is across the text, the larger the statement size w must be in order to achieve high recall. Still, our method is generic in that it allows to integrate arbitrary text analysis algorithms and in that it can, in principle, be applied to any IE task.

4. CONCLUSION

Existing approaches to run-time efficiency in IE mostly rely on the development of fast *algorithms*. In contrast, we presented a method to increase the efficiency of IE *pipelines*. Using a system-independent measure, we showed for a complex IE task and three algorithm sets that a pipeline can be significantly sped up without impairing its effectiveness. The improvement is especially high if the response times of the employed algorithms differ largely. Also, the impact depends on the density and the distribution of task-relevant information. These observations—along with the four optimization paradigms of our method—are a step towards a theory of efficiency in IE, which we plan to extend and underpin within forthcoming research.

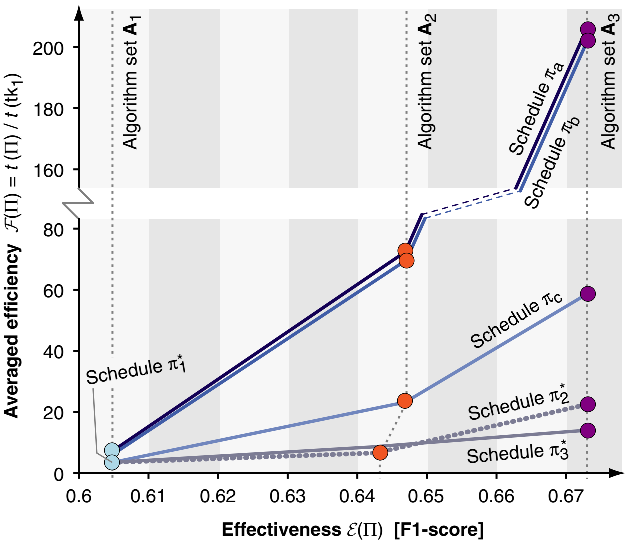

Figure 3: Averaged efficiency at effectiveness $\mathcal{F}@\mathcal{E}$ of all admissible pipelines $\Pi_{i,j} = \langle \mathbf{A}_i, \pi_j \rangle$ at the three effectiveness levels represented by $\mathbf{A}_1$, $\mathbf{A}_2$, and $\mathbf{A}_3$.

5. REFERENCES

[1] E. Agichtein and L. Gravano. Querying Text Databases for Efficient Information Extraction. In *ICDE*, pp. 113–124, 2003.

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

[3] A. Björkelund, B. Bohnet, L. Hafdell, and P. Nugues. A High-Performance Syntactic and Semantic Dependency Parser. In *COLING: Demonstrations*, pp. 33–36, 2010.

[4] B. Bohnet. Very High Accuracy and Fast Dependency Parsing is not a Contradiction. In *COLING*, pp. 89–97, 2010.

[5] M.J. Cafarella, D. Downey, S. Soderland, and O. Etzioni. KnowItNow: Fast, Scalable Information Extraction from the Web. In *HLT and EMNLP*, pp. 563–570, 2005.

[6] G. Forman and E. Kirshenbaum. Extremely Fast Text Feature Extraction for Classification and Indexing. In *CIKM*, pp. 1221–1230, 2008.

[7] U. Germann, M. Jahr, K. Knight, D. Marcu, and Y. Yamada. Fast Decoding and Optimal Decoding for Machine Translation. In *ACL*, pp. 228-235, 2001.

[8] A. Pauls and D. Klein. k-best A* Parsing. In *ACL and IJCNLP*, pp. 958–966, 2009.

[9] S. Petrov. *Coarse-to-Fine Natural Language Processing.* PhD Thesis, University of California at Berkeley, 2009.

[10] L. Ratinov and D. Roth. Design Challenges and Misconceptions in Named Entity Recognition. In *CoNLL*, pp. 147–155, 2009.

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

[12] B. Stein, S. Meyer zu Eissen, G. Gräfe, and F. Wissbrock. Automating Market Forecast Summarization from Internet Data. In *WWW/Internet*, pp. 395–402, 2005.

[13] H. Wachsmuth, P. Prettenhofer, and B. Stein. Efficient Statement Identification for Automatic Market Forecasting. In *COLING*, pp. 1128–1136, 2010.

[14] D.C. Wimalasuriya and D. Dou. Components for Information Extraction: Ontology-Based Information Extractors and Generic Platform. In *CIKM*, pp. 9–18, 2010.