Optimal Scheduling of Information Extraction Algorithms

Henning WACHSMUTH\textsuperscript{1} Benno STEIN\textsuperscript{2}

(1) UNIVERSITÄT PADERBORN, s-lab – Software Quality Lab, Paderborn, Germany
(2) BAUHAUS-UNIVERSITÄT WEIMAR, Web Technology and Information Systems, Weimar, Germany
hwachsmuth@s-lab.upb.de, benno.stein@uni-weimar.de

Abstract
Most research on run-time efficiency in information extraction is of empirical nature. This paper analyzes the efficiency of information extraction pipelines from a theoretical point of view in order to explain empirical findings. We argue that information extraction can, at its heart, be viewed as a relevance filtering task whose efficiency traces back to the run-times and selectivities of the employed algorithms. To better understand the intricate behavior of information extraction pipelines, we develop a sequence model for scheduling a pipeline’s algorithms. In theory, the most efficient schedule corresponds to the Viterbi path through this model and can hence be found by dynamic programming. For real-time applications, it might be too expensive to compute all run-times and selectivities beforehand. However, our model implies the benchmarks of filtering tasks and illustrates that the optimal schedule depends on the distribution of relevant information in the input texts. We give formal and experimental evidence where necessary.

Title and Abstract in German

Optimales Scheduling von Information-Extraction-Verfahren


Keywords: information extraction, theory, efﬁciency, scheduling.

Keywords in German: Information Extraction, Theorie, Efﬁzienz, Scheduling.
1 Introduction

Information extraction deals with the analysis of natural language text in order to find relevant information about entities, relations, and events. Relations typically involve two named or numeric entities, such as “Apple was founded in 1976”, whereas events model more complex dependencies between a number of entities, as for example: “IBM ended Q1 2011 with $13.2 billion of cash on hand and free cash flow of $0.8 billion.” If event templates with three or more arguments have to be filled, several analysis steps are performed. In terms of run-time efficiency, it is reasonable to filter only those portions of text after each step that contain the information sought for and, thus, may be relevant for one of the events in question. In this respect, a conjunctive filtering task is to be solved for each event type. Consequently, the organization of the analysis will have a noticeable impact on the efficiency of the extraction process.

In information extraction, a conjunctive filtering task is addressed with an algorithm pipeline \( \Pi = \langle A, \pi \rangle \), comprised of a set of algorithms \( A \) and a schedule \( \pi \) that prescribes the order of algorithm application. Each algorithm in \( A \) filters an argument of the event in question by classifying a portion of its input text as relevant. In the first example above, both “Apple” and “1976” are such arguments. The output of \( \Pi \) is given by the arguments in the intersection of the filtered portions. In order to work properly, an algorithm may require as input a preprocessing of the text by other algorithms. As in (Wachsmuth et al., 2011), we call a pipeline admissible if its schedule ensures that the input constraints of all algorithms are fulfilled. Accordingly, all admissible pipelines \( \langle A, \pi_1 \rangle, \ldots, \langle A, \pi_l \rangle \) classify the same portion of text as relevant. I.e., they entail the same effectiveness, e.g. quantified as \( F_1 \)-score, in solving an extraction task.

We observe an increasing demand to extract complex information structures in applications of computational linguistics, e.g. the BioNLP Shared Task 2011 included event types where entities of three types had to be related to an event in four roles (Kim et al., 2011). Our research question refers to the outlined efficiency potential and can be stated as optimization problem:

**Given an algorithm set \( A \) that solves an information extraction task. Determine a schedule \( \pi^* \) such that the pipeline \( \langle A, \pi^* \rangle \) is run-time optimal under all admissible pipelines \( \langle A, \pi_1 \rangle, \ldots, \langle A, \pi_l \rangle \).**

While a few practical scheduling approaches exist, in this paper we discuss the problem’s nature. We consider a text as an ordered set of atomic text units. The efficiency of a pipeline \( \Pi = \langle A, \pi \rangle \) depends on the run-times and selectivities of the algorithms in \( A \) when being applied to the text units. The selectivity defines the portion of text units classified as relevant by an algorithm in \( A \). Only this portion forms the input of the next algorithm in \( \pi \). Hence, the optimization problem consists in finding the schedule that minimizes the sum of the algorithms’ run-times. An optimal solution requires a global analysis due to the recurrent structure of the run-times and selectivities. We represent this structure in a sequence model and solve it with dynamic programming. The optimization view raises an important question: To what extent does the optimality of a schedule depend on the distribution of relevant information in input texts?

**Contributions.** We provide a theoretical approach and the theoretically optimal solution to the construction of run-time efficient information extraction pipelines. First, we model the scheduling of a set of extraction algorithms as a dynamic program, which yields the optimal pipeline schedule (Section 3). Then, we offer formal and quantitative evidence that the distribution of relevant information is decisive for the efficiency of a pipeline (Section 4).

\[\text{[1]Different event types imply disjunctions of conjunctive filtering tasks. Filtering is common in information extraction (Cowie and Lehnert, 1996; Agichtein, 2005), but until today most approaches rely only on heuristics (Sarawagi, 2008).}\]
2 Related Work

One of the most recognized approaches to efficient information extraction refers to the open domain system TextRunner (Banko et al., 2007). TextRunner employs special index structures and fast extraction algorithms, but it is restricted to simple relations. In contrast, we target at template filling tasks that relate several entities to events (Cunningham, 2006). We approach such tasks with classic pipelines where each algorithm takes on one analysis, e.g. a certain type of entity recognition (Grishman, 1997). The decisions within a pipeline can be viewed as irreversible, which allows to perform filtering. Hence, an algorithm can never make up for false classifications of its predecessors, as in iterative or probabilistic pipeline approaches (Finkel et al., 2006; Hollingshead and Roark, 2007). Accordingly, we do not deal with joint extraction, which often suffers from its computational cost (Poon and Domingos, 2007).

In (Wachsmuth et al., 2011), we introduced a generic method to construct efficient pipelines that achieves run-time improvements of one order of magnitude without harming a pipeline’s effectiveness. Similarly, Shen et al. (2007) and Doan et al. (2009) optimize schedules in a declarative extraction framework. These works give only heuristic hints on the reasons behind empirical results. While some algebraic foundations of scheduling are established for rule-based approaches by Chiticariu et al. (2010), we explain the determinants of efficiency for any set of extraction algorithms. To the best of our knowledge, we are the first to address scheduling in information extraction with dynamic programming, which relies on dividing a problem into smaller subproblems and solving recurring subproblems only once (Cormen et al., 2009).

In our research we analyze the impact of text types on the efficiency of information extraction pipelines. Existing work on text types in information extraction mainly deals with the filtering of promising documents, such as (Agichtein and Gravano, 2003). Instead, we identify the properties of a text that influence run-time optimality. For optimizing rule-based pipelines, samples from a text corpus are analyzed by Wang et al. (2011) in order to collect statistics similar to the ones used for optimizing database queries.

In the database community, run-time optimization has a long tradition. While dynamic programming is used for join operations since the pioneer System R (Selinger et al., 1979), template filling corresponds to processing And-conditioned queries that select those tuples of a database table whose values fulfill a desired attribute conjunction. The optimal schedule for such a query is obtained by ordering the involved attribute tests according to their increasing number of expected matches, i.e., without having to solve a dynamic program (Ioannidis, 1997). The filtering problem in information extraction looks pretty similar, but a simple ordering strategy fails because the effort for extracting one type of information (which is the analog of an attribute test) is not constant; it depends on the applied extraction algorithm.

3 Optimal Scheduling of Information Extraction Algorithms

We now develop a theoretical model for the efficiency of scheduling a fixed set of extraction algorithms. In order to maintain effectiveness, we consider only admissible information extraction pipelines, i.e., pipelines where the input constraints of all algorithms are fulfilled (cf. Section 1). For an algorithm set \( \mathbf{A} \), different admissible pipelines can vary in efficiency if they apply the algorithms in \( \mathbf{A} \) to different numbers of text units. The run-time \( t(\Pi) \) of a pipeline \( \Pi = \langle \mathbf{A}, \pi \rangle \) on an ordered set of text units \( U \) depends on the run-time \( t_i \) and the filtered portion of text units \( R_i \) of each algorithm \( A_i \in \mathbf{A} \) within the schedule \( \pi \). Assume that \( \pi \) schedules \( \mathbf{A} \) as \( (A_1, \ldots, A_m) \). If \( \Pi \) analyzes only the filtered portions of text units, then \( A_1 \) processes \( U \), while \( A_2 \)
processes $R_1(U), A_3$ processes $R_1(U) \cap R_2(U)$, and so on. Therefore, the run-time of $\Pi$ is
\[
t(\Pi) = t_1(U) + \sum_{i=2}^{m} t_i(\bigcap_{k=1}^{i-1} R_k(U)). \tag{3.1}
\]
From [Wachsmuth et al. 2011] we infer that, in the optimal schedule of two independent extraction algorithms $A_1$ and $A_2$, $A_1$ precedes $A_2$ on an ordered set of text units $U$ if and only if
\[
t_1(U) + t_2(R_1(U)) < t_2(U) + t_1(R_2(U)). \tag{3.2}
\]
Obviously, the run-time of an algorithm within a pipeline results from the portion of text units filtered by its preceding algorithm. Hence, the run-time $t(\Pi^{(m)})$ of a pipeline $\Pi^{(m)} = (A_1, \ldots, A_m)$ is the sum of the run-time $t(\Pi^{(m-1)})$ of $\Pi^{(m-1)} = (A_1, \ldots, A_{m-1})$ and the run-time of $A_m$ on the text units $R(\Pi^{(m-1)})$ filtered by $\Pi^{(m-1)}$. This recursive definition resembles the one used by the Viterbi algorithm [Viterbi 1967], which operates on hidden Markov models to compute the Viterbi path, i.e. the most likely sequence of states for a given sequence of observations. In the following, we adapt the Viterbi algorithm to schedule an algorithm set $A$, such that the Viterbi path corresponds to the run-time optimal schedule for an ordered set of text units $U$.

**The Sequence Model.** To represent scheduling, we define a sequence model similar to a hidden Markov model. Each state $a_i$ in the model corresponds to having applied an algorithm $A_i \in A$, $i \in \{1, \ldots, m\}$. A transition to $a_i$ denotes the application of $A_i$ with run-time $t_i$. Instead of observations, the model contains the positions $p^{(1)}, \ldots, p^{(m)}$ of a schedule, and having applied $A_i$ at $p^{(j)}$ means having filtered a portion of text units $R_i$. In contrast to the state and emission probabilities of a hidden Markov model, however, $t_i$ and $R_i$ are not directly influenced by the preceding state or the current position, but they depend on the currently filtered portion of text units. For this reason, we include a running variable $R(\Pi_{k-1})$ in the model that stores the filtered portion of $A_k$ at position $p^{(j-1)}$. Initially, the running variable is set to the ordered set of text units $U$. Figure 1(a) illustrates the described sequence model.

In classic information extraction pipelines, no algorithm is applied multiple times. This means that each state must occur exactly once in a path through the model. Also, for admissibility, a state may only be reached if all input constraints of the associated algorithm are fulfilled. Hence, we define that an algorithm $A_i \in A$ is applicable at position $p^{(j)}$ if $A_i$ has not been applied at $p^{(1)}$ to $p^{(j-1)}$, and if all input constraints of $A_i$ are fulfilled by the algorithms at these positions.

**The Pipeline Viterbi Algorithm.** For an observation $x_j$, the original Viterbi algorithm computes the most likely path with state $y_i$ at $x_j$ in an iterative (dynamic programming) manner. Accordingly, we store for each position $p^{(j)}$ the run-time optimal pipeline from $p^{(1)}$ to $p^{(j)}$ and algorithm $A_i$ at $p^{(j)}$. To this end, we iteratively compute the run-time of each $\Pi_{i}^{(j)}$ based on the set of run-time optimal pipelines $\Pi_{k}^{(j-1)}$ after which $A_i$ is applicable. If $\Pi_{k}^{(j-1)}$ is empty, then $A_i$ is not applicable, denoted as $\perp$. The recursive function of the Pipeline Viterbi algorithm can be derived from Equation 3.1 and Inequality 3.2.
\[
t(\Pi_{i}^{(j)}) = t_i(U) \quad \text{if } j = 1
\]
\[
t(\Pi_{i}^{(j)}) = \min_{\Pi_{j-1} \in \Pi^{(j-1)}} (t(\Pi_{i}^{(j-1)}) + t_i(R(\Pi_{i}^{(j-1)}))) \quad \text{else}
\]
To solve the problem of optimal scheduling with dynamic programming, we keep track of all values $t(\Pi_{i}^{(j)})$ and $R(\Pi_{i}^{(j)})$. Additionally, we store $\Pi_{i}^{(j)}$ to finally obtain the optimal pipeline $\langle A, \pi^* \rangle$ for $A = \{A_1, \ldots, A_m\}$ and a set of text units $U$, as sketched in the following pseudocode.

\[\text{Notice that the sequence model does not have the Markov property [Manning and Schütze 1999], but we define the sequence model accordingly in order to make it viable for the Viterbi algorithm.}\]
The optimality of the returned pipeline follows from the optimal solutions to all subproblems, i.e., all pipelines \( \Pi_i^{(j)} \). We do not prove the correctness of the Pipeline Viterbi algorithm formally here. The proof idea is that, by definition, the order of any two algorithms \( A_1, A_2 \in A \) is only variable if neither \( A_1 \) depends on \( A_2 \) nor vice versa. In this case, applying \( A_1 \) and \( A_2 \) in sequence is a commutative operation. Thus, \( \Pi_k^{(j-1)} \) will always be optimal for \( A_i \) at position \( p^{(j)} \), no matter what comes afterwards. Consequently, \( \Pi_i^{(j)} \) is optimal.

**Computational Cost.** The cost of running the developed algorithm for an algorithm set
\( \mathbf{A} = \{A_1, \ldots, A_m\} \) can be inferred from the pseudocode above: if both the run-times \( t_i \) and the filtered portions of text units \( R_i \) of all algorithms \( A_i \in \mathbf{A} \) were given, the cost would follow from the \( n^2 \) loop iterations, where \( \Pi_i^{(j)} \) is determined based on at most \( m - 1 \) pipelines \( \Pi_i^{(j-1)} \). This results in \( O(m^3) \) operations. Practically, these values are not known beforehand but need to be measured during execution. In the worst case, all algorithms have an equal run-time \( t_{\text{max}}(U) \) on \( U \) and they filter the whole text, i.e., \( R_i(U) = U \) for each \( A_i \). So, all algorithms must indeed be applied to \( U \), which leads to an overall upper bound of \( O(m^3 \cdot t_{\text{max}}(U)) \). \(^3\)

### 4 The Impact of Text Types on the Efficiency of Pipelines

In this section, we first analyze the influence of text types on the optimality of schedules. Then, we reveal that the efficiency of an information extraction pipeline is governed by the distribution of relevant entities, relations, and events in the input texts. In all experiments, we evaluated the following set-up on a 2 GHz Intel Core 2 Duo MacBook with 4 GB memory:

**Data.** We processed the training sets of two German text corpora: the Revenue corpus introduced in (Wachsmuth et al., 2010) and the corpus of the CoNLL’03 shared task (Tjong Kim Sang and De Meulder, 2003).\(^4\) The former contains 752 online business news articles with 21,586 sentences, whereas 553 mixed classic newspaper articles with 12,713 sentences refer to the latter.

**Task.** The conjunctive filtering task that we consider emanates from the purpose of the Revenue corpus, namely, we define a text unit to be classified as relevant as a sentence that represents a forecast and that contains a money entity, a time entity, and an organization name.

**Algorithms.** We employed four algorithms that filter text units: regex-based money and time entity recognizers \( A_M \) and \( A_T \), the CRF-based Stanford NER system \( A_N \) (Finkel et al., 2005; Faruqui and Padó, 2010) for organization names, and the SVM-based forecast event detector \( A_F \) from (Wachsmuth et al., 2011) that needs time entities as input. Further algorithms were used only as preprocessors. In all experiments we executed each preprocessing algorithm right before its output was needed. Hence, we simply speak of the algorithm set \( \mathbf{A}_1 = \{A_M, A_T, A_N, A_F\} \) in the following without loss of generality. All algorithms in \( \mathbf{A}_1 \) operate on sentence-level.

**Application of the Pipeline Viterbi Algorithm.** On both corpora, we executed all applicable pipelines \( \Pi_i^{(j)} \) for \( \mathbf{A}_1 \) to obtain the filtered portions \( R(\Pi_i^{(j)}) \) and to measure all run-times \( t(\Pi_i^{(j)}) \), averaged over ten runs. All standard deviations were lower than 1.0s on the Revenue corpus and 0.5s on the CoNLL’03 corpus, respectively. For clarity, we omitted them in Figure 2 and 3, which visualize the Pipeline Viterbi algorithm as a trellis. Also, the two figures state only the number of sentences of each filtered portion of text units instead of the portions themselves. In the trellises the bold arrows denote the Viterbi paths. \( A_T \) is scheduled first and \( A_N \) is scheduled last in both optimal cases, but only on the Revenue corpus it is more efficient to apply \( A_F \) before \( A_M \). So, the run-time optimality of schedules is corpus-dependent.

Seemingly, one reason lies in the text units classified as relevant by \( \mathbf{A}_1 \): 215 of the sentences in the Revenue corpus are returned by each admissible pipeline, which is about 1%, as opposed to 2 sentences of the CoNLL’03 corpus (0.01%). A closer look uncovers significant differences between the trellises, e.g. the pipeline \((A_T, A_M)\) filters 3818 sentences of the Revenue corpus (17.7%), but only 82 CoNLL’03 sentences (0.6%). These values originate in the distribution

---

\(^3\)Besides the unrealistic nature of the worst case, the value \( m^3 \) ignores that an algorithm is applied only once and only if its input constraints are fulfilled. Thus, the cost of the Pipeline Viterbi algorithm will be much lower in practice.

\(^4\)In general, the evaluated determination of optimal schedules works on input texts of any language, of course.
of entities in the two corpora. Additionally, the run-times of $A_N$ (Revenue: 91.63s, CoNLL'03: 48.03s) emphasize the general importance of optimizing the efficiency of pipelines.

**The Distribution of Relevant Information.** It seems reasonable to assume that the fraction of text units, which are classified as relevant, influences the run-time optimality of a schedule. In fact, it is not the relevant but the irrelevant text units that matter as follows from Theorem 1.

**Theorem 1.** Let $\Pi^* = \langle A, \pi^* \rangle$ be run-time optimal on a set of text units $U$ under all admissible pipelines for a set of extraction algorithms $A$. Let $R \subseteq U$ be the set of text units classified as relevant by $\Pi^*$. Now let $R' \subseteq U'$ be any other set of text units classified as relevant by $\Pi^*$. Then $\Pi^*$ is run-time optimal on $(U \setminus R) \cup R'$.

**Proof.** Within the proof, we denote the run-time of an arbitrary pipeline $\Pi$ on $U$ as $t(U)(\Pi)$ and accordingly on other sets of text units. By hypothesis, $\Pi^* = \langle A, \pi^* \rangle$ is run-time optimal on $U$, i.e., for any pipeline $\Pi' = \langle A, \pi' \rangle$ with $\pi' \neq \pi^*$, we have

$$t(U)(\Pi^*) \leq t(U)(\Pi'). \quad (4.1)$$

Now, for an algorithm set $A$, all admissible pipelines classify the same set of text units $R \subseteq U$ as relevant. So, on each text unit in $R$, all algorithms must be applied irrespective of the schedule. Hence, for any two admissible pipelines $\Pi_1 = \langle A, \pi_1 \rangle$ and $\Pi_2 = \langle A, \pi_2 \rangle$, we can expect

$$t(R)(\Pi_1) = t(R)(\Pi_2). \quad (4.2)$$

Obviously, the same holds for $R'$. Thus, from Equation 4.1 and 4.2, Theorem 1 follows:
We provide a theoretical model to explain empirical findings when optimizing a pipeline’s run-time efficiency at a given effectiveness. Based on this model, we propose a dynamic programming algorithm to determine the optimal schedule of a fixed set of extraction algorithms on input texts of any language. Together, the model and the algorithm give a comprehensive insight into the scheduling problem of conjunctive filtering tasks, such as template filling. Also, they represent a fast means to compute the theoretically optimal solution for benchmarks in

5 The declining curves in Figure 4(c) seem counterintuitive. However, sentences with money entities often also contained other relevant information such as time entities. So, the average time to process them was rather high.
future research and applications of computational linguistics. Our experiments showed that different types of texts may lead to different optimal schedules. For homogeneous input texts, a solution is to transform dynamic programming into an $A^*$ algorithm (Huang, 2008) with a heuristic based on run-time estimations of the employed algorithms. $A^*$ can then be executed on a sample of texts in order to find a near-optimal schedule. For more heterogeneous texts, a schedule should be chosen in respect of a classification of the input text at hand.

References


