Automatic Pipeline Construction for Real-Time Annotation

Henning Wachsmuth, Mirko Rose, and Gregor Engels

Universität Paderborn
s-lab – Software Quality Lab
Paderborn, Germany
hwachsmuth@s-lab.upb.de, {mrose,engels}@upb.de

Abstract. Many annotation tasks in computational linguistics are tackled with manually constructed pipelines of algorithms. In real-time tasks where information needs are stated and addressed ad-hoc, however, manual construction is infeasible. This paper presents an artificial intelligence approach to *automatically* construct annotation pipelines for given information needs and quality prioritizations. Based on an abstract ontological model, we use partial order planning to select a pipeline's algorithms and informed search to obtain an efficient pipeline schedule. We realized the approach as an expert system on top of Apache UIMA, which offers evidence that pipelines can be constructed ad-hoc in near-zero time.

1 Introduction

Information extraction and other applications of computational linguistics deal with the annotation of text, which often takes several interdependent steps. A typical annotation task is to relate different entity types to event anchors in a text. E.g., the BioNLP shared task GENIA [13] included event types like *PositiveRegulation(Theme, Cause, Site, CSite)* whose instances are contained in sentences like: "Eo-VP16, but not the empty GFP retrovirus, increased perforin expression in both WT and T-bet-deficient CD8+ T cells." Before entities and events can be related, the respective types must have been recognized, which normally requires linguistic annotations, e.g. part-of-speech tags. These in turn can only be added to a text that has been segmented into lexical units, e.g. into sentences.

Because of the interdependencies of the steps, the standard way to address such an information need is with an annotation pipeline $\Pi = \langle \mathbf{A}, \pi \rangle$, where $\mathbf{A}$ is a set of algorithms and π is the schedule of these algorithms. Each algorithm in $\mathbf{A}$ takes on one analysis by annotating certain types of output information, and it requires certain types of information as input. The schedule π has to ensure that all input requirements are fulfilled. Different algorithms for an analysis may vary in their requirements, thereby placing different constraints on π. Moreover, Π may have to meet efficiency and effectiveness criteria, e.g. measured as run-time or F_1-score. Often, an increase in effectiveness is paid with a decrease in efficiency and vice versa. The set $\mathbf{A}$ must therefore be composed in respect of the quality criteria at hand. When Π incorporates filtering steps, π affects the efficiency of Π

as well [21]. As a result, pipeline construction faces two involved challenges: (1) The selection of a set of algorithms that addresses a given information need and that complies with the given quality criteria. (2) The determination of an efficient schedule that fulfills the input requirements of all algorithms.

Traditionally, the construction of efficient and effective annotation pipelines is performed manually by experts with domain knowledge. However, recent developments suggest that the future of annotation tasks will be real-time search applications, where regular internet users state information needs ad-hoc [7,15]. In such scenarios, the only way to be able to directly respond to a user is to construct and execute annotation pipelines in an automatic manner.

In this paper, we present an approach to automatically construct annotation pipelines based on techniques from artificial intelligence [18]. To this end, we formalize expert knowledge on algorithms and information types within an abstract ontological model. Given an information need, we rely on *partial order planning* to select a set of algorithms with a defined partial order. Where appropriate, filtering steps are integrated, while a prioritization of quality criteria allows to influence the trade-off between a pipeline's presumed efficiency and effectiveness. To obtain a correct and efficient schedule, we apply *informed search* using estimations of the algorithms' run-times. In large-scale scenarios, A^* search could find a near-optimal schedule on a sample of texts. For ad-hoc pipeline construction, however, we argue that a greedy best-first search strategy is more reasonable.

We realized our approach as an open-source *expert system* on top of Apache UIMA [2]. Experiments with this system suggest that automatic pipeline construction can be performed in near-zero time for realistic numbers of algorithms. Altogether, our main contributions are three-fold:

1. We formalize the expert knowledge that is needed to automatically address annotation tasks in an abstract ontological model (Section 3).
2. We approach the automatic construction of efficient and effective annotation pipelines with partial order planning and informed search (Section 4).
3. We provide an expert system that constructs and executes annotation pipelines ad-hoc, thereby qualifying for real-time applications (Section 5).

2 Related Work

Planning and informed search represent algorithmic foundations of artificial intelligence [18]. The latter has been used to speed up several tasks in natural language processing, e.g. parsing [16]. In [20], we solve optimal scheduling theoretically, but we suggest to apply informed search in practice. In case of planning, our work resembles [6] where partial order planning is proposed to compose information extraction algorithms. The authors do not consider quality criteria, though, nor do they realize and evaluate their approach. This also holds for [22], in which knowledge discovery workflows of minimum length are planned for an ontology of data mining algorithms. A planning approach that sums up the costs of steps is given in [17] for data stream processing. In annotation tasks, however, the values of criteria such as precision cannot simply be aggregated.

Annotation tasks are often tackled with pipelines of algorithms [3]. For manually constructing such pipelines, software frameworks like Apache UIMA [2] and GATE [10] provide tool support. Still, manual pipeline construction tends to be error-prone and cost-intensive [5]. Recently, a UIMA-based system that automatically composes and executes a defined set of algorithms was presented in [11], but it is not yet available. The author previously worked on U-Compare [12], which allows a fast but manual construction and evaluation of pipelines. In contrast, we perform construction fully automatically. We realized our approach as an expert system for regular users. Expert systems have been used for scheduling and construction since the early times of artificial intelligence [9].

Automatic pipeline construction is important when different tasks require different algorithms. Major open information extraction systems such as ReVerb currently avoid this scenario by analyzing task-independent syntactical structures rather than semantic concepts [8]. However, they are restricted to binary relations. More complex tasks are e.g. addressed by declarative approaches like SystemT [4]: users specify needs by defining logical constraints of analysis steps, while the system manages the workflow. SystemT optimizes schedules, but it targets at expert users and works only with rule-based algorithms.

Efficient schedules benefit from the filtering of relevant information, which has a long tradition in information extraction [1]. In [21], we show how to construct efficient pipelines for any set of extraction algorithms. We adopt parts of this approach here, but we address arbitrary annotation tasks.

3 An Ontological Model of Annotation Tasks

In this section, we model the expert knowledge needed to automatically address annotation tasks. Many annotation tasks target at information of domain- and application-specific *type systems*, as e.g. in the GENIA example from Section 1. Still, these type systems instantiate the same abstract structures. In particular, most works distinguish between *primitive types*, such as integers or strings, and hierarchically organized *annotation types*, which assign syntactic or semantic concepts to spans of text. Annotation types can have *features* whose values are either primitives or annotations. Among others, features thereby allow to model concepts like relations and events as annotations. Not all features of an annotation are always set. We call a feature *active* if it has a value assigned.

Now, an information need refers to a set of annotation types and features. Besides, it may place *value constraints* on the text spans to be annotated. Within GENIA, a constraint could be to keep only positive regulation events whose cause is “Eo-VP16”, i.e., *PositiveRegulation(_, “Eo-VP16”, _, _)*. In general, we define the abstract *information type* to be found in annotation tasks as follows.

Information Type. A set of annotations of an arbitrary but fixed type denotes an information type C if it contains all annotations that meet two conditions:

1. **Active Feature.** The annotations in C either have no active feature or they have the same single active feature.
2. **Constraints.** The annotations in C fulfill the same set of value constraints.

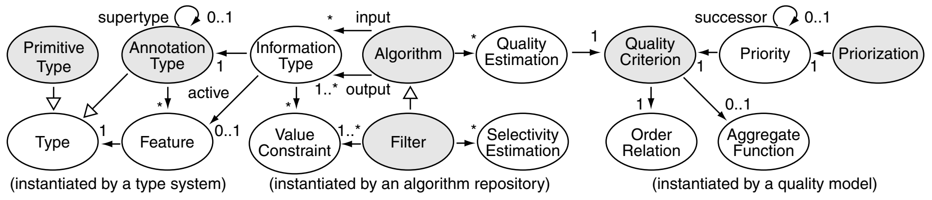

Fig. 1. Abstract ontological model of the expert knowledge for addressing annotation tasks. White and black arrowheads denote "subclass" and "has" relations, respectively. Grey-colored concepts are instantiated by concrete classes within an application.

By defining C to have at most one active feature, we obtain a normalized unit of information in annotation tasks. A single information need can be stated as a set of information types $\mathbf{C} = \{C_1, \ldots, C_{|\mathbf{C}|}\}$, meaning a conjunction $C_1 \wedge \ldots \wedge C_{|\mathbf{C}|}$. Different information needs result in disjunctions of such conjunctions.

When fulfilling a need like *PositiveRegulation(_, "Eo-VP16", _, _)*, a certain effectiveness and efficiency is achieved. Such *quality criteria* are used to evaluate whether a solution is good or is better than another one. Conceptually, a quality criterion Q defines an *order relation* for a set of values. Q may have an *aggregate function* that maps any two values $q_1, q_2 \in Q$ (say, run-times) to a value $q \in Q$ (say, the total run-time). An aggregate function allows to infer the quality of a solution from the quality of partial solutions. However, far from all criteria entail such a function. For instance, there is no general way of inferring an overall precision from the precision of two algorithms. Similarly, functions that aggregate values of *different* quality criteria rarely make sense. In contrast to other multi-criteria optimization problems [14], weighting different Pareto-optimal solutions (where any improvement in one criterion worsens others) hence does not seem reasonable in annotation tasks. Instead, we propose *quality prioritizations*:

Quality Prioritization. A prioritization $P = (Q'_1, \ldots, Q'_{|\mathbf{Q}|})$ is a permutation of a set of quality criteria $\mathbf{Q} = \{Q_1, \ldots, Q_{|\mathbf{Q}|}\}$ that defines an order of importance.

E.g., *(run-time, precision, recall)* targets at the solution with highest recall under all solutions with highest precision under all solutions with lowest run-time. Together, $\mathbf{Q}$ and a set of prioritizations $\mathbf{P} = \{P_1, \ldots, P_{|\mathbf{P}|}\}$ define a *quality model.*

To select a set of algorithms $\mathbf{A}$ for an information need $\mathbf{C}$ that complies with a prioritization P, internal operations of the algorithms do not matter, but only their *input* and *output* behavior. The actual efficiency and effectiveness of an algorithm on a collection or a stream of texts is unknown beforehand. For many algorithms, typical *quality estimations* are known from evaluations, though.

Algorithm. Let $\mathbf{C}$ be a set of information types and $\mathbf{Q}$ a set of quality criteria. Then an algorithm A is a 3-tuple $\langle \mathbf{C}_{in}, \mathbf{C}_{out}, \overrightarrow{q} \rangle$ such that $\mathbf{C}_{in} \neq \mathbf{C}_{out}$ and

– **Input.** $\mathbf{C}_{in} \subseteq \mathbf{C}$ is the set of input information types required by A,
– **Output.** $\mathbf{C}_{out} \subseteq \mathbf{C}$ is the set of output information types A produces, and
– **Estimations.** $\overrightarrow{q} \in (Q_1 \cup \{\perp\}) \times \ldots \times (Q_{|\mathbf{Q}|} \cup \{\perp\})$ contains one value q_i for each $Q_i \in \mathbf{Q}$. q_i defines a quality estimation or it is unknown, denoted as $\perp$.

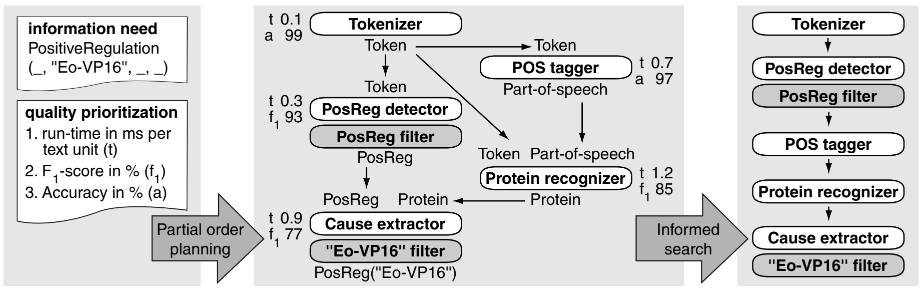

Fig. 2. Pipeline construction for an information need and a quality prioritization. Seven algorithms are selected with partial order planning and scheduled with informed search.

Assume that an algorithm has produced a type $C \in \mathbf{C}$, e.g. *PositiveRegulation*. Then a means to improve efficiency is to further analyze only text units (say, sentences) that contain positive regulation events and that, hence, may be relevant for $\mathbf{C}$ [21]. We call an algorithm a *filter* if it discards text units that do not meet some defined value constraints. In practice, algorithms may combine annotation and filtering operations. Here, we separate filters as they can be created on-the-fly for given information types. Such a filter has a single input type $\mathbf{C}_{in} = \{C_{in}\}$ that equals its output type $\mathbf{C}_{out} = \{C_{out}\}$ except that C_{out} additionally meets the filter's value constraints.[1] While a filter implies a certain selectivity (i.e., a fraction of filtered text units), selectivities strongly depend on the input [20]. So, reasonable *selectivity estimations* can only be obtained during analysis.

Altogether, Figure 1 associates the described concepts in an ontological model. The helper concepts *Type* and *Priority* realize the diversity of features and the permutations of prioritizations. As depicted, the modeled knowledge can be partitioned into three parts that are instiantiated within a concrete ontology:

Annotation Task Ontology. An annotation task ontology Ω denotes a 3-tuple $\langle \mathbf{C}_\Omega, \mathbf{P}_\Omega, \mathbf{A}_\Omega \rangle$ such that $\mathbf{C}_\Omega$ is a set of available information types, $\mathbf{P}_\Omega$ is a set of available quality prioritizations, and $\mathbf{A}_\Omega$ is a set of available algorithms.

4 Automatic Pipeline Construction

We now introduce an artificial intelligence approach to automatically construct annotation pipelines. In particular, we use *partial order planning* to select a set of algorithms and a greedy *informed search* strategy to find an efficient schedule. Figure 2 exemplarily illustrates the application of our approach.

We consider pipeline construction as a planning problem. In artificial intelligence, the term "planning" denotes the process of generating a sequence of actions that transforms an initial state of the world into a specified goal state [18]. A planning problem is defined by its domain and by the task to be addressed. For pipeline construction, we describe the planning problem as follows:

[1] For filters that check the presence of an information type in a text unit, we define that all information types have an implicit constraint "is contained in text unit".

```
Algorithm pipelinePartialOrderPlanning($\mathbf{C}_0, \mathbf{C}_\phi, P_\phi, \mathbf{A}_\Omega$)
 1: Algorithm set $\mathbf{A}_\phi \leftarrow \{A_\phi\}$
 2: Partial schedule $\pi_\phi \leftarrow \emptyset$
 3: Input requirements $\mathbf{\Lambda} \leftarrow \{\langle C_\phi, A_\phi\rangle \mid C_\phi \in \mathbf{C}_\phi \backslash \mathbf{C}_0\}$
 4: while $\mathbf{\Lambda} \neq \emptyset$ do
 5:     Input requirement $\langle C_\mathbf{\Lambda}, A_\mathbf{\Lambda}\rangle \leftarrow \mathbf{\Lambda}$.poll()
 6:     if $C_\mathbf{\Lambda} \in \mathbf{C}_\phi$ then
 7:         Filter $A_F \leftarrow$ createFilter($C_\mathbf{\Lambda}$)
 8:         $\mathbf{A}_\phi \leftarrow \mathbf{A}_\phi \cup \{A_F\}$
 9:         $\pi_\phi \leftarrow \pi_\phi \cup \{(A_F < A_\mathbf{\Lambda})\}$
10:         $\langle C_\mathbf{\Lambda}, A_\mathbf{\Lambda}\rangle \leftarrow \langle A_F.C_{in}, A_F\rangle$
11:     Algorithm $A \leftarrow$ selectBestAlgorithm($C_\mathbf{\Lambda}, P_\phi, \mathbf{A}_\Omega$)
12:     if $A = \bot$ then return $\bot$
13:     $\mathbf{A}_\phi \leftarrow \mathbf{A}_\phi \cup \{A\}$
14:     $\pi_\phi \leftarrow \pi_\phi \cup \{(A < A_\mathbf{\Lambda})\}$
15:     $\mathbf{\Lambda} \leftarrow \mathbf{\Lambda} \cup \{\langle C, A\rangle \mid C \in A.\mathbf{C}_{in} \backslash \mathbf{C}_0\}$
16: return $\langle \mathbf{A}_\phi, \pi_\phi\rangle$
```

Fig. 3. Pseudocode of partial order planning for selecting a set of algorithms $\mathbf{A}_\phi$ (with a partial schedule π_ϕ) that addresses a planning problem $\phi^{(\Omega)} = \langle \mathbf{C}_0, \mathbf{C}_\phi, P_\phi, \mathbf{A}_\Omega\rangle$

Planning Problem. Let $\Omega = \langle \mathbf{C}_\Omega, \mathbf{P}_\Omega, \mathbf{A}_\Omega\rangle$ be an annotation task ontology. Then a planning problem $\phi^{(\Omega)}$ denotes a 4-tuple $\langle \mathbf{C}_0, \mathbf{C}_\phi, P_\phi, \mathbf{A}_\Omega\rangle$ such that

– **Initial State.** $\mathbf{C}_0 \subseteq \mathbf{C}_\Omega$ is the initially given information,
– **Goal.** $\mathbf{C}_\phi \subseteq \mathbf{C}_\Omega$ is the information need to be fulfilled,
– **Quality.** $P_\phi \in \mathbf{P}_\Omega$ is the quality prioritization to be met, and
– **Actions.** $\mathbf{A}_\Omega$ is the set of algorithms available to fulfill $\mathbf{C}_\phi$.

We implicitly model states of a planning domain as sets of information types, thereby reflecting the states of analysis of an input text.[2] So, all states $\mathbf{C}$ with $\mathbf{C}_\phi \subseteq \mathbf{C}$ are goal states. Algorithms represent actions in that they modify states by adding new information types. To solve a problem $\phi^{(\Omega)}$, we hence need a pipeline of algorithms that produces $\mathbf{C}_\phi \backslash \mathbf{C}_0$ while complying with P_ϕ.

4.1 Algorithm Selection Based on Partial Order Planning

We chose partial order planning to select a set of algorithms $\mathbf{A}_\phi$ and to define a partial schedule π_ϕ. In general, this backward approach recursively generates and combines subplans for all preconditions of those actions that achieve a planning goal [18]. Actions may conflict, namely, if an effect of one action violates a precondition of another one. In annotation tasks, however, algorithms only produce information. While filters reduce the input to be processed, they never prevent subsequent algorithms from being applicable [6].

In Figure 3, we adapt partial order planning for algorithm selection. For planning purposes only, a helper "finish algorithm" A_ϕ is initially added to $\mathbf{A}_\phi$. Also, a set of input requirements $\mathbf{\Lambda}$ (the "agenda") is derived from $\mathbf{C}_\phi \backslash \mathbf{C}_0$ (lines 1–3).

[2] In practice, $\mathbf{C}_0$ will often be the empty set, meaning that an analysis starts on plain text. Accordingly, a non-empty set $\mathbf{C}_0$ indicates texts that already have annotations.

Each requirement specifies an information type, e.g. *PositiveRegulation*, and an algorithm that needs this type as input. While $\mathbf{\Lambda}$ is not empty, lines 4–15 insert algorithms into $\mathbf{A}_\phi$ and update both π_ϕ and $\mathbf{\Lambda}$. In particular, line 5 retrieves an input requirement $\langle C_{\mathbf{\Lambda}}, A_{\mathbf{\Lambda}}\rangle$ with the deterministic method *poll()*. If $C_{\mathbf{\Lambda}}$ belongs to $\mathbf{C}_\phi$, a filter A_F is integrated on-the-fly, whereas $\langle C_{\mathbf{\Lambda}}, A_{\mathbf{\Lambda}}\rangle$ is replaced with the input requirement of A_F (lines 6–10). Then, line 11 selects an algorithm A that produces $C_{\mathbf{\Lambda}}$. If any input requirement cannot be fulfilled, planning fails (line 12) and does not reach line 16 to return a partially ordered pipeline $\langle \mathbf{A}_\phi, \pi_\phi\rangle$.

For space reasons, we only sketch selectBestAlgorithm: First, the set $\mathbf{A}_{\mathbf{\Lambda}}$ of algorithms that produce $C_{\mathbf{\Lambda}}$ is determined. These algorithms are compared iteratively for each quality criterion Q of the prioritization P_ϕ. If Q has no aggregate function, $\mathbf{A}_{\mathbf{\Lambda}}$ is reduced to the algorithms with the best estimation for Q. Else, the estimations of possible predecessor algorithms of $\mathbf{A}_{\mathbf{\Lambda}}$ are aggregated. In the worst case, this requires to recursively create plans for all input requirements of $\mathbf{A}_{\mathbf{\Lambda}}$. However, predecessors are stopped taken into account as soon as a filter is encountered: filtering changes the input to be processed, hence it does not make sense to aggregate estimations of algorithms before and after filtering. In case, only one algorithm remains for any $Q \in P_\phi$, it constitutes the single best algorithm. Otherwise, any best algorithm is returned.

The filtering view conveys a benefit of partial order planning: As we discussed in [21], *early filtering* of information and *lazy evaluation* improve the efficiency of pipelines while leaving their effectiveness unaffected. Since our planner proceeds backwards, the constraints in π_ϕ prescribe only to execute algorithms right before needed, which implies lazy evaluation. Also, π_ϕ allows to execute a filter directly after its respective annotation algorithm, thereby enabling early filtering.

We defined an information need as one set $\mathbf{C}_\phi$, but many tasks address $k > 1$ needs concurrently. Aside from *PositiveRegulation*, for instance, GENIA faced eight other event types, e.g. *Binding* [13]. The principle generalization for k problems $\phi_1, \ldots, \phi_k$ is simple: We apply partial order planning to each ϕ_i, which results in k partially ordered pipelines $\langle \mathbf{A}_{\phi_1}, \pi_{\phi_1}\rangle, \ldots, \langle \mathbf{A}_{\phi_k}, \pi_{\phi_k}\rangle$. Then, we unify all these pipelines as $\langle \mathbf{A}_\phi, \pi_\phi\rangle = \langle \bigcup_{i=1}^{k} \mathbf{A}_{\phi_i}, \bigcup_{i=1}^{k} \pi_{\phi_i}\rangle$. However, attention must be paid to filters, e.g. a text unit without positive regulations still may yield a binding event. To handle such cases, a set of relevant text units should be maintained independently for each ϕ_i, which is beyond the scope of this paper. Below, we assume that an according maintenance system is given.

4.2 Scheduling with Informed Best-First Search

Informed search aims at efficiently finding solutions by exploiting problem-specific knowledge [18]. During search, a directed acyclic graph is generated stepwise, in which nodes correspond to partial solutions and edges to solving subproblems. For scheduling the set of algorithms $\mathbf{A}_\phi$, we let a node with depth d in the graph denote a pipeline $\langle \mathbf{A}, \pi\rangle$ with d algorithms. The graph's root node is the empty pipeline, and each leaf a pipeline $\langle \mathbf{A}_\phi, \pi\rangle$ with a correct schedule π. An edge represents the execution of an applicable *filter stage*. Here, a filter stage $\langle \mathbf{A}_F, \pi_F\rangle$ is a pipeline where $\mathbf{A}_F$ consists of a filter A_F and all algorithms

```
Algorithm greedyBestFirstScheduling($\mathbf{A}_\phi$, $\pi_\phi$)
 1: Algorithm set $\mathbf{A} \leftarrow \emptyset$
 2: Schedule $\pi \leftarrow \emptyset$
 3: while $\mathbf{A} \neq \mathbf{A}_\phi$ do
 4:     Filter stages $\mathbf{\Pi} \leftarrow \emptyset$
 5:     for each Filter $A_F \in \{A \in \mathbf{A}_\phi \backslash \mathbf{A} \mid A$ is a filter$\}$ do
 6:         Algorithm set $\mathbf{A}_F \leftarrow \{A_F\} \cup$ getAllPredecessors($\mathbf{A}_\phi \backslash \mathbf{A}, \pi_\phi, A_F$)
 7:         Schedule $\pi_F \leftarrow$ getAnyTotalOrdering($\mathbf{A}_F, \pi_\phi$)
 8:         Estimated cost $h[\langle \mathbf{A}_F, \pi_F \rangle] \leftarrow$ getAggregateEstimation($\mathbf{A}_F$)
 9:         $\mathbf{\Pi} \leftarrow \mathbf{\Pi} \cup \{\langle \mathbf{A}_F, \pi_F \rangle\}$
10:     Filter stage $\langle \mathbf{A}_t, \pi_t \rangle \leftarrow \underset{\langle \mathbf{A}_F, \pi_F \rangle \in \mathbf{\Pi}}{\arg\min}\ h[\langle \mathbf{A}_F, \pi_F \rangle]$
11:     $\pi \leftarrow \pi \cup \pi_t \cup \{(A < A_t) \mid A \in \mathbf{A} \wedge A_t \in \mathbf{A}_t\}$
12:     $\mathbf{A} \leftarrow \mathbf{A} \cup \mathbf{A}_t$
13: return $\langle \mathbf{A}_\phi, \pi \rangle$
```

Fig. 4. Pseudocode of greedy best-first search for scheduling the filter stages $\langle \mathbf{A}_t, \pi_t \rangle$ of a partially ordered pipeline $\langle \mathbf{A}_\phi, \pi_\phi \rangle$ according to increasing estimated run-time

in $\mathbf{A}_\phi \backslash \mathbf{A}$ that precede A_F within π_ϕ. $\langle \mathbf{A}_F, \pi_F \rangle$ is *applicable* at node $\langle \mathbf{A}, \pi \rangle$ if for all ordering constraints $(A' < A) \in \pi_\phi$ with $A \in \mathbf{A}_F$, we have $A' \in \mathbf{A}$. Given A_F is scheduled last, all schedules of the algorithms in a filter stage entail the same run-time. Thus, it suffices to schedule all filter stages instead of all algorithms.

To efficiently find solutions, a common informed search strategy, called "best-first search", is to generate successor nodes of the node with the lowest *estimated solution cost* first. For this purpose, a *heuristic function h* provides an estimated cost of a path from a node to a leaf. The widely used best-first approach A* then obtains the estimated solution cost of a path through a node by aggregating the cost of reaching the node with the value of h. If h is optimistic (i.e., h never overestimates costs), the first solution found by A* is optimal [18].

Now, let $R(\langle \mathbf{A}, \pi \rangle)$ be the units of an input text filtered by a pipeline $\langle \mathbf{A}, \pi \rangle$. Further, let $t(\langle \mathbf{A}_F, \pi_F \rangle)$ be the estimation of the aggregate run-time per text unit of each filter stage $\langle \mathbf{A}_F, \pi_F \rangle$, and $\mathbf{\Pi}$ the set of all applicable filter stages at node $\langle \mathbf{A}, \pi \rangle$.[3] Then we estimate the costs of reaching a leaf from $\langle \mathbf{A}, \pi \rangle$ as:

$$h(\langle \mathbf{A}, \pi \rangle) \;=\; |R(\langle \mathbf{A}, \pi \rangle)| \;\cdot\; \min \big\{ t(\langle \mathbf{A}_F, \pi_F \rangle) \,|\, \langle \mathbf{A}_F, \pi_F \rangle \;\in\; \mathbf{\Pi} \big\}$$

In case $t(\langle \mathbf{A}_F, \pi_F \rangle)$ is optimistic, $h(\langle \mathbf{A}, \pi \rangle)$ is also optimistic, since at least one stage must process $R(\langle \mathbf{A}, \pi \rangle)$. In large-scale scenarios, A* can use h to find an efficient schedule on a sample of texts. To compute $R(\langle \mathbf{A}, \pi \rangle)$, however, the filtered text units must be processed by each successor of the current best node. For ad-hoc scenarios, A* thus imposes much computational overhead, as the information contained in the sample may already suffice to directly return first results.

Instead, we propose greedy best-first search, using the algorithms' estimated run-times only. As sketched in Figure 2, we always apply the filter stage $\langle \mathbf{A}_t, \pi_t \rangle$ with lowest $t(\langle \mathbf{A}_t, \pi_t \rangle)$ first. Since no text unit is taken into account, scheduling can be performed without a sample of texts. Figure 4 shows the greedy search

[3] Here, we assume that run-time estimations of all algorithms in $\mathbf{A}_\phi$ are given. For algorithms without run-time estimations, at least default values can be used.

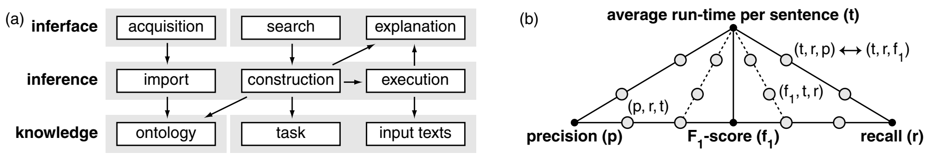

Fig. 5. (a) Architecture of the expert system. (b) Visualization of the system's quality model. The grey partially labeled circles represent the set of quality prioritizations $\mathbf{P}_\Omega$.

for $\langle \mathbf{A}_\phi, \pi \rangle$. Lines 3–12 subsequently add filter stages to $\langle \mathbf{A}, \pi \rangle$. Here, the set $\mathbf{\Pi}$ is built with one stage for each filter A_F in $\mathbf{A}_\phi \backslash \mathbf{A}$. Line 6 identifies all remaining algorithms that must precede A_F, and lines 7–8 compute an ordering and a run-time estimation of the filter stage. Then, line 10 determines $\langle \mathbf{A}_t, \pi_t \rangle$. Before $\mathbf{A}$ and $\mathbf{A}_t$ are merged in line 12, π_t is inserted into π as well as additional ordering constraints to schedule the algorithms in $\mathbf{A}_t$ after those in $\mathbf{A}$ (line 11).

greedyBestFirstScheduling may fail when rather slow filter stages filter much less text. However, real-time applications will often not allow to preprocess a statistically significant sample of texts. Moreover, the first applied stage always processes the whole input, which makes a greedy strategy seem reasonable.

5 An Expert System for Real-Time Annotation

We implemented our approach as an expert system on top of Apache UIMA in order to evaluate whether it qualifies for real-time annotation. The system can be accessed at http://www.CICLing.org/2013/data/122 together with its Java source code, usage instructions, and the algorithms used below.

Apache UIMA is a software framework for applications that annotate natural language text [2]. It allows to compose *primitive analysis engines* (say, annotation algorithms) as *aggregate analysis engines* (say, pipelines). To this end, analysis engines are accompanied by a descriptor file with metadata, such as its input and output annotation types. The respective type system itself is also specified in a descriptor file. These files are all that we need for pipeline construction.[4]

Figure 5(a) shows the architecture of our expert system. All permanent knowledge is stored in an OWL *ontology* according to the model from Section 3. Via an *acquisition* module, the automatic *import* of an annotation task ontology $\langle \mathbf{C}_\Omega, \mathbf{P}_\Omega, \mathbf{A}_\Omega \rangle$ from a set of descriptor files can be triggered, except for $\mathbf{P}_\Omega$: For convenience, we built in the quality model in Figure 5(b) and defined additional "implies" relations between prioritizations. In this manner, the expert system can compare algorithms whose effectiveness is e.g. given as accuracy, when e.g. F_1-score is prioritized. The system's prototypical *search* interface enables users to specify a collection of *input texts* as well as a *task* consisting of a prioritization, a set of information types, and different kinds of value constraints. The system then starts the ontology-based ad-hoc *construction* and the *execution* of an aggregate analysis engine. Analysis results are presented by an *explanation* module.

[4] By default, quality estimations are not specified in descriptor files. We integrated them in the files' informal description field via a fixed notation, e.g. "@Recall 0.7".

Table 1. The time in ms for algorithm selection ($t_{\mathbf{A}}$), scheduling (t_π), and automatic pipeline construction in total (t_{apc}) as well as the number of employed algorithms $|\mathbf{A}|$ for each $\mathbf{C}_\phi$ and the prioritization (p, r, t) based on the 38 / 76 algorithms of $\mathbf{A}_{\Omega_1}$ / $\mathbf{A}_{\Omega_2}$.

Information need $\mathbf{C}_\phi$	$t_{\mathbf{A}}$	t_π	t_{apc}	$\lvert\mathbf{A}\rvert$
PositiveRegulation(_, _, _, _)	5.0 / 5.4	1.0 / 1.4	20.2 / 22.9	6
PositiveRegulation(_, "Eo-VP16", _, _)	5.6 / 12.1	2.9 / 3.3	26.5 / 37.7	13
PositiveReg.(Theme, "Eo-VP16", Site, CSite)	14.6 / 17.9	5.0 / 5.7	39.5 / 46.9	20

Table 2. Properties, run-time t in seconds averaged over 10 runs, precision p, and recall r of the pipelines constructed for the information need *StatementOnRevenue(Time, Money)* and each quality prioritization P on the test set of the Revenue corpus [19].

Priorization P	**Properties of constructed pipeline**	t	p	r
$(t, p, r), (t, r, p)$	fully rule-based; fast preprocessing	**3.5**	0.6	0.48
(r, t, p)	mostly rule-based; exact preprocessing	21.3	0.67	0.58
$(p, r, t), (p, t, r), (r, p, t)$	mostly statistical; exact preprocessing	125.4	**0.76**	**0.66**

5.1 Experimental Analysis of Automatic Pipeline Construction

We experimented with two ontologies $\langle \mathbf{C}_\Omega, \mathbf{P}_\Omega, \mathbf{A}_{\Omega_1} \rangle$ and $\langle \mathbf{C}_\Omega, \mathbf{P}_\Omega, \mathbf{A}_{\Omega_2} \rangle$ based on $\mathbf{P}_\Omega$ and a set $\mathbf{C}_\Omega$ that refers to 40 concrete annotation types of GENIA [13] and the statement on revenue task [19]. $\mathbf{A}_{\Omega_2}$ consists of 76 preprocessing and extraction algorithms, while $\mathbf{A}_{\Omega_1}$ contains only half of them. Up to three algorithms exist for an information type in both cases. All experiments were conducted on a 2 GHz Intel Core 2 Duo MacBook with 4 GB memory.

We evaluated the expert system for information needs of different complexity. In particular, we measured the pipeline construction time for both ontologies and three needs related to *PositiveRegulation*, as detailed in Table 1.[5] Irrespective of the underlying ontology, algorithm selection and scheduling take only a couple of milliseconds in all cases. The remaining part of t_{apc} refers to operations such as creating descriptor files. Altogether, the measured run-times seem to grow linear in the number of employed algorithms and even sublinear in the number of available algorithms. Hence, we argue that our approach is suitable for real-time annotation. In contrast, manual construction would take at least minutes.

In a second experiment with $\mathbf{A}_{\Omega_2}$, we ran the expert system for the information need *StatementOnRevenue(Time, Money)* and each possible prioritization of *run-time*, *precision*, and *recall.* This resulted in the three different pipelines listed in Table 2, which we then executed on the test set of the *Revenue corpus* [19]. The pipeline for (t, p, r) and (t, r, p) relies only on fast rule-based algorithms. As expected, this pipeline was one to two orders of magnitude faster than the other ones, while achieving much less precision and recall. The low recall of 0.58 under (r, t, p) may seem surprising. However, it indicates the restricted feasibility of predicting quality in annotation tasks: favoring high quality algorithms does not

[5] We averaged the run-times $t_{\mathbf{A}}$, t_π, and t_{apc} over 25 runs, as their standard deviations were partly over half as high as the run-times themselves due to I/O operations.

guarantee a high overall quality, since the latter is also influenced by the interactions of the algorithms. In the end, high quality can never be ensured, though, as it depends on the domain of application and the processed input texts.

Finally, our realization revealed additional challenges of automatic pipeline construction, which we summarize under three distinct aspects:

Joint Annotation. Algorithms that produce more than one information type can compromise a quality prioritization. For instance, a constructed pipeline may schedule a tagger A_t before a chunker A_{tc} that also performs tagging but less accurate. In this case, the tags of A_t are overwritten by A_{tc}. On the contrary, our expert system recognizes "dominating" algorithms. E.g., if A_{tc} precedes A_t and efficiency is of upmost priority, then A_t is omitted. Still, automatic pipeline construction benefits from a maximum decomposition of the analysis steps.

Inheritance. By concept, information types can inherit features from supertypes. In an information need *PositiveRegulation(Theme, _, _, _)*, for instance, *Theme* might be inherited from a general type *Event*. To handle such cases, we normalize the need into *PositiveRegulation* $\wedge$ *Event(Theme)*, while ensuring that only positive regulation events are kept. However, *Theme* also exemplifies a more complex problem: In GENIA, different event types can be themes of regulations. But, for scheduling, it suffices to detect one event type before theme extraction, so the expert system does not select further algorithms. A solution would be to require all subtypes for types like *Event*, but this is left to future work.

Limitations of Automation. Some limitations result from performing pipeline construction fully automatically. E.g., our approach does not allow to force certain algorithms to be employed (such as a sentence splitter tuned for biomedical texts), except for cases that can be realized based on quality criteria. Also, algorithms that target at a middle ground between efficiency and effectiveness tend not to be chosen because of the prioritization concept. Similarly, it is not possible to prioritize efficiency in one stage (say, preprocessing) and effectiveness in another (say, relation extraction). While such possibilities could be integrated in our approach, they require more user interaction and thereby reflect the inherent trade-off of pipeline construction between automation and manual tuning.

6 Conclusion

Annotation tasks like information extraction are often tackled with pipelines of algorithms manually constructed by experts. In contrast, we provide an artificial intelligence approach to automatically construct pipelines, which we realized as an open-source expert system. Experiments with this system suggest that our approach renders it possible to efficiently and effectively tackle ad-hoc annotation tasks in real-time applications. Currently, the approach relies on abstract expert knowledge and techniques like relevance filtering. However, it uses *general* efficiency and effectiveness estimations only. In the future, we will investigate how to exploit samples of input texts and knowledge about the domain of application in order to optimize pipeline construction. Also, we will work on a relevance maintenance system to filter information for different tasks at the same time.

Acknowledgments. This work was partly funded by the German Federal Ministry of Education and Research (BMBF) under contract number 01IS11016A.

References

1. Agichtein, E.: Scaling Information Extraction to Large Document Collections. Bulletin of the IEEE Computer Society TCDE 28, 3–10 (2005)
2. Apache UIMA, `http://uima.apache.org`
3. Bangalore, S.: Thinking Outside the Box for Natural Language Processing. In: Gelbukh, A. (ed.) CICLing 2012, Part I. LNCS, vol. 7181, pp. 1–16. Springer, Heidelberg (2012)
4. Chiticariu, L., Krishnamurthy, R., Li, Y., Raghavan, S., Reiss, F.R., Vaithyanathan, S.: SystemT: An Algebraic Approach to Declarative Information Extraction. In: Proc. of the 48th ACL, pp. 128–137 (2010)
5. Das Sarma, A., Jain, A., Bohannon, P.: Building a Generic Debugger for Information Extraction Pipelines. In: Proc. of the 20th CIKM, pp. 2229–2232 (2011)
6. Dezsényi, C., Dobrowiecki, T.P., Mészáros, T.: Adaptive Document Analysis with Planning. In: Pěchouček, M., Petta, P., Varga, L.Z. (eds.) CEEMAS 2005. LNCS (LNAI), vol. 3690, pp. 620–623. Springer, Heidelberg (2005)
7. Etzioni, O.: Search Needs a Shake-up. Nature 476, 25–26 (2011)
8. Fader, A., Soderland, S., Etzioni, O.: Identifying Relations for Open Information Extraction. In: Proc. of the EMNLP, pp. 1535–1545 (2011)
9. Fox, M.S., Smith, S.F.: ISIS: A Knowledge-based System for Factory Scheduling. Expert Systems 1, 25–49 (1984)
10. GATE, `http://gate.ac.uk`
11. Kano, Y.: Kachako: Towards a Data-centric Platform for Full Automation of Service Selection, Composition, Scalable Deployment and Evaluation. In: Proc. of the 19th IEEE ICWS, pp. 642–643 (2012)
12. Kano, Y., Dorado, R., McCrohon, L., Ananiadou, S., Tsujii, J.: U-Compare: An Integrated Language Resource Evaluation Platform Including a Comprehensive UIMA Resource Library. In: Proc. of the Seventh LREC, pp. 428–434 (2010)
13. Kim, J.D., Wang, Y., Takagi, T., Yonezawa, A.: Overview of Genia Event Task in BioNLP Shared Task 2011. In: BioNLP Shared Task Workshop, pp. 7–15 (2011)
14. Marler, R.T., Arora, J.S.: Survey of Multi-Objective Optimization Methods for Engineering. Structural and Multidisciplinary Optimization 26(6), 369–395 (2004)
15. Pasca, M.: Web-based Open-Domain Information Extraction. In: Proc. of the 20th CIKM, pp. 2605–2606 (2011)
16. Pauls, A., Klein, D.: k-best A^* Parsing. In: Proc. of the Joint Conference of the 47th ACL and the 4th IJCNLP, pp. 958–966 (2009)
17. Riabov, A., Liu, Z.: Scalable Planning for Distributed Stream Processing Systems. In: Proc. of the 16th ICAPS, pp. 31–41 (2006)
18. Russell, S.J., Norvig, P.: Artificial Intelligence: A Modern Approach, 3rd edn. Prentice-Hall (2009)
19. Wachsmuth, H., Prettenhofer, P., Stein, B.: Efficient Statement Identification for Automatic Market Forecasting. In: Proc. of the 23rd COLING, pp. 1128–1136 (2010)
20. Wachsmuth, H., Stein, B.: Optimal Scheduling of Information Extraction Algorithms. In: Proc. of the 24th COLING: Posters, pp. 1281–1290 (2012)
21. Wachsmuth, H., Stein, B., Engels, G.: Constructing Efficient Information Extraction Pipelines. In: Proc. of the 20th CIKM, pp. 2237–2240 (2011)
22. Ž áková, M., Křemen, P., Železný, F., Lavrač, N.: Automating Knowledge Discovery Workflow Composition through Ontology-based Planning. IEEE Transactions on Automation Science and Engineering 8(2), 253–264 (2011)