Metaheuristics for Ontology-Based Information Extraction Rule Learning

Michel Capelle, Flavius Frasincar* and Finn van der Knaap

Erasmus School of Economics, Erasmus University Rotterdam, 3062 PA Rotterdam, the Netherlands
E-mail: 294357mc@eur.nl; frasincar@ese.eur.nl; 573834fk@eur.nl
**Corresponding Author*

Received 21 January 2026; Accepted 04 March 2026

Abstract

The Semantic Web aims to make information intelligible for computers. In the Semantic Web, unstructured information from text is represented using ontologies, such that computers can understand text better. However, adding text information to existing ontologies by hand is time-consuming. Information extraction rules can help to automate this process. In the process of learning information extraction rules, patterns are constructed that consist of lexico-syntactic and lexico-semantic features from text, which aim to extract Resource Description Framework subject-predicate-object expressions. In this paper, we investigate the following four metaheuristics for learning ontology-based information extraction rules: Particle Swarm Optimization, 2-Phase Optimization, Ant Colony Optimization, and Genetic Algorithm (GA). We evaluate all methods using financial news data. GA gives the best F_1-measure results, but the other metaheuristics are faster.

Keywords: Metaheuristics, information extraction rules, ontology learning, Semantic Web.

Journal of Web Engineering, Vol. 25_4, 699–736.
doi: 10.13052/jwe1540-9589.2549

1 Introduction

Due to the increasing popularity of the World Wide Web in the last three decades, a huge amount of information has become available. Currently, the Web has approximately 4.47 billion indexed web pages [32]. Most of these Web pages contain unstructured information, which is information that a computer cannot understand. Examples of Web pages with unstructured information are news portals and community forums, which contain text that can be easily interpreted by humans, but which are far more challenging for computers to comprehend.

In order to store unstructured information in a structured manner, such that it is understandable to computers, one of the founders of the World Wide Web, Tim Berners-Lee, invented the Semantic Web [6]. The Semantic Web is an extension of the World Wide Web that offers technologies for specifying and handling structured data. The structured data can be directly available or can be extracted from unstructured data.

In the Semantic Web, information is stored in ontologies, in which domain-specific concepts and concept relations are specified. Ontology languages are employed to construct ontologies and to support ontologies with processing information. One ontology representation language is the Resource Description Framework (RDF) [31]. RDF consists of triples, which are subject-predicate-object expressions that represent an event or a fact in a text. The sentence 'Google has CEO Sundar Pichai' is such an example, in which 'Google' is the subject, 'has CEO' is the predicate, and 'Sundar Pichai' is the object. So, the object (Sundar Pichai) gives information (is CEO of) about the subject (Google).

The more descriptions of concepts and concept relations one specifies in an ontology, the better a computer is at extracting useful information from text. For a human, it is time-consuming to add information to an ontology by hand. Therefore, information extraction rules are employed to perform this task automatically [40]. Information extraction rules are constructed using an information extraction language, which combines several lexico-syntactic and lexico-semantic features in text, such as sequences, syntactic categories, logical operators, wildcards, ontological instances, and ontological classes.

Companies that employ learning information extraction rules are capable of extracting rich information from text. This valuable information could lead to a competitive advantage over their competitors. Consider a broker on the stock market. If the broker is the first to obtain information from a news item about a takeover of a company of interest to him or her, (s)he could act accordingly on the market before competitors know about the takeover.

Therefore, it is crucial to obtain information that is accurate and valuable to the broker and to extract this information in a timely manner.

In order to learn information extraction rules in an ontology-based environment, several approaches that employ machine learning have been proposed [4, 9, 12, 15, 16, 46]. Besides machine learning approaches, metaheuristics are also popular for learning information extraction rules. We do not have the computational power to extensively search through the entire search space. For the task at hand, metaheuristics are more likely to find feasible solutions in less computation time than machine learning approaches [7].

In this work, we investigate the applicability of metaheuristic approaches to the problem of ontology population and evaluate them in terms of the speed and quality of the obtained rules. We consider the following metaheuristics: Particle Swarm Optimization (PSO), 2-Phase Optimization (2PO), Ant Colony Optimization (ACO), and Genetic Algorithm (GA). We employ learning extraction rules on news articles gathered from a personalized news portal [18]. Furthermore, the focus is on learning Hermes Information Extraction Language (HIEL) [27] rules, which is the lexico-semantic information extraction language in the Hermes News Portal [18]. The Hermes News Portal is the news personalization framework we use. Hermes is written in Java and it combines several Semantic Web technologies, such as HIEL [27], Web Ontology Language (OWL) [5], and SPARQL [37]. Furthermore, Hermes contains a rule engine, in which one can construct and apply information extraction rules.

The dataset used for evaluation contains 500 news articles with an average of 700 words. News articles from the financial domain are extracted from Reuters Business, Technology RSS feed, and the New York Times Business RSS feed. The news items were annotated with events by experts in the financial field. Furthermore, experts constructed information extraction rules using a domain ontology. The domain ontology consists of 1286 concepts in the financial domain, such as companies, persons, countries, and products. Each concept has associated lexical representations that represent synonyms.

The rest of the paper is structured as follows. First, Section 2 discusses previous work related to machine learning and metaheuristic approaches. Then, Section 3 presents the framework in which the information extraction rules are constructed. Next, Section 4 gives an overview of the proposed metaheuristics followed by, in Section 5, the evaluation of the aforementioned metaheuristics. Last, Section 6 presents our concluding remarks and suggestions for future work.

2 Related Work

This section presents an overview of previous work. We first discuss previously proposed machine learning approaches on ontology-based information extraction in Section 2.1. Then, Section 2.2 provides general descriptions of the considered metaheuristics and explains the benefit of using metaheuristics for the task at hand.

2.1 Machine Learning Approaches

The WebKB system [12] is a Web information extraction system that uses a domain ontology and hand-labeled training data, containing example instances of the domain ontology. It employs a statistical bag-of-words approach and a first-order rule learning approach. In the bag-of-words approach, a probabilistic model is built, in which a Naïve Bayes classifier determines the ontology class. The first-order rule learning approach takes into account whether a Web page contains a given word and hyperlinks to other pages. It employs the First-Order Inductive Learner (FOIL) algorithm for learning first-order logic information extraction rules. The WebKB system reports accuracies of 0.70, 0.80, and 0.97 on the three considered relations.

In contrast to WebKB, ExDisco [46] proposes an unsupervised learning approach. However, a set of seed patterns must be provided by the user, which should cover only the most trivial patterns. ExDisco then iteratively extends these seed patterns with new patterns by constructing a set of candidate patterns and adding the candidate that appears most often in the relevant documents compared to non-relevant documents. ExDisco reports 0.60 on F_1-measure.

Robust Automated Production of Information Extraction Rules (RAPIER) [9] employs a supervised relational learning algorithm by using techniques from Inductive Logic Programming (ILP) systems, which induce rules of first-order logic. RAPIER's rule representation has three parts: a slot-filler pattern (i.e, a pattern that is matched positively to the text), a pre-filler pattern (i.e., a pattern that matches positively preceding the slot-filler pattern), and a post-filler pattern (i.e., a pattern that matches positively following the slot-filler pattern). A pattern can have three types of constraints: on the word, on the part-of-speech, or on the semantic class. Each pattern is a sequence of pattern elements, which are either pattern items or a pattern list. Rules are evaluated using a heuristic measure that takes into account the number of covered positive and negative examples and rule complexity. The proposed RAPIER algorithm is a bottom-up approach, which means that first specific

rules are constructed, and then general rules are derived from the specific rules. RAPIER reports on four relations a precision score ranging from 0.80 to 0.95 and a recall score ranging from 0.39 to 0.94.

KnowItAll [15, 16] populates an ontology with facts and relations extracted from Web sites. Its rule learning algorithm starts with a seed set of rules which is generated with bootstrapping. For each seed rule, a search engine query is performed and the returned Web pages are extracted. Context strings (substrings containing surrounding words of the query) are then converted into information extraction rules using automatically generated discriminative phrases. Using the extracted page counts of the constructed queries, it computes the Pointwise Mutual Information (PMI) between a fact and a discriminative phrase. Bootstrapping is employed to train the conditional probabilities that are combined with the computed PMI scores in a Naïve Bayes classifier. The classifier distinguishes between correct and incorrect rules. KnowItAll reports a precision score ranging from 0.59 to 0.80 over nine facts.

Simple Learner with Iterative Pruning to Produce Error Reduction (SLIPPER) [11] employs a boosting algorithm to create an ensemble of rules. SLIPPER generates rules by boosting a greedy rule-builder, in which a weak learner learns a single rule by first growing the rule using elements from the rules in a subset of the training data, and then pruning the rule using the elements of the rules in another subset. A confidence metric is assigned to each of the ensemble of rules. To check whether a rule is informative for a given instance, SLIPPER sums the confidence for each rule covering the instance. SLIPPER reports an error rate of 0.17 on test data.

A supervised information extraction rule learning approach is proposed by [2]. FOIL Learner is an ILP technique that is employed to learn rules for adding new information to ontologies. Similar to the SLIPPER learning approach, FOIL Learner first grows rules and then prunes them. Whether a rule is extended or pruned any further depends on its evaluation on the training data. [2] uses an information-gain heuristic, which aims to extract as many as possible positive labeled cases. FOIL Learner reports F_1-measures of 0.19, 0.68, and 0.41 on three relations.

A simple semi-automated pattern extraction approach for learning WordNet relations is proposed in the works of [23] and [24]. The Lexico-Syntactic Pattern Extraction (LSPE) method is applied to lexical relations but can be easily adapted to use domain ontologies. LSPE first selects a WordNet relation, and then the user picks word pairs for which the relation holds. Then, sentences from a text corpus are selected in which both words appear, and

the lexical and syntactical context is stored. Last, this context is used to find commonalities for which the relation holds. The first two steps can be used as input for a machine learning classifier.

[19] proposes another ILP approach, called SRV. A training set is given, containing positive and negative examples that are represented as text fragments. Starting with an empty rule, SRV greedily extends it with new literals, covering as many positive examples from the training set as possible. The algorithm stops extending a rule if all positive examples are covered or if no literals exist for which the information-gain metric is positive. SRV uses the same information-gain metric as FOIL. The difference with FOIL is that SRV only grows rules and does not prune rules, which makes it faster but less accurate. SRV reports F_1-measures ranging from 0.58 to 0.98 on seminar announcements, from 0.33 to 0.87 on Web pages, and from 0.21 to 0.60 on Newswire articles.

In the ILP system KLUSTER [29], a set of relation examples is given as input. Using these input relations, a basic taxonomy is built. This taxonomy is then used to determine Mutually Disjoint Concepts (MDCs), which are concepts that have the same superconcept and are mutually disjoint. For all MDCs, KLUSTER builds all Most Specific Generalizations (MSGs) for all examples. If a generalization is not discriminative enough (i.e., the number of misclassified examples is not lower or equal than a given threshold), more terms are added to the generalization. When all MSGs are built for all MDCs, then for each MDC, the most general discriminative rule is selected from its MSGs.

Another ILP approach is proposed in the work of [21]. From a parse tree and its conceptual interpretation, concept hypotheses about a text item are derived. The extracted linguistic and conceptual information about the text item is used to select good concept hypotheses. The goodness of the remaining concept hypotheses is measured using linguistic and conceptual quality labels. With these quality labels, a credibility score is estimated using a classifier with an evaluation measure. The concept hypotheses are ranked by credibility score and the ones with a score that exceeds a given threshold are added to the knowledge base. This ILP approach reports an average accuracy of 0.53.

In Text-to-Onto [34], a learning algorithm is proposed that takes into account the extracted associations between a set of concepts. The input for the algorithm is a set of transactions, where each transaction or relation consists of a pair of items, each item representing a set of concepts from a domain ontology. The learning algorithm then computes association rules

and selects those that exceed given thresholds for the measures support and confidence. In order to choose the right level of abstraction, the algorithm prunes association rules that have an ancestor rule (i.e., an association rule that contains one of the concepts' ontological ancestors) with equal or higher support and confidence. Text-to-Onto reports 0.67 on rule learning accuracy. [10] propose the successor of Text-to-Onto, called Text2Onto, that uses a probabilistic ontology model. Compared to Text-to-Onto, it increases efficiency and also allows a user to follow the development of the ontology with respect to changes in the corpus. Several extensions have been proposed for Text2Onto [22, 35].

The WHISK [40] algorithm is a supervised learning approach that starts with a seed instance from a training set. From this seed instance, WHISK begins with an empty rule that covers all positive examples. Then, it grows the rule, first on terms from the seed instance, then on terms that are not included in the seed instance. The rules are initially grown to the number of slots the seed instance has. Then, the rules are extended with terms from the seed instance. The Laplacian expected error [43] estimates how many positive examples are covered by the rule. The best extension to the rule with respect to the Laplacian expected error is then selected. The rule is extended until the rule makes no more errors or the Laplacian expected error is below a given threshold. WHISK reports 0.48 precision and 0.61 recall on the management succession domain.

[42] propose Crystal, which is a bottom-up induction rule learning algorithm. A definition (i.e., a concept and its extraction pattern constraints) is constructed for each positive example from the training data for each relation. Since the definition's extraction pattern constraints are too specific, Crystal generalizes the constraints by converting two definitions that are similar into one general definition. Similarity is measured by counting the relaxations that are required to match two definitions. The common properties of these two definitions form the properties of the general definition.

2.2 Metaheuristics

PSO [28] has its roots in genetic algorithms and evolutionary programming. It is based on the social metaphor of physical movements in a bird flock. It embraces the fact that birds return stochastically towards previously discovered successful regions. In the PSO algorithm, particles (i.e., candidate solutions) are iteratively moved in the search space depending on the particles' position and velocity. The movement of the particles is influenced by the

local best solution and the movement of the swarm, which is determined using a fitness function, and is therefore directed to the best solutions in the solution space. At the end of each iteration, the position and velocity of the particles are updated. Since its invention, many variants [30, 36, 39] of PSO have been proposed to improve performance, focusing on for example improving the solution rule update or the exploration ability of the heuristic.

2PO [41] combines the Iterative Improvement (II) and Simulated Annealing (SA) algorithms. II randomly generates a set of initial solutions and iteratively walks through their neighbors in the solution space. II updates the set of solutions when the proposed neighbor leads to an improvement in solution costs. The algorithm results in several local minima, which constitute the input for SA. In the SA algorithm, the neighborhood of each local minimum is discovered. While the temperature of the system drops, the system becomes more stable and is more likely to accept solutions with lower costs. In the context of our problem, when the temperature drops, the probability of mutating an information extraction rule drops as well. SA stops when the temperature of the system has decreased below a certain threshold or if several temperature decrements have not led to an improvement. As SA is dependent on its starting point, a variant of SA that deals with this problem is proposed in the work of [3].

ACO [14] is a computing technique that simulates the natural behavior of ant colonies. Ants leave pheromone traces while walking from their nest to a food source. Ants tend to walk the paths where the pheromone concentration is the highest. Since the shortest paths are then increasingly chosen by ants, they leave more pheromones on this path and this attracts even more ants. Therefore, the preferred route converges to an optimum, which is the shortest path. In our context, a rule is built as an ant walks a path, starting with an empty rule and choosing each new edge with a probability that is proportional to the fitness value of the considered edge. The fitness values of regularly chosen edges increase, similar to regularly walked paths by ants.

GA [25] is based on Darwin's phrase 'survival of the fittest' in his evolution theory. GA tries to imitate the evolutionary process of a set of chromosomes (the population), which in this context is a set of solutions. This set of chromosomes is exposed to evolution: It faces selection, implying that a subset of chromosomes is selected for the next generation, crossovers, meaning that some of the chromosomes are combined, and finally mutations. The evolution (i.e., simulation) finishes when a predefined number of iterations is reached, or when several generations have not resulted in an improvement. The concept of improvement is measured with a fitness

function, which determines its chance of survival. Many variants of GA [25] have been proposed, using for example updated evolutionary processes [33], or hybridizing GA with other optimization methods [20, 38].

An important difference between metaheuristics and machine learning approaches is that metaheuristics are better suitable for optimization problems that deal with computation capacity problems. For the task we perform, we do not have the computational power to extensively search through the entire search space. Therefore, we deal with an optimization problem. Metaheuristics are in this situation more likely to find feasible solutions in less computation time than machine learning approaches [7].

3 Framework

In this paper, we present metaheuristics for constructing ontology-based information extraction rules. These metaheuristics are built in Hermes, which is a framework for news item extraction [8]. In Hermes, the metaheuristics are implemented in its rule learning component. The rule learning component is based on HIEL [27], an information extraction language used by the metaheuristics to define the information extraction rules. The previously mentioned rule learning component is extensively described by [26]. In Section 3.1, we discuss the Hermes framework. Then, Section 3.2 describes the HIEL language followed by, in Section 3.3, a brief discussion of the rule learning component of Hermes.

3.1 Hermes

The Hermes framework [8] was originally built for classifying news items. In order to give stock traders a competitive advantage, news items are classified based on concepts from an ontology, such that traders do not have to read all available news items. A knowledge base keeps track of the information about the domain the reader is interested in. The knowledge base is constructed as a Web ontology instantiation based on the principles of the Semantic Web.

The ontology is represented in OWL. For each concept, synonyms in the form of lexical representations and their associated WordNet senses are stored in the database. WordNet [17] is a semantic lexicon for the English language that contains synsets and synset relations.

Hermes combines a knowledge base with a news item ontology. The news item ontology contains meta-information about news items, such as hyperlinks and sources. The classification process links concepts from the

domain ontology to the news items in which they appear. The classification process also matches the concepts and their lexical representations with the words in the considered news item. Moreover, the classification process matches the concepts' synonyms and hyponyms, which are extracted from WordNet to text. After the classification phase, the news items are evaluated. If enough relations between a news item and the concepts from the knowledge base in which the user is interested are found, the news item is considered as 'interesting' to the user.

3.2 Hermes Information Extraction Language

HIEL [27] is inspired by the patterns that are proposed in the works of [23] and [24]. All rules that are considered in this work are written in HIEL. In HIEL, patterns are represented by a Left-Hand Side (LHS) and a Right-Hand Side (RHS). RHS is matched against the news items and, once matched, the text is annotated with LHS. LHS consists of a subject, an object, and a predicate, in which the predicate describes the relation between the subject and the object. For example, 'Company hasCEO Person' can be employed to extract that the company Google has a CEO named Sundar Pichai from a news item, depending on the pattern that is defined as RHS.

The RHS pattern is matched against the complete news item and the patterns consist of tokens as literals, lexical categories, orthographic categories, labels, logical operators, repetition, wildcards, and concepts.

The following example rule is generated by the ACO metaheuristic and extracts the event 'Company hasProduct Product':

$subject: NASDAQCompany (!Subsidiary)
+ $object: Product

This rule matches with text in which a NASDAQ company, followed by one or more words (i.e., the '+' sign) that do not (i.e., the '!' sign) represent a subsidiary, followed by the object that represents a product, is stated.

In order to match rules, the Hermes Information Extraction Engine (HIEE) is implemented. HIEE is a pipeline containing a preprocessing stage and a rule engine. Preprocessing components include 'Document Reset', 'ANNIE English tokenizer', 'ANNIE Gazetteer', 'ANNIE Sentence Splitter', 'ANNIE Part-of-Speech Tagger', and 'OntoGazetteer', which handle several Natural Language Processing steps that need to be performed before matching RHS. The rule engine employs pattern compilation and pattern matching.

These components employ semantic concepts from the knowledge base ontology and syntactic elements which are retrieved during the preprocessing stage. We refer to the work of [27] for a detailed description of HIEE.

HIEL is chosen as the information extraction language because other languages, such as the Java Annotations Pattern Engine [13] and SPARQL [37], result in more verbose rules. HIEL results in more compact rules, which reduces the computation time. HIEL rules are thus easier to produce.

3.3 Rule Learning

Hermes has a rule learning component [26] in its framework that learns rules in HIEL, visualized as a tab in the user interface. In turn, each metaheuristic discussed in Section 4 has a tab under the rule learning tab in Hermes. Each metaheuristic tab contains tabs for testing the metaheuristic, which includes the tabs 'Population', 'Stratified testing', and 'Parameters'.

Figures 1 and 2 show two screenshots of Hermes. The first row in Figure 1 contains all metaheuristics. In Figure 1, the user can configure the settings for the selected metaheuristic, which in this figure is PSO. Figure 2 shows the stratified testing user interface for the metaheuristic ACO. The first table in the figure shows the rules which are generated by the algorithm. The second table shows the rule group. In each iteration, the best rule from the first table is proposed to the second table. At the top of the screen, the performance measures for the rules in the rule group are shown.

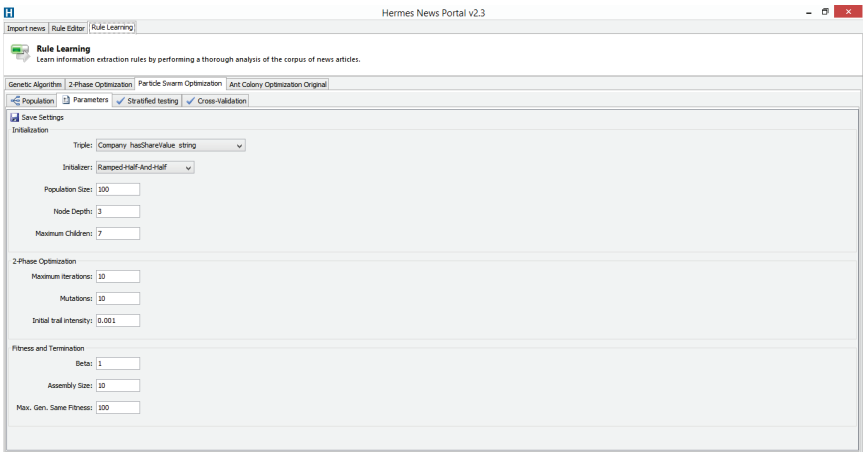

Figure 1 The parameters tab in the rule learning component in Hermes.

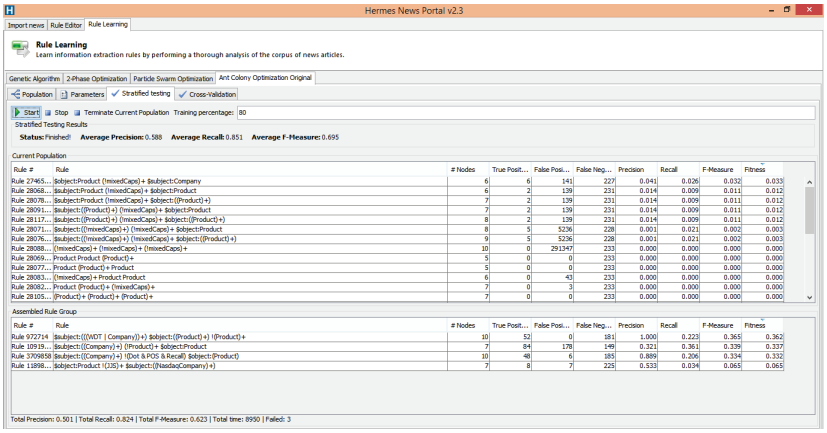

Figure 2 The stratified testing tab in the rule learning component in Hermes.

The rule learning process runs one of the proposed metaheuristics. Once an algorithm has finished, it returns a set of rules from which the rule with the highest fitness score is selected and proposed to be added to the currently developed rule group. The rule group is a set of rules that aim to extract a selected event from news items. The goal of the rule group is to maximize its total F_1-measure. The proposed rule can be rejected by the rule group if the total F_1-measure does not improve when adding the proposed rule to the rule group. The rule learning process stops when the rule group is full (i.e., the predefined maximum number of five rules in the rule group is reached) or when the rule group has subsequently rejected three proposed rules. When the maximum number of rules in the rule group is exceeded or when the rule group accepts proposed rules after rejecting three rules, we found that it is unlikely that the accuracy on the test set increases.

The produced rules are represented as trees, in which nodes with children are operators and leaf nodes are terminals. Operators can be subdivided into the HIEL tokens ‘sequence’, ‘and operator’, ‘or operator’, ‘not operator’, and ‘repetition’. Terminals can be categorized in the HIEL tokens ‘syntactic category’, ‘orthographic category’, ‘concept category’, and ‘wildcard category’. The root of the tree is always a sequence with one or more child nodes.

In all algorithms, a rule first needs to be initialized. We use the Ramped-Half-And-Half (RHAH) initializer [26]. With RHAH, one has to specify the maximum node depth and the maximum number of children. Half of the trees are constructed ‘full’, which means that all leaf nodes are on the maximum node depth. The other half of the trees are constructed ‘grow’, in which not all

leaf nodes are at the maximum node depth. This leads to various shaped trees, such that we explore the search space better. We compared the performance of RHAH with an initializer that constructs all trees 'full', and with an initializer that constructs all trees 'grow'.

4 Metaheuristics

In this section, we explain the proposed metaheuristics informally, as well as in pseudocode. In Section 4.1, we discuss PSO. Then, Section 4.2 presents 2PO, which is a combination of the II and SA algorithms followed by, in Section 4.3, ACO. Last, in Section 4.4, GA is discussed.

4.1 Particle Swarm Optimization

PSO [28] is a computing algorithm that is based on the social behavior of birds in their flock or fish in their school. Particles in a swarm move around in a solution space but tend to return to their best-known position in the swarm. They determine their best position based on their current position and velocity. The best-known positions of the swarm's particles direct the swarm to an optimal location in the solution space.

Our proposed heuristic also considers particles in a swarm. It is comparable to Accelerated PSO [45]. In the original work [28], each particle has a current position, a best-known position, and a velocity. Our contribution to the PSO metaheuristic is to make the algorithm applicable to the ontology population problem. In our algorithm, a particle contains a Current Rule (CR) and a Best-Known Rule (BKR), but no velocity. The 'original' particle takes into account the particle's velocity to determine its best-known position, but 'our' particle calculates fitness scores to determine its best-known rule.

Our algorithm starts with initializing the particles of a swarm. A swarm is initialized with a given number of particles. Each particle receives a CR, which is a rule tree that is constructed by one of the aforementioned tree initializers in Section 3.3. The current rule is the BKR rule at the beginning of the first iteration. After the initialization, the algorithm iterates a given number of iterations. In each iteration, each particle computes three fitness measures: one for the CR, one for the BKR, and one for the best-known rule of all particles in the swarm, which we refer to as the Global Best-Known Rule (GBKR).

Next, in the same iteration, the current rule for each particle is updated depending on the values of the three fitness measures. Suppose the CR has a

fitness that is worse than the fitness of the GBKR, it is not beneficial to update the CR to a mutation of the CR. However, if the fitness of the CR is similar to the fitness of the GBKR, it is useful to update the CR to a mutation of the CR. This also applies to the fitness of the CR compared to the fitness of the BKR, and the fitness of the BKR compared to the fitness of the GBKR. It seems tempting to always mutate the GBKR, but this would direct the swarm to a narrow solution, which does not explore a large diversity of rules.

Therefore, we choose to mutate either the CR, the BKR, or the GBKR, and we add a small Initial Constant (IC) to the fitness measures of the CR, the BKR, and the GBKR, in order to allow the algorithm to give some probabilistic advantage to choose the CR. If the selected rule has a fitness measure of 0, it is useless and we initialize the CR again. Otherwise, the CR is updated to the mutation of the selected rule.

In the mutation process, we randomly pick a node from the rule, and we replace the node with a subtree that is newly generated by the rule initializer. The new rule is added to the mutation set. We also remove the randomly picked node instead of replacing it. Again, this new rule is added to the mutation set. We repeat this replacement and removal action ten times on randomly chosen nodes. At the end of the mutation process, the mutation set contains ten rules retrieved from the replacement action and ten rules retrieved from the removal action. The mutation process returns the highest fitness tree from the twenty rules in the mutation set. We repeat this process ten times, as repetition increases the chance that one of the generated rules outperforms the original rule.

The selection procedure is as follows: we pick a random number between zero and one. If the random number is smaller than the fitness of the CR divided by the fitness of the BKR, we mutate the CR. If this is not the case, but the random number is smaller than the fitness of the BKR divided by the fitness of the GBKR, we mutate the BKR. Otherwise, we mutate the GBKR. Using this structure, we give each mutation type a probability of being chosen for mutation proportional to its fitness measure. Thus, a mutation type with a higher fitness measure is preferred over a mutation type with a worse fitness measure.

Figure 3 illustrates the individual chances that the CR, the BKR, and the GBKR are mutated. If the random number R is in the range of the bottom part of the figure, the CR is chosen for mutation. If R is in the range of the middle part, the BKR is chosen for mutation. Otherwise, the GBKR is chosen.

After the mutation process, we calculate the fitness of the chosen mutation. If its fitness is better than the BKR, we update the BKR to the mutation.

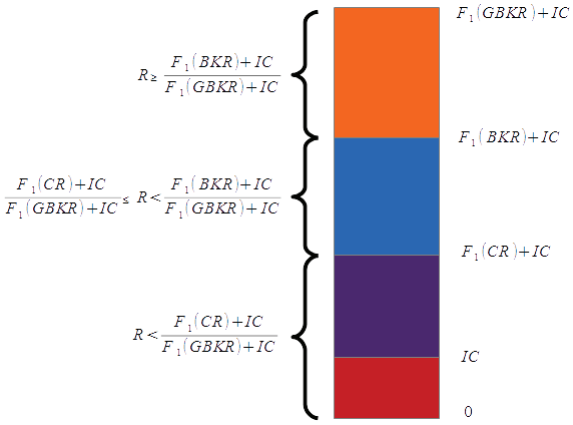

Figure 3 For PSO, depending on the value of the random variable R between zero and one, the CR, BKR, or GBKR is chosen for mutation. The CR is chosen if R is in the bottom third of the figure, the BKR is chosen if R is in the middle third of the figure, and the GBKR is chosen if R is in the upper third of the figure.

We update the GBKR in the same manner. After all iterations, we select the rule with the highest fitness. In Algorithm 1, we show the PSO algorithm in pseudocode. Table 7 in the appendix shows the settings we use for PSO. These are the optimal settings found using the hill climbing method.

4.2 2-Phase Optimization

2PO [41] is a metaheuristic that consists of two algorithms: II and SA. In II, an empty rule set is created. Then, at the beginning of each iteration, a rule is initialized using a tree initializer. The considered rule is mutated until no improvement in fitness occurs, resulting in a local optimum rule. If the fitness of the local optimum rule is better than the fitness of the best-known rule over all previous iterations, the best-known rule is updated to the local optimum rule. After all iterations, the best-performing rule is added to the rule set. This process is repeated a predefined number of iterations. Algorithm 2 describes II in pseudocode.

The second algorithm of 2PO is SA, which is an optimization algorithm that aims to find an optimal solution by narrowing the candidate solutions when the temperature drops until a local optimum is found. After II finishes, SA starts with a subset of the rule set that II returns. Rules are selected using a predefined elitism rate that selects a percentage of the highest fitness rules in the rule set. Then, for each selected rule, a given initial temperature

Algorithm 1 Particle swarm optimization

```
input : ruleSetSize as the size of the rule set
input : iterations as the number of iterations
input : initialConstant as the initial constant
output: The highest fitness rule

gbkr ← null
cr[ruleSetSize] ← ∅
bkr[ruleSetSize] ← ∅
/* Initialize CRs and BKRs, calculate their fitness, and select the GBKR                    */
for ruleID ← 0 to ruleSetSize do
    cr[ruleID] bkr[ruleID] ← initializeTree()
    if calculateFitness(cr[ruleID]) > calculateFitness(gbkr) then
        gbkr ← cr[ruleID]

for iteration ← 0 to iterations do
    for ruleID ← 0 to ruleSetSize do
        /* Calculate fitness for CR, BKR, and GBKR, including the initial constant          */
        fitness ← calculateFitness(cr[ruleID]) + initialConstant
        bestFitness ← calculateFitness(bkr[ruleID]) + initialConstant
        globalBestFitness ← calculateFitness(gbkr) + initialConstant
        /* Select the mutation type: CR, BKR, or GBKR                                       */
        random ← randomDouble()
        if globalBestFitness == 0 or random < fitness / globalBestFitness then
            selected ← cr[ruleID]
        else if random < bestFitness / globalBestFitness then
            selected ← bkr[ruleID]
        else
            selected ← gbkr

        /* Mutate the selected mutation type if its fitness is greater than 0, else
           initialize again                                                                 */
        if calculateFitness(selected) == 0 then
            cr[ruleID] ← initializeTree()
        else
            cr[ruleID] ← mutate(selected)

        /* Update the BKR and GBKR                                                          */
        fitness ← calculateFitness(cr[ruleID])
        if fitness > calculateFitness(bkr[ruleID]) then
            bkr[ruleID] ← cr[ruleID]
            if fitness > globalBestFitness then
                gbkr ← cr[ruleID]

return gbkr
```

is set. Until the dropping temperature reaches a frozen state, the algorithm iterates through the dropping temperature loop. In the dropping temperature loop, the considered rule is randomly mutated. The higher the fitness of the mutation in comparison to the fitness of the considered rule, and the higher the temperature, the higher the probability that the considered rule is updated to the mutation. This implies that better mutations are preferred over worse and that the algorithm narrows to a local optimum while the temperature drops.

If the mutation has a higher fitness than the best rule from previous iterations, the best rule is updated to the current mutation. At the end of each

Algorithm 2 Iterative improvement in 2-phase optimization

```
input  : ruleSetSize as the size of the rule set
input  : iterations as the number of iterations
output: The rule set ready for SA

ruleSet[ruleSetSize] ← ∅
for t ← 0 to ruleSetSize do
    best ← null
    for i ← 0 to iterations do
        rule ← initialize()
        fitness ← calculateFitness(rule)
        mutation ← null
        hasImproved ← true
        while hasImproved do
            hasImproved ← false
            mutation ← mutate(rule)
            mutationFitness ← calculateFitness(mutation)
            if mutation == null or mutationFitness > fitness then
                rule ← mutation
                hasImproved ← true
        if fitness > calculateFitness(best) then
            best ← rule
    addRulesToPopulation(ruleSet, best)
return ruleSet
```

dropping temperature loop, the temperature is reduced by a given reduction factor. After all the rules in the rule set are considered, the algorithm returns the highest fitness rule from the rule set. Algorithm 3 shows a summary of SA in pseudocode. The mutation procedure for II and SA is similar to the mutation procedure in PSO. Table 7 in the appendix shows the used parameters for 2PO.

4.3 Ant Colony Optimization

ACO [14] derives its name from the metaphor of the natural behavior of ants in their colony. When ants search for food, they aim to optimize their supply. In order to arrange this efficiently, ants try to minimize the distance between their nest and a food source. Ants leave pheromone traces on their path to and from the food source. The shortest routes can be traveled more often since these paths take less time to complete. Hence, paths with a shorter distance gather more pheromone traces than paths that are longer and thus less traveled. When ants choose their path from node to node, they have the ability to prefer paths with higher pheromone traces, which in turn leaves even more pheromones on these shorter tours. Over time, the ants' path choices converge to an optimal path from nest to food source and back.

Algorithm 3 Simulated annealing in 2-phase optimization

```
input : ruleSet after II
input : elitismRate the percentage highest fitness rules from rule set to be selected
input : initialTemperature as the initial temperature
input : frozenTemperature as the temperature that indicates a frozen state
input : temperatureReduction as the reduction of the temperature after each iteration
output: The highest fitness rule

/* Elitism: proceed with highest fitness rules                                  */
ruleSet ← selectHighestFitnessRule(ruleSet, elitismRate)
forall rule ∈ ruleSet do
    bestRule ← rule
    temperature ← initialTemperature
    fitness ← calculateFitness(rule)
    while temperature > frozenTemperature do
        mutation ← mutate(rule)
        mutationFitness ← calculateFitness(mutation)
        if mutationFitness > calculateFitness(bestRule) then
            bestRule ← mutation
        /* Difference between previous fitness and mutation fitness         */
        delta ← fitness − mutationFitness
        /* Probability to be mutated                                        */
        probability ← exp −delta / temperature
        /* Mutation becomes rule when mutation fitness is higher than rule fitness or random
           number between 0 and 1 is smaller than probability               */
        if delta < 0 or randomDouble() < probability then
            rule ← mutation
        temperature ← temperature × (1 − temperatureReduction)
    rule ← bestRule
return selectHighestFitnessRule(ruleSet)
```

In the proposed ACO algorithm, rules represent paths from nest to food. The ant's aim is to optimize the shortest path, while our algorithm aims to optimize the rule with regard to its fitness measure. Each node in a path is represented by a node in a rule. The order of the edges that the ants pass is represented by the order of nodes in the rule. The algorithm starts with filtering all nodes that are considered. For all terminal nodes, an F_1-measure is calculated. All function nodes are considered in our algorithm. In addition to this, the terminal nodes that have an F_1-measure equal to or larger than a predefined terminal threshold remain. Then, an initial amount of pheromones is deposited on all paths between the filtered nodes. A predefined number of ants start 'running' a number of iterations on the paths between the nodes.

In each iteration, a rule is built as an ant walks a path: the ant starts at its nest (i.e., the root tree, or empty rule), and chooses node for node on which edge it walks. Our contribution to the ACO algorithm is that we adjust the original graph structure to make it fit the tree structure of the rules. When ants walk their paths in our algorithm, they first visit the parent node and visit

the rest of the nodes in a depth-first manner. We select a maximum depth of five and also set the maximum number of children nodes per node to five.

The choice for the next node is proportionally equal to a probability that is computed and takes into account a local measure and a global measure:

$$p_{xy}^k(i) = \begin{cases} \dfrac{\tau_{xy}^\alpha (i-1)\eta_{xy}^\beta}{\sum_{z \in V_k(x)} \tau_{xz}^\alpha (i-1)\eta_{xz}^\beta} & z \in V_k(x) \\ 0 & z \notin V_k(x) \end{cases}, \tag{1}$$

where $p_{xy}^k(i)$ is the probability to move ant k from node x to node y in iteration i, $\tau_{xy}^a(i-1)$ is the global pheromone amount on edge xy in iteration $i-1$, η_{xz}^β is the local measure, η is the F_1-measure of the next node, $V_k(x)$ are the unvisited edges of ant k after visiting node x, and α and β are parameters that control the influence of the local and global measure, respectively.

The global measure is the amount of pheromones on the edge between the current and the next node. For the local measure, a good solution is to measure the next node's contribution to the fitness measure of the complete (by complete we mean including the next node) rule (i.e., the difference in the rule's fitness measure between the rule with and the rule without the next node). However, in order to compute the optimal local measure, an immense number of fitness scores are required which is computationally infeasible. Therefore, we opt to set the local measure of the next node to the node's F_1-measure that is computed in the previously described node filtering process. This is only possible for terminal nodes, because the F_1-measure cannot be computed for function nodes. The local measure for function nodes is 0, such that the choice for the next function node is completely based on its global measure.

When the ant has walked its path, a rule is constructed from all nodes the ant has passed. Then, the fitness is measured for the rule. The best rule found so far is updated if the considered rule is better than the current best rule. At the end of each iteration, once all ants have walked their paths, the pheromones on the traveled edges are updated as follows:

$$\tau_{xy}(i) = (1-\rho)\tau_{xy}(i-1) + \sum_{k=1}^{m} \Delta\tau_{xy}^k(i), \tag{2}$$

where ρ is the evaporation rate, which determines how fast pheromone traces evaporate, and $\Delta\tau_{xy}^k(i)$ is the pheromone amount that is deposited by ant k

in iteration i, which is defined as follows:

$$\Delta\tau_{xy}^{k}(i) = \begin{cases} F1_{T_k(i)} & w_{xy} \in T_k(i) \\ 0 & w_{xy} \notin T_k(i) \end{cases}, \tag{3}$$

where $F1_{T_k(i)}$ is the F_1-measure of the traversed path $T_k(i)$ by ant k in iteration i, and w_{xy} is the edge between x and y in the graph.

This results in higher pheromone traces on edges that have resulted in higher fitness rules, and these edges are thus traveled more often in each iteration. Finally, when all ants have finished their iterations, the best-found rule for each ant is added to the empty rule set. The ACO algorithm returns the rule with the highest fitness measure from the rule set. Details on the pseudocode can be found in Algorithm 4, and the settings we employ for ACO can be found in Table 7 in the appendix. We again use the hill climbing method to optimize these settings.

4.4 Genetic Algorithm

GA [25] is a metaheuristic that imitates a population of chromosomes, which is subjected to evolution. In genetics, chromosomes mutate over time, such that the chromosomes that adapt the best to their environment survive. The evolutionary process consists of selection, mutation, and crossover.

In the GA proposed in the work of [26], a population and its chromosomes are represented by a set of rules. A chromosome's genome is represented by a node or RHS entity in the rule tree. The algorithm starts by constructing a rule set with a tree initializer. Then, the algorithm starts iterating. In each iteration, a new rule set is constructed, which contains rules or mutations of rules from previous iterations. Using the elitism rate variable, the best n performing rules are copied to the next generation without any changes. Then, the mutation probability selects a subset of the rules that are subjected to mutation. In the mutation process, one node is chosen from the rule tree and is replaced by a randomly generated subtree. Last, crossover is performed on a subset of the rule set. For each crossover, two rules are selected and from each rule tree, one node is selected. The two selected nodes are then swapped. Parent nodes are selected with a probability of 90%, whereas this is only 10% for terminal nodes. Most nodes in a rule tree are terminal nodes, but swapping these terminal nodes does not change the fitness much, whereas swapping parent nodes has a larger influence on the fitness score. For example, swapping the parent node operators 'and' and 'or' has a significant impact on the fitness.

Algorithm 4 Ant colony optimization

```
input : nodes as an array containing all nodes
input : terminalThreshold as the F1 threshold for filtering terminal nodes
input : initialTraillntensity as the initial trail intensity
input : numberOfAnts as the number of ants
input : iterations as the number of iterations
input : α as the global measure influence factor
input : β as the local measure influence factor
input : ρ as the pheromone evaporation rate
input : filterTerminalNodes as the filtering of terminal nodes
input : buildRule as building the rule as described in the text
input : updatePheromones as updating pheromone amounts
output: The highest fitness rule tree

nodes ← filterTerminalNodes(nodes,terminalThreshold)
/* Initialize trail intensities on all edges                                  */
for i ← 0 to sizeOf(nodes) do
    for j ← 0 to sizeOf(nodes) do
        trailIntensities[i,j] ← initialTrailIntensity
shortestTour ← null
for iteration ← 0 to iterations do
    for a ← 0 to numberOfAnts do
        /* Build rule based on trail intensities                              */
        ants[a] ← buildRule(nodes, trailIntensities, α, β)
        fitness ← calculateFitness(ants[a])
        /* Update shortest path                                               */
        if fitness > calculateFitness(shortestTour) then
            shortestTour ← ants[a]
    /* Update trail intensities for all edges                                 */
    for a ← 0 to numberOfAnts do
        for i ← 0 to sizeOf(nodes) do
            for j ← 0 to sizeOf(nodes) do
                trailIntensities[i,j] ← updatePheromones(calculateFitness(ants[a]), ρ)
return shortestTour
```

The algorithm stops iterating when the highest fitness rule from the rule set does not improve in a predefined last number of iterations. When finished, the GA algorithm returns the rule with the highest fitness measure. In Algorithm 5, GA is explained in pseudocode, and Table 7 in the appendix reports the settings that are used for GA. The settings are identical to the ones proposed by [26].

5 Evaluation

In this section, we compare four metaheuristics on ontology-based rule learning. First, Section 5.1 presents the evaluation measures we use. Then, Section 5.2 discusses the setup of the annotated events, as not all experts annotate in the same manner. Last, Section 5.3 discusses the results in detail.

Algorithm 5 Genetic algorithm

```
input : ruleSetSize as the size of the population
input : maximumSameFitness as the maximum number of generations with the same fitness
input : elitismRate as the elitism rate
input : mutationProbability as the probability a rule is mutated
input : selectHighestFitness selects the n rules from a rule set with the highest fitness
output: The highest fitness rule

population ← initialize(ruleSetSize)
sameFitness ← 0
previousFitness ← 0
while sameFitness < maximumSameFitness do
    newPopulation ← ∅
    /* Elitism                                                              */
    sizeElite ← round(sizeOf(population) × elitismRate)
    elite ← selectHighestFitness(population, sizeElite)
    addRulesToPopulation(newPopulation, elite)
    /* Mutation                                                             */
    sizeMutation ← round(sizeOf(population) × mutationProbability)
    mutation ← mutate(selectHighestFitness(population, sizeMutation))
    addRulesToPopulation(newPopulation, mutation)
    /* Crossover                                                            */
    population ← crossover(newPopulation)
    if calculateFitness(selectHighestFitness(population, 1)) ≤ previousFitness then
        sameFitness ← sameFitness +1
    else
        sameFitness ← 0

    previousFitness ← calculateFitness(selectHighestFitness(population, 1))
return selectHighestFitness(population, 1)
```

5.1 Experimental Setup

The aim of the four considered metaheuristics is to automatically extract events from news items. In order to measure the accuracy of this extraction task, we compare the extracted events with the events that knowledge experts extracted manually from the same news items.

Events that are extracted automatically as well as manually are called True Positives (TP). Events that are extracted automatically but not manually are called False Positives (FP). Last, events that are extracted manually but not automatically are called False Negatives (FN). A metaheuristic's performance improves if it extracts more TP, and fewer FP and FN.

From the obtained TP, FP, and FN, we compute precision, the automatically correctly extracted events divided by all automatically extracted events, and recall, the automatically correctly extracted events out of the manually extracted events. There is a trade-off between precision and recall, as it is possible to increase precision but at the cost of recall. Therefore, we evaluate the performance of the proposed metaheuristics using the F_1-measure, which is the harmonic mean between precision and recall:

$$F_1 = \frac{2 \times precision \times recall}{precision + recall}. \tag{4}$$

The harmonic mean is a well-known performance measure. However, rule learning tends to result in an explosion of nodes in a tree without a significant increase in F_1-measure. For the performance measure of seen data, this is not a problem. However, when evaluating those very specific rules on unseen data, this could lead to overfitting, which results in a significant drop in F_1-measure. According to Occam's razor [44], from rules that perform equally well, the smallest rule should be selected. Therefore, [26] impose a penalty on tree complexity, which is defined as follows:

$$Fitness(R) = \begin{cases} 0 & \text{if } F_1(R) = 0 \\ F_1(R) - \alpha \times l(R) & \text{if } F_1(R) > 0 \end{cases}, \tag{5}$$

where $F_1(R)$ is the F_1-measure of the proposed rule, α is the penalty on tree complexity, and $l(R)$ is the length of the proposed rule (i.e., the number of nodes in the rule). All algorithms employ this fitness measure to measure the performance of the constructed rules. Precision, recall, and the F_1-measure are employed for evaluation of the rule groups.

In order to find the optimal solution of the aforementioned task, it often takes algorithms much time to finish the calculations. In contrast, the four proposed metaheuristics approximate optimal solutions. This implies that an increase in accuracy comes at the cost of more computation time. Therefore, a trade-off has to be made between an acceptable F_1-measure and an acceptable computation time. To combine F_1-measure and computation time into one performance measure, we propose Time-Adjusted F_1 (TAF_1), which is similar to the A3R formula proposed in the work of [1], and is defined as follows:

$$TAF_1 = \frac{F_1}{\sqrt{T}}, \tag{6}$$

where T is the computation time in minutes. The proposed A3R formula [1] measures the relative performance of two models. Due to the number of considered models, we propose TAF_1, an absolute performance measure that takes into account both accuracy and computation time. We put more weight on accuracy rather than on computation time, and therefore take the square root of the computation time.

Our experiment is based on a selection of 500 news items with an average of 700 words from different sources, such as the New York Times, Reuters, Washington Post, and Businessweek. The news items are all in the financial and technology domain. For our experiment, we split the 500 news items into a training set and a test set. First, the rules are learned on the training set.

To take into account possible overfitting, we measure the performance of the rules on unseen data, which in this case is the test set.

We employ stratified testing on the selected news items, which implies that we split the number of annotations between the training and test set with the same percentage as we split the number of news items between the training and test set. We split the dataset into 80% training set and 20% test set. All the metaheuristics are time-consuming (but less than doing an exhaustive search), ranging from 1 hour to more than 2.5 hours on average for testing one event. Since we have ten events in our dataset, it takes time to gather all results. We acknowledge that our experiment would be more reliable if we employed cross-validation for testing, but, due to the large computation times, we decided to limit our experiment to stratified testing.

5.2 Inter-Annotator Agreement

Knowledge experts were asked to manually annotate the news items with events that are also available for the automatic extraction task. However, not all experts annotate the same events. One expert could discover an event in a sentence, while another expert does not assess this event in the same manner. Experts could for example not have the same knowledge about events.

To gain insight into this subjectiveness, we report the Inter-Annotator Agreement (IAA) of all events. When two out of three experts annotate the same event in a sentence, it is said to be 'agreed'. Only agreed events are considered in the evaluation. The IAA produces an objective measure of the consensus of the events among annotators. Its formula is defined as follows:

$$IAA = \frac{\sum_{a \in A} \frac{N(a)}{|K|}}{|A|}, \tag{7}$$

where K is the set of annotators, A is the set of annotations found by all annotators, and $N(a)$ is the number of annotators that agreed on annotation a. Table 1 shows the IAA for all events. The total IAA for all events is 71%.

As Table 1 indicates, some events do not need implicit knowledge (i.e., information that is not explicitly stated in text, but which the reader is supposed to know) and score high on IAA, whereas other events do need implicit knowledge and score low on IAA. Events such as 'CEO', 'Sales', and 'Share value' are most of the time stated explicitly in text with recognizable words. Events such as 'Competitor', 'Partner', and 'Subsidiary' require more implicit knowledge about the subject. Such company-to-company relations are assumed to be part of the reader's knowledge.

Table 1 Inter-annotator agreement per event

Event	Full Article	Single Sentence	IAA
CEO	161	135	0.83
Competitor	157	126	0.62
Partner	61	59	0.63
President	64	58	0.68
Product	344	300	0.73
Subsidiary	115	97	0.63
Loss	56	31	0.67
Profit	68	46	0.72
Sales	45	20	0.78
Share value	82	77	0.78
Total	1153	949	0.71

Table 2 Average results for particle swarm optimization, 2-phase optimization, ant colony optimization, and genetic algorithm over all events

	Training Set			Test Set				
Meta-heuristic	P	R	F_1	P	R	F_1	Time (s)	TAF_1
PSO	0.210	0.462	0.233	0.158	0.435	0.179	6,411	0.0174
2PO	0.106	0.206	0.085	0.062	0.159	0.068	4,895	0.0075
ACO	0.247	0.433	0.223	0.236	0.504	0.258	4,629	0.0294
GA	0.664	0.689	0.663	0.479	0.449	0.434	9,410	0.0346

5.3 Experimental Results

In this section, the obtained results are discussed, which we evaluate using precision, recall, and F_1-measure on the training and test set. We evaluate the performance using TAF_1 on the test set only.

Table 2 presents an overview of the average performance over all ten events on all considered performance measures. With a score of 0.434, GA outperforms all other metaheuristics with respect to F_1-measure, followed by ACO (0.258), PSO (0.179), and 2PO (0.068), respectively. The models obtain relatively low F_1-scores. However, the task at hand is difficult. The data is multi-labeled, and the metaheuristics extract 10 events, which makes the aforementioned task not easy. The best-performing metaheuristic on F_1-measure, GA, is also the most time-consuming method with an average of 9,410 seconds. The second-best metaheuristic on F_1-measure, ACO, is the fastest method with an average of 4,629 seconds. PSO (6,411) and 2PO (4,895) perform average considering computation time.

Table 3 Results for particle swarm optimization

Event	Training Set			Test Set			Time (s)	TAF_1
	P	R	F_1	P	R	F_1		
CEO	0.364	0.778	0.496	0.421	0.857	0.565	12,804	0.0387
Competitor	0.158	0.062	0.089	0.125	0.034	0.054	2,404	0.0085
Partner	0.133	0.174	0.151	0.000	0.000	0.000	4,212	0.0000
President	0.203	0.638	0.308	0.149	0.636	0.241	14,376	0.0156
Product	0.463	0.337	0.390	0.591	0.333	0.426	3,051	0.0597
Subsidiary	0.269	0.182	0.217	0.000	0.000	0.000	5,551	0.0000
Loss	0.167	0.077	0.105	0.000	0.000	0.000	2,156	0.0000
Profit	0.046	0.658	0.086	0.038	0.625	0.071	5,780	0.0072
Sales	0.044	0.750	0.082	0.082	1.000	0.151	7,777	0.0133
Share value	0.253	0.968	0.401	0.171	0.867	0.286	6,002	0.0286
Average	0.210	0.462	0.233	0.158	0.435	0.179	6,411	0.0174
Average*			0.289			0.232		0.0224

If we consider TAF_1, we observe that the less accurate, but faster metaheuristic ACO performs almost up to par with the most accurate, but slowest metaheuristic GA. GA outperforms the other models with a score of 0.0346, followed by ACO with 0.0294, PSO with 0.0174, and 2PO with 0.0075. It appears that often the precision of the other three considered metaheuristics is substantially lower than the precision of GA, whereas PSO reports similar recall, and ACO even reports a higher recall on the test set. From the average results, it appears that when one is solely interested in precision, GA seems to be the preferred method. However, if recall is important, ACO or even PSO might be interesting as a baseline, especially factoring in the lower computation time. The next paragraphs explain the differences in performance in detail and suggest possible solutions to increase the performance of PSO, 2PO, and ACO.

A detailed overview of the results per event is shown in Table 3 for PSO, Table 4 for 2PO, Table 5 for ACO, and Table 6 for GA. 'Average' is the sum of all events, divided by the number of events. 'Average*' for F_1-measure is the harmonic mean between the 'Average' precision and 'Average' recall, and 'Average*' for TAF_1 is computed using the 'Average*' F_1-measure and time. In the following paragraphs, we only discuss the results of the test set. The results on the training set are better than on the test set, but this is most likely due to overfitting.

For PSO, the events 'CEO', 'Product', and 'Share value' score relatively high on F_1-measure, while 'Partner', 'Subsidiary', 'Loss', and 'Sales' score null. 'Product' is also a good performer for 2PO. It has by far the

Table 4 Results for 2-phase optimization

	Training Set			Test Set				
Event	P	R	F_1	P	R	F_1	Time (s)	TAF_1
CEO	0.085	0.162	0.111	0.068	0.240	0.106	5,153	0.0114
Competitor	0.002	0.699	0.004	0.001	0.522	0.002	1,456	0.0004
Partner	0.027	0.220	0.048	0.000	0.000	0.000	13,080	0.0000
President	0.033	0.104	0.051	0.029	0.100	0.045	6,542	0.0043
Product	0.527	0.407	0.459	0.492	0.469	0.480	9,662	0.0378
Subsidiary	0.368	0.090	0.144	0.000	0.000	0.000	3,328	0.0000
Loss	0.010	0.040	0.015	0.000	0.000	0.000	1,968	0.0000
Profit	0.009	0.027	0.013	0.024	0.111	0.039	2,626	0.0059
Sales	0.000	0.133	0.000	0.000	0.000	0.000	1,842	0.0000
Share value	0.002	0.179	0.003	0.002	0.143	0.004	3,289	0.0005
Average	0.106	0.206	0.085	0.062	0.159	0.068	4,895	0.0075
Average*			0.140			0.089		0.0098

Table 5 Results for ant colony optimization

	Training Set			Test Set				
Event	P	R	F_1	P	R	F_1	Time (s)	TAF_1
CEO	0.331	0.435	0.376	0.462	0.857	0.600	6,550	0.0574
Competitor	0.095	0.158	0.119	0.091	0.120	0.103	4,203	0.0123
Partner	0.048	0.240	0.081	0.034	0.222	0.060	3,589	0.0078
President	0.172	0.708	0.276	0.148	0.900	0.254	3,883	0.0316
Product	0.519	0.796	0.628	0.667	0.865	0.753	11,712	0.0539
Subsidiary	0.818	0.130	0.225	0.000	0.000	0.000	3,860	0.0000
Loss	0.308	0.182	0.229	0.750	0.333	0.462	3,654	0.0592
Profit	0.148	0.622	0.240	0.190	0.889	0.314	3,463	0.0413
Sale	0.005	0.077	0.010	0.000	0.000	0.000	2,577	0.0000
Share value	0.022	0.984	0.042	0.019	0.857	0.038	2,801	0.0056
Average	0.247	0.433	0.223	0.236	0.504	0.258	4,629	0.0294
Average*			0.322			0.314		0.0366

best F_1-measure. However, 'Subsidiary', 'Sales', and 'Loss' score null for 2PO. Similar to PSO, for GA, 'Product' and 'CEO' perform relatively well. 'Partner' performs the worst for GA. The 'Product' event for ACO is the best-performing event of all metaheuristics with a F_1-measure of 0.753. 'CEO' and 'Loss' also perform well for ACO, while 'Subsidiary' and 'Sales' score null.

It appears that all considered metaheuristics extract events concerning products well, as the F_1-measure is the highest for the event 'Product' for

Table 6 Results for genetic algorithm

	Training Set			Test Set				
Event	P	R	F_1	P	R	F_1	Time (s)	TAF_1
CEO	0.744	0.908	0.818	0.593	0.593	0.593	20,631	0.0320
Competitor	0.745	0.406	0.526	0.833	0.200	0.323	5,903	0.0326
Partner	0.500	0.550	0.524	0.100	0.053	0.069	5,451	0.0072
President	0.756	0.660	0.705	0.538	0.636	0.583	3,866	0.0726
Product	0.665	0.893	0.762	0.583	0.836	0.687	22,406	0.0356
Subsidiary	0.488	0.541	0.513	0.419	0.565	0.481	10,355	0.0366
Loss	0.609	0.583	0.596	0.200	0.143	0.167	7,051	0.0154
Profit	0.744	0.784	0.763	0.400	0.667	0.500	4,860	0.0556
Sales	0.750	0.563	0.643	0.333	0.250	0.286	4,096	0.0346
Share value	0.640	1.000	0.781	0.786	0.550	0.647	9,483	0.0515
Average	0.664	0.689	0.663	0.479	0.449	0.434	9,410	0.0346
Average*			0.676			0.463		0.0370

all four metaheuristics. Furthermore, the event 'CEO' also performs well for all metaheuristics, whereas other events, such as 'Partner', are often not extracted by the proposed metaheuristics. This indicates that all metaheuristics find certain events easier to extract, while other events are often not automatically extracted.

Events that have a high IAA score tend to have a high F_1-measure as well. An explanation for this could be that the events which can be easily recognized by humans can be easily extracted by metaheuristics as well. If an event is difficult to detect for humans, computers are probably less likely to extract the event correctly. Furthermore, events that need more implicit knowledge score relatively lower on F_1-measure compared to events that are stated explicitly in text.

In the proposed PSO metaheuristic, the rule to be mutated depends on the F_1-measures of the CR, the BKR, and the GBKR. This heuristic leads to a broad exploration of the search space. Worse-performing rules are dropped fast, and well-performing rules survive. However, the probabilities of choosing between the CR, the BKR, and the GBKR are linear (with respect to their fitness measures) in our proposed PSO metaheuristic. A more sophisticated probability distribution could result in an improvement in F_1-measure or a decrease in computing time. Besides, the mutation process proposed for PSO and 2PO consists of mutation and exclusion of nodes, whereas in GA, the evolutionary processes elitism, mutation, and crossover are applied. It is likely that adding more evolutionary processes to the PSO and 2PO metaheuristics increases the F_1-measure, as the search space could

be explored more. We, however, did not add more evolutionary processes to PSO and 2PO in order to increase their speed.

Besides the fact that 2PO lacks a broader set of evolutionary processes, 2PO fails to explore a more complete search space. In the II step, 2PO is missing a smart heuristic that combines the local and global performance measures. Now, the II step does not build further upon promising rules. A similar conclusion can be drawn for the SA algorithm. It moves away from its best solution, resulting in rules that are not good enough. A better heuristic that regularly returns to its best solution and explores a more promising search space could improve performance. However, the dropping temperature process in SA is well designed to direct the considered rule to a feasible solution.

In the proposed ACO metaheuristic, we suggested a local performance measure that is computed as the F_1-measure for the considered node. We proposed this local measure because the actual local measure was not feasible, as it was too time-consuming. ACO seems to be a smart metaheuristic, but, due to the large number of nodes in our database, we do not take full advantage of the local performance measure. The heuristic that is applied to the local performance measure needs to be improved. However, the global performance measure seems to be a good indicator of the accuracy of the rule. Furthermore, our proposed ACO metaheuristic lacks a good stopping or pruning heuristic. By adding a node to the rule, the metaheuristic does not question whether the proposed node is beneficial for the rule. Therefore, ACO produces rules that are too large, which leads to overfitting. This results in poor performance on the test set. The idea to transform the graph structure of the original ACO approach into the rules' tree structure seems to work.

6 Conclusion

To make text easier to understand for computers, unstructured information in text can be stored in ontologies as concepts and relations between those concepts. In this paper, the ontology language Resource Description Framework (RDF) is employed to learn information extraction rules. Learning information extraction rules helps to automate the time-consuming process of adding information to ontologies by hand. RDF rules are subject-predicate-object expressions that express certain events in text. These expressions are extracted from the text when positively matched against the expression's pattern, which is composed of lexico-syntactic and lexico-semantic features.

At the time of writing this paper, mostly machine learning approaches have been proposed for ontology-based information extraction. In this work, we focus on metaheuristics. Metaheuristics are applicable to the optimization task we perform and are often preferred for the considered task to machine learning approaches since they more frequently find feasible solutions in a shorter time. On learning ontology-based information extraction rules, we compare the metaheuristics Particle Swarm Optimization (PSO) [28], 2-Phase Optimization (2PO) [41], Ant Colony Optimization (ACO) [14], and Genetic Algorithm (GA) [25].

The aforementioned metaheuristics are implemented as tabs in the rule learning component [26] of Hermes [8], which is a framework for news item extraction. The rules in the rule learning component are defined using the Hermes Information Extraction Language [27]. In this work, we evaluate the models on financial news items, which have been annotated by financial experts. The news items are multi-labeled data. In addition to this, the models extract ten events, which makes the task at hand difficult.

GA outperforms all other considered metaheuristics with an F_1-measure of 0.434. The metaheuristics PSO (0.179) and 2PO (0.068) perform worse than GA with respect to F_1-measure, but ACO's performance is promising compared to these two metaheuristics, with an F_1-measure of 0.258. However, on computing time, GA (9,410 seconds) performs worse than all other metaheuristics. If we combine time and accuracy in the proposed Time-Adjusted F_1 (TAF_1) performance measure, we find that ACO (0.0294) performs up to par with GA (0.0346) with respect to TAF_1, whereas PSO (0.0174) and 2PO (0.0075) again perform worse. An important note to add is that TAF_1 values accuracy more than computation time, which explains the higher value of TAF_1 for GA. In addition to this, GA significantly outperforms the other methods when considering the obtained precisions. However, ACO obtains a higher recall than GA, which is in line with the trade-off between precision and recall. Therefore, when recall and a shorter computation time are favored, GA might not be the preferred baseline. However, it appears that GA should be preferred in the financial domain when favoring precision or F_1-measure.

With regards to computation time, GA is the slowest metaheuristic. The trade-off between F_1-measure and speed is apparent in the performed experiment. To our knowledge, this is one of the first papers to investigate metaheuristics for ontology-based information extraction rule learning from both accuracy and computation time points of view.

For future research, a more sophisticated probability distribution between the current rule, the best-known rule, and the global best-known rule for PSO could increase performance. An example of this is to assign weights (to be tuned) to the probabilities. For PSO and 2PO, we suggest adding more evolutionary processes to their metaheuristics. This, however, should not result in a steep increase in computing time, again referring to the trade-off between accuracy and computation time. For 2PO, future research could investigate an improvement of Iterative Improvement (II) and Simulated Annealing (SA), such that 2PO explores the search space in a more optimal manner. For example, II now just iterates over possible solutions. Future research could look into replacing II by a more complex algorithm that also iterates over possible solutions, but that continues to build on promising solutions. Also, we suggest pruning the rules that are produced by ACO, such that the rules do not grow unnecessarily large. This can be achieved by recursively removing nodes from the rule and continuing to remove nodes when the F_1-measure does not improve.

Appendix

In this appendix, we present the used settings for all four considered metaheuristics. The values are found using the hill climbing method.

Table 7 Settings of the four considered metaheuristics

Parameters	PSO	2PO	ACO	GA
ruleSetSize	50	10	–	100
iterations	20	10	25	–
elitismRate	–	1	–	0.3
initialConstant	0.001	–	–	–
initialTemperature	–	0.05	–	–
frozenTemperature	–	0.001	–	–
temperatureReduction	–	0.1	–	–
terminalThreshold	–	–	0.005	–
initialTrailIntensity	–	–	1	–
numberOfAnts	–	–	300	–
α	–	–	2	–
β	–	–	1	–
ρ	–	–	0.2	–
mutationProbability	–	–	–	0.3
sameFitness	–	–	–	50

References

[1] S. M. Abdulrahman and P. Brazdil. Measures for combining accuracy and time for meta-learning. In *2014 International Workshop on Meta-Learning and Algorithm Selection at the 21st European Conference on Artificial Intelligence*, volume 1201, pages 49–50. CEUR-WS.org, 2014.

[2] J. S. Aitken. Learning information extraction rules: An inductive logic programming approach. In *15th European Conference on Artificial Intelligence (ECAI 2002)*, volume 77, page 355. IOS Press, 2002.

[3] A. Askarzadeh, L. dos Santos Coelho, C. E. Klein, and V. C. Mariani. A population-based simulated annealing algorithm for global optimization. In *2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC 2016)*, pages 4626–4633. IEEE, 2016.

[4] A. Ayadi, A. Samet, F. d. B. de Beuvron, and C. Zanni-Merk. Ontology population with deep learning-based NLP: A case study on the biomolecular network ontology. In *23rd International Conference on Knowledge-Based and Intelligent Information & Engineering Systems (KES 2019)*, volume 159, pages 572–581. Elsevier, 2019.

[5] S. Bechhofer, F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider, and L. A. Stein. OWL web ontology language reference – W3C recommendation 10 February 2004, 2004. http://www.w3.org/TR/owl-ref/.

[6] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[7] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.

[8] J. Borsje, L. Levering, and F. Frasincar. Hermes: A Semantic Web-based news decision support system. In *2008 ACM Symposium on Applied Computing (SAC 2008)*, pages 2415–2420. ACM, 2008.

[9] M. E. Califf and R. J. Mooney. Bottom-up relational learning of pattern matching rules for information extraction. *The Journal of Machine Learning Research*, 4:177–210, 2003.

[10] P. Cimiano and J. Völker. Text2Onto: A framework for ontology learning and data-driven change discovery. In *10th International Conference on Applications of Natural Language to Information Systems (NLDB 2005)*, volume 3513 of *LNCS*, pages 227–238. Springer, 2005.

[11] W. W. Cohen and Y. Singer. A simple, fast, and effective rule learner. In *16th National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence*, AAAI '99/IAAI '99, pages 335–342. AAAI, 1999.

[12] M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery. Learning to extract symbolic knowledge from the world wide Web. In *Fifteenth National Conference on Artificial Intelligence and Tenth Innovative Applications of Artificial Intelligence*, AAAI '98/IAAI '98, pages 509–516. AAAI, 1998.

[13] H. Cunningham, D. Maynard, and V. Tablan. JAPE: A Java annotation patterns engine. Technical report, University of Sheffield, Department of Computer Science, 2000.

[14] M. Dorigo, V. Maniezzo, and A. Colorni. Ant system: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics (SMC 1996)*, 26(1):29–41, 1996.

[15] O. Etzioni, M. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll. In *13th International Conference on World Wide Web (WWW 2004)*, pages 100–110. ACM, 2004.

[16] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. Weld, and A. Yates. Methods for domain-independent information extraction from the Web: An experimental comparison. In *19th National Conference on Artificial Intelligence*, pages 391–398. AAAI, 2004.

[17] C. Fellbaum. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.

[18] F. Frasincar, J. Borsje, and L. Levering. A Semantic Web-based approach for building personalized news services. *International Journal of E-Business Research*, 5(3):35–53, 2009.

[19] D. Freitag. Machine learning for information extraction in informal domains. *Machine Learning*, 39(2-3):169–202, 2000.

[20] R. M. Ghoniem, N. Alhelwa, and K. Shaalan. A novel hybrid genetic-whale optimization model for ontology learning from Arabic text. *Algorithms*, 12(9):182, 2019.

[21] U. Hahn and K. Schnattinger. Towards text knowledge engineering. *Hypothesis*, 1(2):524–531, 1998.

[22] M. Hajji, M. Qbadou, and K. Mansouri. An adaptation of Text2Onto for supporting the french language. *International Journal of Electrical & Computer Engineering*, 10(4):3743–3750, 2020.

[23] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *14th International Conference on Computational Linguistics (COLING 1992)*, volume 2, pages 539–545. ACL, 1992.

[24] M. A. Hearst. *WordNet: An Electronic Lexical Database and Some of its Applications*, chapter Automated Discovery of WordNet Relations, pages 131–151. MIT Press, 1998.

[25] J. H. Holland. Genetic algorithms. *Scientific American*, 267(1):66–73, 1992.

[26] W. IJntema, F. Hogenboom, F. Frasincar, and D. Vandic. A genetic programming approach for learning semantic information extraction rules from news. In *15th International Conference on Web Information Systems Engineering (WISE 2014)*, volume 8786 of *LNCS*, pages 418–432. Springer, 2014.

[27] W. IJntema, J. Sangers, F. Hogenboom, and F. Frasincar. A lexico-semantic pattern language for learning ontology instances from text. *Journal of Web Semantics*, 15:37–50, 2012.

[28] J. Kennedy and R. Eberhart. Particle swarm optimization. In *International Conference on Neural Networks (ICNN 1995)*, pages 1942–1948. IEEE, 1995.

[29] J.-U. Kietz and K. Morik. A polynomial approach to the constructive induction of structural knowledge. *Machine Learning*, 14(1):193–217, 1994.

[30] M. S. Kiran. Particle swarm optimization with a new update mechanism. *Applied Soft Computing*, 60:670–678, 2017.

[31] G. Klyne and J. J. Carroll. Resource description framework (RDF): Concepts and abstract syntax - W3C recommendation 10 February 2004, 2004. http://www.w3.org/TR/rdf-concepts/.

[32] Kunder, M. de. The size of the world wide Web (the internet), 2024. http://www.worldwidewebsize.com.

[33] S. M. Lim, A. B. M. Sultan, M. N. Sulaiman, A. Mustapha, and K. Y. Leong. Crossover and mutation operators of genetic algorithms. *International Journal of Machine Learning and Computing*, 7(1):9–12, 2017.

[34] A. Maedche and S. Staab. Ontology learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2):72–79, 2001.

[35] S. Mittal and N. Mittal. Tools for ontology building from texts: Analysis and improvement of the results of Text2Onto. *IOSR Journal of Computer Engineering*, pages 2278–0661, 2013.

[36] K. E. Parsopoulos and M. N. Vrahatis. UPSO: A unified particle swarm optimization scheme. In *International Conference of Computational Methods in Sciences and Engineering*, pages 868–873. CRC Press, 2019.

[37] E. Prud'hommeaux and A. Seaborne. SPARQL query language for RDF – W3C recommendation 15 January 2008, 2008. http://www.w3.org/TR/rdf-sparql-query/.

[38] D. K. Sharma, R. Pamula, and D. Chauhan. A hybrid evolutionary algorithm based automatic query expansion for enhancing document retrieval system. *Journal of Ambient Intelligence and Humanized Computing*, pages 1–20, 2019.

[39] N. Singh and S. Singh. Hybrid algorithm of particle swarm optimization and grey wolf optimizer for improving convergence performance. *Journal of Applied Mathematics*, 2017:2030489, 2017.

[40] S. Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 34:233–272, 1999.

[41] A. Swami and A. Gupta. Optimization of large join queries. In *1988 ACM SIGMOD International Conference on Management of Data*, pages 8–17. ACM, 1988.

[42] M. Vargas-Vera, E. Motta, J. Domingue, S. B. Shum, M. Lanzoni, et al. Knowledge extraction by using an ontology-based annotation tool. In *K-CAP 2001 Workshop Knowledge Markup and Semantic Annotation*, volume 99 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2001.

[43] Wikipedia. Laplace distribution, 2023. http://en.wikipedia.org/wiki/Laplace_distribution.

[44] Wikipedia. Occam's razor, 2023. http://en.wikipedia.org/wiki/Occam%27s_razor.

[45] X.-S. Yang, S. Deb, and S. Fong. Accelerated particle swarm optimization and support vector machine for business optimization and applications. In *Third International Conference on Networked Digital Technologies (NDT 2011)*, volume 136, pages 53–66. Springer, 2011.

[46] R. Yangarber, R. Grishman, P. Tapanainen, and S. Huttunen. Automatic acquisition of domain knowledge for information extraction. In *18th Conference on Computational Linguistics (COLING 2000)*, volume 2, pages 940–946. Morgan Kaufmann, 2000.

Biographies

Michel Capelle received the B.S. degree in economics and informatics from Erasmus University Rotterdam, Rotterdam, the Netherlands, in 2012, and the M.S. degree in computational economics, specialized in Semantic Web technology, at Erasmus University Rotterdam, Rotterdam, the Netherlands, in 2015. His research interests include machine learning, linked data, information extraction rule learning, natural language processing, and sentiment analysis. He has published in various conferences and journals in the area of machine learning and the Semantic Web.

Flavius Frasincar received the M.S. degree in computer science, in 1996, and the M.Phil. degree in computer science, in 1997, from Politehnica University of Bucharest, Bucharest, Romania, and the P.D.Eng. degree in computer science, in 2000, and the Ph.D. degree in computer science, in 2005, from Eindhoven University of Technology, Eindhoven, the Netherlands.

Since 2005, he has been an Assistant Professor in computer science at Erasmus University Rotterdam, Rotterdam, the Netherlands. He has published in numerous conferences and journals in the areas of databases, Web information systems, personalization, machine learning, and the Semantic

Web. He is a member of the editorial board of *Computational Linguistics in the Netherlands Journal* and co-editor-in-chief of the Journal of Web Engineering. He is also a member of the Association for Computing Machinery.

Finn van der Knaap received the B.S. degree cum laude in Econometrics and Operations Research from Erasmus University Rotterdam, the Netherlands, in 2023, and the M.S. degree summa cum laude in Econometrics and Management Science, specialized in Quantitative Finance, at Erasmus University Rotterdam, the Netherlands, in 2024. In 2025, he received the M.S. degree with distinction in Artificial Intelligence at the University of Edinburgh, Scotland. His research interests include machine learning, finance, and sentiment analysis. He has published in various conferences and journals in the areas of quantitative finance, machine learning, and the Semantic Web.