Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2018) 26: 830 – 843

doi:10.3906/elk-1703-108

Relation extraction via one-shot dependency parsing on intersentential, higher-order, and nested relations

Gözde Gül ŞAHİN[1,*], Erdem EMEKLİGİL[2], Seçil ARSLAN[2], Onur AĞIN[2], Gülşen ERYİĞİT[1]

[1]Department of Computer Engineering, İstanbul Technical University, İstanbul, Turkey
[2]R&D and Special Projects, Yapı Kredi Technology, İstanbul, Turkey

Received: 09.03.2017 • **Accepted/Published Online:** 13.07.2017 • **Final Version:** 30.03.2018

Abstract: Despite the emergence of digitalization, people still interact with institutions via traditional means such as submitting free formatted petitions, orders, or applications. These noisy documents generally consist of complex relations that are nested, higher-order, and intersentential. Most of the current approaches address extraction of only sentence-level and binary relations from grammatically correct text and generally require high-level linguistic features coming from preprocessors such as a parts-of-speech tagger, chunker, or syntactic parser. In this article, we focus on extracting complex relations in order to automate the task of understanding user intentions. We propose a novel language-agnostic and noise-immune approach that does not require preprocessing of input text. Unlike previous literature that uses dependency parsing outputs as input features, we formulate the relation extraction task directly as a one-shot dependency parsing problem. The presented method was evaluated using a representative dataset from the banking domain and obtained 91.84% labeled attachment score (LAS), which provides an improvement of 42.85 percentage points over a rule-based baseline.

Key words: Natural language processing, relation extraction, dependency parsing

1. Introduction

Relation extraction (RE) is a crucial step to many natural language understanding tasks, such as question answering [1] and information extraction, e.g., from business reports [2], health records [3], or medical documents [4]. Despite recent advances in technology, people still continue to interact with institutions like banks, government offices, and law firms via traditional means such as submitting free formatted petitions, orders, or applications. Considering the volume of data, understanding user intentions from these documents is a tedious, time-consuming task that is mostly performed by humans. Automatic understanding of user intentions can be considered a type of RE problem, which usually consists of extracting related parameters of intention as named entities (e.g., Person, Location) and finding relations between these (e.g., APPLYJOB (Person, Organization)).

Previous literature in RE usually focuses on binary relations $r = \{e_i\ ,\ e_j\}$, where e_i is the ith entity in a document D. However, the structure of user intentions is generally of higher order, such as $r = \{e_1, e_2, ..., e_n\}$ (e.g., REMIND (Person, Date, Location, Topic)). Although D is mostly considered a single well-formed sentence by previous work, documents like petitions, applications, and orders generally contain multiple sentences along with supporting unstructured text, e.g., tables, address information, and signatures where entities of relations

*Correspondence: isguderg@itu.edu.tr

may occur in different parts of the document. Furthermore, entities may have useful additional "properties," as in the popular knowledge graph Freebase [5]. For example, an application form or a case file may contain "name," "gender," and "age" properties related to a Person entity, which is actually another kind of relation that should be detected.

The problem of extracting predefined relations between entities is generally considered as a machine learning (ML) task. In addition to supervised feature-based and kernel-based methods, distantly supervised and neural models have been employed to address RE. Most of the previous methods heavily rely on outputs produced by syntactic parsers and perform RE at the sentence level. However, document D may contain noisy, ungrammatical text due to the input channels (such as fax, e-mail, or scanner) in addition to its unstructured format. This complicates using linguistic preprocessors (e.g., parts-of-speech tagging, chunking, and syntactic parsing) that are tailored for well-formatted text. Furthermore propagation of preprocessor errors hinders the performance of RE systems.

In this work, we focus on extracting higher-order, nested, intersentential relations between entities along with their additional properties from free-format documents in order to automate the task of understanding user intentions. We propose a novel language-independent approach that does not require high-level preprocessing of input text. Unlike previous relation extraction literature that uses dependency parsing outputs as input features, we formulate the relation extraction task directly as a one-shot dependency-parsing task by considering syntactic units as named entities occurring within an entire document instead of tokens of a single sentence. This method is evaluated using a representative dataset from the banking domain, where "transaction" intentions were extracted, and a labeled attachment score (LAS) of 91.84% was obtained.

2. Related work

In this section, we discuss previous literature on supervised and semisupervised machine learning methods used for RE among with commonly used data sets. Later we give a brief description of dependency parsing methods. Kambhatla [6] and GuoDong et al. [7] train separate binary classifiers such as one-versus-many SVM for each relation that employ features, such as dependency-tree paths, bag of chunk heads, chunk paths, bag of words, named entity types, and number of words between entities, that need careful feature design. To address this problem, kernel-based methods that explore a higher dimensional input space have been proposed [8–10]. However, most features used by these supervised methods require preprocessing of input via syntactic parsers. More recently, end-to-end neural-network–based methods have been employed for RE [11,12]. Although these methods do not require syntactic parsing, they rely on a large number of parameters and generally execute at the sentence level.

Higher-level tasks such as relation extraction [13], event extraction [14], and nested named entity recognition [15] have previously been modeled within the dependency-parsing framework. In [14], researchers form a tree of entities and event anchors (phrases that anchor events) similar to our work. They train a dependency parser from the "event trees" that are composed of extracted entities and event triggers. Although this approach is close to ours in spirit, they supply features that require a prior syntactic parsing (e.g., syntactic paths in the original sentence between nodes in an event dependency) and domain-specific information (e.g., ontology) and extract events at sentence granularity. On the other hand, our approach does not require any high-level syntactic knowledge and executes at document level.

Automatic content extraction (ACE) [16] is the most common RE evaluation corpus used by the majority of supervised techniques. These multilingual corpora have different types of relations, e.g., location, family,

employment, affiliation. However, relations are limited to those that are expressed within a single sentence and occur between two entities. AIMED [8] is a collection of articles in the medical domain, where interactions between protein pairs are labeled. Although relations are intersentential (document-level), AIMED contains only one entity type (protein) and one relation type (interaction). SemEval2010 Task8 [17] introduces another corpus, which contains sentence-level relations between nominals rather than entities.

Dependency-parsing methods aim to grammatically analyze sentences via extracting this dependency structure based on dependency grammar theory [18]. In other words, they establish parent–child relationships between words and form directed, connected, and acyclic dependency trees of sentences. Graph-based and transition-based parsing are the two most commonly used data-driven dependency-parsing methods. Graph-based methods predict entire graph given the input using maximum spanning tree algorithms [19] while transition-based parsers [20,21,22] predict transitions given the input and history. Later transition-based methods are extended to handle long-range dependencies that cause nonprojective structures in exchange for increased processing time [23,24].

3. Problem in hand

Customer interactions with banks appear in the form of a free-format order document similar to Figure 1[1]. Each customer order for divergent process types (e.g., money transfers, tax payments, salary payments etc.) is received from different communication channels. Then back office operators manually process the order type, enter details of valid transactions, and send the order to the approval workflow. The automatization of this process, although still needing human approval, has a crucial impact on total operator workload in such organizations.

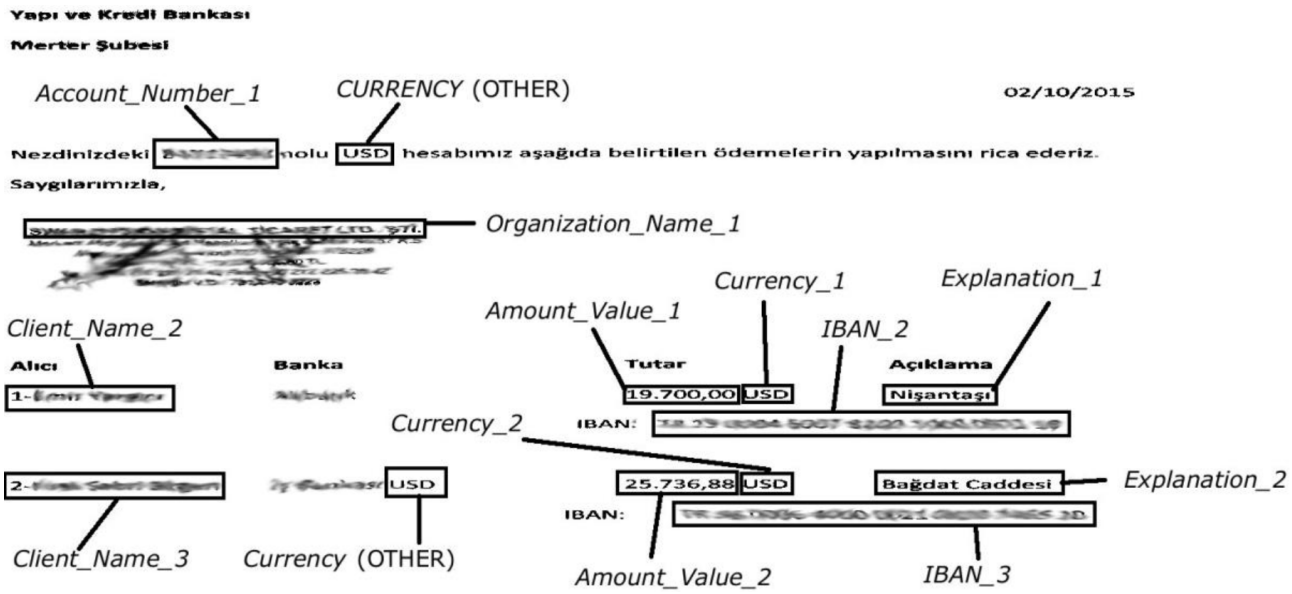

Figure 1. A sample document with one sender and two receivers. (Due to customer privacy, real values are blurred.)

This study focuses on 4 main bank process types: EFT, book-to-book money transfers, foreign currency

[1] Glossary for Figure 1 - "Nezdinizdeki": under your possession; "nolu": with number; "hesabımız aşağıda belirtilen ödemelerin yapılmasını rica ederiz": We kindly request the following payments to be processed.; "Alıcı": Receiver; "Banka": Bank; "Tutar": Amount; "Açıklama": Explanation.

transfers, and import/export payments. For most of the banks, these process types constitute 70% of all back office processes. These process types cover 4.5 million completed transactions yearly. As customers are not provided any structured form, all customer orders are unique and free-formatted.

Different from public datasets used in RE, the data in our case are composed of many complex relations and some domain specific relations. Each document contains at least one transaction (1.2 transactions on average), which consists of a sender, receiver, and process details. The complexity of the problem arises from the fact that order documents may include multiple transactions having separate or common senders related to single or multiple receivers. In addition, different process types may coexist in the same order document and low resolution of order images may lead to noisy text.

In Table 1, the first column shows the named entity types previously produced by a named entity recognition (NER) system [25], where the second column represents the transaction entity types, which are compositions of named entities. For example, $\langle AccountHolder \rangle$ has the properties 1 to 4, where $\langle Amount \rangle$ has a value and currency given with ids 5 to 6. Then we define a transaction relation as

$$r_t = \{\langle AccountHolder \rangle, \langle AccountHalder \rangle, \langle Amount \rangle, \langle Explanation \rangle\},$$

where the two $\langle AccountHolder \rangle$s represent sender and receiver, respectively.

Table 1. Named entities (properties) versus transaction entities.

ID	Named entity	Transaction entity
1	Account_Number	$\langle AccountHolder \rangle$
2	IBAN	
3	Client_Name	
4	Organization_Name	
5	Amount_Value	$\langle Amount \rangle$
6	Currency	
7	Explanation	$\langle Explanation \rangle$

Figure 1 provides a typical banking document that contains two transactions. In total, the document contains 7 transaction entities, namely 3 account holders, 2 amounts, and 2 explanations. In this sample order, the sender $\langle AccountHolder1 \rangle$ sends two different amounts ($\langle Amount1 \rangle$, $\langle Amount2 \rangle$) to the other two account holders (receivers $\langle AccountHolder2 \rangle$ and $\langle AccountHolder3 \rangle$) with different explanations ($\langle Explanation1 \rangle$ and $\langle Explanation2 \rangle$). The document contains an introductory natural language sentence and a table-like structure to denote the receivers and process details information. For clarification, the related properties for each transaction entity are represented with a unique identifier within the figure. For example, *Account_Number_1* and *Organization_Name_1* are properties of $\langle AccountHolder1 \rangle$ in the figure. This sample D may be represented as $\cup_{i=1}^{2} r_i$ where the relations are as follows:

$$r_1 = \{\langle AccountHolder1 \rangle, \langle AccountHalder2 \rangle, \langle Amount1 \rangle, \langle Explanation1 \rangle\}$$

$$r_2 = \{\langle AccountHolder1 \rangle, \langle AccountHalder3 \rangle, \langle Amount2 \rangle, \langle Explanation2 \rangle\}$$

In our sample domain, the RE task may be seen as jointly extracting both the (1) intratransaction (a.k.a., "PropertyOf" relations) and (2) intertransaction relations that exist between the two transaction entities.

4. Modeling and representation

The overall architecture (Figure 2) receives images of customer orders as input and aims to output all valid transactions within these so that they can be integrated into a banking system for approval process. ABBY FineReader Engine 11's optical character recognition (OCR) tool is used for digitizing images into text format. However, like any other OCR system, this one is also prone to recognition errors and therefore the texts are often noisy. After image-to-text conversion, the resulting noisy text is provided to a NER system [25]. As the output of the NER system, noisy text is tagged with named entities described in Table 1 and sent to the data modeling and representation module, which transforms text into a processable format for dependency parsers. After that, the formatted text is fed to the dependency-parsing module, which extracts relations between entities. Finally, these binary relations are combined into transaction relations (r_t).

Figure 2. System architecture.

5. Modeling

Traditional dependency-parsing methods aim to extract the syntactic structure of a sentence in terms of binary relations between tokens, such that collection of these relations yields to a rooted, acyclic, connected tree. In contrast to previous methods [7,10] that employ dependency trees as linguistic features or input to tree kernels, here the RE is formulated directly as a dependency-parsing problem by treating named entities as syntactic units. In Section 3, the RE task is introduced as jointly extracting intratransaction and intertransaction relations. In order to formulate the problem as a valid dependency-parsing task, a dependency grammar needs to be designed such that 1) higher-order relations can be represented with a collection of binary relations; 2) intersentential relations can be processed as a single input sequence, and 3) nested relations (intra- and intertransaction relations) can be deduced by following dependency links between entities.

For a classical syntactic dependency parser, the input is a natural language sentence composed of sequentially ordered words/tokens. For this purpose, all the detected named entities over an entire document are ordered according to their position in the document and used as tokens of the input sequence. A special punctuation node is placed to finalize the sequence. Therefore, binary dependency relations that traditionally occur between tokens become relations between named entities (Table 1, first column) in this case.

Table 2 provides the dependency relation labels used in our grammar. The first four rows represent intertransaction entity relations and the next three give the intratransaction ("PropertyOf") relations. In our modeling, all kinds of relations are treated at the same level and extracted in a single stage. In other words, the transaction entities are abstract data types and are later formed from binary relations extracted by the dependency parser. In this respect, named entities may be seen as words in a phrase of a natural language sentence and transaction entities may be seen as subtrees with their own heads and internal dependency structures. In our proposed dependency grammar, the head node of an $\langle AccountHolder \rangle$ is linked to the sentence dependency tree head (PUNCT) by either the SENDER or RECEIVER relations. Figure 3 illustrates a simple dependency graph with one receiver and one sender providing most of the possible dependency links between named entities. The head node within an $\langle AccountHolder \rangle$ (subtree) may be either an *IBAN* or an *Account_Number*. NAME_OF defines the internal dependency structure of an $\langle AccountHolder \rangle$ composed of an

Organization_Name and *IBAN*. For ease of understanding, the abstract transaction entity types are shown within dotted rectangles. One should note that although an additional transaction entity ⟨*AccountHolder*⟩ (receiver) exists, it is not shown in the figure since its conforming named entities (*Client_Name* and *Account_Number*) are not sequential.

Table 2. Dependency labels.

Name	Explanation
SENDER	From an *IBAN* or an *Account_Number* to a PUNCT node
RECEIVER	From an *IBAN* or an *Account_Number* to a PUNCT node
TRANS_EXPL	From an *Explanation* to a receiver's *IBAN* or an Account_Number
TRANS_AMOUNT	From an *Amount-Value* to a receiver's *IBAN* or an Account_Number
NAME_OF	From an *Organization_Name* or a *Client_Name to an Account_Number or an IBAN*
CURR_OF	From *Currency* to *Amount_Value*
SAME	Between duplicate entities of any type
OTHER	Between uninformative entities and PUNCT node

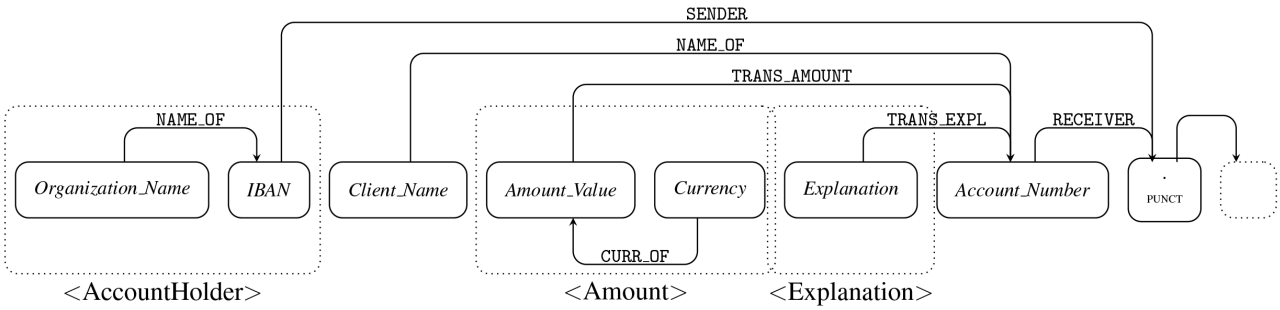

Figure 3. Abstract representation of transactions on the toy example.

Each transaction defined in the same document has distinct receivers, although they may have the same sender. Since receivers are unique to a transaction, we bind the ⟨*Explanation*⟩ and ⟨*Amount*⟩ of the transaction to the receiver's head (which should be either *IBAN* or *Account_Number* as previously explained) with the dependency relations TRANS_EXPL and TRANS_AMOUNT accordingly. The head node within an ⟨*Amount*⟩ is always an *Amount_Value* and a *Currency* entity is linked to this node via CURR_OF label.

Unfortunately, real-world documents may look more complex than Figure 3. In Figure 4, the dependency tree of a real-world document (from Figure 1) is shown. This time, not only the named entity types but also their surface forms from the original text are provided within the nodes. There may be cases where an entity is detected but is not involved in any transaction. For example, in Figure 4 there are four Currency entities where two of them are properties of a transaction ⟨*Amount*⟩ and linked to related *Amount_Value* with the relation type CURR_OF. The remaining two currency entities specify the account type (i.e. a USD account), which is not defined as a property of an ⟨*AccountHolder*⟩ in our representation scheme. In order to preserve the connectedness property of the dependency tree, we connect these entities to the PUNCT node with the label OTHER. Another case may be duplication of entities that took part in the transaction, usually caused by mentioning organization name (in the header/footer) or account number information multiple times. We connect such entities to each other with the label SAME according to their occurrence order.

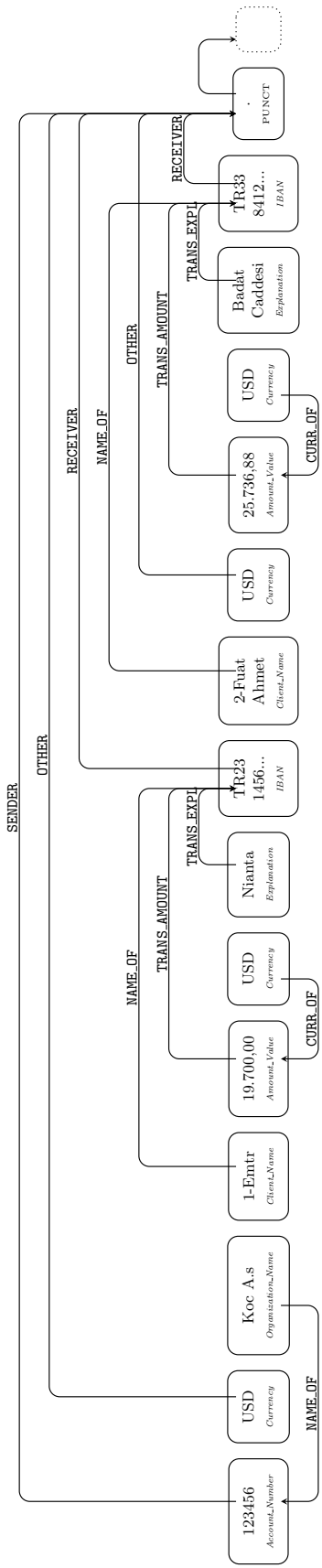

Figure 4. Dependency tree of the real-world document.

5.1. Representation for parsing

Once we have modeled the problem, the next stage is to represent documents with a representation scheme suitable for the available dependency parsers. Most dependency parsing algorithms work with a column based input format named after a CoNLL shared task [26]. Although there appeared many variations in the following years, CoNLL-style formats became a de facto standard both for treebank representations [27,28] and dependency parsers. In this format, each row represents tokens within a sentence and columns correspond to fields of tokens. Traditionally, these fields are linguistic properties such as lemma, postag, and morphological features followed by the identifier (id) of a parent node and its dependency type.

Inspired from the CoNLL scheme, we introduce 9 column/fields namely as ID, Form, NE, MINUS2, MINUS1, PLUS1, PLUS2, HEAD, and DEPREL. The representation of our real-world example (Figure 1) is provided in Table 3 (MINUS2 and PLUS2 is not shown due to space constraints). Similar to parts-of-speech tags crucial to traditional dependency parsers, NE columns define the type of our syntactic units (named entities). Following the relation-extraction literature that made great use of surrounding/context words, words are included in a context window of 4 as input features in minus and plus columns. In order to address data sparsity of context columns, entity types are used instead of word surface forms when available.

Table 3. Input data representation in columnar format.

ID	Form	NE	MINUS1	PLUS1	HEAD	DEPREL
1	123456	*Account*	Nezdlnîzdeki	nolu	15	SENDER
2	USD	*Currency*	nolu	hesabımız	15	OTHER
3	Koc A.s	*Organization*	Saygılarımızla	Merkez	1	NAME_OF
4	1-Emtr	*Client*	Alıcı	Yargıcı	8	NAME_OF
5	19.700,00	*Amount*	Tutar	*Currency*	8	TRANS_AMOUNT
6	USD	*Currency*	*Amount*	Açıklama	5	CUR_OF
7	Nişantaşı	*Expl*	Açıklama	*IBAN*	8	TRANS_EXPL
8	TR231456..	*IBAN*	19	Alıcı	15	RECEİVER
9	2-Fuat Ahmet	*Client*	Alıcı	Banka	14	NAME_OF
10	USD	*Currency*	Bank	Tutar	15	OTHER
11	25.736,88	*Amount*	Tutar	*Currency*	14	TRANS_AMOUNT
12	USD	*Currency*	*Amount*	Açıklama	11	CUR_OF
13	Bağdat Cadde	*Expl*	Açıklama	*IBAN*	14	TRANS_EXPL
14	TR338412..	*IBAN*	*IBAN*	_	15	RECEIVER
15	.	PUNCT	*IBAN*	_	0	ROOT

Formatting documents with a well-known representation scheme provides the possibility of using many different preexisting parsers. In this article, we chose a well-known open-source transition-based parser, Maltparser [29], which has nine built-in deterministic parsing algorithms and easily modifiable feature models where users can define new features of arbitrary complexity.

The learning problem in transition-based parsing is to induce a classifier for predicting the next transition given a feature representation of the current parser configuration. SVMs are widely used algorithms for this task [20,22] and are reported to perform better than most other machine learning algorithms [29]. MaltParser offers two built-in machine learning packages for SVM: LIBSVM and LIBLINEAR. The difference between these two algorithms can be summarized as follows: LIBSVM employs the kernel mechanism, which has the ability to implicitly add conjoined features, whereas LIBLINEAR expects an explicit definition of all feature combinations. The capability of modeling implicit interactions between features greatly eases our feature

engineering process. LIBSVM is considered more memory-efficient than LIBLINEAR since it does not store weight vectors. LIBLINEAR executes in O(k) both for training and parsing, where k is the number of features, while LIBSVM's worst-case running time can be O(n), where n is the number of training instances. Therefore, LIBLINEAR can be considered much faster than LIBSVM.

6. Experimental results and discussions

In this section, we first explain the annotation process and the resulting transaction dataset and present our experiments on different deterministic parsing algorithms.

6.1. Dataset and evaluation methodology

First, the human annotators were asked to label the named entities and their related transaction entities such that entities of the same transaction will have the same unique identifier. For example, if an *Amount_Value* and a *Currency* are properties of the same $\langle Amount \rangle$, they will be attached a unique number i as in Figure 1 (similar for other $\langle TransactionEntity \rangle$s). Then annotators fill the transaction information (r_t) for each document. Annotation was performed by two trained annotators, where the first annotator labeled the document and the second one (a more experienced annotator), reviewed the document, correcting annotation mistakes. The resulting dataset contains 11741 dependency labels from 1603 documents. Out of these, 1141 of the documents contain nonprojective dependencies[2] while 21 documents contain multiple transactions.

A 10-fold cross validation was used for evaluation and LIBSVM was used as the learning algorithm for all parsing models. Considering that our features are highly interactive (e.g., *NE-FORM, NE-MINUS1, MINUS1-FORM*), the capability of modeling implicit feature combinations has been the primary reason for choosing LIBSVM. However, the best model was also tested with LIBLINEAR and provided the results in the following section.

The most commonly used measures to evaluate dependency parsing results are attachment scores, known as a label attachment score (LAS) and unlabeled attachment score (UAS), and label scores known as label accuracy (LA). LAS is calculated as percentage of tokens for which a system has predicted the correct HEAD and DEPREL, whereas UAS is percentage of tokens with correct HEAD. Similarly, LA is measured as ratio of tokens with correct DEPREL. The experiments were evaluated on different parsing algorithms by the use of the above measurements and the precision, recall and F-measure on the detection of specific relation types are also provided.

6.2. Experiments and discussions

A rule-based approach currently in use in a real-world scenario (Section 3) was used as a baseline system. This algorithm basically attaches the first encountered *IBAN* or *Account_Number* as SENDER and remaining ones as RECEIVERS. Similarly, the first *Amount_Value*s and *Explanation*s are assigned to the first *IBAN* or *Account_Number*, while others are assigned to remaining receivers. *Organization_Name* and *Client_Name* are attached to *IBAN* or *Account_Number* with respect to their order. Likewise, *Currency*s are adjoined to *Amount_Value*s according to their order. A 48.22% LAS was obtained using this baseline algorithm.

First, we experiment with standard Nivre algorithms [20]: nivreeager and nivrestandard with the default feature models provided by MaltParser. These methods perform in linear time but are limited to projective dependency structures. Since 71% of our dataset contains nonprojective samples, we then focus on parsing

[2] A tree structure is nonprojective if there exists crossing dependency edges.

algorithms that are capable of nonprojectivity, such as covnonproj [20], stackeager [23], and stacklazy [24]. According to Table 4, stackeager and stacklazy[3] methods were the highest scoring algorithms, with 91.84% and 91.67% LAS scores.

Table 4. Evaluation of different parsing algorithms.

Algorithm	LA	UAS	LAS
nivreeager	55.43	54.07	53.6
nivrestandard	54.71	53.54	53.14
covproj	56.59	55.56	55.37
covnonproj	92.63	91.54	90.96
stacklazy	93.65	92.23	91.67
stackeager	93.81	92.33	91.84

Figure 5 shows the experimental feature models for the best-performing algorithm, stackeager. Stack algorithms use a stack to keep partially processed tokens, represented with σ; a buffer (list of inputs) to keep all nodes that have been on stack, shown with τ; and another buffer (lookahead) containing all nodes that have not been on stack, denoted with δ. σ_i represents the $(i+1)th$ entity from the top of the stack. τ_i and δ_i denote the elements within the input buffers similarly. A base model that uses only FORM and NE features (shown with base tab in Figure 5) provides a LAS score of 88.58%. FM#2, which includes all context features (M1, M2, P1, P2) of σ_0 achieves 90.21% LAS. In FM#3, we have included the surrounding words of the entity in σ_1 and in FM#4 a smaller window (M1, P1) of τ_0 and τ_1 are employed. While FM#3 increased the score by 0.80%, our best feature model (FM#4) achieved 91.84%. We also compared the best feature model by using LIBLINEAR. This model achieved a LAS score of 91.14%, although performing faster (45 s versus 11 min with 10-fold CV on an Intel Core I5-3470 CPU @ 3.20 GHz machine with 12 GB of RAM.

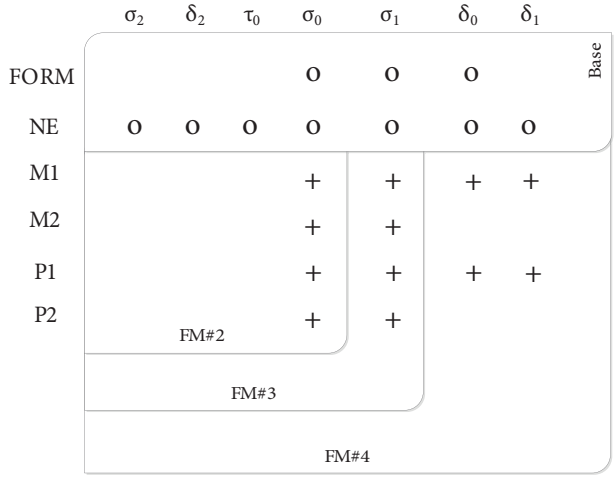

Figure 5. Feature models (FM) used for stack algorithms.

In Table 5, comparison of baseline and best-performing parsing algorithm is given. According to this table, while SENDER and RECEIVER have the highest precision and recall scores, followed by TRANS AMOUNT,

[3] Stack algorithms change the order of nodes with a SWAP operation, turning nonprojective sentences into projective ones in expected linear time, while covnonproj is a list-based quadratic time algorithm thath tries to link each new node to each preceding node.

TRANS EXPL and OTHER are the most difficult labels to detect. One of the reasons is the low occurrence rates of the latter two in our dataset, i.e. there are not enough training samples to teach these labels. Another important reason is their "free" form. The OTHER dependency label can originate from any type of entity that is not part of a transaction, which may confuse the classifier. The baseline system handles SENDER and RECEIVER relations relatively decently; in general, however, it performs very poorly with other relation types. As we can see from Table 5, the baseline system does not support OTHER/SAME complex relation types and has a total labeled F1 score of 48.99%.

Table 5. Evaluation based on relation types.

		Baseline system			Best system		
Dependency label	#	P	R	F1	P	R	F1
CURR_OF	1300	43.50	56.08	48.99	94.21	96.46	95.32
NAME_OF	2475	32.57	32.89	32.73	85.11	88.24	86.65
OTHER	164	0.00	0.00	0.00	15.60	20.73	17.80
RECEIVER	1635	76.19	96.27	85.06	97.87	98.17	98.02
SAME	2688	0.00	0.00	0.00	94.37	87.28	90.68
SENDER	1613	90.52	89.96	90.24	96.88	98.26	97.57
TRANS_AMOUNT	1635	60.45	56.94	58.65	96.85	95.96	96.41
TRANS_EXPL	231	69.66	70.56	70.11	84.43	89.18	86.74
TOTAL	11741	49.79	48.22	48.99	91.84	91.84	91.84

In order to investigate the impact of data size on the performance of our system, we designed an experiment that iteratively increases the training set size from 1- to 9-fold and measures the performance over the same 1-fold test set. The results (Figure 6) reveal that the performance increases drastically by the addition of the initial folds but converges late, designating that size of the used training data is near a saturation point.

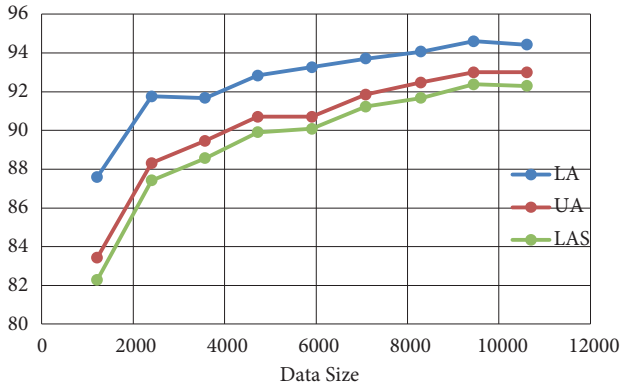

Figure 6. The impact of training data size.

The use of the introduced relation extraction approach in the banking system improves customer satisfaction and overall customer experience due to fast response times and eliminated reworking caused by error-prone manual operations. After the system integration, 20% of the manual workforce is saved. This is accomplished by digitalizing 53% of the process workflow. As a result, overall cycle time of target processes is reduced significantly. Book-to-book money transfer cycle time is decreased from 45 min to 13 min and EFT cycle time is decreased from 31 min to 19 min. On a yearly basis, 53% digitalization means 3.2 million transactions can be

completed without a manual workforce, since there are roughly 6.5 million transactions arriving each year. As a future study, we aim to increase the coverage of the system with different process types like bill payments, automatic bill payment orders, and credit card payments, which include different entities.

7. Conclusion

In this paper, we have proposed a language-independent relation extraction method based on the idea of dependency-parsing, which can handle nested, intersentential, and higher-order relations. Unlike previous RE literature that use syntactic trees for input to their models, RE was formulated directly as a one-shot dependency-parsing problem that does not require high-level preprocessing of input text. By designing a simple dependency grammar for a sequence of named entities, we were able to use existing dependency analysis tools and parsing algorithms. This method was evaluated using banking transaction documents, where each document contained at least one 4-ary relation as {Sender, Receiver, Amount, Explanation} , and obtained a LAS of 91.84%.

The encouraging results of this novel application suggest that a problem that contains such complex relations from any domain may be considered within dependency-parsing framework and solved with well-established parsing algorithms. Due to the lack of labeled data with nested, higher-order, and document-level relations, this method could not be evaluated with other datasets. However, we are convinced that our application has a great potential for understanding user intentions from free formatted text, such as application forms, petitions, customer orders, or company reports.

A major drawback of this method is the requirement of a labeled dataset. However, user intentions are usually parsed from domain-specific documents, e.g., finance and law, which reduces the data sparsity and the need for high amounts of training data. Another drawback may be designing a separate language (dependency grammar) for each intention/relation. However, the template grammar introduced in this work contains the basic elements of such grammars and can be used as a guide for new intention types. For the evaluation, only the basic, language-independent features were used to avoid feature engineering. However, further optimization of the used features and the employment of language-specific features [30] deserve more investigation in the light of previous studies [31] in classical dependency parsing.

To conclude, the contributions of this paper can be summarized as (1) proposing a language-independent and syntax-agnostic method for addressing the extraction of higher-level and nested relations at document granularity, (2) handling noisy, ungrammatical, and unstructured text, and (3) reducing the overall cycle time and saving 20% of the manual workforce.

References

[1] Lee C, Hwang YG, Jang MG. Fine-grained named entity recognition and relation extraction for question answering. In: 30th ACM SIGIR Conference on Research and Development in Information Retrieval; July 23–27 July 2007; Amsterdam, the Netherlands. New York, NY, USA: ACM. pp. 799-800.

[2] Arendarenko E, Kakkonen T. Ontology-based information and event extraction for business intelligence. In: Ramsay A, Agre G, editors. Artificial Intelligence: Methodology, Systems, and Applications. AIMSA 2012. Lecture Notes in Computer Science, vol 7557. Berlin, Germany: Springer. pp. 89-102.

[3] Aramaki E, Miura Y, Tonoike M, Ohkuma T, Masuichi H, Waki K, Ohe K. Extraction of adverse drug effects from clinical records. St Heal T 2010; 160: 739-743.

[4] Rink B, Harabagiu S, Roberts K. Automatic extraction of relations between medical concepts in clinical texts. J Am Med Inform Assn 2011; 18: 594-600.

[5] Bollacker K, Evans C, Paritosh P, Sturge T, Taylor J. Freebase: a collaboratively created graph database for structuring human knowledge. In: Proceedings of the ACM SIGMOD International Conference on Management of Data; 9–12 June 2008; Vancouver, BC, Canada. New York, NY, USA: ACM. pp. 1247-1250.

[6] Kambhatla N. Combining lexical, syntactic, and semantic features with maximum entropy models for extracting relations. In: Proceedings of the ACL 2004 on Interactive Poster and Demonstration Sessions; 21–26 July 2004; Barcelona, Spain. Stroudsburg, PA, USA: Association for Computational Linguistics. p. 22.

[7] GuoDong Z, Jian S, Jie Z, Min Z. Exploring various knowledge in relation extraction. In: ACL '05 Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics; 25–30 June 2005; Ann Arbor, MI, USA. Stroudsburg, PA, USA: Association for Computational Linguistics. pp. 427-434.

[8] Bunescu RC, Mooney RJ. Subsequence kernels for relation extraction. In: Buntine W, Grobelnik M, Mladenić D, Shawe-Taylor J, editors. Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2009. Lecture Notes in Computer Science, vol 5782. Berlin, Germany: Springer, p. 171.

[9] Zelenko D, Aone C, Richardella A. Kernel methods for relation extraction. J Mach Learn Res 2003; 3: 1083-1106.

[10] Culotta A, Sorensen J. Dependency tree kernels for relation extraction. In: ACL '04 Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics; 21–26 July 2004; Barcelona, Spain. Stroudsburg, PA, USA: Association for Computational Linguistics. p. 423.

[11] Zeng D, Liu K, Lai S, Zhou G, Zhao J. Relation classification via convolutional deep neural network. In: COLING, 2014; pp. 2335-2344.

[12] Lin Y, Shen S, Liu Z, Luan H, Sun M. Neural relation extraction with selective attention over instances. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics; 7–12 August 2016; Berlin, Germany. Stroudsburg, PA, USA: Association for Computational Linguistics. pp. 2124-2133.

[13] Miller S, Fox H, Ramshaw L, Weischedel R. A novel use of statistical parsing to extract information from text. In: ACL, 2000; pp. 226-233.

[14] McClosky D, Surdeanu M, Manning CD. Event extraction as dependency parsing. In: HLT '11 Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1; 19–24 June 2011; Portland, OR, USA. Stroudsburg, PA, USA: Association for Computational Linguistics. pp. 1626-1635.

[15] Finkel JR, Manning CD. Nested named entity recognition. In: EMNLP; 2009. pp. 141-150.

[16] Doddington GR, Mitchell A, Przybocki MA, Ramshaw LA, Strassel S, Weischedel RM. The ACE program – tasks, data, and evaluation. In: Proceedings of The Fourth International Conference on Language Resources and Evaluation; 26–28 May 2004; Lisbon, Portugal. p. 1.

[17] Hendrickx I, Kim SN, Kozareva Z, Nakov P, Ó Séaghdha D, Padó S, Pennacchiotti M, Romano L, Szpakowicz S. Semeval-2010 task 8: Multi-way classification of semantic relations between pairs of nominals. In: SEW; 2009. pp. 94-99.

[18] Tesnière L. Eléments de syntaxe structurale: Librairie C. Klincksieck; 1959.

[19] McDonald R, Pereira F, Ribarov K, Hajic J. Non-projective dependency parsing using spanning tree algorithms. In: HLT '05 Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing; 6-8 October 2005; Vancouver, BC, Canada. Stroudsburg, PA, USA: Association for Computational Linguistics. pp. 523-530.

[20] Nivre J. Algorithms for deterministic incremental dependency parsing. Comput Linguist 2008; 34: 513-553.

[21] Kudo T, Matsumoto Y. Japanese dependency analysis using cascaded chunking. In: CoNLL, 2002; pp. 1-7.

[22] Yamada H, Matsumoto Y. Statistical dependency analysis with support vector machines. In IWPT, 2003; pp. 195-206.

[23] Nivre J. Non-projective dependency parsing in expected linear time. In: Proceedings of the 47th Annual Meeting of the Association for Computational Linguistics and the 4th International Joint Conference on Natural Language Processing of the AFNL; 2–7 August 2009; Singapore. Stroudsburg, PA, USA: Association for Computational Linguistics. pp. 351-359.

[24] Nivre J, Kuhlmann M, Hall J. An improved oracle for dependency parsing with online reordering. In: Proceedings of the 11th International Conference on Parsing Technologies; October 2009; Paris, France. Stroudsburg, PA, USA: Association for Computational Linguistics. pp. 73-76.

[25] Emekligil E, Arslan S, Agin O. A bank information extraction system based on named entity recognition with CRFs from noisy customer order texts in Turkish. In: Proceedings of the Knowledge Engineering and Semantic Web Conference; 2016; Prague, Czech Republic. Cham, Switzerland: Springer. pp. 93-102.

[26] Buchholz S, Erwin M. CoNLL-X shared task on multilingual dependency parsing. In: CoNLL-X '06 Proceedings of the Tenth Conference on Computational Natural Language Learning; 8–9 June 2006; New York, NY, USA. Stroudsburg, PA, USA: Association for Computational Linguistics.

[27] Nivre J, de Marneffe MC, Ginter F, Goldberg Y, Hajic J, Manning CD, McDonald R, Petrov S, Pyysalo S, Silveira N et al. Universal dependencies v1: A multilingual treebank collection. In: Proceedings of the Tenth International Conference on Language Resources and Evaluation; 23–28 May 2016; Portorož, Slovenia. pp. 1659-1666.

[28] Buchholz S, Marsi E. CoNLL-X shared task on multilingual dependency parsing In: CoNLL-X '06 Proceedings of the Tenth Conference on Computational Natural Language Learning; 8–9 June 2006; New York, NY, USA. Stroudsburg, PA, USA: Association for Computational Linguistics. pp. 149-164.

[29] Nivre J, Hall J, Nilsson J, Chanev A, Eryiğit G, Kübler S, Marinov S, Marsi E. MaltParser: A language-independent system for data-driven dependency parsing. Nat Lang Eng 2007; 13: 95-135.

[30] Tsarfaty R, Seddah D, Kübler S, Nivre J. Parsing morphologically rich languages: Introduction to the special issue. Comput Linguist 2013; 39: 15-22.

[31] Ballesteros M, Nivre J. MaltOptimizer: Fast and effective parser optimization. Nat Lang Eng 2016; 22: 187-213.