Joint Extraction of Entities and Relations Based on a Novel Graph Scheme

Shaolei Wang¹, Yue Zhang*², Wanxiang Che¹, Ting Liu¹
¹ Center for Social Computing and Information Retrieval, Harbin Institute of Technology, China
² Singapore University of Technology and Design
{slwang, car, tliu}@ir.hit.edu.cn, yue_zhang@sutd.edu.sg

Abstract
Both entity and relation extraction can benefit from being performed jointly, allowing each task to correct the errors of the other. Most existing neural joint methods extract entities and relations separately and achieve joint learning through parameter sharing, leading to a drawback that information between output entities and relations cannot be fully exploited. In this paper, we convert the joint task into a directed graph by designing a novel graph scheme and propose a transition-based approach to generate the directed graph incrementally, which can achieve joint learning through joint decoding. Our method can model underlying dependencies not only between entities and relations, but also between relations. Experiments on New York Times (NYT) corpora show that our approach outperforms the state-of-the-art methods.

1 Introduction
Extraction of entities and relations is a fundamental task of information extraction (IE). An example is shown in Figure 1, where the input is unstructured texts and the output includes entities and their semantic relations. There are strong connections between entities and relations, and also between relation labels in a sentence. For example, a Live_In relation suggests Person and Location entities, and vice versa. The Live_In relation (between “John” and “California”) can be inferred from the Live_In (between “John” and “Los Angeles”) and Loc_In (between “Los Angeles” and “California”) relations.

The task has been traditionally solved as a pipeline of two separate sub-tasks: entity recognition [Nadeau and Sekine, 2007] and relation extraction [Zhou et al., 2005]. This separation neglects the relevance between these two sub-tasks. Joint extraction of entities and relations can integrate information of entities and relations, and has achieved better results on this task. Joint models have been investigated using both statistical methods [Ren et al., 2017; Miwa and Sasaki, 2014; Li and Ji, 2014] and neural methods [Zheng et al., 2017; Katiyar and Cardie, 2017; Miwa and Bansal, 2016]. The performances of statistical models [Miwa and Sasaki, 2014; Li and Ji, 2014] heavily rely on complicated feature engineering and it is difficult to exploit global features.

Neural methods, in contrast, automatically learn non-local features by exploiting recurrent neural networks for learning sentence-level representations and have given the state-of-the-art results. However, most existing neural models [Katiyar and Cardie, 2017; Miwa and Bansal, 2016] extract entities and relations separately, achieving joint learning only through parameter sharing, but not joint decoding. This leads to a drawback that information between output entities and relations cannot be fully exploited, since no explicit features are used to model output-output dependencies. Zheng et al. [2017] is the only exception, designing a novel tagging scheme and converting the joint extraction task to a tagging problem. In their joint model, information of entities and relations is integrated into a unified tagging scheme and can be fully exploited. However, due to the transformation into a tagging task, the method only indirectly captures output structural correspondences, and is incapable of identifying overlapping relations (e.g. one entity can only have at most one relation).

To address this issue, we convert the joint task into a directed graph by designing a novel graph scheme, solved using a transition-based parsing framework [Zhang and Clark, 2011]. Different from traditional parsing tasks, nodes in our output structures may have multiple or no heads, as shown in Figure 1. We propose a novel transition system, which is a variant of the list-based arc-eager algorithm for non-projective tree parsing [Choi and McCallum, 2013]. By incrementally integrating entity information and their corresponding relation information, our method can model underlying dependencies not only between entities and relations, but also between relations. One challenge for designing a transition-based parsing system is the representation of parsing states (i.e. configurations), based on which the transition actions are disambiguated. We borrow the idea of neural parsing [Dyer et al., 2015; Kiperwasser and Goldberg, 2016], designing a
special recursive neural network to model underlying entity-relation and relation-relation dependencies. We also use a Bi-LSTM to represent each sentence token to capture richer contextual information. The main contributions of this work are concluded as follows:

- We propose an intuitive graph scheme to jointly represent entities and relations, so that end to end relation extraction can be easily transformed into a parsing-like task.

- Based on our graph scheme, we propose a novel transition system to generate the directed graph. In addition, we design a special recursive neural network to better model underlying entity-relation and relation-relation dependencies.

- We conduct our experiments on NewYork Times (NYT) corpora and the results show our method outperforms the state-of-the-art end-to-end methods.

The code is released¹.

2 Problem Definition

2.1 Baseline: Tagging Scheme

Zheng et al. [2017] treat the joint extraction task as a sequence labeling problem, proposing a novel tagging scheme. Figure 2 is an example of the tagging scheme. Tag “O” means that the corresponding word is independent of extracted entities and relations. In addition to “O”, the other tags consist of three parts: the word position in the entity, the relation type, and the relation role. It uses the “BIES” (Begin, Inside, End, Single) signs to represent the position information of a word in the entity. The relation type information is obtained from a predefined set of relations. The relation role information is represented by the numbers “1” and “2”, where “1” means that the word belongs to the first entity in the relation and “2” means that the word belongs to the second entity. As shown in Figure 2, “Los” is the first word of entity “Los Angeles” belonging to the second element of relation Live_In, so its tag is “B-LI-2”. The other entity “John”, which is the first element of relation Live_In, is labeled as “S-LI-1”.

Based on this tagging scheme, Zheng et al. [2017] develop an end-to-end model with a biased loss function for the sequence labeling problem. State-of-the-art results are achieved thanks to the association between related entities in joint decoding. However, the method is incapable of identifying the overlapping relations. For instance, the sentence in Figure 1 contains three relations, in which every entity has two relations with other entities. But only one of the relations can be extracted under the tagging scheme.

¹https://github.com/hitssl/joint-entity-relation

<table>
<thead>
<tr>
<th>Transitions</th>
<th>Change of State</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT1-REDUCE</td>
<td>({\sigma[i^<em>], \delta, e, [j^</em>[\beta], R, E}</td>
</tr>
<tr>
<td></td>
<td>(\sigma[i^<em>], \delta, e, [j^</em>[\beta], R, E, {i^* \leftrightarrow j^*}, E)</td>
</tr>
<tr>
<td>RIGHT1-SHIFT</td>
<td>(\sigma[i^<em>], \delta, e, [j^</em>[\beta], R, E)</td>
</tr>
<tr>
<td>NO-SHIFT</td>
<td>(\sigma[i^<em>], \delta, e, [j^</em>[\beta], R, E)</td>
</tr>
<tr>
<td>NO-REDUCE</td>
<td>(\sigma[i^<em>], \delta, e, [j^</em>[\beta], R, E)</td>
</tr>
<tr>
<td>LEFT1-PASS</td>
<td>(\sigma, \delta[\beta], e, [j^*[\beta], R, E)</td>
</tr>
<tr>
<td>RIGHT1-PASS</td>
<td>(\sigma, \delta[\beta], e, [j^*[\beta], R, E)</td>
</tr>
<tr>
<td>NO-PASS</td>
<td>(\sigma, \delta[\beta], e, [j^*[\beta], R, E)</td>
</tr>
<tr>
<td>O-DELETE</td>
<td>(\sigma[i^*], \delta, e, [j[\beta], R, E)</td>
</tr>
<tr>
<td>GEN-SHIFT</td>
<td>(\sigma[i^*], \delta, e, [j[\beta], R, E)</td>
</tr>
<tr>
<td>GEN-NER(y)</td>
<td>(\sigma[i^*], \delta, e, [j[\beta], R, E)</td>
</tr>
</tbody>
</table>

Table 1: Transition actions, * indicates an entity.

2.2 The Graph Scheme

Instead of label sequences, we transform entity mentions and their relations into a directed graph. The nodes in the graph correspond to words in the input sentence. The directed arcs are broadly categorized into: 1) entity arcs that represent internal structures of entities; 2) relation arcs that represent relations between entities, where head node means the first element of relation and modifier node means the second element of relation. To cope with the overlapping relations, the node in our directed graph can have multiple heads, which is different from traditional constituent parsing or dependency parsing graph. Figure 1 illustrates our graph representation, where the input sentence contains: 1) three entities, which are converted into corresponding green arcs with entity labels; and 2) three relations, which are converted into corresponding blue arcs with relation labels. Besides, the other words irrelevant to the final result have no corresponding arcs.

3 Method

3.1 Transition System

We propose a novel neural transition-based method, inspired by the list-based arc-eager algorithm [Choi and McCallum, 2013]. There are two types of transition actions: 1) entity actions, which are used to recognize entities; 2) relation actions, which are used to recognize relations between entities.
Formally, we use a tuple \((\sigma, \delta, e, \beta, R, E)\) to represent each state, where \(\sigma\) is a stack holding processed entities, \(\delta\) is a stack holding entities that are popped out of \(E\) but will be pushed back in the future, \(e\) is a stack storing the partial entity chunk, and \(\beta\) is a buffer holding unprocessed words. \(R\) is a set of relation arcs. \(E\) is a set of entity arcs. We use an index \(i\) to represent word \(w_i\) and entity \(e_i\), respectively. \(A\) is used to store the action history.

The set of actions is shown in Table 1. The first seven actions are used to generate relations, and the last three actions are used to generate entities. In particular, LEFT\(_T\)-REDUCE adds a relation arc with label \(l\) from \(e_j\) to \(e_i\), and pops \(e_i\) out of \(E\). RIGHT\(_T\)-SHIFT adds a relation arc with label \(l\) from \(e_i\) to \(e_j\), and pushes all entities in \(\delta\) and \(e_j\) into \(\sigma\). NO-SHIFT pushes all entities in \(\delta\) and \(e_j\) into \(\sigma\). NO-REDUCE pops \(e_i\) out of \(E\). LEFT\(_T\)-PASS adds a relation arc with label \(l\) from \(e_j\) to \(e_i\), and moves \(e_j\) to the front of \(\delta\). RIGHT\(_T\)-PASS adds a relation arc with label \(l\) from \(e_i\) to \(e_j\), and moves \(e_i\) to the front of \(\delta\). NO-PASS simply moves \(e_i\) to the front of \(\delta\). \((i^* \rightarrow j^*)\) is used to denote a relation arc from \(e_i\) to \(e_j\) with label \(l\). \((i^* \rightarrow j^*\) and \((i^* \rightarrow j^*)\) indicate that \(e_i\) is a head and an ancestor of \(e_j\) respectively. Note that all the relation actions are forbidden when the top element of \(\beta\) is a word. O-DELETE pops \(w_j\) out of \(\beta\). GEN-SHIFT moves \(w_j\) from \(\beta\) to \(e\). GEN-NER\((y)\) pops all items from the top of \(e\) creating a “chunk”, labels this with label \(y\), pushes a representation of this chunk onto \(\beta\), and an entity is added to \(E\). All the entity actions are forbidden when the top element of \(\beta\) is an entity.

Each action needs to satisfy certain preconditions to ensure the properties of a well-formed directed graph of entities and relations, as described in Table 2. To produce arcs pointing to entities with multiple heads, we design the preconditions of LEFT\(_T\)-* and RIGHT\(_T\)-* so that the dependency between a head and its modifier can be generated even if the modifier already has a head. Furthermore, we want to confirm that all heads and children of a word are found before the word is reduced. To this end, we set the head confirmation in the precondition of \(*\)-REDUCE to make sure no extra head of \(e_i\) is in the buffer \(\beta\).

Table 2 shows the sequence of state transitions given the sentence in Figure 1. The initial state is \([0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0]\), while the terminal state is \((\sigma, \delta, [0, 0, R, E])\). Transition actions are generated by consulting the gold-standard graph during training and a neural network classifier during decoding.
where \( E^* \) is the best output entities, and \( R^* \) is the best relations. Thus the extraction of entities and relations are merged in one transition-based system. To label a new input sequence at test time, the maximum probability action is chosen greedily until the algorithm reaches a termination state.

### 3.3 Input Representation

Two neural layers are used to represent the input, where the bottom layer is token embedding and the next layer is a Bi-LSTM layer to capture richer contextual information.

**Token Encoding** We use two vectors to represent each input token \( t_i \): a learned word embedding \( w_i \) and a fixed word embedding \( \tilde{w}_i \). The two vectors are concatenated, transformed by a matrix \( V \) and fed to a rectified layer to learn a feature combination:

\[
x_i = \max(0, V[\tilde{w}; w] + b),
\]

**Bi-LSTM Encoding** Given the token embeddings of an input sequence \( x = \{x_1, \ldots, x_n\} \), a bidirectional LSTM is used to process the sequence in both directions with two separate LSTM layers. The forward LSTM layer \( \overrightarrow{h}_t \) encodes the input sequence from \( x_1 \) to \( x_n \). In the similar way, the backward LSTM layer \( \overleftarrow{h}_t \) will encode the input sequence from \( x_n \) to \( x_1 \). We then concatenate \( \overrightarrow{h}_t \) and \( \overleftarrow{h}_t \) to represent word \( t \)’s encoding information, denoted as \( h_t = [\overrightarrow{h}_t, \overleftarrow{h}_t] \). Finally, the Bi-LSTM Embedding of the input sequence will be sent to the above structures of state representation.

### 3.4 State Representation

Shown in Figure 3, for better capturing non-local context information, we use stack LSTM [Dyer et al., 2015] to represent different components of each state. For a conventional LSTM, new inputs are always added in the right-most position; but in a stack LSTM, the current location of a stack pointer determines which cell in the LSTM provides \( c_{t-1} \) and \( h_{t-1} \) when computing the new memory cell contents. In order to move the stack pointer from the right-most position, the stack LSTM provides a \( pop \) operation which moves the stack pointer to the previous element. Thus, the stack-LSTM can be understood as a stack implemented so that contents are never overwritten. By querying the output vector to which the stack pointer points, a continuous-space “summary” of the current stack configuration is available. As shown in Table 3, once a \( RIGHT-PASS \) action is taken (state 12), the \( pop \) operation will be taken to move the stack pointer from position of 5\(^{st}\) to 1\(^{st}\). Once a \( RIGHT-SHIFT \) action is taken (state 13), the elements in \( \delta \) and top of \( \beta \) will be added in the position that the stack pointer points (1\(^{st}\)) in order.

### 3.5 Composition Functions

To model underlying entity-relation and relation-relation dependencies, our method incrementally integrates the entity information and corresponding relation information into the transition-based model.

**Entity Chunks** When GEN-NER(\( y \)) is executed, the algorithm shifts the sequence of words on \( e \) to the top of \( \beta \) as a single completed chunk. To compute an embedding of this sequence, we run a bidirectional LSTM over the embeddings of its constituent words together with the chunk type (i.e., \( y \)). This function is denoted as \( g(u, \ldots, v, r_y) \), where \( r_y \) is a learned embedding of a label type. Thus, \( \beta \) contains a single vector representation for each labeled entity chunk.

**Relation Labels** Recursive neural network models enable complex phrases to be represented compositionally in terms of their components and relations [Dyer et al., 2015]. Given a directed relation arc, which points from a head node \( h \) to a modifier node \( m \), we combine both head-modifier pair and
modifier-head pair, and use the combinations to update the embeddings of head node and modifier node separately. Formally, for the head-modifier pair, we have

\[ c = \tanh(W^h[H; M; R]) + e^h \]

where \( W^h \) is a learned parameter matrix, \( H \) is neural embedding of the head entity, \( M \) is neural embedding of the modifier entity, \( R \) is vector embedding of the relation, \( e^h \) is a bias term.

Similarly, for the modifier-head pair, we have

\[ c = \tanh(W^f[M; H; R]) + e^f \]

where \( W^f \) is a learned parameter matrix, \( e^f \) is a bias term, and the others are the same with head-modifier pair.

To simplify the parameterization of our composition function, we combine the pairs one at a time, building up more complicated structures in the order they are “reduced” in the model. Figure 4 shows an example when updating the entity “Los Angeles”, where the Live_In relation is generated firstly.

4 Experiments

4.1 Experimental settings

Dataset To directly compare with Zheng et al. [2017], we use the public dataset NYT2 as our main data set, which is produced by distant supervision [Ren et al., 2017]. The training data set is obtained by means of distant supervision methods without manually labeling and contains 353k triplets in total. While the test set is manually labeled and contains 3,880 triplets.

Evaluation Metrics We adopt standard Precision (Prec), Recall (Rec) and F1 score to evaluate the model. The labels of entity types are not considered when computing the final F1-score [Ren et al., 2017; Zheng et al., 2017]. In other words, a triplet is regarded as correct when its relation type and head offsets of the two corresponding entities are both correct. We follow Zheng et al. [2017], creating a validation set by randomly sampling 10% data from test set and use the remaining data as evaluation.

4.2 Hyperparameters and Training Details

Given a set of training data, the training goal is to maximize the likelihood of each gold action given the current model state. We update all model parameters by backpropagation using stochastic gradient descent (SGD) with a learning rate of 0.01 and gradient clipping at 5.0. Following Dyer et al. [2015], we use a variant of the skip n-gram model, namely structured skip n-gram [Ling et al., 2015], to create word embeddings. We have 2 hidden layers in our network and the dimensionality of the hidden units is 100.

4.3 Experimental Results

Baselines We compare our method with several state-of-the-art extraction methods, which can be divided into the following categories: the pipelined methods, the jointly extracting methods, and the end-to-end methods. For the pipelined methods, the NER results are obtained by Ren et al. [2017], then several classical relation classification methods are applied to detect the relations. These methods include: (1) DS-logistic [Mintz et al., 2009] is a distant supervised and feature based method, which combines the advantages of supervised IE and unsupervised IE features; (2) LINE [Tang et al., 2015] is a network embedding method, which is suitable for arbitrary types of information networks; (3) FCM [Gormley et al., 2015] is a compositional model that combines lexicalized linguistic context and word embeddings for relation extraction.

The joint methods are listed as follows: (4) DS-Joint [Li and Ji, 2014] is a supervised method, which jointly extracts entities and relations using structured perceptron on human-annotated dataset; (5) MultiR [Hoffmann et al., 2011] is a typical distant supervised method based on multi-instance learning algorithms to combat the noisy training data; (6) CoType [Ren et al., 2017] is a domain independent framework by jointly embedding entity mentions, relation mentions, text features and type labels into meaningful representations.

LSTM-LSTM-Bias [Zheng et al., 2017] is the baseline end-to-end method in §2, LSTM-LSTM-Bias* is our implementation.

Results Table 4 shows the results. Our transition-based method achieves significant improvements over all the baselines in F1 score. In particular, it achieves 4.6 point improvement over the best jointly extracting method [Ren et al., 2017], and 1.4 point improvement over the best end-to-end sequence labeling method [Zheng et al., 2017], demonstrating the effectiveness of our model on modeling and predicting entities and relations.

The joint methods by multi-task learning are better than pipelined methods, and the end-to-end methods are better than most of the joint methods. This result indicates the importance of joint decoding, which has stronger power of exploiting the dependencies between entities and relations, and also between relation labels in a sentence.

### Table 4: Comparison with previous state-of-the-art methods on NYT.

<table>
<thead>
<tr>
<th>Method</th>
<th>Prec.</th>
<th>Rec.</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>FCM [Gormley et al., 2015]</td>
<td>55.3</td>
<td>15.4</td>
<td>24.0</td>
</tr>
<tr>
<td>DS-logistic [Mintz et al., 2009]</td>
<td>25.8</td>
<td>39.3</td>
<td>31.1</td>
</tr>
<tr>
<td>LINE [Tang et al., 2015]</td>
<td>33.5</td>
<td>32.9</td>
<td>33.2</td>
</tr>
<tr>
<td>MultiR [Hoffmann et al., 2011]</td>
<td>33.8</td>
<td>32.7</td>
<td>33.3</td>
</tr>
<tr>
<td>DS-Joint [Li and Ji, 2014]</td>
<td>57.4</td>
<td>25.6</td>
<td>35.4</td>
</tr>
<tr>
<td>CoType [Ren et al., 2017]</td>
<td>42.3</td>
<td>51.1</td>
<td>46.3</td>
</tr>
<tr>
<td>LSTM-LSTM-Bias</td>
<td>61.5</td>
<td>41.4</td>
<td>49.5</td>
</tr>
<tr>
<td>LSTM-LSTM-Bias*</td>
<td>60.8</td>
<td>41.3</td>
<td>49.1</td>
</tr>
<tr>
<td>Our Method</td>
<td>64.3</td>
<td>42.1</td>
<td>50.9</td>
</tr>
</tbody>
</table>

Table 5: Ablation test on NYT.

<table>
<thead>
<tr>
<th>Method</th>
<th>Prec.</th>
<th>Rec.</th>
<th>F1</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>64.3</td>
<td>42.1</td>
<td>50.9</td>
</tr>
<tr>
<td>-composition</td>
<td>62.3</td>
<td>41.2</td>
<td>49.6</td>
</tr>
<tr>
<td>-Bi-LSTM</td>
<td>62.3</td>
<td>40.5</td>
<td>49.1</td>
</tr>
</tbody>
</table>

2The dataset can be downloaded at: https://github.com/shanzhenren/CoType.
It is worth noting that the precision of our method is much higher compared to all the other methods. We attribute the success to the strong ability to model feature representations of entities and relations, and also the joint decoding.

Ablation Tests To demonstrate the effect of Bi-LSTM representation and relation composition function, we further conduct a set of ablation experiments. For the former, we directly send the token embedding of the input sentence to the above structures of state representation. For the latter, we only update each entity embedding by concatenating its original embedding with relation embedding once a relation arc is generated, ignoring the corresponding head or modifier entity. As shown in Table 5, the F1-score decreases heavily without each strategy, which indicates that it is very important to capture richer contextual information for token embedding and model feature representations of entities and relations.

Case Study We compare our method with LSTM-LSTM-Bias [Zheng et al., 2017] on some cases, as shown in Table 6. As demonstrated by S1, when the distance between two interrelated entities is large, it is more difficult for the LSTM-LSTM-Bias method to identify their relation. However, thanks to the use of stack LSTM state representation, our transition-based method can capture more global feature representations, which make it more powerful identifying long-term relations.

Unlike the LSTM-LSTM-Bias method, our method can identify overlapping relations. S2 in Table 6 shows an example, which cannot be identified by the LSTM-LSTM-Bias method because of its model restriction. Our transition systems have the ability to handle multiple head or tail nodes, which make it suitable for such situation. In addition, our method directly models feature representations of entities and relations by using specially designed composition function, which makes it more powerful when dealing with overlapping relations.

5 Related Work

Two main methods have been proposed for entity and relation extraction, namely the pipeline method and the joint learning method. The former treats this task as two separate tasks, i.e., named entity recognition (NER) [Florian et al., 2010; Kuru et al., 2016] and relation extraction [Bunescu and Mooney, 2005; Liu et al., 2015; Lin et al., 2016].

Existing joint methods include feature-based statistical systems [Ren et al., 2017; Miwa and Sasaki, 2014; Li and Ji, 2014], and neural models [Katiyar and Cardie, 2017; Miwa and Bansal, 2016; Zheng et al., 2017]. Miwa and Bansal [2016] propose a neural method comprised of a sequence-based LSTM for entity identification and a separate tree-based dependency LSTM layer for relation classification. Their model depends critically on access to dependency trees and achieves joint learning only through parameter sharing. Katiyar and Cardie [2017] propose an attention-based joint neural model without accessing to dependency trees. This model extracts the entities and relations from left to right incrementally. However, it does not perform joint decoding and does not model the dependencies between different relations. Zheng et al. [2017] convert the joint task to a sequence labeling problem, achieving joint decoding for entities and relations in one task. Similar with Zheng et al. [2017], we build a neural model with joint decoding. Different from Zheng et al. [2017], however, we transform the joint task into a graph problem and propose a transition-based method.

There has been work using transition-based methods [Zhang and Clark, 2011; Nivre, 2008] to produce dependency trees and directed acyclic graphs (DAGs), but little work on more accurately directed graph in our joint tasks. Choi and McCallum [2013] use the list-based arc-eager algorithm for non-projective trees. Our model extends the method for yielding directed graph structure.

Recently, neural transition-based parsers have achieved highly competitive accuracies thanks to the modeling of output-output relations [Dyer et al., 2015; Kiperwasser and Goldberg, 2016; Lample et al., 2016; Liu and Zhang, 2017; Zhou et al., 2015; Wang et al., 2018]. We are inspired by these methods. To our knowledge, we are the first to investigate neural transition-based parsing framework for entity and relation extraction.

6 Conclusion

We proposed a neural transition-based method for joint entity and relation extraction. Compared with existing neural methods, our method can model underlying dependencies not only between entities and relations, but also between relations. Experiments show that our model achieves the state-of-the-art F-scores on a standard New York Times (NYT) benchmark.
Acknowledgments

This work was supported by the National Key Basic Research Program of China via grant 2014CB340503, the National Natural Science Foundation of China (NSFC) via grant 61632011 and 61772156, and BlueFire AI Ltd via Singapore Grant IGIPTET1702001 at Singapore University and Technology and Design, through which the first author was funded as a visiting student to undertake this research. Yue Zhang is the corresponding author.

References


[Li and Ji, 2014] Qi Li and Heng Ji. Incremental joint extraction of entity mentions and relations. In ACL (1), 2014.


[Ren et al., 2017] Xiang Ren, Zeqiu Wu, Wenqi He, Meng Qu, Clare R Voss, Heng Ji, Tarek F Abdelzaher, and Jiawei Han. Cotype: Joint extraction of typed entities and relations with knowledge bases. In Proc. of the 26th International Conference on World Wide Web, 2017.


