Intent Segmentation of User Queries Via Discourse Parsing

V. Ivan Sanchez Carmona
Lenovo AI Lab
vcarmona@lenovo.com

Yibing Yang
Lenovo AI Lab
yangyb13@lenovo.com

Ziyue Wen
Lenovo AI Lab
wenzy3@lenovo.com

Ruosen Li[*]
Northeastern University
li.ruos@northeastern.edu

Xiaohua Wang
Lenovo AI Lab
wangxh40@lenovo.com

Changjian Hu
Lenovo AI Lab
hucj1@lenovo.com

Abstract

In this paper, we explore a new approach based on discourse analysis for the task of intent segmentation. Our target texts are user queries from a real-world chatbot. Our results show the feasibility of our approach with an F1-score of 82.97 points, and some advantages and disadvantages compared to two machine learning baselines: BERT and LSTM+CRF.

1 Introduction

Chatbots, a type of dialogue system, have been widely deployed for commercial applications, such as flight booking and troubleshooting support. One key step of frame-based chatbots is to extract the *intent* of the user, i.e. to recognize the goal of the user's query (Jurafsky and Martin, 2019). While the task of single-intent prediction, when a query is assumed to encode only one intent, can be tackled as a multi-class classification problem (Wang et al., 2018b), a more complex scenario arises when a query encodes more than two intents–a multi-intent query. In this case, a multi-class classification strategy is applicable if single-intent labels are combined together to form new intent labels; however, this strategy may suffer from data sparsity problems (Xu and Sarikaya, 2013). Another option is to treat multi-intent prediction as a multi-label task where binary classifiers are trained for each single-intent label, and so an arbitrary number of single-intent labels can be predicted for any given query; however, this strategy may not yield the best accuracy results (Xu and Sarikaya, 2013). A third strategy is to segment a user query into intent segments where each segment encodes a single-intent label; these segments then are passed to a pre-trained intent classifier. In this paper we explore this strategy by focusing on the intent segmentation problem.

[*] Work done while doing an internship at Lenovo AI Lab.

To tackle this problem, we use real-world user queries from our online chatbot *Moli* as our data. *Moli* receives queries from Motorola cellphone users related to troubleshooting, warranty and sales issues. We use multi-intent queries which encode from two up to six different intents. Each of these intents may have the same importance as the rest, so predicting a single-intent label for a multi-intent query may lead to a poor user experience. This situation motivates us to propose a segmentation method that is able to find the boundaries among intents in a given query; in this way, single-intent classifiers from previous works can be used to predict the intent label for each segment of a query.

A possible option to segment a query into intent segments are statistical models previously used for sentence segmentation (Beeferman et al., 1999). These models have been widely used in dialogue systems to segment a user's utterance into dialogue acts (Granell et al., 2010; Ang et al., 2005; Martinez-Hinarejos, 2009), and also for intent segmentation (Xu and Sarikaya, 2013). Even though these models have achieved good results in the literature, in this paper we aim to explore a symbolic approach. To the best of our knowledge, this is the first work where discourse analysis is used for intent-segmentation of user queries for chatbots.

Previous works in discourse analysis have proposed different ways of structuring texts into symbolic representations, being RST trees the most popular one. An RST parser is a system that seeks to find hierarchical relationships between adjacent discourse units within a text (Mann and Thompson, 1988; Marcu, 2000). To do so, a text is segmented into discourse units and then a tree structure is built by finding the discourse relationships between adjacent discourse units or sub-tree structures; the final tree structure is a representation of the discourse information contained in the text. RST has been widely used in NLP applications such as topic segmentation (Cardoso et al., 2013), question answering (Verberne et al., 2007), sentence compression (Sporleder and Lapata, 2005) and sentiment analysis (Bhatia et al., 2015).

We propose to use an RST parser to segment user queries into intent segments. We aim to answer the research questions: How can we use an RST parser for the task of intent segmentation of user queries? To what extent discourse information is useful for intent segmentation? What are advantages and disadvantages of segmenting queries via an RST parser? We hypothesize that discourse information is helpful to find tokens within a query that are boundaries of intent segments. In other words, we hypothesize that discourse information is helpful to segment a user query into meaningful pieces of text such that each of these pieces encodes a single intent. Our results seem to support our hypothesis. Also, we compare our approach with two machine learning models, namely BERT and LSTM+CRF. Furthermore, we care on the accuracy-speed trade-off of the RST parser for our intent segmenter to be used online; so, we propose a new RST parser that elaborates on top of a well-known and fast parser in the literature, namely HILDA (Hernault et al., 2010). Overall, our results on the task of intent segmentation–above 80% F1-score–show the feasibility of our approach.

2 Background and Related Work

2.1 Rhetorical Structure Theory (RST)

RST is a formalism proposed by Mann and Thompson (1988) to analyze the discourse structure of texts of any length. To do so, a text is represented as an RST parse tree where adjacent spans of text are related to each other by discourse relations. In any relationship, it is assumed that one of the arguments (the *nucleus*) contains more relevant information than the other argument (the *satellite*).[1] Thus, an RST tree accounts for the discourse information encoded in the text. One reason for the popularity of RST is the feasibility of RST trees to be processed by an NLP system or interpreted by a human.

2.2 RST Parsing

The steps to induce an RST tree are: a) to segment the text into elementary discourse units (EDUs), b) to identify adjacent EDUs to be composed under an RST-relation, c) to identify the nuclearity status of the arguments of a relation, and d) to predict the RST-relation type. While some previous works consider problems b), c), and d) to be the actual *tree-building* problem (Feng and Hirst, 2014) and assume that the correct segmentation is already given, some works focus only on the task of discourse segmentation (Tofiloski et al., 2009; Wang et al., 2018a). In this work, we assume that we have the correct discourse segmentation of a query.

Previous works on discourse segmentation have used both hand-crafted features (Hernault et al., 2010; Joty et al., 2015) and learned-representations (Wang et al., 2018a). We use the system from (Wang et al., 2018a) to split a query into EDUs which predicts either label *boundary* or *not-boundary* for each token in a text. Each token x_i in a text $\mathbf{x}$ is encoded using the concatenation of GloVe and ELMo embeddings; the resulting vector is passed to both a Bi-LSTM layer and to a restricted self-attention module to compute attention vectors which are then passed to a second Bi-LSTM to join both. Finally, the last hidden representations are passed to a CRF layer to compute the probability of a sequence of labels. We chose this system due to its high accuracy on the RST-DT dataset ($F1 = 94.3$).

A popular type of RST parser in the literature is a CRF-based parser. It computes two conditional probabilities, namely that of composing two EDUs into a sub-tree and that of the type of discourse relation (Joty et al., 2012, 2013; Feng and Hirst, 2014). To build the highest-scoring RST tree based on the probabilities computed, previous works have used either dynamic programming or greedy algorithms. An advantage of this approach is that it can capture probabilistic dependencies among adjacent sub-structures. However, using CRFs involves expensive computations (Feng and Hirst, 2014). Due to its popularity, we use a CRF-based baseline.

Another approach for RST parsing is the HILDA parser (Hernault et al., 2010). It uses two SVM classifiers to build the RST tree. The first classifier predicts the likelihood of two EDUs being under a discourse relation for all pairs of adjacent EDUs, and the highest-scoring pair is composed into a sub-tree. Then, the second classifier predicts both the RST relation type and nuclearity for the sub-tree. This cycle continues until all EDUs and sub-trees are composed into a single RST tree. We

[1]Except for some relations where both arguments act as nucleus.

draw inspiration from this approach due to both its linear-time complexity and its simplicity. However, one drawback is that it uses a greedy search to build the RST tree which is not optimal. Besides, both classifiers require hand-crafted features. We designed our parser to work similar to HILDA but we enhance it by improving the search. Additionally, we use only one classifier and we use learned-representations. In this way, we obtained a fast, simple and optimal parser.

3 A Working-Example

In this section we provide an example of the problem of segmenting a user query into intent segments. Consider the user query in Example 1 which describes problems a user has with a cellphone. The gold intent-segmentation of this query is shown in Table 1. One possible way to segment this query into intent-segments is by using punctuation marks as splitting points (Kiss and Strunk, 2006). However, this would result in an over-segmentation (five intent segments instead of three segments as in Table 1) because there are four dots. Thus, it is important that the segmentation system learns when punctuation marks (or keywords) are boundaries and when they are not. Furthermore, since user queries are a type of informal discourse, we can expect some of them to miss punctuation marks, which can make a harder case for the segmenter; in this case, the segmenter should rely on other type of information than just the tokens in the query. We believe discourse information can help a segmentation system to overcome these issues. In order to use RST parsing for intent segmentation, we annotated a set of user queries with their corresponding RST parses (Section 4.3). The RST parse of the query in Example 1 is shown at the bottom of Table 2 and the tree representation is shown in Figure 1. We will elaborate on this example to explain our approach in subsequent sections.

Example 1. I am having a trouble with my power button. It's not functioning at all. it seems that my phone has got switched off. I am trying to reboot it. but no success so far

Gold Intent Segments
s_1: I am having a trouble with my power button. It's not functioning at all.
s_2: it seems that my phone has got switched off.
s_3: I am trying to reboot it. but no success so far

Table 1: Gold intent segmentation of the query in Example 1. The intent labels for segments s_1, s_2, and s_3 are *power_button_broken*, *random_shut_down*, and *cannot_power_on*, respectively.

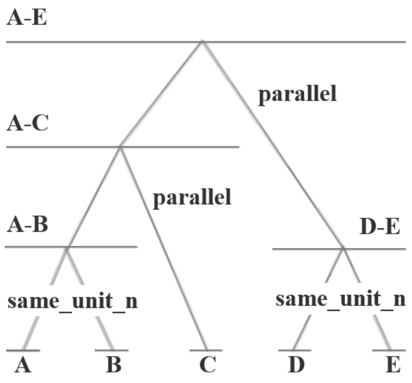


Figure 1: RST tree of the query in Example 1. At the bottom of the tree we see the EDUs A, B, C, D, E. As we move upwards in the tree, we see the RST-relations under which text spans are composed. For example, the span A-B formed by the EDUs A and B is composed with C under relation *parallel*.

4 Our Approach: Discourse Analysis for Intent Segmentation

In this section, we present our RST parser, our datasets and our approach to segment chatbot's user queries into intent segments.

4.1 Problem Definition

Given a user query Q_i that encodes the set of intents $I = i_1, ..., i_k$ where $k \geq 2$, $i_1 \neq i_2 ... \neq i_k$, and it contains EDUs $e_1, ..., e_m$ where $m \geq 2$, we define the problem of intent segmentation as partitioning query Q_i into segments $s_1, ..., s_k$ such that each segment s_j has a one-to-one correspondence to each intent i_j from set I. We note that a segment s_j may be formed by more than one EDU.

4.2 Our Discourse Parser

We follow a similar pipeline as that of the HILDA parser for our parser. We first split a query by EDUs; then we iteratively build a complete RST tree by searching for the best-scoring sub-tree at each iteration, i.e. we search for the RST-relation $r_j \in R$ and the two arguments (EDUs or sub-trees) in the form of text spans x_i and x_j that when composed as $r_j(x_i, x_j)$ give the highest score. However, there are some differences between our parser and HILDA (Section 2.2). First, we search for the

Discourse Information
A. I am having a trouble with my power button.
B. It's not functioning at all.
C. it seems that my phone has got switched off.
D. I am trying to reboot it.
E. but no success so far
(*parallel*,(*parallel*,(*same_unit_n*,A,B),C),(*same_unit_n*,D,E))

Table 2: Discourse information from the query in Example 1. The first five rows show the discourse segmentation where A, B, C, D, E are the EDUs. The last row shows the RST parse of the query which was built by composing the EDUs through RST relations.

highest-scoring tree using a policy that combines a greedy search with an optimal search (Dijkstra search). Second, we only use one classifier to predict the RST relation r_j, the nuclearity structure, and the probability of a pair of arguments to be composed into a sub-tree under relation r_j. Third, we use dense vectors since our classifier is a neural network.[2] In what follows, we describe the components of our RST parser.

To obtain the EDUs of a query we used the state-of-the-art system from (Wang et al., 2018a), which was trained on the RST-DT dataset. However, since the domain of the RST-DT (newspaper's articles) is different from our domain (user queries) we added hand-crafted rules on top of this system to account for possible mistakes in the discourse segmentation due to the shift in domain.[3] Thus, the EDUs generated from a query by this system pass through our rules and are adjusted accordingly; after that, the resulting adjusted EDUs are used to search for the highest-scoring RST tree by our parser.

To do a search on the space of RST trees, we use an RST-relation classifier, a search algorithm and pre-computed probabilities of RST trees. We obtain from our classifier both the most suitable RST-relation type for two arguments and the confidence of this prediction. Thus, inputs to this classifier are two text spans and output is a confidence score for each RST-relation label in our set of relations R. We train this classifier using our data (Section 4.3). To find an RST tree for long queries, Dijkstra algorithm may take an impractical amount of time, thus we apply a policy to shift between Dijkstra and a greedy search according to the number of EDUs the query is segmented into. For both algorithms, Dijkstra and greedy, at each step in the search, we call our RST-relation classifier and give it as input a pair of adjacent spans of text, and in turn we receive an RST label and the classifier's confidence. We then compose via a logarithm function the classifier's confidence with the probability of observing the current candidate sub-tree in our training data.[4] We use this composite function as the score for the current candidate sub-tree. An advantage of Dijkstra over the greedy algorithm is that Dijkstra not only scores sub-trees independently but also gets a score for the whole tree built up to the current step.[5] Thus, Dijkstra algorithm is able to find an optimal RST tree.

4.3 Our Datasets

Due to data-confidentiality policies, we cannot release the user queries, but we provide an overview of our data. We created three datasets, namely one for the task of intent segmentation, and two for the task of RST parsing. We note that the first step for annotating the datasets is the same, namely to segment each user query into EDUs. Then, we annotate each dataset accordingly for each task.

Intent-segmentation dataset: It consists of 1356 user queries. Based on the discourse segmentation of each query, annotators manually concatenated EDUs in order to obtain intent segments. For example, based on the EDUs of the query in Example 1 (see Table 2), annotators decided to concatenate EDUs A and B to obtain the intent segment s_1 (Table 1), and similarly for segments s_2 and s_3. We used 1037 queries for train, 206 queries for test, and 113 queries for development data.

RST datasets: We used the same train, test, and development sets as those used for creating our intent-segmentation dataset. In our first dataset each user query is manually annotated with its corresponding RST tree.[6] Based on this dataset, we

[2]We tried hand-crafted features and SVMs but our results were not as good as when using neural networks.

[3]We could not re-train the system of Wang et al. (2018a) due to the small size of our dataset.

[4]We pre-computed the probability for each type of subtree. We used a Laplacian smoothing for sub-trees not seen in the data. For example, if the current candidate sub-tree is the composition of EDU e_1 with the sub-tree (r_i, e_2, e_3) under the RST label r_j: (r_j, e_1, (r_i, e_2, e_3)), then we look for the probability of a sub-tree with root node r_j whose left argument is an EDU and whose right argument is a sub-tree with root node r_i.

[5]To get the score for a whole tree we simply sum the scores of the sub-trees contained in it.

[6]We use this dataset to test our RST parser; since our parser uses search algorithms to generate a parse tree we do not need to use train data.

generated a dataset for our RST-relation classifier. It consists of 5517 train, 951 test, and 543 development instances; this figure is because the RST tree of one query is decomposed into several sub-trees, each of which has an RST-relation at the parent node; in this way we obtained several instances from one RST tree.[7] This is a labeled dataset since the input space is defined by text spans and the output space is defined by RST relations.

To annotate queries with their RST trees, we follow a similar annotation scheme and annotator training to that from Carlson et al. (2001) where each annotation is human-validated. We followed the definitions of RST relations in (Carlson and Marcu, 2001) as it is a well-known manual for RST annotation. However, the domain of the data in this manual (newspaper articles) differs from our data domain; hence, we faced similar issues as those in (Stent, 2000); for example, the definition of an RST relation may not directly apply to a user query, or the definitions of two RST relations seem to overlap with each other. Since we optimized for simplicity in our annotation, we decided to select the fewest number of RST relations as possible.

RST-relation	Nuclearity
background	x_1:satellite, x_2:nucleus
elaboration	x_1:nucleus, x_2:satellite
parallel	x_1:nucleus, x_2:nucleus
same_unit_n	x_1:nucleus, x_2:nucleus
same_unit_s	x_1:satellite, x_2:satellite

Table 3: Our choice of RST relations. First column indicates the name of the relations. Second column indicates the nuclearity status of the first (x_1) and second (x_2) arguments of a relation.

We chose $|R| = 5$ RST relations to label our data. We took four of them from (Carlson and Marcu, 2001), namely *background*, *elaboration*, *same_unit_s*, *same_unit_n* relations, which we slightly re-defined, and we proposed a new RST relation, which we call *parallel*. We decided to apply two constraints to our RST relations, namely a nuclearity constraint and an arity constraint. We fixed the position of the nucleus and the satellite for every relation, so the nuclearity is implicitly predicted by our RST-relation classifier. And we chose all our RST relations to be binary, following the work in (Marcu, 2000), due to simplicity in parsing.

[7]See the query in Example 1 and its arrangement into 4 RST relations shown in the last row of Table 2; thus, from this query we obtained 4 instances.

Table 3 shows the RST relations and their nuclearity constraints. While both relations *background* and *elaboration* aim to provide additional information for the nucleus through the satellite, relations *same_unit_s* and *same_unit_n* have the purpose of joining the two arguments into a single piece of text when the information in both arguments is closely related. Finally, relation *parallel* serves to indicate that two arguments are independent of each other, i.e. they are equally important and the information they contain is unrelated between them.

We computed F1-scores for inter-annotator agreement, as in (Marcu, 2000), for both sub-tree structures: $Prec = 82.87\%$, $Rec = 82.48\%$, $F1 = 82.67\%$, and for text spans: $Prec = 87.50\%$, $Rec = 87.09\%$, $F1 = 87.29\%$. These high agreements suggest that our RST dataset is consistent.

4.4 Intent Segmentation Via RST Parsing

In this section, we explain our strategy to segment a user query Q_i into intent-segments $s_1, ..., s_n$. The main idea is to use the RST-parse tree of the query to decide which adjacent text spans form an intent segment s_j. We hypothesize that there is a correlation between the information that RST relations convey and the change in the type of intent label across segments in a query. More concretely, for each query and predicted RST tree, we traverse the tree in a bottom-up fashion going from leaf nodes up to the root node. Whenever we see the application of the RST-relations *same_unit_s* or *same_unit_n* on two arguments, we join the two spans of text into a single piece of text based on our hypothesis that these RST relations indicate a continuity in intent-segment. On the other hand, when we see the application of relations *parallel*, *background*, or *elaboration* we place a boundary symbol indicating that the text spans corresponding to the two arguments of these RST relations pertain to different intent segments.[8]

For example, consider the query in Example 1 as in Figure 1. We start our procedure with the sub-tree (*same_unit_n*,D,E) where EDUs D and E are concatenated into a single text span, namely D-E. Next, EDUs A and B from sub-tree (*same_unit_n*,A,B) are concatenated into A-B. The next parse, due to the relation *parallel*, indicates

[8]We label the *satellite* arguments of *background* and *elaboration* relations with a *null* intent label since we consider such information to be irrelevant to the actual problem posed by the user captured in the *nucleus* argument.

that the text span A-B is independent from unit C, and similarly with the text span D-E. In this way, we obtain the intent segmentation: s_1: A-B, s_2: C, and s_3: D-E, and recovered the gold intent-segmentation of this query (Table 1). Thus, we segmented a query into intent segments using its corresponding RST tree. As we will see in Section 6, discourse information helps us to better find the boundaries of intent segments when compared to a discourse segmenter and an LSTM+CRF.

5 Experiments

5.1 RST-Relation Classifier

In our RST parser, the only component that requires training is the RST-relation classifier. We used Keras (Chollet et al., 2015) with TensorFlow (Abadi et al., 2016) and our RST-relations dataset to train this classifier. We setup this problem as a multi-class classification where each RST-relation type is a label to be predicted. We used ELMO word embeddings[9] (Peters et al., 2018) as the representation for each token in a query[10] and we allowed these representations to be updated at training time.[11] We picked the best hyperparameters based on performance on the development set.[12]

5.2 RST Parsing

Though our RST parser does not need to be trained since it uses search algorithms to find a parse tree, it requires the RST-relation classifier to be already trained to score candidate sub-trees. We note that our parser contains a hyperparameter which we manually tuned on the development set, namely the scoring function used by both greedy and Dijkstra search. We found the best-scoring function among three functions: $log(x * y)$, $log(x + y)$, $log(x)$ where the term x is the RST-relation classifier's confidence and the term y is the probability of observing a candidate sub-tree in training data. We test our parser on our test set.

5.3 Intent Segmentation

We evaluate our intent-segmentation approach by computing precision, recall, and F1 scores on the number of tokens correctly found to be boundaries of intent-segments (true positives) on test data. For example, in the gold intent-segmentation of the query in Example 1 (Table 1) we can see that there are three intent segments, hence the number of gold boundaries are three, namely the tokens at the end of each intent segment.[13]

5.4 Baselines

We compare our intent-segmentation approach against three baselines. The first is the discourse segmenter from Wang et al. (2018a) where the outputs of the discourse segmenter are taken as intent segments. We use this baseline to prove our hypothesis. The second baseline is based on BERT (Devlin et al., 2019). We apply the standard BERT implementation for a sequence labeling problem where boundary tokens in a query are labeled as 1 while non-boundary tokens are labeled as 0. The input (a text sequence) is tokenized by BERT tokenizer and is embbeded by BERT embeddings. After that, it passes through a 12-layers Transformer model; then, encoding sequences corresponding to each token are obtained. A linear layer with softmax over the top-layer outputs a sequence of distribution over classification results with given encodings (Fig. 2). Finally, the classification results are converted into separate marks and the whole sequence of tokens is transformed back to a text sequence. We train over each batch, backpropagating into BERT's parameters. We maximize the log-likelihood of boundary classification specifically.[14] The third baseline is a BiLSTM+CRF model which has a full-connection layer between the BiLSTM and CRF layers.[15] We trained both models BERT and BiLSMT+CRF on our intent-

[9]Dimensionality $d = 256$.

[10]We also tried GloVe embeddings but our results were slightly better with ELMO embeddings.

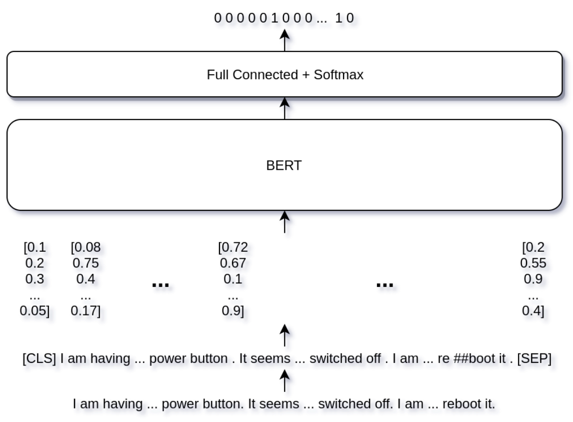
[11]We used a batch size of $b = 32$; we trained a BiLSTM of size $h = 100$; we placed two dropout layers (both with rate $dr = 0.5$), one after the embedding layer and another one after the BiLSTM. On top of the BiLSTM we added a densely-connected layer with a softmax activation. We used a categorical cross-entropy loss function and Adagrad as optimizer.

[12]We tried two more experiments that we do not report because we did not improve accuracy score. In the first one, we experimented with the minimum number of possible RST-relation types for our approach to work, namely when there are only two types of relations (*parallel* and *same_unit*). In the second experiment, we used $|R| = 14$ RST relation types. For all our experiments we used the same setup for training, but the best results were achieved with $|R| = 5$ RST relation types.

[13]The dot after the word "all" in s_1, the dot after the word "off" in s_2, and the word "far" in s_3

[14]The learning rate is $r = 2e - 5$; batch size is $b = 16$; we set the max length of an input sequence to be $L = 256$ which is wrapped by the symbol [CLS] at the beginning and the symbol [SEP] at the end; embedding dimension is $d = 768$.

[15]The size of the BiLSTM is $h = 300$, the full-connection embeddings size is $d = 600$, we use Adam optimzer, a learning rate of $lr = 0.001$ and dropout rate of $dr = 0.5$.



000001000... 10
Full Connected + Softmax
BERT
[0.1 0.2 0.3 ... 0.05]
[0.08 0.75 0.4 ... 0.17]
...
[0.72 0.67 0.1 ... 0.9]
...
[0.2 0.55 0.9 ... 0.4]
[CLS] I am having ... power button . It seems ... switched off . I am ... re ##boot it . [SEP]
I am having ... power button. It seems ... switched off. I am ... reboot it.

Figure 2: BERT Baseline.

segmentation dataset. We use these two baselines as a point of comparison for the generalization ability of our approach.

6 Results

In this section, we report the results of our experiments. The accuracy score of our RST-relation classifier is $Acc = 74.22\%$ across the 5 RST-relation types on test data.

In Table 5, we see the results of our approach against the baselines for the task of intent segmentation. The results from the first and the second rows seem to confirm our hypothesis. Our strategy is more accurate (by 5 points) than the approach based only on discourse segmentation. This figure seems to show that our RST parser captures the continuity and shift in intent between adjacent text spans within a query. We see that the discourse segmenter achieves a higher recall score than our approach, but this is simply due to the fact that this baseline over-segments a query and thus is more likely to correctly hit most of the tokens which are boundaries. Nevertheless, we see that our approach achieves a high recall score by surpassing the 80% threshold by 4 points. Overall, the scores that our approach obtains show the advantage of using an RST-parser on top of a discourse segmenter: The trade-off between precision and recall is in a sweet-spot which means that probably neither of both will be "sacrificed" for the other as more data is collected and used for re-training the RST-relation classifier. This figure does not seem to occur in the case of the baseline; the F1-score is justified only by the high recall rate due to the query's over-segmentation.

From the first and third rows of Table 5 we now compare our approach with the BERT baseline. Even though our approach achieves 82% F1-points, this baseline obtains a higher score by slightly more than 2 points. A similar figure arises when we compare precision and recall scores from both approaches. These figures seem to show that this baseline has also found a sweet-spot in the precision-recall trade-off, similar to our approach. Surprisingly, the LSTM+CRF model achieves the lowest F1-score from all the approaches, which may be explained by this baseline overfitting on the training data. Furthermore, we also note that the difference in F1-score between our approach and the discourse segmenter is more than twice the difference in score between BERT and our RST parser. This figure may suggest that our approach is considerably closer in terms of performance to an state-of-the-art system than to a system based on an over-segmentation strategy.

We now turn to Table 4 where we see the effect of different scoring functions for the search algorithms on the results of our RST parser on test data.[16] We can see that depending on the form of the scoring function we obtain a particular trade-off among different indicators of performance. For example, if we aim to optimize running time, we may choose the scoring function $log(x + y)$ which allows our RST parser to achieve the lowest time at the cost of minimally sacrificing F1-score on the intent-segmentation task with respect to the highest-scoring function ($log(x)$). On the other hand, if we aim for the best-scoring parser in terms of recovering RST trees, we may choose the scoring function $log(x * y)$ which allows our RST parser to recover the highest rate of sub-trees ($F1 = 46.16\%$) and text spans ($F1 = 63.89\%$) from test data while minimizing the number of illegal parses.[17]

7 Discussion and Conclusions

We worked on the task of intent segmentation of user queries posed to a real-world chatbot. We explored an approach based on discourse analysis

[16] We setup the policy that if the number of EDUs in a query is $|E_{Q_i}| \leq 7$ then we resort to Dijkstra search, else we resort to greedy search. We optimized this policy on the development set.

[17] An illegal parse is when the RST tree contains at least one illegal sub-tree; for example, the sub-tree (*same_unit_n*, (*parallel*, A, B), C) is illegal because the most internal sub-tree indicates that EDUs A and B are parallel, therefore the attempt to concatenate them with EDU C results in an illegal action because this implies that A and B must be concatenated beforehand.

Function	Intent Segmentation			Average Time		Illegal Parses	No Split	F1-scores	
	Prec	Rec	F1	Dijkstra	Greedy			Sub-tree	Span
$log(x*y)$	81.7	79.6	80.6	0.37	0.21	1	15	46.16	63.89
$log(x+y)$	83.6	81.7	82.7	0.14	0.19	4	11	43.30	61.75
$log(x)$	81.5	84.4	82.9	0.18	0.19	7	11	42.86	60.33

Table 4: Effect of scoring functions in our RST parser. In column Function, x is the RST-relation classifier's confidence, and y is the probability of the candidate sub-tree. Column Intent Segmentation shows the Precision, Recall, and F1 scores on the task of intent segmentation. Column Average Time shows the average running-time in seconds to find the RST tree of a query. Column Illegal Parses shows the number of predicted RST trees that contain an illegal sub-tree. Column No Split shows the number of incorrectly built RST trees due to only containing RST relation *same_unit_s* or *same_unit_n* and thus not having any split point. Column F1-scores shows the performance of our RST parser for correctly recovering sub-trees and text spans from the test data.

Model	Precision	Recall	F1
Our approach	81.53	84.47	82.97
DS	63.97	98.39	77.53
BERT	85.11	86.01	85.55
LSTM+CRF	68.28	72.70	70.42

Table 5: Scores for the task of intent segmentation for four models: our approach, a discourse segmenter (DS), a BERT model, and an LSTM+CRF model.

where we built an RST tree for any given query and based on this symbolic representation we segmented the query into intent segments. We built our RST parser inspired on the HILDA parser: Our RST parser builds an RST tree using a search on the space of parse trees. However, we improved our parser by combining Dijkstra and greedy search. Furthermore, our parser uses only one RST-relation classifier to inform the search algorithm.

We hypothesized that discourse information could inform a text segmenter the tokens where there is a change or a continuity in intent label. Our results seem to support our hypothesis. Compared to a pure discourse segmentation approach, our RST parser seems to find correlations between RST-relation types and intents. Given two adjacent text spans, our parser can accurately decide whether these spans should be concatenated into one intent segment or should remain independent of each other. We note that if the discourse information was not helpful, the score of our approach, in the best scenario, would be the same as that of the baseline (it would have no effect), and in the worst scenario the score would drop lower than the baseline's score by incorrectly concatenating adjacent EDUs.

We compared our system's generalization ability against two ML approaches: BERT and LSMT+CRF. We found that BERT achieves better scores than our approach by 2.58 F1-points; we think this gap is mainly due to the pre-training of BERT on a large amount of data. However, our approach surpasses the LSMT+CRF model by a large margin; we believe the LSTM+CRF may overfitted the training data due to its small size. Our approach does not seem to overfit the data because it uses search algorithms, which poses another advantage of our approach over ML models: These algorithms have parameters that function as a type of *knob* which allows us to tweak the parser to a desired configuration without the need of training. For example, we can tweak our search algorithm to favour for running-time just by changing the scoring function. This clear but sometimes forgotten advantage of symbolic models comes in handy when we want to have a manual control over the behavior of our model. Another advantage of our approach over the BERT baseline is the simplicity in training; we did not require to use any GPU to train our RST-relation classifier. Nevertheless, a clear disadvantage of our approach is the need to create an RST dataset which requires an annotation scheme and annotator training.

In conclusion, we believe our approach is suitable as an end-to-end application for intent segmentation of user queries posed to a chatbot. We believe future work should focus on better models for the RST-relation classifier. Also, due to the symbolic nature of the RST trees obtained from user queries, we propose to explore to what extent we can analyse and interpret them in order to better understand what knowledge the parser has captured, what types of mistakes it has made, and how to further improve it based on error analysis.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA. USENIX Association.

Jeremy Ang, Yang Liu, and Elizabeth Shriberg. 2005. Automatic dialog act segmentation and classification in multiparty meetings. In *Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.*, volume 1, pages I/1061–I/1064 Vol. 1.

Doug Beeferman, Adam Berger, and John Lafferty. 1999. Statistical models for text segmentation. *Machine Learning*, 34(1):177–210.

Parminder Bhatia, Yangfeng Ji, and Jacob Eisenstein. 2015. Better document-level sentiment analysis from RST discourse parsing. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 2212–2218, Lisbon, Portugal. Association for Computational Linguistics.

Paula Cardoso, Maite Taboada, and Thiago Pardo. 2013. On the contribution of discourse structure to topic segmentation. In *Proceedings of the SIGDIAL 2013 Conference*, pages 92–96, Metz, France. Association for Computational Linguistics.

Lynn Carlson and Daniel Marcu. 2001. *Discourse Tagging Reference Manual*. ISI. Technical Report. ISI-TR-545.

Lynn Carlson, Daniel Marcu, and Mary Ellen Okurovsky. 2001. Building a discourse-tagged corpus in the framework of rhetorical structure theory. In *Proceedings of the Second SIGdial Workshop on Discourse and Dialogue*.

François Chollet et al. 2015. Keras. `https://keras.io`.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Vanessa Wei Feng and Graeme Hirst. 2014. A linear-time bottom-up discourse parser with constraints and post-editing. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 511–521, Baltimore, Maryland. Association for Computational Linguistics.

Ramón Granell, Stephen Pulman, Carlos D. Martínez-Hinarejos, and José-Miguel Benedí. 2010. Dialogue act tagging and segmentation with a single perceptron. In *Interspeech: 11th Annual Conference of the International Speech Communication Association*, pages 3074–3077.

Hugo Hernault, Helmut Prendinger, David duVerle, and Mitsuru Ishizuka. 2010. Hilda: A discourse parser using support vector machine classification. *Dialogue & Discourse; Vol 1, No 3 (2010)*, 1.

Shafiq Joty, Giuseppe Carenini, and Raymond Ng. 2012. A novel discriminative framework for sentence-level discourse analysis. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 904–915, Jeju Island, Korea. Association for Computational Linguistics.

Shafiq Joty, Giuseppe Carenini, Raymond Ng, and Yashar Mehdad. 2013. Combining intra- and multi-sentential rhetorical parsing for document-level discourse analysis. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 486–496, Sofia, Bulgaria. Association for Computational Linguistics.

Shafiq Joty, Giuseppe Carenini, and Raymond T. Ng. 2015. Codra: A novel discriminative framework for rhetorical analysis. *Computational Linguistics*, 41(3):385–435.

Dan Jurafsky and James H. Martin. 2019. *Speech and Language Processing*, 3rd (draft) edition.

Tibor Kiss and Jan Strunk. 2006. Unsupervised multilingual sentence boundary detection. *Computational Linguistics*, 32(4):485–525.

William Mann and Sandra Thompson. 1988. Rethorical structure theory: Toward a functional theory of text organization. *Text*, 8(3):243–281.

Daniel Marcu. 2000. The rhetorical parsing of unrestricted texts: A surface-based approach. *Computational Linguistics*, 26(3):395–448.

Carlos D. Martinez-Hinarejos. 2009. A study of a segmentation technique for dialogue act assignation. In *Proceedings of the Eighth International Conference in Computational Semantics IWCS8*, pages 299–304. Tilburg University, Department of Communication and Information Sciences.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke

Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Caroline Sporleder and Mirella Lapata. 2005. Discourse chunking and its application to sentence compression. In *Proceedings of Human Language Technology Conference and Conference on Empirical Methods in Natural Language Processing*, pages 257–264, Vancouver, British Columbia, Canada. Association for Computational Linguistics.

Amanda Stent. 2000. Rhetorical structure in dialog. In *INLG'2000 Proceedings of the First International Conference on Natural Language Generation*, pages 247–252, Mitzpe Ramon, Israel. Association for Computational Linguistics.

Milan Tofiloski, Julian Brooke, and Maite Taboada. 2009. A syntactic and lexical-based discourse segmenter. In *Proceedings of the ACL-IJCNLP 2009 Conference Short Papers*, pages 77–80, Suntec, Singapore. Association for Computational Linguistics.

Suzan Verberne, Lou Boves, Nelleke Oostdijk, and Peter-Arno Coppen. 2007. Discourse-based answering of why-questions. *Traitement Automatique des Langues*, 47.

Yizhong Wang, Sujian Li, and Jingfeng Yang. 2018a. Toward fast and accurate neural discourse segmentation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 962–967, Brussels, Belgium. Association for Computational Linguistics.

Yu Wang, Yilin Shen, and Hongxia Jin. 2018b. A bimodel based RNN semantic frame parsing model for intent detection and slot filling. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 309–314, New Orleans, Louisiana. Association for Computational Linguistics.

Puyang Xu and Ruhi Sarikaya. 2013. Exploiting shared information for multi-intent natural language sentence classification. In *Interspeech*.