Transition-based Sentence Compression with Lexical and Length Constraints

Anonymous NAACL submission

Abstract

Traditional approaches to extractive sentence compression seek to reduce the length of a sentence, while retaining the most "important" information from the source. But query-focused applications such as document search engines or exploratory search interfaces place additional lexical and length requirements on compression systems. This study introduces a new transition-based, neural compression method which accommodates such requirements by pruning syntactic dependency subtrees. We show that our technique is more computationally efficient than previous ILP-based approaches, and achieves competitive performance in reconstructing known-good shortenings under constraints.

1 Introduction

Traditional study of extractive sentence compression seeks to create short, readable compressions which retain the most "important" information from source sentences. But query-focused and user-facing applications impose additional requirements on the output of a compression system. Compressions must be short enough to be shown in a user interface and must contain a user's query term, which defines the "important" information in the sentence. An example of such a compression is shown in Figure 1.

This study examines the English-language compression problem with such length and lexical requirements: compressed sentences are (1) required to include a list of query words q and (2) required to be shorter than or equal in length to a maximum character length, $b \in \mathbb{Z}^+$.

While older compression methods based on integer linear programming could trivially accommodate such restrictions (Clarke and Lapata, 2008; Filippova and Altun, 2013), recent work adopts a sequence to sequence paradigm which

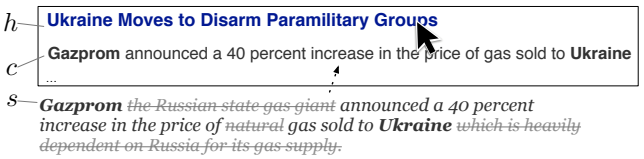

Figure 1: A search interface (boxed, top) returns a headline h above a constrained compression c, shortened from a sentence s in a query-relevant document (italics, bottom). The constrained compression must contain the users' query terms (bold), and must not exceed 75 characters in length.

can often reconstruct gold-standard shortenings, but which does not give practitioners such control (Filippova et al., 2015). This makes existing sequence to sequence techniques unsuitable for search engines (Hearst, 2009), concept map browsers (Falke and Gurevych, 2017) and new forms of exploratory textual interfaces (Marchionini, 2006), where length and lexical constraints are paramount.

Therefore, in this work, we present a new method for compressing sentences, which efficiently accommodates lexical and length constraints. Our transition-based, stateful compression method is inspired by similar approaches to transition-based dependency parsing (Nivre, 2003; Chen and Manning, 2014). We compare this technique to supervised, integer linear programming (ILP) systems, which also accommodate length and lexical restrictions. We show that our method has lower computational costs while better reconstructing known-good, constrained shortenings.

2 Methods for sentence compression

This work contrasts traditional ILP-based compression with our novel transition-based framework. Both ILP-based methods and transition-based methods can accommodate constrained compression. We also briefly discuss sequence to

Approach	Worst-case complexity	Constrained
Sequence to sequence taggers (Filippova et al., 2015)	linear	no
Transition-based deletion (this work)	**quadratic**	**yes**
ILP (Filippova and Altun, 2013; Wang et al., 2017)	exponential	yes

Table 1: Three algorithms for sentence compression. Integer linear programming methods (Clarke and Lapata, 2008; Filippova and Altun, 2013; Wang et al., 2017) can easily accommodate length and lexical constraints, and can make use of large parallel corpora of sentence–compression pairs. But these methods formulate the compression task as an NP-hard problem (§2.1), with worst-case exponential runtime. LSTM taggers (Filippova et al., 2015) achieve comparable results with a linear runtime, but cannot accommodate length or lexical requirements. This work introduces a supervised, transition-based approach (§3) which can be used to compress sentences under lexical and length constraints in quadratic time.

sequence compression methods, which do not accommodate length or lexical requirements. Table 1 provides a summary of each approach.

2.1 ILP-based compression

One common approach for shortening sentences formulates compression as an ILP task. ILP-based methods assign binary variables to each token in a sentence (Clarke and Lapata, 2008) or subtree in a dependency parse (Filippova and Strube, 2008). These variables indicate if the corresponding sentence component is included in a compression. Each such component is also assigned a local weight, indicating its worthiness for inclusion in a shortened sentence. Local weights are either learned from direct supervision (Filippova and Altun, 2013; Wang et al., 2017), or inferred from sources like corpus statistics, importance score metrics and n-gram language models (Clarke and Lapata, 2008; Filippova and Strube, 2008).

Such ILP methods represent the overall quality of a compression by summing the local weights of sentence components to compute a global objective score. The task of identifying the best possible solution to an integer programming objective is a well-known, NP-hard problem (Clarke and Lapata, 2008). In the worst case, such problems require exponential computation in the length of the input sequence. Researchers use off-the-shelf ILP solvers to identify the highest-scoring compression, from among all possible configurations of binary variables (each may be set to 0 or 1).

This integer linear programming approach also easily accommodates constrained compression. Researchers will customarily add constraint terms to the ILP objective to enforce hand-build semantic restrictions (Clarke and Lapata, 2008) or syntactic requirements (Filippova and Strube, 2008). Adding additional length or lexical requirements to ILPs is straightforward: practitioners must specify that optimal solutions must be shorter than some character budget, and must specify that binary variables marking inclusion of particular words must be set to 1.

In practice, we have found that translating a sentence compression objective into an integer linear program can be a challenging and error-prone programming task. These engineering difficulties are a disadvantage of ILPs.

2.2 Sequence to sequence compression

We contrast transition-based compression and integer programming approaches with sequence to sequence methods for the compression task (Filippova et al., 2015). These techniques achieve high automatic evaluation scores by labeling input tokens with a 1 or a 0, indicating if the word should be included in a shortening. Such methods are also linear in the token length of the input sequence, where ILPs incur exponential cost. However, at this time, sequence to sequence taggers are unsuitable for query-focused applications because such methods cannot enforce lexical or length requirements. This limitation might be re-examined in future work, by modifying or adapting new constrained generation techniques (Post and Vilar, 2018; Miao et al., 2019).

2.3 Transition-based compression

In this work, we present a new, neural, transition-based method for shortening sentences under lexical and length constraints. Our method compresses a sentence over N time steps, by adding and removing N different subtrees from a dependency parse, one after another. Our stateful approach recalls early solutions to the sentence compression problem (Jing, 2000; Knight and Marcu, 2000), which also shortened sentences by execut-

Operation	Definition	Description
START	START $\Rightarrow (V = \emptyset, B = [v_1, v_2 ... v_n])$	Initialize the buffer with the vertexes in the original sentence s, arranged breadth-first
PRUNE	$(V, [v\|B]), v \in V, \Rightarrow (V \setminus T(v), B)$	Remove the subtree rooted at v in s from V
INSERT	$(V, [v\|B]), v \notin V, \Rightarrow (V \cup T(v), B)$	Insert the subtree rooted at v in s into V
NOPRUNE	$(V, [v\|B]), v \in V, \Rightarrow (V, B)$	Don't remove the subtree rooted at v from V
NOINSERT	$(V, [v\|B]), v \notin V, \Rightarrow (V, B)$	Don't insert the subtree rooted at v into V
STOP	$(V, B = []) \Rightarrow$ STOP	Compression ends when the buffer is empty

Table 2: A transition-based sentence compression system with a PRUNE and INSERT operation. The state of the compression system is a tuple (V, B), where V is a subset of vertexes from the original sentence s, B is an ordered buffer of tokens and $[v|B]$ indicates that $v \in V$ is at the head of the buffer. $T(v)$ denotes the subtree rooted at v in the original sentence (i.e. v and all its descendants). Section 3.1 describes each operation informally. The appendix presents a complete, worked example. These operations can fully reconstruct all shortenings in a standard compression corpus (Filippova and Altun, 2013).

ing grammatically-motivated operations on syntax trees. We present the formal details in Section 3.

Like ILP-based methods, transition-based approaches can easily accommodate lexical and length restrictions. At a high-level, such methods need to add subtrees which contain query terms and remove subtrees which do not contain query terms, until identifying a compression which satisfies the length constraint. We show that our transition-based method has a lower computational cost (§5), while achieving better performance in constrained compression (§4) than the leading, supervised ILP technique (Filippova and Altun, 2013).

3 Transition-based sentence compression

In this work, we present a new transition-based sentence compression system, inspired by similar approaches in transition-based parsing (Nivre, 2003; Chen and Manning, 2014). Our compression system maintains a state representing the current compression, which is repeatedly modified by a small set of operations. We formally describe the system (§3.1), present a method for generating oracle paths to gold standard shortenings (§3.2), and detail a computational model of oracle compression (§3.3). We then demonstrate and evaluate a method which uses this model for constrained compression (§4.2), and analyze its computational costs (§5).

3.1 Formal description

Our transition-based sentence compression system shortens a sentence s by executing a sequence of operations. Each operation modifies the state, denoted $(V, [v|B])$, where V is a subset of tokens from s, and B is an ordered buffer of tokens from s headed by the token v. In defining each operation, we use the notation $T(v)$ to refer to v and all its descendants in the unchanging, precomputed dependency parse of s. We say that $T(v)$ is the subtree rooted at v.

We define four principal operations, which reference the vertex v at the head of the buffer, $[v|B]$. The operation **PRUNE** removes all vertexes in $T(v)$ from V. The operation **INSERT** adds $T(v)$ into V. Executing a PRUNE or INSERT also pops the token v from the head of the buffer. The compression system can also execute the operations **NOPRUNE** and **NOINSERT**, which pop v from the buffer without inserting or pruning $T(v)$.[1]

To generate a compression, we first execute the **START** operation which sets $V = \emptyset$ and defines $B = [v_1, v_2 ... v_{|s|}]$, where $v_1, v_2 ... v_{|s|}$ are the tokens in s, arranged breadth-first from the root of the sentence, and $|s|$ is the token length of the original sentence. After the buffer is empty, we execute a **STOP** operation and all tokens in V are linearized in their original order to return a shortened sentence. Table 2 formally defines all operations. The appendix also includes a complete, worked example.

In our transition-based compressor, only some operations are valid for some configurations of the state. If $v \notin V$, then PRUNE is not valid as $T(v)$ cannot be removed from V. Similarly, if $v \in V$, then INSERT is not valid as $T(v)$ cannot be added to V. Table 2 includes these preconditions in the definition of each operation.

[1] These two operations have different preconditions (Table 2). The distinction between each operation is important for modeling (§3.3).

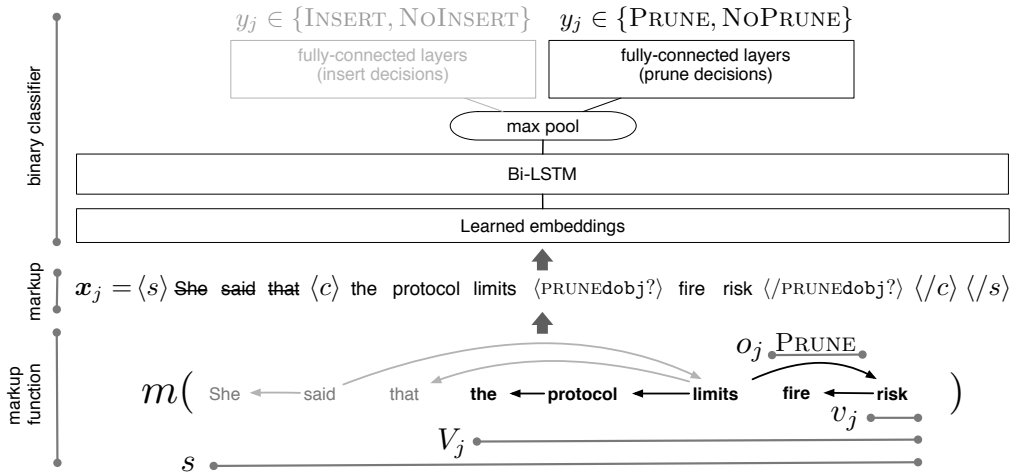

Figure 2: Our model learns to perform binary classification based on the markup $\boldsymbol{x}_j = m(V_j, v_j, o_j, s)$ (§3.3). The model uses two separate sequences of fully-connected layers: one is responsible for binary PRUNE decisions, the other for binary INSERT decisions. The appropriate task for the LSTM (i.e. prune decision vs. insert decision) is always clear from context (§3.2). The markup "shows" the LSTM what a proposed change to the state of the compression system would "look" like if the operation proposal o_j were to be carried out. At each timestep, LSTM must decide if the stateful compression system should accept the operation proposal, $o_j \in \{\text{PRUNE}, \text{INSERT}\}$. In this case, s is "She said that the protocol limits fire risk" and V_j is "the protocol limits fire risk". The LSTM must decide if the compression system should execute o_j= PRUNE on the subtree headed at v_j = *risk*, governed by a `dobj` edge in the parse of s.

3.2 Oracle paths

We identified the operations in our compression system empirically: we found that for all compressions in a large, standard corpus (Filippova and Altun, 2013) there exists an oracle path of at most $|s|$ operations which can fully reconstruct the shortened sentence, where $|s|$ denotes the token length of the original sentence.

We identify the oracle path by executing a sequence of operations. Let $(V_j, v_j|B)$ denote the state at timestep j. The oracle operation at step j is unambiguously determined by c_g, the vertexes in the gold compression. For instance, if $v_j \in V_j$ but $v_j \notin c_g$, then the oracle must execute PRUNE. This is because B is arranged breadth-first, so our system can only remove v_j at timesteps $j' \leq j$. If the vertex is not pruned at timestep j, the system will never have an opportunity to remove it and will never recover c_g. Similarly, if $v_j \in c_g \cap V$ then the oracle operation at timestep j must be NOPRUNE. This is because if the system were to prune v_j at timestep j, it will not have the opportunity to reinsert the vertex again at a later timestep. We use analogous reasoning to identify oracle INSERT and NOINSERT operations. By executing each oracle operation in sequence, we compress s to c_g and identify the oracle path.

Because it is only possible to prune v_j if $v_j \in V$, and only possible to insert v_j if $v_j \notin V$ (§3.1), we interpret the oracle path as a sequence of binary decisions. If $v_j \in V_j$, the compression system must decide to execute PRUNE or NOPRUNE. If $v_j \notin V_j$, it must decide to execute INSERT or NOINSERT. We use this interpretation of the oracle path during modeling.

3.3 Modeling

In our framework (§3.2), the oracle compression process is a series of binary decisions. At each step j the oracle must decide to execute (or not execute) some operation proposal, $o_j \in \{\text{PRUNE}, \text{INSERT}\}$. We train an LSTM (Hochreiter and Schmidhuber, 1997) to predict the binary oracle decision $y_j \in \{0, 1\}$ where $y_j = 1$ indicates that the compression system accepts the proposal at timestep j (e.g. executes PRUNE) and $y_j = 0$ indicates that the compression system does not accept the proposal (e.g. executes NOPRUNE).

To train the LSTM, we encode both the state of the compression system and the operation proposal into a sequence of input symbols $\boldsymbol{x}_j$, called the markup. The LSTM predicts $p(y_j = 1|\boldsymbol{x}_j)$, the probability of a binary oracle decision, given the markup.

The markup is the deterministic (rather

than learned) output of a function $\boldsymbol{x}_j = m(V_j, v_j, o_j, s)$, which inserts additional symbols into s, the original sequence of tokens in the sentence (see Figure 3). The markup function adds four kinds of symbols.

1. **Deletion indicators** tag tokens from s which are not included in the compression at timestep j. (These indicators are denoted with strikethrough text in Figure 3.) The markup function concatenates a deletion indicator to all $t_i \in \{s\} \setminus V_j$, where $\{s\}$ denotes the unordered set of tokens from s.

2. **Sentence brackets** represent the start and end of the original, uncompressed sentence s. Sentence brackets are shown with $\langle s \rangle$ and $\langle /s \rangle$ in Figure 3.

3. **Compression brackets** show the location of a compression within s. If the operation proposal is PRUNE, these brackets show the start and end of the compression V_j within s. If the operation proposal is INSERT, these brackets show the start and end of the compression V_{j+1}, if the proposed operation were to be accepted at timestep j. Compression brackets are shown with $\langle c \rangle$ and $\langle /c \rangle$ in Figure 3.

   In some cases, some of the tokens in the span from $\langle c \rangle$ to $\langle /c \rangle$ might not be in V_j. (For instance, a system might have pruned a modifier in this span at an earlier timestep.) The markup function will tag these tokens (among others) with deletion indicators.

4. **Subtree brackets** represent the start and end of $T(v)$, defined previously (§3.1) as the tokens in the subtree which would be pruned or inserted by the operation proposal o_j. Subtree brackets have a more complex structure which encodes (i) the type of the operation proposal and (ii) the syntactic role of $T(v)$ within s. Figure 3 shows the subtree brackets $\langle$PRUNE`dobj?`$\rangle$ and $\langle /$PRUNE`dobj?`$\rangle$. Details of the structure of these brackets are provided in the appendix.

Including the three sets of inline bracket symbols, along with the deletion indicator symbols, allows the LSTM to model the relationship between (a) the tokens to be pruned or inserted, (b) the current compression, (c) the operation proposal, and (d) the original sentence s. Prior work shows that LSTMs appear capable of modeling bracketing within a sentence (Vinyals et al., 2015; Karpathy et al., 2016; Aharoni and Goldberg, 2017).

Our architecture (Figure 3) follows recent work on LSTM classification for sentence-level tasks (Conneau et al., 2017). Specifically, we predict binary, oracle operations using a Bi-LSTM layer, a max pooling layer, and two separate sequences of fully-connected layers: one for PRUNE vs. NO-PRUNE decisions, and one for INSERT vs. NOINSERT decisions. We interpret the shared Bi-LSTM layer as learning deep features from the markup, and each sequence of fully-connected layers as using those features to perform a different kind of binary classification.

3.4 Training

We use the training set from a large corpus of sentence–compression pairs (Filippova and Altun, 2013) to define a set of 8 million training instances of the form $(\boldsymbol{x}_j, y_j)$, where $\boldsymbol{x}_j$ is the markup at timestep j on some oracle path, and $y_j \in \{0, 1\}$ is the binary oracle choice at timestep j.

We initialize with GloVe embeddings (Pennington et al., 2014) and train using weighted cross entropy loss with Adam (Kingma and Ba, 2014). We weight the cost function in proportion to the prevalence of each class in our training set; the appendix contains additional details. Word vectors are updated during training. We train for 20 epochs, with early stopping if validation accuracy does not improve within 2 epochs.

Our model includes several hyperparameters including: the width of the max pooling layer, the learning rate, the Adam weight decay setting, the activation function, the dropout rate in the fully-connected layers and the dimensionality of input embeddings. The final parameters are included in a table in the appendix. We tuned parameters by first searching coarsely and at random over the parameter space (Bergstra and Bengio, 2012); and then searching finely and at random over smaller regions of the parameter space which achieve high decision-level accuracy. Our best model achieves a validation accuracy of 94.5% on a set of 100,000 held-out oracle operations in its highest-performing epoch. Our best model also achieves a token-level F1 score of 0.788 on test data (described in §4), for the unconstrained compression task. We use the learned parameters from the best model in subsequent constrained com-

pression experiments. We implement with AllenNLP 0.7.1 (Gardner et al., 2017).

4 Automatic evaluation of constrained compression

We automatically evaluate different approaches to constrained compression, using token-level F1 to measure how well each compression method can reconstruct a known-good shortening. (Token-level F1 is the standard automatic evaluation metric for the compression task.) We also examine the readability of compressions with the automated metric SLOR (§4.4).

Our method better reconstructs known-good compressions, and achieves higher automatic readability scores (Table 3). We evaluate the significance of each difference with bootstrap sampling (Berg-Kirkpatrick et al., 2012). The differences are significant ($p < 10^{-2}$).

4.1 Synthetic constrained compressions dataset

In order to evaluate different approaches to constrained compression, we require an evaluation dataset of sentences, constraints and known-good shortenings (which respect the constraints). Formally, this means we need input (s, q, b) where s is a sentence, q is a list of query tokens and $b \in \mathbb{Z}^+$ is a maximum character budget. For each input triple (s, q, b) we need an output compression c, where c is a known-good shortening that (1) includes all tokens from q and (2) is no longer than b (by character length).

To support large-scale automatic evaluation, we reinterpret Filippova and Altun's standard compression corpus as a collection of input triples and output compressions. In 6,827 of the 10,000 (s, c) pairs in its test set, c includes one or more named entities.[2] We interpret these named entities as the query q, and we interpret the length of c (in characters) as the character budget b. This defines our test set of 6,827 tuples, each of the form (s, q, b, c). Each tuple consists of a sentence, a lexical constraint, a budget constraint and a known-good compression.

[2]We re-tokenize, parse and tag NER spans from the original dataset with Stanford CoreNLP 3.8.0 (Manning et al., 2014). We use a standard 3-class definition of NER; for our purposes, entities are people, locations or organizations. Prior work transforms compressions during preprocessing (Filippova and Altun, 2013). We reimplement these transformations using UD (see Appendix), after parsing and tagging the untransformed compressions from the dataset.

Approach	F1	SLOR
Query terms only (lower bound)	0.381	-0.088
Supervised ILP	0.854	0.776
Transition-based deletion	**0.875**	**0.800**

Table 3: Test F1 scores for two compression methods, for the constrained compression task. We also show scores for SLOR, an automated readability metric (§4.4). The difference in token-level F1 scores and SLOR scores for each compression method is statistically significant ($p < 10^{-2}$). Selecting only query terms for a compression achieves an F1 = 0.381, which is the lower bound for this task.

4.2 Implementation: transition-based deletion

Our model of transition-based sentence compression (§3.3) predicts the oracle operation y, given the state $(V, [v|B])$. There are many possible ways to use $p(y = 1|\boldsymbol{x})$ in a query-focused and length-constrained compression system. In this work, we use a simple, greedy, iterative **transition-based deletion** technique: at each step j we PRUNE the subtree rooted at

$$\underset{v \in V, q \notin T(v)}{\arg\max}\ p(y_j = 1|\boldsymbol{x}_j)$$

where $\boldsymbol{x}_j$ is the markup and y_j is the binary decision to perform a PRUNE or NOPRUNE operation (§3.3). We continue pruning subtrees until the length of the linearized tokens in V is less than b, in which case we stop compression.

We initialize $(V, [v|B])$ with the smallest subtree from the dependency parse of s which is (1) rooted at a verb and (2) contains all of the tokens in q. For roughly two-thirds of sentences V is simply equal to all tokens in the original sentence. In the remaining cases, all tokens in q are contained in some subclause of the sentence; the compression is formed by shortening this subclause, instead of shortening the whole sentence. In more than 95% of sentences, the compression system must make additional PRUNE decisions after initializing with the smallest subtree.

4.3 Implementation: ILP-based compression

We compare our system to a state-of-the-art, ILP-based method, presented in Filippova and Altun (2013). This approach proposes representing each edge in a syntax tree with a vector of binary features, then learning weights for each feature using a structured perceptron trained on a corpus of

(s, c) pairs. Learned weights are used to compute a global compression objective, subject to structural constraints which ensure the output is a valid tree.

To our knowledge, a public implementation of this method does not exist. We reimplement from scratch using Gurobi Optimization (2018), achieving a test-time, token-level F1 score of 0.690 on the unconstrained compression task. The F1 score in our reimplementation is lower than the than the result reported by the original authors. There are some important differences between our reimplementation and the method reported in Filippova and Altun (2013). We describe these differences in detail in the appendix.

4.4 Importance and readability evaluation

Researchers often use human judgements of *importance* and *readability* to evaluate extractive sentence compression techniques (Knight and Marcu, 2000; Clarke and Lapata, 2008; Filippova et al., 2015).

In a traditional importance evaluation, humans judge the degree to which a compression retains the most "important" information from a source sentence. However, in our constrained compression setting, a user's query determines the "important" information from s, which must be included in a compression. Thus, human importance evaluations are inappropriate.

We use the automated SLOR metric (Lau et al., 2015) to check the readability of compressions. Prior work shows that this metric correlates with human readability judgements for the compression task (Kann et al., 2018). SLOR normalizes the probability of a token sequence assigned from a language model, by adjusting for both the probability of the individual unigrams in the sentence and for the sentence length.[3] Our method achieves a slightly higher SLOR score, which indicates that it might produce slightly more readable compressions (Table 3). The appendix contains additional details of our implementation of SLOR.

5 Computational costs of transition-based compression

ILP-based approaches to sentence compression formalize the task as an integer linear programming optimization task, a well-known NP-hard problem with exponential worst-case complexity (§2.1). One advantage of our transition-based framework is that it can perform query-focused, budget-constrained compression in worst-case quadratic time.

We define one unit of computational cost to be the computation required to evaluate one subtree for possible pruning. This cost scales with the token length of the sentence: the more tokens in a sentence, the more subtrees must be considered for deletion.

In the worst-case, our method will prune one singleton subtree (i.e. a subtree with one vertex) at each of the N timesteps during compression.[4] If a dependency parse of s contains $|s|$ original vertexes, and $|s_i|$ refers to the token length of the remaining sentence at timestep i, then the transition-based deletion method requires at most $\sum_{i=0}^{|s|} |s_i| = O(|s|^2)$ possible operations, where it is always the case that $|s_{i+1}| = |s_i| - 1$ because one token is removed at each step. (This analysis ignores the effect of query and budget constraints on worst-case complexity, which are examined in the following section.)

5.1 Empirical costs of transition-based compression

The theoretical worst-case is a poor representation of the empirical costs of our transition-based deletion algorithm. In the worst-case, transition-based deletion only prunes one vertex from a singleton subtree at each timestep. But in practice, the method often prunes large subtrees which remove many vertexes in a single step.

We therefore measure the empirical costs of our algorithm during the experiment described in Section 4. We count the total number of vertexes considered for possible pruning during the actual compression process, for each sentence in the experiment. We also compute the total number of such operations during an actual, worst-case run of the algorithm. (In a worst case run, only a single token[5] is removed at each timestep, without removing query tokens, until the character budget is satisfied.)

We find that while transition-based deletion is polynomial in the worst case (§5), in practice the

[3] Longer sentences are always less probable than shorter sentences; rarer words make a sequence less probable.

[4] Intuitively, pruning in this manner would remove a leaf from the (modified) parse tree at each timestep.

[5] Tokens will have different character lengths. We choose the single vertex for removal at random, from among all pruneable singleton subtrees.

computational cost scales linearly in the token length of the input sentence (Figure 3). The observed worst case appears to obey the theoretical upper bound of $O(|s|^2)$ operations.

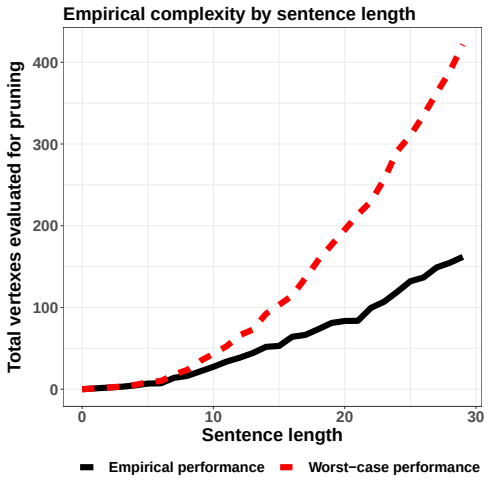

Figure 3: Mean observed operations in a worst case run vs. mean observed operations in an average run of the transition-based deletion algorithm, at different sentence lengths. In the worst case (top), our method is polynomial in the token length of the input sentence (§5). In practice (bottom), empirical costs scale linearly for constrained compression (§5.1).

6 Applications and related work

6.1 Applications

Traditional study of sentence compression is motivated by text summarization techniques, which create synopses by selecting and (sometimes) shortening sentences (Knight and Marcu, 2000; Vanderwende et al., 2007; Martins and Smith, 2009). In these settings, it is important for compressions to retain "important" information from sources because they must stand-in for longer sentences within summaries.

Our concern with constrained compression is better suited to search applications, in which user input defines important information in source documents. For instance, constrained compression could be used to generate query-biased snippets on a search engine results page (Tombros and Sanderson, 1998; Metzler, 2008; Kanungo and Orr, 2009).[6] Constrained compressions could also be used as part of new forms of search user interfaces (Hearst, 2009), such as concept map browsers (Falke and Gurevych, 2017). Particular forms of query-focused summarization, like summarizing people (Zhou et al., 2004) or companies (Filippova et al., 2009), also require compressions with hard lexical constraints.

[6]We note that version 7.5 of Apache Lucene, the leading open source search engine, does not perform sentence compression in generating snippets. https://lucene.apache.org/core/7_5_0/highlighter/index.html

6.2 Related work

To our knowledge, this work is the first study of length and lexically constrained compression. We formalize the problem as a strictly extractive task, following a line of research in which compressions are formed by deleting tokens from an original sentence (Clarke and Lapata, 2008; Filippova and Strube, 2008; Filippova et al., 2015). However, our transition-based framework might be extended with abstractive operations in future work, following recent interest in blending summarization techniques (See et al., 2017). Other approaches compress sentences by generating instead of deleting words (Rush et al., 2015; Mallinson et al., 2018).

Finally, Li et al. (2013) solicit annotations for "guided" compression, in which humans are directed to include particular words in manually shortening sentences. The authors hypothesize that this form of supervision can improve downstream summarization tasks, but do not examine the compression problem under lexical constraints.

7 Conclusion and future work

This work introduces a new, neural, transition-based method for extractive sentence compression, in the spirit of early syntactic approaches to the compression task (Jing, 2000; Knight and Marcu, 2000). We show that this top-down approach is both more computationally efficient than ILP-based methods, and better reconstructs known-good sentence shortenings.

In future work, we plan to examine bottom-up compression methods, which start from the leaves of the original sentence and proceed to the root. Such methods might require a smaller number of local decisions about adding (or not adding) vertexes, leading to greater computational efficiency.

References

Roee Aharoni and Yoav Goldberg. 2017. Towards string-to-tree neural machine translation. In *ACL*.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in NLP. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*.

James Bergstra and Yoshua Bengio. 2012. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

Danqi Chen and Christopher Manning. 2014. A fast and accurate dependency parser using neural networks. In *EMNLP*.

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31:399–429.

Alexis Conneau, Douwe Kiela, Holger Schwenk, Loïc Barrault, and Antoine Bordes. 2017. Supervised learning of universal sentence representations from natural language inference data. In *EMNLP*.

Tobias Falke and Iryna Gurevych. 2017. Graphdocexplore: A framework for the experimental comparison of graph-based document exploration techniques. In *EMNLP: System Demonstrations*.

Katja Filippova, Enrique Alfonseca, Carlos A Colmenares, Lukasz Kaiser, and Oriol Vinyals. 2015. Sentence compression by deletion with LSTMs. In *EMNLP*.

Katja Filippova and Yasemin Altun. 2013. Overcoming the lack of parallel data in sentence compression. In *EMNLP*.

Katja Filippova and Michael Strube. 2008. Dependency tree based sentence compression. In *Proceedings of the Fifth International Natural Language Generation Conference*.

Katja Filippova, Mihai Surdeanu, Massimiliano Ciaramita, and Hugo Zaragoza. 2009. Company-oriented extractive summarization of financial news. In *EACL*.

Matt Gardner, Joel Grus, Mark Neumann, Oyvind Tafjord, Pradeep Dasigi, Nelson F. Liu, Matthew Peters, Michael Schmitz, and Luke S. Zettlemoyer. 2017. Allennlp: A deep semantic natural language processing platform.

LLC Gurobi Optimization. 2018. Gurobi optimizer reference manual.

Marti Hearst. 2009. *Search user interfaces*. Cambridge University Press, Cambridge New York.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural Computation*.

Hongyan Jing. 2000. Sentence reduction for automatic text summarization. In *Proceedings of the Sixth Conference on Applied Natural Language Processing*.

Katharina Kann, Sascha Rothe, and Katja Filippova. 2018. Sentence-Level Fluency Evaluation: References Help, But Can Be Spared! In *CoNNL 2018*.

Tapas Kanungo and David Orr. 2009. Predicting the readability of short web summaries. In *Proceedings of the Second ACM International Conference on Web Search and Data Mining*.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2016. Visualizing and understanding recurrent networks. In *ICLR (Workshop track)*.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980v9.

Kevin Knight and Daniel Marcu. 2000. Statistics-based summarization - step one: Sentence compression. In *AAAI/IAAI*.

Jey Han Lau, Alexander Clark, and Shalom Lappin. 2015. Unsupervised prediction of acceptability judgements. In *ACL*.

Chen Li, Fei Liu, Fuliang Weng, and Yang Liu. 2013. Document summarization via guided sentence compression. In *EMNLP*.

Jonathan Mallinson, Rico Sennrich, and Mirella Lapata. 2018. Sentence Compression for Arbitrary Languages via Multilingual Pivoting. In *EMNLP*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *ACL System Demonstrations*.

Gary Marchionini. 2006. Exploratory search: From finding to understanding. *Commun. ACM*, 49(4).

André FT Martins and Noah A Smith. 2009. Summarization with a joint model for sentence extraction and compression. In *Proceedings of the Workshop on Integer Linear Programming for Natural Langauge Processing*.

Donald Metzler. 2008. Machine learned sentence selection strategies for query-biased summarization. In *SIGIR Workshop on Learning to Rank for Information Retrieval*.

Ning Miao, Hao Zhou, Lili Mou, Rui Yan, and Lei Li. 2019. Constrained sentence generation by metropolis-hastings sampling. In *AAA19*.

Joakim Nivre. 2003. An efficient algorithm for projective dependency parsing. In *International Conference on Parsing Technologies*.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *EMNLP*.

Matt Post and David Vilar. 2018. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. In *NAACL*.

Alexander M. Rush, Sumit Chopra, and Jason Weston. 2015. A neural attention model for abstractive sentence summarization. In *EMNLP*.

Abigail See, Peter J. Liu, and Christopher D. Manning. 2017. Get to the point: Summarization with pointer-generator networks. In *ACL*.

Anastasios Tombros and Mark Sanderson. 1998. Advantages of query biased summaries in information retrieval. In *SIGIR*.

Lucy Vanderwende, Hisami Suzuki, Chris Brockett, and Ani Nenkova. 2007. Beyond sumbasic: Task-focused summarization with sentence simplification and lexical expansion. *Information Processing & Management*, 43(6):1606–1618.

Oriol Vinyals, Lukasz Kaiser, Terry K Koo, Slav Petrov, Ilya Sutskever, and Geoffrey E. Hinton. 2015. Grammar as a foreign language. In *NIPS*.

Liangguo Wang, Jing Jiang, Hai Leong Chieu, Chen Hui Ong, Dandan Song, and Lejian Liao. 2017. Can syntax help? Improving an LSTM-based sentence compression model for new domains. In *ACL*.

Liang Zhou, Miruna Ticrea, and Eduard Hovy. 2004. Multi-document biography summarization. In *EMNLP*.