Polynomial Time Joint Structural Inference for Sentence Compression

Xian Qian and **Yang Liu**
The University of Texas at Dallas
800 W. Campbell Rd., Richardson, TX, USA
{qx,yangl}@hlt.utdallas.edu

Abstract

We propose two polynomial time inference algorithms to compress sentences under bigram and dependency-factored objectives. The first algorithm is exact and requires $O(n^6)$ running time. It extends Eisner's cubic time parsing algorithm by using virtual dependency arcs to link deleted words. Two signatures are added to each span, indicating the number of deleted words and the rightmost kept word within the span. The second algorithm is a fast approximation of the first one. It relaxes the compression ratio constraint using Lagrangian relaxation, and thereby requires $O(n^4)$ running time. Experimental results on the popular sentence compression corpus demonstrate the effectiveness and efficiency of our proposed approach.

1 Introduction

Sentence compression aims to shorten a sentence by removing uninformative words to reduce reading time. It has been widely used in compressive summarization (Liu and Liu, 2009; Li et al., 2013; Martins and Smith, 2009; Chali and Hasan, 2012; Qian and Liu, 2013). To make the compressed sentence readable, some techniques consider the n-gram language models of the compressed sentence (Clarke and Lapata, 2008; McDonald, 2006). Recent studies used a subtree deletion model for compression (Berg-Kirkpatrick et al., 2011; Morita et al., 2013; Qian and Liu, 2013), which deletes a word only if its modifier in the parse tree is deleted. Despite its empirical success, such a model fails to generate compressions that are not subject to the subtree constraint (see Figure 1). In fact, we parsed the Edinburgh sentence compression corpus using the MSTparser[1],

[1]http://sourceforge.net/projects/mstparser/

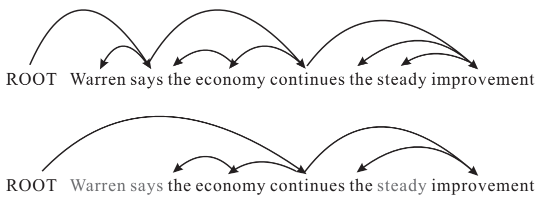
ROOT Warren says the economy continues the steady improvement
ROOT Warren says the economy continues the steady improvement

Figure 1: The compressed sentence is not a subtree of the original sentence. Words in gray are removed.

and found that 2561 of 5379 sentences (47.6%) do not satisfy the subtree deletion model.

Methods beyond the subtree model are also explored. Trevor et al. proposed synchronous tree substitution grammar (Cohn and Lapata, 2009), which allows local distortion of the tree topology and can thus naturally capture structural mismatches. (Genest and Lapalme, 2012; Thadani and McKeown, 2013) proposed the joint compression model, which simultaneously considers the n-gram model and dependency parse tree of the compressed sentence. However, the time complexity greatly increases since the parse tree dynamically depends on the compression. They used Integer Linear Programming (ILP) for inference which requires exponential running time in the worst case.

In this paper, we propose a new exact decoding algorithm for the joint model using dynamic programming. Our method extends Eisner's cubic time parsing algorithm by adding signatures to each span, which indicate the number of deleted words and the rightmost kept word within the span, resulting in $O(n^6)$ time complexity and $O(n^4)$ space complexity. We further propose a faster approximate algorithm based on Lagrangian relaxation, which has $TO(n^4)$ running time and $O(n^3)$ space complexity (T is the iteration number in the subgradient decent algorithm). Experiments on the popular Edinburgh dataset show that

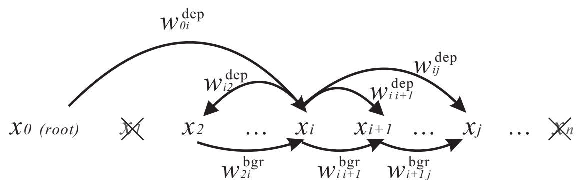

Figure 2: Graph illustration for the objective function. In this example, words x_2, x_i, x_{i+1}, x_j are kept, others are deleted. The value of the objective function is $w_2^{\text{tok}} + w_i^{\text{tok}} + w_{i+1}^{\text{tok}} + w_j^{\text{tok}} + w_{0i}^{\text{dep}} + w_{i2}^{\text{dep}} + w_{ii+1}^{\text{dep}} + w_{ij}^{\text{dep}} + w_{2i}^{\text{bgr}} + w_{ii+1}^{\text{bgr}} + w_{i+1j}^{\text{bgr}}$.

the proposed approach is 10 times faster than a high-performance commercial ILP solver.

2 Task Definition

We define the sentence compression task as: given a sentence composed of n words, $\mathbf{x} = x_1, \ldots, x_n$, and a length $L \le n$, we need to remove $(n - L)$ words from $\mathbf{x}$, so that the sum of the weights of the dependency tree and word bigrams of the remaining part is maximized. Formally, we solve the following optimization problem:

$$\begin{aligned} \max_{\mathbf{z},\mathbf{y}} \quad & \sum_i w_i^{\text{tok}} z_i + \sum_{i,j} w_{ij}^{\text{dep}} z_i z_j y_{ij} \qquad (1) \\ & + \sum_{i<j} w_{ij}^{\text{bgr}} z_i z_j \prod_{i<k<j} (1 - z_k) \\ \text{s.t.} \quad & \mathbf{z} \text{ is binary }, \sum_i z_i = L \\ & \mathbf{y} \text{ is a projective parse tree over the} \\ & \quad \text{subgraph: } \{x_i | z_i = 1\} \end{aligned}$$

where $\mathbf{z}$ is a binary vector, z_i indicates x_i is kept or not. $\mathbf{y}$ is a square matrix denoting the projective dependency parse tree over the remaining words, y_{ij} indicates if x_i is the head of x_j (note that each word has exactly one head). w_i^{tok} is the informativeness of x_i, w_{ij}^{bgr} is the score of bigram $x_i x_j$ in an n-gram model, w^{dep} is the score of dependency arc $x_i \to x_j$ in an arc-factored dependency parsing model. Hence, the first part of the objective function is the total score of the kept words, the second and third parts are the scores of the parse tree and bigrams of the compressed sentence, $z_i z_j \prod_{i<k<j}(1 - z_k) = 1$ indicates both x_i and x_j are kept, and are adjacent after compression. A graph illustration of the objective function is shown in Figure 2.

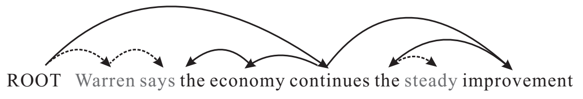

Figure 3: Connect deleted words using virtual arcs.

3 Proposed Method

3.1 Eisner's Cubic Time Parsing Algorithm

Throughout the paper, we assume that all the parse trees are projective. Our method is a generalization of Eisner's dynamic programming algorithm (Eisner, 1996), where two types of structures are used in each iteration, incomplete spans and complete spans. A span is a subtree over a number of consecutive words, with the leftmost or the rightmost word as its root. An incomplete span denoted as I_j^i is a subtree inside a single arc $x_i \to x_j$, with root x_i. A complete span is denoted as C_j^i, where x_i is the root of the subtree, and x_j is the furthest descendant of x_i.

Eisner's algorithm searches the optimal tree in a bottom up order. In each step, it merges two adjacent spans into a larger one. There are two rules for merging spans: one merges two complete spans into an incomplete span, the other merges an incomplete span and a complete span into a large complete span.

3.2 Exact $O(n^6)$ Time Algorithm

First we consider an easy case, where the bigram scores w_{ij}^{bgr} in the objective function are ignored.

The scores of unigrams w_i^{tok} can be transfered to the dependency arcs, so that we can remove all linear terms $w_i^{\text{tok}} z_i$ from the objective function. That is:

$$\begin{aligned} & \sum_i w_i^{\text{tok}} z_i + \sum_{i,j} w_{ij}^{\text{dep}} z_i z_j y_{ij} \\ = \; & \sum_{i,j} (w_{ij}^{\text{dep}} + w_j^{\text{tok}}) z_i z_j y_{ij} \end{aligned}$$

This can be easily verifed. If $z_j = 0$, then in both equations, all terms having z_j are zero; If $z_j = 1$, i.e., x_j is kept, since it has exactly one head word x_k in the compressed sentence, the sum of the terms having z_j is $w_j^{\text{tok}} + w_{kj}^{\text{dep}}$ for both equations.

Therefore, we only need to consider the scores of arcs. For any compressed sentence, we could augment its dependency tree by adding a virtual

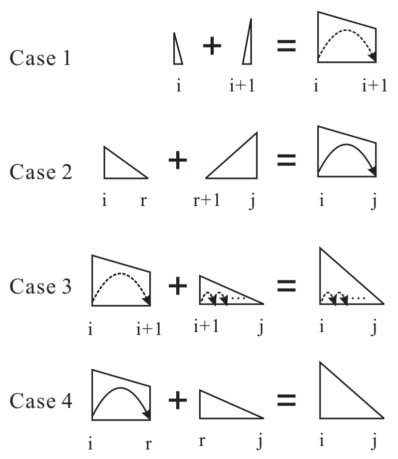

Figure 4: Merging rules for dependency-factored sentence compression. Incomplete spans and complete spans are represented by trapezoids and triangles respectively.

arc $i-1 \rightarrow i$ for each deleted word x_i. If the first word x_1 is deleted, we connect it to the root of the parse tree x_0, as shown in Figure 3. In this way, we derive a full parse tree of the original sentence. This is a one-to-one mapping. We can reversely get the the compressed parse tree by removing all virtual arcs from the full parse tree. We restrict the score of all the virtual arcs to be zero, so that scores of the two parse trees are equivalent.

Now the problem is to search the optimal full parse tree with $n-L$ virtual arcs.

We modify Eisner's algorithm by adding a signature to each span indicating the number of virtual arcs within the span. Let $I_j^i(k)$ and $C_j^i(k)$ denote the incomplete and complete spans with k virtual arcs respectively. When merging two spans, there are 4 cases, as shown in Figure 4.

- **Case 1** Link two complete spans by a virtual arc : $I_{i+1}^i(1) = C_i^i(0) + C_{i+1}^{i+1}(0)$.

  The two complete spans must be single words, as the length of the virtual arc is 1.

- **Case 2** Link two complete spans by a non-virtual arc: $I_j^i(k) = C_r^i(k') + C_{r+1}^j(k''), k' + k'' = k$.

- **Case 3** Merge an incomplete span and a complete span. The incomplete span is covered by a virtual arc: $I_j^i(j-i) = I_{i+1}^i(1) + C_j^{i+1}(j-i-1)$. The number of the virtual arcs within C_j^{i+1} must be $j-i-1$, since the descendants of the modifier of a virtual arc x_j must be removed.

- **Case 4** Merge an incomplete span and a complete span. The incomplete span is covered by a non-virtual arc: $C_j^i(k) = I_r^i(k') + C_j^r(k''), k' + k'' = k$.

The score of the new span is the sum of the two spans. For case 2, the weight of the dependency arc $i \rightarrow j$, w_{ij}^{dep} is also added to the final score. The root node is allowed to have two modifiers: one is the modifier in the compressed sentence, the other is the first word if it is removed.

For each combination, the algorithm enumerates the number of virtual arcs in the left and right spans, and the split position (e.g., k', k'', r in case 2), thus it takes $O(n^3)$ running time. The overall time complexity is $O(n^5)$ and the space complexity is $O(n^3)$.

Next, we consider the bigram scores. The following proposition is obvious.

Proposition 1. *For any right-headed span I_j^i or C_j^i, $i > j$, words x_i, x_j must be kept.*

Proof. Suppose x_j is removed, there must be a virtual arc $j-1 \rightarrow j$ which is a conflict with the fact that x_j is the leftmost word. As x_j is a descendant of x_i, x_i must be kept. $\square$

When merging two spans, a new bigram is created, which connects the rightmost kept words in the left span and the leftmost kept word in the right span. According to the proposition above, if the right span is right-headed, its leftmost word is kept. If the right span is left-headed, there are two cases: its leftmost word is kept, or no word in the span is kept. In any case, we only need to consider the leftmost word in the right span.

Let $I_j^i(k,p)$ and $C_j^i(k,p)$ denote the single and complete span with k virtual arcs and the rightmost kept word x_p. According to the proposition above, we have, for any right-headed span $p = i$.

We slightly modify the two merging rules above, and obtain:

- **Case 2'** Link two complete spans by a non-virtual arc: $I_j^i(k,j) = C_r^i(k',p) + C_{r+1}^j(k'',j), k' + k'' = k$. The score of the new span is the sum of the two spans plus $w_{ij}^{\text{dep}} + w_{p,r+1}^{\text{bgr}}$.

- **Case 4'** Merge an incomplete span and a complete span. The incomplete span is covered by a non-virtual arc. For left-headed spans, the rule is $C_j^i(k,q) = I_r^i(k',p) + C_j^r(k'',q), k' + k'' = k$, and the score of the new span is the sum of the two spans plus w_{pr}^{bgr}; for right-headed spans, the rule is $C_j^i(k,i) = I_r^i(k',i) + C_j^r(k'',r)$, and the score of the new span is the sum of the two spans.

The modified algorithm requires $O(n^6)$ running time and $O(n^4)$ space complexity.

3.3 Approximate $O(n^4)$ Time Algorithm

In this section, we propose an approximate algorithm where the length constraint $\sum_i z_i = L$ is relaxed by Lagrangian Relaxation. The relaxed version of Problem (1) is

$$
\begin{aligned}
\min_{\lambda} \max_{\mathbf{z},\mathbf{y}} \quad & \sum_i w_i^{\text{tok}} z_i + \sum_{i,j} w_{ij}^{\text{dep}} z_i z_j y_{ij} \qquad (2) \\
& + \sum_{i<j} w_{ij}^{\text{bgr}} z_i z_j \prod_{i<k<j} (1 - z_k) \\
& + \lambda(\sum_i z_i - L) \\
\text{s.t.} \quad & \mathbf{z} \text{ is binary} \\
& \mathbf{y} \text{ is a projective parse tree over the} \\
& \quad \text{subgraph: } \{x_i | z_i = 1\}
\end{aligned}
$$

Fixing λ, the optimal $\mathbf{z}, \mathbf{y}$ can be found using a simpler version of the algorithm above. We drop the signature of the virtual arc number from each span, and thus obtain an $O(n^4)$ time algorithm. Space complexity is $O(n^3)$. Fixing $\mathbf{z}, \mathbf{y}$, the dual variable is updated by

$$
\lambda = \lambda + \alpha(L - \sum_i z_i)
$$

where $\alpha > 0$ is the learning rate. In this paper, our choice of α is the same as (Rush et al., 2010).

4 Experiments

4.1 Data and Settings

We evaluate our method on the data set from (Clarke and Lapata, 2008). It includes 82 newswire articles with manually produced compression for each sentence. We use the same partitions as (Martins and Smith, 2009), i.e., 1,188 sentences for training and 441 for testing.

Our model is discriminative – the scores of the unigrams, bigrams and dependency arcs are the linear functions of features, that is, $w_i^{\text{tok}} = \mathbf{v}^T \mathbf{f}(x_i)$, where $\mathbf{f}$ is the feature vector of x_i, and $\mathbf{v}$ is the weight vector of features. The learning task is to estimate the feature weight vector based on the manually compressed sentences.

We run a second order dependency parser trained on the English Penn Treebank corpus to generate the parse trees of the compressed sentences. Then we augment these parse trees by adding virtual arcs and get the full parse trees of their corresponding original sentences. In this way, the annoation is transformed into a set of sentences with their augmented parse trees. The learning task is similar to training a parser. We run a CRF based POS tagger to generate POS related features.

We adopt the compression evaluation metric as used in (Martins and Smith, 2009) that measures the macro F-measure for the retained unigrams (F_{ugr}), and the one used in (Clarke and Lapata, 2008) that calculates the F1 score of the grammatical relations labeled by RASP (Briscoe and Carroll, 2002).

We compare our method with other 4 state-of-the-art systems. The first is linear chain CRFs, where the compression task is casted as a binary sequence labeling problem. It usually achieves high unigram F1 score but low grammatical relation F1 score since it only considers the local interdependence between adjacent words. The second is the subtree deletion model (Berg-Kirkpatrick et al., 2011) which is solved by integer linear programming (ILP)[2]. The third one is the bigram model proposed by McDonald (McDonald, 2006) which adopts dynamic programming for efficient inference. The last one jointly infers tree structures alongside bigrams using ILP (Thadani and McKeown, 2013). For fair comparison, systems were restricted to produce compressions that matched their average gold compression rate if possible.

4.2 Features

Three types of features are used to learn our model: unigram features, bigram features and dependency features, as shown in Table 1. We also use the in-between features proposed by (McDonald et

[2]We use Gurobi as the ILP solver in the paper. http://www.gurobi.com/

Features for unigram x_i
$w_{i-2}, w_{i-1}, w_i, w_{i+1}, w_{i+2}$
$t_{i-2}, t_{i-1}, t_i, t_{i+1}, t_{i+2}$
$w_i t_i$
$w_{i-1}w_i, w_i w_{i+1}$
$t_{i-2}t_{i-1}, t_{i-1}t_i, t_i t_{i+1}, t_{i+1}t_{i+2}$
$t_{i-2}t_{i-1}t_i, t_{i-1}t_i t_{i+1}, t_i t_{i+1} t_{i+2}$
whether w_i is a stopword
Features for selected bigram $x_i x_j$
distance between the two words: $j - i$
$w_i w_j, w_{i-1}w_j, w_{i+1}w_j, w_i w_{j-1}, w_i w_{j+1}$
$t_i t_j, t_{i-1}t_j, t_{i+1}t_j, t_i t_{j-1}, t_i t_{j+1}$
Concatenation of the templates above
$\{t_i t_k t_j \vert i < k < j\}$
Dependency Features for arc $x_h \rightarrow x_m$
distance between the head and modifier $h - m$
dependency type
direction of the dependency arc (left/right)
$w_h w_m, w_{h-1}w_m, w_{h+1}w_m, w_h w_{m-1}, w_h w_{m+1}$
$t_h t_m, t_{h-1}t_m, t_{h+1}t_m, t_h t_{m-1}, t_h t_{m+1}$
$t_{h-1}t_h t_{m-1}t_m, t_h t_{h+1} t_{m-1} t_m$
$t_{h-1}t_h t_m t_{m+1}, t_h t_{h+1} t_m t_{m+1}$
Concatenation of the templates above
$\{t_h t_k t_m \vert x_k$ lies between x_h and $x_m\}$

Table 1: Feature templates. w_i denotes the word form of token x_i and t_i denotes the POS tag of x_i.

al., 2005), which were shown to be very effective for dependency parsing.

4.3 Results

We show the comparison results in Table 2. As expected, the joint models (ours and TM13) consistently outperform the subtree deletion model, since the joint models do not suffer from the subtree restriction. They also outperform McDonald's, demonstrating the effectiveness of considering the grammar structure for compression. It is not surprising that CRFs achieve high unigram F scores but low syntactic F scores as they do not consider the fluency of the compressed sentence.

System	C Rate	F_{uni}	RASP	Sec.
Ours(Approx)	0.68	**0.802**	**0.598**	**0.056**
Ours(Exact)	0.68	0.805	0.599	0.610
Subtree	0.68	0.761	0.575	0.022
TM13	0.68	0.804	0.599	0.592
McDonald06	0.71	0.776	0.561	0.010
CRFs	0.73	0.790	0.501	0.002

Table 2: Comparison results under various quality metrics, including unigram F1 score (F_{uni}), syntactic F1 score (RASP), and compression speed (seconds per sentence). C Rate is the compression ratio of the system generated output. For fair comparison, systems were restricted to produce compressions that matched their average gold compression rate if possible.

Compared with TM13's system, our model with exact decoding is not significantly faster due to the high order of the time complexity. On the other hand, our approximate approach is much more efficient, about 10 times faster than TM13' system, and achieves competitive accuracy with the exact approach. Note that it is worth pointing out that the exact approach can output compressed sentences of all lengths, whereas the approximate method can only output one sentence at a specific compression rate.

5 Conclusion

In this paper, we proposed two polynomial time decoding algorithms using joint inference for sentence compression. The first one is an exact dynamic programming algorithm, and requires $O(n^6)$ running time. This one does not show significant advantage in speed over ILP. The second one is an approximation of the first algorithm. It adopts Lagrangian relaxation to eliminate the compression ratio constraint, yielding lower time complexity $TO(n^4)$. In practice it achieves nearly the same accuracy as the exact one, but is much faster.[3]

The main assumption of our method is that the dependency parse tree is projective, which is not true for some other languages. In that case, our method is invalid, but (Thadani and McKeown, 2013) still works. In the future, we will study the non-projective cases based on the recent parsing techniques for 1-endpoint-crossing trees (Pitler et al., 2013).

Acknowledgments

We thank three anonymous reviewers for their valuable comments. This work is partly supported by NSF award IIS-0845484 and DARPA under Contract No. FA8750-13-2-0041. Any opinions expressed in this material are those of the authors and do not necessarily reflect the views of the funding agencies.

References

Taylor Berg-Kirkpatrick, Dan Gillick, and Dan Klein. 2011. Jointly learning to extract and compress. In *Proceedings of ACL-HLT*, pages 481–490, June.

[3]Our code is available at http://code.google.com/p/sent-compress/

T. Briscoe and J. Carroll. 2002. Robust accurate statistical annotation of general text.

Yllias Chali and Sadid A. Hasan. 2012. On the effectiveness of using sentence compression models for query-focused multi-document summarization. In *Proceedings of COLING*, pages 457–474.

James Clarke and Mirella Lapata. 2008. Global inference for sentence compression: An integer linear programming approach. *J. Artif. Intell. Res. (JAIR)*, 31:399–429.

Trevor Cohn and Mirella Lapata. 2009. Sentence compression as tree transduction. *J. Artif. Int. Res.*, 34(1):637–674, April.

Jason M. Eisner. 1996. Three new probabilistic models for dependency parsing: an exploration. In *Proceedings of COLING*.

Pierre-Etienne Genest and Guy Lapalme. 2012. Fully abstractive approach to guided summarization. In *Proceedings of the ACL*, pages 354–358.

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

Fei Liu and Yang Liu. 2009. From extractive to abstractive meeting summaries: Can it be done by sentence compression? In *Proceedings of ACL-IJCNLP 2009*, pages 261–264, August.

André F. T. 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*, pages 1–9.

Ryan McDonald, Koby Crammer, and Fernando Pereira. 2005. Online large-margin training of dependency parsers. In *Proceedings of ACL*.

Ryan McDonald. 2006. Discriminative Sentence Compression with Soft Syntactic Constraints. In *Proceedings of EACL*, April.

Hajime Morita, Ryohei Sasano, Hiroya Takamura, and Manabu Okumura. 2013. Subtree extractive summarization via submodular maximization. In *Proceedings of ACL*, pages 1023–1032, August.

Emily Pitler, Sampath Kannan, and Mitchell Marcus. 2013. Finding optimal 1-endpoint-crossing trees. In *Transactions of the Association for Computational Linguistics, 2013 Volume 1*.

Xian Qian and Yang Liu. 2013. Fast joint compression and summarization via graph cuts. In *Proceedings of EMNLP*, pages 1492–1502, October.

Alexander M Rush, David Sontag, Michael Collins, and Tommi Jaakkola. 2010. On dual decomposition and linear programming relaxations for natural language processing. In *Proceedings of EMNLP*.

Kapil Thadani and Kathleen McKeown. 2013. Sentence compression with joint structural inference. In *Proceedings of the CoNLL*, August.