Fast k-best Sentence Compression

Katja Filippova & Enrique Alfonseca
Google Research
katjaf|ealfonseca@google.com

Abstract

A popular approach to sentence compression is to formulate the task as a constrained optimization problem and solve it with integer linear programming (ILP) tools. Unfortunately, dependence on ILP may make the compressor prohibitively slow, and thus approximation techniques have been proposed which are often complex and offer a moderate gain in speed. As an alternative solution, we introduce a novel compression algorithm which generates k-best compressions relying on local deletion decisions. Our algorithm is two orders of magnitude faster than a recent ILP-based method while producing better compressions. Moreover, an extensive evaluation demonstrates that the quality of compressions does not degrade much as we move from single best to top-five results.

1 Introduction

There has been a surge in sentence compression research in the past decade because of the promise it holds for extractive text summarization and the utility it has in the age of mobile devices with small screens. Similar to text summarization, *extractive* approaches which do not introduce new words into the result have been particularly popular. There, the main challenge lies in knowing which words can be deleted without negatively affecting the information content or grammaticality of the sentence. Given the complexity of the compression task (the number of possible outputs is exponential), many systems frame it, sometimes combined with summarization, as an ILP problem which is then solved with off-the-shelf tools (Martins & Smith, 2009; Berg-Kirkpatrick et al., 2011; Thadani & McKeown, 2013). While ILP formulations are clear and the translation to an ILP problem is often natural (Clarke & Lapata, 2008), they come with a high solution cost and prohibitively long processing times (Woodsend & Lapata, 2012; Almeida & Martins, 2013). Thus, robust algorithms capable of generating informative and grammatically correct compressions at much faster running times are still desirable.

Towards this goal, we propose a novel supervised sentence compression method which combines *local* deletion decisions with a recursive procedure of getting most probable compressions at every node in the tree. To generate the top-scoring compression a single tree traversal is required. To extend the k-best list with a $k+1$th compression, the algorithm needs $O(m \times n)$ comparisons where n is the node count and m is the average branching factor in the tree. Importantly, approximate search techniques like beam search (Galanis & Androutsopoulos, 2010; Wang et al., 2013), are not required.

Compared with a recent ILP method (Filippova & Altun, 2013), our algorithm is two orders of magnitude faster while producing shorter compressions of equal quality. Both methods are supervised and use the same training data and features. The results indicate that good readability and informativeness, as perceived by human raters, can be achieved without impairing algorithm efficiency. Furthermore, both scores remain high as one moves from the top result to the top five. To our knowledge we are the first to report evaluation results beyond single best output.

To address cases where local decisions may be in-sufficient, we present an extension to the algorithm where we tradeoff the guarantee of obtaining the top scoring solution for the benefit of scoring a node subset as a whole. This extension only moderately affects the running time while eliminating a source of suboptimal compressions.

Comparison to related work Many compression systems have been introduced since the very first approaches by Grefenstette (1998), Jing & McKeown (2000) and Knight & Marcu (2000). Almost all of them make use of syntactic information (e.g., Clarke & Lapata (2006), McDonald (2006), Toutanova et al. (2007)), and our system is not an exception. Like Nomoto (2009), Wang et al. (2013) we operate on syntactic trees provided by a state-of-the-art parser. The benefit of modifying a given syntactic structure is that the space of possible compressions is significantly constrained: instead of all possible token subsequences, the search space is restricted to all the subtrees of the input parse. While some methods *rewrite* the source tree and produce an alternative derivation at every consituent (Knight & Marcu, 2000; Galley & McKeown, 2007), others *prune* edges in the source tree (Filippova & Strube, 2008; Galanis & Androutsopoulos, 2010; Wang et al., 2013). Most of these approaches are supervised in that they learn from a parallel compression corpus either the rewrite operations, or deletion decisions. In our work we also adopt the pruning approach and use parallel data to estimate the probability of deleting an edge given context.

Several text-to-text generation systems use ILP as an optimization tool to generate new sentences by combining pieces from the input (Clarke & Lapata, 2008; Martins & Smith, 2009; Woodsend et al., 2010; Filippova & Altun, 2013). While off-the-shelf general purpose LP solvers are designed to be fast, in practice they may make the compressor prohibitively slow, in particular if compression is done jointly with summarization (Berg-Kirkpatrick et al., 2011; Qian & Liu, 2013; Thadani, 2014). Recent improvements to the ILP-based methods have been significant but not dramatic. For example, Thadani (2014) presents an approximation technique resulting in a 60% reduction in average inference time. Compared with this work, the main practical advantage of our system is that it is very fast without trading compression quality for speed improvements. On the modeling side, it demonstrates that local decisions are sufficient to produce an informative and grammatically correct sentence.

Our recursive procedure of generating k best compressions at every node is partly inspired by frame semantics (Fillmore et al., 2003) and its extension from predicates to any node type (Titov & Klementiev, 2011). The core idea is that there are two components to a high-quality compression at every node in the tree: (1) it should keep all the essential arguments of that node; (2) these arguments should themselves be good compressions. This motivates an algorithm with a recursively defined scoring function which allows us to obtain k-best compressions nearly as fast as the single best one. In this respect our algorithm is similar to the k-best parsing algorithm by Huang & Chiang (2005).

2 The Top-down Approach

Our approach is syntax-driven and operates on dependency trees (Sec. 2.1). The input tree is pruned to obtain a valid subtree from which a compression is read off. The pruning decisions are carried out based on predictions of a maximum entropy classifier which is trained on a parallel corpora with a rich feature set (Sec. 2.2). Section 2.3 explains how to generate the single, top-scoring compression; Section 2.4 extends the idea to arbitrary k.

2.1 Preprocessing

Similar to previous work, we have a special treatment for function words like determiners, prepositions, auxiliary verbs. Unsurprisingly, dealing with function words is much easier than deciding whether a content word can be removed. Approaches which use a constituency parser and prune edges pointing to constituents, deal with function words implicitly (Berg-Kirkpatrick et al., 2011; Wang et al., 2013). Approaches which use a dependency representation either formulate hard constraints (Almeida & Martins, 2013), or collapse function words with their heads. We use the latter approach and transform every input tree (Nivre, 2006) following Filippova & Strube (2008) and also add edges from the dummy root to finite verbs. Finally, we run an entity tagger and collapse nodes referring to entities.

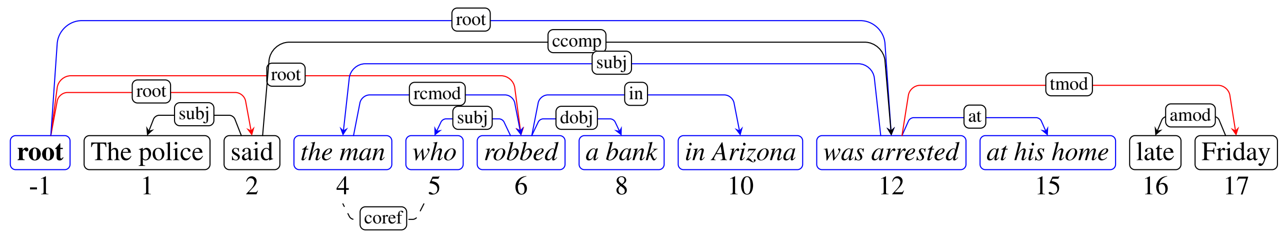

Figure 1: Transformed parse tree of the example sentence. The compression subtree is highlighted with blue color.

Figure 1 provides an example of a transformed tree with extra edges from the dummy root node and an undirected coreference edge for the following sentence to which we will refer throughout this section:

(1) The police said the man who robbed a bank in Arizona was arrested at his home late Friday.

2.2 Estimating deletion probabilities

The supervised component of our system is a binary maximum entropy classifier (Berger et al., 1996) which is trained to estimate the probability of *deleting* an edge given its local context. In what follows, we are going to refer to two probabilities, $p_{del}(e)$ and $p_{ret}(e)$:

$$p_{del}(e_{n,m}) + p_{ret}(e_{n,m}) = 1, \tag{1}$$

where *del* stands for *deleting*, *ret* stands for *retaining* edge *e* from node *n* to node *m*, and $p_{del}(e_{n,m})$ is estimated with MaxEnt.

The features we use are inspired by most recent work (Almeida & Martins, 2013; Filippova & Altun, 2013; Wang et al., 2013) and are as follows:

syntactic: edge labels for the child and its siblings; NE type and PoS tags;

lexical: head and child lemmas; negation; concatenation of parent lemmas and labels;

numeric: depth; node length in words and characters; children count for the parent and the child.

Note that no feature refers to the compression generated so far and therefore the probability of removing an edge needs to be calculated only once on a first tree traversal.

Assuming that we have a training set comprising pairs of a transformed tree, like the one in Figure 1, and a compression subtree (e.g., the subtree covering all the nodes from *the man* to *at his home*), the compression subtrees provide all the negative items for training (blue edges in Fig. 1). The positive items are all other edges originating from the nodes in the compression (red edges). The remaining edges (black) cannot be used for training.

Although we chose to implement hard constraints for function words (see Sec. 2.1 above), we could also apply no tree transformations and instead expect the classifier to learn that, e.g., the probability of deleting an edge pointing to a determiner is zero. However, given the universality of these rules, it made more sense to us to encode them as preprocessing transformations.

2.3 Obtaining top-scoring compression

To find the best compression of the sentence we start at the dummy root node and select a child n with the highest $p_{ret}(e_{root,n})$. The root of the example tree in Figure 1 has three children (*said*$_2$, *robbed*$_6$, *was arrested*$_{12}$). Assuming that p_{ret}'s for the three predicates are .07, .5, .9, the third child is selected. From there, we recursively continue in a top-down manner and at every node n whose children are $M = \{m_1, m_2, ...\}$ search for a children subset $C_n \subseteq M$ maximizing

$$score(C_n) = \sum_{m \in M \setminus C_n} \log p_{del}(e_{n,m}) + \sum_{m \in C_n} \log p_{ret}(e_{n,m}). \tag{2}$$

Since p_{del} and p_{ret} sum to one, this implies that every edge with $p_{ret} < 0.5$ is deleted. However, we can take any $\rho \in [0, 1]$ to be a threshold for deciding between keeping vs. deleting an edge and linearly scale p_{del} and p_{ret} so that after scaling $\hat{p}_{del} < 0.5$ if and only if $p_{del} < \rho$. Of course, finding a single ρ value that would be universally optimal is hardly possible and we will return to this point in Sec. 3.

Consider the node *was arrested* in Figure 1 and its three children listed in Table 1 with p_{ret} given in brackets.

	was arrested$_{12}$	
the man$_4$ (1.0)	*at his home*$_{15}$ (.22)	*Friday*$_{17}$ (.05)

Table 1: Arguments of *was arrested* with their p_{ret}'s.

With $\rho = 0.5$, the top scoring subset is $C_{12} = \{4\}$, its score being $0 + \log .78 + \log .95$. The next step is to decide whether node 4 (*the man*) should retain its relative clause modifier or not. There is no need to go further down the *Friday* node and consider the score of its sole argument (*late*).

2.4 From top-scoring to k-best

A single best compression may appear too long or too short, or fail to satisfy some other requirement. In many cases it is desirable to have a pool of *k-best* results to choose from and in this subsection we will present our algorithm for efficiently generating a *k*-best list (summarized in Fig. 2).

First, let us slightly modify the notation used up to this point to be able to refer to the *k*th best result at node *n*. Instead of $C_n \subseteq M$, we are going to use C_n^k, where $k \in \mathbb{N} \cup \{-1\}$. Unlike C_n, every C_n^k is an ordered sequence of exactly $|M|$ elements, corresponding to *n*'s children:

$$C_n^k = [C_{m_1}^{k_1}, C_{m_2}^{k_2}, ..., C_{m_{|M|}}^{k_{|M|}}]. \quad (3)$$

For every child m_i not retained in the compression, the superscript k_i is *-1*. For example, for the singleton subset C_{12} containing only node 4 in the previous subsection the corresponding best result C_{12}^0 is:

$$C_{12}^0 = [C_4^0, C_{15}^{-1}, C_{17}^{-1}]. \quad (4)$$

Note that at this point we do not need to know what C_4^0 actually is. We simply state that the best result for node 12 must include the best result for node 4.

The scoring function for C_n^k is the averaged sum of the scores of *n*'s chlidren and must be decreasing over $k \geq 0$ ($score(C_n^{k+a}) \leq score(C_n^k), a > 0$):

$$score(C_n^k) = \frac{1}{|M|} \sum_{C_{m_i}^{k_i} \in C_n^k} score(C_{m_i}^{k_i}). \quad (5)$$

When $k \in \{-1, 0\}$, i.e., when we either delete a child or take its best compression, the score is the familiar probabilities:

$$score(C_m^k) = \begin{cases} \log p_{del}(e_{n,m}) & \text{if } k = -1 \\ \log p_{ret}(e_{n,m}) & \text{if } k = 0 \end{cases} \quad (6)$$

Greater values of *k* correspond to *k+1*'th best result at node *n*. Consider again node 12 from Table 1. The *k*-best results at that node may include any of the following variants (the list is not complete):
$[C_4^0, C_{15}^{-1}, C_{17}^{-1}]$, $[C_4^0, C_{15}^0, C_{17}^{-1}]$, $[C_4^2, C_{15}^{-1}, C_{17}^0]$, $[C_4^1, C_{15}^{-1}, C_{17}^1]$, $[C_4^0, C_{15}^0, C_{17}^1]$ $[C_4^{-1}, C_{15}^0, C_{17}^0]$.

How should these be scored so that high quality compressions are ranked higher? Our assumption is that the quality of a compression at any node is subject to the following two conditions:

1. The child subset includes essential arguments and does not include those that can be omitted.
2. The variants for the children retained in the compression are themselves high-quality compressions.

For example, a compression at node 12 which deletes the first child (*the man*) is of a poor quality because it misses the subject and thus violates the first condition. A compression which retains the first node but with a misleading compression, like *the man robbed in Arizona* ($C_4^k = [C_6^l], C_6^l = [C_5^{-1}, C_8^{-1}, C_{10}^0]$), is not good either because it violates the second condition, which is in turn due to the first condition being violated in C_6^l. Hence, a robust scoring function should balance these two considerations and promote variants with good compression at every node retained. Note that for finding the single best result it is sufficient to focus on the first condition only, ignoring the second one, because the best possible result is returned for every child, and the scoring function in Eq. 2 does exactly that. However, once we begin to generate more than a single best result, we start including compressions which may no longer be optimal. So the main challenge in extending the scoring function lies in how to propagate the scores from node's descendants so that both conditions are satisfied.

Given the best result at node *n*, which is obtained in a single pass (Sec. 2.3), the second best result must be one of the following:

- The next best scoring child subset whose score we know how compute from Eq. (5-6) (e.g., for node 12 it would be $[C_4^0, C_{15}^0, C_{17}^{-1}]$, see Eq. 4).
- A subset of the same children as the best one but with one of k_i's which were *0* in the best result increased to *1* (e.g., for node 12 it would be $[C_4^1, C_{15}^{-1}, C_{17}^{-1}]$, see Eq. 4):

No other variant can have a higher score than either of these. Unless there is a tie in the scores, there is a single new second-best subset. And it follows from the decreasing property and the definition of the scoring function that if more than a single k_i is increased from zero, the score is lower than when only one of the k_i's is modified. For example, $score([C_4^2, C_{15}^{-1}, C_{17}^1]) \leq score([C_4^0, C_{15}^{-1}, C_{17}^0]) \leq score([C_4^0, C_{15}^0, C_{17}^{-1}])$, the latter comparison is between two new subsets whose scores can be computed directly from Eq. (5-6). Hence, the second best result C_n^1 is either the next best subset, or one of the at most $|M|$ candidates.

Assuming that $k_j = 0$ in the best result, the score of candidate $C_n^{k_j^*}$ generated from C_n^0 by incrementing k_j is defined as

$$score(C_n^{k_j^*}) = score(C_n^0) + \frac{score(C_{m_j}^{0+1})}{|M|}. \quad (7)$$

Generalizing to an arbitrary *k*, the *k+1*'th result is also either an unseen subset, whose score is defined in Eq. 5, or it can be obtained by increasing a k_i from a non-zero value in one of the *k*-best results generated so far. Given a C_n^k, the score of a candidate generated by incrementing the value of k_j is:

$$\begin{aligned} score(C_n^{k_j^*}) &= score(C_n^k) + \frac{1}{|M|} score(C_{m_j}^{k_j+1}) \\ &- \begin{cases} 0 & \text{if } k_j = 0, \\ \frac{1}{|M|} score(C_{m_j}^{k_j}) & \text{if } k_j > 0. \end{cases} \end{aligned} \quad (8)$$

Notice the similarity between Eq. 7 and Eq. 8. The difference is that when we explore candidates of k_j's greater than zero, we replace the contribution of m_j'th child: the k_j'th best score is replaced with $k_j + 1$'th best score. However, the edge score ($C_{m_j}^0$) is never taken out of the total score of C_n^k. This is motivated by the first of the two conditions above. As an illustratation to this point, consider the predicate from Table 1 one more time and assume that $p_{ret}(e_{17,16}) = 0.4$, i.e., the probability of *late* being the argument of *Friday* is 0.4. The information that the temporal modifier (node 17) is an argument with a very low score should not disappear from the subsequent scorings of node 12's candidates. Otherwise a subsequent result may get a higher score than the best one, violating the decreasing property of the scoring function, as the final line below shows:

$[C_4^0, C_{15}^{-1}, C_{17}^{-1}]$	$(0 + \log .78 + \log .95)/3$
$[C_4^0, C_{15}^{-1}, C_{17}^0]$	$(0 + \log .78 + \log .05)/3$
$[C_4^0, C_{15}^{-1}, C_{17}^1]$	$(0 + \log .78 + \log .05 + \log .4)/3$
$[C_4^0, C_{15}^{-1}, C_{17}^1]^*$	$(0 + \log .78 + \log .4)/3.$

To sum up, we have defined a monotonically decreasing scoring function and outlined our algorithm for generating *k*-best compressions (see Fig. 2). As at every request the pool of candidates for node *n* is extended by not more than $|M| + 1$ candidates, the complexity of the algorithm is $O(k \times N \times m)$ (k times node count times the average branching factor).

3 Adding a Node Subset Scorer

On the first pass, the top-down compressor attempts to find the best possible children subset of every node by considering every child separately and making the retain-or-delete decisions independently of one another. How conservative or aggressive the algorithm is, is determined by a single parameter $\rho \in [0, 1]$ which places a boundary between the two decisions. With smaller values of ρ a low probability of deletion (p_{del}) would suffice for a node child to be removed. Conversely, a greater value of ρ would mean that only children about which the classifier is fairly certain that they must be deleted would be removed.

Unsurprisingly, the value of ρ is hard to optimize as it may be too low or too high, depending on a node. While retaining a child which could be dropped would not result in an ungrammatical sentence, omitting an important argument may make the compression incomprehensible. When doing an error analysis on a development set, we did not encounter many cases where the compression was clearly ungrammatical due to a wrongly omitted argument. However, results like that do have a high cost and thus need to be addressed. Consider the following example:

```
function KBESTCOMPRESS(G, k)
    $C_r^0$ = FINDBESTCOMPRESSION(G)
    $result \leftarrow \{C_r^0\}$, $heaps \leftarrow \{\}$        ▷ Heaps for every node n.
    while $|result| < k$ do
        $result \leftarrow result \cup \{$FINDNEXTBEST($C_r^{|result-1|}$, heaps)$\}$
    end while
    return result
end function

function FINDNEXTBEST($C_n^k$, heaps)
        ▷ Generate candidates by increasing a $k_i$ in the recent result.
    for all $C_{m_i}^{k_i} \in C_n^k$ do
        if $k_i > -1$ then        ▷ Copy the result and update one field.
            $C_n^{k_i^*} \leftarrow C_n^k$
            $C_{m_i}^{k_i+1} \leftarrow$FINDNEXTBEST($C_{m_i}^{k_i}$, heaps)
            $C_n^{k_i^*}[m_i] \leftarrow C_{m_i}^{k_i+1}$, UPDATESCORE($C_n^{k_i^*}$)
            heaps[n] $\leftarrow$ heaps[n] $\cup \{C_n^{k_i^*}\}$
        end if
    end for
    $C_n^{k^*} \leftarrow$ GENERATENEXTBESTSUBSET$(G, n)$
    heaps[n] $\leftarrow$ heaps[n] $\cup \{C_n^{k^*}\}$
    return pop(heaps[n])        ▷ $C_n^{k+1}$, the best candidate for n.
end function

function FINDBESTCOMPRESSION(G)
    $C_r^0 \leftarrow [], best \leftarrow -1, max \leftarrow -1$
    for all $n \in children(root(G))$ do
        $C_r^0 \leftarrow C_r^0 +$ FINDBESTRESULT(n, G)
        if $p_{ret}(e_{r,n}) > max$ then
            $max \leftarrow p_{ret}(e_{r,n}), best \leftarrow n$
        end if
    end for
    for all $n \in children(root(G))$ do
        if $n \neq best$ then $C_r^0[n] \leftarrow C_n^{-1}$
        end if
    end for
    return $C_r^0$        ▷ In the list, only one child is selected.
end function

function FINDBESTRESULT(n, G)
    $C_n^0 \leftarrow []$
    for all $m \in children(n)$ do
        if $p_{ret}(e_{n,m}) \geq 0.5$ then
            $C_n^0 \leftarrow C_n^0 +$ FINDBESTRESULT(m, G)
        else $C_n^0 \leftarrow C_n^0 + C_m^{-1}$
        end if
    end for
    return $C_n^0$
end function
```

Figure 2: Pseudocode of the algorithm for finding *k*-best compressions of graph G. Obvious checks for termination conditions and empty outputs are not included for readability. $C_n^k[m]$ refers to the result for child *m* of *n* in C_n^k. Scoring is defined in Equations 5-8. Similar to Huang & Chiang (2005) we use a heap for efficiency; *heaps[n]* refers to the heap of candidates for node *n*.

(2) Yesterday the world was ablaze with the news that the CEO will step down.

In this sentence, *ablaze* is analyzed as an adverbial modifier of the verb *to be* and the classifier assigns a score of 0.35 to the edge pointing to *ablaze*. With a decision boundary above 0.35, the meaningful part of the predicate is deleted and the compression becomes incomplete. With the boundary at 0.5, the top scoring subset is a singleton containing only the subject. However, there are hardly any cases where the verb *to be* has a single argument, and our algorithm could benefit from this knowledge.

In the extended model, the score of a children subset (Eq. 5) gets an additional summand, $\log p(|C_n^k|)$, where $|C_n^k|$ refers to the number of *n*'s children actually retained in C_n^k, i.e., with $k_i \geq 0$:

$$score^*(C_n^k) = \log p(|C_n^k|) + \frac{1}{|M|} \sum_{C_{m_i}^{k_i} \in C_n^k} score(C_{m_i}^{k_i}). \tag{9}$$

Unfortunately, with the updated formula, we can no longer generate *k*-best compressions as efficiently as before. However, we can keep a beam of *b* subset candidates for every node and select the one maximizing the new score.

To estimate the probability of a children subset size after compression, $p(|C_n^k|)$, we use an averaged perceptron implementation (Freund & Shapire, 1999) and the features described in Sec. 2.2. We do not differentiate between sizes greater than four and have five classes in total (*0, 1, 2, 3, 4*+).

4 Evaluation

The purpose of the evaluation is to validate the following two hypotheses, when comparing the new algorithm with a competitive ILP-based sentence compressor (Filippova & Altun, 2013):

1. The top-down algorithm was designed to perform local decisions at each node in the parse tree, as compared to the global optimization carried out by the ILP-based compressor. We want to verify whether the local model can attain similar accuracy levels or even outperform the global model, and do so not only for the single best but the top *k* results.

2. Automatic ILP optimization can be quite slow when the number of candidates that need to be evaluated for any given input is large. We want to quantify the speed-up that can be attained without a loss in accuracy by taking simpler, local decisions in the input parse tree.

4.1 Evaluation settings

Training, development and test set The aligned sentences and compressions were collected using the procedure described in Filippova & Altun (2013). The training set comprises 1,800,000 items, each item consisting of two elements: the first sentence in a news article and an extractive compression obtained by matching content words from the sentence with those from the headline (see Filippova & Altun (2013) for the technical details). A part of this set was held out for classifiers evaluation and development. For testing, we use the dataset released by Filippova & Altun (2013)[1]. This test set contains 10,000 items, each of which includes the original sentence and the extractive compression and the URL of the source document. From this set, we used the first 1,000 items only, leaving the remaining 9,000 items unseen, reserved for possible future experiments. We made sure that our training set does not include any of the sentences from the test set.

The training set provided us with roughly 16 million edges for training MaxEnt with 40% of positive examples (deleted edges). For training the perceptron classifier we had about 6 million nodes at our disposal with the instances distributed over the five classes as follows:

0	1	2	3	4+
19.5%	40.6%	31.2%	7.9%	1%

Baseline We used the recent ILP-based algorithm of Filippova & Altun (2013) as a baseline. We trained the compressor with all the same features as our model (Sec. 2.2) on the same training data using an averaged perceptron (Collins, 2002). To make this system comparable to ours, when training the model, we did not provide the ILP decoder with the oracle compression length so that the model learned to produce compressions in the absense of length argument. Thus, both methods accept the same input and are comparable.

[1]http://storage.googleapis.com/sentencecomp/compression-data.json

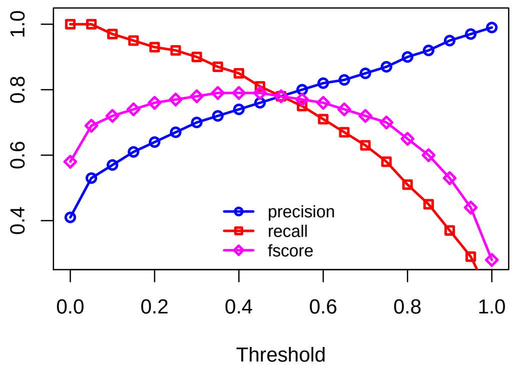

Figure 3: Per-edge precision, recall and F1-score using different thresholds on the prediction values of MaxEnt.

4.2 Automatic evaluation

To measure the quality of the two classifiers (MaxEnt from Sec. 2.2 and perceptron from Sec. 3), we performed a first, direct evaluation of each of them on a small held out portion of the training set. The MaxEnt classifier predicts the probability of deleting an edge and outputs a score between zero and one. Figure 3 plots precision, recall and F1-score at different threshold values. The highest F1-score is obtained at 0.45. Regarding the perceptron classifier that predicts the number of children that we should retain for each node, its accuracy and per-class precision and recall values are given in Table 2.

Acc	0	1	2	3	4+
72.7	69 / 63	75 / 78	76 / 81	60 / 42	44 / 16

Table 2: Accuracy and precision / recall for the classifier predicting the optimal children subset size.

For an automatic evaluation of the quality of the sentence compressions, we followed the same approach as (Riezler et al., 2003; Filippova & Altun, 2013) and measured F1-score by comparing the trees of the generated compressions to the golden, extractive compression. Table 3 shows the results of the ILP baseline and the two variants of the Top-down approach on the test data (TOP-DOWN + NSS is the extended variant described in Sec. 3). The NSS version, which incorporates a prediction on the number of children to keep for each node, is slightly better than the original Top-down approach, but the results are not statistically significant.

	F1-score	*Compr. rate*
ILP	73.9	46.5%
TOP-DOWN	76.7	38.3%
TOP-DOWN + NSS	77.2	38.1%

Table 3: Results for the baseline and our two algorithms.

It is important to point out the difference in compression rates between ILP and TOP-DOWN: 47% vs. 38% (the average compression rate on the test set is 40.5%). Despite a significant advantage due to compression rate (Napoles et al., 2011, see next subsection), ILP performs slightly worse than the proposed methods.

Finally, Table 4 shows the results when computing the F1-score for each of the top-5 compressions as generated by the Top-down algorithms. As can be seen, in both cases there is a sharp drop between the top two compressions but further scores are very close. Since the test set only contains a single oracle compression for every sentence, to understand how big the gap in quality really is, we need an evaluation with human raters.

TOP-DOWN	TOP-DOWN + NSS
76.7; 60.4; 62; 60.9; 59.6	77.2; 60.5; 64; 62.6; 60

Table 4: F1 scores for the top five compressions ($k = 1, 2, 3, 4, 5$).

4.3 Manual evaluation

The first 100 items in the test data were manually rated by humans. We asked raters to rate both readability and informativeness of the compressions for the golden output, the baseline and our systems[2]. For both metrics a 5-point Likert scale was used, and three ratings were collected for every item. Note that in a human evaluation between ILP and TOP-DOWN (+ NSS) the baseline has an advantage because (1) it prunes less aggressively and thus has more chances of producing a grammaticaly correct and informative outputs, and (2) it gets a hint to the optimal compression length in edges. We have used Intra-Class Correlation (ICC) (Shrout & Fleiss, 1979; Cicchetti, 1994) as a measure of inter-judge agreement. ICC for readability was 0.59 (95% confidence interval [0.56, 0.62]) and for informativeness it was 0.51 (95% confidence interval [0.48, 0.54]), indicating fair reliability in both cases.

Results are shown in Tables 5 and 6. As in the automatic evaluations, the two Top-down systems produced indistinguishable results, but both are significantly better than the ILP baseline at 95% confidence. The top-down results are also now indistinguishable from the extractive compressions.

	Readability	Informativeness
EXTRACTIVE	4.33	3.84
ILP	4.20	3.78
TOP-DOWN	4.41	3.91
TOP-DOWN + NSS	4.38	3.87

Table 5: Results of the manual evaluation.

k	ILP	TOP-DOWN	TOP-DOWN + NSS
1	4.20 / 3.78	4.41 / 3.91	4.38 / 3.87
2	3.85 / 3.09	4.11 / 3.31	4.26† / 3.37
3	3.53 / 2.73	4.03† / 3.37†	3.97† / 3.40†
4	3.31 / 2.27	3.80† / 3.16†	3.90† / 3.19†
5	3.00 / 2.42	3.90† / 3.41†	4.12† / 3.41†

Table 6: Readability and informativeness for the top five compressions; † indicates that one of the systems is statistically significantly better than ILP at 95% confidence using a t-test.

4.4 Efficiency

The average per-sentence processing time is 32,074 microseconds (Intel Xeon machine with 2.67 GHz CPU) using ILP, 929 using TOP-DOWN + NSS, and 678 using TOP-DOWN. This means that we have obtained almost a 50x performance increase over ILP. Figure 4 shows the processing time for each of the 1,000 sentences in the test set with sentence length measured in tokens.

For obtaining k-best solutions, the decrease in time is even more remarkable: the average time for generating each of the top-5 compressions using ILP is 42,213 microseconds, greater than that of the single best result. Conversely, the average time for each of the top-5 results decreases to 143 microsec-

[2]The evaluation template and rated sentences are included in the supplementary material.

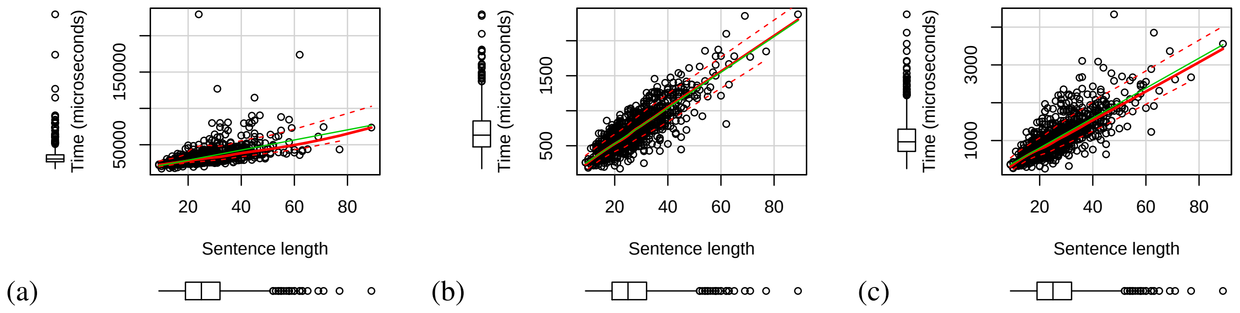

Figure 4: Per-sentence processing time for the test set: (a) ILP; (b) Top-down; (c) Top-down + NSS.

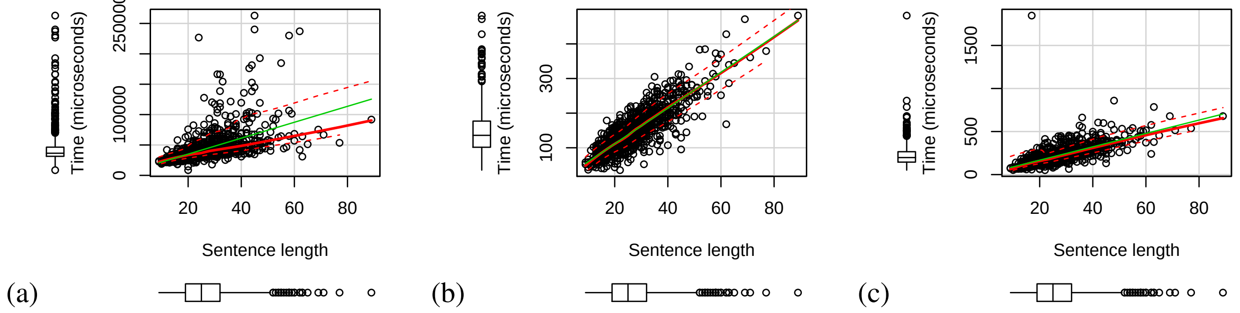

Figure 5: Average processing time for getting all of the top-5 results: (a) ILP; (b) Top-down; (c) Top-down + NSS.

onds using TOP-DOWN, and 195 microseconds using TOP-DOWN + NSS, which means a 300x improvement. The reason is that the Top-down methods, in order to produce the top-ranked compression, have already computed all the per-edge predictions (and the per-node NSS predictions in the case of TOP-DOWN + NSS), and generating the next best solutions is cheap.

5 Conclusions

We presented a fast and accurate supervised algorithm for generating *k*-best compressions of a sentence. Compared with a competitive ILP-based system, our method is 50x faster in generating the best result and 300x faster for subsequent *k*-best compressions. Quality-wise it is better both in terms of readability and informativeness. Moreover, an evaluation with human raters demonstrates that the quality of the output remains high for the top-5 results.

References

Almeida, M. B. & A. F. T. Martins (2013). Fast and robust compressive summarization with dual decomposition and multi-task learning. In *Proc. of ACL-13*.

Berg-Kirkpatrick, T., D. Gillick & D. Klein (2011). Jointly learning to extract and compress. In *Proc. of ACL-11*.

Berger, A., S. A. Della Pietra & V. J. Della Pietra (1996). A maximum entropy approach to natural language processing. *Computational Linguistics*, 22(1):39–71.

Cicchetti, D. V. (1994). Guidelines, criteria, and rules of thumb for evaluating normed and standardized assessment instruments in psychology. *Psychological Assessment*, 6(4):284.

Clarke, J. & M. Lapata (2006). Constraint-based sentence compression: An integer programming approach. In *Proc. of COLING-ACL-06 Poster Session*, pp. 144–151.

Clarke, J. & M. Lapata (2008). Global inference for sentence compression: An integer linear programming approach. *Journal of Artificial Intelligence Research*, 31:399–429.

Collins, M. (2002). Discriminative training methods for Hidden Markov Models: Theory and experiments with perceptron algorithms. In *Proc. of EMNLP-02*, pp. 1–8.

Filippova, K. & Y. Altun (2013). Overcoming the lack of parallel data in sentence compression. In *Proc. of EMNLP-13*, pp. 1481–1491.

Filippova, K. & M. Strube (2008). Dependency tree based sentence compression. In *Proc. of INLG-08*, pp. 25–32.

Fillmore, C. J., C. R. Johnson & M. R. Petruck (2003). Background to FrameNet. *International Journal of Lexicography*, 16:235–260.

Freund, Y. & R. E. Shapire (1999). Large margin classification using the perceptron algorithm. *Machine Learning*, 37:277–296.

Galanis, D. & I. Androutsopoulos (2010). An extractive supervised two-stage method for sentence compression. In *Proc. of NAACL-HLT-10*, pp. 885–893.

Galley, M. & K. R. McKeown (2007). Lexicalized Markov grammars for sentence compression. In *Proc. of NAACL-HLT-07*, pp. 180–187.

Grefenstette, G. (1998). Producing intelligent telegraphic text reduction to provide an audio scanning service for the blind. In *Working Notes of the Workshop on Intelligent Text Summarization,* Palo Alto, Cal., 23 March 1998, pp. 111–117.

Huang, L. & D. Chiang (2005). *Better k-best parsing.* Technical Report MS-CIS-05-08: University of Pennsylvania.

Jing, H. & K. McKeown (2000). Cut and paste based text summarization. In *Proc. of NAACL-00*, pp. 178–185.

Knight, K. & D. Marcu (2000). Statistics-based summarization – step one: Sentence compression. In *Proc. of AAAI-00*, pp. 703–711.

Martins, A. F. T. & N. A. Smith (2009). Summarization with a joing model for sentence extraction and compression. In *ILP for NLP-09*, pp. 1–9.

McDonald, R. (2006). Discriminative sentence compression with soft syntactic evidence. In *Proc. of EACL-06*, pp. 297–304.

Napoles, C., C. Callison-Burch & B. Van Durme (2011). Evaluating sentence compression: Pitfalls and suggested remedies. In *Proceedings of the Workshop on Monolingual Text-to-text Generation,* Prtland, OR, June 24 2011, pp. 91–97.

Nivre, J. (2006). *Inductive Dependency Parsing.* Springer.

Nomoto, T. (2009). A comparison of model free versus model intensive approaches to sentence compression. In *Proc. of EMNLP-09*, pp. 391–399.

Qian, X. & Y. Liu (2013). Fast joint compression and summarization via graph cuts. In *Proc. of EMNLP-13*, pp. 1492–1502.

Riezler, S., T. H. King, R. Crouch & A. Zaenen (2003). Statistical sentence condensation using ambiguity packing and stochastic disambiguation methods for Lexical-Functional Grammar. In *Proc. of HLT-NAACL-03*, pp. 118–125.

Shrout, P. E. & J. L. Fleiss (1979). Intraclass correlations: Uses in assessing rater reliability. *Psychological bulletin*, 86(2):420.

Thadani, K. (2014). Approximating strategies for multi-structure sentence compression. In *Proc. of ACL-14*, p. to appear.

Thadani, K. & K. McKeown (2013). Sentence compression with joint structural inference. In *Proc. of CoNLL-13*, pp. 65–74.

Titov, I. & A. Klementiev (2011). A Bayesian model for unsupervised semantic parsing. In *Proc. of ACL-11*, pp. 1445–1455.

Toutanova, K., C. Brockett, M. Gamon, J. Jagarlamundi, H. Suzuki & L. Vanderwende (2007). The Pythy summarization system: Microsoft Research at DUC 2007. In *Proc. of DUC-07*.

Wang, L., H. Raghavan, V. Castelli, R. Florian & C. Cardie (2013). A sentence compression based framework to query-focused multi-document summarization. In *Proc. of ACL-13*, pp. 1384–1394.

Woodsend, K., Y. Feng & M. Lapata (2010). Title generation with Quasi-Synchronous Grammar. In *Proc. of EMNLP-10*, pp. 513–523.

Woodsend, K. & M. Lapata (2012). Multiple aspect summarization using Integer Linear Programming. In *Proc. of EMNLP-12*, pp. 233–243.