Learning Shortest Paths in Word Graphs*

Emmanouil Tzouridis and Ulf Brefeld†
Technische Universität Darmstadt,
Hochschulstr. 10, 64289 Darmstadt, Germany
{tzouridis,brefeld}@kma.informatik.tu-darmstadt.de

Abstract
In this paper we briefly sketch our work on text summarisation using compression graphs. The task is described as follows: Given a set of related sentences describing the same event, we aim at generating a single sentence that is simply structured, easily understandable, and minimal in terms of the number of words/tokens. Traditionally, sentence compression deals with finding the shortest path in word graphs in an unsupervised setting. The major drawback of this approach is the use of manually crafted heuristics for edge weights. By contrast, we cast sentence compression as a structured prediction problem. Edges of the compression graph are represented by features drawn from adjacent nodes so that corresponding weights are learned by a generalised linear model. Decoding is performed in polynomial time by a generalised shortest path algorithm using loss augmented inference. We report on preliminary results on artificial and real world data.

1 Introduction
In this paper we study the intelligent summarisation of related sentences to quickly serve information needs of users. Given a collection of sentences dealing with the same real-world event, we aim at generating a single sentence that is (i) a summarisation of the input sentences, (ii) simply structured and easily understandable, and (iii) minimal in terms of the number of words/tokens. Traditionally, sentence compression deals with finding the shortest path in word graphs. Filippova [Filippova, 2010] casts the problem as finding a function \( f : \mathcal{X} \times \mathcal{Y} \rightarrow \mathbb{R} \) such that

\[
\hat{y} = \arg\min_y f(x, y)
\]

is the desired output for any input \( x \) [Tschantaridis et al., 2005; Taskar et al., 2004]. The function \( f \) is a linear model in a joint space \( \Phi(x, y) \) of input and output variables and the computation of the argmin is performed by an appropriate decoding strategy such as a shortest-path algorithm.

The remainder is organised as follows. Section 2 introduces preliminaries. Our main contribution is presented in 3. We briefly discuss empirical results in Section 4 and Section 5 concludes.

2 Preliminaries
2.1 Related Work
Barzilay and Lee [Barzilay and Lee, 2003] study sentence compression using dependency trees. Aligned trees are represented by a lattice from which a compression sentence is extracted by an entropy-based criterion over all possible traversals of the lattice. Wan et al. at [Wan et al., 2007] use a language model in combination with maximum spanning trees to rank candidate aggregations that satisfy grammatical constraints.

While the previous approaches to multi-sentence compression are based on syntactic parsing of the sentences, word graph approaches have been proposed, that do not make use of dependency trees or other linguistic concepts. Filippova [Filippova, 2010] casts the problem as finding the shortest path in directed word graphs, where each node is a unique word and directed edges represent the word ordering in the original sentences. The costs of these edges are given by a heuristic that is based on word frequencies. Recently, Boudin and Morin [Boudin and Morin, 2013] propose a re-ranking scheme to identify summarising sentences that contain many keyphrases. The underlying idea is that representative key phrases for a given topic give rise to more informative aggregations.

2.2 Word Graphs and Shortest Paths
Word graphs intend to build a non-redundant representation for possibly redundant sequences by merging identical observations. From a collection of related sentences we iteratively construct a word graph by adding sentences one-by-one as follows: We begin with an empty graph and add the first sentence, where every word in the sentence becomes a
node and a directed edge connects nodes of adjacent words. Words from the next sentences are incorporated by creating a new node for the word or by mapping the word to the corresponding already existing node. A directed edge is inserted to connect the word to its predecessor. We continue until all sentences are incorporated.

Auxiliary words indicating the start (e.g., $x_r$) and the end (e.g., $x_e$) of the sentence are added to the sentences. The sketched procedure merges identical words but preserves the structure of the sentences along the contained paths and the original sentences can often be reconstructed from the compressed representation. Fig. 1 shows related sentences and the corresponding word graph.

The described construction gives us a directed weighted graph $\mathbf{x} = (\mathbf{N}, E)$, where $\mathbf{N}$ is the set of nodes and $E$ the set of edges. As every word graph $\mathbf{x}$ also defines the sets $\mathbf{N}$ and $E$, we will use $\mathbf{N}(\mathbf{x})$ and $E(\mathbf{x})$ in the remainder to denote the set of nodes and edges of graph $\mathbf{x}$, respectively. Every edge $(x_i, x_j) \in E(\mathbf{x})$ is assigned a positive weight given by a cost function $\text{cost} : (x_i, x_j) \mapsto \mathbb{R}^+$. A path $\mathbf{y}$ in the graph $\mathbf{x}$ is a sequence of connected nodes of $\mathbf{x}$ and the cost of such a path is given by the sum of the edge costs for the graph $\mathbf{x}$ (e.g., $\mathbf{y}$ corresponding already existing node. A directed edge is inserting a new node for the word or by mapping the word to the node and a directed edge connects nodes of adjacent words.

"Yahoo buys Tumblr as David Karp stays as CEO", "Yahoo to buy Tumblr for $1.1bn". The corresponding shortest path is highlighted.

![Figure 1: The word graph constructed from the sentences: "Yahoo in rumoured $1.1bn bid to buy white-hot Tumblr", "Yahoo buys Tumblr as David Karp stays as CEO", "Yahoo to buy Tumblr for $1.1bn". The corresponding shortest path is highlighted.](image)

To learn the shortest path, we need to draw features from adjacent nodes in the word graph to learn the score of the connecting edge. Let $x_i$ and $x_j$ be connected nodes of the compression graph $\mathbf{x}$, that is $x_i, x_j \in \mathbf{N}(\mathbf{x})$ and $(x_i, x_j) \in E(\mathbf{x})$. We represent the edge between $x_i$ and $x_j$ by a feature vector $\phi(x_i, x_j)$. A path in the graph is represented as an $n \times n$ binary matrix $\mathbf{y}$ with $n = |\mathbf{N}(\mathbf{x})|$ and elements $y_{ij}$ given by $y_{ij} = \{[(x_i, x_j) \in \text{path}]\}$ where $[\cdot]$ is the indicator function returning one if $\cdot$ is true and zero otherwise. The cost of using the edge $(x_i, x_j)$ in a path is given by a linear combination of those features which is parameterised by $\mathbf{w}$,

$$
cost(x_i, x_j) = \mathbf{w}^\top \phi(x_i, x_j).
$$

Replacing the constant costs by the parameterised ones, we arrive at the following objective function (ignoring the constraints for a moment) that can be rewritten as a generalised linear model.

$$
\sum_{(x_i, x_j) \in E(\mathbf{x})} y_{ij} \mathbf{w}^\top \phi(x_i, x_j) = \mathbf{w}^\top \Phi(\mathbf{x}, \mathbf{y}) = f(\mathbf{x}, \mathbf{y})
$$

Given a word graph $\mathbf{x}$, the shortest path $\hat{\mathbf{y}}$ for a fixed parameter vector $\mathbf{w}$ can now be computed by

$$
\hat{\mathbf{y}} = \arg\min_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}),
$$

where $f$ is exactly the objective of the shortest path algorithm and the argmin consequently computed by an appropriate solver, such as Yen’s algorithm [Yen, 1971].

### 3.2 Learning Shortest Paths with SVMs

In our setting, word graphs $\mathbf{x} \in \mathcal{X}$ and the best summarising sentence $\mathbf{y} \in \mathcal{Y}$ are represented jointly by a feature map $\Phi(\mathbf{x}, \mathbf{y})$ that allows to capture multiple-way dependencies between inputs and outputs. We apply a generalised linear model $f(\mathbf{x}, \mathbf{y}) = \mathbf{w}^\top \Phi(\mathbf{x}, \mathbf{y})$ to decode the shortest path

$$
\hat{\mathbf{y}} = \arg\min_{\mathbf{y}} f(\mathbf{x}, \mathbf{y}),
$$

where the quality of $f$ is measured by the Hamming loss

$$
\Delta_H(\mathbf{y}, \hat{\mathbf{y}}) = \frac{1}{2} \sum_{(x_i, x_j) \in E(\mathbf{x})} [[y_{ij} \neq \hat{y}_{ij}]])
$$

details the differences between the true $\mathbf{y}$ and the prediction $\hat{\mathbf{y}}$, where $[\cdot]$ is again the indicator function from Section 3.1. Using the loss $\Delta_H$, structural support vector machines [Tsochantaridis et al., 2005] minimise the regularised empirical risk

$$
\hat{R}[f] = \|f\|^2 + \sum_{i=1}^{m} \Delta_H\left(\mathbf{y}, \arg\min_{\hat{\mathbf{y}}} f(\mathbf{x}, \hat{\mathbf{y}})\right).
$$

It is often beneficial to rescale the induced margin by the loss to implement the intuition that the confidence of rejecting a mistaken output is proportional to its error. Combining everything, we arrive at the following optimisation problem

$$
\min_{\mathbf{f}, \xi} \|\mathbf{f}\|^2 + \sum_{i=1}^{m} \xi_i
$$

s.t. $\forall i \forall y \neq y_i : f(x_i, y) - f(x_i, y_i) \geq \Delta_H(y_i, y) - \xi_i$

$\forall i : \xi_i \geq 0$

which can be solved in polynomial time by cutting planes. The idea behind cutting planes is to instantiate only a minimal subset of the exponentially many constraints. This is achieved by decoding for every training sample the shortest path using our current model, if this is not the correct path, then it is added to the constrains and the model is updated. If the decoded path is the correct one, we need to
Proof. Omitted for lack of space.

Proposition 1 (Loss augmented inference for shortest path problems). The maximum $y^*$ of $Q$ in Equation (2) can be equivalently computed by minimizing a shortest path problem with $\text{cost}(x_i, x_j) = y_{ij} + w^T \phi(x_i, x_j)$.

Given a parameter vector $w$ and start and end nodes $x_s$ and $x_e$, respectively, the optimisation of $Q$ can be performed with the following linear program.

$$\min_{\bar{y}} \sum_{ij} (y_{ij} + w^T \phi(x_i, x_j)) \bar{y}_{ij}$$

$$\text{s.t.} \forall k \in N(x)/\{s, t\} : \sum_j \bar{y}_{kj} - \sum_i \bar{y}_{ik} = 0$$

$$\sum_j \bar{y}_{sj} - \sum_i \bar{y}_{is} = 1$$

$$\sum_i \bar{y}_{ie} - \sum_j \bar{y}_{ej} = 1$$

$$\forall (i, j) : y_{ij} \leq x_{i(j)}$$

$$\forall (i, j) : y_{ij} \in \{0, 1\}$$

The first constraint guarantees that every inner node of the path must have as many incoming as outgoing edges, the second line of constraints guarantees the path to start in $x_s$ and, analogously, the third line ensures that it terminates in $x_e$. The last line of constraints forces the edges of the path $\bar{y}$ to move along existing paths of $x$.

4 Empirical Results

In this section, we empirically compare learning shortest paths to traditional unsupervised approaches. To this end, we also deploy a structural perceptron [Collins and Duffy, 2002; Altun et al., 2003] as a special-case of the presented large-margin approach.

4.1 Artificial Data

We generate artificial graphs with $|N| \in \{10, 20, 30, 40\}$ nodes as follows. For every node in a graph, we sample the number of outgoing edges uniformly in the interval $\lfloor \frac{|N|}{2}, |N| \rfloor$, and for every edge, a receiving node is sampled uniformly from the remaining nodes. To annotate the optimal path we first draw its length uniformly in the interval $\lfloor \frac{|N|}{2}, |N| \rfloor$ and randomly select the respective number of nodes from $N$, while enforcing that every edge in the path is included in the graph as well.

To ensure that the optimal path is actually the one with lowest costs, edge features are sampled from a one-dimensional Gaussian mixture distribution, where the generating component is chosen according to whether the respective edge lies on the the shortest path or not. That is, we introduce two Gaussian components $G_{1,2}$, so that costs for edges lying on the shortest path are drawn from $G_1(\mu_1, \sigma_1^2)$ while costs for all other edges are sampled from $G_2(\mu_2, \sigma_2^2)$.

The difficulty of the experimental setup is controlled by a parameter $\alpha$ that measures the distance of the two means, i.e., $\alpha = |\mu_1 - \mu_2|$. We sample the means from the following normal distributions $\mu_1 \sim G(-\frac{3}{2}, 0.1)$ and $\mu_2 \sim G(\frac{3}{2}, 0.1)$. We use $\sigma_1 = \sigma_2 = 0.01$ and report on averages over 100 repetitions. The results for perceptrons and SVMs are shown in Figure 2. The distance $\alpha$ is depicted on the $x$-axis. The $y$-axis shows the top-one accuracy. The performance of both algorithms highly depends on the distance of the cost-generating components and the size of the graph. Both algorithms perform similarly.

4.2 News Headlines

The real-world data originates from titles of crawled news articles from several web sites on different days. We use categories Technology, Sports, Business and General. Related sets with more than 4 news headlines are manually identified and grouped together, and word graphs are built according to the procedure described in Section 2.2. Ground truth is annotated manually by selecting the best sentence among the 20 shortest paths computed by Yen’s algorithm [Yen, 1971] using frequencies as edge weights. This process leaves us with 87 training examples.

We intend to learn the costs for the edges that give rise to the optimal compression of the training graphs and compare our algorithms to the unsupervised approach presented in [Filippova, 2010] that uses $(\#(x_1) + \#(x_2))/\#(x_1, x_2)$ as edge weights. We devise two different sets of features. The first feature representation consists of only two features that are inspired by the heuristic. That is, for an edge $(x_1, x_2)$, we use

$$\phi_1(x_1, x_2) = \left( \frac{\#(x_1)}{\#(x_1, x_2)}, \frac{\#(x_2)}{\#(x_1, x_2)} \right)^T,$$

where $\#$ denotes the frequency of nodes and edges, respectively. The second feature representation which is again inspired by [Filippova, 2010] and uses the ingredients of the heuristic instead of precomputing the surrogates to have the

![Figure 2: Performance on artificial data for perceptrons (top) and SVMs (bottom).](image-url)
algorithm pick the best combination, 
\[
\phi_2(x_1, x_j) = (\#(x_1), \#(x_2), \#(x_1, x_2), \\
\log(\#(x_1)), \log(\#(x_2)), \log(\#(x_1, x_2)))^T.
\]

Figure 3 shows average accuracy (top) and average rank (bottom) for perceptrons and SVMs, respectively, for different training set sizes, depicted on the x-axis. Every curve is the result of a cross-validation that uses all available data. Thus, the rightmost points are generated by a 2-fold cross validation while the leftmost points result from using 11-folds. Due to the small training sets, interpreting the figures is difficult. The unsupervised baseline outperforms the learning methods although there are indications that more training data could lead to better performances of perceptrons and SVMs. The first feature representation shows better performances than the second. However, these conjectures need to be verified by an experiment on a larger scale.

Using only the second feature representation, Table 1 shows average accuracies and average ranks for a leave-one-out setup to increase the sizes of the training sets. The results are promising and not too far from the baseline, however, as before, the evaluation needs to be based on larger sample sizes to allow for interpretations.

5 Conclusion

In this paper, we proposed to learn shortest paths in compression graphs for summarising related sentences. We addressed the previously unsupervised problem in a supervised context and devised structural support vector ma-

chines that effectively learn the edge weights of compression graphs, so that a shortest path algorithm decodes the best possible summarisation. We showed that the most strongly violated constrains can be computed directly by loss-augmented inference and rendered the use of two-best algorithms unnecessary. Empirically, we presented preliminary results on artificial and real world data sets. Due to small sample sizes, conclusions cannot be drawn yet, although the results indicate that learning shortest paths could be an alternative to heuristic and unsupervised approaches. Future work will address this question in greater detail.

References


