Query-focused Sentence Compression in Linear Time

Abram Handler and **Brendan O'Connor**
College of Information and Computer Sciences
University of Massachusetts Amherst
{ahandler,brenocon}@cs.umass.edu

Abstract

Search applications often display shortened sentences which must contain certain query terms and must fit within the space constraints of a user interface. This work introduces a new transition-based sentence compression technique developed for such settings. Our query-focused method constructs length and lexically constrained compressions in linear time, by growing a subgraph in the dependency parse of a sentence. This theoretically efficient approach achieves an 11x empirical speedup over baseline ILP methods, while better reconstructing gold constrained shortenings. Such speedups help query-focused applications, because users are measurably hindered by interface lags. Additionally, our technique does not require an ILP solver or a GPU.

1 Introduction

Traditional study of extractive sentence compression seeks to create short, readable, single-sentence summaries which retain the most "important" information from source sentences. But search user interfaces often require compressions which must include a user's query terms and must not exceed some maximum length, permitted by screen space. Figure 1 shows an example.

This study examines the English-language compression problem with such length and lexical requirements. In our constrained compression setting, a source sentence S is shortened to a compression C which (1) must include all tokens in a set of query terms Q and (2) must be no longer than a maximum budgeted character length, $b \in \mathbb{Z}^+$. Formally, constrained compression maps $(S, Q, b) \rightarrow C$, such that C respects Q and b. We describe this task as query-focused compression because Q places a hard requirement on words from S which must be included in C.

Figure 1: A search user interface (boxed, top) returns a snippet consisting of three compressions which must contain a users' query Q (bold) and must not exceed $b = 75$ characters in length. The third compression C was derived from source sentence S (italics, bottom).

Existing techniques are poorly suited to constrained compression. While methods based on integer linear programming (ILP) can trivially accommodate such length and lexical restrictions (Clarke and Lapata, 2008; Filippova and Altun, 2013; Wang et al., 2017), these approaches rely on slow third-party solvers to optimize an NP-hard integer linear programming objective, causing user wait time. An alternative LSTM tagging approach (Filippova et al., 2015) does not allow practitioners to specify length or lexical constraints, and requires an expensive graphics processing unit (GPU) to achieve low runtime latency (access to GPUs is a barrier in fields like social science and journalism). These deficits prevent application of existing compression techniques in search user interfaces (Marchionini, 2006; Hearst, 2009), where length, lexical and latency requirements are paramount. We thus present a new stateful method for query-focused compression.

Our approach is theoretically and empirically faster than ILP-based techniques, and more accurately reconstructs gold standard compressions.

2 Related work

Extractive compression shortens a sentence by removing tokens, typically for summarization

Approach	Complexity	Constrained
ILP	exponential	yes
LSTM tagger	linear	no
VERTEX ADDITION	**linear**	**yes**

Table 1: Our VERTEX ADDITION technique (§3) constructs constrained compressions in linear time. Prior work (§2) has higher computational complexity (ILP) or does not respect hard constraints (LSTM tagger).

(Knight and Marcu, 2000; Clarke and Lapata, 2008; Filippova et al., 2015; Wang et al., 2017).[1] To our knowledge, this work is the first to consider extractive compression under hard length and lexical constraints.

We compare our VERTEX ADDITION approach to ILP-based compression methods (Clarke and Lapata, 2008; Filippova and Altun, 2013; Wang et al., 2017), which shorten sentences using an integer linear programming objective. ILP methods can easily accommodate lexical and budget restrictions via additional optimization constraints, but require worst-case exponential computation.[2]

Finally, compression methods based on LSTM taggers (Filippova et al., 2015) cannot currently enforce lexical or length requirements. Future work might address this limitation by applying and modifying constrained generation techniques (Kikuchi et al., 2016; Post and Vilar, 2018; Gehrmann et al., 2018).

3 Compression via VERTEX ADDITION

We present a new transition-based method for shortening sentences under lexical and length constraints, inspired by similar approaches in transition-based parsing (Nivre, 2003). We describe our technique as VERTEX ADDITION because it constructs a shortening by *growing* a (possibly disconnected) subgraph in the dependency parse of a sentence, one vertex at a time. This approach can construct constrained compressions with a linear algorithm, leading to 11x lower latency than ILP techniques (§4). To our knowledge, our method is also the first to construct compressions by *adding* vertexes rather than *pruning* subtrees in a parse (Knight and Marcu, 2000; Almeida and Martins, 2013; Filippova and Alfonseca, 2015). We assume a boolean relevance model: S must contain Q. We leave more sophisticated relevance models for future work.

3.1 Formal description

VERTEX ADDITION builds a compression by maintaining a state (C_i, P_i) where $C_i \subseteq S$ is a set of added candidates, $P_i \subseteq S$ is a priority queue of vertexes, and i indexes a timestep during compression. Figure 2 shows a step-by-step example.

During initialization, we set $C_0 \leftarrow Q$ and $P_0 \leftarrow S \setminus Q$. Then, at each timestep, we pop some candidate $v_i = h(P_i)$ from the head of P_i and evaluate v_i for inclusion in C_i. (Neighbors of C_i in P_i get higher priority than non-neighbors; we break ties in left-to-right order, by sentence position). If we accept v_i, then $C_{i+1} \leftarrow C_i \cup v_i$; if not, $C_{i+1} \leftarrow C_i$. We discuss acceptance decisions in detail in §4.3. We continue adding vertexes to C until either P_i is empty or C_i is b characters long.[3] The appendix includes a formal algorithm.

VERTEX ADDITION is linear in the token length of S because we pop and evaluate some vertex from P_i at each timestep, after $P_0 \leftarrow S \setminus Q$. Additionally, because (1) we never accept v_i if the length of $C_i \cup v_i$ is more than b, and (2) we set $C_0 \leftarrow Q$, our method respects Q and b.

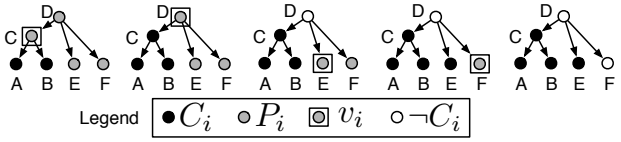

Figure 2: A dependency parse of a sentence S, shown across five timesteps of VERTEX ADDITION (from left to right). Each node in the parse is a vertex in S. Our stateful method produces the final compression {A,C,B,E} (rightmost). At each timestep, each candidate v_i is boxed; rejected candidates $\neg C_i$ are unshaded.

4 Evaluation

We observe the latency, readability and token-level F1 score of VERTEX ADDITION, using a standard dataset (Filippova and Altun, 2013). We compare our method to an ILP baseline (§2) because ILP methods are the only known technique for constrained compression. All methods have similar

[1] Some methods compress via generation instead of deletion (Rush et al., 2015; Mallinson et al., 2018). Our extractive method is intended for practical, interpretable and trustworthy search systems (Chuang et al., 2012). Users might not trust abstractive summaries (Zhang and Cranshaw, 2018), particularly in cases with semantic error.

[2] ILPs are exponential in $|V|$ when selecting tokens (Clarke and Lapata, 2008) and exponential in $|E|$ when selecting edges (Filippova et al., 2015).

[3] We linearize C by left-to-right vertex position in S, common for compression in English (Filippova and Altun, 2013).

compression ratios (shown in appendix), a well-known evaluation requirement (Napoles et al., 2011). We evaluate the significance of differences between VERTEX ADDITION$_{LR}$ and the ILP with bootstrap sampling (Berg-Kirkpatrick et al., 2012). All differences are significant ($p < .01$).

4.1 Constrained compression experiment

In order to evaluate different approaches to constrained compression, we require a dataset of sentences, constraints and known-good shortenings, which respect the constraints. This means we need tuples (S, Q, b, C_g), where C_g is a known-good compression of S which respects Q and b (§1).

To support large-scale automatic evaluation, we reinterpret a standard compression corpus (Filippova and Altun, 2013) as a collection of input sentences and constrained compressions. The original dataset contains pairs of sentences S and compressions C_g, generated using news headlines. For our experiment, we set b equal to the character length of the gold compression C_g. We then sample a small number of nouns[4] from C_g to form a query set Q, approximating both the observed number of tokens and observed parts of speech in real-world search (Jansen et al., 2000; Barr et al., 2008). Sampled Q include reasonable queries like "police, Syracuse", "NHS" and "Hughes, manager, QPR".

By sampling queries and defining budgets in this manner, we create 198,570 training tuples and 9949 test tuples, each of the form (S, Q, b, C_g). Filippova and Altun (2013) define the train/test split. We re-tokenize, parse and tag with CoreNLP v3.8.0 (Manning et al., 2014). We reserve 25,000 training tuples as a validation set.

4.2 Model: ILP

We compare our system to a baseline ILP method, presented in Filippova and Altun (2013). This approach represents each edge in a syntax tree with a vector of real-valued features, then learns feature weights using a structured perceptron trained on a corpus of (S, C_g) pairs.[5] Learned weights are used to compute a global compression objective, subject to structural constraints which ensure C is a valid tree. This baseline can easily perform constrained compression: at test time, we add optimization constraints specifying that C must include Q, and not exceed length b.

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 0.76 on the unconstrained compression task, lower than the result (F1 = 84.3) reported by the original authors. There are some important differences between our reimplementation and original approach (described in detail in the appendix). Since VERTEX ADDITION requires Q and b, we can only compare it to the ILP on the *constrained* (rather than traditional, unconstrained) compression task.

4.3 Models: VERTEX ADDITION

Vertex addition accepts or rejects some candidate vertex v_i at each timestep i. We learn such decisions $y_i \in \{0, 1\}$ using a corpus of tuples (S, Q, b, C_g) (§4.1). Given such a tuple, we can always execute an oracle path shortening S to C_g by first initializing VERTEX ADDITION and then, at each timestep: (1) choosing $v_i = h(P_i)$ and (2) adding v_i to C_i iff $v_i \in C_g$. We set $y_i = 1$ if $v_i \in C_g$; we set $y_i = 0$ if $v_i \notin C_g$. We then use decisions from oracle paths to train two models of inclusion decisions, $p(y_i = 1 | v_i, C_i, P_i, S)$. At test time, we accept v_i if $p(y_i > .5)$.

Model One. Our VERTEX ADDITION$_{NN}$ model broadly follows neural approaches to transition-based parsing (e.g. Chen and Manning (2014)): we predict y_i using a LSTM classifier with a standard max-pooling architecture (Conneau et al., 2017), implemented with a common neural framework (Gardner et al., 2017). Our classifier maintains four vocabulary embeddings matrixes, corresponding to the four disjoint subsets $C_i \cup \neg C_i \cup P_i \cup \{v_i\} = V$. Each LSTM input vector x_t comes from the appropriate embedding for $v_t \in V$, depending on the state of the compression system at timestep i. The appendix details network tuning and optimization.

Model Two. Our VERTEX ADDITION$_{LR}$ model uses binary logistic regression,[6] with 3 classes of features.

Edge features describe the properties of the edge (u, v_i) between $v_i \in P_i$ and $u \in C_i$. We

[4] 1 to 3 nouns; cardinality chosen uniformly at random.

[5] Another ILP (Wang et al., 2017) sets weights using a LSTM, achieving similar in-domain performance. This method requires a multi-stage computational process (i.e. run LSTM *then* ILP) that is poorly-suited to query-focused settings, where low latency is crucial.

[6] We implement with Python 3 using scikit-learn (Pedregosa et al., 2011). We tune the inverse regularization constant to $c = 10$ via grid search over powers of ten, to optimize validation set F1.

Approach	F1	SLOR	*Latency
RANDOM (lower bound)	0.653	0.377	0.5
ABLATED (edge only)	0.827	0.669	3.7
VERTEX ADDITION$_{NN}$	0.873	0.728	2929.1 (CPU)
ILP	0.852	0.756	44.0
VERTEX ADDITION$_{LR}$	0.881	0.745	4.1

Table 2: Test results for constrained compression. *Latency is the geometric mean of observed runtimes (in milliseconds per sentence). VERTEX ADDITION$_{LR}$ achieves the highest F1, and also runs 10.73 times faster than the ILP. Differences between all scores for VERTEX ADDITION$_{LR}$ and ILP are significant ($p < .01$).

use the edge-based feature function from Filippova and Altun (2013), described in detail in the appendix. This allows us to compare the performance of a vertex addition method based on local decisions with an ILP method that optimizes a global objective (§4.5), using the same feature set.

Stateful features represent the relationship between v_i and the compression C_i at timestep i. Stateful features include information such as the position of v_i in the sentence, relative to the rightmost and left-most vertex in C_i, as well as history-based information such as the fraction of the character budget used so far. Such features allow the model to reason about which sort of v_i should be added, given Q, S and C_i.

Interaction features are formed by crossing all stateful features with the type of the dependency edge governing v_i, as well as with indicators identifying if u governs v_i, if v_i governs u or if there is no edge (u, v_i) in the parse.

4.4 Metrics: F1, Latency and SLOR

We measure the token-level F1 score of each compression method against gold compressions in the test set. F1 is the standard automatic evaluation metric for extractive compression (Filippova et al., 2015; Klerke et al., 2016; Wang et al., 2017).

In addition to measuring F1, researchers often evaluate compression systems with human *importance* and *readability* judgements (Knight and Marcu, 2000; Filippova et al., 2015). In our setting Q determines the "important" information from S, so importance evaluations are inappropriate. To check readability, we use the automated readability metric SLOR (Lau et al., 2015), which correlates with human judgements (Kann et al., 2018).

We evaluate theoretical gains from VERTEX ADDITION (Table 1) by measuring empirical latency. For each compression method, we sample and compress $N = 300,000$ sentences, and record the runtime (in milliseconds per sentence). We observe that runtimes are distributed log-normally (Figure 3), and we thus summarize each sample using the geometric mean. ILP and VERTEX ADDITION$_{LR}$ share edge feature extraction code to support to fair comparison. We test VERTEX ADDITION$_{NN}$ using a CPU: the method is too slow for use in search applications in areas without access to specialized hardware (Table 2). The appendix further details latency and SLOR experiments.

4.5 Analysis: ABLATED & RANDOM

For comparison, we implement an ABLATED vertex addition method, which learns inclusion decisions using only edge features from Filippova and Altun (2013). ABLATED has a lower F1 score than ILP, which uses the same edge-level information to optimize a global objective: adding stateful and interaction features (i.e. VERTEX ADDITION$_{LR}$) improves F1 score. Nonetheless, strong performance from ABLATED hints that edge-level information alone (e.g. dependency type) can mostly guide acceptance decisions.

We also evaluate a RANDOM baseline, which accepts each v_i randomly in proportion to $p(y_i = 1)$ across training data. RANDOM achieves reasonable F1 because (1) $C_0 = Q \in C_g$ and (2) F1 correlates with compression rate (Napoles et al., 2011), and b is set to the length of C_g.

5 Future work: practical compression

This work presents a new method for fast query-focused sentence compression, motivated by the need for query-biased snippets in search engines (Tombros and Sanderson, 1998; Marchionini, 2006). While our approach shows promise in simulated experiments, we expect that further work will be required before the method can be employed for practical, user-facing search.

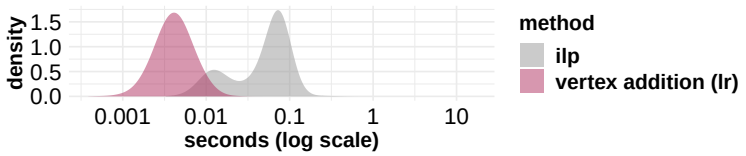

Figure 3: Density plot of log transformed latencies for VERTEX ADDITION$_{LR}$ (left) and ILP (right). Theoretical gains (Table 1) create real speedups. The ILP shows greater runtime variance, possibly reflecting varying approaches from Gurobi Optimization (2018).

To begin, both our technique and our evaluation ignore the conventions of search user interfaces, which typically display missing words using ellipses. This convention is important, because it allows snippet systems to transparently show users which words have been removed from a sentence. However, we observe that some well-formed compressions are difficult to read when displayed in this format. For instance the sentence "Aristide quickly fled Haiti in September 1991" can be shortened to the well-formed compression "Aristide fled in 1991." But this compression does not read fluidly when using ellipses ("Aristide...fled...in...1991"). Human experiments aimed at enumerating the desirable and undesirable properties of compressions displayed in ellipse format (e.g. compressions should minimize number of ellipses?) could help guide user-focused snippet algorithms in future work.

Our method also assumes access to a reliable, dependency parse, and ignores any latency penalties incurred from parsing. In practical settings, both assumptions are unreasonable. Like other NLP tools, dependency parsers often perform poorly on out of domain text (Bamman, 2017), and users looking to quickly investigate a new corpus might not wish to wait for a parser. Faster approaches based on low-latency part-of-speech tagging, or more cautious approaches based on syntactic uncertainty (Keith et al., 2018), each offer exciting possibilities for additional research.

Our approach also assumes that a user already knows a reasonable b and reasonable Q for a given sentence S.[7] However, in some cases, there is no well-formed shortening of which respects the requirements. For instance, if Q="Kennedy" and b=15 there is no reasonable shortening for the toy sentence "Kennedy kept running", because the compressions "Kennedy kept" and "Kennedy running" are not well-formed. We look forward to investigating which (Q, S, b) triples will never return well-formed compressions in later work.

Finally, some shortened sentences will modify the meaning of a sentence, but we ignore this important complication in this initial study. In the future, we hope to apply ongoing research into textual entailment (Bowman et al., 2015; Pavlick and Callison-Burch, 2016; McCoy and Linzen, 2018) to develop semantically-informed approaches to the task.

[7] Recall that we simulate b and Q based on the well-formed shortening C_g, see §4.1.

6 Conclusion

We introduce a query-focused VERTEX ADDITION$_{LR}$ method for search user interfaces, with much lower theoretical complexity (and empirical runtimes) than baseline techniques. In search applications, such gains are non-trivial: real users are measurably hindered by interface lags (Nielsen, 1993; Liu and Heer, 2014). We hope that our fast, query-focused method better enables snippet creation at the "pace of human thought" (Heer and Shneiderman, 2012).

7 Acknowledgments

Thanks to Javier Burroni and Nick Eubank for suggesting ways to optimize and measure performance of Python code. Thanks to Jeffrey Flanigan, Katie Keith and the UMass NLP reading group for helping us clarify our work.

References

Miguel Almeida and Andre Martins. 2013. Fast and robust compressive summarization with dual decomposition and multi-task learning. In *ACL*.

David Bamman. 2017. Natural language processing for the long tail. In *Digital Humanities*.

Cory Barr, Rosie Jones, and Moira Regelson. 2008. The linguistic structure of english web-search queries. In *EMNLP*.

Taylor Berg-Kirkpatrick, David Burkett, and Dan Klein. 2012. An empirical investigation of statistical significance in NLP. In *EMNLP*.

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

Samuel R. Bowman, Gabor Angeli, Christopher Potts, and Christopher D. Manning. 2015. A large annotated corpus for learning natural language inference. In *EMNLP*.

Ted Briscoe, John Carroll, and Rebecca Watson. 2006. The second release of the RASP system. In *Proceedings of the COLING/ACL 2006 Interactive Presentation Sessions*.

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

Jason Chuang, Daniel Ramage, Christopher D. Manning, and Jeffrey Heer. 2012. Interpretation and trust: Designing model-driven visualizations for text analysis. In *CHI*.

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*.

John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Katja Filippova and Enrique Alfonseca. 2015. Fast k-best sentence compression. *CoRR*, abs/1510.08418.

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*. https://github.com/google-research-datasets/sentence-compression.

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

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.

Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. 2018. Bottom-up abstractive summarization. In *EMNLP*.

LLC Gurobi Optimization. 2018. Gurobi optimizer reference manual (v8).

Kenneth Heafield. 2011. KenLM: faster and smaller language model queries. In *EMNLP: Sixth Workshop on Statistical Machine Translation*.

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

Jeffrey Heer and Ben Shneiderman. 2012. Interactive dynamics for visual analysis. *Queue*, 10(2).

Bernard J. Jansen, Amanda Spink, and Tefko Saracevic. 2000. Real life, real users, and real needs: a study and analysis of user queries on the web. *Information Processing and Management*, 36:207–227.

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

Katherine Keith, Su Lin Blodgett, and Brendan O'Connor. 2018. Monte Carlo syntax marginals for exploring and using dependency parses. In *NAACL*.

Yuta Kikuchi, Graham Neubig, Ryohei Sasano, Hiroya Takamura, and Manabu Okumura. 2016. Controlling output length in neural encoder-decoders. In *EMNLP*.

Sigrid Klerke, Yoav Goldberg, and Anders Søgaard. 2016. Improving sentence compression by learning to predict gaze. In *NAACL*.

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

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

Zhicheng Liu and Jeffrey Heer. 2014. The effects of interactive latency on exploratory visual analysis. *IEEE Transactions on Visualization and Computer Graphics*, 20:2122–2131.

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).

Marie-Catherine de Marneffe, Bill MacCartney, and Christopher D. Manning. 2006. Generating typed dependency parses from phrase structure parses. In *LREC*.

Thomas R. McCoy and Tal Linzen. 2018. Non-entailed subsequences as a challenge for natural language inference. *CoRR*. Version 1.

Courtney Napoles, Benjamin Van Durme, and Chris Callison-Burch. 2011. Evaluating sentence compression: Pitfalls and suggested remedies. In *Proceedings of the Workshop on Monolingual Text-To-Text Generation*.

Jakob Nielsen. 1993. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

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

Joakim Nivre, Marie-Catherine de Marneffe, Filip Ginter, Yoav Goldberg, Jan Hajic, Christopher D. Manning, Ryan T. McDonald, Slav Petrov, Sampo Pyysalo, Natalia Silveira, Reut Tsarfaty, and Daniel Zeman. 2016. Universal Dependencies v1: A multilingual treebank collection. In *LREC*.

Ellie Pavlick and Chris Callison-Burch. 2016. So-called non-subsective adjectives. In **SEM*.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830.

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*.

Sebastian Schuster and Christopher D. Manning. 2016. Enhanced english universal dependencies: An improved representation for natural language understanding tasks. In *LREC*.

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

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*.

Amy X. Zhang and Justin Cranshaw. 2018. Making sense of group chat through collaborative tagging and summarization. *Proceedings of the ACM on Human-Computer Interaction*, 2(CSCW).

A Appendix

A.1 Algorithm

We formally present the VERTEX ADDITION compression algorithm, using notation defined in §3.1. ℓ linearizes a vertex set, based on left-to-right position in S. $|P|$ indicates the number of tokens in the priority queue.

Algorithm 1: VERTEX ADDITION

input: $s = (V, E)$, $Q \subseteq V$, $b \in \mathbb{R}^+$
$C \leftarrow Q$; $P \leftarrow V \setminus Q$;
while $\ell(C) < b$ *and* $|P| > 0$ **do**
  $v \leftarrow \text{pop}(P)$;
  if $p(y = 1) > .5$ *and* $\ell(C \cup \{v\}) \leq b$ **then**
    $C \leftarrow C \cup \{v\}$
  end
end
return $\ell(C)$

A.2 Neural network tuning and optimization

We learn network parameters for VERTEX ADDITION$_{NN}$ by minimizing cross-entropy loss against oracle decisions y_i. We optimize with ADAGRAD (Duchi et al., 2011). We learn input embeddings after initializing randomly. The hyperparameters of our network and training procedure are: the learning rate, the dimensionality of input embeddings, the weight decay parameter, the batch size, and the hidden state size of the LSTM. We tune via random search (Bergstra and Bengio, 2012), selecting parameters which achieve highest accuracy in predicting oracle decisions for the validation set. We train for 15 epochs, and we use parameters from the best-performing epoch (by validation accuracy) at test time.

Learning rate	0.025
Embedding dim.	315
Weight decay	1.88×10^{-9}
Hidden dim.	158
Batch size	135

Table 3: Hyperparameters for VERTEX ADDITION$_{NN}$

A.3 Reimplementation of Filippova and Altun (2013)

In this work, we reimplement the method of Filippova and Altun (2013), who in turn implement a method partially described in Filippova and Strube (2008). There are inevitable discrepancies between our implementation and the methods described in these two prior papers.

1. Where the original authors train on only 100,000 sentences, we learn weights with the full training set to compare fairly with VERTEX ADDITION (each model trains on the full training set).

2. We use Gurobi Optimization (2018) (v8) to solve the liner program. Filippova and Strube (2008) report using LPsolve.[8]

3. We implement with the common Universal Dependencies (UD, v1) framework (Nivre et al., 2016). Prior work (Filippova and Strube, 2008) implements with older dependency formalisms (Briscoe et al., 2006; de Marneffe et al., 2006).

4. In Table 1 of their original paper, Filippova and Altun (2013) provide an overview of the syntactic, structural, semantic and lexical features in their model. We implement every feature described in the table. We do not implement features which are not described in the paper.

5. Filippova and Altun (2013) augment edge labels in the dependency parse of S as a preprocessing step. We reimplement this step using off-the-shelf augmented modifiers and augmented conjuncts available with the enhanced dependencies representation in CoreNLP (Schuster and Manning, 2016).

6. Filippova and Altun (2013) preprocess dependency parses by adding an edge between the root node and all verbs in a sentence.[9] We found that replicating this transform literally (i.e. only adding edges from the original root to all tokens tagged as verbs) made it impossible for the ILP to recreate some gold compressions. (We suspect that this is due to differences in output from part-of-speech taggers). We thus add an edge between the root node and *all* tokens in a sentence during preprocessing, allowing the ILP to always return the gold compression.

[8] http://sourceforge.net/projects/lpsolve

[9] This step ensures that subclauses can be removed from parse trees, and then merged together to create a compression from different clauses of a sentence.

We assess convergence of the ILP by examining validation F1 score on the traditional sentence compression task. We terminate training after six epochs, when F1 score stabilizes (i.e. changes by fewer than 10^{-3} points).

A.4 Implementation of SLOR

We use the SLOR function to measure the readability of the shortened sentences produced by each compression system. 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.[10]

Following (Lau et al., 2015), we define the function as

$$\text{SLOR} = \frac{\log P_m(\xi) - \log P_u(\xi)}{|\xi|} \qquad (1)$$

where ξ is a sequence of words, $P_u(\xi)$ is the unigram probability of this sequence of words and $P_m(\xi)$ is the probability of the sequence, assigned by a language model. $|\xi|$ is the length (in tokens) of the sentence.

We use a 3-gram language model trained on the training set of the Filippova and Altun (2013) corpus. We implement with KenLM (Heafield, 2011). Because compression often results in shortenings where the first token is not capitalized (e.g. a compression which begins with the third token in S) we ignore case when calculating language model probabilities.

A.5 Latency evaluation

To measure latency, for each technique, we sample 100,000 sentences with replacement from the test set. We observe the mean time to compress each sentence using Python's built-in *timeit* module. In order to minimize effects from unanticipated confounds in measuring latency, we repeat this experiment three separate times (with a one hour delay between experiments). Thus in total we collect 300,000 observations for each compression technique. We observe that runtimes are log normal, and thus report each latency as the geometric mean of 300,000 observations. We use an Intel Xeon processor with a clock rate of 2.80GHz.

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

A.6 Compression ratios

When comparing sentence compression systems, it is important to ensure that all approaches use the same rate of compression (Napoles et al., 2011). Following Filippova et al. (2015), we define the compression ratio as the character length of the compression divided by the character length of the sentence. We present test set compression ratios for all methods in Table 4. Because ratios are similar, our comparison is appropriate.

RANDOM	0.405
ILP	0.408
ABLATED	0.387
VERTEX ADDITION$_{LR}$	0.403
VERTEX ADDITION$_{NN}$	0.405
C_g Train	0.384
C_g Test	0.413

Table 4: Mean test time compression ratios for all techniques. We also show mean ratios for gold compressions C_g across the train and test sets.

.