Generating Natural Anagrams: Towards Language Generation Under Hard Combinatorial Constraints

Masaaki Nishino[1] **Sho Takase**[2*] **Tsutomu Hirao**[1] **Masaaki Nagata**[1]
[1] NTT Communication Science Laboratories, NTT Corporation
{masaaki.nishino.uh,tsutomu.hirao.kp,masaaki.nagata.et}@hco.ntt.co.jp
[2] Tokyo Institute of Technology
sho.takase@nlp.c.titech.ac.jp

Abstract

An anagram is a sentence or a phrase that is made by permutating the characters of an input sentence or a phrase. For example, "Trims cash" is an anagram of "Christmas". Existing automatic anagram generation methods can find possible combinations of words form an anagram. However, they do not pay much attention to the naturalness of the generated anagrams. In this paper, we show that simple depth-first search can yield natural anagrams when it is combined with modern neural language models. Human evaluation results show that the proposed method can generate significantly more natural anagrams than baseline methods.

1 Introduction

Making an *anagram* is one of the most popular wordplay games and has a long history that can be traced back to the time of the Ancient Greeks (Wheatley, 1862). An anagram is a sentence or a phrase that is made as a permutation of characters of an input sentence or phrase. For example, "Trims cash" is an anagram of "Christmas" since the former can be made by permutating the characters contained in the latter.

The automatic generation of anagrams is a challenging task since it is NP-hard[1]. Existing automatic anagram generation methods mainly focus on search efficiency in finding word combinations that can form anagrams (Jordan and Monteiro, 2003). However, they place little consideration on the word orders that seem natural to humans. Fortunately, recent progress in NLP techniques now enables natural sentences to be generated in many NLP tasks (Józefowicz et al., 2016; Gehrmann et al., 2018; Skadina and Pinnis, 2017). However, beam search, the de-facto decoding algorithm used in many language generation methods, cannot be used as-is to generate anagrams since it cannot ensure that the generated sentences are anagrams. Although beam search variants that can generate sentences that satisfy some constraints have been proposed (Anderson et al., 2017; Hokamp and Liu, 2017), applying them to anagram generation tasks soon becomes intractable. To summarize, generating natural anagrams is a challenging task that cannot be solved by simply applying existing methods.

In this paper, we propose an anagram generation algorithm. The proposed algorithm is very simple; we run depth-first search to enumerate anagrams. As to the generation procedure, we use a neural language model and character frequency to suppress unnecessary search operations. Due to the power of the currently available pre-trained neural language models, this simple approach works surprisingly well. In experiments that employ human evaluations we confirm that the proposed natural anagram generation method can generate more natural anagrams than a baseline method.

2 Related Work

Although anagram generation is NP-hard, the basic algorithm that can output a set of words as anagrams is relatively simple (Jordan and Monteiro, 2003). As a result many anagram generation software and web services exist including the Internet Anagram Server[2] and the Anagram Artist[3]. Unfortunately, these methods pay scant attention to the naturalness of the generated anagrams. In Wikipedia (Wikipedia, 2019), it is said that the

This work was conducted while the author was at NTT Communication Science Laboratories.

[1] NP-hardness of anagram generation can be proved by using the fact that it contains an exact cover problem, which has been proven to be NP-complete (Karp, 1972), as a special case.

[2] https://wordsmith.org/anagram/

[3] https://www.anagrammy.com/resources/anagram_artist.html

Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, pages 6408–6412, Hong Kong, China, November 3–7, 2019.

automatic anagram generators work poorly when challenged with multi-word anagrams since they cannot take word meanings into account.

The anagram generation task can be seen as a variant of the language generation problem under some constraints. Anderson et al. (2017) proposes a constrained beam search algorithm for image captioning. Their method can generate sentences containing pre-specified words. Their method first creates a finite-state automaton (FSA) representing the constraints. Then it runs beam search over the FSA to obtain results that satisfy the constraints. It is possible to apply their method to anagram generation. However, since the number of FSA states grows exponentially with input length, it rapidly becomes intractable. Hokamp and Liu (2017) proposes the constrained decoding method called grid beam search for machine translation. It is also impractical if applied to the anagram generation task.

3 Preliminaries

We represent sentence s as a sequence of words, $s = (w_1, w_2, \ldots, w_{|s|})$, where $|s|$ is the number of words contained in s. Every word w_i is selected from vocabulary V, which consists of $|V|$ distinct words. We use $C(s)$ to represent the multiset of characters contained in s. For example, if $s =$ "Christmas", then $C(s) = \{\mathrm{a}, \mathrm{c}, \mathrm{h}, \mathrm{i}, \mathrm{m}, \mathrm{r}, \mathrm{s}, \mathrm{s}, \mathrm{t}\}$. In the following, we ignore case and we assume that every word $w \in V$ consists of the characters from a to z. We say sentence t is an *anagram* of sentence s if $C(s) = C(t)$. For example, if $t =$ "Trims cash", then $C(t) = \{\mathrm{a}, \mathrm{c}, \mathrm{h}, \mathrm{i}, \mathrm{m}, \mathrm{r}, \mathrm{s}, \mathrm{s}, \mathrm{t}\}$, which is equivalent to $C(s)$ for $s =$ "christmas". The *anagram generation problem* is the problem of finding sentence t given input sentence s, where t is an anagram of s.

We use a language model when generating anagrams. Let $p(s)$ be the probability that sentence $s = (w_1, w_2, \ldots, w_{|s|})$ appears. A language model is a stochastic model that gives conditional probability $p(w_{i+1}|w_1, \ldots, w_i)$. A language model derives probability $p(s)$ by

$$p(s) = p(w_1) \prod_{i=1}^{|s|-1} p(w_{i+1}|w_1, \ldots, w_i). \quad (1)$$

While our anagram generation algorithm can use any language model, our experiments use the neural language model of ELMo (Peters et al., 2018).

4 Anagram Generation

Our anagram generation algorithm uses simple DFS. Given input string s, we start the search with the initial string $t =$"$\langle \mathtt{S} \rangle$" and then try to append a word $w \in V$ to the tail of t for each step to search for anagrams of s. If we find that t yields no solution by appending any words to it, we pop back the last word from t and continue the search by appending a different word to the tail of t. Since there are $|V|^N$ possible sentences consisting of N words, naive DFS is intractable. Thus we need to reduce the search steps. For this we introduce the following two criteria.

The first criteria checks frequency of each character appearing in t. Partial sentence t cannot be an anagram of s by appending any words to it once the condition $C(t) \setminus C(s) \neq \emptyset$ is satisfied. For example, partial sentence $t =$"$\langle \mathtt{S} \rangle$ The" cannot form an anagram of $s =$"Christmas" since $C(t) \setminus C(s) = \{\mathrm{e}\} \neq \emptyset$.

The second criterion exploits sentence probability. Since a language model can evaluate how natural a sentence is by calculating sentence occurrence probability, we threshold this probability and remove candidates with insufficient sentence probability. We use score function $f(w)$ defined as

$$f(w) = p(w \mid t) \cdot |w|^2 \quad (2)$$

as the score yielded by adding word w to sentence t, where $|w|$ is the number of characters of w. We select word w if it makes $f(w)$ larger than threshold value η. We use the number of characters of word w in the score function since shorter words tend to have high conditional probability $p(w \mid t)$ but using them would often read to unnatural sentences that consist of only very short ($|w| \leq 2$) words.

Branch and Bound for top-K search If we need only a few solutions, then we can further reduce the number of search steps by employing a simple branch and bound technique. Here we assume that we need K anagrams with high sentence probability $p(t)$. We use the fact that the sentence probability $p(t)$ defined in (1) is monotone decreasing when we append new word w to the sentence tail. Therefore, once we have found K anagrams and their probabilities are larger than θ, we can omit unnecessary subsequent search steps if the current partial sentence t satisfies $\theta > p(t)$.

We show the final natural anagram generation

Algorithm 1 Anagram generation algorithm

Input: Input sentence s
Output: At most K anagrams of s

```
1: t ←"⟨S⟩"
2: A ← a set of K dummy solutions
3: θ ← 0.0
4: SEARCH(t, A)
5: Output anagrams in A.
6:
7: procedure SEARCH(t, A)
8:     if t is an anagram of s then
9:         if p(t) > min_{a∈A} p(a) then
10:            Replace a_min = argmin_{a∈A} p(a) with t.
11:            θ ← min_{a∈A} p(a)
12:    else
13:        for all w ∈ V do
14:            if f(w) > η and
15:        C(t + w) \ C(s) = ∅ and p(t + w) > θ then
16:                SEARCH(t + w)
```

Score	Description
5	Natural as a written language.
4	Natural as a spoken language.
3	A few grammatical errors.
2	Meaning is guessable.
1	Meaning is unclear.

Table 1: Human evaluation criteria.

procedure in Algorithm 1. It first initializes sentence t and set of top-K anagrams A and then calls the recursive search procedure SEARCH. SEARCH first checks whether t is an anagram or not (line 8). If t is an anagram whose probability $p(t)$ is larger than $\min_{a \in A} p(a)$, then replace anagram $a^* = \text{argmin}_{a \in A} p(a)$ with t (line 10). If t is not an anagram, we proceed with the search by appending a word $w \in V$ to the tail of t. Here we use $t+w$ to represent the sentence obtained by appending w to the tail of t. We check that appending word w satisfies the conditions shown above (lines 14, 15), then we recursively call SEARCH$(t + w)$.

Implementation Details We tried to use the data structure called *dancing links* (Knuth, 2000) for managing the set of words satisfying the character frequency constraints. Dancing links is a link-based data structure that supports efficient DFS for some combinatorial search problems (Details are shown in (Knuth, 2019)). However, we found that using a simple array-based representation is sufficient since the conditional probability evaluation consumes most of the time.

5 Experiments

We conduct human evaluations to assess the performance of the proposed method. For inputs, we randomly select 100 titles of English Wikipedia articles whose length lies between 15 to 20 characters. We use the language model included in the TensorFlow (Abadi et al., 2015) implementation of ELMo[4], which is a CNN-LSTM language model trained with a 1 billion word benchmark corpus and that consists of approximately 800 million tokens[5]. It is also possible to combine the proposed method with conventional n-gram language models. However, since the superiority of neural language models over n-gram models has been reported in previous works (e.g., (Du et al., 2016; Józefowicz et al., 2016)), we conducted experiments using only ELMo. We use parameter $\eta = 0.005$ and we conduct top-K search with $K = 5$. We terminate the search after 20000 steps, which takes about 40 minutes.

Method	Average Score	
Proposed	4.11	(0.99)
Baseline	2.18	(1.17)

Table 2: Human evaluation results. The values in parentheses are standard deviations.

The baseline method used Anagram Artist, a freely available anagram generation software package. Anagram Artist can generate anagrams consisting of at most 5 words given inputs with lengths less than 25 characters. Although the algorithm used in anagram artist is not explained, it seems to use dictionary matching combining with some heuristics for deciding word order. We also tried an Internet anagram server. However, the quality of the anagrams generated by those two baseline methods are not much different. We compute the perplexities of the outputs of the baseline method and use the values to select the top-K anagrams. We run all experiments on computers with GeForce GTX 1080 Ti GPUs. The proposed method was mainly implemented in Python and the module for checking character frequency constraints was implemented in C++. We asked three native speakers to rate each anagram with scores ranging from 1 to 5 following the criteria shown in Table 1.

Table 2 shows the human evaluation results, where we compute the average score by first com-

[4] https://github.com/allenai/bilm-tf

[5] We also considered the use of pre-trained BERT (Devlin et al., 2018) since it is trained with a larger dataset. However, the pre-trained model does not contain weights of output layers and we cannot use it as-is for our task.

Inputs	Proposed outputs		Baseline outputs	
Economic experiment	Expect more in income	5.0	Nonce epimer toxemic	1.0
	Mine economic expert	3.0	Peen commix neoteric	1.0
	One comic experiment	4.3	Entice opener commix	1.3
	Common experience it	2.7	Mine compeer exciton	1.7
	More except in income	3.7	Peen commix erection	1.0
What would Reagan do	Two had a large wound	3.3	Two waul dragonhead	1.0
	We had a long draw out	4.7	What wauled gadroon	1.0
	What would a range do	3.3	What godown radulae	1.0
	We laugh and draw too	4.7	Thaw godown radulae	1.0
	We had no law to guard	5.0	Wood gulden waratah	1.0
Ashgate publishers	Belarus gas the ship	2.0	Litharge subphases	1.3
	Higher sales at pubs	5.0	Is the alpha burgess	2.0
	This has a superb leg	4.7	This huge sparables	1.3
	The bags are plus his	2.7	A lighter subphases	3.3
	She has a subtle grip	5.0	The agas publishers	2.7

Table 3: Example anagrams generated by the proposed method and baseline methods.

Anagrams	Proposed outputs
William Shakespeare ↓ I am a weakish speller	Phillies was a remake
	We all make his praise
	Speaker William Shea
	Hawaii keeps smaller
	William Ashe speaker
Tom Marvolo Riddle ↓ I am Lord Voldemort	I told Mr Lord a move
	Roll met David Romo
	Mr Ali drove to Mold
	Dr Millar moved too
	Mr Orlov told media

Table 4: Results of generating famous anagrams.

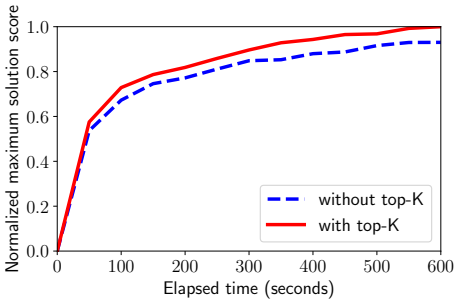

Figure 1: Running time comparison of the proposed method when applying the branch and bound method for retaining top-K solutions.

puting the average score of the three evaluators for each of the top-K outputs, then selecting the maximum of the K scores as the score of the method. The average shown in the table is the average of such maximum scores. The proposed method failed to generate any anagram for 3 instances. We therefore set the score of the proposed method to 1.0 for them. We can see that the proposed method yielded significantly higher human evaluation scores. Table 3 shows example anagrams generated by the proposed method and the baseline method. While the baseline method seems to employ some heuristics for deciding the word order, the combinations of selected words tend to form meaningless sentences. On the other hand, the proposed method tends to generate readable sentences. The anagrams shown in the table are listed in decreasing order of sentence probability. We can find that sentences with high probability are not always highly rated by the human evaluations. However, the normalized discounted cumulative gain (NDCG) score for the proposed method was 0.86. It suggests that using sentence probability works well for finding natural anagrams.

To evaluate the effectiveness of applying the branch and bound technique, we select problem instances that took longer than 600 seconds to complete and compared the maximum sentence probabilities of solutions found at a given time when top-K search is used. Figure 1 shows the comparison results, where the vertical axis plots the maximum scores of found anagrams normalized by the maximum one found within 600 seconds. This figure suggests using branch and bound accelerates the output of high score solutions. For example, with top-K, solutions with 0.8 maximum solution scores are found after 150 seconds, while it takes about 250 seconds without top-K search.

We also evaluated whether some famous anagrams appearing in Wikipedia (Wikipedia, 2019) can be generated with the proposed method. Table 4 shows the results of trying to generate fa-

mous anagrams. Unfortunately, the proposed method fails to find the famous anagram for these inputs. This is due to the fact that the famous anagrams use relatively rare words. However, the generated anagrams contain natural sentences.

6 Conclusion

We proposed an algorithm for generating natural anagrams. Surprisingly, a simple depth-first search approach is shown to be sufficient for generating natural anagrams when it is combined with modern neural language models. Since our current search procedure is relatively naive, there is room for improvement. More efficient use of GPUs in performing the DFS procedure is a promising future work.

Acknowledgements

We thank the anonymous reviewers for their insightful comments.

References

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Peter Anderson, Basura Fernando, Mark Johnson, and Stephen Gould. 2017. Guided open vocabulary image captioning with constrained beam search. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 936–945, Copenhagen, Denmark. Association for Computational Linguistics.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Jun Du, Yan-Hui Tu, Lei Sun, Feng Ma, Hai-Kun Wang, Jia Pan, Cong Liu, Jing-Dong Chen, and Chin-Hui Lee. 2016. The ustc-iflytek system for chime-4 challenge. *Proc. CHiME*, pages 36–38.

Sebastian Gehrmann, Yuntian Deng, and Alexander Rush. 2018. Bottom-up abstractive summarization. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 4098–4109, Brussels, Belgium. Association for Computational Linguistics.

Chris Hokamp and Qun Liu. 2017. Lexically constrained decoding for sequence generation using grid beam search. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1535–1546, Vancouver, Canada. Association for Computational Linguistics.

Timothy R. Jordan and Axel Monteiro. 2003. Generating anagrams from multiple core strings employing user-defined vocabularies and orthographic parameters. *Behavior Research Methods, Instruments, & Computers*.

Rafal Józefowicz, Oriol Vinyals, Mike Schuster, Noam Shazeer, and Yonghui Wu. 2016. Exploring the limits of language modeling. *CoRR*, abs/1602.02410.

Richard M Karp. 1972. Reducibility among combinatorial problems. In *Complexity of computer computations*, pages 85–103. Springer.

Donald E Knuth. 2000. Dancing links. In *Millenial Perspectives in Computer Science*, pages 187–214.

Donald E Knuth. 2019. *The Art of Computer Programming, Volume 4 Pre-fascicle 5c, Dancing Links*. Addison-Wesley.

Matthew Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

Inguna Skadina and Mārcis Pinnis. 2017. NMT or SMT: Case study of a narrow-domain English-Latvian post-editing project. In *Proceedings of the Eighth International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 373–383, Taipei, Taiwan. Asian Federation of Natural Language Processing.

Henry Benjamin Wheatley. 1862. Of anagrams: A monograph treating of their history from the earliest ages to the present time.

Wikipedia. 2019. Anagram — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Anagram. [Online; accessed 21-May-2019].