Exploring Chunk Based Templates for Generating a subset of English Text

Nikhilesh Bhatnagar
LTRC, IIIT Hyderabad

Manish Shrivastava
LTRC, IIIT Hyderabad

Radhika Mamidi
LTRC, IIIT Hyderabad

Abstract

Natural Language Generation (NLG) is a research task which addresses the automatic generation of natural language text representative of an input non-linguistic collection of knowledge. In this paper, we address the task of the generation of grammatical sentences in an isolated context given a partial bag-of-words which the generated sentence must contain. We view the task as a search problem (a problem of choice) involving combinations of smaller chunk based templates extracted from a training corpus to construct a complete sentence. To achieve that, we propose a fitness function which we use in conjunction with an evolutionary algorithm as the search procedure to arrive at a potentially grammatical sentence (modeled by the fitness score) which satisfies the input constraints.

1 Introduction

One of the reasons why NLG is a challenging problem is because there are many ways in which a given content can be represented. These are represented by the stylistic constraints which address syntactic and pragmatic choices (largely) independent of the information conveyed.

Classically, there are two major subtasks recognized in NLG: Strategic Generation and Tactical Generation (Sentence Planning and Surface Realization)[1](Reiter and Dale, 2000). Strategic Generation - "what to say" deals with identifying the relevant information to present to the audience and Tactical Generation - "how to say" addresses the problems of linguistic representation of the input concepts. In this work, we address the problem of tactical generation, with a focus on the grammaticality of the generated sentences. We formulate our task as follows: to generate syntactically correct sentences given a set of constraints such as a bag-of-words, partial ordering, etc.

So, for example, given a bag of words such as "man", "plays", "football" and length constraints, a sentence like "The man plays football in October." would be acceptable.

Our approach involves a corpus derived formulation of template based generation. Templates are instances of canned text with a slot-filler structure ("gaps") which can be filled with the appropriate information thus realizing the sentence. Since they are a manual resource, it is rather expensive and hard to generalize over different types or domains of text.

Thus, it is desirable to be able to automatically extract templates from a corpus. Also, to increase the syntactic coverage, we use sub-sentence level (smaller) templates to generate a sentence.

2 Background and Related Work

Traditionally, template based systems are used in scenarios where the output text is structurally very well defined and/or requires very high quality text as output with little variance. This work is inspired from (Van Deemter et al., 2005) who point out that template based systems and "real" NLG systems are "Turing equivalent" meaning that at least in terms of expressiveness, there is no theoretical disparity between the two. (Rudnicky and Oh, 2002) use language models to generate text. In recent years, (Kondadadi et al., 2013) present a hybrid NLG system which generates text by ranking tagged clusters of templates. NaturalOWL (Galanis and Androutsopoulos, 2007) use templates for

[1]Because we follow a template based approach, there is some overlap between the Content Determination and Aggregation steps.

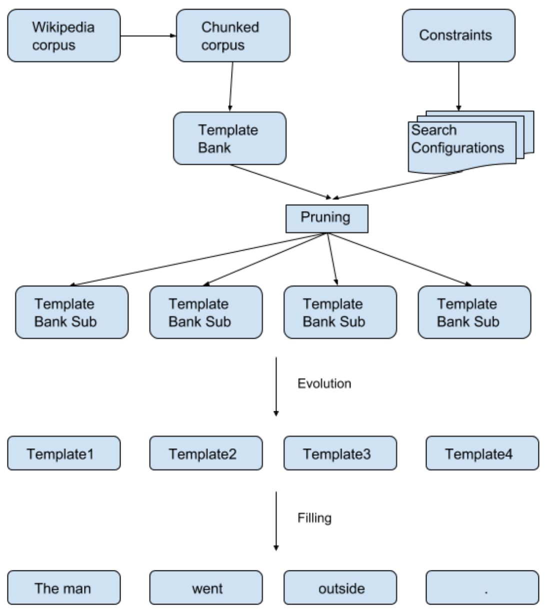

Figure 1: System architecture

their sentence plans and rules to manipulate them.

3 Core Assumptions and Motivation

As an extension of our previous work (Bhatnagar and Mamidi, 2016), in this work, we adopt a simplistic model of language - a sequence of linguistic units. Given a set of such units, each possible sentence is contained in the search space of all possible permutations. Thus, given a grammaticality fitness function, pruning and vocabulary reduction is essential to be able to tractably search this space.

Since templates are based on canned text, templates are locally grammatical. The core idea is to effectively use the local grammaticality guarantee of a corpus extracted template to combine, rather than construct the component templates to generate a sentence. These templates themselves contain individual tokens, effectively considering templates as a unit of sentence construction instead of tokens. The obvious trade-off is that since there are a lot more templates than tokens, which results in a much larger search space. However, if appropriate abstractions are used, perhaps the vocabulary problem can be mitigated somewhat.

The task is then twofold: how to determine which templates to combine (pruning) and constraining that with a measure of grammaticality of the generated sentence (fitness).

4 Templates and Sentence Representation

We use chunks as a basis for the linear templates because chunks are linguistically sound and hence the vocabulary increase is lesser compared to unconstrained spans of text. In general, an extended feature has syntactic identifiers added to the feature space. This is to better inform syntactic behavior of the template even though it increases the feature space a little. Abstracted features have multiple features clustered together which reduces the feature space. We describe the extension process below.

4.1 Abstraction

The chunks should be abstracted in such a way so as to have a minimal impact on their syntactic combination behavior.

All punctuation and stopwords are not abstracted because they are highly relevant, syntactically. Following are the mappings applied to the chunks:

1. Named Entity Abstraction (NE): Each NE is mapped to a unique symbol corresponding to its category.
2. POS Abstraction (POS): POS categories such as “CD”, “FW” and “SYM” and continuous “NNP” and “NNPS” sequences are mapped to their corresponding POS tags.
3. Cluster Abstraction (WC): Each token (except punctuation and stopwords), is mapped to the cluster ID of its syntactico-semantic cluster. Since a token can have multiple POS categories which in turn effect its syntactic behavior, we consider a token with different POS categories as distinct while clustering. The clusters are obtained by computing the KMeans cluster for token-POS pairs with euclidean distance of L2-normed vector embeddings.

4.2 Extended Categories

We extend the POS and chunk tag categories to better inform template combinations:

1. Extended POS (EPOS): Each punctuation, stopword and NE is assigned its own unique POS category.
2. Extended Chunk Tags (ECTag): Since the “O” (outside) chunk tag is a “default” category, it contributes a lot of syntactic confusion, we assign a separate chunk tag for all

Table 1: Template Example

Chunk Feature	Chunk	Template	Template Feature
Chunk Tag	"NP"	"NP"	Extended Chunk Tag (ECTag)
Tokens	"a", "popular", "wrestler"	"a", "JJ213", "NN5266"	Tok-POS cluster (WC)
POS	"DT", "JJ", "NN"	"DT", "JJ" ,"NN"	Extended POS (EPOS)
Head token	"wrestler"	"NN5266"	Head Tok-POS cluster (HWC)
Head POS	"NN"	"NN"	Head Extended POS (HEPOS)
		"a" and "NN5266"	Junction Tok-POS clusters (WCJ)
		"DT", "NN"	Junction Extended POS (EPOSJ)
		"a_BLANK_BLANK"	"Blank" Construction Feature (BlankCo)
		"a_JJ_NN"	Extended POS Construction Feature (EPOSCo)

chunks tagged "O" which contain sentence endings (".", ",", or "!") or brackets ("(" or ")"). In addition, chunk tags for chunks containing wh-words (POS tags "WDT", "WP", "WP$" or "WRB") are marked (eg. "NP" becomes "WNP").

4.3 Template features

A template is created after applying abstractions to a chunk and extending its syntactic categories. The template, however has two more feature types:

1. Junction Features (WCJ, EPOSJ): Junction features are comprised of the leftmost and rightmost features of the template. These features are used to predict if two templates "glue" together well at the point of contact (the junction). Both factors (tok-pos clusters and extended POS category) are applicable.
2. Head Features (HWC, HEPOS): Head features represent the "external" features for a chunk used to compute a global grammaticality component. Both factors - extended POS and tok-pos clusters are applicable.
3. Construction features (BlankCo, EPOSCo): Construction features represent the chunk as a syntactic construction or a layout, encoding the positional combination of the components comprising it. It is constructed by creating a feature for the ordered tuple of tokens comprising the chunk where all tokens are mapped to a single "blank" symbol or its extended POS except punctuations and stopwords. Two factors (single "blank" and extended POS) are applicable.

Table 1 shows an example chunk and its corresponding derived template. A sentence is represented as a sequence of templates described above.

5 Scoring the template combinations

It should be noted that this score is not equivalent to a syntactic correctness score, but rather a subset of it. This is because here we are dealing with configurations of untampered templates whose local syntactic correctness still holds and the syntactic incorrectness is a matter of their combinatorial configuration while a grammaticality score needs to deal with "broken" chunks as well.

The fitness score F is a linear combination of length-normalized total log-probabilities TP of different sequences derived from the sentence computed using an NGram language model. The total probability of an NGram model is defined as:

$$\begin{aligned} TP(s) = P(w_1|bos)P(w_2|w_1bos)... \\ ..P(eos|w_kw_{k-1}...w_{k-n+1}) \end{aligned} \tag{1}$$

where n is the order of the language model, s is a sequence and w_i is the ith element in s. F is a weighted sum of length-normalized total log-probabilities of five different sequences:

$$nTP(s) = log_{10}(TP(s))/(l_s + 1) \tag{2}$$

$$\begin{aligned} F(s) = \alpha_c nTP(ec) + \alpha_{co} nTP(co)/2 \\ + \alpha_j nTP(j)/2 + \alpha_h nTP(h)/2 \\ + \alpha_l nTP(l)/2 \end{aligned} \tag{3}$$

where $nTP(s)$ is the length normalized total log-probability for the sequence s where l_s is the number of elements in that sequence. ec, co, j, h and l represent the sequences in chunk score, construction score, junction score, head score and lexical score. α_i are the weight for each component score which are discussed:

1. Extended Chunk Score $nTP(ec)$ - This score is calculated using the extended chunk tags (ECTags) for the sentence. It is useful in determining the global syntactic structure.

Table 2: A sentence and sequences used to compute the fitness

Template Feature Sequence	Sentence
	NP[Sand] VP[blows] PP[in] NP[the strong wind] .[.]
	NP[NN969/NN] VP[VBZ29/VBZ] PP[in/in] NP[the/the JJ347/JJ NN628/NN] .[./.]
Extended Chunk Tags (ECTags)	NP VP PP NP .
Blank Construction (BlankCo)	BLANK BLANK in the_BLANK_BLANK .
Extended POS Construction (EPOSCo)	NN VBZ in the_JJ_NN .
Tok-POS Junction (WCJ)	(bos, NN969), (NN969, VBZ29), (VBZ29, in), (in, the), (NN628, .), (., eos)
Extended POS Junction (EPJ)	(bos, NN), (NN, VBZ), (VBZ, in), (in, the), (NN, .), (., eos)
Head Tok-POS (HWC)	NN969 VBZ29 in NN628 .
Head Extended POS (HEPOS)	NN VBZ in NN .
Tok-POS Sequence (WCs)	NN969 VBZ29 in the JJ347 NN628 .
Extended POS Sequence (EPOSs)	NN VBZ in the JJ NN .

2. Construction Score $nTP(co)$ - This score is calculated using blank construction layout (BlackCo) and extended POS construction layout (EPOSCo). It is useful in augmenting the overall grammaticality of the sentence as it encodes the syntactic layout of the chunk as a whole.
3. Junction Score $JP(j)$ - This score is calculated as the sum of the bigram probability of the left junction of the right template conditioned on the right junction of the left template for both tok-pos clusters (WC) and extended POS (EPOS) for all junctions in the sentence. This score represents a local view of inter-template cohesiveness. It represents how a sentence “glues” together.
4. Head Score $nTP(h)$ - This score is calculated using head tok-pos clusters (HWC) and head extended POS (HEPOS). Counter to the junction score, it represents a more semantic view of the chunk interactions. This is because generally a chunk head is often the primary content word in that chunk.
5. Lexical Score $nTP(l)$ - This score is computed using the tok-pos clusters (WC) and extended POS (EPOS). This score represents a baseline grammaticality score on a lexical level.

A sentence with all the sequences are fed to the fitness function are shown in table 2.

6 Parameters for pruning the search space

The search space is the space of all permutations of the templates to form the sentence. Such a search space is huge. We observe that depending on the particular constraints described in section 7, we can prune the permutation search space. This is done by finding a subset of the template bank that can occur at each position in the template sequence. The subsets are represented by a set of constraints which we call a search configuration. Thus, say, if a sentence is determined to have 5 templates, there will be a search configuration computed for each template slot, in accordance with the global constraints for the sentence. Described below is the specification of a search configuration:

1. Length The templates must contain exactly the specified number of tokens (tok-pos clusters) in it.
2. Extended chunk tag (ECtag) The templates must have the specified extended chunk tag.
3. Tok-POS clusters and extended POS tags (WC-EPOS) All the specified tok-pos clusters and their corresponding extended POS tags must be present in the templates.

Note that neither of the above constraints needs to be specified for a search configuration, in which case all templates can be considered to fill that position. We use memoization to make the pruning computationally feasible.

Eg. if a search configuration has a length of 5, no preference specified for the extended search tag and (cluster(“run”/NN), “NN”) in the tok-pos (WC-EPOS) list, all templates containing 5 tokens which also contain the cluster for run used as an “NN” are valid for that configuration.

7 Search

To search the very large permutation space, we use a population based searching method which uses only mutation as the genetic operator for generating new solutions. Following are the components of the evolutionary search:

7.1 Population Selection

The search can be parametrized by specifying a collection of sentence level constraints which have their own individual sub-constraints. These constraints are different from search configurations as defined in section 6 as these constraints operate on a sentence level, while search configurations, which are derived from these constraints operate on a template level. The sentence level constraints are listed below:

1. Number of tokens: The number of tokens in the generated sentence must be in a specified range. A maximum value is required.
2. Number of templates: The number of templates in the generated sentence must be in the specified range.
3. Chunk specifications (inclusion): For each chunk specification, at least one template must be present in the sentence which follows it and it must satisfy all the sub-constraints such as extended chunk tag and constituent tok-pos clusters.
4. Position Chunk specifications: This is a chunk constraint like the one described above, with the additional constraint of a fixed position.
5. Ordered Chunk specifications: These are list of chunk constraints with the additional constraint that they be in order in the generated sentence.
6. Tok-pos specifications (inclusion): The sentence must contain the specified tok-pos clusters.
7. Ordered Tok-pos specifications: The specified tok-pos clusters must occur in the same order in the sentence.

Note that every constraint and sub-constraint can be left empty which means that there could be no specification for the extended chunk tag for a chunk constraint. Based on these sentence level constraints, search configurations for each template position are derived.

7.1.1 Sentence level constraints to template level SearchConfigs

Based on the number of templates selected between the range given, it distributes the total length specified between all the templates randomly. Then, it arranges the Chunks, OrderedChunks and PositionFixedChunks and distributes the token level constraints (OrderedTokPOSs and TokPOSConstraints) to the templates. Now, all three parameters of the SearchConfigs have been minimally inferred and the search space can now be pruned.

7.1.2 Sampling from the pruned space

Since we are dealing with naturally occurring text, the distribution of the templates grouped by extended chunk tags follows a Zipf curve meaning that almost 40% of the time, an ”NP” is selected and a “.” almost never gets a chance even though both templates are prominent in the set of templates having the same chunk tag as them. This drastically hampers the chances of getting structural variance in the templates constructed. To remedy this, we assign weights to the set of templates with a dampening exponent of 0.4 which makes the distribution more uniform, yet preserves the selectional biases.

1. Dampen the Zipf for selection of extended chunk tag
2. Dampen the Zipf for selection of templates given an extended chunk tag
3. Obtain the distribution for selection of templates by multiplying probabilities taken from the above two distribution.

This results in a damped Zipf curve for the selection of templates which allows for much more variance in the extended chunk tags of the generated sentences.

7.2 Mutation

To perform mutation, we randomly select a template to mutate and using the dampened distribution obtained, we sample a template from the subspace corresponding to the search configuration of that template position.

7.3 Selection and Evolution Strategy

We use elitist selection (pick the top k organisms) with enforced variability. We enforce that at least 8% and 15% of the population have unique blank construction layout and extended POS construction layout respectively. Following are the steps for evolving the population.

1. Initialize population given a set of Constraints and sample a population of size 1000
2. Mutate the first half of the population and assign it to the second half.
3. compute fitnesses for all organisms and sort population based on fitness score

4. Retain organisms such that the variability constraints are met.
5. Re-initialize 25% of the population so that different search configurations are searched.
6. Repeat 1 to 5 until 100 generations.

This evolution run gives a population consisting of grammatical configurations which adhere to the constraints given as input. There are still possibly word clusters remaining which need to be filled since sentences which are generated are comprised of templates which contain clusters, not word forms. We fill these cluster slots with the tokens we gave in the initial bag-of-words constraints. To do that, we run a random search on the best generated sentence and fill the cluster slots which maximizes the token and head token perplexity.

8 Experiment Setup

Following are the steps we took to conduct our experiments:

1. We tokenized, POS tagged, NER tagged, and chunked and abstracted the English Wikipedia corpus using Stanford CoreNLP (Manning et al., 2014), spaCy (Honnibal and Montani, 2017) and LM-LSTM-CRF (Liu et al., 2018) to extract the templates.
2. The number of clusters, k was chosen to be 7500.
3. We used lexvec(Salle et al., 2016) pretrained vectors for clustering.
4. The weights for the fitness function were empirically chosen to be 75, 10, 10, 5 and 2.

9 Results

Following are the top sentences generated with the following constraints:

1. “cat” should be present:
   the heated water plant is likewise formed entirely of cat .
2. “what” as the first token and sentence contains a “?”:
   what does the right thing do ?
3. “the” in position=0 and “.” in sentence:
   the mid product can readily be used in polynomial practices .
4. “and”, “ate” and “ran” in sentence:
   the division ate a plant of cape , ran a principal prying need and stockpiled superconductivity .
5. “on” and “in” in sentence:
   PERSON bumped ORG in DATE on spirits of error .
6. “influential”, “large” and “red” in sentence:
   large influential player vocals defined red to the convenience of the detector .

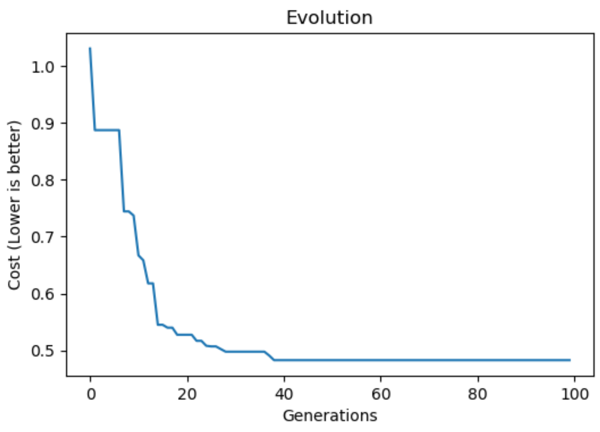

Figure 2: Evolution

We show the generation improvements for the first sentence. As we can see, the algorithm is able to generate a question with minimal supervision. Also, in sentence 5, multiple PPs can be seen.

Some level of supervision is necessary to drive the required syntax and content words to generate predictable outputs. We observe that there is a bias in the model e.g. usage of “the” in the starting noun phrase, and chaining verb phrases and prepositional phrases. Hence, to generate a question, one has to specify the position of the wh word, otherwise the sentences often start with a noun phrase.

Computationally, the search time increases with increasing sentence lengths. On a reasonably modern machine, our implementation generated the above sentences in about 150 seconds while using 2.2 GB of memory.[2]

10 Future Work

As a future work, detailed qualitative analysis and minimum constraints needed to generate specific linguistic structures can be done. Also, automatic extraction of content words and other relevant constraints can be explored for generation.

11 Acknowledgements

The authors would like to thank Prof. Owen Rambow for helping gain insight into the problem.

[2]The code used for this research will be made available on https://github.com/tingc9/LinearTemplatesSRW

References

Nikhilesh Bhatnagar and Radhika Mamidi. 2016. Experiments in linear template combination using genetic algorithms. *22nd Himalayan Languages Symposium*.

Dimitrios Galanis and Ion Androutsopoulos. 2007. Generating multilingual descriptions from linguistically annotated owl ontologies: the naturalowl system. In *Proceedings of the Eleventh European Workshop on Natural Language Generation*, pages 143–146. Association for Computational Linguistics.

Matthew Honnibal and Ines Montani. 2017. spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*.

Ravi Kondadadi, Blake Howald, and Frank Schilder. 2013. A statistical nlg framework for aggregated planning and realization. In *ACL (1)*, pages 1406–1415.

L. Liu, J. Shang, F. Xu, X. Ren, H. Gui, J. Peng, and J. Han. 2018. Empower Sequence Labeling with Task-Aware Neural Language Model. In *AAAI*.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Ehud Reiter and Robert Dale. 2000. *Building natural language generation systems*. Cambridge university press.

Alexander I Rudnicky and Alice H Oh. 2002. Dialog annotation for stochastic generation.

Alexandre Salle, Marco Idiart, and Aline Villavicencio. 2016. Matrix factorization using window sampling and negative sampling for improved word representations. *CoRR*, abs/1606.00819.

Kees Van Deemter, Emiel Krahmer, and Mariët Theune. 2005. Real versus template-based natural language generation: A false opposition? *Computational Linguistics*, 31(1):15–24.