Language Generation via Combinatorial Constraint Satisfaction: A Tree Search Enhanced Monte-Carlo Approach

Maosen Zhang[†], **Nan Jiang**[†], **Lei Li**[‡], **and Yexiang Xue**[†]
[†]Department of Computer Science, Purdue University, Indiana, USA
[‡]ByteDance AI Lab
{maosen,jiang631,yexiang}@purdue.edu, lileilab@bytedance.com

Abstract

Generating natural language under complex constraints is a principled formulation towards controllable text generation. We present a framework to allow specification of combinatorial constraints for sentence generation. We propose TSMH[1], an efficient method to generate high likelihood sentences with respect to a pre-trained language model while satisfying the constraints. Our approach is highly flexible, requires no task-specific training, and leverages efficient constraint satisfaction solving techniques. To better handle the combinatorial constraints, a tree search algorithm is embedded into the proposal process of the Markov chain Monte Carlo (MCMC) to explore candidates that satisfy more constraints. Compared to existing MCMC approaches, our sampling approach has a better mixing performance. Experiments show that TSMH achieves consistent and significant improvement on multiple language generation tasks.

1 Introduction

Supervised techniques still dominate in natural language generation tasks. Despite its success, supervised approaches need to be trained with massive datasets of input-output pairs, which is non-trivial to acquire. In addition, it is hard to guarantee that the output sentences satisfy constraints. Recent approaches first pre-train a language model on a general-purpose dataset, then fine-tune the neural net on a task-specific dataset (Devlin et al., 2019; Radford et al., 2019). These approaches partially mitigate data hunger in training large and flexible neural networks. Nevertheless, they still require carefully crafted datasets for fine-tuning.

We present a constraint satisfaction driven approach for language generation. In particular, we sample sentences that attain high likelihoods from a language model and satisfy task-specific constraints. Sampling sentences that attain high likelihoods in the language model ensures the quality of the generated sentence. Constraints guarantee that the sentences fit the specific language task. The constraints can be hard ones such as the grammar rules, or soft ones such as attaining positive sentiment scores.

Our method harnesses constraint satisfaction,

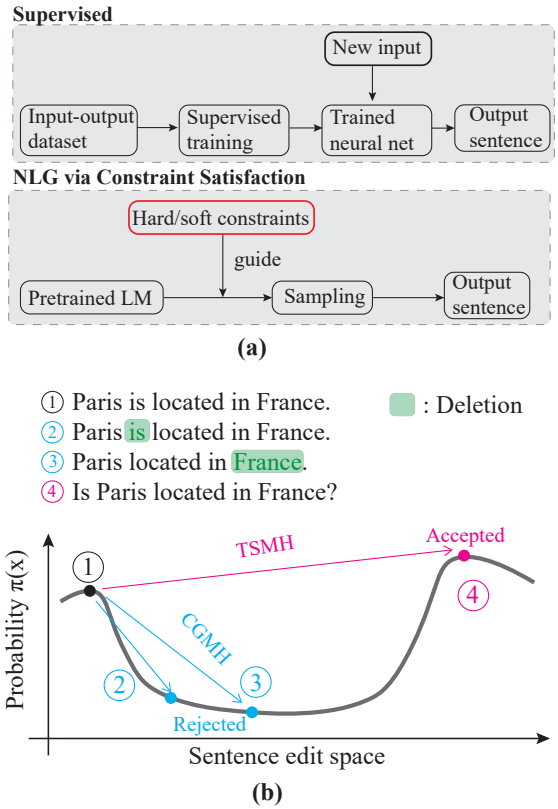

Figure 1: **(a)** Natural language generation via constraint satisfaction (bottom), comparing to supervised approach (up). **(b)** Our proposed tree search enhanced MCMC (TSMH, pink line) traverses the probabilistic space of high-quality sentences more effectively than the baseline (blue line).

[1]https://github.com/Milozms/TSMH

Findings of the Association for Computational Linguistics: EMNLP 2020, pages 1286–1298

rather than learning, to guide language generation. In fact, there is no task-specific training in our approach. Our approach is highly flexible since constraints can be switched quickly to be adapted to a different task, even faster than fine-tuning. It also allows us to leverage the latest developments of automated reasoning for language generation. Although the field of language generation is dominated by learning, reasoning should play an equally important role. Human beings can write beautiful words from reasoning over what is needed in the specific writing task, without learning from previous examples.

To better handle the combinatorial constraints, a tree search is embedded into the proposal process of the Markov chain Monte Carlo (MCMC) for constrained language generation, which suggests candidate proposals that satisfy more constraints. Our approach is motivated by Sample-Search (Gogate and Dechter, 2007a,b, 2011), which integrates backtrack search into importance sampling. Making multiple word-level changes within one proposal step of MCMC allows the direct transition between legitimate sentences, while previous approaches must go through infeasible intermediate states. Such moves are typically rejected by MCMC and therefore result in a slow mixing rate (See Figure 1(b) and Section 3.1).

In literature, constrained language generation has been attacked in a supervised way in (Sutskever et al., 2014; Berglund et al., 2015; Hu et al., 2017; Zhang et al., 2019; Miao et al., 2020). There are also multiple works of literature which model language rules as decomposed tree structures (Lee et al., 2019) or sentiment tags (Su et al., 2018). Markov Logic network (Richardson and Domingos, 2006; Khot et al., 2015) are also used to formulate grammar rules. The distance between vectors representing sentences meaning is considered as soft constraints in (Prabhumoye et al., 2018; Belanger and McCallum, 2016; Amato and MacDonald, 2010). In a nutshell, we summarize our contributions as follows:

1. We define the problem of constraint satisfaction driven natural language generation, and propose a sampling-based approach to tackle the problem with combinatorial constraints.
2. We propose a Tree Search enhanced Metropolis-Hastings approach (TSMH) for the proposed task, which mixes faster than standard MCMC in the presence of combinatorial constraints.
3. Experiment results on generating interrogative, imperative sentences with keywords, and sentences with given sentiments demonstrate that our TSMH is able to generate sentences that satisfy more hard and soft constraints as well as retain good quality.

2 Language Generation via Combinatorial Constraint Satisfaction

We provide a general framework for the constrained natural language generation. In this framework, sentences are generated by sampling from a probability distribution that is proportional to the score of a pre-trained language model times the constraint score. Formally, let x be a sentence, $\pi(x)$ be the probability that x is sampled, then $\pi(x)$ should be:

$$\pi(x) \propto P_{\text{LM}}(x) \cdot \text{Constraint}(x). \quad (1)$$

Here, $P_{\text{LM}}(x)$ is the score of a language model (Sundermeyer et al., 2012; Radford et al., 2019), which measures the quality of sentence x. Higher $P_{\text{LM}}(x)$ means the sentence x is better in quality.

$\text{Constraint}(x)$ is a task-specific penalty term. For example, in interrogative sentences generation, we would enforce $\text{Constraint}(x)$ to guarantee that only sentences in the interrogative form receive high scores. Constraints are composed of hard and soft constraint terms:

$$\text{Constraint}(x) = \Phi_{\text{hard}}(x) \cdot \Phi_{\text{soft}}(x). \quad (2)$$

Both the hard constraint score $\Phi_{\text{hard}}(x)$ and the soft constraint score $\Phi_{\text{soft}}(x)$ are float values ranging from 0 to 1. The closer to 1, the more satisfied the constraints are.

Unlike supervised methods which need to be trained with paired input-output data, our framework can solve language generation tasks without task-specific training. $P_{\text{LM}}(x)$ comes from a language model, only trained on general-purpose language tasks. There is no fine-tuning of $P_{\text{LM}}(x)$ on the specific task. $\Phi_{\text{hard}}(x)$ is based on crafted constraints. $\Phi_{\text{soft}}(x)$ comes from either user-defined functions, or pre-trained neural networks, which again is not fine-tuned on the specific task. The overall formulation composed of the language model and the task-specific constraints allows us to sample sentences which are close to natural language while satisfying constraints.

2.1 Hard Constraints

In this paper, we use propositional logic to define hard constraints $\Phi_{\text{hard}}(x)$. Nevertheless, our sampling approach generalizes to other logic forms. We leave the generalization to first-order logic as future work. For hard constraints, we define $\Phi_{\text{hard}}(x)$ as

$$\Phi_{\text{hard}}(x) = \beta^{M - \sum_i c_i(x)} \tag{3}$$

where $c_i(x)$ is an indicator variable which takes 1 if the sentence x satisfies the i-th constraint, and M is the total number of hard constraints. β is between 0 and 1. We use quite small β values in our experiments, which put a large penalty on violating one hard constraint. We also define *Constraint Error* $C(x)$ as the number of hard constraints a sentence violates, i.e., $C(x) = M - \sum_i c_i(x)$. Constraints are defined in the logical form of word categories.

Word Category Division We divide the entire vocabulary into several categories of words. Given vocabulary set U, we partition U into non-overlapping subsets: $\mathcal{V} = \{V_1, V_2, \ldots, V_{|\mathcal{V}|}\}$, satisfying: (i) all V_i are subsets of U: $V_i \subseteq U,\ \forall V_i \in \mathcal{V}$; (ii) categories are non-overlapping: $V_i \cap V_j = \varnothing,\ \forall V_i, V_j \in \mathcal{V}, i \neq j$; (iii) V_i together cover the whole vocabulary: $\bigcup_i^{|\mathcal{V}|} V_i = U$.

The word category division strategy varies for different tasks. For example, we split the whole vocabulary into $\mathcal{V} = \{\texttt{[QWH]}, \texttt{[AUX]}, \texttt{[OTH]}\}$ for generating interrogative sentences. Here, $V_1 = \texttt{[QWH]}$ represents the set of *wh*-words leading a question: *what, when, where, which, who, whom, whose, why, how*. $V_2 = \texttt{[AUX]}$ represents the set of auxiliary verbs and copula words: *do, does, did, be, am, are, is, ...*, etc. $V_3 = \texttt{[OTH]}$ means all other words in the vocabulary. We may use another division in, e.g., generating imperative sentences. Sometimes we need to generate sentences with keywords. We let each keyword forms a category. For example, to generate interrogative sentences with the keyword *learning*, the division would be: $\mathcal{V} = \{\texttt{[QWH]}, \texttt{[AUX]}, [\textit{learning}], \texttt{[OTH]}\}$.

Hard Constraints Given a sentence with length m [2], let $w_i^{V_j} \in \{true, false\}$ be an indicator variable that the i-th word in the sentence is in category V_j. For example, variable $w_1^{\texttt{[QWH]}} = true$ if and only if the first word in sentence is a *wh*-like word. For sentence-level constraints, we can define them using propositional logic over $w_i^{V_j}$ (`and` ($\wedge$), `or` ($\vee$), `not` ($\neg$)). We give a few examples below.

Enforcing Keywords in a Sentence Given one keyword `K`, we can enforce its existence in the sentence using the following constraint:

$$w_1^{\texttt{[K]}} \vee w_2^{\texttt{[K]}} \vee \cdots \vee w_m^{\texttt{[K]}}.$$

here `[K]` is a set containing the keyword `K`. We formulate this constraint assuming a known sentence length m. Indeed, length m is a variable and can vary over the sampling procedure. Nevertheless, as we can see shortly in the sampling process, the lengths are known for both sentences when transiting from one sentence to another. Therefore, the semantic meaning of m is clear during sampling. Details on the sampling process is in Section 3.2.

Enforcing Imperative Sentence According to the definition in (Aarts, 1989), the starting word of an imperative sentence should be either a verb: $w_1^{\texttt{[VERB]}}$ or an adverb followed by a verb: $w_1^{\texttt{[ADV]}} \wedge w_2^{\texttt{[VERB]}}$. We encode such constraint as:

$$w_1^{\texttt{[VERB]}} \vee (w_1^{\texttt{[ADV]}} \wedge w_2^{\texttt{[VERB]}}).$$

Enforcing Interrogative Sentence We use the following two constraints to enforce the sentence to be interrogative: (i) The first word is in `[QWH]`. (ii) The second or third word in the sentence is in `[AUX]`. (i, ii) can be written together as:

$$w_1^{\texttt{[QWH]}} \wedge \big((w_2^{\texttt{[AUX]}} \wedge \neg w_3^{\texttt{[AUX]}}) \vee (w_3^{\texttt{[AUX]}} \wedge \neg w_2^{\texttt{[AUX]}})\big).$$

This constraint is similar to the definition in (Zhang et al., 2017). We acknowledge that this is a relaxed constraint. Nevertheless, our sampling approach also consider the score from language model. These constraints accompanied with the language model guide us to good interrogative sentences in practice.

2.2 Soft Constraints

A soft constraint assigns a float value between 0 and 1 to indicate how the constraint is satisfied. For tasks with only hard constraints, $\Phi_{\text{soft}}(x)$ is set to 1.0. Soft constraints can be derived quite flexibly. It can be from a user-defined function (see "sentence similarity" for an example), or from a pre-trained neural network (see "sentiment score"):

Sentence Similarity We can define a soft constraint function ensuring that the generated sentence x is close to the reference sentence y in semantic meaning. For one word in sentence x, we first find

[2] As we conduct sampling for the sentence, sentence length is pre-known and we set m as the length of the longest one.

the closest word in sentence y by computing their cosine similarity. Then either the minimum or the average of these words' cosine similarity is taken as the similarity score for sentence x and y.

Sentiment Score We can enforce that the generated sentence must have a given sentiment by enforcing the value for the sentence from a sentiment analysis model. The output score of a sentiment analysis neural net represents whether the sentence has a positive or negative sentiment. We use this score as a soft constraint to control the sentiment of generated sentence with positive or negative attitude. Notice that the sentiment analysis neural net is pretrained on a separate dataset and remains intact in our framework.

This setup gives us additional flexibility. To be specific, if we need to generate sentences that contain keywords while having a given sentiment, it is difficult to find a large dataset of this type and the performance of a pure learning approach may be limited. To summarize, the main attribute of the constraint satisfaction framework is allowing a formulation using both hard and soft constraints, without the need of task-specific training or tuning.

3 Tree Search Enhanced MCMC

Markov chain Monte Carlo (MCMC) is a classical approach to sample sentences from probability distribution $\pi(x)$ as defined in Equation 1. Starting from one sentence x, MCMC moves to the next sentence x^* by first generating a sample x^* from the proposal distribution $Q(x^*|x)$ and then accept x^* with the following acceptance rate $A(x^*|x)$:

$$A(x^*|x) = \min\left\{1, \frac{\pi(x^*)Q(x|x^*)}{\pi(x)Q(x^*|x)}\right\}, \qquad (4)$$

If sentence x^* is rejected, then the sample remains at x. The distribution of samples will converge to the sentence stationary distribution of Markov chain $\pi(x)$. Previous work (Miao et al., 2019) proposes to use MCMC for constrained sentence generation, namely CGMH algorithm. Their proposal distribution only suggests sentences with one-word modification. Nevertheless, CGMH cannot handle the combinatorial constraints in our problem definition, because of the *low acceptance ratio problem* caused by the *locality* of the proposal distribution. In other words, the sampling process can only visit a limited number of neighbors, thus the Markov chain will easily be trapped at one infeasible state, resulting in a lot of rejections. We illustrate this problem in detail and hence motivate our tree search embedded MCMC approach using the following example.

3.1 Motivation: Breaking the low acceptance barrier

Suppose we need to generate a question, whose answer comes from an underlined part of a sentence. For example, suppose we underline *France* in the sentence:

A: Paris is located in France.

The question we would like to generate is:

B: Which country is Paris located in?

Under our constraint satisfaction framework, we define $\text{Constraint}(x)$ so that real interrogative sentences such as question *B* would receive high probability in the defined $\pi(x)$. Our constraints are: (i) the whole sentence is in the interrogative form. (ii) *Paris* and *located* must appear in the sentence. We run MCMC starting from sentence *A*.

It is hard for MCMC without tree search to generate question *B* in a reasonable time starting from *A*. Because the edit distance between sentence *A* and *B* is larger than 2, we cannot generate *B* from *A* with one step of word insertion, removal, or replacement. In order for CGMH to reach *B* from *A*, it has to encounter a few intermediate steps. Without loss of generality, suppose CGMH proposes sentence *C* in one MCMC step by removing *is*:

C: Paris ~~is~~ located in France.

Notice that *C* is not a legitimate English sentence, so its language model score $P_{\text{LM}}(x)$ becomes much smaller compared to the original sentence *A*. In addition, *C* violates more constraints than *A*, which decreases its $\text{Constraint}(x)$ score as well. In MCMC, the probability to accept the move from *A* to sentence *C* is given by Equation 4, in which the dominating term is $\frac{\pi(C)}{\pi(A)} = \frac{P_{\text{LM}}(C)\ \text{Constraint}(C)}{P_{\text{LM}}(A)\ \text{Constraint}(A)}$. Because both $P_{\text{LM}}(C)$ and $\text{Constraint}(C)$ are smaller, the acceptance ratio becomes really small. In fact, we found the acceptance ratio to be 5.93×10^{-12} in our experiment. This means that it will take CGMH many steps (on the order of 10^{12}) to move one step from sentence *A* to *C*. Figure 2 (left) demonstrates this. It is easy to verify that barriers of low acceptance rate exist on every path from sentence *A* to *C* and thus the rejection problem exists.

On the other hand, if we allow the proposal distribution to suggest sentences with multiple word-level changes, one can transit from sentence *A* to *B* through all legitimate sentences as intermediate

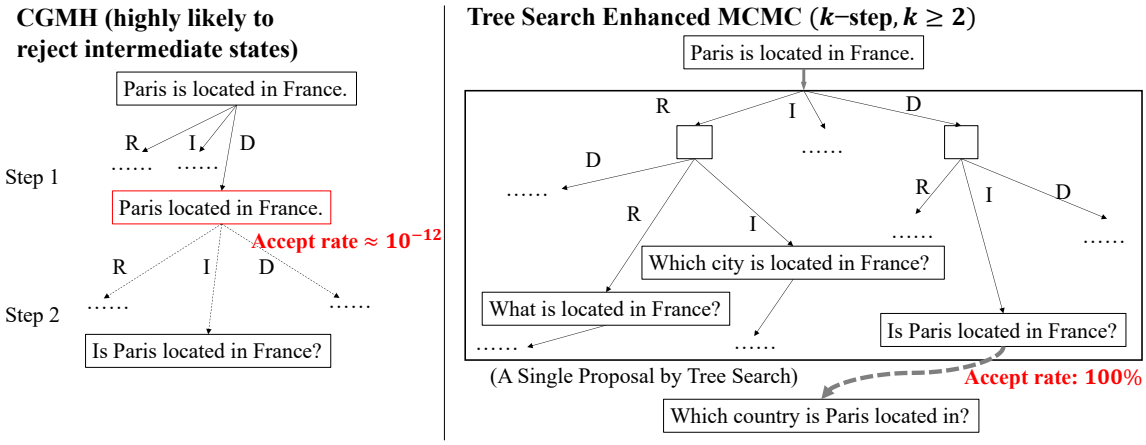

Figure 2: Our method, tree search embedded MCMC (TSMH), outperforms CGMH in generating sentences with complex combinatorial constraints. (Left) CGMH must pass intermediate sentence states (highlighted in red), which have very low acceptance rate to reach the intermediate sentence *Is Paris located in France?* starting from sentence *Paris is located in France*. This results in the poor performance of CGMH when handling combinatorial constraints. (Right) By embedding a tree search into MCMC, TSMH can reach the an intermediate sentence from the starting sentence in one step, and with an acceptance rate of 100%. R, I, D mean *replace, insert, delete*. See Section 3.1 for a detailed discussion.

steps. Consider the following two-step change:

1. First delete *is* and insert *is* before *Paris*. This changes sentence *A* to *D:*
   Is Paris located in France?
2. Delete *France* and insert *Which* and *country*. This changes sentence *D* to *B*.

Because the intermediate step sentence D is a legitimate English sentence and $\text{Constraint}(D) = \text{Constraint}(A)$, $\frac{\pi(D)}{\pi(A)}$ is close to 1, resulting in a 100% acceptance ratio in this step. When changing from D to B, notice that B is also a legitimate sentence and it satisfies more constraints than D. In fact, the acceptance ratio is also 100%. Figure 2 (right) demonstrates this case.

For tasks with soft constraints, there are also similar rejection problems for CGMH. For example, *"Nothing is impossible"* is a sentence with positive sentiment. If we insert, replace or delete one word, it is hard to keep the sentence valid and preserve the positive sentiment.

Motivated by these examples, we propose the embed a tree search into the proposal process of MCMC to solve the rejection problem, which suggests candidate sentences with multiple word-level changes and satisfy more constraints.

3.2 TSMH Algorithm Implementation

Our Tree Search enhanced Metropolis-Hastings (TSMH) still follows the classical MCMC procedure. The only difference is a new proposal distribution $Q(x^*|x)$ generated from a tree search process. The tree search defines a probability distribution over *templates* of sentence moves. Each template defines a subset of possible moves. The sentences within the same template satisfy the same hard constraints score $\Phi_{\text{hard}}(x)$. The proposal probability distribution induced by the tree search algorithm biases towards templates that have high $\text{Constraint}(x)$ scores.

A **template** defines a set of sentences where each word is either given or specified by a word category. For example, a template `[[QWH],[AUX],[OTH],[OTH]]` restricts that the first word must be a *wh*-word, the second word must be an auxiliary verb and the last two words must be other words.

Notice that we can decide how many hard constraints a sentence satisfies at the template level, since the indicator variables in the constraints defined in this paper only restrict the categories of words. For example, the template `[[QWH],[AUX],[OTH],[OTH]]` satisfies the constraints of being an interrogative sentence defined in Section 2. Our proposal procedure first sample a template and then fills in this template with words based on a language model.

Overview of the Proposal Process During the sampling process, suppose we are at one sentence x. We will sample a new sentence x^* from the proposal distribution as follows. First, our algorithm will decide the ***positions*** of the words to change

by random selection. Typically, our algorithm will change more than one word. Then we use a tree ***search***, which enumerates all possible operations on the selected words. This includes deciding the operation on each word (*insert, delete, or replace*) as well as the associated word category in case of *insert* and *replacement*. In this case, every leaf branch of the search tree will be a sentence template. Because the number of word categories is limited, the tree search procedure is often cheap. As discussed, we can infer the number of hard constraints satisfied based on the template associated with each tree leaf. We then ***rank*** these templates based on the number of constraints satisfied and sample one template based on a geometric series, favoring templates that satisfy more constraints. Finally, we ***fill*** in the sampled template with words suggested by a language model, and then select one filled sentence $\hat{x}$ as proposal, according to the language model score times the soft constraint score $P_{\text{LM}}(\hat{x}) \cdot \Phi_{\text{soft}}(\hat{x})$. Soft constraints $\Phi_{\text{soft}}(x)$ give us a real number, which is similar to the language model $P_{\text{LM}}(x)$. We treat them together with the language model in the proposal process.

Our approach alleviates the rejection problem of MCMC by enumerating all possibilities in the space of multiple word change at the template level, based on the analysis in section 3.1. This process enables us to handle combinatorial constraints. Tree search also allows us to prune useless branches.

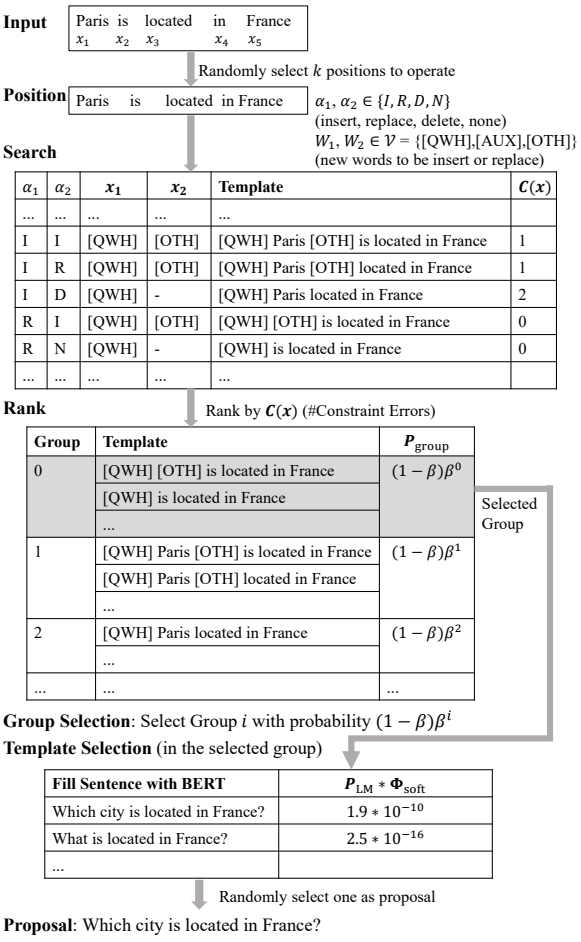

Figure 3: The proposal process of Tree Search Embedded MCMC. The input is the current sentence (state) and the output is the proposed sentence. This proposal process favors sentences satisfying a large number of constraints.

3.2.1 Detailed Search Procedure

The procedure of searching proposals in our tree search embedded MCMC is as follows and shown in Figure 3.

Position Randomly select k positions $\{t_1, \ldots, t_k\}$ to perform word-level operations with uniform probabilities, where k is the size of the search steps. The probability of getting each combination of positions is: $P_{\text{pos}} = 1/\binom{m}{k}$, where m is the length of the sentence.

Search Search and iterate all different operations and all different word categories (mentioned in Section 2.1) for each selected position. For example, if we have $|\mathcal{V}|$ word categories and the operation set {*replace, insert, delete, none*} , we need to enumerate $(2|\mathcal{V}|+2)^k$ different combinations of operations and word categories. We use word placeholders `[MASK]` to represent the unknown inserted or replaced words. We keep track of all the generated **templates** and their corresponding numbers of violated constraints.

Rank and Group Selection We define a group as the set of templates which violate the same number of constraints. We sort all templates by its number of violated constraints (constraint error) C in ascending order, and put templates with the same C into one group. We then randomly select group i with probability: $P_{\text{group}} = (1-\beta) \cdot \beta^{C_i - \min_j C_j}$, where C_i is the constraint error of group i, and β is a very small float value (like 10^{-10}). In this way, we favor choosing the group satisfying the largest amount of constraints, while also ensuring the irreducibility of the Markov chain. Let the chosen group at this step be G_i.

Fill and Template Selection In this step we will first fill every template with words in the selected group G_i, then we select one filled template as the proposal. Because the template restricts the masked word to be chosen only from the corresponding

word category, we ***fill*** it by selecting words from the given word category. The probability of selecting the t_i-th word P_{Fill_i} is the conditional probability of filling words at this locations given contexts: $P_{\text{LM}}(x_{t_i}|x_1, ..., x_{t_i-1}, x_{t_i+1}, ..., x_m)$. The probability of getting one sampled sentence is: $P_{\text{fill}} = \prod_{i=1}^{k} P_{\text{fill}_i}$, where i means the word level action for i-th position we selected. If the operation in t_i is *delete* or *none*, then $P_{\text{fill}_i} = 1$. We sample one template within the group (together with the corresponding sampled sentence) according to the sentence probability times soft constraint score: $P_{\text{template}} = \frac{P_{\text{LM}}(x^*)\cdot\Phi_{\text{soft}}(x^*)}{\sum_{\hat{x}\in G_i} P_{\text{LM}}(\hat{x})\cdot\Phi_{\text{soft}}(\hat{x})}$.

The proposal distribution $Q(x^*|x)$ leading from sentence state x to x^* in this procedure is $Q(x^*|x) = P_{\text{pos}}P_{\text{group}}P_{\text{fill}}P_{\text{template}}$.

4 Experiments

We evaluate our approach on three applications: interrogative, imperative, and fixed sentiment sentences generation. In each task, we construct the specified type of sentences by sampling starting from keywords and enforcing task-specific constraints. For each task, we run our TSMH algorithm for 100 steps, with 100 candidate sentences generated. k is set to 3. Since the tree search in TSMH considers changing 3 words at each iteration, we run the baseline CGMH for 300 steps as a comparison. We select the sentence with the highest $\pi(x)$ value among the sentences generated by each algorithm as the output. Our results are summarized in Table 1.

In general, our method TSMH outperforms baselines and generates sentences that satisfy more constraints, are of good quality and are likely to be close to the natural language. Our main results are summarized in Table 1, in which Valid% denotes the percentage of generated sentences that satisfy all constraints. $\pi(x)$ is the value of the stationary probability $P_{\text{LM}}(x) \cdot \text{Constraint}(x)$. $P_{\text{GPT}-2}(x)$ is language model probability estimated by a pre-trained GPT-2 model, which measures the quality of the sentences. Accept% means the acceptance rate of MCMC. Detailed experiment settings can be reviewed in appendix A.1.

4.1 Interrogative Sentence Generation

In the interrogative sentence generation, we construct interrogative sentences by sampling starting from the keywords. We enforce that sentences with a high probability to be sampled must satisfy grammar constraints of being interrogative and contain a few given keywords. The constraint definition for interrogative sentences is in section 2.1.

According to the results, in the experiment with keywords, 92.67% of the output sentences of our TSMH algorithm satisfy all the constraints, while merely 18.33% satisfy constraints for the baseline. The numbers are 83.17% and 45.50% for the experiment without keywords, respectively. This demonstrates that our TSMH generates sentences with more constraints satisfied. In addition, our method has a higher $\pi(x)$ (stationary probability value) and acceptance rate, suggesting that the tree search embedded help MCMC to mix faster. Overall, our method TSMH can handle more complicated constraints in language generation tasks.

Human Evaluation We conduct human evaluation for interrogative sentences generated with keywords. We present human participants from the Amazon Mechanical Turk with a pair of sentences at a time. One sentence is generated by our TSMH model and the other one is from the baseline CGMH. We ask human participants which sentence is better in terms of fluency and grammar. In terms of the experimental setting, we use 100 sentence pairs generated by CGMH and TSMH with the same keyword inputs. We randomly split these 100 test sentence pairs into 5 survey groups, and then deploy them on the Amazon Mechanical Turk. We randomly assign human participants to survey groups. When showing the sentence pairs, we also provide the keywords that the sentences must contain. We ask human participants to vote which sentence in the pair is better in terms of grammar coherence, keyword coverage and fluency. We use a gold-standard question to detect if the voter is randomly doing the survey. Every valid survey contains a randomized set of 20 questions. We received in all 580 votes. Each question pair receives votes ranging from 5 to 11. As shown in Table 2, sentences from our model receive almost twice times of votes than the baseline, which suggests that the sentences generated by our approach is better in human evaluation.

Case Studies As shown in Table 3, we compare some output sentences of our method with the baseline using the same inputs and keywords. More examples can be seen in the appendix A.2. From these cases, we can see that our method generates sentences with better quality.

Comparison with Other Methods We compare

Tasks	Methods	#sample	step	Valid%	$\pi(x)$	$P_{\text{GPT}-2}(x)$	Accept%
Interrogative	CGMH	300	1	18.33%	2.60E-04	1.78E-18	5.45%
	TSMH (Ours)	100	3	**92.67%**	**1.44E-03**	**5.51E-18**	**24.50%**
Imperative	CGMH	300	1	91.32%	0.0004	9.86E-16	5.49%
	TSMH (Ours)	100	3	**97.75%**	**0.0060**	**6.60E-15**	**15.66%**
Sentiment	CGMH	300	1	96.33%	4.93E-19	4.57E-22	6.72%
	TSMH (Ours)	100	3	**96.67%**	**7.94E-04**	**1.82E-18**	**11.09%**

Table 1: Our method TSMH outperforms CGMH by generating sentences that satisfy more constraints, are of good quality and are likely to be natural language. Column Valid% shows the percentage of generated sentences that satisfy all constraints, which TSMH clearly leads baselines. In addition, TSMH has better acceptance rates (Accept%). The language generated by TSMH is also of good quality, because it matches other models in language model scores $P_{\text{GPT}-2}(x)$. Multiplying both the language model score and the constraint score, the sentences generated by TSMH tend to attain higher stationary probability $\pi(x)$.

Methods	#Votes	Votes%
CGMH	196	33.64%
TSMH (Ours)	**384**	**66.36%**

Table 2: Human evaluation of the quality of the generated interrogative sentences from keywords in terms of fluency and grammar. Most human participants (native speakers) agree that the sentences generated by our TSMH are better in quality compared to CGMH.

Keys	waste heat water
CGMH	what waste is there, it seems now?
TSMH	where was the waste - water heater?
Keys	responses protect lungs
CGMH	how can immune responses also occur by not only infecting pathogens in the central nervous system?
TSMH	what responses do your lungs have to protect you from pathogenic bacteria?
Keys	median temperature winter
CGMH	what do you mean we have median temperature winter and spring, anyways?
TSMH	what is the median temperature range in the winter months?
Keys	catholics concentrated france
CGMH	the catholics are now mainly concentrated there.
TSMH	why are the french roman catholics so densely concentrated in southern france?

Table 3: Case study of generating interrogative sentences with keywords, where Keys stands for keywords. Full case study is in the supplementary materials.

our TSMH method with UQA (Lewis et al., 2019). The setting of UQA is different from us: it takes a paragraph as input and generates a corresponding question. Although this comparison is not fair, the baseline is the most similar and the best framework that we can compare with. To run UQA, we use the corresponding original sentences from which the keywords of TSMH are extracted as the input. In other words, for TSMH, the inputs are keywords extracted from the SQuAD 2.0 (Rajpurkar et al., 2018) questions. For UQA, we take the corresponding paragraphs of the selected questions as input. This also gives UQA additional advantage because it has access to a paragraph, rather than keywords. To make it more comparable, we remove the keyword constraints in this experiment. In Table 4, we compare the language model scores $\log P_{\text{LM}}$ of the generated sentences that reflect the naturalness and fluency, and the stationary probability $\pi(x)$ and valid percentage Valid% that show how good it satisfies our pre-defined constraints. We pointed out that UQA was trained on the specific interrogative sentences while our method was not trained at all.

Methods	$\pi(x)$	Valid%	$\log P_{\text{LM}}$
UQA	0.0024	50%	-92.75
TSMH	**0.0063**	**83.17%**	**-58.27**

Table 4: Comparison with UQA. Our TSMH outperforms UQA in terms of the percentage of satisfying the interrogative sentence constraints, and has a higher score predicted by a language model, despite UQA is trained on specific interrogative sentences while our method is not trained at all.

4.2 Imperative Sentence Generation

We generate imperative sentences via sampling starting from the keywords. We enforce grammar constraints of being an imperative sentence: the starting word should be either a verb $w_1^{[\text{VERB}]}$ or

an adverb followed by a verb $w_1^{[\text{ADV}]} \wedge w_2^{[\text{VERB}]}$. We also enforce keyword constraints in this task.

As shown in Table 1, our method has a higher valid percentage of 97.75% compared to 91.32% of the baseline, showing that the sentences generated by our method can satisfy more constraints. Our method has a higher $\pi(x)$ (stationary probability value) and acceptance rate, suggesting our approach has a better mixing behavior. Overall, results show that our method using Tree Search Embedded MCMC can handle more complicated combinatorial constraints in language generation.

4.3 Sentence Generation with Given Sentiments

In this task, we require the sentences to contain the specified keywords and have positive sentiments (Fu et al., 2019). We enforce the sentences to attain high scores from a sentiment analysis neural network. We also enforce keyword constraints as hard constraints. We need to emphasize that, our method uses a model pre-trained on a separate dataset for sentiment analysis, which is kept intact in our experiment. No additional fine-tuning to the sentiment analysis model was performed. we consider two sub-tasks in Table 5: (i) positive sentiment to positive sentiment (P2P), where the input keywords are extracted from sentences which originally have positive sentiments; (ii) negative sentiment to positive sentiment (N2P), where the keywords are extracted from sentences with negative sentiments. N2P is more difficult as it requires transforming the sentiment.

Our method has a higher sentiment score, suggesting that our method generates sentences with more positive sentiments (better aligned with the target of this experiment). The increase against CGMH is bigger on the more difficult N2P task, which requires flipping the sentiment. Our model also leads in terms of language model scores, suggesting the language quality is better.

Tasks	Method	$\pi(x)$	$P_{\text{GPT-2}}$	Accept%	Senti
P2P	CGMH	9E-19	8E-22	8.16%	0.8647
	TSMH	**4E-04**	**2E-18**	**12.23%**	**0.8801**
N2P	CGMH	5E-20	6E-23	5.65%	0.3470
	TSMH	**1E-03**	**7E-19**	**9.91%**	**0.5254**

Table 5: Generate sentences with positive sentiment. Half of the input are extracted from positive sentences (P2P), and the other half are from negative (N2P), which are harder to transform to positive sentences.

Methods	$\pi(x)$	$P_{\text{GPT-2}}(x)$	Sentiment
CtrlGen	3.19E-07	4.64E-22	0.4614
TSMH	**1.16E-03**	**7.07E-19**	**0.5254**

Table 6: Compare with CtrlGen (Hu et al., 2017) over the N2P subtask with acceptance rate, language score and sentiment score metrics.

Comparison with Other Methods We compare our method with CtrlGen (Hu et al., 2017). The setting is a little different from ours: it takes a sentence with a negative sentiment as input and transforms it to positive, without the guarantee of satisfying keyword constraints. Our method takes a set of keywords as input. To make the outputs comparable, we select the same set of negative sentences as the input of CtrlGen and extract the keywords of those sentences as the input of TSMH. Our method requires no additional training besides a pre-trained sentiment analysis model and a pre-trained language model, while CtrlGen requires training the auto-encoder.

The results in Table 6 show that our method outperforms CtrlGen in terms of both sentence quality and sentiment, as the sentences generated by our method receive higher language model scores and sentiment scores.

5 Conclusions

We propose a framework for constraint-driven language generation via sampling and combinatorial constraint satisfaction. Our solution strategy is to sample sentences from the constrained space with probability proportional to the scores of the language model. To better handle the combinatorial constraints, a tree search is embedded into the proposal process of MCMC to suggest candidate proposals that satisfy more constraints. Experiments demonstrate that our approach generates sentences that satisfy more constraints, are of good quality and are likely to be close in quality to the natural language.

Acknowledgements

This research was supported by the National Science Foundation (Award number IIS-1850243 and CCF-1918327). The computing infrastructure was partially supported by the Microsoft AI for Earth computing award. The authors would like to thank Mr. Ning Miao for valuable suggestions.

References

Flor Aarts. 1989. Imperative sentences in english: semantics and pragmatics. *Studia Linguistica*, 43(2):119–134.

Michael S Amato and Maryellen C MacDonald. 2010. Sentence processing in an artificial language: Learning and using combinatorial constraints. *Cognition*, 116(1):143–148.

David Belanger and Andrew McCallum. 2016. Structured prediction energy networks. In *Proceedings of the 33nd International Conference on Machine Learning*, pages 983–992.

Mathias Berglund, Tapani Raiko, Mikko Honkala, Leo Kärkkäinen, Akos Vetek, and Juha Karhunen. 2015. Bidirectional recurrent neural networks as generative models. In *Advances in Neural Information Processing Systems*, pages 856–864.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics.

Yao Fu, Hao Zhou, Jiaze Chen, and Lei Li. 2019. Rethinking text attribute transfer: A lexical analysis. In *the 12th International Conference on Natural Language Generation (INLG)*.

Vibhav Gogate and Rina Dechter. 2007a. Approximate counting by sampling the backtrack-free search space. In *Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence*, pages 198–203.

Vibhav Gogate and Rina Dechter. 2007b. Samplesearch: A scheme that searches for consistent samples. In *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics, AISTATS*, pages 147–154.

Vibhav Gogate and Rina Dechter. 2011. Samplesearch: Importance sampling in presence of determinism. *Artif. Intell.*, 175(2):694–729.

Zhiting Hu, Zichao Yang, Xiaodan Liang, Ruslan Salakhutdinov, and Eric P. Xing. 2017. Toward controlled generation of text. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, pages 1587–1596.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017, Valencia, Spain, April 3-7, 2017, Volume 2: Short Papers*, pages 427–431. Association for Computational Linguistics.

Tushar Khot, Niranjan Balasubramanian, Eric Gribkoff, Ashish Sabharwal, Peter Clark, and Oren Etzioni. 2015. Exploring markov logic networks for question answering. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 685–694.

Jay Yoon Lee, Sanket Vaibhav Mehta, Michael Wick, Jean-Baptiste Tristan, and Jaime G. Carbonell. 2019. Gradient-based inference for networks with output constraints. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pages 4147–4154.

Patrick S. H. Lewis, Ludovic Denoyer, and Sebastian Riedel. 2019. Unsupervised question answering by cloze translation. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 4896–4910. Association for Computational Linguistics.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Rose Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics, ACL 2014, June 22-27, 2014, Baltimore, MD, USA, System Demonstrations*, pages 55–60. The Association for Computer Linguistics.

Ning Miao, Yuxuan Song, Hao Zhou, and Lei Li. 2020. Do you have the right scissors? tailoring pre-trained language models via monte-carlo methods. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 3436–3441. Association for Computational Linguistics.

Ning Miao, Hao Zhou, Lili Mou, Rui Yan, and Lei Li. 2019. CGMH: constrained sentence generation by metropolis-hastings sampling. In *The Thirty-Third AAAI Conference on Artificial Intelligence*, pages 6834–6842.

George A. Miller. 1995. Wordnet: A lexical database for english. *Commun. ACM*, 38(11):39–41.

Shrimai Prabhumoye, Yulia Tsvetkov, Ruslan Salakhutdinov, and Alan W. Black. 2018. Style transfer through back-translation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*, pages 866–876.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Pranav Rajpurkar, Robin Jia, and Percy Liang. 2018. Know what you don't know: Unanswerable questions for squad. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL*, pages 784–789.

Matthew Richardson and Pedro M. Domingos. 2006. Markov logic networks. *Machine Learning*, 62(1-2):107–136.

Stuart Rose, Dave Engel, Nick Cramer, and Wendy Cowley. 2010. Automatic keyword extraction from individual documents. *Text mining: applications and theory*, 1:1–20.

Jinyue Su, Jiacheng Xu, Xipeng Qiu, and Xuanjing Huang. 2018. Incorporating discriminator in sentence generation: a gibbs sampling method. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence*, pages 5496–5503.

Martin Sundermeyer, Ralf Schlüter, and Hermann Ney. 2012. LSTM neural networks for language modeling. In *13th Annual Conference of the International Speech Communication Association*, pages 194–197.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *CoRR*, abs/1910.03771.

Russ Wolfinger and Michael O'connell. 1993. Generalized linear mixed models a pseudo-likelihood approach. *Journal of statistical Computation and Simulation*, 48(3-4):233–243.

Huangzhao Zhang, Hao Zhou, Ning Miao, and Lei Li. 2019. Generating fluent adversarial examples for natural languages. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, pages 5564–5569. Association for Computational Linguistics.

Shijie Zhang, Lizhen Qu, Shaodi You, Zhenglu Yang, and Jiawan Zhang. 2017. Automatic generation of grounded visual questions. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence*, pages 4235–4243.

Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. 2015. Character-level convolutional networks for text classification. In *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 649–657.

A Appendix

A.1 Detailed Experiment Settings

In this section, we detail our experimental settings for interrogative, imperative, and sentimental sentence generation tasks, along with the process of human evaluation.

In the expression of stationary distribution Eq.(1), the first term $P_{\mathrm{LM}}(x)$ is evaluated by the BERT model, which is based on the huggingface's BERT implementation (Wolf et al., 2019). We use BERT-base in our experiments, with hyper-parameters: L=12, H=768, A=12, Total Parameters=110M. To evaluate the term $P_{\mathrm{LM}}(x)$ with BERT model, we multiply the BERT score of masking and querying the conditional probability of each word in sentence x, close in form of the pseudo-likelihood (Wolfinger and O'connell, 1993). Since we only requires $\pi(x)$ to be proportional to $P_{\mathrm{LM}}(x)$ times the constraint score, $P_{\mathrm{LM}}(x)$ does not need to be normalized.

A.1.1 Interrogative Sentences Generation

According to the adapted definition of interrogative sentence grammar, the first word should be a question word, and there should be an auxiliary verb at a suitable position. The constraint definition for interrogative sentences is in section 2.1. In our actual implementation, we also enforce that there should be only one question word and one auxiliary verb in the sentence in order to improve the quality of generated sentences. The question words include *what, when, where, which, who, whom, whose, why, how*; the auxiliary verbs include *do, does, did, be, am, are, is, was, were, shall, will, should, would, can, could, may, might, must*.

For the task of generating interrogative sentences with keywords, we also enforce the keyword only appear once in the sentence.

The dataset of this task is based on the SQuAD 2.0 dataset (Rajpurkar et al., 2018), where we select 600 questions and removing the stop words using the Rake toolkit (Rose et al., 2010).

A.1.2 Imperative Sentences Generation

The dataset for generating imperative sentences is retrieved from[3]. We select 300 sentences and extract the keywords from the sentences as our input. According to the grammar of imperative sentences, we need to verify if the word is a present tense verb. In the implementation, we use the POS tag information in WordNet and Stanford CoreNLP as the criterion for deciding the word POS tag of the given word. We first select all the words with at least one verb meaning in WordNet (Miller, 1995), then use Stanford CoreNLP (Manning et al., 2014) to get POS tags for each word and only preserve the present tense form of verbs.

A.1.3 Sentiment Sentence Generation

This application requires the set of input keywords and an external sentiment classifier, which is used to estimate whether the sentiment of the sentence is positive or not. To estimate the sentiment score of the sentences, we train a sentiment analysis model with fastText (Joulin et al., 2017) on Yelp Review Polarity dataset (Zhang et al., 2015). The input keywords are extracted from 300 selected sentences in the Yelp test set. Half of the original sentences are positive, and the other half are negative (which is harder to transform to positive sentences).

With input keywords of positive and negative sentiment, we enforce the model to generate sentences with positive sentiment. The second sub-task with negative sentiment keywords is much more difficult than the sub-task with positive sentiment keywords, as it requires transforming from negative to positive sentiment.

A.2 Case Studies

As shown in Table 7, we compare some output sentences of our method with the baseline using the same inputs and keywords. From these cases, we can see that the baseline sometimes generates awkward or disordered sentences. For example, the baseline generates one sentence:"*how was lower normandy ever truly founded?*". Although this sentence seems to satisfy the constraints of an interrogative sentence, its meaning is awkward. The sentence generated by our method is "*when was the duchy of normandy founded?*", which is more realistic. Also, the sentence from the baseline "*and please be a very very careful*" does not follow imperative grammar, and "*the catholics are now mainly concentrated there*" is not a question.

[3]https://github.com/lettergram/sentence-classification

Keys	university warsaw established
TSMH	when was the technical university of warsaw first formally established?
CGMH	polish polytechnical institute - university of technology warsaw - was established here in 1964?
Keys	organization charge running
TSMH	who would charge her with running such an organization?
CGMH	who else would charge him with running a very profitable business?
Keys	tribes khan fight
TSMH	what tribes would fight back against the genghis khans?
CGMH	why else would tribesmen like gen. and gen. genghis khan fight them off?
Keys	european travel amazon
TSMH	why did early european explorers not travel to amazonia?
CGMH	see below, also : did any european settlers ever travel to build the " first north american sailing canoes "?
Keys	economic growth schooling
TSMH	how do economic growth rates in the united states make children receive high - quality schooling?
CGMH	what good is economic growth in comparison with being among the best in public schooling?

(1) Interrogative Sentences

Keys	seat
TSMH	please get up from your seat
CGMH	go on in and take your seat
Keys	careful
TSMH	please be so very very careful.
CGMH	and please be a very very careful
Keys	turn, lights
TSMH	turn on the lights all the time
CGMH	turn on near all the main lights
Keys	close, window
TSMH	stay close enough to the window
CGMH	stick close enough to meet the window
Keys	nice, weekend
TSMH	have yourself a very nice private weekend
CGMH	please be nice about spending the weekend

(2) Imperative Sentences

Table 7: Case study of generating interrogative and imperative sentences with keywords, where Keys stands for keywords.