A Study of Constrained Beam Search for Dialogue State Tracking

Mathieu Tuli *

Abstract

Dialogue State Tracking (DST) is a key component of Task-Oriented Dialogue. Recently, research has trended towards making DST more efficient and scalable using sequence-to-sequence (Seq2Seq) modelling, however this has removed reliable structure that is useful for engineered systems. In this work, I investigate the use of beam search coding of language models applied to DST, and attempt to reintroduce control through the use of constrained beam search. I perform an in-depth analysis of the performance of beam search on a pretrained Seq2Seq model on DST and design a new CBS method that takes advantage of the inherent structure of DST data.

1. Introduction

Task-Oriented Dialogue (TOD) systems are conversational systems that aim to assist users with specific tasks based on their perceived intent through interactive natural dialogue. A fundamental core to TOD systems is Dialogue State Tracking (DST); the task of tracking information (e.g. entities such as time, date, location) throughout turns in a dialogue (Young et al., 2013; Hosseini-Asl et al., 2020; Madotto, 2020). Normally, dialogue states will be seperated into a list of domain, slot, and value tuples. DST is then usually performed through the use of slot-filling architectures, where user utterances (or some context of dialogue) are encoded to then be used to perform multiclass-classification over all possible domain-slot pairs. An abstract visual is shown in Figure 1.

Although these methods are very beneficial for engineered systems, they rely on a pre-defined ontology/set of domain-slot pairs which makes them poorly scalable in a realistic setting. In efforts to improve this, language modelling (sequence-to-sequence (Seq2Seq) and causal language modelling (CLM)) has garnered a lot of attention as a more scalable/flexible alternative by directly translating dialogue context into a string-form of a dialogue state. Although this removes the need for a pre-defined ontology, there is a loss of structure and reliance on our model's unconstrained predictions that makes engineering a system difficult.

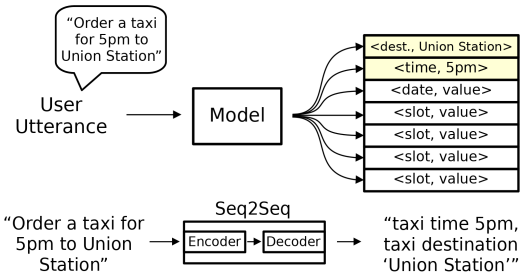


Figure 1. Abstract representation of slot-filling architecture (top) and sequence-to-sequence modelling for dialogue state tracking.

Given this, in this work I investigate how we might regain structure or (re) introduce control into language models. This goal is joint with the underlying desire to improve DST performance for an existing model. Specifically, I will be focusing on decoding a Seq2Seq model. I will be evaluating how the inherent structure of DST data can be exploited to constrain search through the use of Constrained Beam Search (CBS) (Hokamp & Liu, 2017; Post & Vilar, 2018; Anderson et al., 2016). In a classical sense, word select is treated as action selection and my aim is to better incentive certain actions over others based on the current state of knowledge in the decoding process. The application of CBS for text generation in the context of DST is new, so I will be defining my own method based on previous CBS works from neural image captioning. The contributions of this work are as following:

1. An evaluation of a pretrained language model (Seq2Seq) on a DST dataset for various beam widths.
2. An adapted constrained beam search method based on finite state machine (FSM) and grid beam search (Hokamp & Liu, 2017) for application in DST.
3. An investigation of CBS in application to DST using Seq2Seq modelling, evaluating the performance of various types of constraints on various beam widths.
4. An analysis of beam selection criteria.
5. An investigation into CBS applied to an open-domain language model (GPT-2).
6. An investigation of simple heuristics such as length normalization and coverage penalty (Wu et al., 2016) when applied to DST.

2. Related Works

Beam search and its application in language modelling has been extensively covered in many works (Wu et al., 2016; Cohen & Beck, 2019; Chorowski & Jaitly, 2016; Steinbiss et al., 1994). Although it is an "unpublished" method, it is a very popular method for decoding language models and generating more comprehensive outputs (Weng, 2021). Beam search is in fact an application of greedy breadth first search, where selection of actions becomes the selection of words based on the softmax probabilities over words.

Constrained beam search is a modification to beam search where constraints are applied to modify the selection criteria of words. Cohen & Beck (2019) showed that CBS can be used to resolve performance degradation over large beam widths and highlighted how simple heuristics can improve and stabilize performance. Traditionally , CBS has been used extensively in the task of neural image captioning. Anderson et al. (2016) employed the use of finite state machine (FSM) to validate beams throughout decoding, where validation meant the inclusion of certain tokens or token subsequences. In their method, a set of beams W are maintained for each state in the FSM, and beams may move between states of validation as they become validated, and only the top beam in the goal state is returned. In their work, only the presence of tokens *at some point* mattered. Hokamp & Liu (2017) introduced Grid Beam Search (GBS), a CBS method that extends beams by either generating tokens from the model or generating tokens specified by certain subsequence constraints and forcing certain tokens to be generated. Unlike the use of an FSM, there are only W beams total. Post & Vilar (2018) took this method further with Dynamic Beam Allocation (DBA), which introduced some computational efficiencies, but the core idea remained the same.

Constrained beam search is merely one class of heuristic in an attempt to improve beam search. Additionally, the method of constrained beam search (FSM, GBS, or DBA) that should be employed is problem specific, and not every method is applicable always. In addition to CBS, Ott et al. (2018) explored the notion performing vanilla beam search and filtering beam out after completion to weed out "unwanted" predictions (e.g. such as those with multiple repeated tokens). Further, Wu et al. (2016) showed that the use of length normalization and coverage penalty heuristics in beam search can also work to improve results. Specifically, length normalization attempts to account for the fact that beam search is biased towards shorter sequences, since the longer the sequence in the graph, the smaller the overall probability becomes. Coverage penalty attempts to force the beam search to attend to as much of the input as possible, in order to prevent the model from focusing too much in one area.

3. Methodology

3.1. Language Models

In this work I will be primarily focusing on a sequence-to-sequence (Seq2Seq) model. Specifically, I will be looking at the pointer generator network (See et al., 2017). This is a simple encoder-decoder LSTM with a copy-mechanism that allows it to toggle between generating a new token through its hidden state or copying one directly from the input. This model has been trained diretly on the dataset I will be considering. Note that I also tested a vanilla encoder-decoder transformer model, however its baseline performance was worse than this pointer-generator network so I opted not to use it.

In addition, there is an interest of mine to investigate to what degree constrained search can be used to tame a large causal language model that has not be trained at all on the data I am considering. Specifically, I am curious to see to what degree constrained search can guide the decoding of a powerful powerful language model like GPT-2. Therefore I also investigate CBS applied to a distilled version of GPT-2 (Radford et al., 2019) from the HuggingFace library (Wolf et al., 2020). This model has been trained on open-domain language and excels at predicting next tokens given the previous sequence of tokens.

3.1.1. Language Models Modelling Transition Probabilities

Both Seq2Seq models and CLMs – parameterized by θ – operate to produce a sequence of tokens $\hat{\mathbf{y}}$ such that $\hat{\mathbf{y}} = \arg\max_{\mathbf{y} \in Y} P_\theta(\mathbf{y}|\mathbf{x})$, where $\mathbf{x}$ is in the input sequence to the model and Y is the set of all sequences. Note that $\mathbf{y} = \{y_1, \ldots, y_K\}$ and $\mathbf{x} = \{x_1, \ldots, x_{K'}\}$ are sequences of tokens from our vocabulary $\mathcal{V}$, and K, K' are the length of the output and input sequences, respectively. In other words, I use these language models to model the probability of a certain output token (or sequence of tokens) given an observation sequence. Note that this modelling is naturally non-markovian. It is common to deal with the log probabilities, and as such our sequence probability can be factorized under the i.i.d. assumption to form

$$P_\theta(\mathbf{y}|\mathbf{x}) = \sum_{k=1}^{K} \log P_\theta(y_k|x; \{y_1, \ldots, y_{k-1}\}). \quad (1)$$

In this way, language models define the transition dynamics of our environment, where the environment is defined by whatever vocabularly there is.

3.2. Data

The dataset I will be considering is the MultiWOZ-2.1 (Eric et al., 2019) TOD dataset. Originally, I was also

going to consider the SMCalFlow (Andreas et al., 2020) dataset, however the process of developing constraints proved longer than expected, and I opted to cover different heuristics instead (see subsection 3.4). Further, I considered an extra model (GPT-2) instead of another dataset, which I found to be a more interesting and useful direction than simply applying similar techniques on a different domain.

Dialogue state in MultiWOZ-2.1, when used in a Seq2Seq or CLM framework, is represented as a list of comma-seperated tuples of information as "<*domain-name slot-name value*, ... >". Inputs to the model are composed of a concatenation of user utterances U_i and agent utterances A_i from previous turns in the dialogue. That is, inputs $= [U_1; A_1; \ldots; U_{i-1}; A_{i-1}; U_i]$. For the purposes of GPT-2, I modify the model inputs to also include the previous dialogue states D_i. That is, $= [U_1; A_1; D_1; \ldots; U_{i-1}; A_{i-1}; D_{i-1}; U_i]$. The hope here is that by providing a repeating pattern of user/agent utterances followed by dialogue states, GPT-2 should continue generating text in the sequence, where the next set of tokens should be the subsequent dialogue state. See Appendix A for some more details and examples on the dataset.

3.3. Beam Search

From Equation 1, it's straightfoward to see that a simple greedy search would simply pick the most probable token $y_k = \arg\max_{y \in \mathcal{V}} \log P_\theta(y|x; \{y_1, \ldots, y_{k-1}\})$ at each step k in the decoding process. Beam search however will maintain a *beam width* W of the W most probable output sequences at each step k. We can denote the set of beams from the previous timestep as $B_{k-1} = \{\mathbf{y}^1_{k-1}, \ldots, \mathbf{y}^W_{k-1}\}$. At each iteration in the decoding process, beam search selects the top W unique sequences from the set of all possible one token extensions of its beams. Formally, this set of extended beams is represented as $\mathcal{B}_k = \{y_k | \mathbf{y}_{k-1} \in B_{k-1} \wedge y_k \in \mathcal{V}\}$, and finally the update set of beams becomes

$$B_k = \underset{y_{1,k},\ldots,y_{W,k} \in \mathcal{B}_k}{\arg\max} \sum_{w \in [W]} \log P_\theta(y_{w,k}|x) \quad (2)$$
$$s.t.\ y_i \neq y_k \ \ \forall i \neq j; \ \ i, j \in [W]$$

Note that for some sequence $\mathbf{y}$ given input $\mathbf{x}$, I define the scoring function $s(\mathbf{y}, \mathbf{x})$ as

$$s(\mathbf{y}, \mathbf{x}) = \sum_{k=1}^{K} \log P_\theta(y_k|x; \{y_1, \ldots, y_{k-1}\}) \quad (3)$$

Beam search is concluded when either (a) one of the beams reaches the end of its sequence and no possible sequence extensions can result is a higher score or (b) the maximum sequence length is reached for each beam and the beam with the highest score is chosen. Note also that for $W = 1$, we recover the greedy search algorithm, and only select the next token with the highest probability.

3.4. Search Heuristics

In this section I explain the heuristics I will investigate.

3.4.1. Constrained Beam Search

The method of constrained beam search that I will employ differs from previous works, given the repeated structure inherent in dialogue states. Like previous methods (Anderson et al., 2016; Hokamp & Liu, 2017; Post & Vilar, 2018), the inclusion of certain tokens (e.g. domain names) is desired, however now, ordering of tokens is also required. The notion of order in previous methods is only present in GBS, when the beam search is constrained on a subsequence, and once the start token of the subsequence is generated, the remainder of the subsequence is forced to be generated. However, GBS itself does not define where the subsequence should be present nor when a subsequence constraint need be started. As such, I design a new constrained search method, combining both FSM and GBS. Specifically, similar to (Anderson et al., 2016), I utilize a FSM to determine the permitted subset of tokens that may be generated. In this sense, it acts as a controller in the search. Now, given the current state and set of constraint tokens for the current state, I can then employ a version of GBS that forces generation of those constraint tokens by masking out all but the permitted tokens and select the token with the highest probability. I visualize the three controllers (FSM) I designed in Figure 2.

The beam search algorithm is slightly altered, where we must account for the restricted vocabularly determined by the controller. Let $\mathcal{V}^S_k$ denote the vocabulary permitted by the FSM at any given time step k and state S. Now, the set of extended beams is represented as

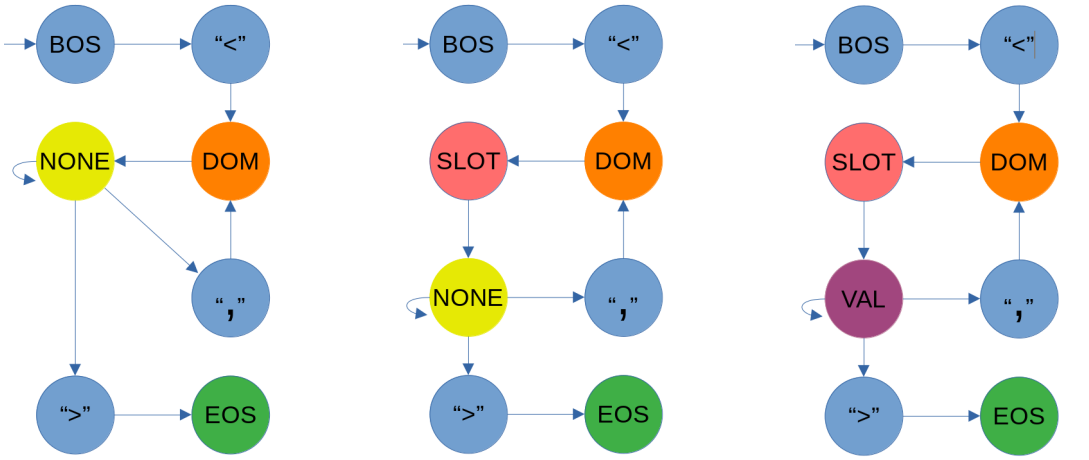
$$\mathcal{B}^S_k = \{y_k | \mathbf{y}_{k-1} \in B_{k-1} \wedge y_k \in \mathcal{V}^S_k\} \quad (4)$$

and

$$B_k = \underset{y_{1,k},\ldots,y_{W,k} \in \mathcal{B}^S_k}{\arg\max} \sum_{w \in [W]} \log P_\theta(y_{w,k}|x) \quad (5)$$
$$s.t.\ y_i \neq y_k \ \ \forall i \neq j; \ \ i, j \in [W]$$

Note that the permitted states are $S = \{[BOS], [EOS], [<], [>], [,], [DOM], [SLOT], [VAL], [NONE]\}$.

Constraint levels: I design three levels of constraints that I will investigate. The first level is the *domain level*, where I ensure that all domain predictions are restricted to the permitted domain tokens (see Appendix A). This controller is shown in Figure 2a. The second level is the *domain-slot*



(a) Domain constraint (b) Domain-slot constraint (c) Domain-slot-value constraint

Figure 2. The controllers used to guide decoding for MultiWOZ-2.1. (a) shows the domain-level constraints, (b) the domain-slot-level constraints, and (c) the domain-slot-value-level constraints. Note that depending on the dataset, you could have different control states the define the specific structure of the dialogue state representation. At each point in the decoding, the controller determines the permitted tokens to be generated, with *DOM, SLOT,* and *VAL* representing the set of domain-names, slot-names, and values respectively, and *NONE* meaning no restrictions. The other states identify the single token permitted (comma, BOS token, etc.). Note that I experiment with varying sets of values, from user-restricted tokens to anything in the input context. Because the value of a slot may have multiple tokens, the *VAL* state is self-looping. The same goes for the *NONE* state. Transitions are determined by Equations 4 & 5. The current state can be identified based on the previous token in the beam. All sequences start at the BOS state.

level, where in addition to ensuring the all domains predictions are restricted, the subsequent slot prediction is also restricted and conditioned on the previous domain prediction, and only permitted domain-slot pairs will be generated in sequence. This controller is shown in Figure 2b. Finally, the last level is the *domain-slot-value* level, where I constrain not only domain-slot pairs, but the value of information as well. Specifically, I restrict the permitted tokens for value-level predictions to be only tokens present in the input. This includes both agent utterances and user utterances. Originally, I had planned only to include user utterances, as the model often mixes up information between the two, however this resulted in poor performance (see Table 1). The reason is that often, the agent will pose a confirmation question, and the user will respond with a simple "yes" or "no", and hence the information of that confirmation is lost if we only consider user utterances. Note also that MultiWOZ-2.1 also has some typos and generalizes certain entities (e.g. confirmations are represented as "yes" even if the person says "sure"). Therefore, I also include the typo-tokens and generalized entities to account for this. This controller is shown Figure 2c. Note finally that in each level, commas and surrounding brackets are constrained as well.

Computational efficiency. This method introduces very little computatinoal overhead compared to vanilla beam search. The additional computational tasks for the k-th step in the decoding include (a) bookkeeping of the permitted set of tokens for each FSM state, (b) softmax masking to constrain what tokens can be predicted, and (c) controller behaviour. For bookkeeping, the additional memory cost is $\mathcal{O}(V_k^S)$, which is usually in the range of 100 tokens, which is a minimal cost considering normal vocabularies themselves range from a few hundred to thousands of tokens. For softmax masking, this process involves a single tensor creation operation and one multiplication, which is a $\mathcal{O}(1)$ operation. Finally for the controller behaviour, this is a $\mathcal{O}(1)$ level operation, which involves remembering (indexing) or identifying the current state and transitioning to the next one. I do not include generation times as the results vary based on the hardware I used and the number of different experiments I ran at once. I leave it as future work to empirically analyze this analysis.

3.4.2. Beam Validation

A naive heuristic I will investigate involves running beam search (as defined in subsection 3.3) and selecting the beam with the highest score that satisfies the constraints defined in subsubsection 3.4.1. In the case where no beams meet the constraints, I will select the beam that satisfies the most constraints. In the case where multiple beams statisfy the same number of constraints, I will simply select the highest scoring one. This is a simple heuristic that does not improve search but might help prune poor solutions.

3.4.3. Length Normalization and Coverage Penalty

Length normalization. Length normalization has been shown to improve the resulting generated sequences in beam search (Wu et al., 2016) and has been commonly employed to level the playing field for longer sequences, which language models tend to avoid. Formally, In the case of DST, length of sequences could be problematic when the model decides to terminate before it has fully translated each tuple of information. In certain cases, similar tokens might also be favoured in the value position over others, even though they ultimately convey different things. For example, for certain beam searches of $W > 1$, it is sometimes common for a sequence containing *..., domain-name slot-name "dontcare", ...* to be selected with higher score over the slightly different but correct sequence of *..., domain-name slot-name "not mentioned", ...*, since *"not mentioned"* is in fact two tokens, meaning the overall score decreases slightly. The goal here is to see whether length normalization might correct for such scenarios.

Coverage penalty. Similarly to length normalization, Wu et al. (2016) showed that coverage penalty can also be used to improve in translation. Therefore I will also investigate the effect of coverage penalty on final accuracy. The idea here being that similar issues to the one just mentioned – where *"dontcare"* was favoured over *"not mentioned"* – might also get resolved with greater coverage/attention to the input.

Length normalization and coverage penalty are formally defined as follows. For the scoring function $s(\mathbf{y}, \mathbf{x})$ as defined in Equation 3, I now recompute score as

$$
\begin{aligned}
s(\mathbf{y}, \mathbf{x}) &= \frac{\sum_{k=1}^{K} \log P_\theta(y_k | x; \{y_1, \ldots, y_{k-1}\})}{LN(\mathbf{y}) + CP(\mathbf{y})} \\
LN(\mathbf{y}) &= \frac{(5 + |\mathbf{y}|)^\alpha}{(5+1)^\alpha} \\
CP(\mathbf{y}) &= \beta \sum_{i}^{|\mathbf{x}|} \log \left[\min(\sum_{j}^{|\mathbf{y}|} p_{i,j}, 1.0) \right]
\end{aligned}
\tag{6}
$$

where $p_{i,j}$ is the attention probability on the i-th token from the input sequence $\mathbf{x}$ to the j-th token in the target sequence $\mathbf{y}$. Note that α and β therefore tune the amount of length normalization and coverage penalty, respectively. See (Wu et al., 2016) for more detail. Note also that when $\beta = \alpha = 0$ the scoring function from Equation 3 is recovered.

3.5. Evaluation

In DST, evaluation is done using the Joint Goal Accuracy (JGA). Turn-level JGA is the percentage of turns that our model has correctly identified and filled the proper slots, independent of order. Formally, I denote the order-independent matching of slots between the predicted state sequence $\mathbf{y}$ and ground truth $\mathbf{y}^{gt}$ as $\mathbf{y} \overset{*}{=} \mathbf{y}^{gt}$. Hence,

$$
\text{JGA-turn } = \frac{1}{N} \sum_{i}^{N} \mathbb{I}[\mathbf{y}_i \overset{*}{=} \mathbf{y}_i^{gt}]
$$

where $\mathbb{I}[\cdot] = 1$ if the inner condition holds, otherwise $\mathbb{I}[\cdot] = 0$.

Note that I also define the notion of *JGA overlap*, which is a softer criterion. In short, it is the percentage of correct domain-slot-value tuples for a single turn. A formal definition isn't really necessary here, and note that it is used as an alternative scoring metric, where each beam in the search will be scored based on their JGA-ovelap with the ground truth. This will be used to investigate the effectiveness of the basic log-likelihood scoring method (Equation 3). In effect though, what I really care about is that this score will be a maximum 1.0 if all tuples in the prediction overlap with the tuples in the ground truth, and using this criterion to score beams will return that beam, if it exists.

4. Experiments

4.1. Setup

I perform the following studies:

Investigating constrained beam search. The primary investigation looks at constrained beam search vs. vanilla beam search. In this larger study, I consider beam widths of $W \in \{1, 2, 3, 5, 10, 25, 100, 250\}$. Following that, I perform the following:

- a baseline using vanilla beam search and final beam selected using their scores as defined in 3.
- a baseline using vanilla beam search and final beam selected based on JGA-overlap with the ground truth. Naturally, such a score is not possible in reality, as the ground truths are not present, however this provides a theoretical/empirical upper bound on the best possible result that my model can get if it perfectly selected the proper beam. This will be useful for identifying errors in sequence selection.
- As another "baseline", I also perform a constrained beam search using the ground truth output sequence. Specifically, for each turn, I constrain the domain, slots, and value of information based on the ground truth output sequence. This constraint baseline provides another performance baseline for me to strive for.
- I also run the above ground truth constrained beam search, but select beams based on JGA overlap.

Method	$W1$	$W2$	$W3$	$W5$	$W10$	$W25$	$W100$	$W250$
Baseline	46.65	47.14	47.19	47.22	47.23	47.23	47.23	47.23
CBS w/ GT	55.97	55.09	54.79	54.94	55.03	55.04	55.04	55.04
Baseline w/ JGA-overlap	55.97	57.51	58.51	67.27	63.24	64.02	65.43	63.15
CBS w/ GT w/ JGA-overlap	55.97	65.31	68.51	71.75	69.28	68.75	68.78	68.78
Baseline w/ validation	46.65	47.14	47.19	47.22	47.23	47.23	47.23	47.23
CBS (domain-level)	46.65	47.14	47.19	47.22	47.23	47.23	47.23	47.23
CBS (domain-slot level)	46.65	47.14	47.19	47.22	47.23	47.23	47.23	47.23
CBS (domain-slot-value level)	46.65	47.14	47.19	47.22	47.23	47.23	47.23	47.23
CBS (domain-slot-user level)	32.30	32.70	33.56	34.89	33.98	33.98	33.98	33.98
CBS (all at once)	46.65	47.14	47.19	47.22	47.23	47.23	47.23	47.23

Table 1. Performance (JGA-turn %) results of each experiment, evaluated using JGA-turn and reported for number beam widths W. Methods that indicate "w/ JGA-overlap" indicate those where beam selection was done using the JGA-overlap with the ground truth, and the "CBS w/ GT" methods are those where the ground truth tokens are used in the CBS. Note that "CBS (domain-slot-user level)" represents the results for constraining the search using the domain, slots, and user utterance tokens (not any agent tokens). "CBS (all at once)" represents the results for the constraint search without the controller and only limiting the vocabulary.

- I perform a naive beam validation, pruning beams that do not statisfy my constraints. Constraints here defined by the domain-slot level constraints.
- Finally, I consider constrained beam search as described in subsubsection 3.4.1. To reiterate, I consider three levels of constraints: domain constraints, domain-slot constraints, domain-slot-value constraints.

Effectiveness of the controller. I also analyze the effectiveness of using the FSM controller. In particular, rather than using the controller to determine which set of tokens to constrain on, I simply aggregate all the domain, slots, and possible value tokens together and always return that set of tokens. I consider beam widths of $W \in \{1, 2, 3, 5, 10, 25, 100250\}$.

Restricted domain analysis. In this approach, I consider a theoretical scenario where a priori knowledge is known about the domain we are operating in. I noticed that sometimes, the selected beam's domains have little to no relation to what the input sequence was refering to. This means that the model made an initial error in the domain prediction, which was carried forward through in the generation process. Because of this, it highlights the possibility of using an initial domain classification step to re-weight the importance of certain domain tokens. To verify this hypothesis, I consider a subset of the dataset where only the *restaurant* domain is present in the dialogue state. I then compare normal beam search against a constrained search where the domain is constrained to only predict the "restaurant" token and associated slots. I put no constraints on the information tokens. I consider beam widths of $W \in \{1, 2, 3, 5, 10\}$.

Length normalization and coverage penalty. In this approach, I consider length normalization and coverage penalties for all permutations of $\alpha, \beta \in \{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$. I consider a beam width of $W = 5$. I perform a baseline using no constraints, and finally compare against the domain-slot-value constrained beam search. Beams are selected by their scores.

GPT-2 constrained beam search. In this investigation, I consider the distilled GPT-2 model trained on open-domain text. I consider a baseline beam search and constrained beam search with beam width $W = 5$. For this approach, I will consider the domain-slot-value level constraints.

4.2. Results

Baseline vs. ground truth. We can see from Table 1 that constraining search using the ground truth tokens greatly improves the accuracy, achieving $\sim 8\%$ better JGA-turn accuracy (when using the normal beam selection scores). This is promising, since it highlights how a *properly* constrained search can result in a significant boost in performance. Further, we see from the methods that used JGA-overlap scoring that beam selection using the standard log-likelihood method does not produce the most optimal results. This is even more promising and opens the door boosting performance further if more intelligent beam selection is made. One concern however is that considering this is a constrained search using the ground truth, the performance isn't as high as I expected, which highlights how designing/extracting good token constraints at runtime is quite challenging, and possibly impossible.

Validating beams in a vanilla search. We can also see that the additional step beam of validation does not affect final results in any way. This highlights two key important facts: (a) that the model is already very good at predicting the proper structure and domain-slot pairs; and (b) that any performance improvements will most likely arise from better searching during the information value phase of text generation. This in turn tells us that most likely, the domain-slot level constraints will have ltittle to no effect.

	α						
β	-	**0.0**	**0.2**	**0.4**	**0.6**	**0.6**	**1.0**
0.0	**47.22**	33.65	29.12	26.67	25.03	23.41	
0.2	47.21	31.10	26.95	24.41	22.58	21.21	
0.4	47.21	28.84	24.64	21.98	20.63	19.46	
0.6	47.14	26.67	22.38	20.18	18.96	18.04	
0.8	47.15	24.45	20.29	18.77	17.47	16.56	
1.0	47.15	22.25	18.85	17.25	16.02	15.27	

	α						
β	-	**0.0**	**0.2**	**0.4**	**0.6**	**0.6**	**1.0**
0.0	**47.22**	33.65	29.12	26.67	25.03	23.41	
0.2	47.21	31.10	26.95	24.41	22.58	21.21	
0.4	47.21	28.84	24.64	21.98	20.63	19.46	
0.6	47.14	26.67	22.38	20.18	18.96	18.04	
0.8	47.15	24.45	20.29	18.77	17.47	16.56	
1.0	47.15	22.25	18.85	17.25	16.02	15.27	

Table 2. Performance results for the length normalized (α) and coverage penaltt (β) heuristics, for the baseline (left) and CBS with domain-slot-value level constraints (right). Results are reported using JGA-turn (%) and a beam width of $W = 5$.

Baseline vs. CBS. We can see from Table 1 that unfortunately, the constrained beam search has no effect on the final JGA-turn accuracy. This is true for all levels of constraints. This is a surprising result, more so for the domain-slot-value constraint level, and particularly given the performance gain when constraining using the ground truth tokens. However when analyzing and comparing generated outputs between the CBS and baseline, it becomes clear that despite the token constraints, the baseline model is already trained well enough such that its predictions are sufficiently constrained in their softmax. It seems that despite ensuring heuristically which tokens should be considered, the model was already considering those tokens in the first place. The obvious question then becomes "what token constraints are required?". If token constraints based on the user and agent utterances are still too broad, how might we be able to improve those constraints without using any form of ground truth? These questions are not simple to answer, nor can I see any immediate solutions, however there are a few future directions that I discuss in section 5.

Effectiveness of the controller. We can see from Table 1 that the controller does not affect the results, and it is simply sufficient to only constrain the vocabularly. Given previous discussions, this is not that surprising, but it shows that what really matters is the quality of token constraints.

Something positive. Despite the failure of results, it is still beneficial to know that constrained decoding does not perform *worse* that the baselines. In this manner, we can at least maintain performance of our model all while ensuring control of the generated output, which was an initial goal.

Domain restricted. We can see from Table 3 that a priori knowledge that can be used to form runtime constraints does optimize the search process and result in better performance. This is encouraging, and highlights how extracting the proper domains at runtime from the user utterances could help form better constraints. I can envision a naive method of restricting our domain tokens by looking for keywords (i.e. the user mentions the token "restaurant") in the input sequence, however this would quickly break down when the user utters a restaurant by name, in which the restaurant domain is implied. Perhaps the use of a database could be used to determine to what domain certain entities are linked to, but even this is brittle, and an additional domain classification model that classifies the set of domains an utterance is referring to might be a better option. This experiment also highlights that the controller is in fact important, where without it, there is a chance that undesired domain names or slots can be said by the user in a different context, removing our level of control and potentially harming performance.

Method	$W1$	$W3$	$W5$	$W10$
Baseline	64.25	64.56	64.47	64.88
CBS (domain level)	**64.46**	**65.31**	**65.96**	**65.96**

Table 3. Performance (JGA-turn%) results of the restaurant domain-restricted experiments. The baseline is normal beam search, and the CBS (domain level) is the constrained search where the model is restricted to only the restaurant token.

Length normalization and coverage penalty. We can see from Table 2 that the effect of length normalization and coverage penalty is only harmful, and in particular, it seems that length normalization is really harmful. It seems that despite my hopes, length normalization cannot resolve prediction errors such as "dontcare" instead of "not mentioned" as previously discussed, nor can coverage penalty. It appears for the metrics such as JGA-turn, these penalties only reduce the model's ability to attend to the input and generate proper-lengthed sequences. For a pre-trained model like the one I use here, it seems better to let the model's predictive power dictate sequence length.

Constraining GPT-2. The GPT-2 experiment proved to be a failure as well. I show some example generations using the CBS method in the Appendix B. Although my method is able to generate some more structured and coherent outputs as compared to the unconstrained model, it fails to know when to terminate, and also when to generate a comma and proceed with the next dialogue state tuple. It often simply gets stuck and repeatedly generates value information without predicting a comma to continue the generation phase. Even attempts at fixing this behaviour (e.g. predict comma every 5 tokens or so) proved unsuccessful. This is not that surprising, and a solution is not immediately apparent to me. Perhaps better design of the model inputs is required here.

5. Discussion and Future Work

General notes on final results. Despite the overwhelmingly negative results of this empirical analysis, it highlights some important features of these language models in application to dialogue state tracking. The first is that clearly, a pre-trained language model will be required for any engineered system; the notion of taming an open-domain language model seems too far-fetched given my initial investigation. This raises the question then of to what degree Seq2Seq modelling is really better than the traditional slot-filling architecture. If we have to train the model anyways, why not stick to the model that we know fits well with engineered systems? That being said, Seq2Seq modelling, with the advent of transformers, has been performing better out-of-the-box than slot-filling architectures, so perhaps further work on large Seq2Seq Transformers is required. The second feature is that even with perfect token constraints, the model has a relatively small performance gain available to it. This is important as the effort that might go into designing/extracting constraints for generation might be better spent on potentially more important things like model design. Finally, it's also clear that the log-probability based scoring method is suboptimal for beam selection. Unfortunately, JGA requires a ground truth for evaluation. We would need some way of evaluating something like logical consistency in order to maybe perform more optimal beam selection.

Better constraints. In continuation with my last point, it seems that one type of constraint that could prove useful is that of logical consistency. To use the running example of the "dontcare" token being predicted over the "not mentioned", I found that beams selected by the normal log-probability score vs. beams selected by the JGA-overlap are often logically inconsistent. Although they might hold similar probabilities, they represent fundamentally different logical truths about our state of knowledge, which isn't captured by the log-likelihood of tokens. Perhaps this is where we could use abstract meaning representation, similar to ideas proposed in (Cohen & Beck, 2019). I'm not immediately confident as to how to combine these ideas yet, but using some model or different representation that can validate logic between our inputs and outputs could potentially boost performance significantly.

Temporally extended search? In a final blue sky idea, an initial idea that floated around when I first considered this direction was how might temporally extended goals be included to guide the decoding of this language model over multiple turns or dialogues. After having completed this analysis, given how a priori knowledge of the domains was successful in improving search, it seems that higher level goals or constraints (something of the form of linear temporal logic) could prove useful. The key difficulty, which has been highlighted to me by this work, is in extracting those higher level goals to begin with. It seems that if it were easy, the model should inherently identify such goals, which for the most part it seems to do. That being said, if we were operating in a pre-defined domain where we knew a priori certain facts about the world or desired behaviours, then I believe that, to a certain extent, constrained beam search, even at the token level, can be beneficial.

6. Conclusion

In this work I investigated the use of constrained beam search on sequence-to-sequence language modelling of dialogue state tracking. I analyzed the effect of various heuristics in an attempt to find more optimal search methods but ultimately could not. I showed that at the very least, constrained search does not reduce performance, which is beneficial from an engineering perspective. In situations, however, where a priori knowledge exists, such as which specific domains are being discussed, then I showed that constrained beam search can be beneficial. Ultimately, the use of token-level constraints is not sufficient to gain reasonable performance boosts out these models, and an investigation into other types of constraints is required.

References

Anderson, P., Fernando, B., Johnson, M., and Gould, S. Guided open vocabulary image captioning with constrained beam search. *arXiv preprint arXiv:1612.00576*, 2016.

Andreas, J., Bufe, J., Burkett, D., Chen, C., Clausman, J., Crawford, J., Crim, K., DeLoach, J., Dorner, L., Eisner, J., et al. Task-oriented dialogue as dataflow synthesis. *Transactions of the Association for Computational Linguistics*, 8:556–571, 2020.

Chorowski, J. and Jaitly, N. Towards better decoding and language model integration in sequence to sequence models. *arXiv preprint arXiv:1612.02695*, 2016.

Cohen, E. and Beck, C. Empirical analysis of beam search performance degradation in neural sequence models. In *International Conference on Machine Learning*, pp. 1290–1299. PMLR, 2019.

Eric, M., Goel, R., Paul, S., Kumar, A., Sethi, A., Ku, P., Goyal, A. K., Agarwal, S., Gao, S., and Hakkani-Tur, D. Multiwoz 2.1: A consolidated multi-domain dialogue dataset with state corrections and state tracking baselines. *arXiv preprint arXiv:1907.01669*, 2019.

Hokamp, C. and Liu, Q. Lexically constrained decoding for sequence generation using grid beam search. *arXiv preprint arXiv:1704.07138*, 2017.

Hosseini-Asl, E., McCann, B., Wu, C.-S., Yavuz, S., and Socher, R. A simple language model for task-oriented dialogue. *arXiv preprint arXiv:2005.00796*, 2020.

Madotto, A. Language models as few-shot learner for task-oriented dialogue systems. *arXiv preprint arXiv:2008.06239*, 2020.

Ott, M., Auli, M., Grangier, D., and Ranzato, M. Analyzing uncertainty in neural machine translation. In *International Conference on Machine Learning*, pp. 3956–3965. PMLR, 2018.

Post, M. and Vilar, D. Fast lexically constrained decoding with dynamic beam allocation for neural machine translation. *arXiv preprint arXiv:1804.06609*, 2018.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

See, A., Liu, P. J., and Manning, C. D. Get to the point: Summarization with pointer-generator networks, 2017.

Steinbiss, V., Tran, B.-H., and Ney, H. Improvements in beam search. In *Third International Conference on Spoken Language Processing*, 1994.

Weng, L. Controllable neural text generation. *lilianweng.github.io/lil-log*, 2021. URL https://lilianweng.github.io/lil-log/2021/01/02/controllable-neural-text-generation.html.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. Huggingface's transformers: State-of-the-art natural language processing, 2020.

Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Young, S., Gašić, M., Thomson, B., and Williams, J. D. Pomdp-based statistical spoken dialog systems: A review. *Proceedings of the IEEE*, 101(5):1160–1179, 2013.

A. Data

Here are some additional details on the dataset, and further the set of constraint tokens for domain and domain-slot level constraints. The set of *domain-name* is {*attraction, hospital, hotel, restaurant, taxi, train*} and *slot-name* is {*area, arriveby, book, day, department, departure, destination, food, internet, leaveat, name, parking, pricerange, stars, type*}. Further, the set of domain-slot-names are {*attraction area, attraction name, attraction type, hospital department, hotel area, hotel book, hotel internet, hotel name, hotel parking, hotel pricerange, hotel stars, hotel type, restaurant area, restaurant book, restaurant food, restaurant name, restaurant pricerange, taxi arriveby, taxi departure, taxi destination, taxi leaveat, train arriveby, train book, train day, train departure, train destination, train leaveat*}.

Table 4. Examples from the MultiWOZ-2.1 dataset.

Input Context (C_1)	__User am looking for a place to to stay that has cheap price range it should be in a type of hotel __Belief
Output Dialogue State (B_1)	< hotel name not mentioned , hotel area not mentioned , hotel parking not mentioned , hotel pricerange cheap , hotel stars not mentioned , hotel internet not mentioned , hotel type hotel >
Input Context (C_{160})	__User whoa whoa , easy there tiger , lets narrow the search down first . in the center , 0 stars , cheap and it can be a hotel or guesthouse . __Agent how about el shaddai ? fits your request perfectly . __User is el shaddai a guest house or hotel ? __Belief
Output Dialogue State (B_{160})	< restaurant food seafood , restaurant pricerange not mentioned , restaurant name not mentioned , restaurant area centre , hotel name el shaddia guesthouse , hotel area centre , hotel parking not mentioned , hotel pricerange cheap , hotel stars 0 , hotel internet not mentioned , hotel type dontcare , train leaveat not mentioned , train destination cambridge , train day wednesday , train arriveby 08:00 , train departure stevenage , train book people 7 >

B. GPT-2 Results

GPT-2 Results.

Table 5. Examples from GPT-2 generation.

Input Sequence	__User i need to book a hotel in the east that has 4 stars . __Belief ¡ hotel name acorn guest house , hotel area north , hotel parking yes , hotel pricerange moderate , hotel stars 4 , hotel internet not mentioned , hotel type not mentioned ¿ __Agent i can help you with that . what is your price range ? __User that does not matter as long as it has free wifi and parking . __Belief
Output Sequence	< taxi leaveat guest guest, taxi destination guest guest, taxi arrive by guest, taxi 23:08 guest ...
Input Sequence	__User i need to book a hotel in the east that has 4 stars . __Belief ¡ hotel name acorn guest house , hotel area north , hotel parking yes , hotel pricerange moderate , hotel stars 4 , hotel internet not mentioned , hotel type not mentioned ¿ __Agent i can help you with that . what is your price range ? __User that does not matter as long as it has free wifi and parking . __Belief
Output Sequence	18:40 ...