Fast Lexically Constrained Decoding with Dynamic Beam Allocation for Neural Machine Translation

Matt Post and **David Vilar**
Amazon Research
Berlin, Germany

Abstract

The end-to-end nature of neural machine translation (NMT) removes many ways of manually guiding the translation process that were available in older paradigms. Recent work, however, has introduced a new capability: *lexically constrained* or *guided* decoding, a modification to beam search that forces the inclusion of pre-specified words and phrases in the output. However, while theoretically sound, existing approaches have computational complexities that are either linear (Hokamp and Liu, 2017) or exponential (Anderson et al., 2017) in the number of constraints. We present an algorithm for lexically constrained decoding with a complexity of $\mathcal{O}(1)$ in the number of constraints. We demonstrate the algorithm's remarkable ability to properly place these constraints, and use it to explore the shaky relationship between model and BLEU scores. Our implementation is available as part of SOCKEYE.

1 Introduction

One appeal of the phrase-based statistical approach to machine translation (Koehn et al., 2003) was that it provided control over system output. For example, it was relatively easy to incorporate domain-specific dictionaries, or to force a translation choice for certain words. These kinds of interventions were useful in a range of settings, including interactive machine translation or domain adaptation. In the new paradigm of neural machine translation (NMT), these kinds of manual interventions are much more difficult, and a lot of time has been spent investigating how to restore them (cf. Arthur et al. (2016)).

At the same time, NMT has also provided new capabilities. One interesting recent innovation is *lexically constrained decoding*, a modification to beam search that allows the user to specify words and phrases that must appear in the system output (Figure 1). Two algorithms have been proposed for this: *grid beam search* (Hokamp and Liu, 2017, GBS) and *constrained beam search* (Anderson et al., 2017, CBS). These papers showed that these algorithms do a good job automatically placing constraints and improving results in tasks such as simulated post-editing, domain adaptation, and caption generation.

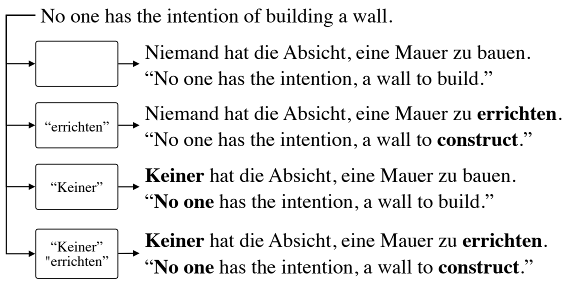
No one has the intention of building a wall.
Niemand hat die Absicht, eine Mauer zu bauen.
"No one has the intention, a wall to build."
"errichten"
Niemand hat die Absicht, eine Mauer zu **errichten**.
"No one has the intention, a wall to **construct**."
"Keiner"
Keiner hat die Absicht, eine Mauer zu bauen.
"**No one** has the intention, a wall to build."
"Keiner" "errichten"
Keiner hat die Absicht, eine Mauer zu **errichten**.
"**No one** has the intention, a wall to **construct**."

Figure 1: An example translating from English to German. The first translation is unconstrained, whereas the remaining ones have one or two constraints imposed. A word-for-word translation of the German output has been provided for the convenience of non-German speaking readers.

A downside to these algorithms is their runtime complexity: linear (GBS) or exponential (CBS) in the number of constraints. Neither paper reported decoding speeds, but the complexities alone suggest a large penalty in runtime. Beyond this, other factors of these approaches (a variable sized beam, finite-state machinery) change the decoding procedure such that it is difficult to integrate with other operations known to increase throughput, like batch decoding.

We propose and evaluate a new algorithm, *dynamic beam allocation* (DBA), that is *constant* in the number of provided constraints (Table 1). Our algorithm works by grouping together hypotheses that have met the same number of constraints into

work	complexity
Anderson et al. (2017)	$\mathcal{O}(Nk2^C)$
Hokamp and Liu (2017)	$\mathcal{O}(NkC)$
This work	$\mathcal{O}(Nk)$

Table 1: Complexity of decoding (sentence length N, beam size k, and constraint count C) with target-side constraints under various approaches.

banks (similar in spirit to the grouping of hypotheses into stacks for phrase-based decoding (Koehn et al., 2003)) and dynamically dividing a fixed-size beam across these banks at each time step. As a result, the algorithm scales easily to large constraint sets that can be created when words and phrases are expanded, for example, by sub-word processing such as BPE (Sennrich et al., 2016). We compare it to GBS and demonstrate empirically that it is significantly faster, making constrained decoding with an arbitrary number of constraints feasible with GPU-based inference. We also use the algorithm to study beam search interactions between model and metric scores, beam size, and pruning.

2 Beam Search and Grid Beam Search

Inference in statistical machine translation seeks to find the output sequence, $\hat{y}$, that maximizes the probability of a function parameterized by a model, θ, and an input sequence, x:

$$\hat{y} = \text{argmax}_{y \in \mathcal{Y}}\, p_\theta(y \mid x)$$

The space of possible translations, $\mathcal{Y}$, is the set of all sequences of words in the target language vocabulary, V_T. It is impossible to explore this entire space. Models decompose this problem into a sequence of time steps, t. At each time step, the model produces a distribution over V_T. The simplest approach to translation is therefore to run the steps of the decoder, choosing the most-probable token at each step, until either the end-of-sentence token, $\langle/\text{s}\rangle$, is generated, or some maximum output length is reached. An alternative, which explores a slightly larger portion of the search space, is beam search.

In beam search (Lowerre, 1976; Sutskever et al., 2014), the decoder maintains a *beam* of size k containing a set of active hypotheses (Algorithm 1). At each time step t, the decoder model is used to produce a distribution over the target-language vocabulary, V_T, for each of these hypotheses. This produces a large matrix of dimensions $k \times |V_T|$, that can be computed quickly with modern GPU hardware. Conceptually, a (row, column) entry (i, j) in this matrix contains the state obtained from starting from the ith state in the beam and generating the target word corresponding to the jth word of V_T. The beam for the next time step is filled by taking the states corresponding to the k-best items from this entire matrix and sorting them.

Algorithm 1 Beam search. *Inputs:* max output length N, beam size k. *Output:* highest-scoring hypothesis.

```
1: function BEAM-SEARCH($N$, $k$)
2:     beam ← DECODER-INIT($k$)
3:     for time step $t$ in 1..$N$ do
4:         scores = DECODER-STEP(beam)
5:         beam ← KBEST(scores)
6:     return beam[0]
7: function KBEST(scores)
8:     beam = ARGMAX_K($k$, scores)
9:     return beam
```

A principal difference between beam search for phrase-based and neural MT is that in NMT, there is no recombination: each hypothesis represents a complete history, back to the first word generated. This makes it easy to record properties of the history of each hypothesis that were not possible with dynamic programming. Hokamp and Liu (2017) introduced an algorithm for forcing certain words to appear in the output called *grid beam search* (GBS). This algorithm takes a set of constraints, which are words that must appear in the output, and ensures that hypotheses have met all these constraints before they can be considered to be completed. For C constraints, this is accomplished by maintaining $C + 1$ separate beams or *banks*, $B_0, B_1, \ldots, B_C$, where B_i groups together hypotheses that have generated (or *met*) i of the constraints. Decoding proceeds as with standard beam decoding, but with the addition of bookkeeping that tracks the number of constraints met by each hypothesis, and ensures that new candidates are generated, such that each bank is filled at each time step. When beam search is complete, the hypothesis returned is the highest-scoring one in bank B_C. Conceptually, this can be thought of as adding an additional dimension to the beam, since we multiply out some base beam size b by (one plus) the number of constraints.

We note two problems with GBS:

- Decoding complexity is linear in the number of constraints: The effective beam size, $k \cdot (C+1)$, varies with the number of constraints.
- It is impractical. The beam size changes for every sentence, whereas most decoders specify the beam size at model load time in order to optimize computation graphs, specially when running on GPUs. It also complicates beam search optimizations that increase throughput, such as batching.

Our extension, fast lexically-constrained decoding via dynamic beam allocation (DBA), addresses both of these issues. Instead of maintaining $C+1$ beams, we maintain a single beam of size k, as with unconstrained decoding. We then dynamically allocate the slots of this beam across the constraint banks at each time step. There is still bookkeeping overhead, but this cost is constant in the number of constraints, instead of linear. The result is a practical algorithm for incorporating arbitrary target-side constraints that fits within the standard beam-decoding paradigm.

3 Dynamic Beam Allocation (DBA)

Our algorithm (Algorithm 2) is based on a small but important alteration to GBS. Instead of *multiplying* the beam by the number of constraints, we *divide*. A fixed beam size is therefore provided to the decoder, just as in standard beam search. As different sentences are processed with differing numbers of constraints, the beam is dynamically allocated to these different banks. In fact, the allocation varies not just by sentence, but across time steps in processing each individual sentence.

We need to introduce some terminology. A *word constraint* provided to the decoder is a single token in the target language vocabulary. A *phrasal constraint* is a sequence of two or more contiguous tokens. Phrasal constraints come into play when the user specifies a multi-word phrase directly (e.g., *high-ranking member*), or when a word gets broken up by subword splitting (e.g., *thou@@ ghtful*). The total number of constraints is the sum of the number of tokens across all word and phrasal constraints. It is easier for the decoder to place multiple sequential tokens in a phrasal constraint (where the permutation is fixed) compared to placing separate, independent constraints (see discussion at the end of §5), but the algorithm does not distinguish them when counting.

DBA fits nicely within standard beam decoding; we simply replace the `kbest` implementation from Algorithm 1 with one that involves a bit more bookkeeping. Instead of selecting the top-k items from the $k \times V_T$ scores matrix, the new algorithm must consider two important matters.

1. Generating a list of candidates (§3.1). Whereas the baseline beam search simply takes the top-k items from the scores matrix (a fast operation on a GPU), we now need to ensure that candidates progress through the set of provided constraints.
2. Allocating the beam across the constraint banks (§3.2). With a fixed-sized beam and an arbitrary number of constraints, we need to find an allocation strategy for dividing the beam across the constraint banks.

3.1 Generating the candidate set

We refer to Figure 2 for discussion of the algorithm. The set of candidates for the beam at time step $t+1$ is generated from the hypotheses in the current beam at step t, which are sorted in decreasing order, with the highest-scoring hypothesis at position 1. The DECODER-STEP function of beam search generates a matrix, *scores*, where each row r corresponds to a probability distribution over all target words, expanding the hypothesis in position r in the beam. We build a set of candidates from the following items:

1. The best k tokens across all rows of *scores* (i.e., normal top-k);
2. for each hypothesis in the beam, all unmet constraints (to ensure progress through the constraints); and
3. for each hypothesis in the beam, the single-best token (to ensure consideration of partially-completed hypotheses).

Each of these candidates is denoted by its coordinates in `scores`. The result is a set of candidates which can be grouped into banks according to how many constraints they have met, and then sorted within those banks. The new beam for timestep $t+1$ is then built from this list according to an allocation policy (next section).

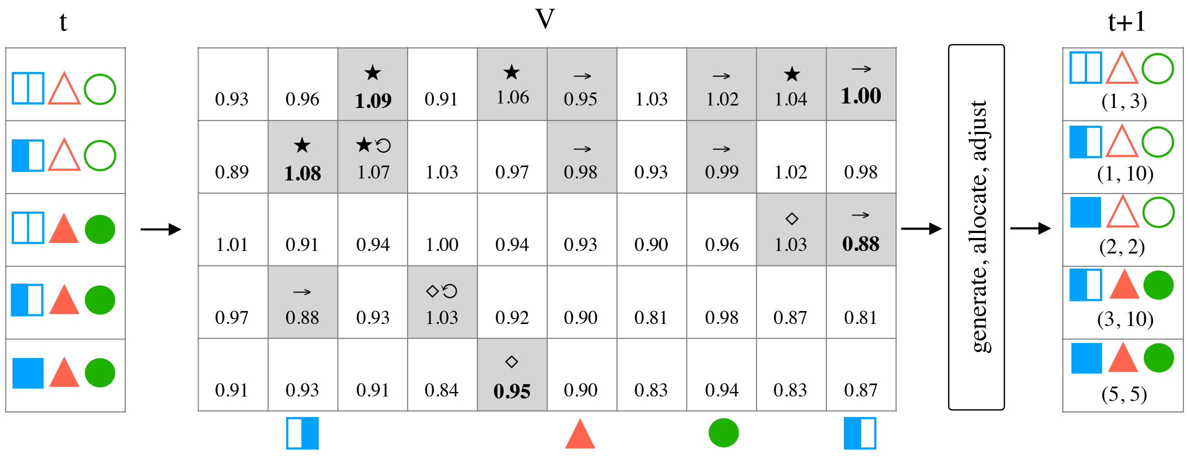

Figure 2: A single step of the constrained decoder. Along the left is the beam ($k = 5$) at time step t. The shapes in this beam represent constraints, both met (filled) and unmet (outlined). The blue square represents a phrasal constraint of length 2, which must be completed in order (left half, then right half). A step of the decoder produces a $k \times V_T$ matrix of scores. Each constraint corresponds to a single token in the vocabulary, and is marked along the bottom. Gray squares denote the set of candidates that are produced (§3.1) from the k best items (★), from extending each hypothesis with all unfilled constraints (→), and from its single-best next token (◇). Items that violate a phrasal constraint (⟲) require the phrasal constraint from that hypotheses to be unwound (set to unmet). From these fifteen candidates, the beam at time step $t + 1$ is filled, according to the bank allocation strategy, which here assigns one slot in the beam to each bank. The final beam includes coordinates indicating the provenance of chosen items (which are also indicated in bold in the grid).

For hypotheses partially through a phrasal constraint, special care must be taken. If a phrasal constraint has been begun, but not finished, and a token is chosen that does not match the next word of the constraint, we must reset or "unwind" those tokens in this constraint that are marked as having been met. This permits the decoder to abort the generation of a phrasal constraint, which is important in situations where a partial prefix of a phrasal constraint appears in the decoded sentence earlier than the entire phrase.

3.2 Allocating the beam

The task is to allocate a size-k beam across $C + 1$ constraint banks, where C may be greater than k. We use the term *bank* to denote the portion of the beam reserved for items having met the same number of constraints (including one bank for hypotheses with zero constraints met). We use a simple allocation strategy, setting each bin size to $\lfloor k/C \rfloor$, irrespective of the timestep. Any remaining slots are assigned to the "topmost" or maximally constrained bank, C.

This may at first appear wasteful. For example, space allocated at timestep 1 to a bank representing candidates having met more than one constraint cannot be used, and similarly, for later timesteps, it seems wasteful to allocate space to bank 1. Additionally, if the number of candidates in a bank is smaller than the allocation for that bank, the beam is in danger of being underfilled. These problems are mitigated by bank adjustment (Figure 3). We provide here only a sketch of this procedure. An overfilled bank is one that has been allocated more slots than it has candidates to fill. Each such overfilled bank, in turn, gives its extra allotments to banks that have more candidates than slots, looking first to its immediate neighbors, and moving outward until it has distributed all of its extra slots. In this way, the beam is filled, up to the minimum of the beam size or the number of candidates.

3.3 Finishing

Hypotheses are not allowed to generate the end-of-sentence token, $\langle/\mathrm{s}\rangle$, unless they have met all of their constraints. When beam search is finished, the highest-scoring completed item is returned.

4 Experimental Setup

Our experiments were done using SOCKEYE (Hieber et al., 2017). We used an English–German model trained on the complete WMT'17 training corpora (Bojar et al., 2017), which we pre-

Algorithm 2 k-best extraction with DBA. *Inputs:* A $k \times |V_T|$ matrix of model states.

```
1:  function KBEST-DBA(beam, scores)
2:      constraints ← [hyp.constraint for hyp in beam]
3:      candidates ← [$(i, j$, constraints[$i$].add($j$)] for $i, j$ in ARGMAX_K($k$, scores)      ▷ Top overall $k$
4:      for $1 \leq h \leq k$ do                                                        ▷ Go over current beam
5:          for all $w \in V_T$ that are unmet constraints for beam[$h$] do               ▷ Expand new constraints
6:              candidates.append( $(h, w$, constraints[$h$].add($w$) ) )
7:          $w$ = ARGMAX(scores[$h$, :])
8:          candidates.append( $(h, w$,  constraints[$h$].add($w$)) )                     ▷ Best single word
9:      selected ← ALLOCATE(candidates, $k$)
10:     newBeam ← [candidates[$i$] for $i$ in selected]
11:     return newBeam
```

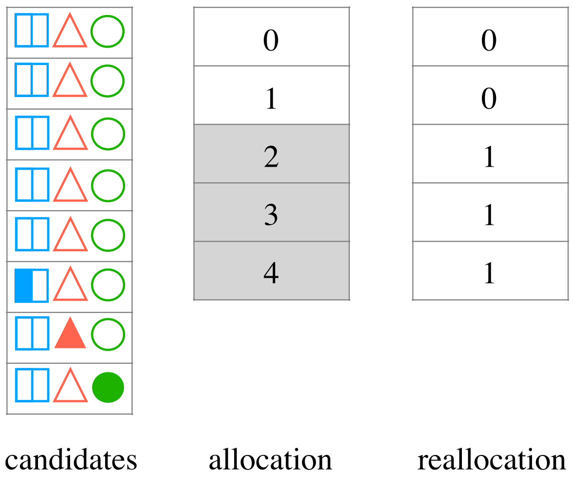

Figure 3: Beam reallocation for $k = 5$ with 4 constraints at timestep t. There are eight candidates, each having met only 0 or 1 constraint. The allocation policy gives one slot of the beam to each bank. However, there are no candidates for banks 2–4 (greyed), so their slots are redistributed to banks 0 and 1.

processed with the Moses tokenizer (preserving case) and with a joint byte-pair-encoded vocabulary with 32k merge operations (Sennrich et al., 2016). The model was a 4 layer RNN with attention. We trained using the Adam optimizer with a batch size of 80 until cross-entropy on the development data (newstest2016) stopped increasing for 10 consecutive iterations.

For decoding, we normalize completed hypotheses (those that have generated $\langle/s\rangle$), dividing the cumulative sentence score by the number of words. Unless otherwise noted, we apply threshold pruning to the beam, removing hypotheses whose log probability is not within 20 compared to the best completed hypothesis. This pruning is applied to all hypotheses, whether they are complete or not. (We explore the importance of this pruning in §6.3). Decoding stops when either all hypotheses still on the beam are completed or the maximum length, N, is reached. All experiments were run on a single a Volta P100 GPU. No ensembling or batching were used.

For experiments, we used the newstest2014 English–German test set (the developer version, with 2,737 sentences). All BLEU scores are computed on detokenized output using SACREBLEU (Post, 2018),[1] and are thus directly comparable to scores reported in the WMT evaluations.

5 Validation Experiment

We center our exploration of DBA by experimenting with constraints randomly selected from the references. We extract five sets of constraints: from one to four randomly selected words from the reference (`rand1` to `rand4`), and a randomly selected four-word phrase (`phr4`). We then apply BPE to these sets, which often yields a much larger number of token constraints. Statistics about these extracted phrases can be found in Table 2.

We simulate the GBS baseline within our framework. After applying BPE, We group together translations with the same number of constraints, C, and then translate them as a group, with the beam set for that group set to $b(C + 1)$, where b is the "base beam" parameter. We use $b = 10$ as reported in Hokamp et al., but also try smaller values of $b = 5$ and 1. Finally, we disable beam adjustment (§3.2), so that the space allocated to each constraint bank does not change.

Table 4 compares speeds and BLEU scores (in the legend) as a function of the number of post-

[1]The signature is BLEU+case.mixed+lang.en-de+numrefs.1+smooth.exp+test.wmt14+tok.13a+v.1.2.6

num	rand1	rand2	rand3	rand4	phr4
1	2,182	0	0	0	0
2	548	3,430	0	0	0
3	516	1,488	4,074	0	0
4	272	1,128	2,316	4,492	4,388
5	150	765	1,860	3,275	2,890
6	30	306	1,218	2,520	2,646
7	42	133	805	1,736	1,967
8	0	112	488	1,096	1,280
9	0	36	171	702	720
10	0	10	140	400	430
11+	0	22	189	417	575
total	3,726	7,477	11,205	14,885	14,926
mean	1.36	2.73	4.09	5.43	5.45

Table 2: Histogram of the number of token constraints for some constraint sets after applying BPE (model trained with 32k merge operations). *mean* denotes the mean number of constraints per sentence in the 2,737-sentence test set.

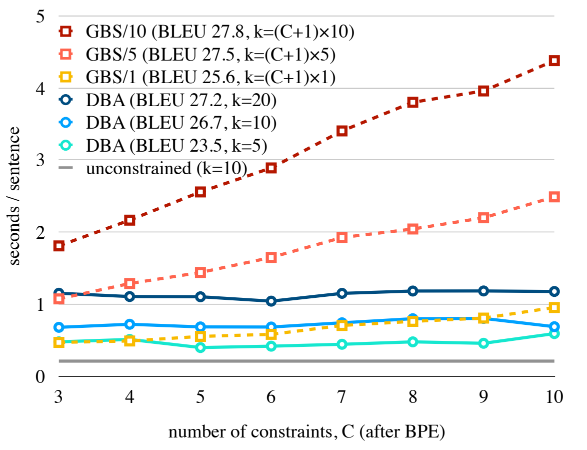

Figure 4: Running time (seconds / sentence, lower is better) as a function of the number of constraints, C (after applying BPE) on the `rand3` dataset. The unconstrained baselines have BLEU scores of 22.3, 22.3, and 22.1 for $k = 5, 10$, and 20, respectively.

BPE constraints for the `rand3` dataset. We plot all points for which there were at least 10 sentences. The times are decoding only, and exclude model loading and other setup. The linear trend in C is clear for GBS, as is the constant trend for DBA. In terms of absolute runtimes, DBA improves considerably over GBS, whose beam sizes quickly become quite large with a non-unit base beam size. On the Tesla V100 GPU, DBA ($k = 10$) takes about 0.6 seconds/sentence, regardless of the number of constraints.[2] This is about 3x slower than unconstrained decoding.

It is difficult to compare these algorithms exactly because of GBS's variable beam size. An important comparison is that between DBA ($k = 10$) and GBS/1 (base beam of 1). A beam of $k = 10$ is a common setting for decoding in general, and GBS/1 has a beam size of $k \geq 10$ for $C \geq 9$. At this setting, DBA finds better translations (BLEU 26.7 vs. 25.6) with the same runtime and with a fixed, instead of variable-sized, beam.

We note that the bank adjustment correction of the DBA algorithm allows it to work when $C >= k$. The DBA ($k = 5$) plot demonstrates this, while still finding a way to increase the BLEU score over GBS (23.5 vs. 22.3). However, while possible, low k relative to C reduces the observed improvement considerably. Looking at Figure 5 across different constraint sets, we can get a better feel for this relationship. DBA is still always able to meet the constraints even with a beam size of 5, but the quality suffers. This should not be too surprising; correctly placing independent constraints is at least as hard as finding their correct permutation, which is exponential in the number of independent constraints. But it is remarkable that the only failure to beat the baseline in terms of BLEU is when the algorithm is tasked with placing four random constraints (before BPE) with a beam size of 5. In contrast, DBA never has any trouble placing phrasal constraints (dashed lines).

6 Analysis

6.1 Placement

It's possible that the BLEU gains result from a boost in n-gram counts due to the mere presence of the reference constraints in the output, as opposed to their correct placement. This appears not to be the case. Experience examining the outputs shows its uncanny ability to sensibly place constrained words and phrases. Figure 6 contains some examples from translating a German sentence into English, manually identifying interesting phrases in the target, choosing paraphrases of those words, and then decoding with them as constraints. Note that the word weak, which doesn't fit in the semantics of the reference, is placed haphazardly.

We also confirm this correct placement quantitatively by comparing the location of the first word of each constraint in (a) the reference and (b) the output of the constrained decoder, represented as a percentage of the respective sentence lengths (Figure 7). We would not expect these numbers to

[2]On a K80, it is about 1.4 seconds / sentence

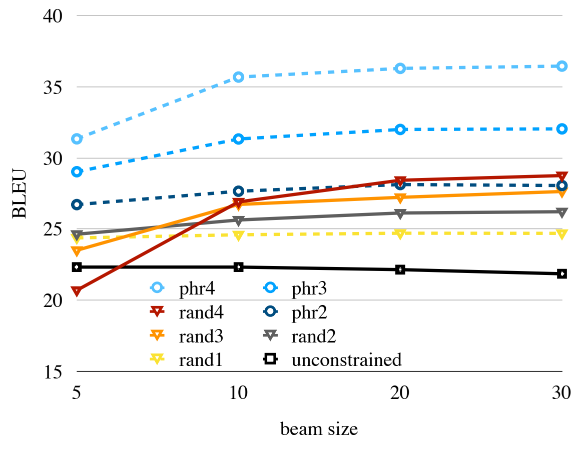

Figure 5: BLEU score as a function of beam size under DBA. All constraint sets improve as the beam gets larger (recall that the actual number of constraints increases after BPE and varies by sentence). `rand4` performs under the unconstrained baseline if the beam is too low.

be perfectly matched, but the strong correlation is pretty apparent (Pearson's $r = 0.82$). Together, Figures 6 and 7 provide confidence that DBA is intelligently placing the constraints.

6.2 Reference Aversion

The inference procedure in SOCKEYE maximizes the length-normalized version of the sentence's log probability. While there is no explicit training towards the metric, BLEU, modeling in machine translation assumes that better model scores correlate with better BLEU scores. However, a general repeated observation from the NMT literature is the disconnect between model score and BLEU score. For example, work has shown that opening up the beam to let the decoder find better hypotheses results in lower BLEU score (Koehn and Knowles, 2017), even as the model score rises. The phenomenon is not well understood, but it seems that NMT models have learned to travel a path straight towards their goal; as soon as they get off this path, they get lost, and can no longer function (Ott et al., 2018).

Another way to look at this problem is to ask what the neural model thinks of the references. Scoring against complete references is easy with NMT (Sennrich, 2017), but lexically-constrained decoding allows us to investigate this in finer-grained detail by including just portions of the references. We observe that forcing the decoder to include even a single word from the reference imposes a cost in model score that is inversely

	0	3	5	10	20	30
none	24.4	24.5	24.5	24.4	24.5	24.4
rand1	25.2	25.1	25.2	25.6	25.5	25.3
rand2	26.0	25.3	25.6	26.1	26.7	26.4
rand3	26.5	24.7	24.9	25.7	26.9	27.2
rand4	26.2	23.7	23.9	24.6	26.0	26.9
phr4	35.1	33.5	33.5	34.0	35.0	35.9

Table 3: BLEU scores decoding with a beam size of 10. Runtimes for unpruned systems (column 0) are nearly twice those of the other columns. But it is only at large thresholds that BLEU scores are higher than the unpruned setting.

correlated with BLEU score, and that this grows with the number of constraints that are added (Figure 8). The NMT system seems quite averse to the references, even in small pieces, and even while it improves the BLEU score. At the same time, the hypotheses it finds in this reduced space are still good, and become better as the beam is enlargened (Figure 5). This provides a complementary finding to that of Koehn and Knowles (2017): in that setting, higher model scores found by a larger beam produce lower BLEU scores; here, lower model scores are associated with significantly higher BLEU scores.

6.3 Effects of Pruning

In the results reported above, we used a pruning threshold of 20, meaning that any hypothesis whose log probability is not within 20 of the best completed hypothesis is removed from the beam. This pruning threshold is far greater than those explored in other papers; for example, Wu et al. (2016) use 3. However, we observed two things: first, without pruning, running time for constrained decoding is nearly doubled. This increased runtime applies to both DBA and GBS in Figure 4. Second, low pruning thresholds are harmful to BLEU scores (Table 3). It is only once the thresholds reach 20 that the algorithm is able to find better BLEU scores compared to the unpruned baseline (column 0).

6.4 Garbage Generation

Why is the algorithm so slow without pruning? One might suspect that the outputs are longer, but mean output length with all constraint sets is roughly the same. The reason turns out to be that the the decoder never quits before the maximum

constraint	score	output
source		*Einer soll ein hochrangiges Mitglied aus Berlin gewesen sein .*
no constraints	-0.217	One should have been a high-ranking member from Berlin .
is said to	-0.551	One is said to have been a high-ranking member from Berlin .
of them	-0.577	One of them was to be a high-ranking member from Berlin .
participant	-0.766	One should have been a high-ranking participant from Berlin .
is thought to	-0.792	One is thought to have been a high-ranking member from Berlin .
considered	-0.967	One is considered to have been a high-ranking member from Berlin .
Hamburg	-1.165	One should have been a high-ranking member from Hamburg .
powerful	-1.360	One is to have been a powerful member from Berlin .
powerful, is said to	-1.496	One is said to have been a powerful member from Berlin .
powerful, is said to, participant	-1.988	One is said to have been a powerful participant from Berlin .
weak	-1.431	One weak point was to have been a high-ranking member from Berlin .
reference		*One is said to have been a high-ranking member from Berlin.*

Figure 6: Example demonstrating the correct placement of manually chosen constraints (beam size 10). The unnatural placement of the constraint *weak* demonstrates what the model does when forced to include a word that is not a semantic fit.

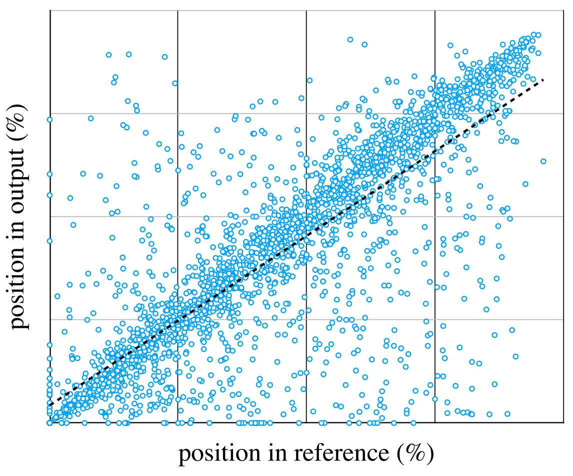

Figure 7: Location of the first word of each constraint from `phr3` in the reference versus the constrained output (Pearson's $r = 0.82$). DBA correctly places its constraints, even though no source word or alignment information is provided.

timestep, N. SOCKEYE's stopping criterium is to wait until all hypotheses on the beam are finished. Without pruning, the decoder generates a finished hypotheses, but continues on until the maximum timestep N, populating the rest of the beam with low-cost garbage. An example can be found in Figure 9. This may be an example of the well-attested phenomenon where NMT systems become unhinged from the source sentence, switching into "language model" mode and generating high-probable output with no end. But strangely, this doesn't seem to affect the best hypotheses, but only the rest of the beam. This seems to be more evidence of reference aversion, where

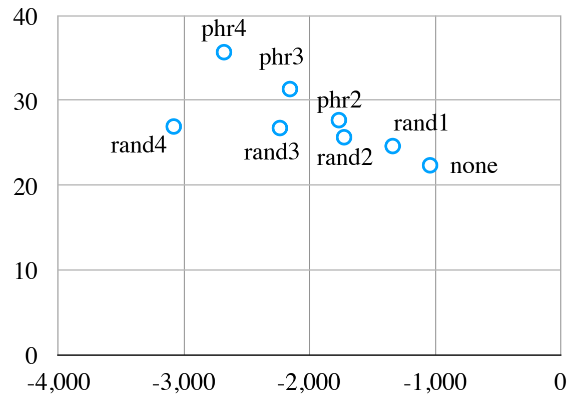

Figure 8: BLEU score as a function of model score (summed over the corpus). The reference model score is -4,396.

the decoder, having been forced into a place it doesn't like, does not know how to generate good competing hypotheses.

An alternative to pruning is *early stopping*, which is to stop when the first complete hypothesis is generated. In our experiments, while this did fix the problem of increasing runtimes, the BLEU scores were lower.

6.5 Conclusions

By setting a large pruning threshold, we produced large speedups over GBS, and demonstrated a constant overhead in the number of constraints. Compared to GBS, our DBA algorithm makes lexically constrained decoding possible, requiring less than half a second on average on a Volta GPU with a 4-layer RNN.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	…	41
-0.51	⟨s⟩	Er	und	Kerr	lieben	einander	noch	immer	,	betonte	die	36-Jährige	.	⟨/s⟩		
-0.52	⟨s⟩	Er	und	Kerr	lieben	einander	noch	immer	,	betonte	der	36-Jährige	.	⟨/s⟩		
-0.56	⟨s⟩	Er	und	Kerr	lieben	einander	noch	immer	,	betonte	die	36-jährige	.	⟨/s⟩		
-0.57	⟨s⟩	Er	und	Kerr	lieben	einander	noch	immer	,	betonte	den	36-jährigen	.	⟨/s⟩		
-25.11	⟨s⟩	Er	und	Kerr	lieben	sich	weiterhin	einander	,	betonte	die	36-Jährige	.	&#160;	…	&#160;
-27.92	⟨s⟩	Er	und	Kerr	lieben	sich	weiterhin	einander	,	betonte	die	36-Jährige	.	&#160;	…	&#160;

Figure 9: The sentence *He and Kerr still love each other , emphasised the 36-year-old .* translated with the constraint *noch immer , betonte* (BPE removed for readability). The first column is the log probability, which is normalized only for finished hypotheses. The decoder completes a few hypotheses well before the maximum timestep, but then fills the lower beam with garbage until forced to stop.

7 Related Work

Hokamp and Liu (2017) was novel in that it allowed the specification of arbitrary target-side words as hard constraints, implemented entirely as a restructuring of beam search, and without reference to the source. A related approach was that of Anderson et al. (2017), who extended beam search with a finite state machine whose states marked completed subsets of the set of constraints, at an exponential cost in the number of constraints.

Lexically-constrained decoding also generalizes prefix decoding (Knowles and Koehn, 2016; Wuebker et al., 2016), since the ⟨s⟩ symbol can easily be included as the first word of a constraint.

Our work here has not explored where to get lexical constraints, but considering that question naturally brings to mind attempts to improve NMT by using lexicons and phrase tables (Arthur et al., 2016; Tang et al., 2016).

Finally, another approach which shares the hard-decision made by lexically constrained decoding is the *placeholder approach* (Crego et al., 2016), wherein identifiable elements in the input are transformed to masks during preprocessing, and then replaced with their original source-language strings during postprocessing.

8 Summary

Neural machine translation removes many of the knobs from phrase-based MT that provided fine-grained control over system output. Lexically-constrained decoding restores one of these tools, providing a powerful and interesting way to influence NMT output. It requires only the specification of the target-side constraints; without any source word or alignment information, it correctly places the constraints. Although we have only tested it here with RNNs, the code works without modification with other architectures generate target-side words one-by-one, such as the Transformer (Vaswani et al., 2017).

This paper has introduced a fast and practical solution. Building on previous approaches, constrained decoding with DBA does away with linear and exponential complexity (in the number of constraints), imposing only a constant overhead. On a Volta GPU, lexically-constrained decoding with DBA is practical, requiring about 0.6 seconds per sentence on average even with 10+ constraints, well within the realm of feasibility even for applications with strict lattency requirements, like post-editing tasks. We imagine that there are further optimizations in reach that could improve this even further.

Acknowledgments We thank Felix Hieber for valuable discussions.

References

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

Philip Arthur, Graham Neubig, and Satoshi Nakamura. 2016. Incorporating discrete translation lexicons into neural machine translation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1557–1567. Association for Computational Linguistics.

Ondřej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Shujian Huang, Matthias Huck, Philipp Koehn, Qun Liu, Varvara Logacheva, Christof Monz, Matteo Negri, Matt Post, Raphael Rubino, Lucia Specia, and Marco Turchi. 2017. Findings of the 2017 conference on machine translation (wmt17). In *Proceedings of the Second Conference on Machine Translation*, pages 169–214. Association for Computational Linguistics.

Josep Crego, Jungi Kim, Guillaume Klein, Anabel Rebollo, Kathy Yang, Jean Senellart, Egor Akhanov, Patrice Brunelle, Aurelien Coquard, Yongchao Deng, Satoshi Enoue, Chiyo Geiss, Joshua Johanson, Ardas Khalsa, Raoum Khiari, Byeongil Ko, Catherine Kobus, Jean Lorieux, Leidiana Martins, Dang-Chuan Nguyen, Alexandra Priori, Thomas Riccardi, Natalia Segal, Christophe Servan, Cyril Tiquet, Bo Wang, Jin Yang, Dakun Zhang, Jing Zhou, and Peter Zoldan. 2016. Systran's pure neural machine translation systems. *CoRR*.

Felix Hieber, Tobias Domhan, Michael Denkowski, David Vilar, Artem Sokolov, Ann Clifton, and Matt Post. 2017. Sockeye: A toolkit for neural machine translation. *Computing Research Repository*, abs/1712.05690.

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

Howard Johnson, Joel Martin, George Foster, and Roland Kuhn. 2007. Improving translation quality by discarding most of the phrasetable. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (EMNLP-CoNLL)*.

Rebecca Knowles and Philipp Koehn. 2016. Neural interactive translation prediction. In *Proceedings of the Association for Machine Translation in the Americas*, pages 107–120.

Philipp Koehn and Rebecca Knowles. 2017. Six challenges for neural machine translation. In *Proceedings of the First Workshop on Neural Machine Translation*, pages 28–39. Association for Computational Linguistics.

Philipp Koehn, Franz J. Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics*.

Pierre Lison and Jörg Tiedemann. 2016. Opensubtitles2016: Extracting large parallel corpora from movie and tv subtitles. In *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Paris, France. European Language Resources Association (ELRA).

Bruce T. Lowerre. 1976. *The HARPY speech recognition system*. Ph.D. thesis, Carnegie-Mellon University.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60. Association for Computational Linguistics.

Myle Ott, Michael Auli, David Grangier, and Marc'Aurelio Ranzato. 2018. Analyzing uncertainty in neural machine translation. *CoRR*, abs/1803.00047.

Matt Post. 2018. A call for clarity in reporting bleu scores. *Computing Research Repository*, abs/1804.08771.

Rico Sennrich. 2017. How grammatical is character-level neural machine translation? assessing mt quality with contrastive translation pairs. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pages 376–382. Association for Computational Linguistics.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725. Association for Computational Linguistics.

Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 3104–3112. Curran Associates, Inc.

Yaohua Tang, Fandong Meng, Zhengdong Lu, Hang Li, and Philip L. H. Yu. 2016. Neural machine translation with external phrase memory. *CoRR*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144.

Joern Wuebker, Spence Green, John DeNero, Sasa Hasan, and Minh-Thang Luong. 2016. Models and inference for prefix-constrained machine translation. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75. Association for Computational Linguistics.

A Appendix: Failed Experiments

The main contribution of this paper is a fast, practical algorithmic improvement to lexically-constrained decoding. While we did not attempt to corroborate the experiments in interactive translation and domain adaptation experiments reported in (Hokamp and Liu, 2017), the gains discovered there only become more salient with this faster algorithm. We did try to apply lexical constraints in a few other settings, but without success. In the spirit of open scientific inquiry and reporting, we provide here a brief report on these experiments.

A.1 Automatic Constraint Selection

Our validation experiments (§5) demonstrate the large potential gains in BLEU score when including random phrases from the reference. Even including just a single random word from the reference increased BLEU score by a point; another point was gained from including two random words, and a four-word phrase yielded 10+ point gains. Only about 18% of these random unigrams were present in the unconstrained output (less for longer n-grams). This raises the question of whether we can automatically identify words that are likely to be in the reference and include them as constraints, in order to improve translation quality.

In order to do that, we first extracted a phrase table using Moses and filtered it with the significance testing approach proposed by Johnson et al. (2007) in order to keep only high quality phrases. We then selected the best phrase for each input sentence according to different criteria (longest phrase, higher significance, highest probability, combination of those). Unfortunately, adding such phrases as constraints when translating WMT or IWSLT data did not help.

A.2 Name Entity Translation

One topic that has received attention in the literature is the tendency of NMT systems to do poorly with rare words, and in particular, named entities (e.g., Arthur et al. (2016)). BPE helps address this by breaking down words into pieces and allowing all words to be represented in the decoder's vocabulary. But even with BPE, many times the correct translation does not follow any pattern, even at the subword level. This is specially true for named entities; e.g. "Aachen" in German is translated as "Aquisgrán" in Spanish or "Aix-la-Chapelle" in French, which bear little resemblance to the original form except for the starting letter. Named entities (NEs) also have the advantage that in several languages they are not inflected; therefore a simple lookup in a dictionary, if available, should produce the correct translation.

To the best of our knowledge, there is no publicly available parallel corpus of named entities. In order to create one, we downloaded the OpenSubtitles database (Lison and Tiedemann, 2016) for German and English and applied a simple method for extracting named entity correspondences. We first tagged the source and target sides with the Stanford NER system (Manning et al., 2014). We then selected a subset of the tags that were produced by both systems ("Person", "Location" and "Organization") and selected those sentences where they appeared only once for each language. From those we extracted the corresponding NEs, selecting the most frequent target side at the corpus level as the translation of a given source NE.

Given such a dictionary, we can add the translation of a NE found in new sentences to translate as decoding constraints. It didn't help. Manual inspection showed that the dictionary extracted with this simple method was still too noisy. We think that a manual, high-quality dictionary may provide a way to produce improvements.