Translation Rules with Right-Hand Side Lattices

Fabien Cromières
Japan Science and Technology Agency
Kawaguchi-shi
Saitama 332-0012
fabien@pa.jst.jp

Sadao Kurohashi
Graduate School of Informatics
Kyoto University
Kyoto 606-8501
kuro@i.kyoto-u.ac.jp

Abstract

In Corpus-Based Machine Translation, the search space of the translation candidates for a given input sentence is often defined by a set of (cycle-free) context-free grammar rules. This happens naturally in Syntax-Based Machine Translation and Hierarchical Phrase-Based Machine Translation (where the representation will be the set of the target-side half of the synchronous rules used to parse the input sentence). But it is also possible to describe Phrase-Based Machine Translation in this framework. We propose a natural extension to this representation by using lattice-rules that allow to easily encode an exponential number of variations of each rules. We also demonstrate how the representation of the search space has an impact on decoding efficiency, and how it is possible to optimize this representation.

1 Introduction

A popular approach to modern Machine Translation is to decompose the translation problem into a modeling step and a search step. The modeling step will consist in defining implicitly a set of possible translations $\mathcal{T}$ for each input sentence. Each translation in $\mathcal{T}$ being associated with a real-valued *model score*. The search step will then consist in finding the translation in $\mathcal{T}$ with the highest model score. The search is non-trivial because it is usually impossible to enumerate all members of $\mathcal{T}$ (its cardinality being typically exponentially dependent on the size of the sentence to be translated).

Since at least (Chiang, 2007), a common way of representing $\mathcal{T}$ has been through a cycle-free context-free grammar. In such a grammar, $\mathcal{T}$ is represented as a set of context-free rules such as can be seen on figure 1. These rules themselves can be generated by the modeling step through the use of phrase tables, synchronous parsing, tree-to-string rules, etc. If the model score of each translation is taken to be the sum of *rule scores* independently given to each rule, the search for the optimal translation is easy with some classic dynamic programming techniques.

However, if the model score is going to take into account informations such as the language model score of each sentence, it cannot be expressed in such a way. Since the language model score has proven empirically to be a very good source of information, (Chiang, 2007) proposed an approximate search algorithm called *cube pruning*.

We propose here to represent $\mathcal{T}$ using context-free lattice-rules such as shown in figure 2. This allows us to compactly encode a large number of rules. One benefit is that it adds flexibility to the modeling step, making it easier: many choices such as whether or not a function word should be included, the relative position of words and non-terminal in the translation, as well as morphological variations can be delegated to the search step by encoding them in the lattice rules. While it is true that the same could be achieved by an explicit enumeration, lattice rules make this easier and more efficient.

In particular, we show that a decoding algorithm working with such lattice rules can be more efficient than one working directly on the enumeration of the rules encoded in the lattice.

A distinct but related idea of this paper is to consider how transforming the structure of the rules defining $\mathcal{T}$ can lead to improvements

Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 577–588,
October 25-29, 2014, Doha, Qatar.

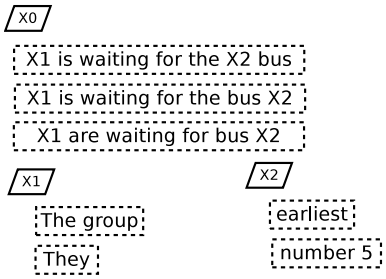

Figure 1: A simple cycle-free context grammar describing a set of possible translations.

in the speed/memory performances of the decoding. In particular, we propose a method to merge and reduce the size of the lattice rules and show that it translates into better performances at decoding time.

In this paper, we will first define more precisely our concept of lattice-rules, then try to give some motivation for them in the context of a tree-to-tree MT system (section 3). In section 4, we then propose an algorithm for preprocessing a representation given in a lattice-rule form that allows for more efficient search. In section 5, we describe a decoding algorithm specially designed for handling lattice-rules. In section 6, we perform some experiments demonstrating the merit of our approach.

2 Notations and Terminology

Here, we define semi-formally the terms we will use in this paper. We assume knowledge of the classic terminology of graph theory and context-free grammar.

2.1 Expansion rules

A *flat expansion rule* is the association of a non-terminal and a "flat" right hand side that we note *RHS*. A *flat RHS* is a sequence of words and non-terminal. See figure 1 for an example of a set of flat expansion rules.

A set of expansion rules is often produced in Hierarchical or Syntax-Based MT, by parsing with synchronous grammars or otherwise. In such a case, the set of rules define a representation of the (weighted) set of possible translations $\mathcal{T}$ of an input sentence.

2.2 Lattice

In the general sense, a lattice can be described as a labeled directed acyclic graph. More precisely, the type of lattice that we consider in this work is such that:

- Edges are labeled by either a word, a non-terminal or an epsilon (ie. an empty string).
- Vertices are only labeled by a unique id by which they can be designated.

Additionally, edges can also be labeled by a real-valued *edge score* and some real-valued *edge features*. Alternatively, a lattice could also be seen as an acyclic Finite State Automaton, with vertices and edges corresponding to states and transitions in the FSA terminology.

For simplicity, we also set the constraint that each lattice has a unique "start" vertex labeled v_S from which each vertex can be reached and a unique "end" vertex v_E that can be reached from each vertex. Each path from v_S to v_E define thus a flat RHS, with score and features obtained by summing the score and features of each edge of the path.

A *lattice expansion rule* is similar to a flat expansion rule, but with the RHS being a lattice. Thus a set of lattice expansion rules can also define a set of possible translations $\mathcal{T}$ of an input sentence.

For a given lattice $\mathcal{L}$, we will often note $v \in \mathcal{L}$ a vertex of $\mathcal{L}$ and $e : v_1 \to v_2 \in \mathcal{L}$ an edge of $\mathcal{L}$ going from vertex v_1 to vertex v_2.

Figures 2 and 3 show examples of such lattices.

2.3 Translation set and Representations

We note $\mathcal{T}$ a set of weighted sentences. $\mathcal{T}$ is intended as representing the set of scored translation candidates generated by a MT system for a given input sentence. As is customary in Corpus-Based MT literature, we will call *decoding* the process of searching for the translation with highest score in $\mathcal{T}$.

A representation of $\mathcal{T}$, noted $\mathcal{R}_T$ is a set of rules in a given formalism that implicitly define $\mathcal{T}$. As we mentioned earlier, in MT, $\mathcal{R}_T$ is often a set of cycle-free context-free grammar rules.

In this paper, we consider representations $\mathcal{R}_T$ consisting in a set of lattice expansion rules. With normal context-free grammar, it is usually necessary that a non-terminal is the

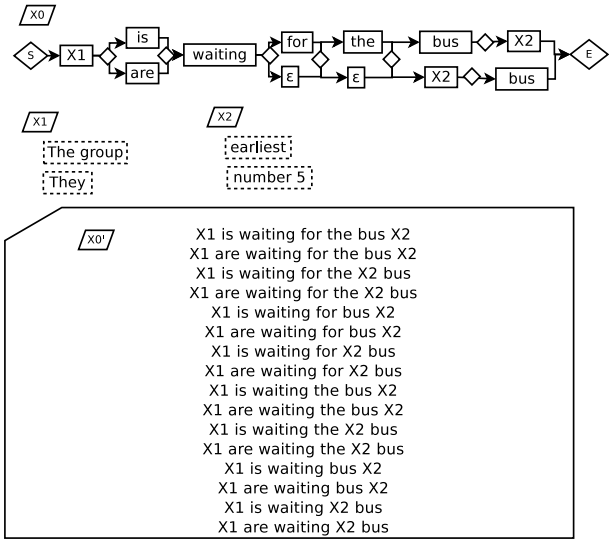

Figure 2: A simple example of lattice rule for non-terminal X0. The lower part list the set of "flat" rules that would be equivalent to the ones expressed by the lattice.

left-hand side of several rules. Using lattice expansion rules, however, it is not necessary, as one lattice RHS can encode an arbitrary number of flat rules (see for example the RHS of X0 in figure 3). Therefore, we set the constraint that there is only one lattice expansion rule for each left-hand non-terminal. And we will note unambiguously $RHS(X)$ the lattice that is the right hand side of this rule.

3 Motivation

3.1 Setting

This work was developed mainly in the context of a syntactic-dependency-based tree-to-tree translation system described in (Richardson et al., 2014). Although it is a tree-to-tree system, we simplify the decoding step by "flattening" the target-side tree translation rules into string expansion rules (keeping track of the dependency structure in state features). Thus our setting is actually quite similar to that of many tree-to-string and string-to-string systems. Aiming at simplicity and generality, we will set aside the question of target-side syntactic information and only describe our algorithms in a "tree-to-string" setting. We will also consider a n-gram language model score as our only stateful non-local feature.

However, this tree-to-tree original setting should be kept in mind, in particular when we describe the issue of the relative position of heads and dependents in section 3.2.2, as such issues do not appear as commonly in "X-to-string" settings.

3.2 Rule ambiguities

Expansion rules are typically created by matching part of the input sentence with some aligned example bilingual sentence. The alignment (and the linguistic structure of the phrase in the case of Syntax-Based Machine Translation) is then used to produce the target-side rule. However, it is often the case that it is difficult to fully specify a rule from an example. Such cases often come from two main reasons:

- Imperfect knowledge (eg. it is unclear whether a given unaligned word should belong to the translation)
- Context dependency (eg. the question of whether "to be" should be in plural form or not, depending on its subject in the constructed translation).

In both situation, it seems like it would be better to delay the full specification of the rule until decoding time, when the decoder can have access to the surrounding context of the rule and make a more informed choice. In particular, we can expect features such as language model or governor-dependent features (in the case of tree-to-tree Machine translation) to help remove the ambiguities.

We detail some cases for which we encode variations as lattice-rule.

3.2.1 Non-aligned words

When rules are extracted from aligned examples, we often find some target words which are not aligned to any source-side word and for which it is difficult to decide whether or not they should be included in the rule. Such words are often function words that do not have an equivalent in the source language. In Japanese-English translations, for example, articles such as "a" and "the" do not typically have equivalent in the Japanese side, and their necessity in the final sentence will often be a matter of context. We can make these edges

optionals by doubling them with an epsilon-edge. Different weights and features can be given to the epsilon edges to balance the tendency of the decoder to skip edges. In figure 2, this is illustrated by the epsilon edges allowing to skip "for" and "the"

3.2.2 Non-terminal positions

In the context of our tree-to-tree translation system, we often find that we know which target word should be the governor of a given non-terminal, but that we are unsure of the order of the words and non-terminals sharing a common governor. It can be convenient to represent such ambiguities in a lattice format as shown in figure 2. In this figure, one can see that the RHS of $X0$ encode two possible ordering for the word "bus" and the non-terminal $X2$.

3.2.3 Word variations

Linguistics phenomenons such as morphological variations can naturally create many minor problems in the setting of Corpus-Based Translation. Especially if the variations in the target language have no equivalence in the source language. An example of this in Japanese-English translation is the fact that verbs in Japanese are "plural-independent", while the verb "to be" in English is not. Therefore, a RHS that is a candidate for translating a large part of a Japanese input sentence can easily use one of the variant of "to be" that is not consistent with the full sentence. To solve this, for each edge corresponding to the words "is" or "are", we add an alternative edge with the same start and end vertices as the other word. The decoder will then be able to choose the edge that gives the best language model score. The same can be done, for example, for the article "a/an". Figure 2 provides an example of this, with two edges "is" and "are" in the RHS of X0.

Alternative edges can be labeled with different weights and features to tune the tendency of the decoder to choose a morphological variation.

While such variations could be fixed in a post-processing step, we feel it is a better option to let the decoder be aware of the possible options, lest it would discard rules due to language model considerations when these rules could actually have been useful with a simple change.

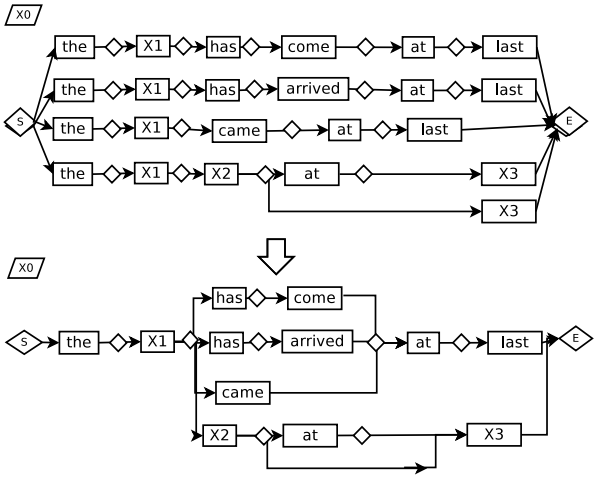

Figure 3: The lattice RHS(X0) optimized with the algorithm described in section 4

4 Representation optimisation

4.1 Goal

Given a description as a set of rule and scores $\mathcal{R}_T^1$ of $\mathcal{T}$, it is often possible to find another description $\mathcal{R}_T^2$ of $\mathcal{T}$ having the same formalism but a different set of rules. Although the $\mathcal{T}$ that is described remains the same, the same search algorithm applied to $\mathcal{R}_T^1$ or $\mathcal{R}_T^2$ might make approximations in a different way, be faster or use less memory.

It is an interesting question to try to transform an initial representation $\mathcal{R}_T^1$ into a representation $\mathcal{R}_T^2$ that will make the search step faster. This is especially interesting if one is going to search the same $\mathcal{T}$ several times, as is often done when one is fine-tuning the parameters of a model, as this representation optimisation needs only be done once.

The optimisation we propose is a natural fit to our framework of lattice rules. As lattice are a special case of Finite-State Automata (FSA), it is easy to adapt existing algorithms for FSA minimization. We describe a procedure in algorithm 1, which is essentially a simplification and adaptation to our case of the more general algorithm of (Hopcroft, 1971) for FSA. The central parts of the algorithm are the two sub-procedures *backward vertex merging* and *forward vertex merging*. An example of the result of an optimisation is given on figure 3.

Data: Representation $\mathcal{R}_T$
Result: Optimized Representation
1 **for** *non-terminal* $X \in \mathcal{R}_T$ **do**
2 | Apply backward vertex merging to $RHS(X)$;
3 | Apply forward vertex merging to $RHS(X)$;
4 **end**
Algorithm 1: Representation optimisation

4.2 Forward and backward merging

We describe the *forward vertex merging* in algorithm 2. This merging will merge vertices and suppress redundant edges, proceeding from left to right. The end result is a lattice with a reduced number of vertices and edges, but encoding the same paths as the initial one.

The basic idea here is to check the vertices from left to right and merge the ones that have identical incoming edges. After having been processed by the algorithm, a vertex is put in the set $\mathcal{P}$ (line 9). At each iteration, the candidate set $\mathcal{C}$ contains the set of vertices that can potentially be merged together. It is updated at each iteration to contain the set of not-yet-processed vertices for which all incoming edges come from processed vertices (done by marking edges at line 6 and then updating $\mathcal{C}$ at line 10). At each iteration, the merging process consists in:

1. Eliminating *duplicate* edges from the processed vertices to the candidate vertices (line 5). These duplicate edges could have been introduced by the merging of previously processed vertices.

2. Merging vertices whose set of incoming edges is *identical*. Here, merging two vertices v_1 and v_2 means that we create a third vertex v_3 such that incoming(v_3) = incoming(v_1) = incoming(v_2), and outgoing(v_3) = outgoing($v_3$1) $\bigcup$ outgoing(v_2), then remove v_1 and v_2.

The backward vertex merging is defined similarly to the forward merging, but with going right to left and inverting the role of the incoming and outgoing edges.

Data: Lattice RHS $\mathcal{L}$
Result: Optimized Lattice RHS
1 $\mathcal{P} \leftarrow \emptyset$ //processed vertices;
2 $\mathcal{C} \leftarrow \{v_S\}$ //candidate set ;
3 **while** $|\mathcal{C}| > 0$ **do**
4 | **for** $v \in \mathcal{C}$ **do**
5 | | Eliminate duplicate edges in incoming(v);
6 | | Mark edges in outgoing(v);
7 | **end**
8 | Merge all vertices $v_1, v_2 \in \mathcal{C}$ such that incoming(v_1) = incoming(v_2);
9 | $\mathcal{P} \leftarrow \mathcal{P} \bigcup \mathcal{C}$;
10 | $\mathcal{C} \leftarrow \{v \in \mathcal{L} \setminus \mathcal{P}$ s.t. all edges in incoming(v) are marked$\}$;
11 **end**
Algorithm 2: Forward Vertex Merging

4.3 Optimizing the whole representation

Algorithm 1 describe the global optimisation procedure. For each lattice RHS, we just perform first a backward merge and then a forward merge.

We have set the constraint in section 2.3 that each non-terminal should have only one lattice RHS. Note here that if there are several RHS for a given non-terminal, we can first merge them by merging their start vertex and end vertex, then apply this optimisation algorithm to obtain a representation with one optimised RHS per non-terminal.

This optimisation could be seen as doing some form of *hypothesis recombination*, but offline.

In term of *rule optimisations*, we only consider here transformations that do not modify the number of non-terminals. But it is worthwhile to note that there are some sequence appearing in the middle of some rules that cannot be merged through a lattice representation, but could be factored as sub-rules appearing in different non-terminals. Indeed, a lattice rule could actually be encoded as a set of "flat" rules by introducing a sufficient number of non-terminals, but this could possibly be less efficient from the search algorithm point of view. We plan to investigate the effects of this type of rule optimisations in conjunction with the described lattice-type opti-

misations in the future.

4.4 Handling of Edge Features

In the context of parameter tuning, we usually want the decoder to output not only the translations, but also a list of features characterizing the way the translation was constructed. Such features are, for example, the number of rules used, the language model of the translation, etc. In out context, some features will be dependent on the specific edges used in a rule. For example, the epsilon edge used to optionally skip non-aligned words (see section 3.2.1) is labeled with a feature "nb-words-skipped" set to 1, so that we can obtain the number of words skipped in a given translation and tune a score penalty for skipping such words. Similar features also exist for picking a word variation (section 3.2.3).

In the description of the merging process of section 4.2, one should thus be aware that two edges are to be considered identical only if both their associated word and their set of feature values are identical. This can sometimes prevent useful merging of states to take place. A solution to this could be to follow (de Gispert et al., 2010) and to discard all these features information during the decoding. The features values are then re-estimated afterward by aligning the translation and the input with a constrained version of the decoder.

We prefer to actually keep track of the features values, even if it can reduce the efficiency of vertex merging. In that setting, we can also adapt the so-called Weight Pushing algorithm (Mohri, 2004) to a multivalues case in order to improve the "mergeability" of vertices. The results of section 6.1 shows that it is still possible to strongly reduce the size of the lattices even when keeping track of the features values.

5 Decoding algorithm

In order to make an optimal use of these lattice-rule representations, we developed a decoding algorithm for translation candidate sets represented as a set of lattice-rules. For the most part, this algorithm re-use many of the techniques previously developed for decoding translation search spaces, but adapt them to our setting.

5.1 Overview

The outline of the decoding algorithm is described by algorithm 3. For simplicity, the description only compute the optimal model score over the translations in the candidate set. It is however trivial to adapt the description to keep track of which sentence correspond to this optimal score and output it instead of the score. Likewise, using the technique described in (Huang and Chiang, 2005), one can easily output k-best lists of translations. For simplicity again, we consider that a n-gram language model score is the only stateful non-local feature used for computing the model score, although in a tree-to-tree setting, other features (local in a tree representation but not in a string representation) could be used. The model score of a translation t has therefore the shape:

$$score(t) = \lambda \cdot lm(t) + \sum_{e} score(e)$$

where λ is the weight of the language model, $lm(t)$ is the language model log-probability of t and the sum is over all edges e crossed to obtain t.

5.2 Scored language model states

Conceptually, in a lattice $\mathcal{L}$, at each vertex v, we can consider the partial translations obtained by starting at v_S and concatenating the words labeling each edge not labeled by a non-terminal until v. If an edge is labeled by a non-terminal X, we first traverse the corresponding lattice $RHS(X)$ following the same process. Such a partial translation can be reduced compactly to a *scored language model state* (l, r, s), where l represent the first n words[1] of the partial translation, r its last n words and s its partial score. It is clear that if two partial translations have the same l and r parts but different score, we can discard the one with the lowest score, as it cannot be a part of the optimal translation.

Further, using the state reduction techniques described in (Li and Khudanpur, 2008) and (Heafield et al., 2011), we can often reduce the size of l and r to less than n, allowing further opportunities for discarding sub-optimal

[1] n being the order of the language mode

partial translations. For better behavior during the cube-pruning step of the algorithm (see later), the partial score s of a partial translation includes rest-costs estimates (Heafield et al., 2012).

We define the concatenation operation on scored language model states to be: $(l_1, r_1, s_1) \oplus (l_2, r_2, s_2) = (l_3, r_3, s_3)$, where $s_3 = s_1 + s_2 + \lambda lm(r_1, l_2)$, with $lm(r_1, l_2)$ being the language model probability of l_2 given r_1 with rest-costs adjustments. r_3 and l_3 are the resulting minimized states. Similarly, if an edge e is labeled by a word, we define the concatenation of a scored state with an edge to be $(l_1, r_1, s_1) \oplus e = (l_2, r_2, s_2)$ where $s_2 = s_1 + score(e) + \lambda lm(word(e)|r_1)$.

Conveniently for us, the KenLM[2] open-source library (Heafield, 2011) provides functionalities for easily computing such concatenation operations.

5.3 Algorithm

Having defined these operations, we can now more easily describe algorithm 3. Each vertex v has a list best[v] of the scored states of the best partial translations found to be ending at v. On line 1, we initialize best[v_S] with $(.,.,0)$, where "." represent an empty language model state. We then traverse the vertices of the lattice in topological order.

For each edge $e : v_1 \rightarrow v_2$, we compute new scored states for best[v_2] as follow:

- if e is labeled by a word or an epsilon, we create a state $st_2 = st_1 \oplus e$ for each st_1 in best[v_1] (line 10).
- if e is labeled by a non-terminal X, we recursively call the decoding algorithm on the lattice $RHS(X)$. The value returned by the line 15 will be a set of states corresponding to optimal partial translations traversing $RHS(X)$. We can concatenate these states with the ones in best[v_1] to obtain states corresponding to partial translations ending at v_2 (line 6).

Results of the calls decode(X) are memoized, as the same non-terminal is likely to appear in several edges of a RHS and in several RHS.

Lines 5 and 6 are the "cube-pruning-like" part of the algorithm. The function $prune_K$ returns the K best combinations of states in best[v] and decode($RHS(X)$), where best means "whose sum of partial score is highest". It can be implemented efficiently through the algorithms proposed in (Huang and Chiang, 2005) or (Chiang, 2007).

The $L \leftarrow_{max} st$ operation on lines 6 and 10 has the following meaning: L is a list of scored language model state and st is a scored language model state. $L \leftarrow_{max} st$ means that, if L already contains a state st_2 with same left and right state as st, L is updated to contain only the scored state with the maximum score. If L do not contain a state similar to st, st in simply inserted into L. This is the "hypothesis recombination" part of the algorithm. The function $trunc_{K'}$ truncate the list best[v] to its K' highest-scored elements.

The final result is obtained by calling decode(X_0), where X_0 is the "top-level" non-terminal. The result of decode(X_0) will contain only one scored state of the form (BOS, EOS, s), with s being the optimal score.

The search procedure of algorithm 3 could be described as "breadth-first", since we systematically visit each edge of the lattice. An alternative would be to use a "best-first" search with an A*-like procedure. We have tried this, but either because of optimisation issues or heuristics of insufficient qualities, we did not obtain better results than with the algorithm we describe here.

6 Evaluation

We now describe a set of experiments aimed at evaluating our approach.

We use the Japanese-English data from the NTCIR-10 Patent MT task[3] (Goto et al., 2013). The training data contains 3 millions parallel sentences for Japanese-English.

6.1 Effect of Lattice Representation and Optimisation

We first evaluate the impact of the lattice representation on the performances of our decoding algorithm. This will allow us to measure

[2] http://kheafield.com/code/kenlm/

[3] http://ntcir.nii.ac.jp/PatentMT-2/

Data: Lattice RHS $\mathcal{L}$
Result: Sorted list of best states
1 best$[v_E] = \{(.,.,0.0)\}$;
2 **for** *vertex* $v \in \mathcal{L}$ *in topological order* **do**
3 **for** *edge* $e : v \rightarrow v_2 \in$ *outgoing(v)* **do**
4 **if** *label(e)* $= X$ **then**
5 **for** $st_1, st_2 \in prune_K$*(best[v], decode(RHS(X))* **do**
6 best$[v_2] \leftarrow_{max} st_1 \oplus st_2$;
7 **end**
8 **else**
9 **for** $st \in trunc_{K'}$*(best[v])* **do**
10 best$[v_2] \leftarrow_{max} st \oplus e$;
11 **end**
12 **end**
13 **end**
14 **end**
15 **return** best$[v_E]$;

Algorithm 3: Lattice-rule decoding. See body for detailed explanations.

the benefits of our compact lattice representation of rules, as well as the benefits of the representation optimisation algorithm of section 4.

We use our Syntactic-dependency system to generate a lattice-rule representation of the possible translations of the 1800 sentences of the development set of the NTCIR-10 Patent MT task. We then produce two additional representations:

1. An optimized lattice-rule representation using the method described in section 4.

2. An expanded representation, that unfold the original lattice-rule representation into "flat rules" enumerating each path in the original lattice-rule representation (like the list $X0'$ enumerate the lattice $X0$ in figure 2).

Table 1 shows 3 columns. One for each of these 3 representations. We can see that, as expected, the performances in term of average search time or peak memory used are directly related to the number of vertices and edges in the representation. We can also see that our representation optimisation step is quite efficient, since it is able to divide by two the number of vertices in the representation, on average. This leads to a 2-fold speed improvement in the decoding step, as well as a large reduction of memory usage.

6.2 Decoding performances

In order to further evaluate the merit of our approach, we now compare the results obtained by using our decoder with lattice-rules with using a state-of-the-art decoder on the set of flat expanded rules equivalent to these lattice rules.

We use the decoder described in (Heafield et al., 2013), which is available under an open-source license[4] (henceforth called K-decoder). In this experience, we expanded the lattice rules generated by our MT system for 1800 sentences into files having the required format for the K-decoder. This basically mean we computed an equivalent of the expanded representation of section 6.1. This process generated files ranging in size from 20MB to 17GB depending on the sentence. We then ran the K-decoder on these files and compared the results with our own. We used a beam-width of 10000 for the K-decoder. Experiments were run in single thread mode. Partly to obtain more consistent results, and partly because the K-decoder was risking using too much memory for our system.

The results on table 3 show that, as the K-decoder do not have access to a more compact representation of the rules, it end up needing a much larger amount of memory for decoding the same sentences.

In term of model score obtained, the performances are quite similar, with the lattice-rule decoder providing slightly better model score.

It is interesting to note that, on "fair-ground" comparison, that is if our decoder do not have the benefit of a more compact lattice-rule representation, it actually perform quite worse as we can see by comparing with the third column of table 1 (at least in term of decoding time and memory usage, while it would still have a very slight edge in term of model score with the selected settings). On the other hand, the K-decoder is a rather strong baseline, shown to perform several times faster than a previous state-of-the-art implementation in (Heafield et al., 2013). It is well opti-

[4] http://kheafield.com/code/search/

Representation:	**Original**	**Optimized**	Expanded
Peak memory used	39 GB	**16GB**	85GB
Average search time	6.13s	**3.31s**	9.95s
#vertices (avg/max)	65K (1300K)	**32K (446K)**	263K (5421K)
#edges (avg/max)	92K (1512K)	**83K (541K)**	263K (5421K)

Table 1: Impact of the lattice representation on performances.

System	**JA–EN**
Lattice	**29.43**
No-variations	28.91
Moses (for scale)	28.86

Table 2: Impact on BLEU of using flexible lattice rules.

mized and makes use of advanced techniques with the language model (as the one described in (Heafield et al., 2013)) for which we do not have implemented an equivalent yet. Therefore, we are hopeful we can further improve our decoder in the future.

Also, note that, for practical reason, while we only measured the decoding time for our decoder [5], the K-decoder time include the time taken for loading the rule files.

6.3 Translation quality

Finally, we evaluate the advantages of extracting lattice rules such as proposed in section 3. That is, we consider rules for which null-aligned words are bypassable by epsilon-edges, for which Non-terminal are allowed to take several alternative positions around the word that is thought to be their governor, and for which we consider alternative morphologies of a few words ("is/are", "a/an"). We compare this approach with heuristically selecting only one possibility for each variation present in the lattice rule extracted from a single example.

Results shown on figure 2 show that we do obtain a significant improvement in translation quality. Note that the Moses score (Koehn et al., 2007), taken from the official results of NTCIR-10 is only here "for scale", as our MT system uses a quite different pipeline.

[5] in particular, we factored out the representation optimisation time, which is reasonable if we are in the setting of a parameter tuning step in which the same sentences are translated repeatedly

7 Related work

Searching for the most optimal translation in an implicitly defined set has been the focus of a lot of research in Machine Translation and it would be difficult to cover all of it. Among the most influential approaches, (Koehn et al., 2003) was using a form of stack based decoding for Phrase-Based Machine Translation. (Chiang, 2007) introduced the cube-pruning approach, which has been further improved in the previously mentioned (Heafield et al., 2013). (Rush and Collins, 2011) recently proposed an algorithm promising to find the optimal solution, but that is rather slow in practice.

Weighted Finite State Machines have seen a variety of use in NLP (Mohri, 1997). More specifically, some other previous work on Machine Translation have used lattices (or more generally Weighted Finite State Machines). In the context of Corpus-Based Machine Translation, (Knight and Al-Onaizan, 1998) was already proposing to use Weighted Transducers to decode the "IBM" models of translation (Brown et al., 1993). (Casacuberta and Vidal, 2004) and (Kumar et al., 2006) also propose to directly model the translation process with Finite State Transducers. (Graehl and Knight, 2004) propose to use Tree Transducers for modeling Syntactic Machine Translation. These approaches are however based on different paradigm, typically trying to directly learn a transducer rather than extracting SCFG-like rules.

Closer to our context, (de Gispert et al., 2010) propose to use Finite-State Transducers in the context of Hierarchical Phrase Based Translation. Their method is to iteratively construct and minimize the full "top-level lattice" representing the whole set of translations bottom-up. It is an approach more focused on the Finite State Machine aspect than our,

System	K-decoder	Lattice-rule decoder
Peak memory used	52G	**16G**
Average search time	3.47s	**3.31s**
Average model score	-107.55	**-107.39**
Nb wins	401	**579**

Table 3: Evaluation of the performances of our lattice-rule decoder compared with a state-of-the-art decoder using an expanded flat representation of the lattice rules. "Nb wins" is the number of times one of the decoder found a strictly better model score than the other one, out of 1800 search.

which is more of an hybrid approach that stays closer to the paradigm of cube-pruning. The merit of their approach is that they can apply minimization globally, allowing for more possibilities for vertex merging. On the other hand, for large grammars, the "top-level lattice" will be huge, creating the need to prune vertices during the construction. Furthermore, the composition of the "top-level lattice" with a language model will imply redundant computations (as lower-level lattices will potentially be expanded several times in the top-level lattice). As we do not construct the global lattice explicitly, we do not need to prune vertices (we only prune language model states). And each edge of each lattice rule is crossed only once during our decoding.

Very recently, (Heafield et al., 2014) also considered using the redundancy of translation hypotheses to optimize phrase-based stack decoding. To do so, they group the partial hypotheses in a trie structure.

We are not aware of other work proposing "lattice rules" as a native format for expressing translational equivalences. Work like (de Gispert et al., 2010) rely on SCFG rules created along the (Chiang, 2007) approach, while work like (Casacuberta and Vidal, 2004) adopt a pure Finite State Transducer paradigm (thus without explicit SCFG-like rules).

8 Conclusion

This work proposes to use a lattice-rule representation of the translation search space with two main goals:

- Easily represent the translation ambiguities that arise either due to lack of context or imperfect knowledge.
- Have a method for optimizing the representation of a search space to make this search more efficient.

We demonstrate that many types of ambiguities arising when extracting translation rules can easily be expressed in this framework, and that making these ambiguities explicit and solvable at compile time through lattice-rules leads to improvement in translation quality.

We also demonstrate that making a direct use of the lattice-rules representation allows a decoder to perform better than if working on the expanded set of corresponding "flat rules". And we propose an algorithm for computing more efficient representations of a translation candidate set.

We believe that the the link between the representation of a candidate set and the decoding efficiency is an interesting issue and we intend to explore further the possibilities of optimizing representations both in the contexts we considered in this paper and in others such as Phrase-Based Machine Translation.

The code of the decoder we implemented for this paper is to be released under a GPL license[6].

Acknowledgements

This work is supported by the Japanese Science and Technology Agency. We want to thank the anonymous reviewers for many very useful comments.

References

Peter F Brown, Vincent J Della Pietra, Stephen A Della Pietra, and Robert L Mercer. 1993. The mathematics of statistical machine translation:

[6] http://nlp.ist.i.kyoto-u.ac.jp/kyotoebmt/

Parameter estimation. *Computational linguistics*, 19(2):263–311.

Francisco Casacuberta and Enrique Vidal. 2004. Machine translation with inferred stochastic finite-state transducers. *Computational Linguistics*, 30(2):205–225.

David Chiang. 2007. Hierarchical phrase-based translation. *computational linguistics*, 33(2):201–228.

Adrià de Gispert, Gonzalo Iglesias, Graeme Blackwood, Eduardo R Banga, and William Byrne. 2010. Hierarchical phrase-based translation with weighted finite-state transducers and shallow-n grammars. *Computational linguistics*, 36(3):505–533.

Isao Goto, Ka Po Chow, Bin Lu, Eiichiro Sumita, and Benjamin K Tsou. 2013. Overview of the patent machine translation task at the ntcir-10 workshop. In *Proceedings of the 10th NTCIR Workshop Meeting on Evaluation of Information Access Technologies: Information Retrieval, Question Answering and Cross-Lingual Information Access, NTCIR-10*.

Jonathan Graehl and Kevin Knight. 2004. Training tree transducers. In *Proceedings of HLT-NAACL*.

Kenneth Heafield, Hieu Hoang, Philipp Koehn, Tetsuo Kiso, and Marcello Federico. 2011. Left language model state for syntactic machine translation. In *Proceedings of the International Workshop on Spoken Language Translation*, pages 183–190, San Francisco, California, USA, December.

Kenneth Heafield, Philipp Koehn, and Alon Lavie. 2012. Language model rest costs and space-efficient storage. In *Proceedings of the Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1169–1178, Jeju Island, Korea, July.

Kenneth Heafield, Philipp Koehn, and Alon Lavie. 2013. Grouping language model boundary words to speed k-best extraction from hypergraphs. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 958–968, Atlanta, Georgia, USA, June.

Kenneth Heafield, Michael Kayser, and Christopher D. Manning. 2014. Faster Phrase-Based decoding by refining feature state. In *Proceedings of the Association for Computational Linguistics*, Baltimore, MD, USA, June.

Kenneth Heafield. 2011. KenLM: faster and smaller language model queries. In *Proceedings of the EMNLP 2011 Sixth Workshop on Statistical Machine Translation*, pages 187–197, Edinburgh, Scotland, United Kingdom, July.

John Hopcroft. 1971. An n log n algorithm for minimizing states in a finite automaton. *Theory of Machines and Computations*, pages 189–196.

Liang Huang and David Chiang. 2005. Better k-best parsing. In *Proceedings of the Ninth International Workshop on Parsing Technology*, pages 53–64. Association for Computational Linguistics.

Kevin Knight and Yaser Al-Onaizan. 1998. Translation with finite-state devices. In *Machine translation and the information soup*, pages 421–437. Springer.

Philipp Koehn, Franz Josef Och, and Daniel Marcu. 2003. Statistical phrase-based translation. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology-Volume 1*, pages 48–54. Association for Computational Linguistics.

Philipp Koehn, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, et al. 2007. Moses: Open source toolkit for statistical machine translation. In *Proceedings of the 45th Annual Meeting of the ACL on Interactive Poster and Demonstration Sessions*, pages 177–180. Association for Computational Linguistics.

Shankar Kumar, Yonggang Deng, and William Byrne. 2006. A weighted finite state transducer translation template model for statistical machine translation. *Natural Language Engineering*, 12(01):35–75.

Zhifei Li and Sanjeev Khudanpur. 2008. A scalable decoder for parsing-based machine translation with equivalent language model state maintenance. In *Proceedings of the Second Workshop on Syntax and Structure in Statistical Translation*, pages 10–18. Association for Computational Linguistics.

Mehryar Mohri. 1997. Finite-state transducers in language and speech processing. *Computational linguistics*, 23(2):269–311.

Mehryar Mohri. 2004. Weighted finite-state transducer algorithms. an overview. In *Formal Languages and Applications*, pages 551–563. Springer.

John Richardson, Fabien Cromières, Toshiaki Nakazawa, and Sadao Kurohashi. 2014. Kyotoebmt: An example-based dependency-to-dependency translation framework. In *Proceedings of ACL (System Demonstration)*, Baltimore, MD, USA, June.

Alexander M Rush and Michael Collins. 2011. Exact decoding of syntactic translation models through lagrangian relaxation. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 72–82. Association for Computational Linguistics.