Handling Complexity in Decoding for Statistical MT

Christoph Tillmann
IBM T.J. Watson Research Center
Yorktown Heights, N.Y. 10598
ctill@us.ibm.com

1 Introduction

This paper presents several extensions of a commonly used phrase-based decoder for SMT (Koehn, 2004; Och and Ney, 2004) [1]. Such decoders are widely used within the GALE project, and the current paper demonstrates ways of making these decoders more efficient in a principled way. It follows work in (Tillmann and Ney, 2003) where the word re-ordering problem in SMT is linked to the traveling salesman problem (TSP) . In that paper, a DP-based (dynamic programming-based) optimization algorithm to solve the TSP (Held and Karp, 1962) serves as a starting point to handle the word re-ordering problem efficiently. The Held-Karp algorithm is a special case of a shortest path finding algorithm in a directed acyclic graph (DAG) (Dreyfus and Law, 1977). Following this line of argumentation, the present paper spells out an approach that handles SMT decoding as a shortest path finding problem in dag's in some detail.

The translation model used in this paper is a so-called block-based model. A block is a pair of phrases which are translations of each other. During decoding, we view translation as a block segmentation process, where the input sentence is segmented from left to right and the target sentence is generated one phrase at a time. For some language pairs, close to state-of-the-art performance can be obtained by generating a largely monotone block sequence. A block sequence is scored as follows:

$$s_w(b_1^n) \quad = \quad \sum_{i=1}^{n} w^T \cdot f(b_i, b_{i-1}) \,. \qquad (1)$$

We try to find a block sequence b_1^n that minimizes $s_w(b_1^n)$ under the restriction that the concatenated source phrases of the blocks b_i yield a segmentation of the input sentence. We use the following block bigram scoring : a block pair $(b; b')$ is represented as a feature-vector $f(b; b') \in \mathbb{R}^N$. w is the feature weight vector which is trained on some development set. The feature-vector components are the negative logarithm of some probabilities . Under this view, SMT becomes quite similar to sequential natural language annotation problems such as part-of-speech tagging, phrase chunking, and shallow parsing [2].

This paper is structured as follows: Section 2 introduces two types of beam-search decoder: a single-beam and a multi-beam decoder. Section 3 shows how these algorithms can be analyzed in terms of a shortest path finding algorithm in a DAG. Section 4 presents a decoder version where re-ordering with a large window is controlled efficiently by POS-based re-order rules. In these sections, citations of empirical work showing the effect of the novel search algorithms are given, but details are omitted for brevity reasons. Section 5 presents an efficient DP-based alignment algorithm . Finally, Section 6 discusses some future application of the presented algorithms.

[1] The work presented in this paper is a juxtaposition of work that has been published previously in (Tillmann, 2001; Tillmann and Ney, 2003; Tillmann, 2006; Tillmann, 2008).

[2] Since standard phrase-based decoders rely mostly on locally computed feature functions, the decoding algorithms presented in this paper are applicable to them as well.

2 Beam-Search Decoder

In DP-based decoders for SMT, the search space is dynamically constructed: suitably defined lists of path hypotheses are maintained. This section focuses on some details how these path hypotheses are managed: using two stacks or multiple stacks. The standard phrase-based decoder in (Koehn, 2004) uses J hypotheses stacks, one stack for each source position in the input sentence. The algorithm finds the Viterbi block translation for the model defined in Eq. 1. Search states are 3-tuples of the following type:

$$[\, \mathcal{C} \;,\; h \;;\; d \,] \,. \tag{2}$$

$h = ([j, j'], [u, v])$ is the state history, where $([j, j'])$ is the interval where the most recent source phrase matched the input sentence, and $[u, v]$ are the final two target words of the partial translation produced so far. d is the partial translation cost. Here, we use a coverage vector $\mathcal{C}$ to ensure that each source position is covered exactly once. That way, we make sure that a consistent alignment is produced. By including a coverage vector $\mathcal{C}$ into the search state definition we obtain an inherently exponential complexity: for an input sentence of length J there are 2^J coverage vectors (Koehn, 2004). By restricting the coverage vector appropriately, the decoding can be carried out efficiently as shown in Section 3. When a state of the type in Eq. 2 is extended by some additional phrase match, a local transition cost Δ is computed. The partial cost d' of the new state is computed as $d' = d + \Delta$. If the original state covers k source positions, and the latest source phrase match is l words long, the new hypothesis is added into the $(k + l)$-th stack.

The second implementation uses two stacks to keep a single beam of active states. This corresponds to a beam-search decoder in speech recognition, where path hypotheses that correspond to word sequences are processed in a time-synchronous way. At a given time step the decoder keeps only hypotheses within some percentage of the best hypothesis (Lowerre and Reddy, 1980; Ney et al., 1992). The state definition in Eq. 3 includes an additional field l that keeps track of how much of the most recent source phrase match has been covered.

$$[\mathcal{C} \;,\; l \;,\; h \;;\; d] \tag{3}$$

When a phrase is matched to the input sentence, the pointer l is set to position j where the most recent block match starts. As long as the pointer l is not yet equal to the match end position j', it is increased: $l' := l + 1$. Here, the single-beam decoder processes hypotheses *cardinality-synchronously*, i.e. states that cover k source positions generate new states that cover $k + 1$ positions. The local transition cost Δ' is spread evenly over the span of the latest match of length l: $\Delta' = \Delta / l$. The algorithm stores the states in only two stacks Γ and Γ': Γ contains the most probable hypotheses that were kept in the last beam pruning step all of which cover k source positions. Γ' contains all the newly generated hypotheses before they are pruned in the next pruning step.

Before expanding a state set, states are pruned based on their coverage vector and the path cost : two different pruning strategies are used that have been introduced in (Tillmann and Ney, 2003): 1) *coverage pruning* prunes states that share the same coverage vector $\mathcal{C}$, 2) *cardinality pruning* prunes states according to the number of covered positions k: all states in the beam are compared with each other. The size of the beam depends locally on the number of hypotheses whose score is close to the top scoring one. In particular the cardinality pruning becomes an efficient pruning strategy in the case of the two-stack decoder: all active hypotheses are compared against each other. Adding a state to a stack includes adding the state if it is not yet present or updating its shortest path cost d if the state is already in that stack. This operation is also called *recombination*. It is a way to prune the stack safely without the risk of additional search errors. The recombination becomes very efficient if the coverage vector is restricted as shown in the next section. The multiple stack and two stack algorithm search slightly different search graphs where the single-beam algorithm with the more 'regularly' structured search graph results in a more efficient pruning strategy . The results in (Tillmann, 2006) show that using two stacks results in a run-time reduction of a factor of 2 when compared to a standard decoder for phrase-based SMT that uses a single stack for each source

position.

3 Shortest Path Algorithm for dag

This section compares the phrase decoding algorithms to a shortest path finding algorithm for directed acyclic graphs (dag). A dag $G = (V, E)$ is a weighted graph for which a topological sort of its vertex set V exists: all the vertexes can be enumerated in linear order. For such a weighted graph, we can compute the shortest path from a single source in $O(|V| + |E|)$ time, where $|V|$ is the number of vertexes and $|E|$ number of edges in the graph.

Here, vertexes in the dag correspond to the states defined in Eq. 2, and edges correspond to transitions between states. The vertex set is finite since there is only a finite number of states. The corresponding graph contains no loops by definition: state transitions exist only from a state that covers fewer source positions to a successor state that covers additional (more) source positions. The fact that states are generated by increasing coverage cardinality can be understood as computing a topological sort of the vertexes on the fly. The analysis in terms of a dag shortest path algorithm can be used for a simple complexity analysis of the proposed algorithms. Local state transitions correspond to an edge traversal in the dag search algorithm. These involve costly look-up operations, e.g. language, distortion and translation model probability look-up. Typically the computation time for update operations on lists Γ is negligible compared to these probability lockup's. That way, the search algorithm's complexity is simply computed as the number of edges in the search graph: $O(|V| + |E|) \approx O(|E|)$. Such an analysis has been presented in (Tillmann, 2001) for a series of re-ordering restrictions. In the case that the number of holes in the coverage vector for a left-to-right traversal of the input sentence is bounded by a constant k, e.g. $k = 3$, (Tillmann, 2001) gives a proof that the complexity for the resulting search algorithm is bounded by J^{k+1} where J is the length of the input sentence (the complexity due to the use of a language model or distortion model is ignored) [3] .

[3]The analysis in (Tillmann, 2001) is carried out for a word-based search where search states are of the form $[\mathcal{C}, j]$, and j denotes the last covered source position j. The search state definition for a phrase-based decoder might result in a slightly higher complexity bound, but a dag complexity analysis remains feasible.

It is interesting to note that these complexity bounds are achieved without the typical re-ordering window size restriction. Designing algorithms with provable upper complexity bounds might be a way to obtain more efficient decoding algorithm in the future.

The stack-based decoding algorithms can also be compared to an Earley-style parsing algorithm that processes lists of parse states in a single left-to-right run over the input sentence. That parsing algorithm can also be analyzed in terms of a dag (Jurafsky and Martin, 2000). Furthermore, the use of the pointer l in the state definition in Eq. 3 is related to the use of the so-called dotted rules in the Early parser states.

4 POS-based Re-order Rules

The paper sketches an extension of the beam-search decoder introduced in Section 2, where POS-based re-order rules (Crego and Marino, 2006) are tightly integrated into a single-beam left-to-right search. Re-order rules are typically applied to the source and/or target sentence as a pre-processing step (Xia and McCord, 2004; Collins et al., 2005; Paulik et al., 2007) A few dozen re-order rules generate a so-called re-ordering graph. In comparison, the novel rule-driven decoder can handle about $30\,000$ rules efficiently. The algorithm described in this paper uses an intermediate data structure called an *edge* that represents a source phrase together with its target phrase translation (A similar data structure is called *translation option* in (Koehn, 2004)). For each input interval that is matched by some source phrase, we store a list of possible target phrase translations as edges in a *chart* . Additionally, we replace the re-order graph generation by a simpler edge generation process which involves only local computation. To formalize the approach, the search state definition in Eq. 2 is modified as follows:

$$[\, s\, ;\, [i, j]\, , r\, ,\, s_r\, ,\, e \in \{\textbf{false}, \textbf{true}\}\,] \qquad (4)$$

Here, the coverage vector $\mathcal{C}$ is replaced by a single number s: a monotone search is carried out and all the source positions up to position s (including s) are covered. $[i, j]$ is the coverage interval for the last source phrase translated (the same as in Eq. 2). r is the rule identifier., s_r is the starting position for the match of the r-th rule in the input sentence, and e

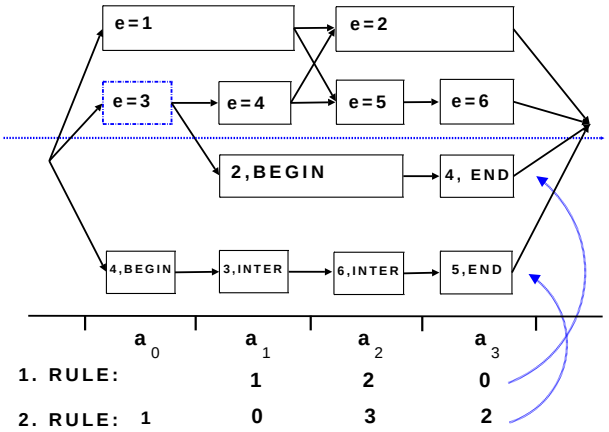

Figure 1: Addition of rule edges to a chart containing 6 simple edges. Rule application results in the generation of special rule edges. These edges are processed in an order consistent with the rule.

is a flag that indicates whether the hypothesis h has covered the entire span of rule r yet [4].

States are extended by finding matching edges: simple or rule edges. We illustrate the generation of these edges in Fig. 1 for the use of 2 overlapping rules on a source segment of 4 words $a_0, \cdots, a_3$. The two rules correspond to matching POS sequences, i.e. the input sentence has been POS tagged and a POS tag p_j has been assigned to each word a_j. Edges are shown as rectangles. In the top half of the picture, 6 simple edges which correspond to 6 phrase-to-phrase translations are enumerated. In the bottom half of Figure 1, we show the newly generated rule edges . A rule edge is generated from a simple edge via a re-order rule application, and we add it into the chart. Here, rule 1 generates two new edges and rule 2 generates four new edges . For a consistent processing of the rule edges within the rule match interval, they are assigned one out of three positions: *BEGIN*, *INTER(mediate)*, and *END*. We carry out the edge generation process in the following way: for each source interval $[k, l]$ all the matching phrase pairs are found and added into the chart as simple edges. In a second run over the input sentence for each source interval $[k, l]$ all matching POS sequences are computed and the corresponding source words $a_k, \cdots, a_l$ are re-ordered according to the rule permutation. On the re-ordered word sequence phrase matches are computed and added into the chart. This edge generation is computationally 'light-weight' and typically takes only a few percent of the overall decoding time. Finally, the rule-driven decoder searches through the dag defined by both simple and rule edges to find the best scoring block translation sequence.

[4]For technical details, see (Tillmann, 2008).

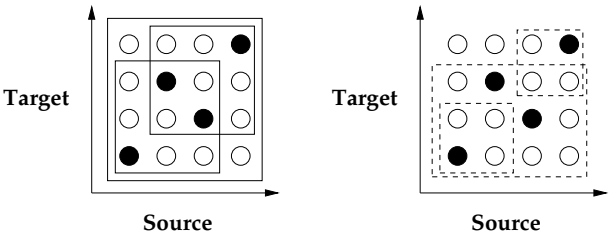

Figure 2: The left picture shows three **consistent** phrase pairs while the right picture shows three **non-consistent** phrase pairs.

It is important to point out that the coverage vector component has been removed altogether from the search state definition in Eq. 4. This way, the rule-driven search search results in an essentially linear time decoding algorithm by 'indexing' the search space according to the rules and making extended use of the basic edge data structure. The rule-driven decoder uses the two stack implementation shown in Section 2 . Here, we use an additional probabilistic feature which is derived from the rule unigram count $N(r)$. In (Tillmann, 2008), the rule-driven decoder is compared to a standard phrase-based decoder. It is shown that decoding can be carried out efficiently and reliably with a large re-ordering window of 15 words: a small improvement in terms of BLEU is reported for a standard Arabic-English translation task. In addition, the rule-driven decoder handles a huge set of about 30 000 rules efficiently.

5 Efficient Block Segmentation Algorithm

A common approach to phrase-based SMT is to learn phrasal translation pairs from word-aligned training data. In this case, one uses a symmetrization step (Och and Ney, 2004): a word-alignment is computed for both translation directions source-to-target and target-to-source, and the intersection of these two alignments is computed to obtain a high-precision word alignment. Here, we note that if this intersection covers all source and target posi-

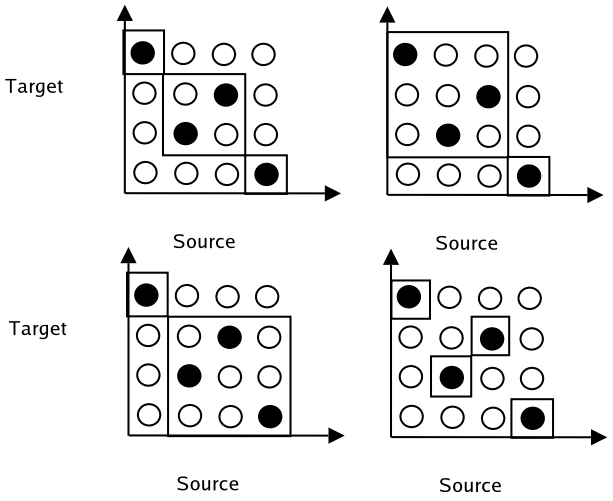

Figure 3: Four block segmentations of a sentence pair if the intersection is a bijection. The segmentation that covers the whole sentence pair with a single block is not shown.

Table 1: Efficient DP-based block segmentation algorithm using an underlying intersection word alignment $\mathcal{A}$.

input: Parallel sentence pair and alignment $\mathcal{A}$.
initialization : $Q_b(i) = -\infty$ for all $i = 0, \cdots, I$ and blocks b . Sentence initial cost $Q_{\hat{b}}(0) = 0.0$.
for each $i = 1, 2, \cdots, I$ do
  for each $i' = 0, 1, 2, \cdots, i-1$ do
    Project interval $T = [i'+1, i]$ and check block link (T, S) for consistency .
    if (T, S) is CONSISTENT
      Set block $b = (\text{words in } T, \text{words in } S)$
      $Q_b(i) = \max_{b'} \Delta(b, b') + Q_{b'}(i')$
recover optimal segmentation b_1^n **:**
  - find best end hypothesis: $\max_b Q_b(I)$

tions, then it constitutes a bijection between source and target sentence positions [5]. Additionally, (Och and Ney, 2004) defines the notion of *consistency* for the set of phrasal translations that are learned from word-aligned training data: when projecting source intervals to target intervals, and target intervals to source intervals no alignment link may lie outside the resulting phrase link. In Fig. 2, we show examples of consistent and inconsistent phrase pairs . The word alignment for that sentence pair links $J = 4$ source and $I = 4$ target words. The alignment links that yield a bijection are shown as solid dots. When the alignment $\mathcal{A}$ is a bijection, by definition each target position i is aligned to exactly one source position j and vice versa and source and target sentence have the same length.

We show that an efficient DP-based phrase alignment algorithm exists if the phrase pairs involved are consistent with an underlying alignment bijection. Without an underlying alignment $\mathcal{A}$, each pair of source interval S and target interval T defines a possible phrase link and this unrestricted phrase alignment problem has been shown to be NP-complete (Nero and Klein, 2008). The consistency restriction drastically reduces the number of phrase pair links.

[5](Tillmann, 2003) reports an intersection coverage of about 65 % for Arabic-English data, and a coverage of 40 % for Chinese-English data under real conditions.

Source and target sentence segmentations are linked in a perfect way. Because of the bijection and the consistency restriction, a target interval segmentation alone is sufficient to determine the source interval segmentation **and** the phrase alignment simultaneously. This close link is illustrated in Fig. 3. The DP algorithm that makes use of the observation above uses the following auxiliary quantity :

$Q_b(i)$:= score of the best partial segmentation that covers the target interval $[1, i]$ and ends in block b.

The DP-based algorithm to compute $Q_b(i)$ is shown in Table 1. A block segmentation is guaranteed to be found: the block that covers all source and target position is consistent by definition. Target intervals are processed from bottom to top. A target interval $T = [i', i]$ is projected using the word alignment $\mathcal{A}$, where a given target interval might not yield a consistent phrase pair, e.g. in Fig. 3, out of all eight possible target segmentations, only five yield segmentations with consistent block links. For the initialization, we set $Q_b(i) = -\infty$ for all target positions $i = 0, \cdots, I$ and blocks b and $Q_{\hat{b}}(0) = 0.0$, where $\hat{b}$ is a special starting block positioned at the sentence beginning. The final score is obtained as $Q_{Final} = \max_b Q_b(I)$, where I is the length of the target sentence. $\Delta(b, b')$ denotes a transition cost for placing consecutive blocks which might use the

same baseline features as in Section 1. Depending on the features that are being used, the auxiliary quantity in Eq. 5 needs to be adjusted , e.g. by including source matching intervals to handle distortion features. The complexity of the algorithm is $\mathcal{O}(|B| \cdot I^2)$ assuming that the transition cost $\Delta(b, b')$ between adjacent blocks can be computed in constant time. Here, $|B|$ denotes the number of blocks that can be extracted from the aligned sentence pair. The presented DP-based algorithm is interesting theoretically because it solves a NP-complete problem efficiently (under the consistency restriction) for arbitrary long sentence pairs and unrestricted phrase-level re-ordering.

6 Discussion

Future work might bring the DP algorithms in Section 2 and Section 5 together: if the alignment intersection covers a source and target sentence only partially , the number of consistent phrase pairs is increased significantly. A stack-based search can be used to find a phrase alignment efficiently under the consistency restriction. Such an algorithm might be helpful in training a phrase-based system discriminatively. The single-pass, left-to-right beam search algorithms presented in Section 2 and Section 4 might be extended to decode with hierarchical model parameters . Under appropriate restrictions, this might result in a more efficient decoding algorithm when compared to a standard chart-based decoder.

References

Michael Collins, Philipp Koehn, and Ivona Kucerova. 2005. Clause restructuring for statistical machine translation. In *Proc. of ACL'05*, pages 531–540, Ann Arbor, Michigan, June. Association for Computational Linguistics.

J.M. Crego and José B. Marino. 2006. Integration of POStag-based Source Reordering into SMT Decoding by an Extended Search Graph. In *Proc. of AMTA06*, pages 29–36, Cambridge, MA, August.

Stuart E. Dreyfus and Averill M. Law. 1977. *The Art and Theory of Dynamic Programming (Mathematics in Science and Engineering; vol. 130)*. Academic Press, New York, N.Y.

Held and Karp. 1962. A Dynamic Programming Approach to Sequencing Problems. *SIAM*, 10(1):196–210.

Daniel Jurafsky and James H. Martin. 2000. *Speech and Language Processing*. Prentice Hall.

Philipp Koehn. 2004. Pharaoh: a Beam Search Decoder for Phrase-Based Statistical Machine Translation Models. In *Proceedings of AMTA'04*, Washington DC, September-October.

Bruce Lowerre and Raj Reddy. 1980. *The Harpy speech understanding system, in Trends in Speech Recognition, W.A. Lea, Ed.* Prentice Hall, EngleWood Cliffs, NJ.

John De Nero and Dan Klein. 2008. The Complexity of Phrase Alignment Problems. In *Companion Vol. of ACL-08*, pages 25–28, Columbus, OH, June.

Hermann Ney, Dieter Mergel, Andreas Noll, and Annedore Paeseler. 1992. Data Driven Search Organization for Continuous Speech Recognition in the SPICOS System. *IEEE Transaction on Signal Processing*, 40(2):272–281.

Franz-Josef Och and Hermann Ney. 2004. The Alignment Template Approach to Statistical Machine Translation. *Computational Linguistics*, 30(4):417–450.

Matthias Paulik, Kay Rottmann, Jan Niehues, Silja Hildebrand, and Stephan Vogel. 2007. The ISL Phrase-Based MT System for the 2007 ACL Workshop on SMT. *In Proc. of the ACL 2007 Second Workshop on SMT*, June.

Christoph Tillmann and Hermann Ney. 2003. Word Reordering and a DP Beam Search Algorithm for Statistical Machine Translation. *CL*, 29(1):97–133.

Christoph Tillmann. 2001. *Word Re-Ordering and Dynamic Programming based Search Algorithm for Statistical Machine Translation*. Ph.D. thesis, University of Technology, Aachen, Germany.

Christoph Tillmann. 2003. A Projection Extension Algorithm for Statistical Machine Translation. In *Proceedings of EMNLP'03*, pages 1–8, Sapporo, Japan, July.

Christoph Tillmann. 2006. Efficient Dynamic Programming Search Algorithms for Phrase-based SMT. In *Proceedings of the Workshop CHPSLP at HLT'06*, pages 9–16, New York City, NY, June.

Christoph Tillmann. 2008. A rule-driven dynamic programming decoder for statistical MT. In *Proceedings of the ACL-08: HLT Second Workshop on Syntax and Structure in Statistical Translation (SSST-2)*, pages 37–45, Columbus, Ohio, June.

Fei Xia and Michael McCord. 2004. Improving a statistical mt system with automatically learned rewrite patterns. In *Proc. of Coling 2004*, pages 508–514, Geneva, Switzerland, Aug 23–Aug 27. COLING.