Efficient Dynamic Programming Search Algorithms for Phrase-Based SMT

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

Abstract

This paper presents a series of efficient dynamic-programming (DP) based algorithms for phrase-based decoding and alignment computation in statistical machine translation (SMT). The DP-based decoding algorithms are analyzed in terms of shortest path-finding algorithms, where the similarity to DP-based decoding algorithms in speech recognition is demonstrated. The paper contains the following original contributions: 1) the DP-based decoding algorithm in (Tillmann and Ney, 2003) is extended in a formal way to handle phrases and a novel pruning strategy with increased translation speed is presented 2) a novel alignment algorithm is presented that computes a phrase alignment efficiently in the case that it is consistent with an underlying word alignment. Under certain restrictions, both algorithms handle MT-related problems efficiently that are generally NP complete (Knight, 1999).

1 Introduction

This paper deals with dynamic programming based decoding and alignment algorithms for phrase-based SMT. Dynamic Programming based search algorithms are being used in speech recognition (Jelinek, 1998; Ney et al., 1992) as well as in statistical machine translation (Tillmann et al., 1997; Niessen et al., 1998; Tillmann and Ney, 2003). Here, the decoding algorithms are described as shortest path finding algorithms in regularly structured search graphs or search grids. Under certain restrictions, e.g. start and end point restrictions for the path, the shortest path computed corresponds to a recognized word sequence or a generated target language translation. In these algorithms, a shortest-path search

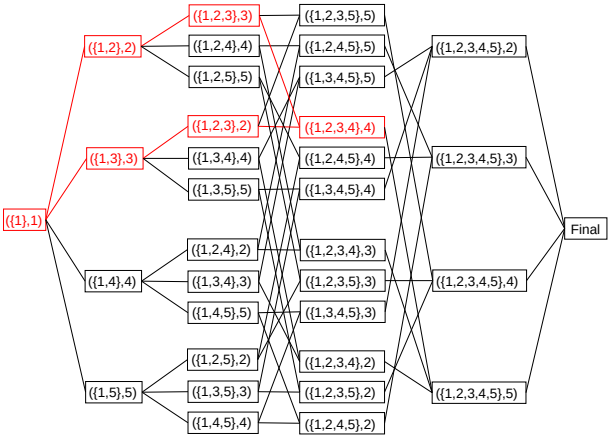

Figure 1: Illustration of a DP-based algorithm to solve a traveling salesman problem with 5 cities. The visited cities correspond to processed source positions.

is carried out in one pass over some input along a specific 'direction': in speech recognition the search is time-synchronous, the single-word based search algorithm in (Tillmann et al., 1997) is (source) position-synchronous or left-to-right, the search algorithm in (Niessen et al., 1998) is (target) position-synchronous or bottom-to-top, and the search algorithm in (Tillmann and Ney, 2003) is so-called cardinality-synchronous.

Taking into account the different word order between source and target language sentences, it becomes less obvious that a SMT search algorithm can be described as a shortest path finding algorithm. But this has been shown by linking decoding to a dynamic-programming solution for the traveling salesman problem. This algorithm due to (Held and Karp, 1962) is a special case of a shortest path finding algorithm (Dreyfus and Law, 1977). The regularly structured search graph for this problem is illustrated in Fig. 1: all paths from the left-most to the right-most vertex correspond to a translation of the in-

Workshop on Computationally Hard Problemsand Joint Inference in Speech and Language Processing, pages 9–16,
New York City, New York, June 2006.

put sentence, where each source position is processed exactly once. In this paper, the DP-based search algorithm in (Tillmann and Ney, 2003) is extended in a formal way to handle phrase-based translation. Two versions of a phrase-based decoder for SMT that search slightly different search graphs are presented: a multi-beam decoder reported in the literature and a single-beam decoder with increased translation speed [1]. A common analysis of all the search algorithms above in terms of a shortest-path finding algorithm for a directed acyclic graph (**dag**) is presented. This analysis provides a simple way of analyzing the complexity of DP-based search algorithm.

Generally, the regular search space can only be fully searched for small search grids under appropriate restrictions, i.e. the monotonicity restrictions in (Tillmann et al., 1997) or the inverted search graph in (Niessen et al., 1998). For larger search spaces as are required for continuous speech recognition (Ney et al., 1992) [2] or phrase-based decoding in SMT, the search space cannot be fully searched: suitably defined lists of path hypothesis are maintained that partially explore the search space. The number of hypotheses depends locally on the number hypotheses whose score is close to the top scoring hypothesis: this set of hypotheses is called the beam.

The translation model used in this paper is a phrase-based model, where the translation units are so-called blocks: a block is a pair of phrases which are translations of each other. For example, Fig. 2 shows an Arabic-English translation example that uses 5 blocks. 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 from bottom to top, one block at a time. In practice, a largely monotone block sequence is generated except for the possibility to swap some neighbor blocks. During decoding, we try to minimize the score $s_w(b_1^n)$ of a block sequence b_1^n under the restriction that the concatenated source phrases of the blocks b_i yield a segmentation of the input sentence:

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

Here, $f(b_{i-1}, b_i)$ is 7-dimensional feature vector with real-valued features and w is the corresponding weight vector as described in Section 5. The fact that a given block covers some source interval $[j', j]$ is implicit in this notation.

[1] The multi-beam decoder is similar to the decoder presented in (Koehn, 2004) which is a standard decoder used in phrase-based SMT. A multi-beam decoder is also used in (Al-Onaizan et al., 2004) and (Berger et al., 1996).

[2] In that work, there is a distinction between within-word and between-word search, which is not relevant for phrase-based decoding where only exact phrase matches are searched.

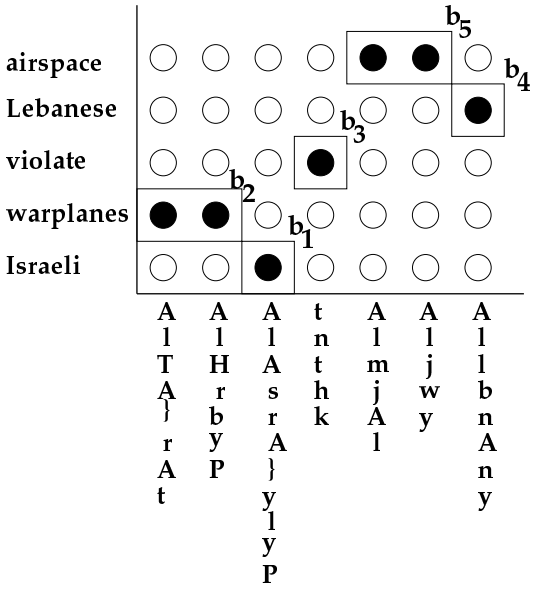

Figure 2: An Arabic-English block translation example, where the Arabic words are romanized. A sequence of 5 blocks is generated.

This paper is structured as follows: Section 2 introduces the multi-beam and the single-beam DP-based decoders. Section 3 presents an analysis of all the graph-based shortest-path finding algorithm mentioned above: a search algorithm for a directed acyclic graph (**dag**). Section 4 shows an efficient phrasal alignment algorithm that gives an algorithmic justification for learning blocks from word-aligned training. Finally, Section 5 presents an evaluation of the beam-search decoders on an Arabic-English decoding task.

2 Beam-Search Decoding Algorithms

In this section, we introduce two beam-search algorithms for SMT: a multi-beam algorithm and single-beam algorithm. The multi-beam search algorithm is presented first, since it is conceptually simpler.

2.1 Multi-Beam Decoder

For the multi-beam decoder makes use of search states that are 3-tuples of the following type:

$$[\mathcal{C}\ ,\ h\,;\ d\,]\,. \quad (2)$$

h is the state history, that depends on the block generation model. In our case, $h = ([j, j'], [u, v])$, where $([j, j'])$ is the interval where the most recent block matched the input sentence, and $[u, v]$ are the final two target words of the partial translation produced thus far. $\mathcal{C}$ is the so-called coverage vector that ensures that a consistent block alignment is obtained during decoding and that the decoding

Table 1: Multi-beam (M-**Beam**) decoding algorithm, which is similar to (Koehn, 2004). The decoders differ in their pruning strategy: here, each state list Γ_c is pruned only once, whereas the decoder in (Koehn, 2004) prunes a state list every time a new hypothesis is entered.

input: source sentence with words $f_1, \cdots, f_J$

$\Gamma_0 := \{\sigma_0\}$ and $\Gamma_k := \emptyset$ for $k = 1, \cdots, J$
for each $c = 0, 1, \cdots, J$ **do**
  Prune state set Γ_c
  for each state σ in Γ_c **do**
  matcher: for each $\sigma' : \sigma \rightarrow_M \sigma'$
  update σ' for $\Gamma_{c+l(\sigma')}$
  end
end

output: translation from lowest cost state in Γ_J

Table 2: Single-beam (S-**Beam**) decoding algorithm (related to (Lowerre and Reddy, 1980)).

input: source sentence with words $f_1, \cdots, f_J$

$\Gamma := \{\sigma_0\}$
for each $c = 0, 1, \cdots, J$ **do**
  $\Gamma' := \{\emptyset\}$
  for each state σ in Γ **do**
  if CLOSED?(σ) **then**
  matcher: for each $\sigma' : \sigma \rightarrow_M \sigma'$
  else
  scanner: for single $\sigma' : \sigma \rightarrow_S \sigma'$
  update σ' **for** Γ'
  end
  Prune state set Γ'
  Swap Γ, Γ'
  end
end

output: translation from lowest cost state in Γ

can be carried out efficiently. It keeps track of the already processed input sentence positions. d is the cost of the shortest path (distance) from some initial state σ_0 to the current state σ. The baseline decoder maintains $J + 1$ state lists with entries of the above type, where J is the number of input words. The states are stored in lists or stacks that support lookup operations to check whether a given state tuple is already present in a list and what its score d is.

The use of a coverage vector $\mathcal{C}$ is related to a DP-based solution for the traveling salesman problem as illustrated in Fig. 1. The algorithm keeps track of sets of visited cities along with the identity of the last visited city. Cities correspond to source sentence positions j. The vertexes in this graph correspond to set of already visited cities. Since the traveling salesman problem (and also the translation model) uses only local costs, the order in which the source positions have been processed can be ignored. Conceptually, the re-ordering problem is **linearized** by searching a path through the set inclusion graph in Fig. 1. Phrase-based decoding is handle by an almost identical algorithm: the last visited position j is replaced by an interval $[j', j]$.

The states are stored in lists or stacks that support lookup operations to check whether a given state tuple is already present in a list and what its score d is. Extending the partial block translation that is represented by a state σ with a single block b' generates a new state σ'. Here, $[k, k']$ is the source interval where block b' matches the input sentence. The state transition is defined as follows:

$$[\mathcal{C}\ ,\ h\ ;\ d] \quad \rightarrow_M \quad [\mathcal{C}'\ ,\ h'\ ;\ d'] \ . \tag{3}$$

The σ' state fields are updated on a component-by-component basis. $\mathcal{C}' = \mathcal{C} \cup [k, k']$ is the coverage vector obtained by adding all the positions from the interval $[k, k']$. The new state history is defined as $h' = ([k, k'], [u', v'])$ where u' and v' are the final two target words of the target phrase T' of b'. Some special cases, e.g. where T' has less than two target words, are taken into account. The path cost d' is computed as $d' = d + d(\sigma, \sigma')$, where the transition cost $d(\sigma, \sigma') := c(b, b')$ is computed from the history h and the matching block b' as defined in Section 5.

The decoder in Table 1 fills $J + 1$ state sets $\Gamma_k : \ k = \{0, \cdots, J\}$. All the coverage vectors $\mathcal{C}$ for states in the set Γ_k cover the same number of source positions k. When a state set Γ_k is processed, the decoder has finished processing all states in the sets Γ_l where $l < k$. Before expanding a state set, the decoder prunes a state set based on its coverage vector and the path costs only: 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 cardinality $c(\mathcal{C})$ of covered positions: all states in the beam are compared with each other. Since the states are kept in $J + 1$ separate lists, which are pruned independently of each others, this decoder version is called **multi-beam** decoder. The decoder uses a **matcher** function when expanding a state: for a state σ it looks for uncovered source positions to find source phrase matches for blocks. **Updating** a state in Table 1 includes adding the state if it is not yet present or updating its shortest path cost d: if the

state is already in Γ_c only the state with the lower path cost d is kept. This inserting/updating operation is also called **recombination** or **relaxation** in the context of a dag search algorithm (cf. Section 3). The **update** procedure also stores for each state σ' its predecessor state in a so-called back-pointer array (Ney et al., 1992). The final block alignment and target translation can be recovered from this back-pointer array once the final state set Γ_J has been computed. $l(\sigma')$ is the source phrase length of the matching block b' when going from σ to σ'. This algorithm is similar to the beam-search algorithm presented in (Koehn, 2004): it allows states to be added to a stack that is not the stack for the successor cardinality. σ_0 is the initial decoder state, where no source position is covered: $\mathcal{C} = \emptyset$. For the final states in Γ_J all source positions are covered.

2.2 Single-Beam Implementation

The second implementation uses two lists to keep a single beam of active states. This corresponds to a beam-search decoder in speech recognition, where path hypotheses corresponding to word sequences are processed in a time-synchronous way and at a given time step only hypotheses within some percentage of the best hypothesis are kept (Lowerre and Reddy, 1980). The single-beam decoder processes hypotheses **cardinality-synchronously**, i.e. the states at stage k generate new states at position $k+1$. In order to make the use of a single beam possible, we slightly modify the state transitions in Eq. 3:

$$[\mathcal{C}\ ,\ l\ ,\ h\ ;\ d] \quad \rightarrow_S \quad [\mathcal{C}'\ ,\ l'\ ,\ h\ ;\ d']\,, \tag{4}$$

$$[\mathcal{C}\ ,\ \cdot\ , h\ ;\ d] \quad \rightarrow_M \quad [\mathcal{C}'\ ,\ l' = k\ ,\ h'\ ;\ d']\,. \tag{5}$$

Here, Eq. 5 corresponds to the matcher definition in Eq. 3. We add an additional field that is a pointer keeping track of how much of the recent source phrase match has been covered. In Eq. 5, when a block is matched to the input sentence, this pointer is set to position k where the most recent block match starts. We use a dot $\cdot$ to indicate that when a block is matched, the matching position of the predecessor state can be ignored. While the pointer l is not yet equal to the end position of the match k', it is increased $l' := l + 1$ as shown in Eq. 4. The path cost d is set: $d' = d + \Delta$, where Δ is the state transition cost $d(\sigma, \sigma')$ divided by the source phrase length of block b': we evenly spread the cost of generating b' over all source positions being matched. The new coverage vector $\mathcal{C}'$ is obtained from $\mathcal{C}$ by adding the scanned position l': $\mathcal{C}' = \mathcal{C} \cup \{l'\}$. The algorithm that makes use of the above definitions is shown in Table 2. The states are stored in only two state sets 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 hypotheses in the current beam that cover $k+1$ source positions. The single-beam decoder in Table 2 uses two procedures: the **scanner** and the **matcher** correspond to the state transitions in Eq. 4 and Eq. 5. Here, the **matcher** simply matches a block to an uncovered portion of the input sentence. After the matcher has matched a block, that block is processed in a cardinality-synchronous way using the scanner procedure as described above. The predicate CLOSED?(σ) is used to switch between matching and scanning states. The predicate CLOSED?(σ) is true if the pointer l is equal to the match end position k' (this is stored in h'). At this point, the position-by-position match of the source phrase is completed and we can search for additional block matches.

3 DP Shortest Path Algorithm for dag

This section analyzes the relationship between the block decoding algorithms in this paper and a single-source shortest path finding algorithm for a directed acyclic graphs (dag). We closely follow the presentation in (Cormen et al., 2001) and only sketch the algorithm here: 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, the shortest path from a single source can be computed in $O(|V| + |E|)$ time, where $|V|$ is the number of vertexes and $|E|$ number of edges in the graph. The dag search algorithm runs over all vertexes σ in topological order. Assuming an **adjacency-list** representation of the dag, for each vertex σ, we loop over all successor vertexes σ', where each vertex σ with its adjacency-list is processed exactly once. During the search, we maintain for each vertex σ' an attribute $d[\sigma']$, which is an upper bound on the shortest path cost from the source vertex s to the vertex σ'. This shortest path estimate is updated or **relaxed** each time the vertex σ' occurs in some adjacency list. Ignoring the pruning, the M-Beam decoding algorithm in Table 1 and the dag search algorithm can be compared as follows: states correspond to dag vertexes and state transitions correspond to dag edges. Using two loops for the multi-beam decoder while generating states in stages is just a way of generating a topological sort of the search states on the fly: a linear order of search states is generated by appending the search states in the state lists Γ_0, Γ_1, etc. .

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 adjacency-list traversal in the dag search algorithm. These involve costly lookup operations, e.g. language, distortion and translation model probability lookup. Typically the computation time for update operations on lists Γ is negligible compared to these probability lookups. So, the search algorithm complexity is simply computed as the number of edges in the search graph: $O(|V| + |E|) \approx O(|E|)$ (this analysis is implicit in (Tillmann,

2001)). Without proof, for the search algorithm in Section 2.1 we observe that the number of states is finite and that all the states are actually reachable from the start state σ_0. This way for the single-word based search in (Tillmann and Ney, 2003), a complexity of $O(|V_T|^3 \cdot J^2 \cdot 2^J)$ is shown, where $|V_T|$ is the size of the target vocabulary and J is the length of the input sentence. The complexity is dominated by the exponential number of coverage vectors $\mathcal{C}$ that occur in the search, and the complexity of phrase-based decoding is higher yet since its hypotheses store a source interval $[j', j]$ rather than a single source position j. In the general case, no efficient search algorithm exists to search all word or phrase reorderings (Knight, 1999). Efficient search algorithms can be derived by the restricting the allowable coverage vectors (Tillmann, 2001) to local word re-ordering only. An efficient phrase alignment method that does not make use of re-ordering restriction is demonstrated in the following section.

4 Efficient Block Alignment Algorithm

A common approach to phrase-based SMT is to learn phrasal translation pairs from word-aligned training data (Och and Ney, 2004). Here, a word alignment $\mathcal{A}$ is a subset of the Cartesian product of source and target positions:

$$\mathcal{A} \subseteq \{1, \cdots, I\} \times \{1, \cdots, J\}.$$

Here, I is the target sentence length and J is the source sentence length. The phrase learning approach in (Och and Ney, 2004) takes two alignments: a source-to-target alignment $\mathcal{A}_1$ and a target-to-source alignment $\mathcal{A}_2$. The intersection of these two alignments is computed to obtain a high-precision word alignment. Here, we note that if the intersection covers all source and target positions (as shown in Fig. 4), it constitutes a bijection between source and target sentence positions, since the intersecting alignments are functions according to their definition in (Brown et al., 1993) [3]. In this paper, an algorithmic justification for restricting blocks based on word alignments is given. We assume that source and target sentence are given, and the task is to compute the lowest scoring block alignment. Such an algorithm might be important in some discriminative training procedure that relies on decoding the training data efficiently.

To restrict the block selection based on word aligned training data, interval projection functions are defined as follows [4]: S is a source interval and T is an target interval. $proj_T(S)$ is the set of target positions i such that the alignment point (i, j) occurs in the alignment set $\mathcal{A}$ and j is covered by the source interval S. $proj_S(T)$ is defined accordingly. Formally, the definitions look like this:

$$\begin{aligned} proj_T(S) &= \{\, i \mid (i,j) \in \mathcal{A} \text{ and } j \in S \,\} \\ proj_S(T) &= \{\, j \mid (i,j) \in \mathcal{A} \text{ and } i \in T \,\} \end{aligned}$$

In order to obtain a particularly simple block alignment algorithm, the allowed block links (S, T) are restricted by an ADMISSIBILITY restriction, which is defined as follows:

$$\begin{aligned} &(T, S) \text{ is admissible } \textbf{iff} \\ &\quad proj_S(T) \subseteq S \text{ and } proj_T(S) \subseteq T \end{aligned} \tag{6}$$

Admissibility is related to the word re-ordering problem: for the source positions in an interval S and for the target positions in an interval T, all word re-ordering involving these positions has to take place **within** the block defined by S and T. Without an underlying alignment $\mathcal{A}$ each pair of source and target intervals would define a possible block link: the admissibility reduces the number of block links drastically. Examples of admissible and non-admissible blocks are shown in Fig. 3.

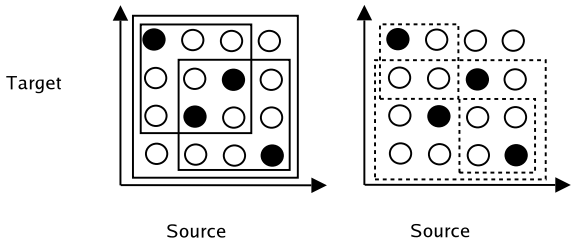

Figure 3: Following the definition in Eq. 6, the left picture shows three **admissible** block links while the right picture shows three **non-admissible** block links.

If 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. Because of the admissibility definition, a target interval clumping alone is sufficient to determine the source interval clumping **and** the clump alignment. In Fig. 4, a bijection word alignment for a sentence pair that consists of $J = 4$ source and $I = 4$ target words is shown, where the alignment links that yield a bijection are shown as solid dots. Four admissible block alignments are shown as well. An admissible block alignment is always guaranteed to exist: the block that covers all source and target position is admissible by definition. The underlying word alignment and the admissibility restriction play together to reduce the number of block alignments: out of all eight possible target clumpings, only

[3] (Tillmann, 2003) reports an intersection coverage of about 65 % for Arabic-English parallel data, and a coverage of 40 % for Chinese-English data. In the case of uncomplete coverage, the current algorithm can be extended as described in Section 4.1.

[4] (Och and Ney, 2004) defines the notion of consistency for the set of phrasal translations that are learned from word-aligned training data which is equivalent.

Table 3: Efficient DP-based block alignment algorithm using an underlying word alignment $\mathcal{A}$. For simplicity reasons, the block score $c(b')$ is computed based on the block identity b' only.

input: Parallel sentence pair and alignment $\mathcal{A}$.
initialization: $Q(0) = 0;\ Q(i) = \infty;\ \gamma(i, i') = \infty;$ for $i, i' = 1, \cdots, I$.
for each $i = 1, 2, \cdots, I$ do
$Q(i) = \min_{i'} \gamma(i, i') + Q(i')$, where
$\gamma(i, i') = c(b')$ if block b' results from admissible block link (T, S), where $T = [i' + 1, i]$
traceback:
- find best end hypothesis: $Q(I)$

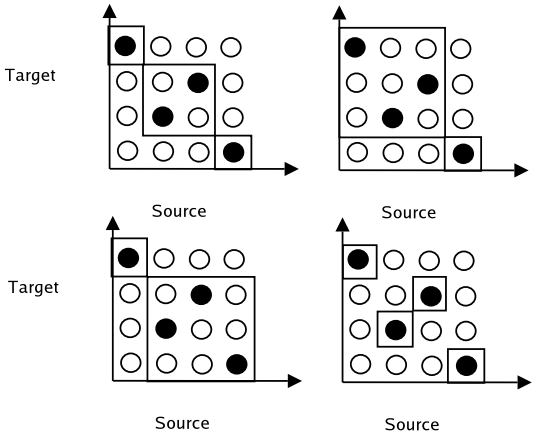

Figure 4: Four admissible block alignments in case the word alignment intersection is a bijection. The block alignment which covers the whole sentence pair with a single block is not shown.

five yield segmentations with admissible block links.
The DP-based algorithm to compute the block sequence with the highest score $Q(i)$ is shown in Table 3. Here, the following auxiliary quantity is used:

$Q(i)$:= score of the best partial segmentation that covers the target interval $[1, i]$.

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 an admissible block. For the initialization, we set $Q(i) = \infty$ and the final score is obtained as $Q_{Final} = Q(I)$. The complexity of the algorithm is $\mathcal{O}(I^2)$ where the time to compute the cost $c(b')$ and the time to compute the interval projections are ignored. Using the alignment links $\mathcal{A}$, the segmentation problem is essentially linearized: the

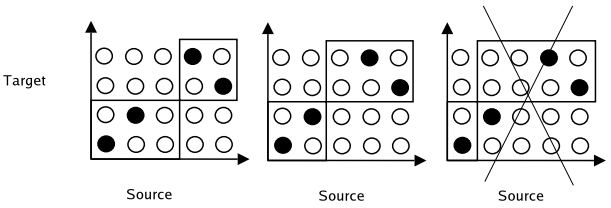

Figure 5: An example for a block alignment involving a non-aligned column. The right-most alignment is not allowed by the closure restriction.

target clumping is generated sequentially from bottom-to-top and it induces some source clumping in an order which is defined by the word alignment.

4.1 Incomplete Bijection Coverage

In this section, an algorithm is sketched that works if the intersection coverage is not complete. In this case, a given target interval may produce several admissible block links since it can be coupled with different source intervals to form admissible block links, e.g. in Fig. 5, the target interval $[0, 1]$ is linked to two source intervals and both resulting block links do not violate the admissibility restriction. The minimum score block translation can be computed using either the one-beam or the multi-beam algorithm presented earlier. The search state definition in Eq. 2 is modified to keep track of the current target position i the same way as the recursive quantity $Q(i)$ does this in the algorithm in Table 3:

$$[\,\mathcal{C}\ ,\ h\ ,\ i\ ;\ d\,]\,. \tag{7}$$

Additionally, a complex block history h as defined in Section 2 can be used. Before the search is carried out, the set of admissible block links for each target interval is precomputed and stored in a table where a simple look-up for each target interval $[i', i]$ is carried out during alignment. The efficiency of the block alignment algorithm depends on the alignment intersection coverage.

5 Beam-Search Results

In this section, we present results for the beam-search algorithms introduced in Section 2. The MT03 Arabic-English NIST evaluation test set consisting of 663 sentences with $16\,278$ Arabic words is used for the experiments. Translation results in terms of uncased BLEU using 4 reference translations are reported in Table 4 and Table 5 for the single-beam (S-**Beam**) and the multi-beam (M-**Beam**) search algorithm. For all re-ordering experiments, the notion of skips is used (Tillmann and Ney, 2003) to restrict the phrase re-ordering: the number of skips restricts the number of holes in the coverage vector for a left-to-right traversal of the input sentence. All

Table 4: Effect of the skip parameter for the two search strategies. $t_c = 2.5, t_{\mathcal{C}} = 1.0$ and window width $w = 6$.

Skip	BLEU	CPU	BLEU	CPU
	S-**Beam**	[secs]	M-**Beam**	[secs]
0	40.7 ± 1.4	108	40.9 ± 1.5	116
1	44.1 ± 1.5	729	44.1 ± 1.6	2459
2	44.3 ± 1.6	4408	44.4 ± 1.6	8437
3	44.3 ± 1.6	7467	44.5 ± 1.6	10048

re-ordering takes place in a window of size $w = 6$, such that only local block re-ordering is handled.

The following block bigram scoring is used: a block pair $(b; b')$ with corresponding source phrase matches $([j, j'], [k, k'])$ is represented as a feature-vector $f(b; b') \in \mathbb{R}^7$. The feature-vector components are the negative logarithm of some probabilities as well as a word-penalty feature. The real-valued features include the following: a block translation score derived from phrase occurrence statistics (1), a trigram language model to predict target words $(2-3)$, a lexical weighting score for the block internal words (4), a distortion model $(5-6)$ as well as the negative target phrase length (7). The transition cost is computed as $c(b\ ,\ b') = w^T \cdot f(b; b')$, where $w \in \mathbb{R}^7$ is a weight vector that sums up to 1.0: $\sum_{i=1}^{7} w_i = 1.0$. The weights are trained using a procedure similar to (Och, 2003) on held-out test data. A block set of 9.5 million blocks, which are not filtered according to any particular test set is used, which has been generated by a phrase-pair selection algorithm similar to (Al-Onaizan et al., 2004). The training data is sentence-aligned consisting of 3.3 million training sentence pairs.

Beam-search results are presented in terms of two pruning thresholds: the coverage pruning threshold $t_{\mathcal{C}}$ and the cardinality pruning threshold t_c (Tillmann and Ney, 2003). To carry out the pruning, the minimum cost with respect to each coverage set $\mathcal{C}$ and cardinality c are computed for a state set Γ. For the coverage pruning, states are distinguished according to the subset of covered positions $\mathcal{C}$. The minimum cost $\hat{Q}_\Gamma(\mathcal{C})$ is defined as:$\hat{Q}_\Gamma(\mathcal{C}) = \min_d\{d \mid [\mathcal{C}, h; d] \in \Gamma\}$. For the cardinality pruning, states are distinguished according to the cardinality $c(\mathcal{C})$ of subsets $\mathcal{C}$ of covered positions. The minimum cost $\hat{Q}_\Gamma(c)$ is defined for all hypotheses with the same cardinality $c(\mathcal{C}) = c$: $\hat{Q}_\Gamma(c) = \min_{\substack{\mathcal{C} \\ c(\mathcal{C})=c}} \hat{Q}_\Gamma(\mathcal{C})$.

States σ in Γ are pruned if the shortest path cost $d(\sigma)$ is greater than the minimum cost plus the pruning threshold:

$$\begin{aligned} d(\sigma) &> t_{\mathcal{C}} + \hat{Q}_\Gamma(\mathcal{C}) \\ d(\sigma) &> t_c + \hat{Q}_\Gamma(c) \end{aligned}$$

The same state set pruning is used for the S-Beam and the M-Beam search algorithms. Table 4 shows the effect of the skip size on the translation performance. The pruning thresholds are set to conservatively large values: $t_c = 2.5$ and $t_{\mathcal{C}} = 1.0$. Only if no block re-ordering is allowed ($skip = 0$), performance drops significantly. The S-Beam search is consistently faster than M-Beam search algorithm. Table 5 demonstrates the effect of the coverage pruning threshold. Here, a conservatively large cardinality pruning threshold of $t_c = 10.0$ and the so-called **histogram** pruning to restrict the overall number of states in the beam to a maximum number of 2500 are used to restrict the overall search space. The S-Beam search algorithm is consistently faster than the M-Beam search algorithm for the same pruning threshold, but performance in terms of BLEU score drops significantly for lower coverage pruning thresholds $t_{\mathcal{C}} < 0.5$ as a smaller portion of the overall search space is searched which leads to search errors. For larger pruning thresholds $t_{\mathcal{C}} \geq 0.5$, where the performance of the two algorithms in terms of BLEU score is nearly identical, the S-Beam algorithm runs significantly faster. For a coverage threshold of $t_{\mathcal{C}} = 0.1$, the S-Beam algorithm is as fast as the M-Beam algorithm at $t_{\mathcal{C}} = 0.01$, but obtains a significantly higher BLEU score of 42.6 versus 41.0 for the M-Beam algorithm. The results in this section show that the S-Beam algorithm generally runs faster since the beam search pruning is applied to all states simultaneously making more efficient use of the beam search concept.

Table 5: Effect of the coverage pruning threshold $t_{\mathcal{C}}$ on BLEU and the overall CPU time [secs]. To restrict the overall search space the cardinality pruning is set to $t_c = 10.0$ and the cardinality histogram pruning is set to 2500.

$t_{\mathcal{C}}$	BLEU	CPU	BLEU	CPU
	S-**Beam**	[secs]	M-**Beam**	[secs]
0.001	37.5 ± 1.4	106	40.5 ± 1.5	198
0.01	38.3 ± 1.4	109	41.0 ± 1.5	213
0.05	40.7 ± 1.5	139	43.2 ± 1.6	301
0.1	42.6 ± 1.5	215	44.2 ± 1.6	508
0.25	44.1 ± 1.6	1018	44.4 ± 1.6	1977
0.5	44.3 ± 1.6	4527	44.4 ± 1.6	6289
1.0	44.3 ± 1.6	6623	44.5 ± 1.6	8092
2.5	44.3 ± 1.6	6797	44.5 ± 1.6	8187
5.0	44.3 ± 1.6	6810	44.5 ± 1.6	8191

6 Discussion

The decoding algorithm shown here is most similar to the decoding algorithms presented in (Koehn, 2004) and (Och and Ney, 2004), the later being used for the Alignment Template Model for SMT. These algorithms also

include an estimate of the path completion cost which can easily be included into this work as well ((Tillmann, 2001)). (Knight, 1999) shows that the decoding problem for SMT as well as some bilingual tiling problems are NP-complete, so no efficient algorithm exists in the general case. But using DP-based optimization techniques and appropriate restrictions leads to efficient DP-based decoding algorithms as shown in this paper.

The efficient block alignment algorithm in Section 4 is related to the inversion transduction grammar approach to bilingual parsing described in (Wu, 1997): in both cases the number of alignments is drastically reduced by introducing appropriate re-ordering restrictions. The list-based decoding algorithms can also be compared to an Earley-style parsing algorithm that processes list of parse states in a single left-to-right run over the input sentence. For this algorithm, the comparison in terms of a shortest-path algorithm is less obvious: in the so-called completion step the parser re-visits states in previous stacks. But it is interesting to note that there is no multiple lists variant of that parser. In phrase-based decoding, a multiple list decoder is feasible only because exact phrase matches occur. A block decoding algorithm that would allow for a 'fuzzy' match of source phrases, e.g. insertions or deletions of some source phrase words are allowed, would need to carry out its computations using two stacks since the match end of a block is unknown.

7 Acknowledgment

This work was partially supported by DARPA and monitored by SPAWAR under contract No. N66001-99-2-8916. The author would like to thank the anonymous reviewers for their detailed criticism on this paper.

References

Yaser Al-Onaizan, Niyu Ge, Young-Suk Lee, Kishore Papineni, Fei Xia, and Christoph Tillmann. 2004. IBM Site Report. In *NIST 2004 MT Workshop*, Alexandria, VA, June. IBM.

Adam L. Berger, Peter F. Brown, Stephen A. Della Pietra, Vincent J. Della Pietra, Andrew S. Kehler, and Robert L. Mercer. 1996. Language Translation Apparatus and Method of Using Context-Based Translation Models. *United States Patent, Patent Number 5510981*, April.

Peter F. Brown, Vincent J. Della Pietra, Stephen A. Della Pietra, and Robert L. Mercer. 1993. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19(2):263–311.

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2001. *Introduction to Algorithms*. MIT Press, Cambridge Massachusetts.

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

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

Fred Jelinek. 1998. *Statistical Methods for Speech Recognition*. The MIT Press, Cambridge, MA.

Kevin Knight. 1999. Decoding Complexity in Word-Replacement Translation Models. *CL*, 25(4):607–615.

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

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

S. Niessen, S. Vogel, H. Ney, and C. Tillmann. 1998. A DP-Based Search Algorithm for Statistical Machine Translation. In *Proc. of ACL/COLING 98*, pages 960–967, Montreal, Canada, August.

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

Franz Josef Och. 2003. Minimum error rate training in statistical machine translation. In *Proceedings of ACL'03*, pages 160–167, Sapporo, Japan.

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, Stefan Vogel, Hermann Ney, and Alex Zubiaga. 1997. A DP-based Search Using Monotone Alignments in Statistical Translation. In *Proc. of ACL 97*, pages 289–296, Madrid,Spain, July.

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 *Proc. of EMNLP 03*, pages 1–8, Sapporo, Japan, July.

Dekai Wu. 1997. Stochastic Inversion Transduction Grammars and Bilingual Parsing of Parallel Corpora. *Computational Linguistics*, 23(3):377–403.