MACHINE TRANSLATION USING SCARCE BILINGUAL CORPORA

KRZYSZTOF JASSEM AND TOMASZ KOWALSKI

Department of Computer Science, Adam Mickiewicz University,
Umultowska 87, 61-614 Poznan, Poland
{jassem, kowalski}@amu.edu.pl

(Received 29 December 2006; revised manuscript received 30 January 2007)

Abstract: We propose a method for automatic extraction of translation rules suitable for a rule-based machine translation system by using a target language syntactic parser and scarce bilingual resources as linguistic knowledge sources. We propose an algorithm that assembles translation rules in order to translate an input sentence.

Keywords: machine translation, parellel corpora analysis, rule extracion

1. Introduction

We have observed growing interest in Statistical Machine Translation (SMT). To a large extent, this is due to the increasing volume of parallel bilingual data for major languages. Moreover, the statistical approach involves a lot of computation time to train the model but almost no manpower.

At the same time, linguistic knowledge acquired in the training phase of a statistical model does not allow for any human tuning, which makes it difficult to correct potential errors.

In Rule-based Machine Translation (RMT), linguistic knowledge is hand-coded into the system. The translation process may be tuned according to the observed language-specific phenomena. RMT systems deliver translations of better quality than SMT systems, but are much more expensive as they require manpower with both linguistic and computer programming skills in order to encode linguistic knowledge into a machine-readable form.

It would appear desirable to combine the advantages of both approaches. One way to do so is to build an RMT system with rules being achieved automatically – by means of statistical calculations.

The idea to acquire transfer rules automatically from a word-aligned corpus was presented in [1–4], where transfer units were based on subtrees in the source language parse tree. This schema should work best for pairs of languages, one of which is well known for the system's developers whereas the other is not.

In this paper, we would like to focus on the possibility of obtaining translation rules without using a source language parser. We assume the availability of a parser for a target language and scarce bilingual resources in the form of word-aligned sentences.

Galley *et al.* (see [5]) have proposed a theory offering formal semantics to word-level alignments defined over parallel corpora. They have introduced a linear algorithm that can be used to derive the minimal set of syntactically motivated transformation rules from word-aligned parallel corpora. The transformation rules are extracted from the parse tree of the target sentence and the pre-determined equivalents of its nodes in the source sentence.

Here, we modify this algorithm by using a different annotation schema for analysed graphs and by using a binary vector algebra instead of set operations. The modified algorithm is used to extract translation rules for an RMT German-Polish system. We perform experiments with small German-Polish corpora and parse target sentences with the state-of-the-art non-statistical parser for Polish (see [6]) used in the TRANSLATICA system[1].

Moreover, we propose an algorithm assembling the acquired rules in the process of translating German sentences into Polish.

Finally, we suggest further work towards enhanced translation quality of an MT system based on this method.

The development of an RMT system based on this idea requires some human knowledge required for tagging the equivalence between simple components (usually words) in parallel sentences (possibly by verifying suggestions of a computer program). Such tagging, however, does not require specialized linguistic skills of human operators.

2. Rule extraction

We extract rules further used in the translation process from a structure called an alignment graph [5].

We use standard notations form the graph theory: for a graph G, $V(G)$ denotes the set of nodes, $E(G)$ denotes the set of edges, (n_1, n_2) denotes the edge from node n_1 to node n_2 in graph G.

An alignment graph AG is a rooted, directed (direction in diagrams is usually presented from top down), connected, acyclic graph spanned over the tokens of the source sentence and the parse tree of the target sentence. It consists of the set of source sentence nodes, S, *i.e.* nodes that represent words or, more precisely, tokens of the source sentence, a parse tree, P, of the target sentence, where $T \subset V(P)$ denotes the set of leaves of the parse tree, or the target sentence nodes, and an alignment A, *i.e.* edges between T and S: $A = \{(t, s) \in E(AG) : t \in T, s \in S\}$. We also require that $\forall_{s \in S} \exists_{t \in T} (t, s) \in A$. An exemplary alignment graph is shown in Figure 1.

The idea is to automatically extract subgraphs of the alignment graph that would generate translation rules.

DEFINITION 1. Let n be a node of the parse tree $(n \in V(P))$, such that removing the edge between node n and its parent dissects the alignment graph into two disjoint

1. TRANSLATICA (http://www.translatica.pl) is a commercial name of a system formerly developed under the name POLENG

graphs. We call the graph rooted in node n *rule-inducing* if it covers a contiguous part of the source sentence, *i.e.* source sentence nodes of that graph form a substring of the source sentence.

The subgraph of the graph shown in Figure 1 rooted in node NP is rule-inducing. The subgraph is shown in Figure 2.

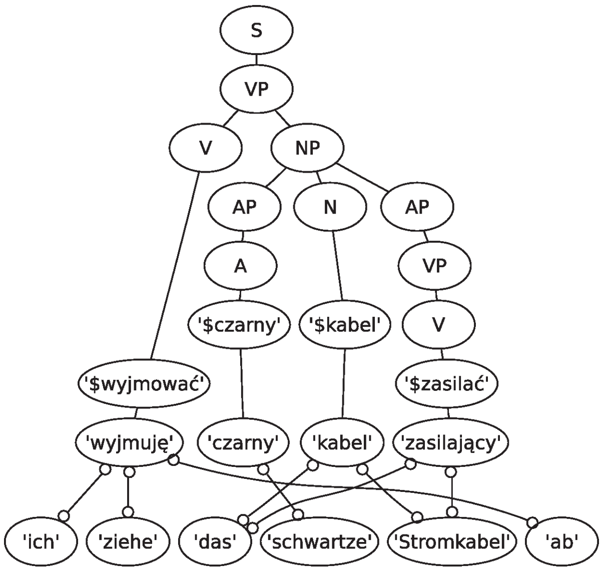

Figure 1. An example of an alignment graph

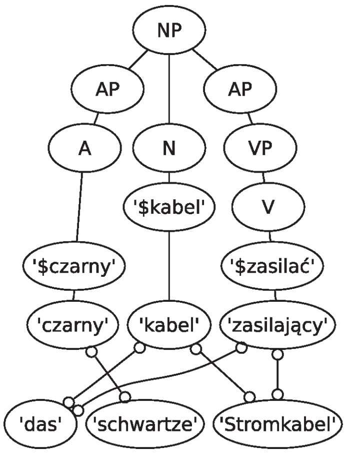

Figure 2. An example of a rule-inducing graph

In Figure 3a a subgraph is shown that does not induce a translation rule because removing the edge between its root and the root's parent does not dissect the alignment graph into two disjoint graphs. The subgraph shown in Figure 3b does not induce a translation rule because it covers a non-contiguous part of the source sentence. The algorithm we describe below analyses the alignment graph in a single traversal in order to find all rule-inducing subgraphs.

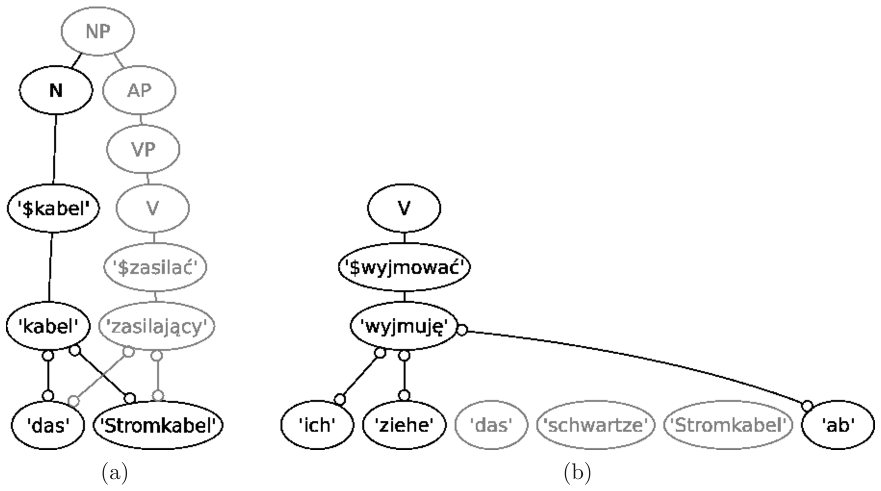

Figure 3. Examples of non rule-inducing graphs:
(a) removing the edge between N and NP does not dissect the graph;
(b) the V-rooted graph does not cover a contiguous part of the source sentence

First, the algorithm defines the order of the alignment edges "from left to right": number 1 is assigned to the leftmost edge of the alignment, *i.e.* the edge that links the first token of the source sentence to the leftmost token of the corresponding target sentence tokens; $v = card(A)$ is assigned to the rightmost edge of the assignment. Or, more precisely: Let $(t_1, s_1), (t_2, s_2) \in A$, and $s_1 \prec_S s_2$ if and only if node s_1 precedes s_2 in sentence S. Each alignment edge is assigned a number, $n((t,s) \in A) \in [1, v]$, so that the following should hold:

$$n(t_1,s_1) < n(t_2,s_2) \;\Leftrightarrow\; s_1 \underset{S}{\prec} s_2 \;\vee\; s_1 = s_2 \wedge t_1 \underset{T}{\prec} t_2 \tag{1}$$

For each node m in the alignment graph, the algorithm calculates two binary vectors, $span(m)$ and $mask(m)$, of length v equal to the number of edges in the alignment ($|span(m)| = |mask(m)| = card(A)$). For each $s \in S$, the vectors are as follows:

$$span(s \in S) = 0 \tag{2}$$

$$mask(s \in S) = [m_v, \ldots, m_1] : m_i = \begin{cases} 1, & \exists t \in T : i = n((t,\, s) \in A) \\ 0, & \text{otherwise} \end{cases} \tag{3}$$

For each $t \in T$, they are:

$$span(t \in T) = [m_v, \ldots, m_1] : m_i = \begin{cases} 1, & \exists s \in S : i = n((t,\, s) \in A) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

$$mask(t \in T) = \bigvee_{(t,s) \in A} mask(s) \tag{5}$$

The vectors for the remaining nodes are calculated by traversing the alignment graph bottom-up:

$$span(n \in V(P) \setminus T) = \bigvee_{(n,\, m) \in E(P)} span(m) \tag{6}$$

$$mask(n \in V(P) \setminus T) = \bigvee_{(n,\, m) \in E(P)} mask(m) \tag{7}$$

Listing 1. The rule extraction algorithm

```
#initializing source sentence nodes
$edge_number = 1;
foreach $s (@S) {
  foreach $t ($s -> {aligned_to}) {
    $t -> {span} |= $edge_number;
    $s -> {mask} |= $edge_number;
    $edge_number <$< 1;
  }
  create_a_discard_rule ($s) if (0 == $s -> {mask});
}
#initializing target sentence nodes
foreach $t (@T) {
  $t -> {mask} |= $s -> {mask} foreach $s ($t -> {aligned_to});
  $seen {$n} = 1;
}
#traversing the rest of the alignment graph bottom-top
@stack = ($r); #root of the alignment graph
while ($n = pop @stack) {
  unless ($seen {$n}) {
    $seen {$n} = 1;
    push @stack, $n;
    push @stack, $n -> {siblings};
  } else {
    foreach $c in ($n -> {siblings}) {
     $n -> {span} |= $c -> {span};
     $n -> {mask} |= $c -> {mask};
    }
    if ($n -> {span} == $n -> {mask}
     && binary_to_string ($n -> {span}) =~ //^0*1+0*$//
     ) {
     convert_to_rule ($n);
     @n -> {siblings} = (); #detaching all siblings
    }
  }
}
```

It is claimed here that:

Lemma 1. For any node $p \in P$, $mask(p) = span(p)$ iff removing the edge e' between the node p and its parent dissects the alignment graph into two disjoint graphs.

Proof. Let R be a p-rooted graph.

Let us suppose that $mask(p) = span(p)$ and that removing edge e' does not result in partitioning the alignment graph into two disjoint graphs.

Since P is a tree and thus removing any of its edges results in its partition into two disjoint trees, there has to be an edge $e = (t, s)$ in the alignment such that either $t \notin V(R) \wedge s \in V(R)$ or $t \in V(R) \wedge s \notin V(R)$.

Let us suppose that $t \in V(R) \wedge s \notin V(R)$. According to the Equation (4): $span(t)_{n(e)} = 1$. Since $span(p)$, calculated according to Equation (6), it is effectively:

$$span(p) = \bigvee_{x \in V(R) \cap T} span(x) \tag{8}$$

and when $span(t)_{n(e)} = 1$ then $span(p)_{n(e)} = 1$.

As we have assumed that $mask(p) = span(p)$, we obtain $mask(p)_{n(e)} = 1$. By unwinding the recursive Equation (7) we obtain:

$$mask(p) = \bigvee_{x \in V(R) \cap S} mask(x) \tag{9}$$

Therefore $\exists_{s_1 \in V(R) \cap S}\, mask(s_1)_{n(e)} = 1$. Since edges are numbered uniquely (relation (1)) and $mask(s_1)_{n(e)} = 1 = mask(s)_{n(e)}$, then $s_1 = s$, which results in contradiction as we have assumed that $s \notin V(R)$ but $s_1 \in V(R)$.

Similar reasoning for the other case $(t \notin V(R) \wedge s \in V(R))$ also leads to contradiction.

Let us now suppose that removing edge e' results in a partition of the alignment graph into two disjoint graphs and $mask(p) \neq span(p)$. If $mask(p) \neq span(p)$ then $\exists i = n(e) = n((t, s)) :\ mask(p)_i \neq span(p)_i$. Since $\{e'\}$ is the edge cut off the alignment graph then $t, s \in V(R)$ or $t, s \notin V(R)$.

Let us assume that $span(p)_i = 0$ and $mask(p)_i = 1$. Given Equation (8), we obtain $1 = mask(p)_{i=n(e)=n(t,s)} = mask(s)_{n(t,s)}$. By definition of the *span* and *mask* vectors, $mask(s)_{n(t,s)} = span(t)_{n(t,s)}$. Since $s \in V(R)$ then $t \in V(R)$ and the following holds: $1 = span(t)_{n(t,s)} = span(p)_{n(t,s)} = span(p)_i$.

A similar reasoning for the case of $span(p)_i = 1$ and $mask(p)_i = 0$ also leads to contradiction. ■

Lemma 2. For any node $p \in P$, the p-rooted graph is rule-inducing iff

$$span(p) = mask(p) = [\underbrace{0 \ldots 0}_{a-1}, \underbrace{1 \ldots 1}_{b-a+1}, \underbrace{0 \ldots 0}_{v-b}], \text{ where } a \in [1 \ldots v],\ b \in [a \ldots v]$$

Proof. According to Definition 1, a p-rooted subgraph of the alignment graph is rule-inducing iff (a) the alignment graph may be dissected into two disjoint graphs and (b) the leaves of the p-rooted graph form a contiguous part of the source string.

According to Lemma 1, the first condition is met when $span(p) = mask(p)$. Thus, it suffices to prove that condition (b) is satisfied by a "$0*1+0*$" vector, *i.e.* a sequence of one or more zeros, followed by one or more ones, followed by one or more zeros.

($\Leftarrow$) Let us assume that $span(p) = mask(p)$ and $mask(p)$ has the form of "$0*1+0*$" and that the p-rooted subgraph (R) of the alignment graph does not cover a substring of the source sentence. This implies that there has to be a sequence of source nodes of length m such that the first node in sequence $s_1 \in S \cap V(R)$ is followed (in terms of the source sentence) by one or more nodes such that $\forall_{i \in (1,m)} s_i \in S \wedge s_i \notin V(R)$, followed by a node $s_m \in S \cap V(R)$. According to Equation (9):

$$\begin{aligned} \forall t \in T &: [mask(p)]_{n_1 = n(t, s_1)} = 1 \\ \forall i \in (1, m)\ \forall t \in T &: [mask(p)]_{n_i = n(t, s_m)} = 0 \\ \forall t \in T &: [mask(p)]_{n_m = n(t, s_m)} = 1 \end{aligned}$$

But since $\forall_{i \in (1,m)} n_1 < n_i < n_m$, we observe a sequence of the form "$10+1$" (a single one, followed by one or more zeros, followed by a single one), which contradicts the assumption that the *mask* vector should have a "$0*1+0*$" form.

($\Rightarrow$) Let L be the set of leaves of the p-rooted rule-inducing graph. Since the leaves of a rule-inducing graph constitute a substring of the source sentence and the order of numbering is "left-to-right" (relation (1)), the alignment edges such that $\forall_{s \in L} \exists_{t \in T} (t, s) \in A$ are labelled with subsequent numbers. Moreover, let $a = \min\{n((t, s) \in A)\}$ and $b = \max\{n((t, s) \in A)\}$ for any $t \in T$ and $s \in L$. According to Equation (9), we obtain $\forall_{i \in [a, b]} [mask(p)]_i = 1$. For any source node s_i that precedes (in terms of the source sentence) any of the nodes in L, we have $\forall_{t \in T} n(t, s_i) \in A) < a$. Moreover, for any node $s_i \notin V(R)$ we have $\forall_{i \in [0 \dots a)} [mask(p)]_i = 0$. Similarly, for any source node s_j that follows (in terms of the source sentence) any of the nodes in L, we have $\forall_{t \in T} b < n(t, s_i) \in A)$. Since any node $s_j \notin V(R)$, we have $\forall_{i \in (b \dots v]} [mask(p)]_i = 0$. ■

Listing 1 shows the complete algorithm in the form of a Perl-like code. In order to simplify the exposition, we have converted the *span* vector into a string and used a regular expression to check whether it has the required form.

Once the root of a rule-inducing graph has been found, we convert it into a rule using the `convert_to_rule procedure`. The procedure forms the input of a rule from leaves of the given graph (see [5]). We order the leaves according to the *mask* vector. Please note that the sorted list of leaves often differs from the list obtained by in-depth traversal of the rule-inducing graph. Therefore, we replace the leaves of the rule-inducing graph (now the rule body) by references to the sorted list (rule input).

Examples of translation rules formed by the `convert_to_rule` procedure are shown in Figure 4. The input of a rule is shown at the bottom of every diagram with the rule's body above it. Empty circles represent placeholders for nodes that match the rule's input.

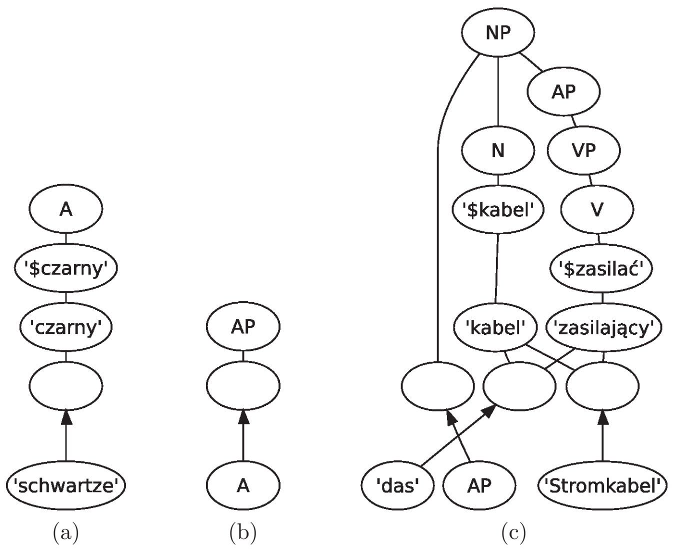

Figure 4. Examples of translation rules: (a) a dictionary-like rule, (b) a parser rule, (c) a translation rule with source tokens at the input

The `form_a_discard_rule` procedure creates a special kind of translation rules, having empty bodies. They store the information that a particular source sentence token is irrelevant to the translation process, since it has not been aligned to any target token.

A rule-inducing graph may contain smaller rule-inducing graphs. If all rule-inducing graphs were converted into rules, a lot of the information contained in the rules would be stored multiple times. In order to avoid such overload, each time a rule-inducing graph is found and converted into a rule we reduce the alignment graph by removing all nodes of the rule-inducing graph except for its root.

Such a rule is shown in Figure 4b, created after the A-rooted rule-inducing subgraph of the AP-rooted rule-inducing subgraph had been reduced to a single node.

3. Translating via rule assembling

Given a set of rules acquired in the way described above, it is possible to generate translations of sentences by constructing an alignment graph over the source sentence from graphs acquired in the training process.

Let us suppose that the rule set is derived from the alignment graphs shown in Figures 1 and 5, and the sentence to be translated is "ich ziehe das grüne Stromkabel ab".

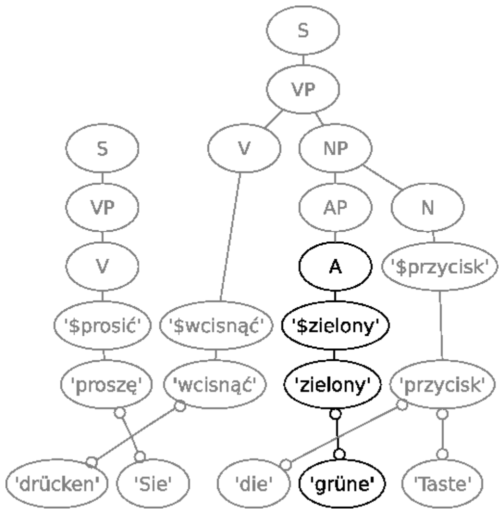

Figure 5. An example of an alignment graph with one of the rule-inducing graphs marked

Having processed the first alignment graph, we obtain a set of 5 rules (shown in Figures 4 and 6). An analysis of the second alignment graph results in 7 rules, but only one of these is relevant to this example. The rule-inducing graph of this rule has been marked in Figure 5.

We are able to translate the input sentence on the basis of our set of rules in 4 steps:

(a) first, the node representing the token 'grüne' of the input sentence is matched with the rule-inducing graph shown in Figure 5;
(b) next, we merge token A with the graph constituting the rule in Figure 4b;

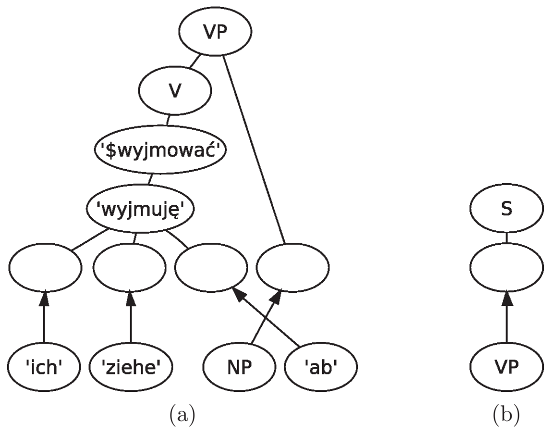

Figure 6. Translation rules acquired from the alignment graph of Figure 1

(c) the 'das', AP and 'Stromkabel' nodes correspond to leaf nodes of the graph shown in Figure 4c. They should be merged with respect to the reordering constraints of that rule;

(d) rule in Figure 6a is applied.

The process of subsequent application of rule-graphs is illustrated in Figure 7.

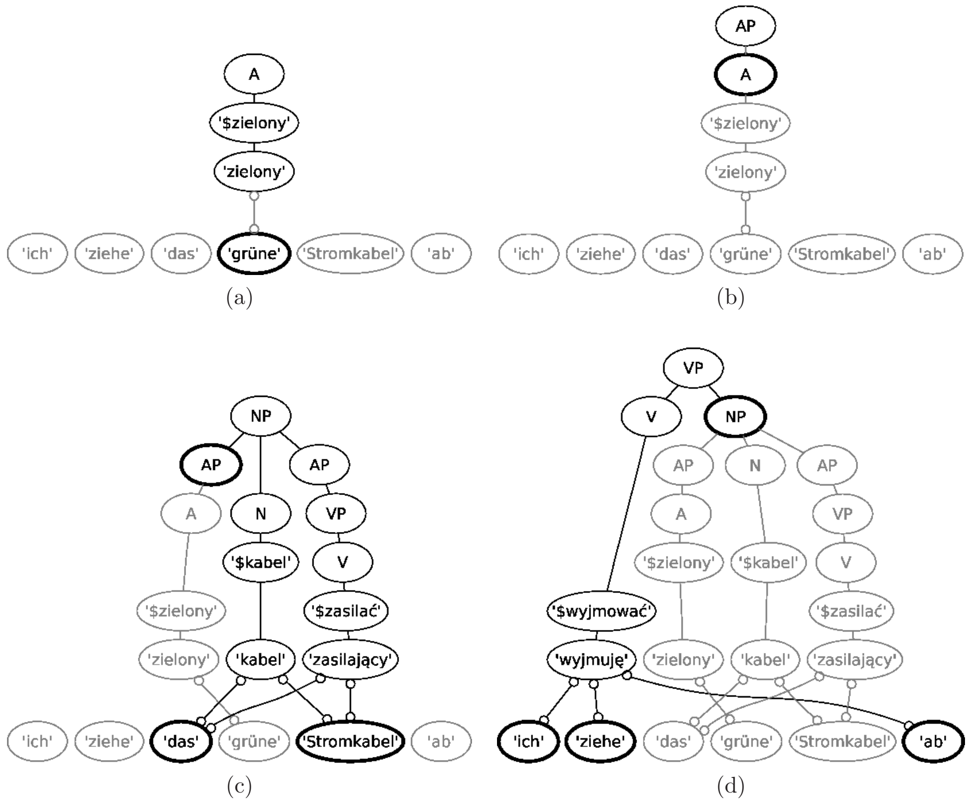

Figure 7. Subsequent steps of translation; nodes present before the application of a rule are grey, the merged nodes are marked by a thick outline

Since the constructed alignment graph rooted in VP already spans over the entire input sentence, there is no need to expand it any further, *e.g.* by applying the rule shown in Figure 6b. The translation is to be read from the target sentence nodes of the alignment graph.

The process shown in the above example may be implemented using the concept of dynamic programming. A pseudo-code of its algorithm is shown in Listing 2.

The algorithm operates on an $n \times n$ matrix, where n denotes the length of the input sentence. Every matrix element is a pair of lists, one of which contains references to the nodes of the alignment graph being constructed, and the other one – to virtual nodes called anchors (a concept explained below).

The matrix is initialized with the source sentence tokens forming the bottom level of the alignment graph (*i.e.* source sentence nodes). Then a database query is performed in order to find rule-graphs matching the source sentence nodes. If a match is found, the rule-graph is merged with its matching source sentence node and a reference to the root node of the rule-graph (now part of the alignment graph) is recorded.

After this initial step, the algorithm analyses longer segments of the input sentence, trying to construct alignment graph fragments spanning over more and more source sentence nodes. It stops after analysing a segment of length n, the length of the whole sentence.

While analysing a segment of length $m \in [2, n]$ starting from position $i \in [1, n-m+1]$, the algorithm performs a database query in order to find a rule-graph matching the nodes referenced as spanning over source nodes in positions from i to $i+j-1$ and from $i+j$ to $i+m-1$, where $j \in [1, m-1]$.

LISTING 2. The translation algorithm

```
#prerequisite $matrix is already initialized with source sentence tokens
sub compute_matrix ($matrix) {
  foreach $segment_start (0 .. $matrix -> {width}$-1) {
    expand_segment ($matrix -> [0] -> [$segment_start]);
  }
  foreach $segment_length (2 .. $matrix -> {height}) {
    foreach $segment_start (0 .. $matrix -> {width}$-$segment_length) {
      $current_cell = $matrix -> [$segment_length-1] -> [$segment_start];
      foreach $segment_partition (1 .. $segment_length-1) {
        push @$current_cell,
        join_segments (
          $matrix
             -> [$segment_partition-1]
             -> [$segment_start]
          , $matrix
             -> [$segment_length-$segment_partition-1]
             -> [$segment_start+$segment_partition]
        );
      }
      $current_cell = expand_segment ($current_cell);
    }
  }
}
```

The algorithm performs database look-ups only for rules that, if applied, merge two graph fragments into one. In order to allow the algorithm to use rules that merge more that two graph fragments without increasing its complexity, we have developed the concept of an anchor. An anchor is a virtual node linking two alignment graphs spanned over adjacent fragments. The algorithm treats anchors in the same way as real nodes of an alignment graph. The use of anchors is shown in Listing 3.

LISTING 3. Procedures used by the translation algorithm (Listing 2)

```
sub expand_segment (@list_of_nodes) {
  @to_process = @list_of_nodes;
  @processed = ();
  while ($current_node = pop @to_process) {
    push @processed, $current_node;
    unless ($current_node -> {is_anchor}) {
      @rules = run_db_query ($current_node);
      foreach $current_rule (@rules) {
        #apply rule
        merge_rule_graph_to_nodes ($current_rule, $current_node);
        push @to_process, $current_rule -> {root};
      }
    }
  }
  return @processed;
}

sub join_segments (@nodes_A, @nodes_B) {
  @roots = ();
  foreach $node_a (@nodes_A) {
    foreach $node_b (@nodes_B) {
      $query = [
        $node_a -> {is_anchor}? $node_a -> {query}: $node_a,
        $node_b -> {is_anchor}? $node_b -> {query}: $node_b
      ];
      @rules = run_db_query($query);
      foreach $current_rule (@rules) {
        $nodes = [
          $node_a -> {is_anchor}? $node_a -> {nodes}: $node_a,
          $node_b -> {is_anchor}? $node_b -> {nodes}: $node_b
        ];
        if ($current_rule -> {exact_match}) {
          #apply rule
          merge_rule_graph_to_nodes ($current_rule, $nodes);
          push @roots, $current_rule -> {root};
        } else {
          $anchor = new Anchor ($query);
          push @{$anchor -> {nodes}$}, @$nodes;
        }
      }
    }
  }
  return @roots;
}
```

We have designed the database query to look not only for rule-graphs matching the request exactly, but also for rules that match it partially. Every time an exact match is found, it is applied by the algorithm and real alignment graph nodes are created. The information about partial matches is recoded in the form of an anchor.

The idea of "anchoring" is analogous to that of normalization of a context-free grammar into the Chomsky form (usually done in order to use the CYK parser, see [7]). There, the rules with more than two right-hand symbols are replaced by rules with exactly two right-hand symbols by introducing virtual symbols – equivalents of anchors.

We have implemented the database query by using the standard B-tree index (see [8]) on rule-graphs' leafs. The properties of a B-tree index make it equally efficient in searching for prefixes as it is for exact matches. Moreover, the use of a B-tree index enables the use of a database engine as a store of rules.

4. Discussion and further work

Both algorithms were implemented in the Perl language. A corpora of 100 sentences from a user manual of a microelectronic device was used for testing. The source and target sentences were aligned manually.

We have noted the performance of the rule extraction algorithm to depend strongly on the quality of the alignment. During the initial run of the rule extraction algorithm, a half of the presented sentence pairs were stored entirely (a single rule covering the entire sentence was created). This was due to the relatively large density of the alignment: there were many source nodes aligned to more than one target node in those sentences. After removing some edges from the alignments, over 80% of the sentence pairs contributed to the rule set without being stored as a whole sentence. The remaining sentences tended to be long (over 20 tokens) and contributed to the rule set when presented as shorter fragments. Assuming the availability of only small corpora, different alignments of the same sentence pairs (*e.g.* prepared by different people) should be presented to the rule extraction algorithm for optimal performance.

We have observed three types of rules acquired by the above algorithm: (a) dictionary-like rules, (b) parsing rules and (c) mixed rules. Examples of rules of each type are shown in Figure 4.

A dictionary-like rule consists of four nodes: (i) a source sentence node – the source token, (ii) a target sentence node – its equivalent in the target language, (iii) a node that identifies the base form of the equivalent and (iv) a node containing grammatical information. The rule shown in Figure 4a was actually achieved in a post-processing step from three extracted rules, each of them consisting of only two connected nodes. It is worth noting that it would be certainly possible to acquire an exhaustive bilingual dictionary in this way, although this would require large corpora. Having only scarce bilingual resources, dictionary rules have to be supplied from external sources.

A parsing rule consists of P-nodes only. These rules will most likely form only a subset of the knowledge base for the parser of target sentences because the parser may be able to deal with cases not observed in the corpora. Thus, it would be better to transfer the parser's complete knowledge base into parsing rules by other means.

The third type of the acquired rules are mixed rules, *i.e.* rules consisting of source sentence nodes and parse tree nodes. We believe that rules of this type allow for capturing translation nuances that a human operator (*e.g.* an author of translation rules for a transfer-based MT system) may not be aware of.

Actually, the parser used here takes into account also semantic features of sentence components. This shows that acquired rules may have not only syntactical but also semantic motivation.

The ability of the assembling algorithm to produce translations is highly dependent on the graph merging procedure. The procedure used in the translation algorithm assembles subtrees of translation rules on the basis of agreement of basic syntactical attributes only, which makes the translation algorithm generate multiple hypotheses. It is thus necessary to introduce further constraints on the merging procedure. One of the possible solutions is to consider other syntactic and semantic attributes in assembling the translation rules. This would substantially limit the number of generated hypotheses. Another possibility is to choose hypotheses that have the highest probability with respect to the target language model [9].

Furthermore, the quality of translations obtained with this method could be improved by applying post-processing procedures similar to those used in transfer translation, *e.g.* morphological synthesis [10].

References

[1] Carbonell J, Probst K, Peterson E, Monson Ch, Lavie A, Brown R and Levin L 2002 *Proc. 5th Conf. Association for Machine Translation in the Americas (AMTA-02)*, London, UK, Springer-Verlag, pp. 1–10

[2] Dekang L 2004 *Proc. 20th Int. Conf. on Computational Linguistics: COLING-04*, Geneva, Switzerland, pp. 625

[3] Lavoie B, White M and Korelsky T 2002 *Proc. COLING-02 Workshop on Machine Translation*, Taipei, Taiwan, Asia, pp. 60–66

[4] Richardson S, Dolan W, Menezes A and Pinkham J 2001 *Proc. MT Summit VIII*, Santiago De Compostela, Spain, pp. 293–298

[5] Galley M, Hopkins M, Knight K and Marcu D 2004 *Proc. Human Language Technology and North American Association for Computational Linguistics Conference (HLT/NAACL-04)*, Boston, Massachusetts, USA, pp. 273–280

[6] Graliński F 2002 *Speech and Language Technology* **6** 263 (in Polish)

[7] Younger D H 1967 *Information and Control* **10** (2), pp. 189–208

[8] Bayer R and McCreight E M 2002 *Software Pioneers: Contributions to Software Engineering*, Springer-Verlag New York, Inc., pp. 245–262

[9] Song F and Croft W B 1999 *Proc. 8th Int. Conf. on Information and Knowledge Management (CIKM'99)*, Kansas City, Missouri, United States, ACM Press, pp. 316–321

[10] Jassem K 2002 *Speech and Language Technology* **6** 277 (in Polish)