Chapter 9

LEARNING TRANSLATION TEMPLATES FROM BILINGUAL TRANSLATION EXAMPLES

Ilyas Cicekli
ilyas@cs.bilkent.edu.tr

H. Altay Güvenir
guvenir@cs.bilkent.edu.tr

Abstract A mechanism for learning lexical correspondences between two languages from sets of translated sentence pairs is presented. These lexical level correspondences are learned using analogical reasoning between two translation examples. Given two translation examples, any similarities in the source language sentences must correspond to the similar parts of the target language sentences, while any differences in the source strings must correspond to the respective parts in the translated sentences. The correspondences between similarities and between differences are learned in the form of translation templates. A translation template is a generalized translation exemplar pair where some components are generalized by replacing them with variables in both sentences and establishing bindings between these variables. The learned translation templates are generalizations obtained by replacing differences or similarities by variables. This approach has been implemented and tested on a set of sample training datasets and produced promising results for further investigation.

Keywords: exemplar-based machine learning, difference translation templates, similarity translation templates

M. Carl and A. Way (eds.), Recent Advances in Example-Based Machine Translation, 255–286.

1. Introduction

Researchers in the Machine Learning community have widely used exemplar-based representations. The basic idea in exemplar-based learning is to use past experiences or cases to understand, plan, or learn from novel situations (Kolodner, 1988; Hammond, 1989; Ram, 1993). Collins and Somers (Chapter 4, this volume) view the EBMT approach as a special case of Case-Based Reasoning. Medin & Schaffer, 1978 were the first researchers who proposed exemplar-based learning as a model of human learning. In this chapter, we formulate the acquisition of translation rules as a machine learning problem.

Our first attempt in this direction was to construct parse trees between the example translation pairs (Güvenir & Tunç, 1996). However, the difficulty we found was the lack of reliable parsers for both languages. In later work, we have proposed a learning technique (Cicekli & Güvenir, 1996; Güvenir & Cicekli, 1998) to learn *translation templates* from translation examples and store them as *generalized exemplars*, rather than parse trees. A template is defined as an example translation pair, where some components (e.g. word stems and morphemes) are generalized by replacing them with variables in both sentences. In that earlier work, we replaced only the differing parts by variables to obtain a generalized exemplar. In this chapter, we have extended and generalized our learning algorithm by adding new heuristics to form a complete framework for EBMT. In this new framework (Cicekli, 2000; Cicekli and Güvenir, 2001), we are also able to learn generalized exemplars by replacing similar parts in the sentences. These two distinct learning heuristics are called *similarity template learning* and *difference template learning.* These algorithms are also able to learn new translation templates from examples in which the number of differing or similar components between the source and target sentences differs. We refer to this technique as *Generalized Exemplar Based Machine Translation.*

The translation template learning framework presented in this chapter is based on a heuristic to infer correspondences between the patterns in the source and target languages given two translation pairs. According to this heuristic, given two translation examples, if the sentences in the source language exhibit some similarities, then the corresponding sentences in the target language must have similar parts, and they must be translations of the similar parts of the sentences in the source language. Furthermore, the remaining differing constituents of the source sentences should also match the corresponding differences of the target sentences. However, if the sentences do not exhibit any similarities, then no correspondences are inferred.

Our learning algorithm is called *Translation Template Learner* (TTL). Given a corpus of translation pairs, TTL infers correspondences between the source and target languages in the form of templates. These templates can be used for translation in both directions. Therefore, in the rest of the chapter we will refer to these languages as L^1 and L^2. Although the examples and experiments used here are on English and Turkish, we believe the model is equally applicable to many other language pairs.

The rest of the chapter is organized as follows. Section 9.2 explains the representation in the form of translation templates. The TTL algorithm is described in Section 9.3, and some of its performance results are given in Section 9.4. Section 9.5 describes how these translation templates can be used in translation, and the general system architecture. Our system is evaluated in Section 9.6. The limitations of the learning heuristics are described in Section 9.7, and Section 9.8 concludes the chapter with pointers for further research.

2. Translation Templates

A translation template is a generalized translation exemplar pair, where some components (e.g. word stems and morphemes) are generalized by replacing them with variables in both sentences, and establishing bindings between these variables. Let us asume the example translations in (1):

(1) I will drink orange juice $\leftrightarrow$ portakal suyu içeceğim
I will drink coffee $\leftrightarrow$ kahve içeceğim

Given these examples, the translation templates in (2) can be learned using our first learning heuristic:

(2) I will drink $X^1 \leftrightarrow X^2$ içeceğim
if $X^1 \leftrightarrow X^2$
orange juice $\leftrightarrow$ portakal suyu
coffee $\leftrightarrow$ kahve

The first translation template is read as the sentence "I will drink X^1" in L^1 and the sentence "X^2 içeceğim" in L^2 are translations of each other, given that X^1 in L^1 and X^2 in L^2 are translations of each other. Therefore, if it has already been acquired that "tea" in L^1 and "çay" in L^2 are translations of each other, for example, i.e. "tea" $\leftrightarrow$ "çay", then the sentence "I will drink tea" can be easily translated into L^2 as "çay içeceğim". In a similar manner, the sentence "çay içeceğim" in L^2 can be translated in L^1 as "I will drink tea". The second and third translation templates are *atomic* templates representing atomic correspondences of two strings in the languages L^1 and L^2. An *atomic translation template* does not contain any variable. The TTL algorithm also stores the given translation examples as atomic translation templates.

Since the TTL algorithm is based on finding the similarities and differences between translation examples, the representation of sentences plays an important role. As explained above, the TTL algorithm may use the sentences exactly as they are found in a regular text. That is, there is no need for grammatical information or preprocessing on the bilingual parallel corpus. Therefore, it constitutes a grammar-less extraction algorithm for phrasal translation templates from bilingual parallel texts.

For agglutinative languages such as Turkish, this surface level representation of the sentences limits the generality of the templates to be learned. For example, the translation of the sentence "they are running" into Turkish is a single word "koşuyorlar", and the translation of "they are walking" is "yürüyorlar". When a surface level representation is used, it is not possible to find a template from these translation examples. In this case, it is assumed that a sentence is a sequence of words and a word is indivisible. Therefore, we will represent a word as its lexical level representation,[1] that is, its stem and its morphemes. For example, the translation pair "They are running" $\leftrightarrow$ "koşuyorlar" will be represented as "they are run+PROG" $\leftrightarrow$ "koş+PROG+3PL". Similarly, the pair "they are walking" $\leftrightarrow$ "yürüyorlar" will be represented as "they are walk+PROG" $\leftrightarrow$ "yürü+PROG+3PL". Here, the + symbol is used to mark the beginning of a morpheme. In the English and Turkish sentences, the PROG morpheme indicates the progressive tense suffix, while 3PL indicates the third person plural agreement marker. In this case, the sentence is treated as a

[1] In the lexical level representation of Turkish words appearing in the examples, we used the following notations which are similar to the notations used in phrase structure grammar papers (Gazdar *et al.*, 1985; Pollard & Sag, 1987): 1SG, 2SG, 3SG, 1PL, 2PL and 3PL for agreement morphemes; AOR, PAST and PROG for aorist, progressive and past tense morphemes, respectively; ABL for ablative morpheme; ACC for accusative morpheme; LOC for locative morpheme; DAT for dative morpheme; P1SG, P2SG, P3SG, P1PL, P2PL, and P3PL for possessive markers; ConvNoun=DHk for a morpheme (DHk) which is used to convert a verb into a noun. The following notations are used in the lexical level representations of English words appearing in the examples: PAST and PROG for past and progressive tense morphemes (for -ed and -ing suffixes); 3SG for the third person agreement morpheme (for the -s suffix) in the verbs. The surface level realizations of these morphemes are determined according to vowel and consonant harmony rules. The surface level realization of the PAST morpheme for English verbs also depends on whether that verb is regular or irregular.

sequence of morphemes (root words and morphemes). According to this representation, these two translation pairs would be as in (3):

(3) they are run+PROG ↔ koş+PROG+3PL
they are walk+PROG ↔ yürü+PROG+3PL

Using our first heuristic, the translation templates in (4) can be learned from the two translation pairs in (3):

(4) they are X^1+PROG ↔ X^2+PROG+3PL
if $X^1 \leftrightarrow X^2$
run ↔ koş
walk ↔ yürü

This representation allows an abstraction over technicalities such as vowel and/or consonant harmony rules in Turkish, and also different realizations of the same verb according to tense in English. We assume that the generation of surface level representation of words from their lexical level representations is unproblematic.

3. Learning Translation Templates

The TTL algorithm infers translation templates using similarities and differences between two translation examples (E_a, E_b) taken from a bilingual parallel corpus. Formally, a translation example $E_a : E_a^1 \leftrightarrow E_a^2$ is composed of a pair of sentences, E_a^1 and E_a^2 that are translations of each other in L_1 and L_2, respectively.

A *similarity* between two sentences of a language is a non-empty sequence of common items (root words or morphemes) in both sentences. A *difference* between two sentences of a language is a pair of two sequences (D_1, D_2) where D_1 is a sub-sequence of the first sentence, D_2 is a sub-sequence of the second sentence, and D_1 and D_2 do not contain any common item.

Given two translation examples (E_a, E_b), we try to find similarities between the constituents of E_a and E_b. A sentence is considered as a sequence of lexical items (i.e. root words or morphemes). If no similarities can be found, then no template is learned from these examples. If similar constituents do exist, then a *match sequence* $M_{a,b}$ of the form in (5) is generated:

(5) $$S_0^1, D_0^1, S_1^1, \cdots, D_{n-1}^1, S_n^1 \leftrightarrow S_0^2, D_0^2, S_1^2, \cdots, D_{m-1}^2, S_m^2 \quad \text{for } 1 \le n, m$$

Here, S_k^1 represents a similarity (a sequence of common items) between E_a^1 and E_b^1. Similarly, $D_k^1 : (D_{k,a}^1, D_{k,b}^1)$ represents a difference between

E^1_a and E^1_b, where $D^1_{k,a}$ and $D^1_{k,b}$ are non-empty differing items between two similar constituents S^1_k and S^1_{k+1}. Corresponding differing constituents do not contain common items. That is, for a difference D_k, $D_{k,a}$ and $D_{k,b}$ do not contain any common item. Also, no lexical item in a similarity S_i appears in any difference. Any of S^1_0, S^1_n, S^2_0 or S^2_m can be empty, but S^1_i for $0<i<n$ and S^2_j for $0<j<m$ must be non-empty. Furthermore, at least one similarity on each side must be non-empty. Note that given these conditions, there exists either a unique match or no match between two example translation pairs.

For instance, let us assume that the translation examples in (6) are given:

(6) "I bought the book for Cathy" $\leftrightarrow$ "Cathy için kitabı satın aldım"

"I bought the ring for Cathy" $\leftrightarrow$ "Cathy için yüzüğü satın aldım".

The lexical level representations of the example pairs in (6) are shown in (7):

(7) I buy+PAST the book for Cathy $\leftrightarrow$
Cathy için kitap+ACC satın al+PAST+1SG

I buy+PAST the ring for Cathy $\leftrightarrow$
Cathy için yüzük+ACC satın al+PAST+1SG

For the translation examples in (7), the match sequence in (8) is obtained by our matching algorithm.

(8) I buy+PAST the (book,ring) for Cathy $\leftrightarrow$
Cathy için (kitap,yüzük) +ACC satın al+PAST+1SG

That is, the match sequences in (9) are obtained:

(9) $S^1_0 =$ I buy+PAST the,
$D^1_0 =$ (book,ring),
$S^1_1 =$ for Cathy,
$S^2_0 =$ Cathy için,
$D^2_0 =$ (kitap,yüzük),
$S^2_1 =$ +ACC satın al+PAST+1SG.

After a match sequence is found for two translation examples, we use two different learning heuristics to infer translation templates from that match sequence. These two learning heuristics try to locate the corresponding differences or similarities in the match sequence, respectively. If the first heuristic can locate all corresponding differences, a new translation template can be generated by replacing all differences with variables. This translation template is called a *similarity translation template* since it contains the similarities in the match sequence. The second heuristic can infer translation templates by replacing similarities with variables, if it is able to locate corresponding similarities in the match sequence. These translation templates are called *difference translation templates* since they contain differences in the match sequence. Note that both similarity and difference translation templates contain variables.

For each pair of examples in the training set, the TTL algorithm tries to infer translation templates using these two learning heuristics. After all translation templates are learned, they are sorted according to how specific they are. Given two templates, the one that has a higher number of terminals is more specific than the other. Note that the templates are ordered according to the source language. For two-way translation, the templates are ordered once for each language as the source.

3.1 Learning Similarity Translation Templates

If there exists only a single difference in both sides of a match sequence, i.e. $n=m=1$, then these differing constituents must be translations of each other. In other words, we are able to locate the corresponding differences in the match sequence. In this case, the match sequence must be of the form in (10):

(10) $$S_0^1, D_0^1, S_1^1 \leftrightarrow S_0^2, D_0^2, S_1^2$$

Since D_0^1 and D_0^2 are the corresponding differences, the similarity translation template in (11) is inferred by replacing these differences with variables:

(11) $$S_0^1\ X^1\ S_1^1 \leftrightarrow S_0^2\ X^2\ S_1^2 \quad \text{if } X^1 \leftrightarrow X^2$$

Furthermore, the two atomic translation templates in (12) are learned from the corresponding differences $(D_{0,a}^1, D_{0,b}^1)$ and $(D_{0,a}^2, D_{0,b}^2)$:

(12) $$D_{0,a}^1 \leftrightarrow D_{0,a}^2 \qquad D_{0,b}^1 \leftrightarrow D_{0,b}^2$$

For example, since the match sequence given in (8) contains a single difference on both sides, the similarity translation template and two additional atomic translation templates in (13) can be inferred from the corresponding differences (book,ring) and (kitap,yüzük):

(13) I buy+PAST the X^1 for Cathy $\leftrightarrow$
Cathy için X^2+ACC satın al+PAST+1SG
if $X^1 \leftrightarrow X^2$
book $\leftrightarrow$ kitap
ring $\leftrightarrow$ yüzük

On the other hand, if the number of differences are equal on both sides but greater than one, i.e. $1<n=m$, without prior knowledge, it is impossible to determine which difference in one side corresponds to which difference on the other side. In such cases, learning depends on those translation templates acquired previously. Our similarity template learning algorithm tries to locate $n-1$ corresponding differences in the match sequence by checking previously learned translation templates. That is, the k^{th} difference $(D^1_{k,a}, D^1_{k,b})$ on the left side corresponds to the l^{th} difference $(D^2_{l,a}, D^2_{l,b})$ on the right side if the two translation templates in (14) have been learned earlier:

(14) $D^1_{k,a} \leftrightarrow D^2_{l,a} \qquad D^1_{k,b} \leftrightarrow D^2_{l,b}$

After finding $n-1$ corresponding differences, two unchecked differences, one on each side, should correspond to each other. Thus, for all differences in the match sequence, we determine which difference on one side corresponds to which difference on the other. Now, let us assume that the list $CDPair_1, CDPair_2, \ldots, CDPair_n$ represents the list of all corresponding differences where $CDPair_n$ is the pair of two unchecked differences, and each $CDPair_i$ is the pair of two differences in the form $(D^1_{k_i}, D^2_{l_i})$. For each $CDPair_i$, we replace $D^1_{k_i}$ with a variable X^1_i, and $D^2_{l_i}$ with a variable X^2_i in a match sequence $M_{a,b}$. Thus, we get a new match sequence $M_{a,b}WDV$ in which all differences are replaced by proper variables. As a result, the similarity translation template in (15) can be inferred:

(15) $M_{a,b}WDV$
if $X^1_1 \leftrightarrow X^2_1$ **and** $\cdots$ **and** $X^1_n \leftrightarrow X^2_n$

In addition, the atomic translation templates in (16) are learned from the last corresponding differences $(D^1_{k_n,a}, D^1_{k_n,b})$ and $(D^2_{l_n,a}, D^2_{l_n,b})$:

procedure SimilarityTTL($M_{a,b}$)
begin
- Assume that the match sequence $M_{a,b}$ for the pair of translation examples E_a and E_b is:
  $S_0^1, D_0^1, \cdots, D_{n-1}^1, S_n^1, \leftrightarrow S_0^2, D_0^2, \cdots, D_{m-1}^2, S_m^2$

if $n{=}m{=}1$ **then**
- Infer the following templates:
  $S_0^1\ X^1\ S_1^1 \leftrightarrow S_0^2\ X^2\ S_1^2$ if $X^1 \leftrightarrow X^2$
  $D_{0,a}^1 \leftrightarrow D_{0,a}^2$
  $D_{0,b}^1 \leftrightarrow D_{0,b}^2$

else if $1 < n{=}m$ **and** $n{-}1$ corresponding differences can be found in $M_{a,b}$ **then**
- Assume that the unchecked corresponding differences is $((D_{k_n,a}^1, D_{k_n,b}^1), (D_{l_n,a}^2, D_{l_n,b}^2))$.
- Assume that the list of corresponding differences is $(D_{k_1}^1, D_{l_1}^2) \cdots (D_{k_n}^1, D_{l_n}^2)$ including unchecked ones.
- For each corresponding difference $(D_{k_i}^1, D_{l_i}^2)$, replace $D_{k_i}^1$ with X_i^1 and $D_{k_i}^2$ with X_i^2 to get the new match sequence $M_{a,b}WDV$.
- Infer the following templates:
  $M_{a,b}WDV$ if $X_1^1 \leftrightarrow X_1^2$ and $\cdots$ and $X_n^1 \leftrightarrow X_n^2$
  $D_{k_n,a}^1 \leftrightarrow D_{l_n,a}^2$
  $D_{k_n,b}^1 \leftrightarrow D_{l_n,b}^2$

end

Figure 9.1. The Similarity TTL (STTL) Algorithm

(16) $D_{k_n,a}^1 \leftrightarrow D_{l_n,a}^2$
$D_{k_n,b}^1 \leftrightarrow D_{l_n,b}^2$

For example, the translation examples in (17) have two differences on both sides:

(17) I break+PAST the window ↔ pencere+ACC kır+PAST+1SG
You break+PAST the door ↔ kapı+ACC kır+PAST+2SG

The match sequence in (18) is obtained for these examples:

(18) (i,you) break+PAST the (window,door) ↔
(pencere,kapı) +ACC kır+PAST (+1SG,+2SG)

Without prior information, we cannot determine whether "i" corresponds to "pencere" or "+1SG". However, if it has already been learned that "i" corresponds to "+1SG" and "you" corresponds to "+2SG", then the similarity translation template and two additional atomic translation templates in (19) can be inferred:

(19) X_1^1 break+PAST the X_2^1 ↔ X_2^2+ACC kır+PASTX_1^2
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
window ↔ pencere
door ↔ kapı

In general, when the number of differences in both sides of a match sequences is greater than or equal to 1, i.e. $1 \leq n = m$, the similarity TTL (STTL) algorithm learns new similarity translation templates only if at least $n-1$ of the differences have already been learned. A formal description of the similarity TTL algorithm is summarized in Figure 9.1.

3.2 Learning Difference Translation Templates

If there exists only a single non-empty similarity in both sides of a match sequence $M_{a,b}$, then these similar constituents must be translations of each other. In this case, each side of the match sequence can contain one or two differences, and they may contain different number of differences. In other words, each side ($M_{a,b}^i$ where i is 1 or 2) of the match sequence $M_{a,b} : M_{a,b}^1 \leftrightarrow M_{a,b}^2$ can be one of the following:

1 S_0^i, D_0^i, S_1^i, where S_0^i is non-empty, and S_1^i is empty.

2 S_0^i, D_0^i, S_1^i where S_1^i is non-empty, and S_0^i is empty.

3 $S_0^i, D_0^i, S_1^i, D_1^i, S_2^i$, where S_1^i is non-empty, and S_0^i and S_2^i are empty.

In this case, we replace the non-empty similarity in $M_{a,b}^i$ with variable X^i, and separate difference pairs in the match sequence to obtain two match sequences with similarity variables, namely M_aWSV and M_bWSV, as shown in (20):

(20) $M_a^1WSV \leftrightarrow M_a^2WSV$
$M_b^1WSV \leftrightarrow M_b^2WSV$

For example, for the third case given above, M_a^1WSV and M_b^1WSV will be as shown in (21):

(21) $M_a^1WSV: \; D_{0,a}^1 \; X^1 \; D_{1,a}^1$
$M_b^1WSV: \; D_{0,b}^1 \; X^1 \; D_{1,b}^1$

As a result, the two difference translation templates in (22) are learned when there is a single non-empty similarity on both sides of a match sequence:

(22) M_aWSV
  if $X^1 \leftrightarrow X^2$
M_bWSV
  if $X^1 \leftrightarrow X^2$

In addition to these templates, the atomic translation template in (23) is also learned from the corresponding non-empty similarities S_k^1 in $M_{a,b}^1$ and S_l^2 in $M_{a,b}^2$:

(23) $S_k^1 \leftrightarrow S_l^2$

For example, the match sequence in (18) contains a single non-empty similarity on both sides. The two difference translation templates, and one additional atomic template in (24) are learned from the corresponding similarities "break+PAST the" and "+ACC kır+PAST", in the match sequence in (23):

(24) i X^1 window $\leftrightarrow$ pencere X^2 +1SG
  if $X^1 \leftrightarrow X^2$
you X^1 door $\leftrightarrow$ kapı X^2 +2SG
  if $X^1 \leftrightarrow X^2$
break+PAST the $\leftrightarrow$ +ACC kır+PAST

Let us assume that the number of non-empty similarities on both sides is equal to n (i.e. they are equal), and n is greater than 1. Without prior knowledge, it is impossible to determine which similarity in one side corresponds to which similarity in the other side. Our difference template learning algorithm can infer new difference translation templates if it can locate $n-1$ corresponding non-empty similarities. We say that non-empty similarity S_k^1 on the left side corresponds to non-empty similarity

S_l^2 on the right side if the translation template in (25) has been learned earlier:

(25) $S_k^1 \leftrightarrow S_l^2$

After finding n–1 corresponding similarities, there will be two unchecked similarities, one on each side. These two unchecked similarities should correspond to each other. Now, let us assume that the list $CSPair_1, CSPair_2, \ldots, CSPair_n$ represents the list of all corresponding similarities in the match sequence. In that list, each $CSPair_i$ is a pair of two non-empty similarities in the form $(S_{k_i}^1, S_{l_i}^2)$, and $CSPair_n$ is the pair of two unchecked similarities. For each $CSPair_i$, we replace $S_{k_i}^1$ with a variable X_i^1 and $S_{l_i}^2$ with a variable X_i^2 in the match sequence $M_{a,b}$. Then the resulting sequence is divided into two match sequences with similarity variables, namely M_aWSV and M_bWSV, by separating difference pairs in the match sequence. As a result, the two difference translation templates in (26) can be inferred:

(26)
$$M_aWSV$$
$$\textbf{if } X_1^1 \leftrightarrow X_1^2 \textbf{ and } \cdots \textbf{ and } X_n^1 \leftrightarrow X_n^2$$
$$M_bWSV$$
$$\textbf{if } X_1^1 \leftrightarrow X_1^2 \textbf{ and } \cdots \textbf{ and } X_n^1 \leftrightarrow X_n^2$$

In addition, the atomic translation template in (27) is learned from the last corresponding similarities:

(27) $S_{k_n}^1 \leftrightarrow S_{l_n}^2$

For instance, from the match sequence $S_0^1, D_0^1, S_1^1 \leftrightarrow S_0^2, D_0^2, S_1^2$ where all similarities are non-empty, and if the list of corresponding similarities is $(S_0^1, S_1^2), (S_1^1, S_0^2)$, the difference translation templates in (28) can be inferred:

(28)
$$X_1^1 D_{0,a}^1 X_2^1 \leftrightarrow X_2^2 D_{0,a}^2 X_1^2$$
$$\textbf{if } X_1^1 \leftrightarrow X_1^2 \textbf{ and } X_2^1 \leftrightarrow X_2^2$$
$$X_1^1 D_{0,b}^1 X_2^1 \leftrightarrow X_2^2 D_{0,b}^2 X_1^2$$
$$\textbf{if } X_1^1 \leftrightarrow X_1^2 \textbf{ and } X_2^1 \leftrightarrow X_2^2$$

In addition, if (S_1^1, S_0^2) is the pair of two unchecked similarities, the atomic translation template in (29) is learned:

(29) $S_1^1 \leftrightarrow S_0^2$

For example, the match sequence in (8) contains two non-empty similarities. Without prior information, we cannot determine whether "for

Cathy" corresponds to "Cathy için" or "+ACC satın al+PAST+1SG". However, if it has been already learned that "for Cathy" corresponds to "Cathy için", then the two difference translation templates and one additional translation template in (30) can be inferred:

(30) X_1^1 book $X_2^1 \leftrightarrow X_2^2$ kitap X_1^2
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
X_1^1 ring $X_2^1 \leftrightarrow X_2^2$ yüzük X_1^2
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
i buy+PAST the $\leftrightarrow$ +ACC satın al+PAST+1SG

In general, when the number of non-empty similarities on both sides of a match sequence is greater than or equal to 1, i.e. $1 \leq n = m$, the difference TTL (DTTL) algorithm learns new difference translation templates only if at least $n-1$ of the similarities have already been learned. A formal description of the difference TTL algorithm is summarized in Figure 9.2.

---

procedure DifferenceTTL($M_{a,b}$)
begin
if *numofsim*($M^1_{a,b}$) = *numofsim*($M^2_{a,b}$) $= n \geq 1$
and $n-1$ corresponding similarities can be found in $M_{a,b}$ **then**
- Assume that the unchecked corresponding similarities is $(S^1_{k_n}, S^2_{l_n})$.
- Assume that the list of corresponding similarities is $(S^1_{k_1}, S^2_{l_1}) \cdots (S^1_{k_n}, S^2_{l_n})$ including unchecked ones.
- For each corresponding difference $(S^1_{k_i}, S^2_{l_i})$, replace $S^1_{k_i}$ with X^1_i and $S^2_{k_i}$ with X^2_i to get the new match sequence $M_{a,b}WSV$.
- Divide $M_{a,b}WSV$ into M_aWSV and M_bWSV by separating differences.
- Infer the following templates:
  M_aWSV if $X_1^1 \leftrightarrow X_1^2$ and $\cdots$ and $X_n^1 \leftrightarrow X_n^2$
  M_bWSV if $X_1^1 \leftrightarrow X_1^2$ and $\cdots$ and $X_n^1 \leftrightarrow X_n^2$
  $S^1_{k_n} \leftrightarrow S^2_{l_n}$

end

Figure 9.2. The Difference TTL (DTTL) Algorithm

3.3 Different Numbers of Similarities or Differences in Match Sequences

The STTL algorithm given in Section 9.3.1 can learn new translation templates only if the number of differences on both sides of a match sequence are equal. Similarly, the DTTL algorithm requires that a match sequence has to have the same number of similarities on both sides. In this section, we describe how to relax these restrictions so that the STTL and the DTTL algorithms can learn new translation templates from a match sequence with different numbers of differences or similarities, respectively. We try to make the number of differences equal on both sides of a match sequence by separating differences, before the STTL algorithm tries to learn from that match sequence. Similarly, we try to equate the number of similarities on both sides of a match sequence for the DTTL algorithm. For example, the match sequence of the two translation examples ("I came" ↔ "geldim" and "You went" ↔ "gittin") in (31) has one difference on the left side, but it has two differences on the right side:

(31) i come+PAST ↔ gel+PAST+1SG
you go+PAST ↔ git+PAST+2SG

Match Sequence:
(i come,you go) +PAST ↔ (gel,git) +PAST (+1SG,+2SG)

The STTL algorithm given in Section 9.3.1 cannot learn translation templates from this match sequence because the number of differences is not the same. Since both constituents of the difference on the left side contain two morphemes, we can separate that difference into two differences by dividing both constituents of that difference into two parts given morpheme boundaries. As a result, we obtain the match sequence in (32):

(32) (i,you) (come,go) +PAST ↔ (gel,git) +PAST (+1SG,+2SG)

Now, the match sequence in (32) has two differences on both sides. If we know that (i,you) corresponds to (+1SG,+2SG), we can learn the translation templates in (33):

(33) $X_1^1\ X_2^1$ +PAST $\leftrightarrow$ X_2^2 +PAST X_1^2
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
come $\leftrightarrow$ gel
go $\leftrightarrow$ git

In general, before we apply the STTL algorithm to a match sequence, we try to create an instance of that match sequence with the same number of differences on both sides by dividing a difference into several differences. A difference (D_a, D_b) can be divided into two differences (D_{a_1}, D_{b_1}), and (D_{a_2}, D_{b_2}) if the lengths of D_a and D_b are greater than 1. The reader should note that if D_{a_1}, D_{a_2}, D_{b_1} and D_{b_2} are non-empty, the equalities $D_a = D_{a_1} D_{a_2}$ and $D_b = D_{b_1} D_{b_2}$ hold. We continue to create an instance of a match sequence with the same number of differences until new translation templates can be learned from that instance, or until there is no other way to create an instance with the same number of differences. We may need to create an instance of the original match sequence even if it has the same number of differences on both sides. For example, the match sequence of the translation examples "I drank water" $\leftrightarrow$ "su içtim" and "You ate orange" $\leftrightarrow$ "portakal yedin" has two differences on both sides, as shown in (34):

(34) i drink+PAST water $\leftrightarrow$ su iç+PAST+1SG
you eat+PAST orange $\leftrightarrow$ portakal ye+PAST+2SG

Match Sequence:
(i drink,you eat) +PAST (water,orange) $\leftrightarrow$
(su iç,portakal ye) +PAST (+1SG,+2SG)

Now, let us assume that we do not know whether the difference (i drink,you eat) corresponds to (su iç,portakal ye) or (+1SG,+2SG), or whether the difference (water,orange) corresponds to (su iç,portakal ye) or (+1SG,+2SG). In fact, none of these correspondences should hold because they will all yield incorrect translation templates. However, if we divide the differences on both sides, we obtain the match sequence in (35), with three differences on both sides:

(35) (i,you) (drink,eat) +PAST (water,orange) $\leftrightarrow$
(su,portakal) (iç,ye) +PAST (+1SG,+2SG)

From this match sequence, if we know two correspondences between the differences above, such as (i,you) corresponds to (+1SG,+2SG), and (water,orange) corresponds to (su,portakal), we can learn the translation templates in (36):

(36) $X_1^1\ X_2^1$ +PAST $X_3^1 \leftrightarrow X_3^2\ X_2^2$ +PAST X_1^2
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$ **and** $X_3^1 \leftrightarrow X_3^2$
drink $\leftrightarrow$ iç
eat $\leftrightarrow$ ye

For the DTTL algorithm, we divide similarities to equate the number of similarities on both sides of a match sequence. A similarity S can be divided into two non-empty similarities S_1 and S_2 in order to increase the number of similarities in one side. Before the DTTL algorithm is executed, we try to equate the number of similarities on both sides. We continue to create an instance of a match sequence with the same number of similarities until the DTTL algorithm can learn new translation templates from this instance or until there is no other way to create an instance.

For example, from the match sequence of the translation examples "I came" $\leftrightarrow$ "geldim" and "I went" $\leftrightarrow$ "gittim", the DTTL algorithm cannot learn new templates because it contains two similarities on the left side and one on the right side, namely those in (37):

(37) i come+PAST $\leftrightarrow$ gel+PAST+1SG
i go+PAST $\leftrightarrow$ git+PAST+1SG

Match Sequence:
i (come,go) +PAST $\leftrightarrow$ (gel,git) +PAST+1SG

On the other hand, we can divide the similarity "+PAST+1SG" into two similarities "+PAST" and "+1SG" by inserting an empty difference between them. Now the new match sequence has two similarities on both sides. If the correspondence of "i" to "+1SG" is already known, the translation templates in (38) can be learned by the DTTL algorithm:

(38) X_1^1 come $X_2^1 \leftrightarrow$ gel $X_2^2\ X_1^2$
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
X_1^1 go $X_2^1 \leftrightarrow$ git $X_2^2\ X_1^2$
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
+PAST $\leftrightarrow$ +PAST

3.4 Differences with Empty Constituents

The current matching algorithm does not allow a difference to contain an empty constituent. For this reason, the matching algorithm fails for certain translation example pairs although we may learn useful translation templates from those pairs. For example, the current matching algorithm fails for the following examples "I saw the man" $\leftrightarrow$ "adamı gördüm" and "I saw a man" $\leftrightarrow$ "bir adam gördüm" because "bir" and "+ACC" have to match empty strings, as shown in (39):

(39) i see+PAST the man $\leftrightarrow$ adam+ACC gör+PAST+1SG
i see+PAST a man $\leftrightarrow$ bir adam gör+PAST+1SG

However, if we relax this restriction in the matching algorithm by allowing a difference to have an empty constituent, then this new version of the matching algorithm will find the match sequence in (40) for the example in (39):

(40) i see+PAST (the,a) man $\leftrightarrow$
(ϵ,bir) adam (+ACC,ϵ) gör+PAST+1SG

In this match sequence, "bir" in the difference (ϵ,bir) and "+ACC" in the difference (+ACC,ϵ) correspond to the empty string. If we apply the DTTL algorithm to this match sequence by assuming that the correspondence of "man" to "adam" is already known, the translation templates in (41) can be learned:

(41) X_1^1 the $X_2^1 \leftrightarrow X_2^2$ +ACC X_1^2
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
X_1^1 a $X_2^1 \leftrightarrow$ bir X_2^2 X_1^2
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
i see+PAST $\leftrightarrow$ gör+PAST+1SG

We do not apply the STTL algorithm to a match sequence containing a difference with an empty constituent. If we apply the STTL algorithm to this kind of a match sequence, a translation template with one side empty can be generated. This would mean that a non-empty string in a language always corresponds to an empty string in another language. This is not a plausible situation. For this reason, we only apply the DTTL algorithm to this kind of match sequence in order to prevent the problem mentioned above. We only try to create a match sequence with a difference having an empty constituent if the original match algorithm cannot find a match sequence without differences which contains no empty constituents.

3.5 Complete Learning Examples

In this section, we describe the behavior of our learning algorithms by giving the details of algorithm steps on the two translation example pairs in (43) and (48). We assume that the two translation templates in (42) have been learned earlier:

(42) i ↔ +1SG
you ↔ +2SG

Assume the translation example pair in (43):

(43) I drank wine ↔ Şarap içtim
You drank beer ↔ Bira içtin

Since our learning algorithms work on the lexical form of sentences, the input for our algorithm will be the two translation examples in (44):

(44) i drink+PAST wine ↔ şarap iç+PAST+1SG
you drink+PAST beer ↔ bira iç+PAST+2SG

We now try to find a match sequence between these two translation examples. To do that, a match sequences between the English sentences "i drink+PAST wine" and "you drink+PAST beer" is found, and a match sequence between the Turkish sentences "şarap iç+PAST+1SG" and "bira iç+PAST+2SG" is found. As a result, the match sequence in (45) is obtained between these two translation examples:

(45) (i,you) drink+PAST (wine,beer) ↔
(şarap,bira) iç+PAST (+1SG,+2SG)

We then try to apply the STTL and DTTL algorithms to this match sequence. Since there is an equal number of differences (namely, two) on both sides, the STTL algorithm is applicable to this match sequence. The STTL algorithm can learn new translation templates from this match sequence, if it can determine the corresponding differences. Given the correspondence between (i,you) and (+1SG,+2SG), (wine,beer) should correspond to (şarap,bira). Thus, the STTL algorithm infers the three translation templates in (46) from this match sequence:

(46) X_1^1 drink+PAST $X_2^1 \leftrightarrow X_2^2$ iç+PAST X_1^2
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
wine ↔ şarap
beer ↔ bira

Since there is an equal number of similarities (namely, one) on both sides, the DTTL algorithm is also applicable to this match sequence. Since there is only one similarity on both sides, these similarities ("drink+PAST" and "iç+PAST") should correspond to each other. Thus, the DTTL algorithm infers the three translation templates in (47) from this match sequence:

(47) i X_1^1 wine ↔ şarap X_1^2 +1SG
  if $X_1^1 \leftrightarrow X_1^2$
 you X_1^1 beer ↔ bira X_1^2 +2SG
  if $X_1^1 \leftrightarrow X_1^2$
 drink+PAST ↔ iç+PAST

Let us now assume the translation example pair in (48):

(48) I drank a glass of white wine ↔ Bir bardak beyaz şarap içtim
 You drank a glass of red wine ↔ Bir bardak kırmızı şarap içtin

The actual input for our algorithm will be the two translation examples in lexical form in (49):

(49) i drink+PAST a glass of white wine ↔
  bir bardak beyaz şarap iç+PAST+1SG
 you drink+PAST a glass of red wine ↔
  bir bardak kırmızı şarap iç+PAST+2SG

Then, a match sequence between the English sentences "i drink+PAST a glass of white wine" and "you drink+PAST a glass of red wine" will be found, as will a match sequence between the Turkish sentences "bir bardak beyaz şarap iç+PAST+1SG" and "bir bardak kırmızı şarap iç+PAST+2SG". As a result, the match sequence in (50) is found between these two translation examples:

(50) (i,you) drink+PAST a glass of (white,red) wine ↔
  bir bardak (beyaz,kırmızı) şarap iç+PAST (+1SG,+2SG)

We then try to apply the STTL and DTTL algorithms to this match sequence. Since there is an equal number of differences (namely, two) on both sides, the STTL algorithm is applicable to this match sequence. The STTL algorithm can learn new translation templates from this match sequence if it can determine the corresponding differences. Since a correspondence between (i,you) and (+1SG,+2SG) has been given, (white,red) should correspond to (beyaz,kırmızı). Thus, the STTL algorithm infers the three translation templates in (51) from this match sequence:

(51) X_1^1 drink+PAST a glass of X_2^1 wine $\leftrightarrow$
bir bardak X_2^2 şarap iç+PAST X_1^2
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
white $\leftrightarrow$ beyaz
red $\leftrightarrow$ kırmızı

Since there is an equal number of similarities (two) on both sides, the DTTL algorithm is also applicable to this match sequence, but we cannot determine the similarity correspondences in this match sequence. In other words, we cannot know whether "drink+PAST a glass of" corresponds to "bir bardak" or "şarap iç+PAST". Therefore, the DTTL algorithm cannot directly learn any new translation template from this match sequence. In this case, we look at instances of this match sequence. A suitable instance should contain an equal number of similarities on both sides, in which cases and correspondences of similarity can be determined. One of the instances of this match sequence can be obtained by separating the similarity "drink+PAST a glass of" into the similarities "drink+PAST" and "a glass of", and by separating the similarity "şarap iç+PAST" into the similarities "şarap" and "iç+PAST". Now we have three similarities on both sides in this instance of the original match sequence. From the first example, we learned the correspondence between "wine" and "şarap", and the correspondence between "drink+PAST" and "iç+PAST". Furthermore, the similarity "a glass of" should correspond to "bir bardak". Since all similarity correspondences can be determined in this instance, the three translation templates in (52) can be inferred from this instance by the DTTL algorithm:

(52) i X_1^1 X_2^1 red X_3^1 $\leftrightarrow$ X_2^2 kırmızı X_3^2 X_1^2 +1SG
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$ **and** $X_3^1 \leftrightarrow X_3^2$
you X_1^1 X_2^1 white X_3^1 $\leftrightarrow$ X_2^2 beyaz X_3^2 X_1^2 +2SG
if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$ **and** $X_3^1 \leftrightarrow X_3^2$
a glass of $\leftrightarrow$ bir bardak

In this example, we looked at the instances of the original match sequence because we could not learn translation templates from it. In this kind of situation, we continue to generate instances of the original match sequence until a translation template can be learned, or until there are no more instances of the original match sequence. In the first example, we did not generate instances of the original match sequence because we were able to learn translation templates from the original one.

Table 9.1. Performance Results for The Training Set with 747 Training Pairs

We applied only the STTL algorithm without dividing differences in match sequences and without having match sequences with an empty difference constituent.	• In the first pass, the STTL algorithm learned 642 translation templates. • No new templates were learned in the second pass. • Each pass took about 53 seconds real time.
We applied only the DTTL algorithm without dividing similarities in match sequences and without having match sequences with an empty difference constituent.	• In the first pass, the DTTL algorithm learned 812 translation templates. • In the second pass, using the initial pairs and these new translation templates, the DTTL algorithm inferred 6 more templates. • No new templates were learned in the third pass. • Each pass took about 54 seconds real time.
We applied both the STTL and the DTTL algorithms on the training set without dividing similarities or differences in match sequences and without having match sequences with an empty difference constituent.	• In the first pass, 1239 translation templates were learned. • In the second pass, the TTL algorithms inferred 6 more templates. • No new templates were learned in the third pass. • Each pass took about 81 seconds real time.
We applied both the STTL and the DTTL algorithms on the training set with dividing similarities or differences in match sequences and without having match sequences with an empty difference constituent.	• In the first pass, 1330 translation templates were learned. • In the second pass, the TTL algorithms inferred 11 more templates. • No new templates were learned in the third pass. • Each pass took about 101 seconds real time. • By dividing similarities or differences, 8% more new templates were learned costing 25% more on the learning time.
We applied both the STTL and the DTTL algorithms on the training set with dividing similarities or differences in match sequences and with having match sequences with an empty difference constituent.	• In the first pass, 2055 translation templates were learned. • In the second pass, the TTL algorithms inferred 55 more templates. • No new templates were learned in the third pass. • Each pass took about 170 seconds real time. • By having match sequences with an empty difference constituent, 57% more new templates were learned costing 68% more on the learning time.

4. Performance Results

In order to evaluate our TTL algorithms empirically, we have implemented them in Prolog and evaluated them on medium-sized bilingual parallel texts. Our training sets are artificially collected because of the unavailiabilty of a large morphologically tagged parallel text between English and Turkish.

In each pass of the learning phase, we applied our learning algorithms for each pair of translation examples in a training set. Since the number of pairs is $\sum_{i=1}^{n-1} i$ when the number of translation examples in a training set is n, the time complexity of each pass of the learning phase is $O(n^2)$. The learning phase continues until its last pass cannot learn any new translation templates. In other words, when the number of new learned translation templates is zero in a pass, the learning process terminates. Although the maximum number of passes of the learning phase theoretically is $n-2$, the maximum number of passes which the learning phase had to do on our training sets was 4. This means that the worst case time complexity of our learning algorithm is $O(n^3)$, but in practice it remained in $O(n^2)$.

One of our training sets contained 747 training pairs, which is enough for the system to learn a small coverage of the basics of English grammar. To find the cost/gain of each portion of our learning algorithm, we applied the different portions of our algorithms to this training set. We obtained the results in Table 9.1 for this training set on a SPARC 20/61 workstation.

5. System Architecture and Translation

The templates learned by the TTL algorithm can be used directly in the translation phase. These templates are in lexical form, and they can be used for translation in both directions. The general system architecture is given in Figure 9.3. As can be seen, the input for the learning module is a set of bilingual examples in lexical form. In order to create a set of bilingual examples in lexical form, a set of bilingual examples in their surface form is created, following which all words in these examples are morphologically tagged using Turkish and English morphological analyzers. In this process, a morphological analyzer produces possible lexical forms of a word from its surface form, and the correct lexical form is selected by a human expert. Thus, a set bilingual examples in lexical form is created. Of course, if morphologically tagged sets of bilingual examples existed between English and Turkish, this step would become redundant.

From the surface form of a sentence, the lexical form of that sentence is created by replacing every word with its correct lexical form. Non-words such as punctuation markers are assumed to have the same lexical and surface form. The only exception to this is a punctuation marker depicting the end of sentences. In this case, we delete these completely from the lexical forms.

In the translation process, a given source language sentence in surface form is translated into the corresponding target language sentence in surface form. The outline of the translation process is given below:

1 First, the lexical level representation of the input sentence to be translated is derived by using the source language lexical analyzer.

2 Then any translation templates matching the input are collected. They are collected from most to least specific. For each selected template, its variables are instantiated with the corresponding values in the source sentence. Then we search for templates matching these bound values. If they are found successfully, target language variables are replaced by the values in the matching template.

3 Finally, the surface level representation of the sentence obtained in the previous step is generated by the target language morphological generator.

Note that if the sentence in the source language is ambiguous, then templates corresponding to each sense will be retrieved, and the corresponding sentences for each sense will be generated. The user can choose the right one among the possible translations according to the context. We hope that the correct answer will be among the first results generated in the translation steps by imposing the constraint that templates are to be used from most to least specific. Although this helps to obtain the correct answer among the top results, it may not be enough. We also investigated using a statistical method (Öz & Cicekli, 1998) to order our learned translation templates. In this statistical method, we assign a confidence factor to the learned translation templates, and we use these confidence factors to sort the results of translations. The training data is again used to collect this statistical information. Using this statistical method, the percentage of the correct results is increased by 50% in the top 5 results.

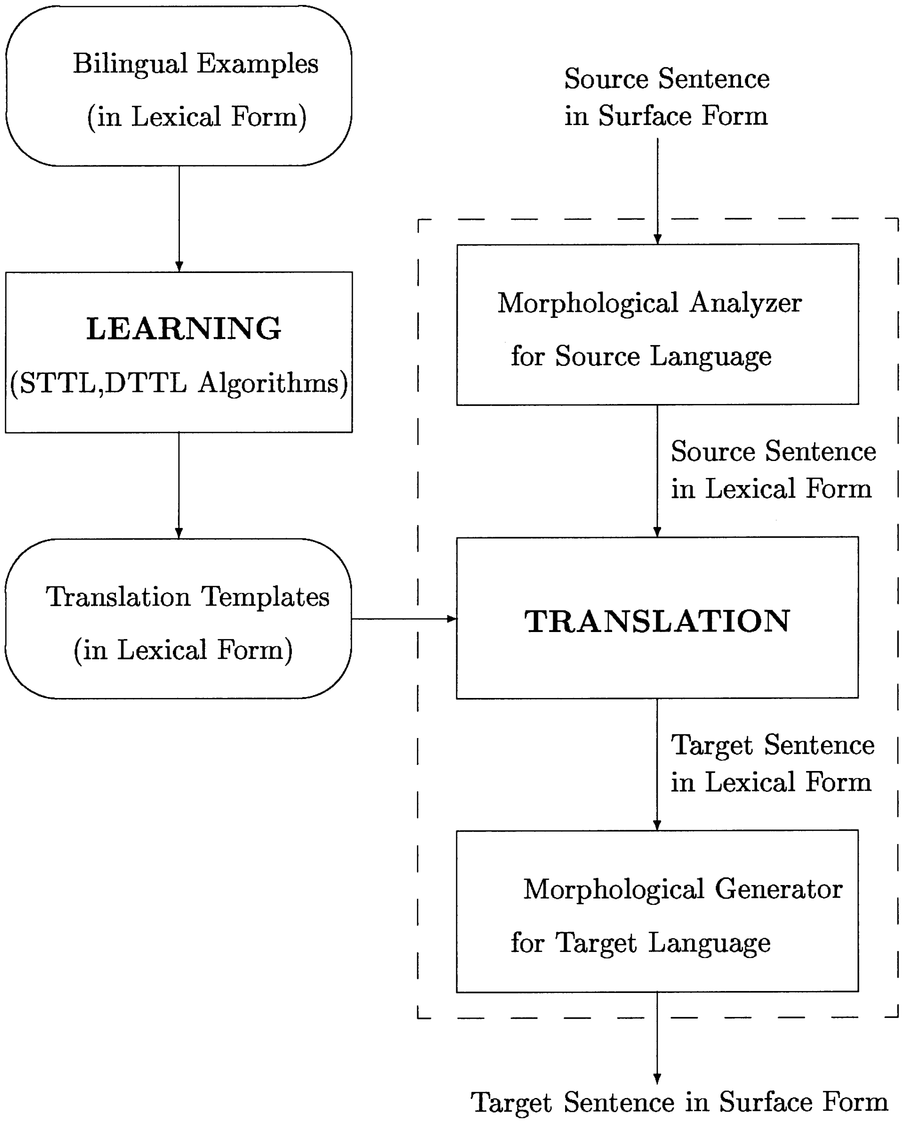

Figure 9.3. The System Architecture

6. Evaluation

Since the TTL algorithm can overspecialize, useless and incorrect templates can be learned. Given this, together with the problem of ambiguity noted above, the translation results produced by our translation algorithm can contain incorrect translations in addition to correct ones. However, our main goal is to obtain correct translations among the top-ranked results. For example, according to our results given in Öz & Cicekli, 1998, we show that the percentage of correct results among the total results is 33%. If we just use the templates according to how specific they are, the percentage of correct results is increased to 44% in the top 5 results. This means that at least 2 of the top 5 results are correct. In addition to our ordering constraint, using the statistical method described in Öz & Cicekli, 1998 increased the percentage of the correct results to 60%. In addition, we investigated whether the top results contain at least one correct translation or not. When we just use our ordering constraint, the top 5 results of 77% of all translations contained at least one correct translation. When the statistical method is used together with our ordering constraint, the percentage is increased to 91%. Thus, a human expert can choose the correct answer by just looking at the top few results.

Our algorithms are tested on training sets constructed by us and others. We only morphologically tagged the training sets prepared by others. Although these training sets are not huge by other standards, they are big enough to be treated as real corpora. One aspect of our future work will be to test our algorithms on huge, morphologically tagged bilingual corpora between other languages (unfortunately there is no huge bilingual corpus between English and Turkish, but we are trying to construct one). The next language pair that we are planning to work on is English and French.

The success of an MT system can be measured according to two criteria: coverage and correctness. Coverage is the percentage of the sentences which can be translated, and correctness is the percentage of correct translations among all translation results produced by that system. However, no MT system is able to guarantee correctness and complete coverage. That is, no MT system will always produce the correct translation for any given sentence, nor produce some translation for any given sentence. This is a direct consequence of the complexity and inherent ambiguity of natural languages. Since natural languages are dynamic, new words enter the language, or new meanings are assigned to old words in time. For the case of English, the word *Internet* is a new addition and the word *web* has a new meaning. In addition, words and sentences are interpreted differently depending on their context. The best way to cope with such issues is to have a translation system that can learn and adapt itself to the changes in the language and the context. The TTL algorithms presented in this chapter achieve this by learning new templates corresponding to the new meaning of the words and interpretation of the sentences from new translation examples.

As a whole, our system can be seen as a human-assisted EBMT system. Our system suggests possible translations which usually contain the correct translation among the top few candidates, and a human expert then chooses the correct translation just by evaluating the given results. The coverage of our system depends on the coverage of the given training sets and how much our learning algorithms learn from these training sets. When the size of training sets is increased, the coverage of our system also increases. Although we cannot say that our learning algorithms can extract all available information in training sets, they can extract most of the available information as translation templates. In measuring the correctness of our system, one needs to look at whether the top few results contain the correct translation or not. In order to increase the level of correctness, we imposed a constraint on the use of the translation templates according to how specific they were, and assigned confidence factors to them. This helps ensure that the correct translation is found among the top few results. The general performance of our system and other EBMT systems depends firstly on the quality of the bilingual corpora used, because they constitute the sole information source, and secondly as to how the available information in the corpora is used in the translation process.

7. Limitations of Learning Heuristics

The preconditions in the definition of the match sequence (see Section 9.3) may appear to be very strong, and they may restrict the practical usage of our learning algorithms. These preconditions are stated as explicitly and strongly as they could be to reduce the number of the useless translation templates which can be learned from match sequences in the expense of missing the opportunity of learning some useful translation templates.

Let us consider the 'translation' examples between American and British English in (53)–(55):

(53) The other day, the president analyzed the state of the union ↔
The other day, the president analysed the state of the union

(54) Recently, the president analyzed the state of the union ↔
Recently, the president analysed the state of the union

(55) Recently, the president analyzed the union ↔
Recently, the president analysed the union

Note that the only difference between source and target sentences here is the alternative spellings of *analyse* vs. *analyze.* Despite the very strong similarity between these sentence pairs, our learning heuristics will not learn any translation templates from these examples. The reader will notice that the lexical item "the" appears 4, 3 and 2 times in the sentences of the three examples in (53)–(55), respectively. Therefore, the lexical item "the" will try to end up in both a similarity and a difference in a match sequence of any two of these examples because the sentences in any two of these examples do not contain same number of lexical items of the form "the". Since we do not allow the same lexical item to appear in both a similarity and a difference, any pair of these examples cannot have a match sequence. Since our system will not learn any translation examples, it will not be able to 'translate' the sentence in (56):

(56) The other day, the president analyzed the union

into British English when only the examples in (53) are given.

Therefore, our learning algorithms can only learn if there is a match sequence between the examples. On the other hand, if we supply two more examples, namely those in (57)–(58):

(57) He analyzed today's situation ↔
He analysed today's situation

(58) Recently, the president analyzed today's situation ↔
Recently, the president analysed today's situation

our learning algorithms will be able learn the required translation templates from the examples (53)–(58). Some of the learned templates will be those in (59):

(59) X_1 analyzed $X_2 \leftrightarrow Y_1$ analysed Y_2
if $X_1 \leftrightarrow Y_1$ **and** $X_2 \leftrightarrow Y_2$
today's situation $\leftrightarrow$ today's situation
He analyzed $\leftrightarrow$ He analysed
Recently, the president analyzed $\leftrightarrow$ Recently, the president analysed
The other day, the president $\leftrightarrow$ The other day, the president
Recently, the president $\leftrightarrow$ Recently, the president
the union $\leftrightarrow$ the union
the state of the union $\leftrightarrow$ the state of the union
analyzed $\leftrightarrow$ analysed
He $\leftrightarrow$ He

These templates will be enough to 'translate' the sentence (55) to British English.

Let us examine the consequences of relaxing the conditions for the definition of a match sequence. If we let a lexical item appear in a similarity and a difference of a match sequence, we will no longer have a unique match sequence for any given two strings and there may be more than one match sequence for those strings. For instance, the American English sentences in the examples (54) and (55) will have the five match sequences in (60) in this case:

(60)

- Recently, the president analyzed the (state of the,ϵ) union
- Recently, the president analyzed (the state of,ϵ) the union
- Recently, (the president analyzed,ϵ) the (state of,president analyzed) the union
- Recently, (the president analyzed the state of,ϵ) the (ϵ,president analyzed the) union
- Recently, (ϵ,the president analyzed) the (president analyzed the state of the,ϵ) union

The British sentences in those examples will also have 5 match sequences. Thus, examples (54) and (55) will have 25 different match sequences because there will be five match sequences for each side of those sentences in this situation. Since only some of these match sequences will be correct, we may learn a lot of useless (wrong) templates from the rest of these match sequences. This is the main reason for insisting on such strong preconditions on the match sequences.

Nevertheless, our learning algorithms may still learn useless wrong translation templates. For example, let us consider the two examples in (61):

(61) I know hardly anybody ↔ Hemen hemen hiç kimseyi tanımam
You know almost everything ↔ Hemen hemen her şeyi bilirsin

From these examples, the correspondence of "know" and "hemen hemen" will be inferred, even though it is wrong. There are two reasons for this problem. First, Turkish differentiates two meanings of "know" as "tanımak" ("to know somebody") and "bilmek" ("to know something"). Second, English phrases "hardly" and "almost" map to the same Turkish phrase ("hemen hemen"). Of course, there can be other situations in which wrong translation templates can be inferred. In order to reduce the effect of these wrong translations, we have also incorporated the statistical methods described in Öz & Cicekli, 1998 into our system. According to these statistical methods, each translation result is associated with a computed confidence factor (a value between 0 and 1) and the translation results are sorted with respect to these computed confidence factors. Each translation template is associated with a confidence factor during the learning phase, and the confidence factor of a translation result is computed from the confidence factors of the translation templates that are used to find that translation result.

8. Conclusion

In this chapter, we have presented a model for learning translation templates between two languages. The model is based on a simple pattern matcher. We integrated this model with an EBMT system to form our Generalized Exemplar-Based Machine Translation model. We have implemented this model as the TTL (Translation Template Learner) algorithms.

The TTL algorithms are illustrated in learning translation templates between Turkish and English. We believe that the approach is applicable to many pairs of natural languages (at least for West-European languages such as English, French, and Spanish), assuming sets of morphologically tagged bilingual examples.

The major contribution of this chapter is that the proposed TTL algorithm eliminates the difficult task of manually encoding the translation templates. The TTL algorithm can work directly on the surface level representation of sentences. However, in order to generate useful translation patterns, it is helpful to use the lexical level representations. For English and Turkish, at least, it is quite trivial to obtain the lexical level representations of words.

Our main motivation was that the underlying inference mechanism is compatible with one of the ways humans learn languages, i.e. learning from examples. We believe that in everyday usage, humans learn general sentence patterns using the similarities and differences between many different example sentences that they are exposed to. This observation led us to the idea that a computer can be trained in a similar fashion, using analogy within a corpus of example translations.

The techniques presented here can be used in an incremental manner. Initially, a set of translation templates can be inferred from a set of translation examples, with extra templates learnable from another set with the help of previously learned translation templates. In other words, the templates learned from previous examples help in learning new templates from new examples, as in the case of natural language learning by humans. This incremental approach allows us to incorporate existing translation templates when new translation examples become available, instead of re-running previous sets of examples along with the new set of examples.

The learning and translation times on the small training set are quite reasonable, and that indicates the program will scale up to large, naturally occurring training corpora. Note that this algorithm is not specific to the English and Turkish languages, but should be applicable to the task of learning to translate automatically between many pairs of languages. Although the learning process on a large corpus will take a considerable amount of time, it is a task to be performed only once. Once the translation templates are learned, the translation process is fast.

The model that we have proposed in this chapter may be integrated with other systems as a natural language front-end, where a small subset of a natural language is used. This algorithm can be used to learn to translate user queries into the language of the underlying system.

This model may also be integrated with an intelligent tutoring system (ITS) for second language learning. The template representation model provides a level of information that may help in error diagnosis and student modelling tasks of an ITS. The model may also be used to tune the teaching strategy according to the needs of the student by analysing the student answers analogically with the closest cases in the corpus. Specific corpora may be designed to concentrate on certain topics that will help in the student's acquisition of the target language. The work presented in this chapter provides an opportunity to evaluate this possibility as a future research topic.

Acknowledgments

This research has been supported in part by *NATO* Science for Stability Program Grant TU-LANGUAGE and The Scientific and Technical Council of Turkey (TÜBİTAK) Grant EEEAG-244.

References

Cicekli, I. 2000. Similarities and Differences. In *SCI2000: Proceedings of the Multi-Conference on Systemics, Cybernetics and Informatics*, Orlando, FL., pp.331–337.

Cicekli, I. and H.A. Güvenir. 1996. Learning Translation Rules From A Bilingual Corpus. In *NeMLaP-2: Proceedings of the 2nd International Conference on New Methods in Language Processing*, Ankara, Turkey, pp.90–97.

Cicekli, I. and H.A. Guvenir. 2001. Learning Translation Templates from Bilingual Translation Examples. *Applied Intelligence* **15**(1):57–76.

Gazdar, G., E. Klein, G.K. Pullum, and I.A. Sag. 1985. *Generalized Phrase Structure Grammar.* Blackwell, Cambridge, MA.

Güvenir, H.A., and A. Tunç. 1996. Corpus-Based Learning of Generalized Parse Tree Rules for Translation. In Gord McCalla (ed.) *New Directions in Artificial Intelligence: Proceedings of the 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence.* Springer-Verlag, LNCS 1081, Toronto, Ontario, Canada, pp.121–131.

Güvenir, H. A, and I. Cicekli. 1998 Learning Translation Templates from Examples. *Information Systems* **23**(6):353–363.

Hammond, K.J. (ed.). 1989. *Proceedings of the DARPA Case-Based Reasoning Workshop, Pensacola Beach, Florida*, San Mateo, CA: Morgan Kaufmann.

Kolodner, J.L. (ed.). 1988. *Proceedings of the DARPA Case-Based Reasoning Workshop, Clearwater Beach, Florida*, San Mateo, CA: Morgan Kaufmann.

Medin, D.L., and M.M. Schaffer. 1978. Context Theory of Classification Learning. *Psychological Review*, **85**:207–238.

Öz, Z., and I. Cicekli. 1998. Ordering Translation Templates by Assigning Confidence Factors. In D. Farwell, L. Gerber and E. Hovy (eds.) *Machine Translation and the Information Soup: Third Conference of the Association for Machine Translation in the Americas, AMTA '98* (Lecture Notes in Computer Science 1529), Berlin: Springer, pp.51-61.

Pollard, C. and I. Sag. 1987. *Information-based Syntax and Semantics.* CSLI, Stanford, CA.

Ram, A. 1993. Indexing, Elaboration and Refinement: Incremental Learning of Explanatory Cases. In J.L. Kolodner (ed.) *Case-Based Learning*, Kluwer Academic Publishers, Dordrecht, The Netherlands.