Applied Intelligence 15, 57–76, 2001

Learning Translation Templates from Bilingual Translation Examples

ILYAS CICEKLI AND H. ALTAY GÜVENIR
Department of Computer Engineering, Bilkent University, TR-06533 Bilkent, Ankara, Turkey
ilyas@cs.bilkent.edu.tr
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, the similar parts of the sentences in the source language must correspond to the similar parts of the sentences in the target language. Similarly, the different parts 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 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, example-based machine translation, corpus-based machine translation, templates

1. Introduction

Due to the requirement of the large-scale knowledge in traditional machine translation (MT) systems, some researchers looked at the alternative methods in machine translation. A traditional knowledge-based machine translation system such as KBMT-89 [1] requires large-scale knowledge resources such as lexicons, grammar rules, mapping rules and an ontology. Acquiring these knowledge resources manually is a time consuming and expensive process. For this reason, researchers have been studying the ways of automatically acquiring some portions of the required knowledge. In the KANT [2] system, which is an immediate descendant of the KBMT-89, a technique for automatic acquisition of the lexicon from a large corpus is used [3]. The technique presented here aims at acquiring all required knowledge except morphological rules for the machine translation task from sentence-level aligned bilingual text corpora only.

Corpus-based machine translation is one of alternative directions that have been proposed to overcome the acquisition problem in traditional systems. There are two fundamental approaches in corpus-based MT: *statistical* and *example-based* machine translation (EBMT). All corpus-based approaches assume the existence of a bilingual parallel text (an already translated corpus) to derive the translation of an input. While statistical MT techniques use statistical metrics to choose the most probable structures in the target language, EBMT techniques employ pattern matching techniques to translate subparts of the given input.

EBMT, originally proposed by Nagao [4], is one of the main approaches of corpus-based machine translation. The main idea behind EBMT is that a given input sentence in the source language is compared with the example translations in the given bilingual parallel text to find the closest matching examples so that these examples can be used in the translation of that input sentence. After finding the closest matchings for the sentence in the source language, parts of the corresponding target language sentence are constructed using structural equivalences and deviances in those matches. Following Nagoa's original proposal,

several machine translation methods that utilize bilingual corpora have been studied [5–10]. Some researchers [11, 12] only utilized bilingual corpora to create a bilingual dictionary and use it during the translation process. In other words, they aligned bilingual corpora at word level to figure out corresponding words in languages. Bilingual corpora is also aligned at phrase level by some other researchers [13–15]. But these correspondences between two languages are only accomplished at atomic level, and they are used in the translation of portions of sentences. Kaji [16] tried to learn correspondences of English and Japanese syntactic structures from bilingual corpora. This is similar to our early work [17] and it needs reliable parsers for both source and target languages. The technique described here learns not only atomic correspondences between two languages, but also general templates describing structural correspondence (not syntactic structure) from bilingual corpora.

Researchers in Machine Learning (ML) community have widely used exemplar-based representation. Medin and Schaffer [18] were the first researchers who proposed exemplar-based learning as a model of human learning. The characteristic examples stored in the memory are called *exemplars*. The basic idea in exemplar-based learning is to use past experiences or cases to understand, plan, or learn from novel situations [19–21]. In EBMT, translation examples should be available prior to the translation of an input sentence. In most of the EBMT systems, these translation examples are directly used without any generalization. Kitano [6] manually encoded translation rules, however this is a difficult and error-prone task for a large corpus. In this paper, we formulate the acquisition of translation rules as a machine learning problem in order to automate this task.

Our first attempt was to construct parse trees between the example translation pairs [17]. However, the difficulty was the lack of reliable parsers for both languages. Later, we have proposed a learning technique [22, 23] 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., words stems and morphemes) are generalized by replacing them with variables in both sentences. In that early work, we only replaced differing parts by variables to get a generalized exemplar. In this paper, we have extended and generalized our learning algorithm by adding new heuristics to form a complete framework for EBMT. In this new framework, we are also able to learn generalized exemplars by replacing similar parts in the sentences. We call these two distinct learning heuristics as the *similarity template learning* and the *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 language sentences is different from the number of differing or similar components between the target language sentences. We refer this technique as GEBMT for *Generalized Exemplar Based Machine Translation*.

The translation template learning framework presented in this paper is based on a heuristic to infer the correspondences between the patterns in the source and target languages from 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. Further, 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. Consider the following translation pairs given in English and Turkish to illustrate the heuristic:

I will drink orange juice
  ↔ portakal suyu içeceğim
I will drink coffee ↔ kahve içeceğim

Similarities between the translation examples are shown as underlined. The remaining parts are the differences between the sentences. According to our first heuristic, the similarities in English sentences are represented as the template "I will drink X^E", and the corresponding similarities in Turkish sentences as the template "X^T içeceğim", and these similarities should correspond each other. Here, X^E denotes a component that can be replaced by *any* appropriate structure in English and X^T refers to its translation in Turkish. This notation represents an *abstraction* of the differences "orange juice" vs. "coffee" in English and "portakal suyu" vs. "kahve" in Turkish. Continuing even further, we infer that "orange juice" should correspond to "portakal suyu" and "coffee" should correspond to "kahve"; hence learning further correspondences between the examples. According to our second heuristic, two differences in English are

represented as the templates "X^E`orange juice`" and "X^E`coffee`", and the corresponding differences in Turkish as the templates "`portakal suyu` X^T" and "`kahve` X^T". The first template in English should correspond to the first template in Turkish and the second one in English should correspond to the second one in Turkish. In addition, "I will drink" in English should correspond to "içeceğim" in Turkish.

Our learning algorithm based on this heuristic is called TTL (for *Translation Template Learner*). Given a corpus of translation pairs, TTL infers the 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 paper we will refer these languages as L^1 and L^2. Although the examples and experiments herein are on English and Turkish, we believe the model is equally applicable to many other language pairs.

The rest of the paper is organized as follows. Section 2 explains the representation in the form of translation templates. The TTL algorithm is described in Section 3, and some of its performance results are given in Section 4. Section 5 illustrates the TTL algorithm on some example translation pairs. Section 6 describes how these translation templates can be used in translation, and the general system architecture. Our system is evaluated in Section 7. The limitations of the learning heuristics are described in Section 8. Section 9 concludes the paper 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. For example, the following translation templates can be learned from the example translations given above using our first learning heuristic.

`I will drink` $X^1 \leftrightarrow X^2$ `içeceğim`
  if $X^1 \leftrightarrow X^2$
`orange juice` ↔ `portakal suyu`
`coffee` ↔ `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, for example, if it has already been acquired that "tea" in L^1 and "çay" in L^2 are translations of each other, i.e., "`tea`" ↔ "`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 can be found in a regular text. That is, there is no need for grammatical information or preprocessing on the bilingual parallel corpus. Therefore, it is a grammarless 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 the 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 in its lexical level representation,[1] that is, its stem and its morphemes. For example, the translation pair "They are running" ↔ "koşuyorlar" will be represented as "`they are run+PROG`" ↔ "`koş+PROG+3PL`". Similarly, the pair "they are walking" ↔ "yürüyorlar" will be represented as "`they are walk+PROG`" ↔ "`yürü+PROG+3PL`". Here, the `+` symbol is used to mark the beginning of a morpheme. In English sentences, `PROG` morpheme indicates progressive tense suffix (`ing` suffix), In Turkish sentences, `PROG` morpheme also indicates progressive tense suffix, and `3PL` indicates third person plural agreement marker. In this case, the sentence is treated as a sequence of morphemes (root words and morphemes) and a morpheme is the smallest unit. According to this representation, these two translation pairs would be given as

`they are run+PROG` ↔ `koş+PROG+3PL`
`they are walk+PROG` ↔ `yürü+PROG+3PL`

Using our first heuristic, the following translation templates can be learned from these two translation pairs.

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, as in Turkish, and also different realizations of the same verb according to tense, as 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 of 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 there are similar constituents then a *match sequence* $M_{a,b}$ in the following form is generated.

$$S_0^1, D_0^1, S_1^1, \ldots, D_{n-1}^1, S_n^1,$$
$$\leftrightarrow S_0^2, D_0^2, S_1^2, \ldots, D_{m-1}^2, S_m^2 \quad \text{for} \quad 1 \leq n, m$$

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_a^1 and E_b^1, where $D_{k,a}^1$ and $D_{k,b}^1$ are non-empty differing items between two similar constituents S_k^1 and S_{k+1}^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 previously formed difference D_k for $k < i$. Any of S_0^1, S_n^1, S_0^2 or S_m^2 can be empty, however, S_i^1 for $0 < i < n$ and S_j^2 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 following translation examples are given: "I bought the book for Cathy" ↔ "Cathy için kitabı satın aldım" and "I bought the ring for Cathy" ↔ "Cathy için yüzüğü satın aldım". The lexical level representations of these example pairs are:

I buy+PAST the book for Cathy ↔
  Cathy için kitap+ACC satın al+PAST+1SG
I buy+PAST the ring for Cathy ↔
  Cathy için yüzük+ACC satın al+PAST+1SG

For these translation examples, the following match sequence is obtained by our matching algorithm.

```
I buy+PAST the (book,ring) for Cathy
  ↔ Cathy için (kitap,yüzük)
  + ACC satın al + PAST + 1SG          (1)
```

That is,

S_0^1 = I buy+PAST the, D_0^1 = (book,ring),
S_1^1 = for Cathy,
S_0^2 = Cathy için, D_0^2 = (kitap,yüzük),
S_1^2 = + 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 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 as *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 as *difference translation templates* since they contain differences in the match sequence. Both similarity and difference translation templates are the templates with 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 their specificities. Given two templates, the one that has a higher number of terminals is more specific than the other. Note that, the specificity is defined 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 the 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 in the following form.

$$S_0^1, D_0^1, S_1^1 \leftrightarrow S_0^2, D_0^2, S_1^2$$

Since D_0^1 and D_0^2 are the corresponding differences, the following similarity translation template is inferred by replacing these differences with variables.

$$S_0^1\ X^1\ S_1^1 \leftrightarrow S_0^2\ X^2\ S_1^2$$
$$\textbf{if } X^1 \leftrightarrow X^2$$

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

$$D_{0,a}^1 \leftrightarrow D_{0,a}^2$$
$$D_{0,b}^1 \leftrightarrow D_{0,b}^2$$

For example, since the match sequence given in (1) contains a single difference in both sides, the following similarity translation template and two additional atomic translation templates from the corresponding differences (`book,ring`) and (`kitap,yüzük`) can be inferred:

```
I buy+PAST the X1 for Cathy ↔ Cathy
  için X2+ACC satın al+PAST+1SG
    if X1 ↔ X2
book ↔ kitap
ring ↔ yüzük
```

On the other hand, if the number of differences are equal on both sides, but more 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. Therefore, learning depends on previously acquired translation templates. Our similarity template learning algorithm tries to locate $n-1$ corresponding differences in the match sequence by checking previously learned translation templates. We say that the kth difference $(D_{k,a}^1, D_{k,b}^1)$ on the left side corresponds to the lth difference $(D_{l,a}^2, D_{l,b}^2)$ on the right side if the following two translation templates have been learned earlier:

$$D_{k,a}^1 \leftrightarrow D_{l,a}^2$$
$$D_{k,b}^1 \leftrightarrow D_{l,b}^2$$

After finding $n-1$ corresponding differences, two unchecked differences, one at each side, should correspond to each other. Thus, for all differences in the match sequence, we determine which difference in one side corresponds to which difference on the other side. 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 $(D_{k_i}^1, D_{l_i}^2)$. For each $CDPair_i$, we replace $D_{k_i}^1$ with a variable X_i^1, and $D_{l_i}^2$ with a variable X_i^2 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 following similarity translation template can be inferred.

$$M_{a,b}WDV$$
$$\textbf{if } X_1^1 \leftrightarrow X_1^2 \textbf{ and } \cdots \textbf{ and } X_n^1 \leftrightarrow X_n^2$$

In addition, the following atomic translation templates are learned from the last corresponding differences $(D_{k_n,a}^1, D_{k_n,b}^1)$ and $(D_{l_n,a}^2, D_{l_n,b}^2)$.

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

For example, the following translation examples have two differences on both sides:

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

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

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 1. The similarity TTL (STTL) algorithm.

The following match sequence is obtained for these examples.

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

Without prior information, we cannot determine if "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 following similarity translation template and two additional atomic translation templates can be inferred.

X_1^1 break+PAST the $X_2^1 \leftrightarrow X_2^2$+ACC kır
+PAST X_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 Fig. 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 the 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:

- S_0^i, D_0^i, S_1^i where S_0^i is non-empty, and S_1^i is empty.
- S_0^i, D_0^i, S_1^i where S_1^i is non-empty, and S_0^i is empty.
- $S_0^i, D_0^i, S_1^i, D_1^i, S_2^i$ 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 get two match sequences with

similarity variables, namely $M_a WSV$ and $M_b WSV$ as follows.

$$M_a^1 WSV \leftrightarrow M_a^2 WSV$$
$$M_b^1 WSV \leftrightarrow M_b^2 WSV$$

For example, $M_a^1 WSV$ and $M_b^1 WSV$ will be as follows for the third case given above.

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

As a result, the following two difference translation templates are learned when there is a single non-empty similarity in the both sides of a match sequence.

$$M_a WSV$$
$$\quad \textbf{if } X^1 \leftrightarrow X^2$$
$$M_b WSV$$
$$\quad \textbf{if } X^1 \leftrightarrow X^2$$

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

$$S_k^1 \leftrightarrow S_l^2$$

For example, the match sequence in (2) contains a single non-empty similarity in both sides. The following two difference translation templates, and one additional atomic template from the corresponding similarities `"break+PAST the"` and `"+ACC kır+PAST"` are learned from this match sequence.

`i` X^1 `window ↔ pencere` X^2 `+1SG`
    if $X^1 \leftrightarrow X^2$
`you` X^1 `door ↔ kapı` X^2 `+2SG`
    if $X^1 \leftrightarrow X^2$
`break+PAST the ↔ +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 following translation template has been learned earlier:

$$S_k^1 \leftrightarrow S_l^2$$

After the finding $n - 1$ corresponding similarities, there will be two unchecked similarities, one at 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 $M_a WSV$ and $M_b WSV$ by separating difference pairs in the match sequence. As a result, the following two difference translation templates can be inferred.

$$M_a WSV$$
$$\quad \textbf{if } X_1^1 \leftrightarrow X_1^2 \textbf{ and} \cdots \textbf{and } X_n^1 \leftrightarrow X_n^2$$
$$M_b WSV$$
$$\quad \textbf{if } X_1^1 \leftrightarrow X_1^2 \textbf{ and} \cdots \textbf{and } X_n^1 \leftrightarrow X_n^2$$

In addition, the following atomic translation template is learned from the last corresponding similarities.

$$S_{k_n}^1 \leftrightarrow S_{l_n}^2$$

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

$$\left(S_0^1, S_1^2\right), \left(S_1^1, S_0^2\right),$$

the following difference translation templates can be inferred.

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

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

$$S_1^1 \leftrightarrow S_0^2$$

For example, the match sequence in (1) 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 following two difference translation template and one additional translation template can be inferred.

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 in both sides of a match sequence is greater than or equal to 1, e.i. $1 \leq n = m$, 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 Fig. 2.

3.3. Different Number of Similarities or Differences in Match Sequences

The STTL algorithm given in Section 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 number of differences or similarities, respectively. We try to make the number of differences to be 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 following two translation examples (`"I came"` $\leftrightarrow$ `"geldim"` and `"You went"` $\leftrightarrow$ `"gittin"`) has one difference on the left side, but it has two differences on the right side:

`i come+PAST` $\leftrightarrow$ `gel+PAST+1SG`
`you go+PAST` $\leftrightarrow$ `git+PAST+2SG`

Match Sequence:
`(i come,you go) +PAST`
  $\leftrightarrow$`(gel,git) +PAST (+1SG,+2SG)`

---

procedure DifferenceTTL($M_{a,b}$)
begin
  if $numofsim(M_{a,b}^1) = numofsim(M_{a,b}^2) = n \geq 1$
    and $n-1$ corresponding similarities can be found in $M_{a,b}$ **then**
    - Assume that the unchecked corresponding similarities is $(S_{k_n}^1, S_{l_n}^2)$.
    - Assume that the list of corresponding similarities is $(S_{k_1}^1, S_{l_1}^2) \cdots (S_{k_n}^1, S_{l_n}^2)$ including unchecked ones.
    - For each corresponding difference $(S_{k_i}^1, S_{l_i}^2)$ replace $S_{k_i}^1$ with X_i^1 and $S_{k_i}^2$ with X_i^2 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 $X_n^1 \leftrightarrow X_n^2$
      $S_{k_n}^1 \leftrightarrow S_{l_n}^2$
end

Figure 2. The difference TTL (DTTL) algorithm.

The STTL algorithm given in Section 3.1 cannot learn translation templates from this match sequence because the number of differences are 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 from morpheme boundaries. As a result, we get the following match sequence

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

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

X_1^1 X_2^1 +PAST ↔ X_2^2 +PAST X_1^2
    if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
come ↔ gel
go ↔ 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 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 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 following translation examples (`"I drank water"` ↔ `"su içtim"` and `"You ate orange"` ↔ `"portakal yedin"`) has two differences on both sides:

```
i drink+PAST water ↔ su iç+PAST+1SG
you eat+PAST orange
    ↔ portakal ye+PAST+2SG
```

Match Sequence:

```
(i drink,you eat)+PAST (water,orange)
  ↔ (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 correspondings should hold because they will yield incorrect translation templates. But, if we divide the differences on both sides, we get the following match sequence with three differences on both sides.

```
(i,you) (drink,eat)+PAST(water,orange)
 ↔(su,portakal)(iç,ye)+PAST (+1SG,+2SG)
```

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

X_1^1 X_2^1 +PAST X_3^1 ↔ 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 ↔ iç
eat ↔ 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 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 there is no other way to create an instance.

For example, from the match sequence of the following translation examples (`"I came"` ↔ `"geldim"` and `"I went"` ↔ `"gittim"`), the DTTL algorithm cannot learn new templates because it contains two similarities on the left side and one on the right side:

```
i come+PAST ↔ gel+PAST+1SG
i go+PAST   ↔ git+PAST+1SG
```

Match Sequence:

```
i (come,go)+PAST ↔ (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 following translation templates can be learned by the DTTL algorithm.

X_1^1 come X_2^1 ↔ 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 ↔ git X_2^2 X_1^2
    if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
+PAST ↔ +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" ↔ "adamı gördüm" and "I saw a man" ↔ "bir adam gördüm" because "bir" and "+ACC" have to match empty strings:

i see+PAST the man ↔ adam+ACC gör+PAST+1SG
i see+PAST a man ↔ bir adam gör+PAST+1SG

However, if we relax this restriction in the matching algorithm by letting a difference to have an empty constituent, this new version of the matching algorithm will find the following match sequence for the example above.

i see+PAST (the:a) man
  ↔(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 following translation templates can be learned:

X_1^1 the X_2^1 ↔ 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 ↔ 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 ↔ 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 whose one side is 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 sequences since it does not cause the problem mentioned above. We only try to get a match sequence with a difference having an empty constituent, if only the original match algorithm cannot find a match sequence without differences having an empty constituent.

3.5. Complete Learning Examples

In this section, we describe the behavior of our learning algorithms by giving the details of algorithm steps on two translation example pairs. We assume that the following two translation templates have been learned earlier.

i ↔ +1SG
you ↔ +2SG

The first translation example pair is:

I drank wine ↔ Şarap içtim
You drank beer ↔ Bira içtin

Since our learning algorithms actually work on the lexical form of sentences, the input for our algorithm will be the following two translation examples in the lexical form.

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

Then, we will try to find a match sequence between these two translation examples. To do that, a match sequence between English sentences "i drink+PAST wine" and "you drink+PAST beer," and a match sequence between Turkish sentences "şarap iç+PAST+1SG" and "bira iç+PAST+2SG" are found. As a result, the following match sequence is obtained between these two translation examples.

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

Then, we try to apply STTL and DTTL algorithms to this match sequence.

Since there are equal number of differences (two differences) on both sides, the STTL algorithm is applicable to this match sequence. But the STTL algorithm can learn new translation templates from this match sequence, if it can determine the corresponding differences. Since the corresponding between (i,you) and (+1SG,+2SG) has been given, (wine,beer) should correspond to (şarap,bira). Thus, the STTL algorithm infers the following three translation templates from this match sequence.

X_1^1 drink+PAST X_2^1 ↔ 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 are equal number of similarities (one similarity) 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 following three translation templates from this match sequence.

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

The second translation example pair is:

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 following two translation examples in the lexical form.

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 kırmızışarapiç+PAST+2SG

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

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

Then, we try to apply STTL and DTTL algorithms to this match sequence.

Since there are equal number of differences (two differences) on both sides, the STTL algorithm is applicable to this match sequence. But the STTL algorithm can learn new translation templates from this match sequence, if it can determine the corresponding differences. Since the corresponding between (i,you) and (+1SG,+2SG) has been given, (white,red) should correspond to (beyaz,kırmızı). Thus, the STTL algorithm infers the following three translation templates from this match sequence.

X_1^1 drink+PAST a glass of X_2^1 wine
  ↔ 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 ↔ beyaz
red ↔ kırmızı

Since there are equal number of similarities (two similarities) on both sides, the DTTL algorithm is also applicable to this match sequence. But we cannot determine similarity correspondings in this match sequence. In other words, we cannot know whether "drink+PAST a glass of" corresponds to "bir bardak" or "şarap iç+PAST". So, 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 hold equal number of similarities at both sides, and similarity correspondings can be determined in that instance. One of instances of this match sequence can be obtained by separating the similarity "drink+PAST a glass of" into similarities "drink+PAST" and "a glass of", and by separating the similarity "şarap iç+PAST" into similarities "şarap" and "iç+PAST". So, we will have 3 similarities on both sides in this instance of the original match sequence. From the first example, we have 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 following three translation templates can be inferred from this instance by the DTTL algorithm.

```
i X1_1 X1_2 red X1_3 ↔ X2_2 kırmızı X2_3 X2_1+1SG
  if X1_1 ↔ X2_1 and X1_2 ↔ X2_2 and X1_3 ↔ X2_3
you X1_1 X1_2 white X1_3 ↔ X2_2 beyaz X2_3 X2_1+2SG
  if X1_1 ↔ X2_1 and X1_2 ↔ X2_2and X1_3 ↔ X2_3
a glass of ↔ bir bardak
```

In this example, we looked at the instances of the original match sequence because we could not learn translation templates from the original match sequence. In this kind of situation, we continue to generate instances of the original match sequence until the first instance from which we can learn translation templates or until there is no more instance 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.

4. Performance Results

In order to evaluate TTL algorithms empirically, we have implemented them in PROLOG and evaluated on medium sized bilingual parallel texts. Our training sets are artificially collected because of the unavaliabilty of a large morphologically tagged bilingual 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 stayed in $O(n^2)$.

One of our training sets contained 747 training pairs which is enough to teach a small coverage of the basic English grammar. To find the cost of each portion of our learning algorithm and our gain from that portion, we applied the different portions of algorithms on this training set. As a result, we got the following measurements on a SPARC 20/61 workstation:

1. 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.
2. 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.
3. 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.
4. 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 percent more new templates were learned costing 25 percent more on the learning time.
5. 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 percent more new templates were learned costing 68 percent more on the learning time.

5. Examples

In this section, we will illustrate the behavior of TTL algorithms further on some sample training sets.

Example 1. Given the example translations "I came" ↔ "geldim", "You came" ↔ "geldin", "I went" ↔ "gittim" and "You went" ↔ "gittin" their lexical level representations are

```
i come+PAST ↔ gel+PAST+1SG
you come+PAST ↔ gel+PAST+2SG
i go+PAST ↔ git+PAST+1SG
you go+PAST ↔ git+PAST+2SG .
```

From the first and second examples, the STTL algorithm learns the first three templates and the DTTL algorithm the next three templates.

X^1 come+PAST ↔ gel+PAST X^2 **if** $X^1 \leftrightarrow X^2$
i ↔ +1SG
you ↔ +2SG
i X^1 ↔ X^2 +1SG **if** $X^1 \leftrightarrow X^2$
you X^1 ↔ X^2 +2SG **if** $X^1 \leftrightarrow X^2$
come+PAST ↔ gel+PAST

From the first and third examples, the STTL algorithm learns the following first three templates and the DTTL algorithm learns the next three templates by separating the similarity "+PAST+1SG" into similarities "+PAST" and "+1SG", and using already learned correspondence "i" ↔ "+1SG".

i X^1 +PAST ↔ X^2 +PAST+1SG **if** $X^1 \leftrightarrow X^2$
come ↔ gel
go ↔ git
X_1^1 come X_2^1 ↔ 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 ↔ git X_2^2 X_1^2
  if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
+PAST ↔ +PAST

From the first and fourth examples, the STTL algorithm learns the first one of the following three new templates by separating the difference (i come,you go) into the differences (i,you) and (come,go) and using already learned correspondences "i" ↔ "+1SG" and "you" ↔ "+2SG". The DTTL algorithm learns the next two templates.

X_1^1 X_2^1 +PAST ↔ X_2^2 +PAST X_1^2
  if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
i come X^1 ↔ gel X^2 +1SG **if** $X^1 \leftrightarrow X^2$
you go X^1 ↔ git X^2 +2SG **if** $X^1 \leftrightarrow X^2$

From the second and third examples, the STTL algorithm does not learn any new template and the DTTL algorithm learns the following two new templates.

you come X^1 ↔ gel X^2 +2SG **if** $X^1 \leftrightarrow X^2$
i go X^1 ↔ git X^2 +1SG **if** $X^1 \leftrightarrow X^2$

From the second and fourth examples, the STTL algorithm learns the following new template and the DTTL algorithm does not learn any new template.

you X^1 +PAST ↔ X^2 +PAST +2SG **if** $X^1 \leftrightarrow X^2$

From the third and fourth examples, the STTL algorithm learns the first one of the following two new templates and the DTTL algorithm learns the next one.

X^1 go +PAST ↔ git +PAST X^2 **if** $X^1 \leftrightarrow X^2$
go +PAST ↔ git +PAST

So, from these four simple translation examples 20 new translation templates are learned from our TTL algorithms. Some of the templates are learned more than once.

Example 2. Given the example translations "red apple" ↔ "kırmızı elma", "green apple" ↔ "yeşil elma", "We ate a pear" ↔ "Bir armut yedik", "We ate a banana" ↔ "Bir muz yedik", "They ate a pear" ↔ "Bir armut yediler", "They ate a banana" ↔ "Bir muz yediler", their lexical level representations are

```
red apple ↔ kırmızı elma
green apple ↔ yeşil elma
we eat+PAST a pear
  ↔ bir armut ye+PAST+1PL
we eat+PAST a banana
  ↔ bir muz ye+PAST+1PL
they eat+PAST a pear
  ↔ bir armut ye+PAST+3PL
they eat+PAST a banana
  ↔ bir muz ye+PAST+3PL.
```

From the first and second examples, the STTL algorithm learns the following first three templates and the DTTL algorithm learns the next three templates.

X^1 apple ↔ X^2 elma **if** $X^1 \leftrightarrow X^2$
red ↔ kırmızı
green ↔ yeşil
red X^1 ↔ kırmızı X^2 **if** $X^1 \leftrightarrow X^2$
green X^1 ↔ yeşil X^2 **if** $X^1 \leftrightarrow X^2$
apple ↔ elma

From the third and fourth examples, the STTL algorithm learns the following three templates.

we eat+PAST a
  X^1 ↔ bir X^2 ye+PAST+1PL **if** $X^1 \leftrightarrow X^2$
pear ↔ armut
banana ↔ muz

From the third and fifth examples, the STTL algorithm learns the following first three templates and the DTTL algorithm learns the next three templates.

X^1 eat+PAST a pear
  ↔ bir armut ye+PAST X^2 **if** $X^1 \leftrightarrow X^2$
we ↔ +1PL
they ↔ +3PL
we X^1 ↔ X^2 +1PL **if** $X^1 \leftrightarrow X^2$
they X^1 ↔ X^2 +3PL **if** $X^1 \leftrightarrow X^2$
eat+PAST a pear ↔ bir armut ye+PAST

From the third and sixth examples, the STTL algorithm learns the following new template.

X_1^1 eat+PAST a X_2^1 ↔ bir X_2^2 ye+PAST X_1^2
  if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^2 \leftrightarrow X_2^2$

From the fourth and sixth examples, the STTL algorithm learns the following first new template and the DTTL algorithm learns the next new template.

X^1 eat+PAST a banana ↔ bir muz ye
  +PAST X^2 **if** $X^1 \leftrightarrow X^2$
eat+PAST a banana ↔ bir muz ye+PAST

From the fifth and sixth examples, the STTL algorithm learns the following new template.

X^1 eat+PAST a banana ↔ bir muz ye
  +PAST X^2 **if** $X^1 \leftrightarrow X^2$

Example 3. Given the example translations "He always washes his face" ↔ "Her zaman yüzünü yıkar", "I watched tv" ↔ "Televizyon seyrettim", "He always washes his face after he wakes up" ↔ "Kalktıktan sonra her zaman yüzünü yıkar", "I watched tv after I ate the dinner" ↔ "Akşam yemeğini yedikten sonra televizyon seyrettim", their lexical level representations are

he always wash+3SG his face
  ↔ her zaman yüz+P3SG+ACC yıka+AOR
i watch+PAST tv ↔ televizyon seyret
  +PAST+1SG
he always wash+3SG his face after he wake
  +3SG up ↔ kalk+ConvNoun=DHk+ABL sonra
  her zaman yüz+P3SG+ACC yıka+AOR
i watch+PAST tv after i eat+PAST the
    dinner ↔ akşam yemek+P3SG+ACC
    ye+ConvNoun=DHk+ABL sonra televizyon
    seyret+PAST+1SG.

From the third and fourth examples, the STTL algorithm learns the following first three templates with the help of the first two examples pairs. The DTTL algorithm learns the next three templates.

X_1^1 after X_2^1 ↔ X_2^2 +ConvNoun=DHk+ABL
  sonra X_1^2
    if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^2 \leftrightarrow X_2^2$
he wake+3SG up ↔ kalk
i eat+PAST the dinner ↔ akşam
  yemek+P3SG+ACC ye
he always wash+3SG his face X^1
  he wake+3SG up ↔ kalk X^2 her zaman
  yüz+P3SG+ACC yıka+AOR **if** $X^1 \leftrightarrow X^2$
i watch+PAST tv X^1 i eat+PAST the
  dinner ↔ akşam yemek+P3SG+ACC ye X^2
  televizyon seyret+PAST+1SG
  if $X^1 \leftrightarrow X^2$
after ↔ ConvNoun=DHk+ABL sonra

6. System Architecture and Translation

The templates learned by the TTL algorithm can be directly used in the translation. These templates are in lexical form, and they can be used for translation in both directions. The general system architecture is given in Fig. 3.

As it is seen in Fig. 3, the input for the learning module is a set of bilingual examples in lexical form. For this purpose, our sets of bilingual examples have been

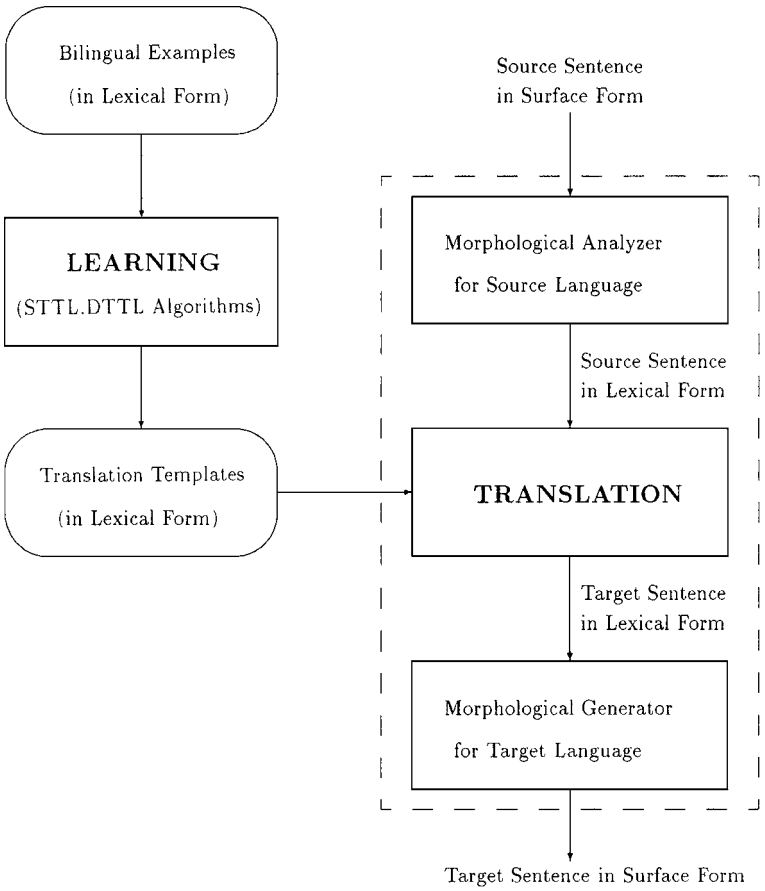

Figure 3. The system architecture.

prepared in lexical form between English and Turkish. To create a set of bilingual examples in lexical form, a set of bilingual examples in surface form is created, then 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 there were morphological tagged sets of bilingual examples between English and Turkish, there wouldn't be any need for this step. Some of sets in surface form are constructed by us, and some of them were prepared by other people. For example, we used the manuals for small house hold items, which contain instructions both in English and Turkish, as a set of bilingual examples in surface form, and then we morphologically tagged the sentences in those manuals. As a result, we got some of our data from other sources, and some of them are collected by us.

From the surface form of a sentence, the lexical form of that sentence is created by replacing every word in that sentence with its correct lexical form. Non-words such as punctuation markers in the surface form, are treated as a single root word (i.e. a punctuation marker appears in the lexical form of the sentences same as in the surface form of the sentence). In other words, a punctuation marker is treated as a single word in the examples. The only exception for this, the punctuation markers marking the end of sentences, they are completely eliminated 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, the translation templates matching the input are collected. These templates are those that are most similar to the sentence to be translated. They are collected in the specificity order. For each selected template, its variables are instantiated with the corresponding values in the source sentence. Then, templates matching these bound values are sought. If they are found successfully, their values are replaced in the variables corresponding to the sentence in the target language.
3. Finally, the surface level representation of the sentence obtained in the previous step is generated by the target language morphological generator.

For instance, after learning the templates in Example 1 and 2, if the input is given as "bir kırmızı elma yedim", first its lexical level representation, which is `"bir kırmızı elma ye+PAST+1SG"`, is derived. Since the following template is the only matching template for this input, that template is used in the translation process.

$$X_1^1 \texttt{ eat+PAST a } X_2^1 \leftrightarrow \texttt{ bir } X_2^2 \texttt{ ye+PAST } X_1^2$$
$$\textbf{if } X_1^1 \leftrightarrow X_1^2 \textbf{ and } X_2^2 \leftrightarrow X_2^2$$

The variable X_1^2 is instantiated with `"+1SG"`, and the variable X_2^2 is instantiated with `"kırmızı elma"`. Then, the translation of `"+1SG"` is found to be `"i"` using

$$\texttt{i} \leftrightarrow \texttt{+1SG}$$

and, the translation of `"kırmızı elma"` is found to be `"red apple"` using

$$\texttt{red apple} \leftrightarrow \texttt{kırmızı elma.}$$

Therefore, replacing `"i"` for X_1^1, and `"red apple"` for X_2^1 in the template, the lexical level representation `"i eat+PAST a red apple"` is obtained. Finally, the surface level representation "I ate a red apple" is derived easily by English 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. Among the possible translations, a human user can choose the right one according to the context. We hope that the correct answer will be among first results generated in the translation steps by using the specificity order of the templates. Although the specificity order of the templates helps to get correct answer among the top results, it may not be enough. We also looked into the ways to use a statistical method [26] to order our learned translation templates. In this statistical method, we assign 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 50 percent in the top 5 results.

7. Evaluation

Since the TTL algorithm can over-specialize, useless and incorrect templates can be learned. Because of this problem and the ambiguity problem, the translation results produced by our translation algorithm can contain incorrect translations in addition to correct ones. But our main goal is to accomplish that top results contain correct translations. For example, according to our results given in [26] for an experiment reflects that the percentage of correct results in total results is 33 percent. If we just use the specificity order of templates, the percentage of correct results are increased to 44 percent in top 5 results. This means that at least 2 of top 5 results are correct. In addition to the specificity order, using the statistical method described in [26] increased the percentage of the correct results to 60 percent. In addition, we look at whether the top results contain at least one correct translation or not. When we just use the specifty order, the top 5 results of 77 percent of all translations contained at least one correct translation. When the statistical method is used together with the specifty order, the percentage is increased to 91 percent. Thus, a human expert can choose the correct answer by just looking to the top 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 as bilingual corpora about United Nations documents, they are big enough to be treated as real corpora. As a future work, we are planning to test our algorithms on morphologically tagged huge bilingual corpora between other languages (unfortunately there is no huge bilingual corpora 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. In fact, we applied our algorithms to small training sets between English and French, and we got similar results.

The success of a machine translation system can be measured according to two criteria: coverage and correctness. The coverage is the percentage of the sentences which can be translated, and the correctness is the percentage of correct translations among all translation results produced by that system. However, for any machine translation system, it cannot be said that it guarantees correctness and completeness. There is no machine translation system that will always produce the correct translation for any given sentence, or it can produce a 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, the words and sentences would be 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 paper 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 example-based machine translation system. Our system suggests possible translations (top translation results, and possibly these results contain the correct translation) for a sentence, and a human expert chooses the correct translation just looking at the results given. 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 the most of the available information as translation templates. When someone measures the correctness of our system, he should look at whether top results contain the correct translation or not. To increase the correctness, we used the specifty order on the translation templates, and assigned confidence factors to them. This will help the correct translation to be among the top results. The general performance of our system and other example-based machine translation systems depend on the quality of bilingual corpora used in them because they the source of the information, and how the available information in the corpora is used in the translation process.

8. Limitations of Learning Heuristics

The preconditions in the definition of the match sequence may look like 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.

Let us consider the following translation examples between American and British English:

```
1.The other day, the president
    analyzed the state of the union ↔
  The other day, the president
    analysed the state of the union
2.Recently, the president analyzed
    the state of the union ↔ Recently,
    the president analysed the state of
    the union
3.Recently, the president analyzed
    the union ↔ Recently, the president
    analysed the union
```

Although these three examples have very similar structures, our learning heuristics will not learn any translation templates from these examples. The reason for this is that the lexical item `"the"` will end up in both a similarity and a difference in a match sequence of any two of these examples. Since this is not allowed, any pair of these examples cannot have a match sequence. As a result, our system will not be able to translate the following sentence to British English when these examples are given.

```
The other day, the president analyzed
  the union                          (a)
```

So, 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 as follows:

```
4.He analyzed today's situation ↔
    He analysed today's situation
5.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 1–5. Some of

the learned templates will be as follows:

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

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

We could have relaxed the conditions for the definition of the match sequence. If we let a lexical item appear in a similarity and a difference of a match sequence, we will not have a unique match sequence for any two strings and there will be more than one match sequence for those strings. For instance, examples 2 and 3 above will have 25 different match sequences because there will 5 match sequences for each side of those sentences in this situation. Since only one of these match sequences will be the correct one, we may learn a lot of useless (wrong) templates from the rest of these match sequences. This is the main reason for insisting on the strong preconditions on the match sequences.

Our learning algorithms may still learn useless wrong translation templates. For example, let us consider the following two examples.

- 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. The reason is that Turkish differentiates between the two meanings "know" ("tanımak" and "bilmek" in Turkish), and hardly and almost map to the same Turkish phrase ("hemen hemen"). 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 some statistical methods in our system [26].

9. Conclusion

In this paper, 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 example-based translation model into Generalized Exemplar-Based Machine Translation. 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 western languages such as English, French, Spanish). Of course, we assume that we have sets of morphologically tagged bilingual examples for these two natural languages. We test this claim by applying our algorithms to morphologically tagged training sets between English and French, the results were similar to the results between English and Turkish. We only deal with the translation on written text, we do not deal with translation in spoken languages.

The major contribution of this paper is that the proposed TTL algorithm eliminates the need for manually encoding the translation templates, which is a difficult task. The TTL algorithm can work directly on surface level representation of sentences. However, in order to generate useful translation patterns, it is helpful to use the lexical level representations. It is usually trivial, at least for English and Turkish, 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 similarly, using analogy within a corpus of example translations.

The technique presented here can be used in an incremental manner. Initially a set of translation templates can be inferred from a set of translation examples, then extra templates can be learned from another set with the help of the 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 becomes available, instead of rerunning 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 real large training corpora. Note that this algorithm is not specific to English and Turkish languages, but should be applicable to the task of learning machine translation between many pairs of languages. Although the learning process on a large corpus will take a considerable amount of time, it is only a one time job. After learning the translation templates, the translation process is fast.

The model that we have proposed in this paper 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 to 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 in our model provides a level of information that may help in error diagnosis and student modeling tasks of an ITS. The model may also be used in tuning the teaching strategy according to the needs of the student by analyzing 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 student's acquisition of the target language. The work presented in this paper provides an opportunity to evaluate this possibility as a future work.

Acknowledgment

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.

Note

1. In the lexical level representation of Turkish words appearing in the examples, we used following notations which are similar to the notations used in phrase structure grammar papers [24, 25]: 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 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 s suffix) in the verbs. Surface level realizations of these morphemes are determined according to vowel and consonant harmony rules. Surface level realization of PAST morpheme for English verbs is also depends on whether that verb is regular or irregular.

References

1. K. Goodman and S. Nirenburg, *KBMT-89: A Case Study in Knowledge Based Machine Translation*, Morgan Kaufmann: San Mateo, CA, 1992.
2. T. Mitumura and E. Nyberg, "The KANT system: Fast, accurate, high-quality translation in practical domains, in *Proceedings of COLING-92*, Nantes, France, 1992, pp. 1069–1073.
3. D. Lonsdale, T. Mitamura, and E. Nyberg, *Acquisition of Large Lexicons for Practical Knowledge-Based MT, Machine Translation*, vol 9, no. 3, Kluwer Academic Publishers, Dordrecht, 1994, pp. 251–283.
4. M.A. Nagao, "Framework of a mechanical translation between Japanese and English by analogy principle," in *Artificial and Human Intelligence*, edited by A. Elithorn and R Banerji, NATO Publications: North-Holland, Edinburg, 1984, pp. 173–180.
5. O. Furuse and H. Iida, "Cooperation between transfer and analysis in example-based framework," in *Proceedings of COLING-92*, Nantes, France, 1992, pp. 645–651.
6. H. Kitano, "A comprehensive and practical model of memory-based machine translation," in *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann, 1993, pp. 1276–1282.
7. S. Sato and M. Nagao, "The memory-based translation," in *Proceedings of COLING-90*, 1990.
8. S. Sato, "MBT2: A method for combining fragments of examples in example-based translation," *Artificial Intelligence*, vol. 75, Elsevier Science, MA, 1995, pp. 31–50.
9. F. Smadja, K.R. McKeown, and V. Hatzivassiloglou, *Translating Collocation for Bilingual Lexicons: A Statistical Approach, Computational Linguistics*, vol. 22, no. 1, The MIT Press: Cambridge, MA, 1996, pp. 1–38.
10. E. Sumita and H. Iida, "Experiments and prospects of example-based machine translation," in *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, 1991.
11. R.D. Brown, "Automated dictionary extraction for "knowledge-free" example-based translation," in *Proceedings of TMI'97*, 1997.
12. D. Wu and X. Xia, *Large-Scale Automatic Extraction of an English-Chinese Translation Lexicon, Machine Translation*, vol. 9, Kluwer Academic Publishers, Dordrecht, 1995, pp. 285–313.
13. C. Brona and C. Padraig, "Translating software documentation by example: An EBMT approach to machine translation," in *Proceedings of Int. ECAI Workshop: Multilinguality in the Software Industry*, 1996.
14. R.D. Brown, "Example-based machine translation in the Pangloss system," in *Proceedings of COLING-96*, 1996.
15. S. Nirenburg, S. Beale, and C. Domashnev, "A full-text experiment in example-based machine translation," in *Proceedings*

of the International Conference on New Methods in Language Processing, NeMLap, Manchester, UK, 1994, pp. 78–87.
16. H. Kaji, Y. Kida, and Y. Morimoto, "Learning translation templates from bilingual text," in *Proceedings of COLING-92*, 1992, pp. 672–678.
17. H.A. Güvenir and A. Tunç, "Corpus-based learning of generalized parse tree rules for translation," in *New Directions in Artificial Intelligence: Proceedings of the 11th Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, edited by G. McCalla LNCS 1081, Toronto, Ontario, Canada, May 1996, Springer Verlag, pp. 121–131.
18. D.L. Medin and M.M. Schaffer, "Context theory of classification learning," *Psychological Review*, vol. 85, pp. 207–238, 1978,.
19. K.J. Hammond, (Ed.), *Proceedings: Second Case-Based Reasoning Workshop*, Pensacola Beach, FL, Morgan Kaufmann, MA 1989.
20. J.L. Kolodner, (Ed.), in *Proceedings of a Workshop on Case-Based Reasoning*, Clearwater Beach, FL, Morgan Kaufmann, 1988.
21. A. Ram, "Indexing, elaboration and refinement: Incremental learning of explanatory cases, in *Case-Based Learning*, edited by Janet L. Kolodner, Kluwer Academic Publishers: Dordrecht, 1993.
22. I. Cicekli and H.A. Güvenir, "Learning translation rules from a bilingual corpus," in *Proceedings of the 2nd International Conference on New Methods in Language Processing (NeMLaP-2)*, Ankara, Turkey, September 1996, pp. 90–97.
23. H.A. Güvenir and I. Cicekli, "Learning translation templates from examples," *Information Systems*, vol. 23, no. 6, pp. 353–363, 1998.
24. G. Gazdar, E. Klein, G.K. Pullum, and I.A. Sag, *Generalized Phrase Structure Grammar*, Blackweell: Cambridge, Mass., 1985.
25. I.A. Sag and C. Pollard, "Head-driven phrase structure grammar: An informal synopsis," CSLI Technical Report CSLI-87-79, Stanford, 1987.
26. Z. Öz and I. Cicekli, *Ordering Translation Templates by Assigning Confidence Factors*, in *Proceedings of AMTA'98*, Langhorne, PA, *Lecture Notes in Computer Science 1529*, Springer Verlag, 1998, pp. 51–61.

Ilyas Cicekli is currently an assitant professor of the Department of Computer Engineering at Bilkent University, Ankara, Turkey. He recieved his Ph.D. degree in Computer and Information Science from Syracuse University in 1991. Dr. Cicekli recieved his M.S. degree in Computer Science from Syracuse University in 1985, and his B.S. degree in Computer Engineering from Middle East Technical University, Turkey in 1982. His research interests include natural language processing, machine translation, and logic programming. He is a member of ACL, ALP and AMTA.

H. Altay Güvenir is currently an associate professor of the Department of Computer Engineering at Bilkent University, Ankara, Turkey. He received his Ph.D. in 1988 from Case Western Reserve University. He received his MS and BS from Instanbul Technical University in 1981 and 1979, respectively, all in electrical engineering. He is a member of ACM and AAAI. Dr. Guvenir's research interests include machine learning, data mining, natural language processing and learning problem solving strategies. He has (co-)authored over 70 articles in conference proceedings, edited books and journals.