Learning Translation Templates with Type Constraints

Ilyas Cicekli

Department of Computer Engineering, Bilkent University
Bilkent 06800, Ankara, TURKEY

ilyas@cs.bilkent.edu.tr

Abstract

This paper presents a generalization technique that induces translation templates from given translation examples by replacing differing parts in these examples with typed variables. Since the type of each variable is also inferred during the learning process, each induced template is associated with a set of type constraints. The type constraints that are associated with a translation template restrict the usage of that translation template in certain contexts in order to avoid some of wrong translations. The types of variables are induced using the type lattices designed for both source language and target language. The proposed generalization technique has been implemented as a part of an EBMT system.

KeyWords: EBMT, Machine Learning

1 Introduction

An example-based machine translation [8] (EBMT) system uses a bilingual corpus to translate a given sentence in a source language into a target language. Some EBMT systems use a bilingual corpus to find translations of the parts of a given sentence, and combine these partial solutions to get the translation of the whole sentence. Some EBMT systems [1,2,3,4,5,6] extract translation templates from example sentences in a given bilingual corpus and use these translation templates in the translation of other sentences. The main differences between these EBMT systems are the assumptions that they made on the structure of the bilingual corpus and their generalization techniques. The EBMT translation system which uses the generalization technique described in this paper also extracts translation templates from a set of translation examples.

In the EBMT system presented in [3,4], a translation template is induced from given two translation examples by replacing differing parts in these examples by variables. A variable replacing a difference that consists of two differing parts (one from the first example, and the other one from the second example) is a generalization of those two differing parts. Later, that variable can be replaced by any string during the translation process without putting any restriction on the possible replacements. Although the learned translation template works correctly in certain environments, it can lead wrong translations in some other unrelated environments because that variable replacement cannot be appropriate in the unrelated environment. In this paper, we propose a generalization heuristic that replaces the differences with variables and it also induces the types of these variables from the differences. Since the types of variables disallow some possible replacements for the variables, the generation of wrong translation results in the unrelated contexts can be avoided.

The type of a variable which replaces a difference is found by using a type lattice for the language of the symbols appearing in the difference. Since the generalization technique described in this paper is used as a part of an EBMT system between English and Turkish, the type lattices for English and Turkish have been developed by hand and they are used in the EBMT system. The quality of the induced translation templates also depends on the quality of the type lattices.

The rest of the paper is organized as follows. The structure of translation templates without type constraints is discussed in Section 2. Section 3 introduces the structure of translation templates with type constraints. The generalization process that learns the translation templates with type constraints is presented in Section 4. We give the concluding remarks and possible future extensions in Section 5.

2 Translation Templates Without Type Constraints

A *language* is a set of strings in the alphabet of that language, and the *alphabet of a language* is a finite set of symbols. For example, a string in a natural language, such as English or Turkish, is a sequence of tokens in that natural language. Each token in a natural language can be a root word or a morpheme. In other words, the set of all root words and morphemes in a natural language will be treated as its alphabet in our discussions. We also associate each language with a finite set of variables. A *generalized string* is a string of the symbols of the alphabet of the language and the variables in the set of variables associated with the language. This means that a generalized string is a string that contains at least one variable. We will assume that each language will be associated with a different set of variables. A string without variables is called as a *ground string*.

A translation template can be an *atomic* or *general translation template*. An *atomic translation template* $T_a \leftrightarrow T_b$ between languages L_a and L_b is a pair of two nonempty strings T_a and T_b where T_a is a ground string in L_a and T_b is a ground string in L_b. An atomic translation template $T_a \leftrightarrow T_b$ means that the strings T_a and T_b correspond to each other. A given *translation example* will be an atomic translation template.

A *general translation template* between languages L_a and L_b is an if-then rule in the following form:

$$T_a \leftrightarrow T_b \ \ \textbf{if} \ \ X_1 \leftrightarrow Y_1 \ \textbf{and} \ldots \textbf{and} \ X_n \leftrightarrow Y_n$$

where $n \geq 1$, T_a is a generalized string of the language L_a, and T_b is a generalized string of the language L_b. Both T_a and T_b must contain n variables. The variables in T_a are $X_1 \ldots X_n$, and the variables in T_b are $Y_1 \ldots Y_n$. Each generalized string (T_a and T_b) in a general translation template should contain at least one token from the alphabet of the language of that string.

For example, if the alphabet of L_a is $A = \{a,b,c,d,e,f,g,h\}$ and the alphabet of L_b is $B = \{t,u,v,w,x,y,z\}$, the following are some examples of translation templates between L_a and L_b.

- $de \leftrightarrow vyz$
- $abX_1c \leftrightarrow uY_1$ **if** $X_1 \leftrightarrow Y_1$
- $aX_1X_2b \leftrightarrow Y_2vY_1$ **if** $X_1 \leftrightarrow Y_1$ **and** $X_2 \leftrightarrow Y_2$

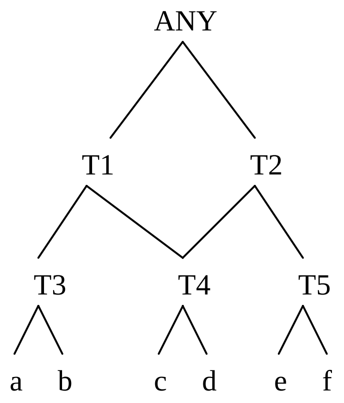
The first translation template is an atomic translation template, and last two are general translation templates. The first atomic translation template means that *de* in the language L_a and *vyz* in the language L_b correspond to each other. A general translation template is a generalization of translation examples, where certain components are generalized by replacing them with variables and establishing bindings between these variables. For example, in the second example above, the generalized string abX_1c represents all sentences of L_a starting with *ab* and ending with *c* where X_1 represents a non-empty string on A, and the generalized string uY_1 represents all sentences of L_b starting with *u* where Y_1 represents a non-empty string on B. That general template says that a sentence of L_a in the form of abX_1c corresponds to a sentence of L_b in the form of uY_1 given that X_1 corresponds to Y_1. If we know the correspondence $de \leftrightarrow vyz$, the correspondence $abdec \leftrightarrow uvyz$ can be inferred from that general template.

3 Translation Templates With Type Constraints

3.1 Type Expressions

All symbols in the alphabet of a language are organized as a *type lattice*. The symbols in the alphabet of the language appear at the bottom of the type lattice. In fact, each symbol is treated as a *ground type name* that represents itself in the type lattice. Inner nodes in the lattice are *type names* that are used for the language, and each type name represents a set of ground type names. Thus, a ground type name represents a singleton set containing that ground type name. At the top of the lattice, there is a special type name, called *ANY*. The type name *ANY* represents the set of all ground type names in the language. If *t* is a type name, we will say that GT_t is the set of the ground type names that are covered by *t*. Each node in the lattice, except *ANY*, can have one or more parents. If node *P* is a parent of node *C* in the type lattice, $GT_P \supset GT_C$ holds. Figure 1 gives a type lattice for a simple language. Since type name *T1* is the parent of type name *T3*, $GT_{T1} \supset GT_{T3}$ will be true for that type lattice.

Each variable of a generalized string in a general translation template with type constraints is associated with a type expression, and the type expression is called *the type of the variable*. The type of a variable indicates the possible ground strings which can replace that variable during the



- Ground Type Names = {a,b,c,d,e,f}
- The set of ground type names is also the alphabet of this simple language.
- The sets of ground type names represented by some type names.
  $GT_a = \{a\}$
  $GT_{T3} = \{a,b\}$
  $GT_{T1} = \{a,b,c,d\}$
  $GT_{T2} = \{c,d,e,f\}$
  $GT_{ANY} = \{a,b,c,d,e,f\}$

Figure 1. A Type Lattice for A Simple Language

translation process. A *type expression* is a non-empty sequence of atomic type expressions. An *atomic type expression* can be either T or nullor(T) where T is a type name from the type lattice. If the type of a variable is a type name T, this means that the variable can be replaced by a ground type name from GT_T. In the second case where the type of a variable is nullor(T), the variable is replaceable with an empty string in addition to a ground type name from GT_T. In other words, $GT_{nullor(T)}$ is equal to $GT_T \cup \{\varepsilon\}$.

The definition of GT can be extended for the type expressions that consist of more than one atomic type expression. If a type expression T is an atomic type sequence T1...Tn, GT_T is equal to the concatenation of the sets GT_{T1} through GT_{Tn}. In general, a variable of type T is replaceable with a ground string from GT_T. For example, let us consider the simple language and its type lattice in Figure 1. If the type of a variable is type T3, this means that it can be replaced with a ground string from $GT_{T3}=\{a,b\}$. When the type of a variable is nullor(T3), it can be replaced with an empty string or a string from GT_{T3}. A variable of the type ANY can be replaced with any ground type name. If a type expression T is an atomic type sequence "T3 T4", GT_T is equal to {ac,ad,bc,bd}.

Type lattices for English and Turkish are partially created by hand in order to be used in the developed EBMT system. Simplified partial type lattices for these languages can be seen in Figure 2. The details of those type lattices are not given in the figure. Major type names in each type lattice are the part of speech tags used for that language. The affixes used in a language are also considered as major type names. For example, the major part of speech tags such as noun, verb, pronoun and adjective are major type names in English type lattice, and they appear as children of ANY. The type names between major type names and ground type names generally represents the subgroups of part of speech tags. The affixes are grouped according to where they can be used. For example, all suffixes can be added to verbs is considered as a major type name.

3.2 Translation Templates With Type Constraints

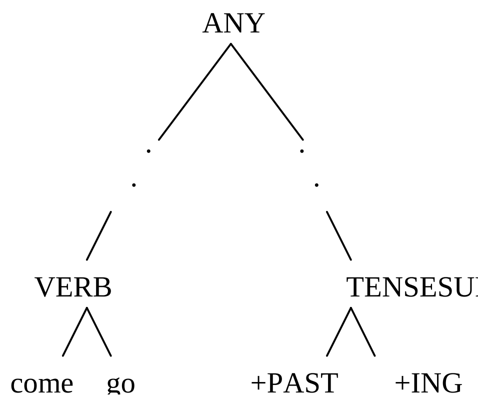
A *translation template with type constraints* is a general translation template where all variables are associated with type expressions. A translation template with type constraints will be a translation template in the following form:

$T_a \leftrightarrow T_b$
if $X_1^{TA1} \leftrightarrow Y_1^{TB1}$ **and**...**and** $X_n^{TAn} \leftrightarrow Y_n^{TBn}$

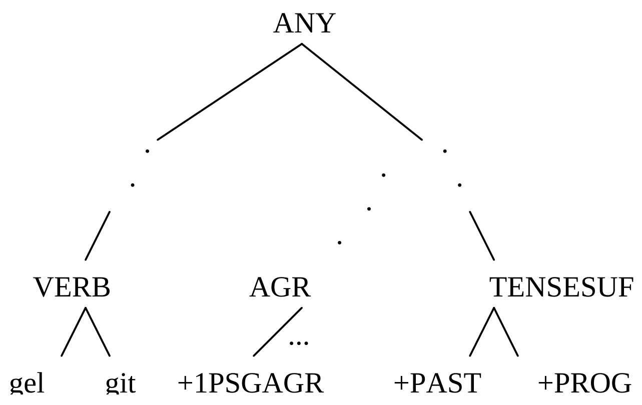
where each of TA1,...,TAn and TB1,...,TBn is a type expression. A translation template with type constraints also puts a restrictriction on the possible replacements of variables during the translation process. For example, the following is a translation template with type constraints

I X^{VERB} +PAST $\leftrightarrow$ Y^{VERB} +PAST +1PSAGR
if $X^{VERB} \leftrightarrow Y^{VERB}$

This general template represents that an English sentence in the form of "I X^{VERB} +PAST" corresponds to a Turkish sentence in the form of "Y^{VERB} +PAST +1PSAGR" given that X corresponds to Y. This template also specifies that X can only be replaced by a verb at English side, and Y can only be replaced by a verb at Turkish side. In this example, "+PAST" means the past tense suffix at English side, and "+PAST" and "+1PSAGR" at Turkish side mean that the past tense suffix and the first person singular agreement suffix, respectively. This translation template can



a) Simplified Type Lattice for English



b) Simplified Type Lattice for Turkish

Figure 2. Simplified Type Lattices for English and Turkish

be used in the translation of the following Turkish sentence

geldim
gel+PAST+1PSAGR

into the following English sentence

I came
I come+PAST

given that the correspondence "gel↔come" is available. During the translation process, both variables are replaced by English and Turkish verbs without violating type constraints in the translation template.

Type constraints in the translation templates restrict wrong usages of templates in certain circumstances. For example, if we try to use the previous translation template without type constraints, it may lead to wrong translation results. Let us assume that we want to translate the following Turkish sentence into English using this translation template without type constraints.

utangaçtım	(I was shy)
utangaç+PAST+1PSGAGR	

Without using the type restrictrictions, variable Y at Turkish side can match with "utangaç" which is an adjective (not a verb). If the correspondence "shy↔utangaç" is available, variable X at English side can match with "shy" (not a verb). Thus, it can lead to the meaningless translation result "I shy +PAST" at the lexical level. Type constraints in the translation template will avoid this wrong translation by rejecting to bind Y with "utangaç" which is an adjective.

4 Learning Translation Templates

In the EBMT system described in [3,4], translation templates are inferred without type constraints from given translation examples. Each translation example consists of an English sentence and a Turkish sentence and their lexical level representations are used for the sentences. A translation template is a generalization of two translation examples where some differing parts of the sentences are generalized by replacing them with variables, and establishing bindings between these variables.

In order to induce a translation template from given two translation examples $E^1_a \leftrightarrow E^1_b$ and $E^2_a \leftrightarrow E^2_b$, we first find the match sequence $M_a \leftrightarrow M_b$ where the match sequence M_a is a match sequence between E^1_a and E^2_a, and the match sequence M_b is a match sequence between E^1_b and E^2_b. A match sequence between two sentences is a sequence of similarities and differences between those sentences. A similarity between two sentences is a non-empty sequence of common items in both sentences. A difference between two sentences is a pair of two sequences (D_1,D_2) where D_1 is a sub-sequence of the first sentence and D_2 is a sub-sequence of the second sentence, and D_1 and D_2 do not contain any common item.

For example, let us assume that the lexical representations of the following two translation examples between English and Turkish are given.

$\underline{\text{I}}$ come $\underline{\text{+PAST}}$ ↔ gel $\underline{\text{+PAST +1PSAGR}}$
$\underline{\text{I}}$ go $\underline{\text{+PAST}}$ ↔ git $\underline{\text{+PAST +1PSAGR}}$

where common parts in the sentences are underlined. From these two examples, the following match sequence is found.

I (come,go) +PAST $\leftrightarrow$
(gel,git) +PAST +1PSAGR

where (come,go) is a difference at English side, (gel,git) is a difference at Turkish side, other parts of the match sequence are similarities.

One of the learning heuristics described in [3,4], infers a translation template by replacing differences by variables and establishing bindings between these variables. This learning heuristic can create a translation template if both sides of the match sequences contain n differences where $n \geq 1$, and the correspondences of n-1 difference pairs have been already learned. For example, for the match sequence above, this learning heuristics infers the following translation templates.

I X +PAST $\leftrightarrow$ Y +PAST +1PSAGR
if X $\leftrightarrow$ Y
come $\leftrightarrow$ gel
go $\leftrightarrow$ git

The first translation template is a general translation template created by replacing differences with variables X and Y. The last two translation templates are atomic translation templates and they are inferred from the correspondence of the differences (come,go) and (gel,git).

Variables X and Y in this translation template do not have any type constraints, and they are replaceable with any ground strings as long as they are translations of each other during translation process. As we discussed in Section 3.1, this can lead to wrong translation results in unrelated environments. In order to reduce the amount of wrong translation results, translation templates will be associated with type constraints. In the rest of this section, we describe how translation templates with type constraints are inferred from the given translation examples.

4.1 Inferring A Type Expression for Two Symbols

When we replace a difference with a variable, we should also find a type expression for that variable. If both constituents of a difference are symbols (strings with length 1), the type expression for those symbols is found using the type lattice of that language, and the found type expression will be used as a type constraint for the variable replacing that difference. For example, when we infer a translation template from the match sequence "I (come,go) +PAST $\leftrightarrow$ (gel,git) +PAST +1PSAGR", we also infer types of the variables replacing the differences (come,go) and (gel,git). Of course, we use English type lattice for the difference (come,go), and Turkish type lattice for the difference (gel,git).

If we have two symbols, they are also ground type names in the type lattice of the language of those symbols. For example, *come* and *go* are ground type names in English type lattice. Since the variable replacing the difference (come,go) represents the symbols *come* and *go*, the type of this variable should cover both of those symbols. We say that a ground type *gt* is covered by a type *t*, if $gt \in GT_t$. So, if type T covers both symbols *come* and *go*, both $come \in GT_T$ and $go \in GT_T$. At the worst case, type *ANY* will cover any given two ground type names in a language.

In general, there can be more than one type covering any given two type names. Since we do not want to over-generalize, we select the most specific type covering both of them. We say that type T_2 is more specific than type T_1, if $GT_{T1} \supset GT_{T2}$ holds. This means that T_1 is one of the ancestors of T_2. So, if both T_1 and T_2 covers given type names and T_2 is more specific than T_1, T_2 is selected as a type expression for the given type names.

In some cases, there can be two ancestors T_1 and T_2 of a given pair of type names, and the ancestors may not hold any specifity relation between them. That is, neither $GT_{T1} \supset GT_{T2}$ nor $GT_{T2} \supset GT_{T1}$ holds. So, the youngest ancestor of the two given types is selected to represent them.

In order to find a youngest ancestor of two given types, the shortest path containing one of their ancestors is found and the ancestor on that shortest path is the youngest ancestor of them. A type is also considered as an ancestor of itself. Thus, the youngest ancestor of types T_1 and T_2 will be T_1 if T_1 is an ancestor of T_2.

According to English type lattice, the youngest ancestor of *come* and *go* is type VERB, and the youngest ancestor of *gel* and *git* is type VERB according to Turkish type lattice in Figure 2. So, the following translation template with type constraints is induced from the match sequence "I (come,go) +PAST $\leftrightarrow$ (gel,git) +PAST +1PSAGR":

I X^{VERB} +PAST $\leftrightarrow$
Y^{VERB} +PAST +1PSAGR
if $X^{VERB} \leftrightarrow Y^{VERB}$

When we replace a difference (t1,t2) where t1 and t2 are two different type names in their type lattice with a type name t3 which is the youngest ancestor of t1 and t2, we generalize (t1,t2) as t3. Each generalization has a generalization score to indicate the amount of that generalization. We use the length of the shortest path between t1 and t2 as a generalization score. For example, the score for the generalization of (come,go) as VERB is 2, because the length of the shortest path between *come* and *go* is 2. In fact, when a difference is generalized, the generalization with the smallest generalization score is used. We will say that *gen(t1,t2)* is t3, and *genscore(t1,t2)* is 2.

4.2 Inferring A Type Expression for Two Strings

If a difference has a constituent whose length is greater than one, the generalization of that difference cannot be an atomic type expression. If *n* is the length of the longest constituent of a difference, its generalization will be a type expression consisting of *n* atomic type expressions. If a difference is (a1...an,b1...bn) where the lengths of the constituents are equal, the generalization gen(a1...an,b1...b4) will be

gen(a1,b1) gen(a2,b2) ... gen(an,bn).

The generalization score genscore(a1...an,b1...bn) for this generalization will be equal to

genscore(a1,b1) + genscore(a2,b2) + ...
+ genscore(an,bn).

If the lengths of constituents are different, we have to consider different possibilities and some symbols have to be generalized with empty strings. For example, we have to consider the following three generalizations for the difference (abc,de):

gen(a,d) gen(b,e) gen(c,ε)
gen(a,d) gen(b,ε) gen(c,e)
gen(a,ε) gen(b,d) gen(c,e)

When there are more than one possible generalization for a difference, we select the one with the smallest generalization score. Since we assume that we have an imaginary type for each ground type name in the type lattice such that it is a parent of that ground type name and the empty string, the score of the generalization of a symbol with the empty string is assumed to be 2. The generalization of a symbol *a* and the empty string is represented by *nullor(a)*.

Let us consider the following two translation examples.

<u>I</u> come +PAST ↔ gel +PAST <u>+1PSAGR</u>
<u>I</u> am go +ING ↔ git +PROG <u>+1PSAGR</u>

For these examples, the following match sequence is found.

I (come +PAST, am go +ING) ↔
(gel +PAST, git +PROG) +1PSAGR

In order to select the generalization for the difference (come +PAST, am go +ING), we have to consider the following three generalizations:

gen(come,am) gen(+PAST,go) gen(ε,+ING)
gen(come,am) gen(ε,go) gen(+PAST,+ING)
gen(ε,am) gen(come,go) gen(+PAST,+ING)

Since the last generalization has the smallest generalization score, it will be selected as the generalization for this difference. So, the generalization for this difference will be the following type expression:

nullor(am) VERB TENSESUF

Similarly, the difference (gel +PAST, git +PROG) has only one possible generalization:

gen(gel,git) gen(+PAST,+PROG)

Thus, the generalization for the difference (gel +PAST, git +PROG) will be the following type expression:

VERB TENSESUF

As a result, the following translation template with type constraints will be inferred from these two translation examples.

I $X^{\text{nullor(am) VERB TENSESUF}}$ ↔
$Y^{\text{VERB TENSESUF}}$ +1PSAGR
if $X^{\text{nullor(am) VERB TENSESUF}}$ ↔ $Y^{\text{VERB TENSESUF}}$

5 Conclusion

In this paper, we have presented a learning technique that induces translation templates from given translation examples, by replacing the differing parts with variables. Types of variables are also learned during the learning phase from the replaced differing parts. The types of variables help to reduce the amount of wrong translation results by restricting the usage of the translation templates in unrelated contexts.

The learning heuristic described in this paper has been implemented as a part of an EBMT system between English and Turkish. When the translation results of the EBMT system using translation templates with type constraints were compared with the translation results of the EBMT system using translation templates without type constraints, the type constraints have eliminated more wrong translations from the translation results.

The type expression that is inferred for a variable replacing a difference with two symbols depends on the shortest path between those two symbols in their type lattice. The youngest ancestor of those symbols is the generalization of that difference. By selecting the youngest ancestor for those symbols, we hope that we get the most specific generalization for those symbols. The youngest ancestor may not be most specific generalization depending on those symbols and the structure of the type lattice. Although there can be another techniques to find the most specific generalization, the shortest path is one of the good techniques.

The inferred type expression by the generalization technique presented here is a most specific generalization. If we do not use any type constraint for a variable, it will be most general generalization. Other generalizations may be preferred by using certain generalization metrics. In this case, the regular expressions can be a better choice to represent type expressions. We are currently investigating these alternatives.

In this paper, the constraints for the variables are type constraints. The generalization technique described here can be also used in the inference of the semantic constraints if the semantic lattices (similar to Wordnet) are available for source and target languages. The quality of translation templates will depend on the quality of the used semantic lattices. The EBMT system in [7] also tries to generalize semantic features.

References

[1] Brown, R. D., Clustered Transfer Rule Induction for Example-Based Translation, in: *Recent Advances in Example-Based Machine Translation,* Carl, M., and Way, A. (eds.), The Kluwer Academic Publishers, Boston, 2003, pp: 287-306.

[2] Carl, M., Inducing Translation Grammars from Bracket Alignments, in: *Recent Advances in Example-Based Machine Translation*, Carl, M., and Way, A. (eds.), The Kluwer Academic Publishers, Boston, 2003, pp:339-361.

[3] Cicekli, I., and Guvenir, H. A., Learning Translation Templates from Bilingual Translation Examples, in: *Recent Advances in Example-Based Machine Translation*, Carl, M., and Way, A. (eds.), The Kluwer Academic Publishers, Boston, 2003, pp: 255-286.

[4] Cicekli, I., and Guvenir, H. A., Learning Translation Templates from Bilingual Translation Examples, *Applied Intelligence*, Vol. 15, No. 1, 2001, pp: 57-76.

[5] Kaji H., Kida Y., and Morimoto Y., Learning Translation Templates from Bilingual Text, in: *Coling* (1992), pp: 672-678.

[6] McTait, K., Translation Patterns, Linguistic Knowledge and Complexity in EBMT, in: *Recent Advances in Example-Based Machine Translation*, Carl, M., and Way, A. (eds.), The Kluwer Academic Publishers, Boston, 2003, pp: 307-338.

[7] Matsumoto, Y., and Kitamura M., Acquisition of Translation Rules from Parallel Corpora, in: Recent Advances in Natural Language Processing, Amsterdam, John Benjamins, 1995, pp: 405-416.

[8] Nagao, M.A., Framework of a Mechanical Translation between Japanese and English by Analogy Principle, in: *Artificial and Human Intelligence*, Elithorn, A., and Banerji, R. (eds.), North Holland, Amsterdam, 1984, pp: 173-180.