TERCÜME KALIPLARININ MAKİNA ÖĞRENMESİ TEKNİKLERİ İLE TERCÜME ÖRNEKLERİNDEN ÖĞRENİLMESİ

PROJE NO: EEEAG-244

DOÇ. DR. İLYAS ÇİÇEKLİ

AĞUSTOS 1999
ANKARA

ÖNSÖZ

Bu projede örneğe dayalı İngilizce ve Türkçe arasında iki yönlü tercüme yapılabilen bilgisayar ile tercüme sistemi geliştirildi. Kullanılan yöntem var olan tercüme örnek kümelerinden tercüme kalıplarının öğrenilmesine ve bu tercüme kalıplarının başka metinlerin tercümelerinde kullanılmasına dayanır. Örneğe dayalı yöntemler bilgisayar ile tercüme alanında klasik yöntemlere alternatif olarak bu alandaki araştırmacılar tarafından yaygınca kullanılmaktadır. İnancımız odur ki, örneğe dayalı yöntemler bilgisayar ile tercüme alanında her zaman çok büyük önem taşıyacaktır.

Bu projede çalışan araştırmacı arkadaşlarım Doç. Dr. H. Altay Güvenir ve Zeynep (Öz) Orhan'a projeye yaptıkları değerli katkılardan dolayı teşekkürü bir borç bilirim. Bu projeye EEEAG-244 kodu ile mali destek veren Türkiye Bilimsel Araştırma Kurumuna da ayrıca teşekkürü bir borç bilirim.

Doç. Dr. İlyas Çiçekli

İÇİNDEKİLER

1. Giriş 7

2. Tercüme Kalıpların Yapıları 9

3. Tercüme Kalıplarının Öğrenilmesi 10
3.1. Benzerlik Tercüme Kalıplarının Öğrenilmesi (STTL Algoritması) 12
3.2. Farklılık Tercüme Kalıplarının Öğrenilmesi (DTTL Algoritması) 15
3.3. Eşleşme Dizisinde Farklı Sayıda Benzerlik ve Farklılık Olması 17
3.4. Boş Parçacıklı Farklılıklar 19
3.5. STTL ve DTTL Algoritmaların Performansları 20

4. Öğrenme Örnekleri 22

5. Tercüme İşlemi 25

6. Güvenirlik Faktörlerin Verilmesi 27
6.1. Sabit Kalıplara Güvenirlik Faktörlerinin Verilmesi 28
6.2. Genel Kalıplara Güvenirlik Faktörlerinin Verilmesi 29
6.3. Kalıp Kombinasyonlarına Güvenirlik Faktörlerinin Verilmesi 30
6.4. Güvenirlik Faktörleri ile Tercüme İşlemi 33

7. Sonuç 33

Kaynakça 35

Yayınlamış Makale ve Bildiriler 37

Bibliyografik Bilgi Formu 38

Şekil Listesi

Şekil 1. STTL Algoritması 14

Şekil 2. DTTL ALgoritması 17

Şekil 3. Sistemin Genel Yapısı 27

Şekil 4. Kalıp Kombinasyonlarına Güvenirlik Faktörlerinin Verilmesi 31

ÖZ

Bu proje kapsamında örneğe dayalı bilgisayar ile herhangi iki dil arasında tercüme yapabilen bir sistem geliştirdik. Bizim önerdiğimiz yöntemde iki dil arasındaki tercüme kalıpları verilen tercüme örneklerinden makina öğrenmesi teknikleri kullanılarak öğrenilir. Daha sonra bu öğrenilen tercüme kalıpları diğer metinlerin tercüme edilmesinde kullanılır. Bizim temel tercüme kalıplarını öğrenme algoritmamız verilen iki tercüme örneğindeki benzerliklerden ve farklılıklardan faydalanarak yeni tercüme kalıplarını öğrenir. Verilen iki tercüme örneğindeki kaynak dildeki cümleler arasındaki benzerlikler hedef dildeki cümleler arasındaki benzerliklere karşılık gelmelidir. Aynı şeklide, kaynak dildeki cümleler arasındaki farklılıklarda hedef dildeki cümleler arasındaki farklılıklara karşılık gelmelidir. Bu temel varsayımdan yola çıkarak makina öğrenmesi yöntemlerini kullanan tercüme kalıplarını öğrenme algoritması geliştirilmiştir. Buna ilavaten tercüme kalıplarına istatiksel yöntemler kullanarak güvenirlik faktörleri veren bir parça sistemize eklenmiştir. Bu güvenirlik faktörlerin amacı üretilen yanıtların bu faktörlere göre sıralayarak, ilk sıradaki yanıtların doğru yanıtı kapsama olasalığını artırmaktır.

Anahtar Sözcükler: Doğal Dil İşleme, Bilgisayar ile Tercüme, Makina Öğrenmesi

ABSTRACT

In this project, we have developed a bi-directional translation system which can work between any two language pairs. In our method, translation templates for a pair of language are learned from a set of translation examples by using machine learning techniques. These learned translation templates are used in the translation of other texts. Our proposed mechanism learns new translation templates from similarities and differences between two given translation examples. Given two translation examples, the similar parts of the sentences in the source language should match the similar parts of the sentences in the target language. Similarly, the difference parts should correspond to the respective parts in the translation examples. Our translation template learner algorithm bases on this basic heuristic. In addition to this, we added a subsystem which gives confidence factors to learned translation templates. The purpose of these confidence factors is to sort the results of translation according to these confidence factors so that the correct result can end-up among the top results.

Keywords: Natural Language Processing, Machine Translation, Machine Learning

1. Giriş

Bu proje kapsamında örneğe dayalı olarak bilgisayar ile herhangi iki dil arasında tercüme yapabilen bir sistem geliştirilmiştir. Bizim önerdiğimiz yöntemde iki dil arasındaki tercüme kalıpları verilen tercüme örneklerinden makina öğrenmesi teknikleri kullanılarak öğrenilir. Daha sonra bu öğrenilen tercüme kalıpları diğer metinlerin tercüme edilmesinde kullanılır. Bizim temel tercüme kalıplarını öğrenme algoritmamız verilen iki tercüme örneğindeki benzerliklerden ve farklılıklardan faydalanarak yeni tercüme kalıpları öğrenir. Verilen iki tercüme örneğindeki kaynak dildeki cümleler arasındaki benzerlikler hedef dildeki cümleler arasındaki benzerliklere karşılık gelmelidir. Aynı şeklide, kaynak dildeki cümleler arasındaki farklılıklarda hedef dildeki cümleler arasındaki farklılıklara karşılık gelmelidir. Bu temel varsayımdan yola çıkarak makina öğrenmesi yöntemlerini kullanan tercüme kalıplarını öğrenme algoritması geliştirilmiştir. Bu projedeki temel amacımız proje başında ufak bir parçası elimizde olan öğrenme algoritmamızı geliştirmek ve öğrenme algoritmamızı büyük örnek kümeleri üzerinde deneyerek önerdiğimiz yönteminin geçerli bir yöntem olduğunu kanıtlamak idi. Bu yüzden öğrenme algoritmamızı geliştirerek bir örneğe dayalı tercüme sistemi geliştirdik. Bu sistem derlediğimiz büyük örnek kümeleri ile test edildi.

Geleneksel bilgisayar ile tercüme (Machine Translation) yöntemleri birbiri arasında tercüme yapılacak doğal diller hakkında o dillerin yapıları, kelimelerinin anlam bilgileri gibi çok fazla bilgiye ihtiyaç duyduklarından, bu tip yöntemlerin gerçekleştirilmesi çok zor ve pahalıdır. Örneğin geleneksel yöntemleri kullanan KBMT-89 [6] sistemi çok büyük boyutlarda sözlük (lexicon), gramer kuralları, bir dilin yapısını diğer bir dile çevirmek için gerekli olan çevirme kuralları ve anlam bilgilerini kapsayan dünya bilgilerine (ontology) ihtiyaç duymaktadır. Bu gerekli bilgilerin uzmanlar tarafından toplanması çok pahalı ve zaman alıcı bir iştir. Bu yüzden bilgisayarla tercüme alanındaki araştırmıcılar bu bilgilerin bir şekilde otomatik olarak elde etmek isterler. KBMT-89 sisteminin devamı olan KANT [15] sisteminde de kullanılan sözlüğün otomatik olarak örnek metinler toplanabilmesi için bir otomatik sözlük yaratma metodu [13] kullanılmıştır. Bu proje kapsamında geliştirelen teknik morfolojik kurallar hariç tercüme için gerekli diğer bütün bilgileri otomatik olarak paralel örnek metinlerden elde etmektedir.

Bilgisayarla tercüme alanında diğer bir yaklaşım da korpusa (örnek metinlere) dayalı yöntemlerdir. Bu yöntemler doğal diller hakkındaki gerekli bilgileri bu büyük korpuslardan elde ederler. Korpusa dayalı yöntemler genel olarak ikiye ayrılır: *istatiksel yöntemler* ve *örneğe dayalı yöntemler*. İstatiksel yöntemler tercüme anında kaynak dildeki bir yapının sonuç dildeki karşılığını bulmak için olası yapılar arasından yapacağı seçime istatiksel ölçümler yaparak karar verir. Örneğe dayalı yöntemler (example-based machine translation - EBMT) ise şekilsel karşılaştırma (pattern matching) yöntemleri kullanarak bu seçimleri yaparlar. Şu ana kadar gerçekleştirilen çoğu bilgisayarla tercüme sistemleri bir şekilde korpusa dayalı yöntemlerin bir kısmını sistemlerin içinde kullanmışlardır.

EBMT (example-based machine translation) ilk olarak olarak Nagao [16] tarafından İngilizce ve Japonca arasında örneğe dayalı bir yöntem olarak önerilmiştir. EBMT teki temel fikir verilen bir cümle tercüme edileceğinde, paralel örnek tercüme kümesindeki tercüme örnekleri

ile karşılaştırılarak verilen cümleye en çok benzeyen tercüme örnekleri bulunmaya çalışılır. Bu benzer tercüme örnekleri o cümlenin tercümesinde kullanılır. Nagao'nun ilk önerisinden sonra bilgisayarla tercüme alanındaki bir çok araştırmacı [5,11,20,21,22,23] paralel örnek tercüme metinlerine dayalı değişik yöntemler önerdiler. Bazı araştırmacılar [3,24] ise örnek metinleri sadece iki dil arasındaki sözlükleri öğrenmek amacıyla kullandılar. Tabi bu örnek metinlere dayalı yöntemleri kullanabilmek için örnek metinlerin cümle bazında diğer dille denkliği ayarlanmış olmalı. Bu yüzden bazı araştırmacılar [1,2,17] bu metinleri cümle bazında ayarlanması üzerinde çalıştılar. Kaji [10] İngilizce ve Japonca arasında şekilsel (syntactic) yapıların denkliğini örnek metinlerden öğrenmeye çalıştı. Onun kullandığı yöntem bizim daha önceki bir çalışmamız [7] ile benzerlik göstermektedir. Bizim bu proje kapasamında kullandığımız teknikte sadece bir sözlükte olduğu gibi sabit kalıpları öğrenmekte kalmayıp, ayni zamanda yapısal benzerlikleri (syntactic yapı değil) öğreniriz.

Örneğe dayalı öğrenme yöntemleri makina ile öğrenme alanınında uzun süredir kullanılmaktadır. Örneğe dayalı öğrenmeyi ilk öneren Medin ve Shaffer [14] olmuştur. Buradaki temel fikir yeni ve işe yarar kuralların bir örnek kümesinden öğrenilmesidir [9,12,19]. Bu yöntem makina ile tercüme alanınında tercümenin daha önceki tercüme örneklerinden faydalınarak yapılması olarak uygulanmaktadır.

Biz ilk yaklaşımızda syntactic yapıları örnek metinlerden öğrenmekti [7]. Daha sonra bu proje kapsımınında geliştirmiş olduğumuz tercüme kalıplarının örnek metinlerden öğrenme tekniği [4,8,18] üzerinde çalıştık. Bu yöntemin detayları bu raporda anlatılmıştır.

Bu raporda anlatılan tercüme kalıpları öğrenme algoritması iki dil arasında verilen iki tercüme örneği arasındaki benzerlikleri ve farklılıkları kullanan bir yönteme dayalıdır. Bu yönteme göre birinci dildeki cümleler arasındaki benzerlikler ikinci dildeki cümleler arasındaki benzerliklere denk düşmelidir. Buna ilavaten farklılıklarda birbirlerine denk düşmelidir. Eğer cümleler arasında hiç bir benzerlik yoksa öğrenme işlemi gerçekleştirilemez. Bu yöntemi biraz daha açıklamak için İngilizce ve Türkçe arasında verilen aşağıdaki iki tercüme örneğine bakalım.

I will drink orange juice ↔ portakal suyu içeceğim (1)
I will drink coffee ↔ kahve içeceğim

Her iki dildeki benzerlikler altları çizili olarak verilmiştir. Cümlelerin diğer parçaları farklılıklardır. Kullandığımız ilk yönteme göre, İngilizce tarafındaki genelleştirilmiş benzerlik yapısı "I will drink *XE*" Türkçe tarafındaki genelleştirilmiş benzerlik yapısı "*XT* içeceğim" e denk düşmelidir. Burada *XE* İngilizce tarafındaki ugun bir parçacığı, ve *XT* de İngilizce tarafında ki parçacığın tercümesi olan Türkçe parçacığı belirtir. Buna ilavaten ilk yöntem "orange juice" nun "portakal suyu" na, ve "coffee" nin de "kahve" ye denk düştüğünü öğrenebilir. Kullandığımız ikinci yöntem ise cümlelerdeki farklılıkları tutar ve benzerlikleri değiştirerek genelleştirilmiş tercüme kalıplarını öğrenir.

Bu raporun yapısı şöyledir. 2. Bölüm tercüme kalıplarının sistemizdeki yapısını anlatır. Öğrenme algoritmasının bütün detayları 3. Bölüm de verilmiştir. 4. Bölüm tercüme örnek

kümelerinden alınan tercüme örnekleri üzerindeki davranışını anlatır. 5. Bölüm ise öğrenilen tercüme kalıplarının diğer cümlelerin tercümesinde nasıl kullanıldığını anlatır. Tercüme sisteminin doğru yanıtı bulma olasalığını artırmak için tercüme kalıplarına eklenen güvenirlik faktörlerin kalıplara nasıl verildiği 6. Bölümde sunulmuştur. 7. Bölüm ise projemizin sonuçlarını sunar.

2. Tercüme Kalıplarının Yapıları

Bir tercüme kalıbı genelleştirilmiş bir tercüme örneğidir. Bir tercüme örneği, örneğin iki tarafındaki belirli parçaları değişkenlerle değiştirip ve bu iki taraftaki değişkenleri birbiri ile ilişkilendirilerek genelleştirilir. Her türcüme kalıbının iki tarafında da ayni sayıda değişken vardır, ve bir taraftaki bir değişken muhakkak diğer bir değişken ile ilişkilendirilmiştir. Bazen bir tercüme kalıbı hiç bir değişken içermeyebilir. Böyle kalıplar *sabit tercüme kalıbı* olarak adlandırlmıştır. Verilen her tercüme örneği sistemde ayni zamanda *sabit tercüme kalıbı* olarak ta saklanır.

Örneğin (1) de verilen tercüme örneklerinden, aşağıdaki tercüme kalıpları öğrenilir:

- `I will drink` *X* ↔ *Y* `içeceğim` **if** *X* ↔ *Y*
- `orange juice ↔ portakal suyu`
- `coffee ↔ kahve`

Birinci tercüme kalıbı, eğer *X* ve *Y* birbirinin tercümeleri ise İngilizce'deki "`I will drink` *X*" ve Türkçe'deki "*Y* `içeceğim`" yapılarınının da birbirlerinin tercümeleri olacağını gösterir. Bu demektir ki, eğer daha önceden İngilizce'deki "`tea`" nin Türkçedeki "`çay`" a denk geldiği öğrenilmiş ise, birinci kalıp kullanılarak "`I will drink tea`" cümlesi "`çay içeceğim`" olarak tercüme edilebilinir. İkinci ve üçüncü kalıplar ise hiç bir değişken içermeyen sabit kalıplardır.

Öğrenme algoritmamız verilen tercüme örnekleri arasındaki benzerlikleri ve farklılıkları kullanarak tercüme kalıplarını öğrendiğinden, tercüme örneklerinde ki cümlelerin nasıl gösterildiği önemli bir rol oynayacaktır. Daha önceden de belirtildiği gibi öğrenme algoritması (1) de olduğu gibi normal cümleleri içeren tercüme örnekleri üzerinde çalışabilir. Bu durumda kullanacağımız bir tercüme örneği kümesi üzerinde öğrenme algoritmasını ona uygulamadan önce hiç bir işlem yapmamıza gerek yoktur, ve hiç bir başka gramer bilgisi kullanmamış oluruz. Bu durumda tercüme örneklerindeki (dolayısıyla tercüme kalıplarında ki) en küçük parça bir kelime olacaktır, ve her cümle kelimelerin bir listesidir. Bu demektir (1) deki "`içeceğim`" kelimesi içindeki gelecek zaman ve birinci tekil şahıs bilgileri hiç kullanılmadan en küçük parça olarak kabul edilecektir.

Türkçe gibi agulanatif bir dilde kelimelerin sadece normal hallerini (surface form) kullanmak öğrenilen tercüme kalıpların genelliğini azaltır. Örneğin aşağıdaki tercüme örneklerini ele alalım:

- they are running ↔ koşuyorlar
- they are walking ↔ yürüyorlar

Türkçe tarafında her iki örnekte de sadece birer kelime vardır. Her kelime en küçük parça olarak düşünüldüğünden "koşuyorlar" ve "yürüyorlar" kelimeleri arasında hiç bir benzerlik olmayacaktır. Bu yüzden eğer kelimelerin normal hallerini kullanırsak, bu iki örnekten hiç bir tercüme kalıbı öğrenilmeyecektir. Bu yüzden kelimelerin normal halleri yerine, onların şekilsel hallerini (lexical form) kullanırız. Bu durumda her bir kelime kök kelime ve eklerden oluşur. Kısaca kelimenin bu şekilsel parçaları morfem (kök kelimede dahil) olarak adlandırılır. Böylece bir cümledeki en küçük parça bir morfem olacaktır, ve bir cümle bir morfemler listesi olacaktır. Biz bu projede kelimelerin şekilsel yapılarını kullandık. Yukarıdaki örnek kelimelerin şekilsel yapıları kullanıldığında aşağıdaki gibi gösterilecektir.

- they are run+ing ↔ koş+Hyor+lAr
- they are walk+ing ↔ yürü+Hyor+lAr

Bu örneklerde + morfemlerin sınırlarını, ve Hyor ve lAr şimdiki ve üçüncü çoğul şahıs eklerine ifade eden morfemlerdir. Buradaki H ve A harfleri kelimenin normal halinde Türkçenin ses uyumuna göre belirlenecek sesli harflerdir. Böylece bu iki tercüme örneğindeki Türkçe cümlelerin ikiside üçer morfemden oluşmaktadır, ve ikişer morfemi benzeşir (altı çizili olanlar). Öğrenme algoritmalarımızdan benzerik tercüme kalıplarını öğrenme algoritmamızı (similarity translation template learning, STTL) bu iki örneğe uygularsak, aşağıdaki 3 tercüme kalıbı öğrenilecektir.

- they are *X* +ing ↔ *Y* +Hyor+lAr **if** *X* ↔ *Y*
- run ↔ koş
- walk ↔ yürü

Kelimelerin şekilsel yapılarını kullanmamız Türkçe deki sesli ve sessiz uyum kuralları üzerinde bir genelleme, İngilizce de fiillerin zamanlara göre farklı ifade edilmeleri üzerinde genelleme yapmamıza olanak sağlar. Bu proje kapsımında geliştirdiğimiz örnek kümelerindeki cümleler kelimelerin şekilsel yapıları kullanılarak gösterilmiştir. Tabi bu durumda kelimelerin normal hallerini şekilsel haline, vede şekilsel halinden normal haline çeviren morfolojik işlemcilere ihtiyacımız oldu. Türkçe ve İngilizce için elimizde mevcut morfolojik işlemcileri kullandık.

3. Tercüme Kalıplarının Öğrenilmesi

Öğrenme algoritması (Translation Template Learning - TTL algoritması) tercüme örnek kümesinden alınan iki tercüme örneği (E_a ,E_b) arasındaki benzerlikleri ve farklılıkları kullanarak tercüme kalıplarını öğrenir. Bir tercüme örneği, bir İngilizce cümle ve bir de onun

karşılığı olan Türkçe cümleden oluşur. İki parçası olan bir tercüme örneği E yi EE↔ET şeklinde ifade ederiz. Burada EE bir İngilizce cümle (veya cümle parçacığı), ve ET bir Türkçe cümledir (veya cümle paçacığı). Bilindiği gibi her cümle morfemlerden oluşan bir listedir.

TTL algoritması ilk önce verilen iki tercüme örneği E_a ve E_b nin parçaları arasında benzerlikleri bulmaya çalışır. Diğer bir deyişle, EE_a ve EE_b arasındaki benzerlikleri, vede ET_a ve ET_b arasındaki benzerlikleri bulmaya çalışırız. Eğer her iki taraftada en az bir benzerlik bulamamış isek bu örneklerden tercüme kalıbı öğrenemeyiz. Eğer her iki taraftada en az bir benzerlik var ise, aşağıdaki gibi bir *eşleşme dizisi* $M_{a,b}$ yaratırız.

$$SE_0,DE_0,SE_1,...,DE_{n-1},SE_n \leftrightarrow ST_0,DT_0,ST_1,...,DT_{m-1},ST_m \quad \text{burada } n,m\geq 1$$

Burada her bir SE_k (ST_k) terimi EE_a ve EE_b arasındaki (ET_a ve ET_b arasındaki) bir *benzerliği*, vede her bir DE_k (DT_k) terimi EE_a ve EE_b arasındaki (ET_a ve ET_b arasındaki) bir *farklılığı* gösterir. Bir benzerlik her iki örnekte de ayni olan bir parçadır. Bir farklılık ilk parçası E_a dan ikinci parçası E_b den gelen iki parçalı ($D_{k,a}$,$D_{k,b}$) bir ifadedir, ve bu iki parça hiç bir ortak morfem içermez. Bir eşleşme dizisi $M_{a,b}$ yarıtılırken aşağıdaki şartlar sağlanmalıdır.

- Bir farklılığın iki parçası hiç bir ortak morfem içeremez, ve bu iki parçanın her biri de en aşağı bir morfem içermelidir.
- SE_0, SE_n, ST_0, ve ST_m benzerlikleri boş liste olabilirler, ama SE_1 ve SE_{n-1} arasındaki benzerlikler vede ST_1 ve ST_{m-1} arasındaki benzerlikler boş liste olamazlar. Her koşulda eşleşme dizisinin her iki tarafında da en aşağı boş olmayan bir benzerlik olmalıdır.
- Bir benzerlik kendinden önceki gözükmüş olan farklılıklardaki bir parçayı içeremez.

Bu şartlar bir eşleşme dizisinin her iki tarafında da en az bir farklılık vede boş olmayan bir benzerlik olmasını garantiler. Ayni zamanda bu şartlar var ise tek bir eşleştirmenin varlığını garanti eder.

TTL algoritması uygulanmadan önce *eşleştirme algoritması* verilen iki tercüme örneğine uygulanır. Eğer eşleştirme algoritması yukarıdaki şartları sağlayan bir eşleşme dizisi bulabilirse, be eşleşme dizisine TTL algoritması uygulanır. Eğer bir eşleşme dizisi bulunamazsa TTL algoritması bu iki örneğe uygulanamıyor demektir.

Aşağıdaki iki tercüme örneğini ele alalım:

- `I bought the book for Cathy ↔ Cathy için kitabı satın aldım`
- `I bought the ring for Cathy ↔ Cathy için yüzüğü satın aldım`

Bu örnekler şekilsel yapıda aşağıdaki gibi ifade edileceklerdir:

- `I buy+p the book for Cathy ↔ Cathy için kitap+yH satın al+DH+m`
- `I buy+p the ring for Cathy ↔ Cathy için yüzük+yH satın al+DH+m`

Bu iki örnekten eşleştirme algoritması aşağıdaki eşleşme dizisini bulacaktır:

```
I buy+p the (book,ring) for Cathy ↔
    Cathy için (kitap,yüzük) +yH satın al+DH+m                    (2)
```

Diğer bir deyişle bu eşleşme dizisinin parçaları aşağıdaki gibi olacaktır:

SE_0 = `I buy+p the` DE_0 = `(book,ring)` SE_1 = `for Cathy`
ST_0 = `Cathy için` DT_0 = `(kitap,yüzük)` ST_1 = `+yH satın al+DH+m`

Eşleştirme algoritması iki tercüme örneği için bir eşleşme dizisi bulduktan sonra, TTL algoritması bu eşleşme dizisini kullanarak yeni tercüme kalıpları öğrenmeye çalışır. Bu amaçla TTL algoritması iki değişik yöntem kullanarak tercüme kalıplarını öğrenmeye çalışır. İlk yöntem eşleşme dizisinin İngilizce tarafındaki her bir farklılığın Türkçe tarafındaki hangi farklılığa denk düştüğünü bulmaya çalışır. Eğer iki taraftaki farklılık sayısı ayni ve her bir farklılığın diğer tarafta hangi farklılığa denk düştüğü bulunabiliniyorsa, ilk yöntem bu farklılıkları uygun değişkenler ile değiştirerek yeni bir tercüme kalıbı öğrenir. Burada eşleşme dizisindeki benzerlikler aynen muhafaza edilerek vede farklılıklar değişkenler ile değiştirilerek tercüme kalıbı öğrenildiğinden, TTL algoritmasının bu ilk yöntemini *benzerlik tercüme kalıplarının öğrenilmesi (similarity translation template learning, STTL)* olarak adlandırıyoruz. Benzer şekilde eğer iki taraftaki benzerlik sayısı ayni ve her bir benzerliğin diğer tarafta hangi benzerliğe denk düştüğü bulunabiliniyorsa, ikinci yöntemde bu benzerlikleri uygun değişkenler ile değiştirerek tercüme kalıplarını öğrenir. Burada da eşleşme dizisindeki farklılıklar aynen muhafaza edilerek vede benzerlikler değişkenler ile değiştirilerek tercüme kalıpları öğrenildiğinden, TTL algoritmasının bu ikinci yöntemi *farklılık tercüme kalıplarının öğrenilmesi (difference translation template learning, DTTL)* olarak adlandırıyoruz. Şekilsel olarak benzerlik tercüme kalıpları ile farklılık tercüme kalıpları arasında hiç bir fark yoktur.

3.1. Benzerlik Tercüme Kalıplarının Öğrenilmesi (STTL Algoritması)

Eğer bir eşleşme dizisinin her iki tarafında da sadece bir farklılık var ise, bu farklılıkların parçaları birbirlerinin tercümesi olması gerekir. Diğer bir deyişle birbirine denk düşen farklılıkları belirleyebiliriz. Bu durumda eşleşme dizisi aşağıdaki gibi olmalıdır:

SE_0, DE_0, SE_1 ↔ ST_0, DT_0, ST_1

Burada DE_0 ve DE_0 birbirine denk düşen farklılıklar olduğundan, STTL algoritması aşağıdaki tercüme kalıbını bu farklılıkları değişkenler ile değiştirerek öğrenebilir.

SE_0 XE_1 SE_1 ↔ ST_0 XT_1 ST_1 **if** $XE_1 \leftrightarrow XT_1$

Buna ilaveten birbirine denk düşen ($DE_{0,a}$,$DE_{0,b}$) ve ($DT_{0,a}$,$DT_{0,b}$) farklılıklarından aşağıdaki iki sabit tercüme kalıbı daha öğrenilir.

$DE_{0,a} \leftrightarrow DT_{0,a}$
$DE_{0,b} \leftrightarrow DT_{0,b}$

Örneğin (2) verilen eşleşme dizisi her iki tarafta da sadece tek bir farklılık içerdiğinden, bunlar birbirine denk düşer. Böylece STTL algoritması tarafından bir tanesi bu farklılıkların değişkenler ile değiştirmesinden, diğer iki taneside `(book,ring)` ve `(kitap,yüzük)` farklılıklarının denkliğinden olmak üzere aşağıdaki üç tercüme kalıbı öğrenilir.

- `I buy+p the` XE_1 `for Cathy ↔ Cathy için` XT_1 `+yH satın al+DH+m`
  if $XE_1 \leftrightarrow XT_1$
- `book ↔ kitap`
- `ring ↔ yüzük`

Fakat kabul edelim ki her iki tarafta da *n* farklılık var, ve *n>1*. Daha önceden elde edilmiş bir bilgiyi kullanmaksızın bir tarafta ki farklılığın diğer tarafta ki hangi farklılığa denk düştüğünü bilmenin imkanı yoktur. Bu yüzden böyle bir durumda yeni tercüme kalıplarının öğrenebilmesi daha önceden öğrenilmiş tercüme kalıplarındaki bilgiye dayanacaktır. Böyle bir durumda STTL algoritması verilmiş olan eşleşme dizisi üzerinde birbirine denk düşen *n-1* farklılığı daha önceden öğrenilmiş tercüme kalıplarının yardımıyla bulmaya çalışır. İngilizce tarafındaki *k* ıncı farklılık ($DE_{k,a}$,$DE_{k,b}$) Türkçe tarafındaki *l* inci farklılığa ($DT_{l,a}$,$DT_{l,b}$) denk düşebilmesi için daha önceden aşağıdaki iki tercüme kalıbı öğrenilmiş olmalıdır.

$DE_{k,a} \leftrightarrow DT_{l,a}$
$DE_{k,b} \leftrightarrow DT_{l,b}$

Eşleşme dizisindeki birbirine denk düşen *n-1* farklılık bulunduğunda, her iki taraftada birer tane kontrol edilmemiş farklılık kalır. Onlar da birbirine denk düşmelidir. Böylece her bir farklılığın diğer tarafta hangi farklılığa denk düştüğü bulunmuş olur. Kabul edelim ki aşağıdaki birbirine denk düşen farklılıkların listesi olsun.

$DD_1, DD_2, \ldots, DD_n$

Burada DD_n kontrol edilmemiş iki farklılık, ve her bir DD_i farklılık çifti de ($DE_{k,i}$,$DT_{l,i}$) şeklinde olup, DE_k ve DT_l nin denkliğini gösterir. Her bir DD_i farklılık çifti için, eşleşme dizisindeki $DE_{k,i}$ farklılığı XE_i değişkeni ile, ve $DT_{l,i}$ farklılığı XT_i değişkeni değiştirilir. Böylece bütün farklılıkları değişkenler ile değiştirilmiş olan yeni bir eşleşme dizisi $M_{a,b}DVars$ elde edilir. Bundan sonra STTL algoritması aşağıdaki tercüme kalıbını öğrenir.

$M_{a,b}DVars$ **if** $XE_1 \leftrightarrow XT_1$ **and** ... **and** $XE_n \leftrightarrow XT_n$

Buna ilaveten DD_n farklılık çiftindeki ($DE_{k,n,a}$,$DE_{k,n,b}$) farklılığının ($DT_{l,n,a}$,$DT_{l,n,b}$) farklığına denkliğinden aşağıdaki iki sabit tercüme kalıbı öğrenilir.

$DE_{k,n,a} \leftrightarrow DT_{l,n,a}$
$DE_{k,n,b} \leftrightarrow DT_{l,n,b}$

Aşağıdaki tercüme örneklerini ele alalım.

- I break+p the window ↔ pencere+yH kır+DH+m
- You break+p the door ↔ kapı+yH kır+DH+n

Bu tercüme örneklerinin aşağıda verilen eşleşme dizisinin her iki tarafında da ikişer farklılık vardır.

```
(i,you) break+p the (window,door)
    ↔ (pencere,kapı) +yH kır+DH (+m,+n)                    (3)
```

Daha önceki elde edişmiş hiç bir bilgiyi kullanmadan, sol taraftaki "i" nin sağ taraftaki "pencere" ye mi, yoksa "+m" e denk düştüğünü bilemeyiz. Ama eğer daha önceden "i" nin "+m" e, vede "you" nun da "+n" e denk geldiği öğrenilmişse, (window,door) farklılığı (pencere,kapı) farklılığına denk düşmelidir. Böylece STTL algoritması aşağıdaki üç tercüme kalıbını öğrenebilir.

- XE_1 break+p the XE_2 ↔ XT_2 +yH kır+DH XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
- window ↔ pencere
- door ↔ kapı

---

procedure STTL($M_{a,b}$)
begin

- Kabul edelim ki tercüme örnekleri E_a ve E_b nin eşleşme dizisi $M_{a,b}$ aşağıdaki yapıda olsun:
  $SE_0,DE_0,...,DE_{n-1},SE_n \leftrightarrow ST_0,DT_0,...,DT_{m-1},ST_m$

if *n=m=1* **then**

- aşağıdaki tercüme kalıplarını öğren:
  - $SE_0\ XE_1\ SE_1 \leftrightarrow ST_0\ XT_1\ ST_1$ *if* $XE_1 \leftrightarrow XT_1$
  - $DE_{0,a} \leftrightarrow DT_{0,a}$
  - $DE_{0,b} \leftrightarrow DT_{0,b}$

else if *n=m>1* **and**
n-1 tane birbirine denk düşen farklılık $M_{a,b}$ eşleşme dizisi içinde belirlenebilir **then**

- Kabul edelim ki $((DE_{k,n,a},DE_{k,n,b}),(DT_{l,n,a},DT_{l,n,b}))$ kontrol edilmeyen farklılık çifti olsun.
- Kabul edelim ki $((DE_{k,1},DT_{l,1}),...,(DE_{k,n},DT_{l,n}))$ kontrol edilmemiş farklılık çiftinide kapsayan denk düşen bütün farklılık çiftleri olsun.
- Her bir farklılık çifti $(DE_{k,i},DT_{l,i})$ için, $M_{a,b}$ eşleşme dizisindeki $DE_{k,i}$ yı XE_i değişkeni ile, ve $DT_{l,i}$ yide XT_i değişkeni ile değiştirerek yeni $M_{a,b}DVars$ eşleşme dizisini elde et.
- aşağıdaki tercüme kalıplarını öğren:
  - $M_{a,b}DVars$ *if* $XE_1 \leftrightarrow XT_1$ *and ... and* $XE_n \leftrightarrow XT_n$
  - $DE_{k,n,a} \leftrightarrow DT_{l,n,a}$
  - $DE_{k,n,b} \leftrightarrow DT_{l,n,b}$

end

Şekil 1. STTL Algoritması

Genelde, STTL algoritması bir eşleşme dizisinin iki tarafındaki farklılıkların sayısı ayni ve bir taraftaki farklılık sayısı 0 dan büyük ise (n=m>0), yeni tercüme kalıplarını sadece n-1 farkılık eşleşmesi daha önceden öğrenilmiş ise öğrenebilir. STTL algoritmasının detayları Şekil 1 de verilmiştir.

3.2. Farklılık Tercüme Kalıplarının Öğrenilmesi (DTTL Algoritması)

Eğer bir eşleşme dizisinin her iki tarafında da sadece boş olmayan birer benzerlik var ise, bu benzerlikler birbirlerinin tercümesi olması gerekir. Diğer bir deyişle birbirine denk düşen benzerlikleri belirleyebiliriz. Bu durumda eşleşme dizisi her bir tarafında 1 veya 2 farklılık bulunacaktır, ve bir taraftaki farklılık sayısı diğer taraftakine eşit olmayabilir. Eşleşme dizisinin herhangi bir tarafı aşağıdaki yapılardan biri gibi olmalıdır:

- S_0, D_0 burada S_1 bir boş dizindir.
- D_0, S_1 burada S_0 bir boş dizindir.
- D_0, S_1, D_1 burada S_0 ve S_2 boş dizindirler.

Bu durumda İngilizce tarafındaki tek benzerliği *XE* değişkeni ile, Türkçe tarafındaki tek benzerliği de *XT* değişkeni ile değiştiririz. Bu elde ettiğimiz yeni eşleşme dizisini de farklılıklardaki parçaları ikiye ayırarak değişkenli iki yeni çeşit eşleşme dizisi (M_aSVars ve M_bSVars) elde ederiz. Bunlar aşağıdaki yapıda olacaklardır.

- $ME_aSVars \leftrightarrow MT_aSVars$
- $ME_bSVars \leftrightarrow MT_bSVars$

Sonuç olarak aşağıdaki tercüme kalıpları DTTL algoritması tarafından öğrenilir.

- $ME_aSVars \leftrightarrow MT_aSVars$ **if** $XE \leftrightarrow XT$
- $ME_bSVars \leftrightarrow MT_bSVars$ **if** $XE \leftrightarrow XT$

Bunlara ek olarak ta DTTL algoritması *SE* ve *ST* benzerliklerinin (SE İngilizce tarafındaki boş olmayan tek benzerlik, ve ST Türkçe tarafında boş olmayan tek benzerliktir) denkliğinden aşağıdaki sabit tercüme kalıbını öğrenir.

- $SE \leftrightarrow ST$

Örneğin (3) te verilen eşleşme dizisi her iki tarafta da sadece tek bir benzerlik içerir. DTTL algoritması bu eşleşme dizisinden aşağıdaki üç tercüme kalıbını öğrenir.

- `i XE window ↔ pencere XT +m` **`if`** `XE ↔ XT`
- `you XE door ↔ kapı XT +n` **`if`** `XE ↔ XT`
- `break+p the ↔ +yH kır+DH`

Şimdi kabul edelim ki bir eşleşme dizisinin her iki tarafında da ayni sayıda benzerlik var, ama her iki taraftaki benzerliklerin sayısı bir den fazla olsun. Bu durumda eğer elimizde daha önceden elde edilmiş bir bilgi yoksa, bir benzerliğin diğer tarafta hangi benzerliğe denk düştüğünü bilmemize imkan yoktur. Eğer DTTL algoritması bir taraftaki benzerliklerin bir tanesi hariç diğer tarafta hangi benzerliklere denk düştüğünü daha önceden öğrenilmiş tercüme kalıplarını kullanarak belirleyebilirse bu eşleşme dizisinden yeni tercüme kalıpları öğrenebilir. Bir benzerlik SE_k diğer taraftaki ST_l benzerliğine ancak daha önceden aşağıdaki sabit tercüme kalıbı öğrenilmiş ise denk düşüyor denir.

$SE_k \leftrightarrow ST_l$

Eşleşme dizisindeki birbirine denk düşen *n-1* benzerlik bulunduğunda, her iki taraftada birer tane kontrol edilmemiş benzerlik kalır. Onlar da birbirine denk düşmelidir. Böylece her bir benzerliğin diğer tarafta hangi benzerliğe denk düştüğü bulunmuş olur. Kabul edelim ki aşağıdaki liste birbirine denk düşen benzerliklerin listesi olsun.

$SS_1, SS_2, \ldots, SS_n$

Burada SS_n kontrol edilmemiş iki benzerlik çiftini, ve her bir SS_i benzerlik çifti de $(SE_{k,i},ST_{l,i})$ şeklinde olup, SE_k ve ST_l nin denkliğini gösterir. Her bir SS_i benzerlik çifti için, eşleşme dizisindeki $SE_{k,i}$ benzerliği XE_i değişkeni ile, ve $ST_{l,i}$ benzerliği XT_i değişkeni ile değiştirilir. Böylece bütün benzerlikleri değişkenler ile değiştirilmiş olan yeni bir eşleşme dizisi $M_{a,b}SVars$ elde edilir. Bu $M_{a,b}SVars$ dizisindeki bütün farklılıklar ikiye parçalanarak, M_aSVars ve M_bSVars eşleşme dizileri elde edilir. Bundan sonra DTTL algoritması aşağıdaki yapılarda iki tercüme kalıbını öğrenir.

- M_aSVars **if** $XE_1 \leftrightarrow XT_1$ **and** ... **and** $XE_n \leftrightarrow XT_n$
- M_bSVars **if** $XE_1 \leftrightarrow XT_1$ **and** ... **and** $XE_n \leftrightarrow XT_n$

Bunlara ek olarak ta kontrol edilmemiş son benzerlik denkliğinden, aşağıdaki sabit tercüme kalıbı öğrenilir.

- $SE_{k,n} \leftrightarrow ST_{l,n}$

Örneğin (2) deki eşleşme dizisi her iki tarafında da ikişer benzerlik içerir. Daha önceden elde edilmiş bir bilgiyi kullanmaksızın, "`for Cathy`" parçacığının Türkçe tarafında "`Cathy için`" parçacığına mı, yoksa "`+yH satın al+DH+m`" parçacığına mı denk düştüğünü bilemeyiz. Ama daha önceden "`for Cathy`" parçacığının Türkçe tarafında "`Cathy için`" parçacığına denk düştüğü öğrenilmiş ise, aşağıdaki 3 tercüme kalıbı DTTL algoritması tarafından öğrenilebilir.

- XE_1 `book` $XE_2 \leftrightarrow XT_2$ `kitap` XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
- XE_1 `ring` $XE_2 \leftrightarrow XT_2$ `yüzük` XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
- `i buy+p the` $\leftrightarrow$ `+yH satın al+DH+m`

Genelde, DTTL algoritması bir eşleşme dizisinin her iki tarafındaki benzerliklerin sayısı ayni ve bir taraftaki benzerlik sayısı 0 dan büyük ise (*n=m>0*), yeni tercüme kalıplarını sadece *n-1* benzerlik eşleşmesi daha önceden öğrenilmiş ise öğrenebilir. DTTL algoritmasın detayları Şekil 2 de verilmiştir.

---

procedure DTTL($M_{a,b}$)
begin
if $M_{a,b}$ eşleşme dizisinin iki tarafında boş olmayan benzerlik sayısı *n* ve *n≥1* **and** *n-1* tane birbirine denk düşen benzerlik $M_{a,b}$ eşleşme dizisi içinde belirlenebilir **then**

- Kabul edelim ki $(SE_{k,n},ST_{l,n})$ kontrol edilmeyen benzerlik çifti olsun.
- Kabul edelim ki $((SE_{k,1},ST_{l,1}),..., (SE_{k,n},ST_{l,n}))$ kontrol edilmemiş benzerlik çiftinide kapsayan denk düşen bütün benzerlik çiftleri olsun.
- Her bir benzerlik çifti $(SE_{k,i},ST_{l,i})$ için, $M_{a,b}$ eşleşme dizisindeki $SE_{k,i}$ yı SE_i değişkeni ile, ve $ST_{l,i}$ yide ST_i değişkeni ile değiştirerek yeni $M_{a,b}SVars$ eşleşme dizisini elde et.
- $M_{a,b}SVars$ eşleşme dizisini içindiki farklılıkları örneklere göre ikiye ayırarak M_aSVars ve M_bSVars eşleşme dizileri elde et.
- aşağıdaki tercüme kalıplarını öğren:
  - M_aSVars *if* $XE_1 \leftrightarrow XT_1$ *and ...* *and* $XE_n \leftrightarrow XT_n$
  - M_bSVars *if* $XE_1 \leftrightarrow XT_1$ *and ...* *and* $XE_n \leftrightarrow XT_n$
  - $SE_{k,n} \leftrightarrow ST_{l,n}$

end

Şekil 2. DTTL Algoritması

3.3. Eşleşme Dizisinde Farklı Sayıda Benzerlik veya Farklılık Olması

Bölüm 3.1 de verilen STTL algoritması yeni tercüme kalıplarını sadece eşleşme dizisinin her iki tarafındaki farklılık sayısı ayni olursa öğrenebilir. Benzer şekilde Bölüm 3.2 de verilen DTTL algoritması yeni tercüme kalıplarını sadece eşleşme dizisinin her iki tarafındaki benzerlik sayısı ayni olursa öğrenebilir. Bu bölümde bu sınırlamaları nasıl azalta bileceğimizi tartışacağız. Bu amaçla eğer bir eşleşme dizisindeki farklılıkların sayısı her iki tarafta da ayni değilse, STTL algoritmasını çağırmadan önce her iki tarafta ki farklılık sayılarını eşitlemeye çalışırız. Benzer şekilde DTTL algoritmasını çağırmadan önce de eşleşme dizisinin her iki tarafındaki benzerlik sayılarını eşitlemeye çalışırız.

Örneğin aşağıdaki iki tercüme örneğinin ("I came" ↔ "geldim" ve "You went ↔ gittin") eşleşme dizisinin İngilizce tarafında tek bir farklılık var, ama Türkçe tarafında iki farklılık var.

- i come+p ↔ gel+DH+m
- you go+p ↔ git+DH+n

Eşleşme Dizisi:

(i come, you go) +p ↔ (gel, git) +DH (+m, +n)

Bu eşleşme dizisinin her iki tarafında ki farklılık sayısı ayni olmadığından, Bölüm 3.1 de verilen STTL algoritması bu eşleşme dizisinden yeni tercüme kalıpları öğrenemez. Ama İngilizce tarafındaki farklılığın parçacıkların uzunluğu 1 den fazla olduğundan bu parçacıkları ikiye ayırabiliriz. Bu durumda yukarıda ki eşleşme dizisi aşağıda ki eşleşme dizisine dönüşecektir.

(i, you) (come, go) +p ↔ (gel, git) +DH (+m, +n)

Şimdi bu eşleşme dizisi her iki tarafta da ikişer farklılığa sahiptir. Eğer (i,you) nun (+m,+n) ne denk düştüğü daha önceden öğrenilmiş ise, aşağıdaki tercüme kalıpları STTL algoritması tarafından öğrenilebilir.

- XE_1 XE_2 +p ↔ XT_2 +DH XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
- come ↔ gel
- go ↔ git

Genel olarak STTL algoritmasını uygulamadan önce, bir eşleşme dizisinin her iki tarafındaki farklılık sayılarını eşitlemeye çalışırız. Bu amaçla her iki parçası da birden fazla morfem içeren bir farklılığı parçalarını ayırarak iki farklılığa dönüştürürüz. Bir farklılık (D_a,D_b) biri ($D_{a,1}$,$D_{b,1}$) ve diğeri de ($D_{a,2}$,$D_{b,2}$) olmak üzere iki ayrı farklılığa ayrılabilir. Burada D_a ve D_b nin uzunlukları 1 den fazla olmalıdır; $D_{a,1}$,$D_{b,1}$, $D_{a,2}$, ve $D_{b,2}$ nin uzunlukları da en az 1 olmalıdır; D_a= $D_{a,1}$ $D_{a,2}$ ve D_b= $D_{b,1}$ $D_{b,2}$ eşitlikleri tutmalıdır. Bu bölünme işlemi STTL algoritması tarafından yeni tercüme kalıpları öğrenilene kadar veya başka türlü bir bölünme olasalığı kalmayana kadar devam eder.

Orijinal eşleşme dizisinin her iki tarafındaki farklılık sayısı ayni olsa bile, farklılıkları ayırırak bu eşleşme dizisini yeni bir versiyonunu yaratmamız gerekebilir. Örneğin aşağıdaki tercüme kalıplarının ("I drank water"↔"su içtim" ve "You ate orange"↔"portakal yedin") eşleşme dizisinin her iki tarafında da ikişer farklılık var.

- i drink+p water ↔ su iç+DH+m
- you eat+p orange ↔ portakal ye+DH+n

Eşleşme Dizisi:

(i drink,you eat) +p (water,orange)
↔ (su iç,portakal ye) +DH (+m,+n)

Şimdi kabul edelim ki (i drink,you eat) farklılığının (su iç,portakal ye) farklılığına mı yoksa (+m,+n) farklığına mı denk düştüğünü, veya (water,orange) farklılığının (su iç,portakal ye) farklılığına mı yoksa (+m,+n) farklığına mı denk düştüğünü önceden bilmemiş olalım. Gerçekte bunların hiç birininin doğru olmaması gerekir,

çünkü hepsi de yanlış tercüme kalıplarının öğrenilmesine neden olacaklardır. Ama her iki taraftaki farklılıklarıda ikiye ayırırsak, aşağıdaki gibi bir eşleşme dizisi elde etmiş oluruz.

```
(i,you) (drink,eat) +p (water,orange)
    ↔ (su,portakal) (iç,ye) +DH(+m,+n)
```

Bu durumda her iki tarafta üçer farklılık vardır. Eğer `(i,you)` farklılğının `(+m,+n)` farklılığına ve `(water,orange)` farklılığının `(su,portakal)` farklılığına denk düştüğü daha önceden öğrenilmiş ise, bu eşleşme dizisinden STTL algoritması aşağıdaki 3 yeni tercüme kalıbını öğrenir.

- XE_1 XE_2 +p XE_3 ↔ XT_3 XT_2 +DH XT_1
  if $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$ **and** $XE_3 \leftrightarrow XT_3$
- drink ↔ iç
- eat ↔ ye

Benzer şekilde DTTL algoritmasını da çağırmadan önce eşleşme dizisinde ki benzerlik sayılarını eşitlemeye çalışırız. Uzunluğu 1 den fazla olan bir benzerlik bir tarafta benzerlik sayısını artırmak için ikiye bölüne bilinir. Benzerliklerin bölünme işlemi DTTL algoritması tarafından yeni tercüme kalıpları öğrenilene kadar, veya başka bölünme olasalığı kalmayana kadar devam eder.

Örneğin aşağıdaki tercüme örneklerinin (“I came”↔”geldim” ve “I went”↔”gittim”) eşleşme dizisinin İngilizce tarafında 2 benzerlik var, ama Türekçe tarafında tek benzerlik bulunduğundan, DTTL algoritması bu eşleşme dizisinden yeni tercüme kalıpları öğrenemez.

- i come+p ↔ gel+DH+m
- i go+p ↔ git+DH+m

Eşleşme Dizisi:

```
i (come,go) +p ↔ (gel,git) +DH+m
```

Eğer “+DH+m” benzerliğini “+DH” ve “+m” benzerliklerine bölersek, her iki tarafta ikişer benzerliğe sahip olacaktır. Eğer “i” nın “+m” morfemine denkliği daha önceden öğrenilmiş ise, aşağıdaki tercüme kalıpları DTTL algoritması tarafından öğrenilir.

- XE_1 come XE_2 ↔ gel XT_2 XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
- XE_1 go XE_2 ↔ git XT_2 XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
- +p ↔ +DH

3.4. Boş Parçacıklı Farklılıklar

Bölüm 3 nün başında bahsedilen eşleştirme algoritması, bir farklılığın boş bir parçacık içermesine izin vermez. Bu sınırlama nedeniyle bazı durumlarda eşleştirme algoritması bazı

tercüme örnek çiftleri için eşleşme dizisi bulamayabilir. Fakat bu tür tercüme örneklerinden de kullanışlı tercüme kalıpları öğrenilebilinir. Örneğin eşleştirme algoritması aşağıdaki tercüme örnekleri ("I saw a man"↔"bir adam gördüm" ve "I saw the man"↔"adamı gördüm") için eşleşme dizisi bulmayacaktır.

- i see+p the man ↔ adam+yH gör+DH+m
- i see+p a man ↔ bir adam gör+DH+m

Bunun nedeni "bir" ve "+yH" nin farklılıklar yaratılırken boş dizin ile eşleşmesi gereğidir. Eğer eşleşme algoritmasında ki bu sınırlımayı kaldırırsak aşağıdaki eşleşme dizisi eşleşme algoritmasının yeni versiyonu tarafından bulunacaktır.

i see+p (the,a) man ↔ (ε,bir) adam (+yH,ε) gör+DH+m

Bu eşleşme dizisinde (ε,bir) farklılığındaki "bir" ve (+yH,ε) farklılığındaki "+yH" boş dizinler ile eşleşmişlerdir. Bu eşleşme dizisine DTTL algoritmasını "man" in "adam" a denk düştüğünü bilerek uyguladığımızda aşağıdaki tercüme kalıpları öğrenilecektir.

- XE_1 the XE_2 ↔ XT_2 +yH XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
- XE_1 a XE_2 ↔ bir XT_2 XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
- i see+p ↔ gör+DH+m

STTL algoritmasını farklılıkları boş parçacık içeren eşleşme dizilerine uygulamayız. Bunun nedeni bir dildeki boş olmayan bir dizinin diğer bir dilde boş dizine denk düşmesi anlamına geleceğidir. Buda pek mantıklı bir yaklaşım olmayacaktır. Bu yüzden bu tür eşleşme dizilerine sadece DTTL algoritmasını uygularız.

3.5. STTL ve DTTL Algoritmalarının Performansları

STTL ve DTTL öğrenme algoritmalarının performansların test edebilmek için, bu algoritmalar Prolog programlama dili kullanılarak yazıldı ve yarattığımız tercüme örnek kümeleri üzerinde test edildi. Örnek kümeleri İngilizce ve Türkçe için bizim tarafımızdan yaratıldı. Örnek kümelerini kendimizin yaratmasının nedeni böyle örnek kümelerine doğal olarak bulamadığımızdan kaynaklanmıştır.

Öğrenme algoritmamımızın her adımınında STTL ve DTTL algoritmalarını her bir tercüme örneği çifti için (onların eşleşme dizisine) uyguladık. Eğer bir tercüme örnek kümesinde n tercüme örneği var olduğunda toplam tercüme örnek çiftlerin sayısı aşağıdaki

$$\sum_{i=1}^{n-1} i$$

formülü ile hesaplandığından, her adımın zaman ihtiyacı $O(n^2)$ ile orantılı olacaktır. Öğrenme algoritmasının adımları bir adımda hiç bir yeni tercüme kalıbı öğrenilmeyinceye kadar tekrar edilir. Diğer bir deyişle öğrenme algoritması eğer bir adımında hiç bir yeni tercüme kalıbı

öğrenmemişse, öğrenme işlemi durur. Teorik olarak öğrenme algoritmasındaki maksimum adım sayısı n-2 olmasına rağmen bizim testlerimizde maksimum adım sayısı (bir kaç yüz tercüme örnekli kümeler için) sadece 4 oldu. Buda bize testlerimizin sonuçlarına göre öğrenme algoritmasının zaman ihtiyacı (teorik olarak $O(n^3)$ olmasına rağmen) $O(n^2)$ olarak kalacağını gösterdi.

Örnek kümelerimizden bir tanesi 746 tercüme örneği içeriyordu. Bu küme ile İngilizce dilinin çok ufak bir bölümünün gramerini öğretmek için yeterli oldu. Öğrenme algoritmamızın değişik parçalarını bu örnek kümelerine uygaladık ve 167MHz Ultra Sparc işlemcisinde aşağıdaki sonuçları aldık. Bu değerlerin amaçları öğrenme algoritmasının zaman ihtiyacını ve her parçanın maliyetini belirlemekti.

1. Bu testte eşleşme dizisindeki farklılıkları bölmeyerek vede boş parçacıklı farklılıklara izin vermeyerek sadece STTL algoritmasını uyguladık. Bu durumda öğrenme algoritmasının ilk adımında STTL algoritması 641 yeni tercüme kalıbı öğrendi. İkinci adımda hiç bir yeni tercüme kalıbı öğrenilmedi. Her bir adım 21 saniye sürdü.

2. İkinci testte eşleşme dizisindeki benzerlikleri bölmeyerek vede boş parçacıklı farklılıklara izin vermeyerek sadece DTTL algoritmasını uyguladık. Bu durumda öğrenme algoritmasının ilk adımında DTTL algoritması 793 yeni tercüme kalıbı öğrendi. İkinci adımda da 6 yeni tercüme kalıbı öğrendi. Üçüncü adımda hiç bir yeni tercüme kalıbı öğrenilmedi. Her bir adım 21 saniye sürdü.

3. Bu kez eşleşme dizisindeki farklılıkları ve benzerlikleri bölmeyerek vede boş parçacıklı farklılıklara izin vermeyerek STTL ve DTTL algoritmalarını birlikte uyguladık. Öğrenme algorıtmasının ilk adımında 1217 yeni tercüme kalıbı, ikinci adımında 6 yeni tercüme kalıbı öğrenildi. Üçüncü adımda hiç bir yeni tercüme kalıbı öğrenilemedi. Her bir adım 28 saniye sürdü.

4. Dördüncü testte eşleşme dizisindeki farklılıkların ve benzerliklerin bölünmesine izin vererek vede boş parçacıklı farklılıklara izin vermeyerek STTL ve DTTL algoritmalarını birlikte uyguladık. Öğrenme algoritmasının ilk adımında 1305 yeni tercüme kalıbı, ikinci adımında 20 yeni tercüme kalıbı öğrenildi. Üçüncü adımda hiç bir yeni tercüme kalıbı öğrenilemedi. Her bir adım 32 saniye sürdü. Eşleşme dizisindeki farklılıkların ve benzerliklerin bölünmesine izin vermek öğrenme zamanını yüzde 14 artırtırdı, karşılığında yüzde 8 daha fazla yeni tercüme kalıbı öğrenildi.

5. Beşinci testte eşleşme dizisindeki farklılıkların ve benzerliklerin bölünmesine vede boş parçacıklı farklılıklara izin vererek STTL ve DTTL algoritmalarını birlikte uyguladık. Öğrenme algoritmasının ilk adımında 2022 yeni tercüme kalıbı, ikinci adımında 63 yeni tercüme kalıbı öğrenildi. Üçüncü adımda hiç bir yeni tercüme kalıbı öğrenilemedi. Her bir adım 56 saniye sürdü. Böylece eşleşme dizisinde boş parçacıklı farklılıklara izin vermek öğrenme zamanını yüzde 75 artırtırdı, karşılığında yüzde 57 daha fazla yeni tercüme kalıbı öğrenildi.

4. Öğrenme Örnekleri

Bu bölümde öğrenme algoritmaların tercüme örnek kümelerinden alınan parçacıklar üzerinde nasıl davrandığını göstereceğiz.

Örnek 1:

Eğer verilen tercüme örnekleri aşağıdaki gibi ise

- I came ↔ Geldim
- You came ↔ Geldin
- I went ↔ Gittim
- You went ↔ Gittin

ve bunların şekilsel gösterimi aşağıdaki gibidir.

- i come+p ↔ gel+DH+m
- you come+p ↔ gel+DH+n
- i go+p ↔ git+DH+m
- you go+p ↔ git+DH+n

Birinci ve ikinci örnekten STTL algoritması aşağıdaki ilk üç tercüme kalıbını, DTTL algoritması son üç tercüme kalıbını öğrenir.

1. XE_1 come+p ↔ gel+DH XT_1 **if** $XE_1 \leftrightarrow XT_1$
2. i ↔ +m
3. you ↔ +n
4. i XE_1 ↔ XT_1 +m **if** $XE_1 \leftrightarrow XT_1$
5. you XE_1 ↔ XT_1 +n **if** $XE_1 \leftrightarrow XT_1$
6. come+p ↔ gel+DH

Birinci ve üçüncü örnekten STTL algoritması aşağıdaki ilk üç tercüme kalıbını, DTTL algoritması "+DH+m" benzerliğini "+DH" ve "+m" benzirliklerine ayrılmasından vede "i↔+m" denkliğinin daha önceden öğrenilmiş olmasından dolayı son üç tercüme kalıbını öğrenir.

7. i XE_1 +p ↔ XT_1 +DH+m **if** $XE_1 \leftrightarrow XT_1$
8. come ↔ gel
9. go ↔ git
10. XE_1 come XE_2 ↔ gel XT_2 XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
11. XE_1 go XE_2 ↔ git XT_2 XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
12. +p ↔ +DH

Birinci ve dördüncü örnekten STTL algoritması (i come,you go) farklılığının (i,you) ve (come,go) farklılıklarına ayrılabilecek olmasından ve "i↔+m" ve "you↔+n"

denkliklerinin daha önceden öğrenilmiş olmasından dolayı aşağıdaki ilk üç yeni tercüme kalıbının birincisini öğrenir (gerçekte üç tercüme kalıbı öğrenir, ama bunlardan iki tanesi daha önceden öğrenilmişti). DTTL algoritması son iki yeni tercüme kalıbını öğrenir (gerçekte üç tercüme kalıbı öğrenir, ama bunlardan bir tanesi daha önceden öğrenilmişti).

13. XE_1 XE_2 +p ↔ XT_2 +DH XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
14. i come XE_1 ↔ gel XT_1 +m **if** $XE_1 \leftrightarrow XT_1$
15. you go XE_1 ↔ git XT_1 +n **if** $XE_1 \leftrightarrow XT_1$

İkinci ve üçüncü örnekten STTL algoritması yeni tercüme kalıpları öğrenmez, DTTL algoritması aşağıdaki iki yeni tercüme kalıbını öğrenir.

16. you come XE_1 ↔ gel XT_1 +n **if** $XE_1 \leftrightarrow XT_1$
17. i go XE_1 ↔ git XT_1 +m **if** $XE_1 \leftrightarrow XT_1$

İkinci ve dördüncü örnekten STTL aşağıdaki yeni tercüme kalıbını öğrenir, DTTL algoritması yeni tercüme kalıbı öğrenemez.

18. you XE_1 +p ↔ XT_1 +DH +n **if** $XE_1 \leftrightarrow XT_1$

Üçüncü ve dördüncü örnekten STTL algoritması aşağıdaki ilk yeni tercüme kalıbını, DTTL algoritması sonraki yeni tercüme kalıbını öğrenir.

19. XE_1 go +p ↔ git +DH XT_1 **if** $XE_1 \leftrightarrow XT_1$
20. go +p ↔ git +DH

Böylece 4 basit tercüme örneğinden 20 yeni tercüme kalıbı öğrenilir. Kalıpların bazıları birden fazla kere öğrenilir.

Örnek 2:

Eğer verilen tercüme örnekleri aşağıdaki gibi ise

- 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

ve bunların şekilsel gösterimi aşağıdaki gibidir.

- red apple ↔ kırmızı elma
- green apple ↔ yeşil elma
- We eat+p a pear ↔ Bir armut ye+DH+k

- We eat+p a banana ↔ Bir muz ye+DH+k
- They eat+p a pear ↔ Bir armut ye+DH+lAr
- They eat+p a banana ↔ Bir muz ye+DH+lAr

Birinci ve ikinci tercüme örneklerinden STTL algoritması aşağıdaki ilk üç tercüme kalıbını ve DTTL algoritması son üç kalıbı öğrenir.

1. XE_1 apple ↔ XT_1 elma **if** $XE_1 \leftrightarrow XT_1$
2. red ↔ kırmızı
3. green ↔ yeşil
4. red XE_1 ↔ kırmızı XT_1 **if** $XE_1 \leftrightarrow XT_1$
5. green XE_1 ↔ yeşil XT_1 **if** $XE_1 \leftrightarrow XT_1$
6. apple ↔ elma

Üçüncü ve dördüncü tercüme örneklerinden STTL algoritması aşağıdaki üç tercüme kalıbını öğrenir ve DTTL algoritması yeni tercüme kalıbı öğrenemez.

7. we eat+p a XE_1 ↔ bir XT_1 ye+DH+k **if** $XE_1 \leftrightarrow XT_1$
8. pear ↔ armut
9. banana ↔ muz

Üçüncü ve dördüncü tercüme örneklerinden STTL algoritması aşağıdaki ilk üç tercüme kalıbını ve DTTL algoritması son üç kalıbı öğrenir.

10. XE_1 eat+p a pear ↔ bir armut ye+DH XT_1 **if** $XE_1 \leftrightarrow XT_1$
11. we ↔ +k
12. they ↔ +lAr
13. we XE_1 ↔ XT_1 +k **if** $XE_1 \leftrightarrow XT_1$
14. they XE_1 ↔ XT_1 +lAr **if** $XE_1 \leftrightarrow XT_1$
15. eat+p a pear ↔ bir armut ye+DH

Üçüncü ve altıncı tercüme örneklerinden STTL algoritması aşağıdaki yeni tercüme kalıbını öğrenir.

16. XE_1 eat+p a XE_2 ↔ bir XT_2 ye+DH XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$

Dördüncü ve altıncı tercüme örneklerinden STTL algoritması aşağıdaki ilk yeni tercüme kalıbını, ve DTTL algoritmasi son yeni tercüme kalıbını öğrenir.

17. XE_1 eat+p a banana ↔ bir muz ye+DH XT_1 **if** $XE_1 \leftrightarrow XT_1$
18. eat+p a banana ↔ bir muz ye+DH

Beşinci ve altıncı tercüme örneklerinden STTL algoritması aşağıdaki yeni tercüme kalıbını öğrenir.

19. they eat+p a XE_1 ↔ bir XT_1 ye+DH+lAr **if** $XE_1 \leftrightarrow XT_1$

Örnek 3:

Eğer verilen tercüme örnekleri aşağıdaki gibi ise

- 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

ve bunların şekilsel gösterimi aşağıdaki gibidir.

- he always wash+s his face ↔ her zaman yüz+Hn+yH yıka+Hr
- i watch+p tv ↔ televizyon seyret+DH+m
- he always wash+s his face after he wake+s up
  ↔ kalk+DHk+tAn sonra her zaman yüz+Hn+yH yıka+Hr
- i watch+p tv after i eat+p the dinner
  ↔ akşam yemek+sH+nH ye+DHk+tAn sonra televizyon seyret+DH+m

STTL algoritması ilk tercüme örneğin yardımı ile üçüncü ve dördüncü örnekler aşağıdaki ilk üç yeni tercüme kalıbını öğrenir. DTTL algoritması da sonarki üç tercüme kalıbını öğrenir.

1. XE_1 after XE_2 ↔ XT_2 +DHk +tAn sonra XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$
2. he wake+s up ↔ kalk
3. i eat+p the dinner ↔ akşam yemek+sH+nH ye
4. he always wash+s his face XE_1 he wake+s up
   ↔ kalk XT_1 sonra her zaman yüz+Hn+yH yıka+Hr **if** $XE_1 \leftrightarrow XT_1$
5. i watch+p tv XE_1 i eat+p the dinner
   ↔ akşam yemek+sH+nH ye XT_1 sonra televizyon seyret+DH+m
   if $XE_1 \leftrightarrow XT_1$
6. after ↔ DHk+tAn sonra

5. Tercüme İşlemi

TTL algoritmaları tarafından öğrenilen tercüme örnekleri doğrudan tercüme işleminde iki yönlü olarak kullanılır. Bilindiği gibi bu tercüme örneklerinde kelimelerin şekilsel yapısı kullanılmıştır. Bu yüzden tercüme anında her iki dil için morfolojik işlemcileri kullanmamız gerekir. Tercüme edilecek bir cümlenin kelimeleri ilk önce o dilin morfolojik işlemcisi tarafından şekilsel yapısına dönüştürülür. Daha sonra bu şekilsel yapıdaki cümle öğrenilen tercüme kalıpları kullanılarak tercüme edilir. Elde edilen cümlede ikinci dilin şekilsel yapısındadır. Bu elde edilen cümle ikinci dilin morfolojik işlemcisi tarafından normal haline

getirilir. Bu tercüme işleminden elde edeceğimiz sonuştur. Tercüme işlemini de kapsayan genel sistem yapısı Şekil 3 de verilmiştir.

Tercüme işlevi şöyle özetlenebilinir:

1. İlk önce tercüme için verilen cümledeki kelimeler o dilin morfolojik işlemcisi tarafından şekilsel haline çevrilir. Böylece verilen cümlenin şekilsel yapısı bulunmuş olur.
2. Cümlenin bu şekilsel yapısını ve öğrenilmiş olan tercüme kalıpları kullanılarak bu cümlenin tercümesi bulunur. Bu cümlenin birden fazla tercümesi olabilir.
3. Son olarak bulunan tercüme sonuşları, o dilin morfolojik işlemcisi tarafından normal haline çevrilir.

Tercüme işlevini birde bir örnek üzerinde açıklayalım. Kabul edelim ki Örnek 1 ve 2 deki tercüme örnekleri öğrenildikten sonra, "bir kırmızı elma yedim" cümlesinin İngilizceye nasıl çevrildiğine bakalım. İlk önce bu cümlenin şekilsel yapısını ifade eden "bir kırmızı elma ye+DH+m" Türkçe morfolojik işlemcisi tarafından bulunur. Sonra bu şekilsel yapı ile uyuşabilecek tercüme kalıpları aranır. Bu durumda sadece aşağıdaki tercüme kalıbı bu cümle ile uyuşabilecektir.

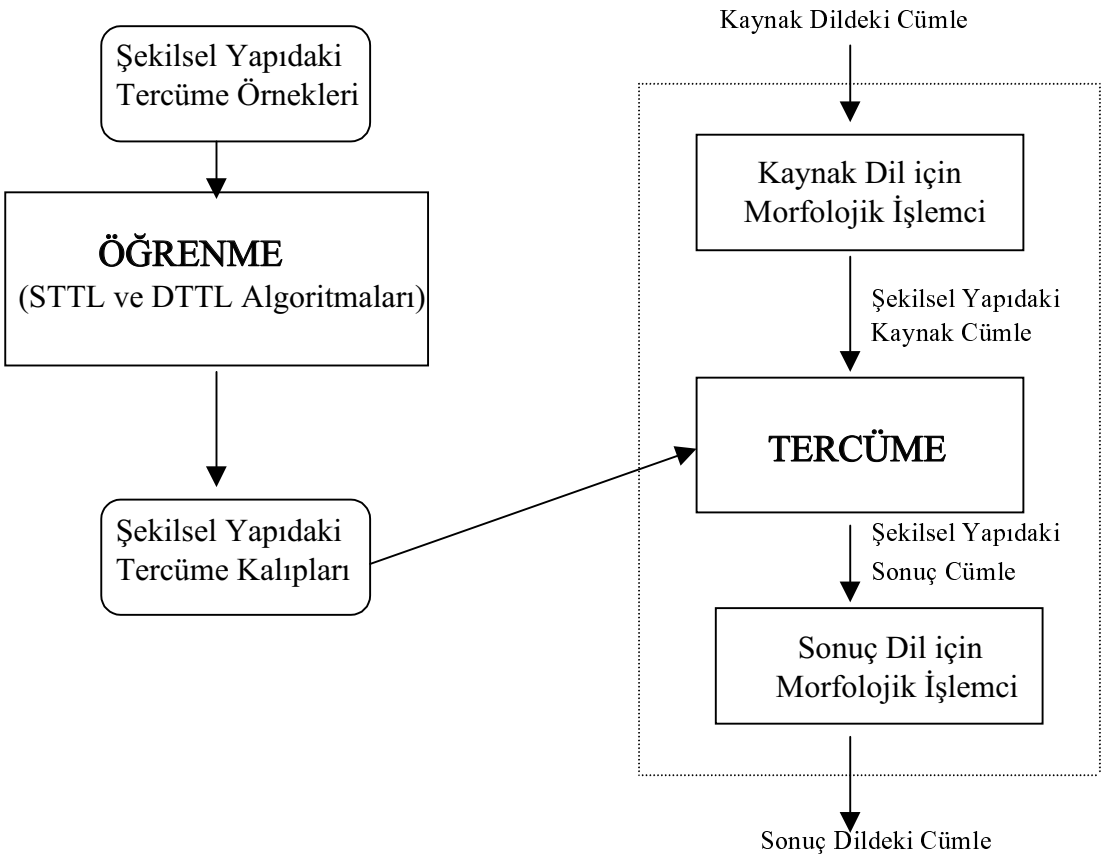
XE_1 eat+p a XE_2 ↔ bir XT_2 ye+DH XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$

Bu uyuşma anında XT_1 değişkeni "+m" parçacığına, XT_2 değişkeni "kırmızı elma" parçacığına bağlanacaktır. Daha sonra bu parçacıkların tercümeleri aranır. "+m" nin tercümesinin "i" ve "kırmızı elma" nın tercümesinin "red apple" olduğu aşağıdaki tercüme kalıpları yardımı ile bulunur.

```
i ↔ +m
red apple ↔ kırmızı elma
```

Bulunan "i" ve "kırmızı elma" parçacıkları XE_1 ve XE_2 değişkenlerinin değerleri olacaktır. Böylece "i eat+p a red apple" sonucu bulunur. Buda İngilizce morfolojik üreticisi tarafından "i ate a red apple" cümlesine çevrilir.

Bir cümle için birden fazla sonuç bulanabilinir. Bu sonuçlar doğru tercümeye ilavaten, doğru tercümeyle eş anlamlı sonuçları ve yanlış sonuçlarıda kapsayabilir. Verilen sonuçlar arasından bir uzman doğru sonucu seçecektir. Bizim burdaki amacımız verilen sonuçların ilk kısmında doğru yanıtın bulunmasıdır. Böylece uzman sadece birkaç sonuca bakarak doğru yanıtı seçebilsin. Bunu sağlayabilmek için öğrenilmiş tercüme kalıplarını içindeki gerçek sembol sayılarına göre sıralanmıştır. Bu bize bir cümleye birden fazla uygulanabilinir tercüme kalıbı var ise bunlar dan en fazla gerçek sembol taşıyan önce ugulanır ve sonuçları sonuç listesini üst taraflarında olur. Bu yöntem doğru yanıtın üst sıralarda yer almasına yardımcı olsa da, tam bir sonuç almamızı sağlamaz. Bu amaçla bir sonraki bölümde de anlatacağımız istatiksel bir metot kullanılmıştır. Bu istatiksel yöntemle doğru yanıtın üst sıralarda yer alması biraz daha olası hale getirilmiştir.



Şekil 3. Sistemin Genel Yapısı

6. Güvenirlik Faktörlerinin Verilmesi

Algoritmamızın bir önceki versiyonunda öğrenilen tercüme kalıpları sadece kalıplar içindeki gerçek sembol sayısına göre sıralanmıştı. Fakat tercüme işleminin daha iyi çalışabilmesi için istatiksel yöntemler kullanmaya karar verdik. Benzer istatiksel yöntemler diğer tercüme sistemlerinde de kullanılmaktadır. Bu yüzden temel öğrenme algoritmamız tercüme kalıplarını öğrendikten sonra, sistemimiz öğrenilen kalıplara güvenirlik faktörleri verme safhasına başlar. Bu safhada kalıplara ve kalıp kombinasyonlarına güvenirlik faktörleri verilir.

Bizim tercüme sistemi iki yönlü çalışabilir. Diğer bir deyişle, L_1 dilindeki bir cümle L_2 dilindeki bir cümleye tercüme edilebilir, veya L_2 dilindeki bir cümle L_1 dilindeki bir cümleye tercüme edilebilir. Algoritmamızın öğrenmiş olduğu tercüme kalıpları iki yön içinde kullanılabilir. Tercüme kalıplarına verilecek olan güvenirlik faktörleri her iki yön için ayrı ayrı verilecektir. Aynı tartışma tercüme kalıp kombinasyonları içinde geçerlidir.

Öğrenilen bir tercüme kalıbı ya içinde hiç bir değişiken bulunmayan sabit bir kalıptır (fact) veya içinde değişkenler bulunan genel bir kalıptır (rule). Aşağıdakiler bunlara birer örnektir.

```
you  ↔  +n                                  sabit kalıp (fact)
you XE +p ↔ XT +DH +n if XE↔XT              genel kalıp (rule)
```

Bölümün gerisinde sabit kalıplara, genel kalıplara ve kalıp kombinasyonlarına nasıl güvenirlik faktörlerinin verildiği anlatılacaktır.

6.1. Sabit Kalıplara Güvenirlik Faktörlerinin Verilmesi

Kabul edelim ki tercüme örneklerimiz *trainpair*(X_i, Y_i) yapısında ve sabit kalıbımızda *k* olsun. Burada sabit kalıbımız $X \leftrightarrow Y$ yapısında olacaktır. Burada X ve Y morfem dizisidir. Bundan sonra yapacağımız iş, X ve Y nin hangi tercüme örneklerinin bir parçasını oluşturduğunu bulmaya kalır. Bu amaçla aşağıdaki iki sayıyı bulmamız gerekir.

N1: X in X_i nin, ve Y nin Y_i nin parçası olarak gözüktüğü tercüme örneklerinin sayısı. Diğer bir deyişle hem X in hem de Y nin beraber göründüğü bütün tercüme örneklerini sayısını buluyoruz.

N2: X in X_i nin parçası fakat Y nin Y_i parçası olmadığı tercüme örneklerin sayısı. Diğer bir deyişle, sadece X in göründüğü (Y nin görünmediği) tercüme kalıplarının sayısını buluruz.

Sabit kalıbımızın soldan sağa tercüme amacıyla kullanılması durumundaki güvenirlik faktörü aşağıdaki gibi hesaplanacaktır. Yani X in Y olarak tercüme edilmesinin güvenirlik faktörü.

$$\text{güvenirlikfaktörü}_k = N1 / (N1+N2)$$

Bu kalıba sağdan sola güvenirlik faktörü verirken yapacağımız tek şey N2 nin tanımını değiştirmektir. Bu durumda N2 nin tanımı şöyle olacaktır.

N2: Y nin Y_i nin parçası fakat X in X_i parçası olmadığı tercüme örneklerin sayısı. Diğer bir deyişle, sadece Y nin göründüğü (X in görünmediği) tercüme kalıplarının sayısını buluruz.

Sabit kalıplara güvenirlik faktörlerinin nasıl verildiğini bir örnekle gösterelim. Kabul edelim ki örnek kümemiz sadece aşağıdaki 4 tercüme örneğininden oluşsun.

```
you come+p ↔ gel+DH+n
you go+p   ↔ git+DH+n
you come+p ↔ gel+DH+nHz
you go+p   ↔ git+DH+nHz
```

Güvenirlik faktörü vermek istediğimiz sabit kalıpta

```
you ↔ +n
```

olsun. Bu kalıp yukarıdaki örnek kümesinden öğrenilecektir. Eğer bu kalıbın soldan sağa kullanımı (you → +n) için güvenirlik faktörü şöyle hesaplanacaktır.

- you ve +n 1. ve 2. örneklerde beraber gözüktüğünden N1 nin değeri 2 olacaktır.
- ve 4. örneklerde you gözükmesine rağmen +n gözükmemektedir. Böylece N2 nin değeride 2 olacaktır.
- Soldan sağa güvenirlik faktörü = N1/(N1+N2) = 2/(2+2) = 0.5

Bu örneklere göre, you nun +n olarak tercüme edilmelisinin güvenirliği yüzde 50 dir.

Bu kalıbın sağdan sola kullanımı (you ← +n) için güvenirlik faktörüde aşağıdaki gibi hesaplanacaktır.

- you ve +n 1. ve 2. örneklerde beraber gözüktüğünden N1 nin değeri yine 2 olacaktır.
- +n gözüküpte you nun gözükmediği bir örnek yok. Böylece N2 nin değeride 0 olacaktır.
- Sağdan sola güvenirlik faktörü = N1/(N1+N2) = 2/(2+0) = 1.0

Bu örneklere göre, +n in you olarak tercüme edilmelisinin güvenirliği yüzde 100 dür.

Bir sabit kalıp için, soldan sağa ve sağdan sola kullanımı için aynı güvenirlik değere alabilecek olabilmelerine rağmen, farklı değer almaları daha muhtemeldir.

6.2. Genel Kalıplara Güvenirlik Faktörlerin Verilmesi

Genel kalıplara vereceğimiz güvenirlik faktörü gerçekten o kalıba verilen kısmi güvenirlik faktörüdür. Bunun nedeni genel kalıplar içlerinde değişkenler içerir ve bu değişkenler tercüme anında başka morfem dizilerine bağlanacaktır. Böylece bir sol taraftaki değişken bir morfem dizisine ve bu değişkene karşılık gelen değişken ise başka bir morfem dizisine bağlanacaktır. Bu bağlanma işleminin de bir güvenirlik faktörü olacaktır. Bu durumda o genel tercüme kalıbının gerçek güvenirlik faktörü o kalıba verilen kısmi güvenirlik faktörünün, değişkenlerin bağlanma güvenirlik faktörleri ile çarpılması ile bulunacaktır. Genel kalıplara verilen kısmi güvenirlik faktörleri sabit kalıplara verilen güvenirlik faktörlerine benzer bir şekilde verilecektir.

Kabul edelim ki bir önceki bölümdeki 4 tercüme örneği örnek kümemizi oluşturuyor olsun ve kısmi güvenirlik faktörü vermek istediğimiz genel kalıpta bu örneklerden öğrenilen

you *XE* +p ↔ *XT* +DH+n **if** *XE*↔*XT*

kalıbı olsun. Kalıbının İngilizce kısmınının (you *XE* +p) hangi örneklerin parçası olduğu araştırılırken, değişken *XE* bir veya daha fazla morfeme denk gelebilir. Bu durumda "you

XE +p″ bütün örneklerin, "*XT* +DH+n″ de 1. ve 2. örneklerin parçası olabilir. Yukarıdaki örneğe soldan sağa kısmi güvenirlik faktörü atarken, aşağıdaki değerler bulunacaktır.

- "you *XE* +p″ ve "*XT* +DH+n" 1. ve 2. örneklerde beraber gözüktüğünden N1 nin değeri 2 olacaktır.
- 3. ve 4. örneklerde "you *XE* +p″ gözükmesine rağmen "*XT* +DH+n″ gözükmemektedir. Böylece N2 nin değeride 2 olacaktır.
- Soldan sağa kısmi güvenirlik faktörü = N1/(N1+N2) = 2/(2+2) = 0.5

Benzer şekilde sağdan sola kısmi güvenirlik faktörünü verilirken aşağıdaki değerler bulunacaktır.

- "you *XE* +p″ ve "*XT* +DH+n″ 1. ve 2. örneklerde beraber gözüktüğünden N1 nin değeri yine 2 olacaktır.
- "*XT* +DH+n″ gözüküpte "you *XE* +p″ nun gözükmediği bir örnek yok. Böylece N2 nin değeride 0 olacaktır.
- Sağdan sola kısmi güvenirlik faktörü = N1/(N1+N2) = 2/(2+0) = 1.0

6.3. Kalıp Kombinasyonlarına Güvenirlik Faktörlerinin Verilmesi

Bazı kalıplara yüksek güvenirlik faktörleri verilmiş olmasına rağmen, o kalıpları belirli kalıplar ile beraber kullanmak pek uygun olmayabilir. Bu durumda o kalıp kombinasyonlarının güvenirliği düşük olmalıdır. Bu yüzden tercüme örneklerindeki cümleleri öğrendimiz kalıplarla tercüme ederiz ve bulunan her yanıt bir kalıp kombinasyonuna denk gelir. Yanıtın doğruluk derecesine göre kalıp kombinasyonlarına bir güvenirlik faktörü verilir. Bir yanıtın doğruluk derecesi hesaplanırken, o yanıtın doğru yanıtlardan ne kadar farklı olduğunu gösteren değerler kullanılır. Böylece tercüme örneklerindeki cümleri kullanırken kullanılan bütün kalıp kombinasyonlarına da bir güvenirlik faktörü verilmiş olur. Bu algoritmanın detayları Şekil 4 de bulunabilir. Bu algoritma İngilizceden Türkçe yönü için verilmiştir. Bu algoritma benzer şekilde Türkçe den İngilizce yönü içinde uygulanır. Bölümün geri kalan kısmında tartışmalar İngilizce den Türkçe ye yönü için verilmiştir, ama benzer tartışmalar Türkçe den İngilizce yönü içinde yapılabilinir.

Şekil 4 te verilen alagoritmanın 3. adımında güvenirlik faktörünün hesaplamısını biraz daha açıklamak gerekecektir. Bu adımda her bir $ÖT_j$ ($ÖT_j \in$ ÖTS) ile T_i arasındaki *uzaklık* bulunur. Burada $ÖT_j$ ve T_i ikili düzlemde iki nokta olarak görülür. $ÖT_j$ nin bu ikili düzlemdeki yeri ($ÖT_j$ nin uzunluğu,0) noktası ile, ve T_i nin bu ikili düzlemdeki yeri ($ÖT_j$ ile T_i arasındaki benzerliklerin uzunluğu, $ÖT_j$ ile T_i arasındaki farklılıkların uzunluğu) noktası ile ifade edilir. Bu iki nokta arasındaki uzaklık (*dist*) aşağıdaki Euclid formülü ile hesaplanır.

$$dist = \sqrt{(ÖT_jUzun - BenzerUzun)^2 + (FarkUzun)^2}$$

Burada $ÖT_jUzun$ $ÖT_j$ nin uzunluğunu (içindeki morfem sayısını), *BenzerUzun* $ÖT_j$ ile T_i arasındaki benzerliklerin toplam uzunluğunu, ve *FarkUzun* $ÖT_j$ ile T_i arasındaki farklılıkların

toplam uzunluğunu gösterir. Bir farklılığın uzunluğu o farklılıktaki iki parçacığın en uzununun uzunluğudur.

- Tercüme örnek kümesindeki her bir tercüme örneği ÖE↔ÖT için aşağıdakilerini yap:

  1. Sol tarafı ÖE ye eşit olan bütün tercüme örneklerini bulalım ve sağ taraftaki parçacıkların kümesini ÖTS olarak adlandıralım. Böylece ÖTS kümesi tercüme örnek kümemizde bulunan ve ÖE denk düşen bütün Türkçe cümleleri içerecektir.

  2. Öğrenilmiş tercüme kalıplarını kullanarak ÖE nin bütün Türkçe tercümelerini (TS) ve bu tercümelerin ispatlarını (proofs) gösteren bütün ispatları (PS) ile birlikte bul. TS kümesindeki her bir T_i ÖE nin tercüme kalıpları kullanılarak bulunan bir Türkçe tercümesini, ve P_i bu tercümenin hangi tercüme kalıp kombinasyonları kullanılarak bulunduğunu gösterecektir.

  3. TS kümesindeki bir Türkçe tercüme T_i eğer ÖTS kümesinde gözüküyor ise ($T_i \in$ ÖTS) kalıp kombinasyonu P_i nin güvenirliği 1 olacaktır ($güvenirlikfaktör_{Pi} = 1$).
     Aksi halde ($T_i \notin$ ÖTS) P_i kalıp kombinasyonunun güvenirliği aşağıdaki gibi hesaplanır.
     a) Her bir $ÖT_j$ ($ÖT_j \in$ ÖTS) ile T_i arasındaki *uzaklığı* bul.
     b) Bulunan bu uzaklıklar arasından en küçüğünü (*mindist*) seç.
     c) Kalıp kombinasyonu P_i nin güvenirliğini $güvenirlikfaktör_{Pi} = 1/(1+mindist)$ formülü ile hesaplanır.

Şekil 4. Kalıp Kombinasyonlarına Güvenirlik Faktörlerinin Verilmesi

Bu algoritmanın çalışmasını bir örnekle açıklayalım. Kabul edelim ki

`you come+p ↔ gel+DH+n`

tercüme örneğinin, örnek kümedeki "`you come+p`" karşılığı olan Türkçe cümleleri gösteren küme aşağıdaki gibi olsun.

ÖTS = { `gel+DH+n, gel+DH+nHz`}

Öğrenilen tercüme kalıpları kullanılarak "`you come+p`" nın tercümesi olarak elde edilen Türkçe cümleleri gösteren küme de aşağıdaki gibi olsun. Burada PS de bu yanıtları elde ederken kullanılan kalıp kombinasyonlarını gösterecektir.

TS = { `gel+DH+n, gel+DH+nHz, gel+Hr+DH+n, siz gel+DH+nHz`}
PS = { $P_{1,1}..P_{1,n1}$, $P_{2,1}..P_{2,n2}$, $P_{3,1}..P_{3,n3}$, $P_{4,1}..P_{4,n4}$ }

Bu durumda PS işindeki her kalıp kombinasyonuna (TS deki tercümelerin bulunmasında kullanılan kalıp kombinasyonları) verilecek güvenirlik faktörü aşağıdaki gibi hesaplanacaktır.

1. T_1= `gel+DH+n` ve $P_1 = P_{1,1}..P_{1,n1}$
   Burada $T_1 \in$ ÖTS olduğundan, $güvenirlikfaktör_{P1}=1$ olacaktır.

2. T_2= `gel+DH+nHz` ve $P_2 = P_{2,1}..P_{2,n2}$
   Burada $T_2 \in$ ÖTS olduğundan, $güvenirlikfaktör_{P2}=1$ olacaktır.

3. T_3= `gel+Hr+DH+n` ve $P_3 = P_{3,1}..P_{3,n3}$
   Burada $T_3 \notin$ ÖTS olduğundan, ilk önce T_3 nin ÖTS deki bütün elamanlar ile arasındaki uzaklıklar bulunur, ve bu uzaklıkların en küçüğü P_3 nin güvenirlik faktörü olacaktır.

   - T_3 ile $ÖT_1$ (ÖTS nin ilk elemanı – "`gel+DH+n`") arasındaki uzaklığın ($dist_1$) bulunması.
     - T_3 ile $ÖT_1$ arasındaki benzerlik ve farklılıkların bulunması:
       `gel(+Hr,ε)+DH+n`
     - T_3 ile $ÖT_1$ arasındaki bütün benzerliklerin toplam uzunluğu 3 dir.
     - T_3 ile $ÖT_1$ arasındaki farklılıkların toplam uzunluğu 1 dir.
     - Böylece $dist_1 = \sqrt{(3-3)^2 + 1^2} = 1$

   - T_3 ile $ÖT_2$ ("`gel+DH+nHz`") arasındaki uzaklığın ($dist_2$) bulunması.
     - T_3 ile $ÖT_2$ arasındaki benzerlik ve farklılıkların bulunması:
       `gel(+Hr,ε)+DH(+n,+nHz)`
     - T_3 ile $ÖT_2$ arasındaki bütün benzerliklerin toplam uzunluğu 2 dir.
     - T_3 ile $ÖT_2$ arasındaki farklılıkların toplam uzunluğu 1+1=2 dir.
     - Böylece $dist_2 = \sqrt{(3-2)^2 + 2^2} = \sqrt{5}$

   - $dist = min(dist_1, dist_2) = 1$

   - P_3 nin güvenirlik faktörü: $güvenirlifaktör_{P3} = 1/(1+dist) = 0.5$

4. T_4= `siz gel+DH+nHz` ve $P_4 = P_{4,1}..P_{4,n4}$
   Burada $T_4 \notin$ ÖTS olduğundan, ilk önce T_4 nin ÖTS deki bütün elamanlar ile arasındaki uzaklıklar bulunur, ve bu uzaklıkların en küçüğü P_4 nin güvenirlik faktörü olacaktır.

   - T_4 ile $ÖT_1$ ("`gel+DH+n`") arasındaki uzaklığın ($dist_1$) bulunması.
     - T_4 ile $ÖT_1$ arasındaki benzerlik ve farklılıkların bulunması:
       `(siz,ε)gel+DH(+nHz,+n)`
     - T_4 ile $ÖT_1$ arasındaki bütün benzerliklerin toplam uzunluğu 2 dir.
     - T_4 ile $ÖT_1$ arasındaki farklılıkların toplam uzunluğu 2 dir.
     - Böylece $dist_1 = \sqrt{(3-2)^2 + 2^2} = \sqrt{5}$

   - T_4 ile $ÖT_2$ ("`gel+DH+nHz`") arasındaki uzaklığın ($dist_2$) bulunması.
     - T_4 ile $ÖT_2$ arasındaki benzerlik ve farklılıkların bulunması:
       `(siz,ε)gel+DH+nHz`
     - T_4 ile $ÖT_2$ arasındaki bütün benzerliklerin toplam uzunluğu 3 dir.
     - T_4 ile $ÖT_2$ arasındaki farklılıkların toplam uzunluğu 1 dir.
     - Böylece $dist_2 = \sqrt{(3-3)^2 + 1^2} = 1$

- $dist = min(dist_1, dist_2) = 1$

- P_4 nin güvenirlik faktörü: $güvenirlifaktör_{P4} = 1/(1+dist) = 0.5$

Kalıp kombinasyonları bir ağaç yapısındadır. Örneğin tercüme kalıbı $kalıp_i$ iki tane değişken içeriyorsa, ve bu değişkenler $kalıp_j$ and $kalıp_k$ ile bağlanmış iseler, $kalıp_i$ ağacın kökü $kalıp_j$ ve $kalıp_k$ bu ağacın dalları olacaklardır. Bu ağaçlar dalları tekrarlanarak (recursive) yaratılır.

6.4. Güvenirlik Faktörleri ile Tercüme İşlemi

Bir cümle tercüme edilmek üzere verildiğinde, o cümlenin bütün tercümeleri kullanılan kalıplarla birlikte daha önceden öğrenilen tercüme kalıpları kullanılarak bulunur. Böylece elimizde bir çok yanıt ve her bir yanıt içinde bir kalıp kombinasyonu olacaktır. Bundan sonraki işlem her yanıtın güvenirliğini bulmaktır.

Bir kalıp kombinasyonun güvenirliğini bulurken, ilk önce o kombinasyona bir güvenirlik faktörü verilip verilmediğine bakılır. Eğer verilmişse o değer kullanılır. Aksi halde o kombinasyon kalıplardan oluştuğundan, o kombinasyonun güvenirlik faktörü kalıpların güvenirlik faktörleri kullanılarak hesaplanır. Böylece bulunan yanıtlar güvenirlik faktörlerine göre bir sıraya konmuş olur. En yüksek değerli yanıtlar sonuç olarak kullanıcaya geri döndürülecek olan yanıtlardır.

Güvenirlik faktörlerinin tercüme işleminde doğru yanıtın üst taraflarda çıkmasına yardım ettiğini bulmak için bir seri testler yaptık. Bu testlerin sonucana göre tercüme işlemin doğru yanıtın üst sıralarda çıkması, güvenirlik faktörlerini kullanıdığımızda yüzde 50 artı. Güvenirlik faktörleri kullanıldığında verilen ilk 5 yanıt içersinde doğru yanıtın olma olasalığı yüzde 60 olarak gerçekleşti.

4. Sonuç

Bu proje kapsamında geliştirilen tercüme sistemin temel yapısı tercüme kalıplarının iki dil arasında verilen tercüme örneklerinden öğrenilmesi üzerine dayanır. Bu temel yapı örneğe-dayalı bilgisayar ile tercüme sistemi olarak gerçekleştirilmiştir. Geliştirilen sistem İngilizce ve Türkçe arasında iki yönlü çalışabilecek şekilde geliştirilmiştir. Geliştirilen teknikler sadece İngilizce ve Türkçe arasında uygulandıysa da, diğer doğal diller içinde uygulanabileciğine inanıyoruz. Bu proje kapsamında olmasada ufak çaplı olarak İngilizce ve Fransızca arasında da algoritmalarımızı denedik, ve benzer sonuçlar aldık. Diğer diller arasında daha kapsamlı olarak daha sonra çalışmalar yapmayı planlıyoruz.

Bu proje başlamadan önceki temel motivasyonumuz insanların öğrenme mekanizmasına benzer bir öğrenme sistemi, yani örneğe dayalı bir sistem geliştirme isteğimizdi. İnanıyoruz ki insanlarda bir dil öğrenirken o dildeki yapıları örnek cümleler üzerinden belirli yapıları öğrenirler. Diğer bir deyişle insanlarda cümlelerdeki benzerlik ve farklılıkları kullanarak belirli kalıpları öğrenirler. Bu gözlem bizi bu projede yaptığımız türde bir sistem üzerinde çalışmaya itmiştir.

Öğrenme algoritmalarımız kelimelerin normal halini (surface level) kullanan cümleler üzerinde de doğrudan çalışabilir. Ama öğrenilen tercüme kalıpları fazla kullanışlı olmayabilir. Bu yüzden kelimelerin yapısal halini (lexical level) kullandık. Tabi bu durumda iki dil içinde morfolojik işlemcilere gerek duyduk. Genelde birçok dil için piyasada morfolojik işlemci mevcuttur. En aşağı bizim kulandığımız diller İngilizce ve Türkçe için mevcuttu.

Burada sunulan teknik tercüme kalıplarını artan (incremental) bir yapıda öğrenir. Diğer bir deyişle tercüme kalıpları bir tercüme örnek kümesi kullanılarak öğrenilir, ve yeni tercüme kalıplarıda başka bir tercüme örnek kümesi ve daha önceden öğrenilen tercüme kalıpları yardımıyla öğrenilebilinir. Bu bize birden fazla tercüme örnek kümesi ile çalışa bilme olanağı vermiştir.

Bu proje kapsamında geliştirilen tekniklerin kesinlikle bilgisyar ile tercüme alanında kullanıbileceğine inanıyoruz. Burada geliştirilen sistem gibi bir sistem belirli bir alandaki (örneğin bilgisayar kullanım kitapçıkları) metinlerin tercümesinde kullanılabilir. Tercüme örnek kümeleri seçilirken de tercüme alanı göre seçilmelidir.

Kaynakça

1. Brona, C., and Padraig, C., Translating Software Documentation by Example: An EBMT Approach to Machine Translation, in: Proceedings of Int. ECAI Workshop: Multilinguality in the Software Industry, 1996.

2. Brown, R. D., Example-Based Machine Translation in the Pangloss System, in: Proceedings of COLING-96, 1996.

3. Brown, R. D., Automated Dictionary Extraction for "Knowledge-Free" Example-Based Translation, in: Proceedings of TMI'97, 1997.

4. Cicekli, I., and Güvenir, H. A., 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.

5. Furuse, O., and Iida, H., Cooperation between Transfer and Analysis in Example-Based Framework, in: Proceedings of COLING-92 , Nantes, France, 1992, pp:645-651.

6. Goodman, K., and Nirenburg, S., KBMT-89: A Case Study in Knowledge Based Machine Translation, Morgan Kaufmann, 1992.

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

8. Güvenir, H. A, and Cicekli, I., Learning Translation Templates from Examples, in: Information Systems, Vol. 23, No. 6, 1998, pp: 353-363

9. Hammond, K.J. (Ed.), Proceedings: Second Case-Based Reasoning Workshop, Pensacola Beach, FL, USA, Morgan Kaufmann, 1989.

10. Kaji, H., Kida Y., and Morimoto, Y., Learning Translation Templates from Bilingual Text, in: Proceedings of COLING-92, 1992, pp:672-678.

11. Kitano, H., 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.

12. Kolodner, J.L.: (Ed.) Proceedings of a Workshop on Case-Based Reasoning, Clearwater Beach, FL, USA, Morgan Kaufmann, 1988.

13. Lonsdale, D., Mitamura, T., and Nyberg, E., Acquisition of Large Lexicons for Practical Knowledge-Based MT, in: Machine Translation, Vol 9(3), Kluwer Academic Publishers, 1994, pp:251-283.

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

15. Mitumura, T., and Nyberg, E., The KANT System: Fast, Accurate, High-Quality Translation in Practical Domains, in: Proceedings of COLING-92, Nantes, France, 1992, pp:1069-1073.

16. Nagao, M. A., Framework of a Mechanical Translation between Japanese and English by Analogy Principle, in: Artificial and Human Intelligence, A. Elithorn and R Banerji (eds.), NATO Publications, 1984.

17. Nirenburg, S., Beale, S., and Domashnev, C., 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.

18. Öz, Z., and Cicekli, I., Ordering Translation Templates by Assigning Confidence Factors, in: Lecture Notes in Computer Science 1529, Springer Verlag, 1998, pp:51-61.

19. Ram, A., Indexing, Elaborationand Refinement: IncrementalLearning of Explanatory Cases, in: Janet L. Kolodner (ed.), Case-Based Learning, Kluwer Academic Publishers, 1993.

20. Sato, S., and Nagao, M., The Memory-Based Translation, in: Proceedings of COLING-90 , 1990.

21. Sato, S., MBT2: A Method for Combining Fragments of Examples in Example-Based Translation, Artificial Intelligence, Vol 75, Elsevier Science, 1995, pp: 31-50.

22. Smadja, F., McKeown, K. R., and Hatzivassiloglou, V., Translating Collocation for Bilingual Lexicons: A Statistical Approach in: Computational Linguistics, Vol 22(1), The MIT Press, 1996, pp:1-38.

23. Sumita, E., and Iida, H., Experiments and Prospects of Example-Based Machine Translation, in: Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics, 1991.

24. Wu, D., Xia, X., Large-Scale Automatic Extraction of an English-Chinese Translation Lexicon in: Machine Translation, Vol 9, Kluwer Academic Publishers, 1995, pp:285-313.

Yayınlanmış Makale ve Bildiriler

Bu rapora aşağıda yayınlanmış olan makale ve bildirilerin birer kopyası eklenmiştir.

1. Güvenir, H. A, and Cicekli, I., Learning Translation Templates from Examples, in: Information Systems, Vol. 23, No. 6, 1998, pp: 353-363
2. Öz, Z., and Cicekli, I., Ordering Translation Templates by Assigning Confidence Factors, in: Lecture Notes in Computer Science 1529, Springer Verlag, 1998, pp:51-61.
3. Öz, Z, Assigning Confidence Factors to Translation Templates, in: Proceedings of The 7th Turkish Artificial Intelligence and Neural Networks, June 1998, Ankara, pp:129-138.

BİBLİYOGRAFİK BİLGİ FORMU

1. Proje No: EEEAG-244 (197E011)

2. Rapor Tarihi: 4 Ekim 1999

3. Projenin Başlangıç ve Bitiş Tarihleri: 19 Ağustos 1997 – 19 Ağustos 1999

4. Projenin Adı:
Tercüme Kalıplarının Makina Öğrenmesi Teknikleri ile
Tercüme Örneklerinden Öğrenilmesi

5. Proje Yürütücüsü ve Yardımcı Araştırıcılar:
- Doç. Dr. İlyas Çiçekli (proje yürütücüsü)
- Doç. Dr. H. Altay Güvenir (araştırıcı)
- Zeynep (Öz) Orhan (araştırıcı)

6. Projenin Yürütüldüğü Kuruluş ve Adresi:
Bilgisayar Mühendisliği Bölümü
Bilkent Üniversitesi
06533 Bilkent, Ankara

7. Destekleyen Kuruluşların Adı ve Adresi:
- Bilkent Üniversitesi, 06533 Bilkent, Ankara
- TÜBİTAK, Kavaklıdere, Ankara

8. Öz (Abstract)

Bu proje kapsamında örneğe dayalı bilgisayar ile herhangi iki dil arasında tercüme yapabilen bir sistem geliştirdik. Bizim önerdiğimiz yöntemde iki dil arasındaki tercüme kalıpları verilen tercüme örneklerinden makina öğrenmesi teknikleri kullanılarak öğrenilir. Daha sonra bu öğrenilen tercüme kalıpları diğer metinlerin tercüme edilmesinde kullanılır. Bizim temel tercüme kalıplarını öğrenme algoritmamız verilen iki tercüme örneğindeki benzerliklerden ve farklılıklardan faydalanarak yeni tercüme kalıplarını öğrenir. Verilen iki tercüme örneğindeki kaynak dildeki cümleler arasındaki benzerlikler hedef dildeki cümleler arasındaki benzerliklere karşılık gelmelidir. Aynı şeklide, kaynak dildeki cümleler arasındaki farklılıklarda hedef dildeki cümleler arasındaki farklılıklara karşılık gelmelidir. Bu temel varsayımdan yola çıkarak makina öğrenmesi yöntemlerini kullanan tercüme kalıplarını öğrenme algoritması geliştirilmiştir. Buna ilavaten tercüme kalıplarına istatiksel yöntemler kullanarak güvenirlik faktörleri veren bir parça sistemize eklenmiştir. Bu güvenirlik faktörlerin amacı üretilen yanıtların bu faktörlere göre sıralayarak, ilk sıradaki yanıtların doğru yanıtı kapsama olasalığını artırmaktır.

Anahtar Sözcükler: Doğal Dil İşleme, Bilgisayar ile Tercüme, Makina Öğrenmesi

9. Proje ile ilgili YayınTebliğler ilgili Bilgiler

1. Güvenir, H. A, and Cicekli, I., Learning Translation Templates from Examples, in: Information Systems, Vol. 23, No. 6, 1998, pp: 353-363
2. Öz, Z., and Cicekli, I., Ordering Translation Templates by Assigning Confidence Factors, in: Lecture Notes in Computer Science 1529, Springer Verlag, 1998, pp:51-61.
3. Öz, Z, Assigning Confidence Factors to Translation Templates, in: Proceedings of The 7th Turkish Artificial Intelligence and Neural Networks, June 1998, Ankara, pp:129-138.
4. Öz, Z, Confidence Factors Assignment to Translation Templates, M.S. Thesis, Bilkent University, Bilkent, Ankara.

10. Bilim Dalı

Doçentlik B. Dalı Kodu:
ISIC Kodu:
Uzmanlık Alanı Kodu: Yapay Zeka, Doğal Dil İşleme

11. Dağıtım: Sınırsız

12. Raporun Gizlilik Durumu: Gizli Değil