2000-7 DİP

TÜBİTAK

TÜRKİYE BİLİMSEL VE
TEKNİK ARAŞTIRMA KURUMU

THE SCIENTIFIC AND TECHNICAL
RESEARCH COUNCIL OF TURKEY

Elektrik, Elektronik ve Enformatik Araştırma Grubu

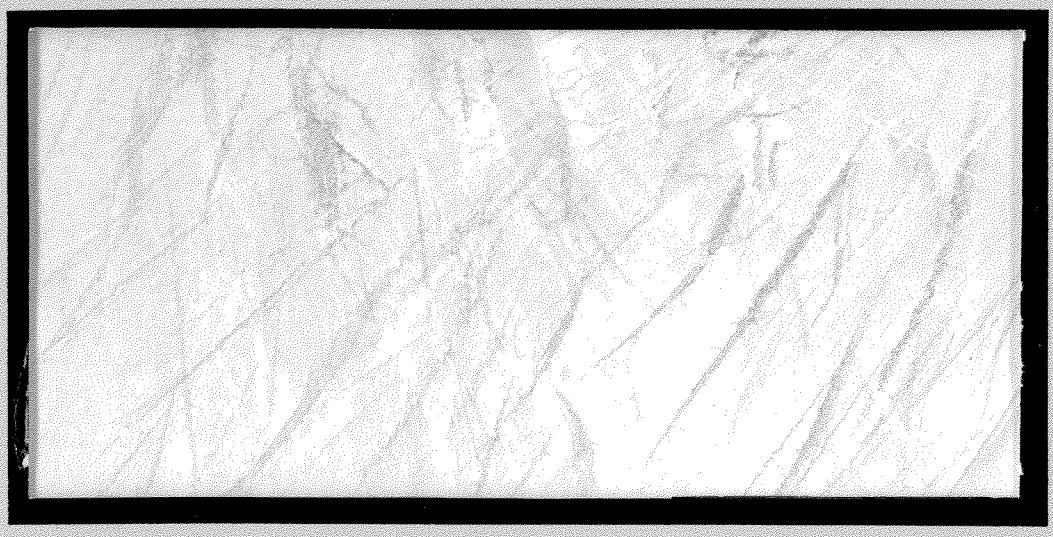
Electric, Electronics and Informatics Research Grant Committee

2000-7 DEP

TÜBİTAK

TÜRKİYE BİLİMSEL VE
TEKNİK ARAŞTIRMA KURUMU

THE SCIENTIFIC AND TECHNICAL
RESEARCH COUNCIL OF TURKEY



Elektrik, Elektronik ve Enformatik Araştırma Grubu

Electric, Electronics and Informatics Research Grant Committee

TERCÜME KALIPLARININ MAKİNA ÖĞRENMESİ TEKNİKLERİ İLE TERCÜME ÖRNEKLERİNDEN ÖĞRENİLMESİ

PROJE NO : EEEAG - 244 (197E011)

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

Şekil Listesi

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 kullanılr. 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 $\leftrightarrow$ Y `içeceğim` **if** $X \leftrightarrow Y$
- `orange juice` $\leftrightarrow$ `portakal suyu`
- `coffee` $\leftrightarrow$ `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 $EE \leftrightarrow ET$ 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, $SE_1 \leftrightarrow ST_0$, 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 $SE_1 \leftrightarrow ST_0$ 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` $\leftrightarrow$ `Cathy için` XT_1 `+yH satın al+DH+m`
  if $XE_1 \leftrightarrow XT_1$
- `book` $\leftrightarrow$ `kitap`
- `ring` $\leftrightarrow$ `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:
  $SE_0, DE_0, \ldots, DE_{n-1}, SE_n \leftrightarrow ST_0, DT_0, \ldots, DT_{m-1}, ST_m$

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 $((DE_{k,1}, DT_{l,1}), \ldots, (DE_{k,n}, DT_{l,n}))$ 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 $n \geq 1$ **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 $((SE_{k,l}, ST_{l,l}), \ldots, (SE_{k,n}, ST_{l,n}))$ 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 ayırarak 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)` farklılığının `(+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 algoritmasının 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.

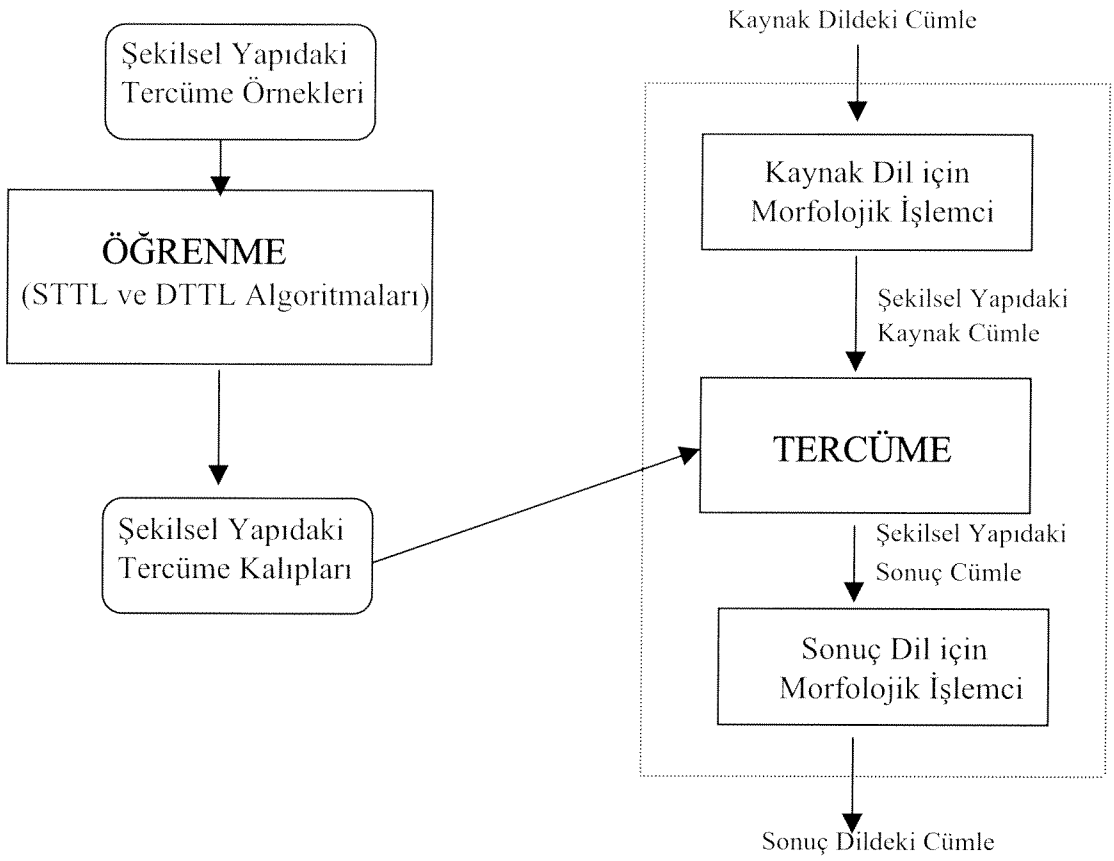
XE_1 eat+p a XE_2 ↔ bir XT_2 ye+DH XT_1 **if** $XE_1 \leftrightarrow XT_1$ **and** $XE_2 \leftrightarrow XT_2$

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 kısmının (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 ($güvenirlikfaktör_{P_i} = 1$).
     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 $güvenirlikfaktör_{P_i} = 1/(1+mindist)$ 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: $güvenirlikfaktör_{P4} = 1/(1+dist) = 0.5$

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 $kalıp_i$ 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

Pergamon

Information Systems Vol. 23, No. 6, pp. 353–363, 1998

Printed in Great Britain
0306-4379/98 $19.00 + 0.00

LEARNING TRANSLATION TEMPLATES FROM EXAMPLES

HALIL ALTAY GÜVENIR and ILYAS CICEKLI

Bilkent University, Department of Computer Engineering and Information Science, Ankara 06533, Turkey

(Received 11 September 1997; in final revised form 12 June 1998)

Abstract — This paper proposes a mechanism for learning lexical level correspondences between two languages from a set of translated sentence pairs. The proposed mechanism is based on an analogical reasoning between two translation examples. Given two translation examples, the similar parts of the sentences in the source language must correspond to the similar parts of the sentences in the target language. Similarly, the different parts should correspond to the respective parts in the translated sentences. The correspondences between the similarities, and also differences are learned in the form of translation templates. The approach has been implemented and tested on a small training dataset and produced promising results for further investigation. Copyright ©1998 Elsevier Science Ltd

Key words: Machine Learning, Machine Translation, Translation Templates

1. INTRODUCTION

Traditional machine translation (MT) systems require large-scale knowledge about both source and target languages in the form of computational grammars, lexicons and domain knowledge. In these traditional knowledge-based machine translation systems such as KBMT-89 [4], this large-scale knowledge is hand-coded and hand-crafted for specific domains. They make the use of this large-scale knowledge residing in different resources such as lexicons, grammar rules and mapping rules during translation. Since the acquisition of this kind of knowledge is time-consuming and expensive, researchers have been studying to the ways of automatically and semi-automatically acquiring some portions of the required knowledge from large corpora. For example, in the KANT [13] system which is an immediate descendant of the KBMT-89 system, some methods for automatic acquisition of lexicons from a large-corpus are used [11]. The work presented in this paper tries to automate the acquisition of the required knowledge for machine translation task (except for morphological rules) from a sentence-level aligned bilingual text corpora only.

Because of the large-scale acquisition problem in traditional approaches to machine translation, corpus-based approach is one of the alternative directions that have been proposed to overcome the difficulties of traditional systems. Two fundamental directions in corpus-based MT have been followed. These are *statistical* and *example-based* machine translation (EBMT), also called *memory-based* machine translation (MBMT). Both approaches assume the existence of a bilingual parallel text (an already translated corpus) to derive a translation for an input. While statistical MT techniques use statistical metrics to choose the most probable structures in the target language, EBMT techniques employ pattern matching techniques to translate subparts of the given input [1].

Exemplar-based representation has been widely used in Machine Learning (ML). According to Medin and Schaffer [12], who originally proposed exemplar-based learning as a model of human learning, examples are stored in memory without any change in the representation. The characteristic examples stored in the memory are called exemplars. The basic idea in exemplar-based learning is to use past experiences or cases to understand, plan, or learn from novel situations [7, 10, 15]. The learning paradigm used in this paper is referred as *Exemplar-Based Generalization* (coined by Salzbeg [17]), where exemplars are not only simple examples, but templates that are obtained by generalizing appropriate components of examples [5].

EBMT has been proposed by Nagao [14] as *translation by analogy* which is in parallel with memory based reasoning [20], case-based reasoning [16] and derivational analogy [2]. Example-based translation relies on the use of past translation examples to derive a translation for a given

input [3, 9, 19, 21]. The input sentence to be translated is compared with the example translations analogically to retrieve the *closest* examples. Then, the fragments of the retrieved examples are translated and recombined in the target language. Prior to the translation of an input sentence, the correspondences between the source and target languages should be available to the system; however this issue has not been given enough consideration by the current EBMT systems. Kitano has adopted the manual encoding of the translation rules, however this is a difficult and an error-prone task for a large corpus [8]. Sato [18] also proposed an exemplar-based system with manually encoded matching expressions which are used as translation templates. This paper formulates the acquisition of translation rules as a machine learning task in order to automate the process.

Our first attempt was to construct parse trees between the example translation pairs [6]. However, the difficulty was the unavailability of a reliable parser for both languages. In this paper, we propose a technique which stores exemplars in the form of *templates* that are *generalized exemplars*, rather than parse trees. A template is an example translation pair where some components (e.g., words stems and morphemes) are generalized by replacing them with variables in both sentences, and establishing bindings between the variables. We will refer this technique as GEBMT for *Generalized Exemplar Based Machine Translation.*

The algorithm we propose here, for learning such templates, is based on a heuristic to learn the correspondences between the patterns in the source and target languages, from two translation pairs. The heuristic can be summarized as follows: Given two translation pairs, if the sentences in the source language exhibit some similarities, then the corresponding sentences in the target language must have similar parts, and they must be translations of the similar parts of the sentences in the source language. Further, the remaining differing constituents of the source sentences should also match the corresponding differences of the target sentences. However, if the sentences do not exhibit any similarity, then no correspondences are inferred. Consider the following translation pairs given in English and Turkish to illustrate the heuristic:

I took a ticket from Mary ↔ Mary'den bir bilet aldım
I took a pen from Mary ↔ Mary'den bir kalem aldım

Similarities between the translation examples are shown as underlined. The remaining parts are the differences between the sentences. We represent the similarities in English as "I took a X^E from Mary", and the corresponding similarities in Turkish as "Mary'den bir X^T aldım". According to our heuristic, these similarities should correspond each other. Here, X^E denotes a component that can be replaced by *any* appropriate structure in English and X^T refers to its translation in Turkish. This notation represents an *abstraction* of the differences "ticket" vs. "pen" in English and "bilet" vs. "kalem" in Turkish. Continuing even further, we infer that "ticket" should correspond to "bilet" and "pen" should correspond to "kalem"; hence learning further correspondences between the examples.

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

The rest of the paper is organized as follows. Section 2 explains the representation in the form of translation templates. The TTL algorithm is described in Section 3, and Section 4 illustrates the TTL algorithm on some example translation pairs. Section 5 describes how these translation templates can be used in translation. Section 6 concludes the paper.

2. TRANSLATION TEMPLATES

A template is a generalized translation exemplar pair, where some components (e.g., word stems and morphemes) are generalized by replacing them with variables in both sentences, and establishing bindings between these variables. For example, the following three translation templates that would be learned from the example translations given above are:

I took a X^1 from Mary $\leftrightarrow$ Mary'den bir X^2 aldım
    if $X^1 \leftrightarrow X^2$
ticket $\leftrightarrow$ bilet
pen $\leftrightarrow$ kalem

The first translation template is read as the sentence "I took a X^1 from Mary" in L^1 and the sentence "Mary'den bir X^2 aldım" in L^2 are translations of each other, given that X^1 in L^1 and X^2 in L^2 are translations of each other. Therefore, for example, if it has already been acquired that "basket" in L^1 and "sepet" in L^2 are translations of each other, i.e., "basket" $\leftrightarrow$ "sepet" then the sentence "I took a basket from Mary" can be translated into L^2 as "Mary'den bir sepet aldım". Each of the second and third templates represents an atomic correspondence of two strings in the languages L_1 and L_2 without any dependency. In fact, we also use given example translations as atomic translation templates.

Since the TTL algorithm is based on finding the similarities and differences between translation examples, the representation of sentences plays an important role. As it is, the TTL algorithm may use the sentences exactly as they can be found in a regular text. That is, no grammatical information or no preprocessing, e.g. bracketing, on the bilingual parallel corpus is needed. Therefore, it is a grammarless extraction algorithm for phrasal translation templates from bilingual parallel texts.

For agglutinative languages such as Turkish, this surface level representation of the sentences limits the generality of the templates to be learned. For example, the translation of the sentence "I am coming" in Turkish is a single word "geliyorum". When the surface level representation is used, it is not possible to find a template from that translation. Therefore, we will represent a word in its lexical level representation, that is, its stem and its morphemes. For example, the translation pair "I am coming" $\leftrightarrow$ "geliyorum" will be represented as "i am come+ing" $\leftrightarrow$ "gel+Hyor+yHm". Similarly, the pair "I am going" $\leftrightarrow$ "gidiyorum" will be represented as "i am go+ing" $\leftrightarrow$ "gid+Hyor+yHm". Here, the + symbol is used to mark the beginning of a morpheme, and the letter H in the morphemes represents a vowel whose surface level realization is determined according to vowel harmony rules of the Turkish language. According to this representation, these two translation pairs would be given as

I am come+ing $\leftrightarrow$ gel+Hyor+yHm
I am go+ing $\leftrightarrow$ gid+Hyor+yHm

The translation templates learned from these two translation pairs are

I am X^1+ing $\leftrightarrow$ X^2+Hyor+yHm
    if $X^1 \leftrightarrow X^2$
come $\leftrightarrow$ gel
go $\leftrightarrow$ gid

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

3. LEARNING TRANSLATION TEMPLATES

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

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

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

Here, S_k^1 represents a similarity (a sequence of common items) between E_a^1 and E_b^1. Similarly, $D_k^1 : (D_{k,a}^1, D_{k,b}^1)$ represents a difference between E_a^1 and E_b^1, where $D_{k,a}^1$ and $D_{k,b}^1$ are non-empty differing items between two similar constituents S_k^1 and S_{k+1}^1. Corresponding differing constituents do not contain common items. That is, for a difference D_k, $D_{k,a}$ and $D_{k,b}$ do not contain any common item. Also, no lexical item in a similarity S_i appears in any previously formed difference D_k for $k < i$. Any of S_0^1, S_n^1, S_0^2 or S_m^2 can be empty, however, S_i^1 for $0 < i < n$ and S_j^2 for $0 < j < m$ must be non-empty. Furthermore, at least one similarity on each side must be non-empty. Note that there exists either a unique match or no match between two example translation pairs.

For instance, let us assume that the following translation examples are given for translation pairs "`I gave the ticket to Mary`" ↔ "`Mary'e bileti verdim`" and "`I gave the pen to Mary`" ↔ "`Mary'e kalemi verdim`":

`I give+p the ticket to Mary` ↔ `Mary+'e bilet+yH ver+DH+m`
`I give+p the pen to Mary` ↔ `Mary+'e kalem+yH ver+DH+m`

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

`I give+p the (ticket,pen) to Mary` ↔ `Mary+'e (bilet,kalem)+yH ver+DH+m`

That is,

S_0^1= `I give+p the,` D_0^1= `(ticket,pen),` S_1^1= `to Mary,`
S_0^2= `Mary+'e,` D_0^2= `(bilet,kalem),` S_1^2= `+yH ver+DH+m.`

After a match sequence is found for two translation examples, we use a learning heuristic to infer translation templates from that match sequence. This heuristic tries to locate corresponding differences in the match sequence. If it can locate all corresponding differences, it can learn a new translation template by replacing all differences with variables.

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

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

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

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

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

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

For example, since the match sequence of the example above contains a single difference in both sides, the following translation template and two additional translation templates from the corresponding differences `(ticket,pen)` and `(bilet,kalem)` can be inferred:

`I give+p the` X^1 `to Mary` ↔ `Mary+'e` X^2`+yH ver+DH+m`
if $X^1 \leftrightarrow X^2$
`ticket` ↔ `bilet`
`pen` ↔ `kalem`

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

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

We call this corresponding pair of differences as *Corresponding Difference Pair* (CDP). After finding $n-1$ CDPs, the last two unchecked differences, one at each side (language), should correspond to each other. Thus, for all differences in the match sequence, we determine which difference in one side corresponds to which difference on the other side.

Let us assume that the list

$$CDP_1, CDP_2, \cdots, CDP_n$$

represents the list of all CDPs, where CDP_n is the pair of the two unchecked differences. A CDP_i is the pair of two differences in the form $(D^1_{k_i}, D^2_{l_i})$. For each CDP_i, we replace $D^1_{k_i}$ with a variable X^1_i and $D^2_{l_i}$ with a variable X^2_i in a match sequence $M_{a,b}$. The newly formed match sequence is called as $M_{a,b}DV$ for *match sequence with difference variables.* As a result, the following translation template can be inferred.

$M_{a,b}DV$
if $X^1_1 \leftrightarrow X^2_1$ **and** $\cdots$ **and** $X^1_n \leftrightarrow X^2_n$

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

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

For example, the following translation examples, which are lexical forms of translation pairs "I gave the book" $\leftrightarrow$ "Kitabı verdim" and "You gave the pen" $\leftrightarrow$ "Kalemi verdin", have two differences on both sides.

```
i give+p the book ↔ kitab+yH ver+DH+m
you give+p the pen ↔ kalem+yH ver+DH+n
```

The following match sequence is obtained for these examples.

```
(i,you) give+p the (book,pen) ↔ (kitab,kalem) +yH ver+DH (+m,+n)
```

Without prior information, we cannot determine if `i` corresponds to `kitab` or `+m`. However, if it has already been learned that `i` corresponds to `+m` and `you` corresponds to `+n`, then the following translation template and two additional translation templates can be inferred.

X^1_1 `give+p the` $X^1_2 \leftrightarrow X^2_2$`+yH ver+DH`$X^2_1$
if $X^1_1 \leftrightarrow X^2_1$ **and** $X^1_2 \leftrightarrow X^2_2$
`book ↔ kitab`
`pen ↔ kalem`

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

procedure TTL($M_{a,b}$)
begin
- Let the match sequence $M_{a,b}$ be:
  $S_0^1, D_0^1, \cdots, D_{n-1}^1, S_n^1, \leftrightarrow S_0^2, D_0^2, \cdots, D_{m-1}^2, S_m^2$

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

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

end

Fig. 1: The TTL Algorithm

The TTL algorithm described in this section can only be applied to the match sequences with the same number of differences on both sides. If we get a match sequence with different number of differences, we try to equate the number of differences by separating differences before we apply our TTL algorithm. If the number of differences in one side is less than the number of differences in the other side, we try to divide the differences in the side with less differences to increase their number. If both constituents of a difference contain more than one morpheme (or root word), we may divide that difference into two differences by separating both constituents of that difference from morpheme boundaries.

In each pass of the learning phase, the TTL algorithm is applied to each pair of examples in the training set to infer translation templates. The learning phase continues until no new translation template is learned. In other words, when the number of new learned translation templates in a pass is zero, the learning process stops. The maximum number of passes of the learning phase is theoretically $n - 1$ when the number of examples in the training set is n. However, in usual training sets, the number of passes needed is much less. For example, only 4 passes were executed for a sample training set of 112 example pairs.

After all translation templates are learned, they are sorted according to their specificities. Given two templates, the one that has a higher number of terminals is more specific than the other. Note that, the specificity is defined according to the source language. For two way translation, the templates are ordered once for each language as the source.

4. LEARNING EXAMPLES

For an empirical evaluation of the TTL algorithm, we have implemented it in PROLOG and tested on sample bilingual parallel texts. One of artificially created training sets contained 747 training pairs. In the first pass, the TTL algorithm learned 642 translation templates. No new templates were learned in the second pass. On a SPARC 20/61 workstation, each pass took about

44 seconds real time. Using these 1389 templates including 747 translation examples, translation of a new sentence, took about 85 milliseconds on the average.

In this section, we illustrate the behavior of TTL algorithms on some sample training example pairs.

Example 1 Given the example translations "I saw you at the garden" ↔ "Seni bahçede gördüm" and "I saw you at the party" ↔ "Seni partide gördüm", their lexical level representations are

```
i see+p you at the garden ↔ sen+yH bahçe+DA gör+DH+m
i see+p you at the party ↔ sen+yH parti+DA gör+DH+m .
```

From these examples with one pair of differences in both sides, the following translation templates are learned:

```
i see+p you at the X1 ↔ sen+yH X2+DA gör+DH+m
    if X1↔X2
garden ↔ bahçe
party ↔ parti.
```

□

Example 2 Given the example translations "It falls" ↔ "Düşer", "I will take the car" ↔ "Arabayı alacağım", "If a pen is dropped then it falls" ↔ "Bir kalem bırakılırsa, düşer" and "If he brought the keys then I will take car" ↔ "anahtarları getirdiyse arabayı alacağım", their lexical level representations are

```
it fall+s ↔ düş+Ar
i will take the car ↔ araba+yH al+yAcAk+yHm
if a pen is drop+pp then it fall+s ↔
    bir kalem bırak+Hl+Hr+ysA, düş+Ar
if he bring+p the keys then i will take the car ↔
    anahtarlarI getir+DH+ysA, araba+yH al+yAcAk+yHm.
```

The match sequence between the last two example translations contains two similarities for `if` and `then`, and two differences. Since there are more than one differences, no translations templates can be learned directly. However, with the help of the first two example pairs, the following translation templates are learned:

```
a pen is drop+pp ↔ bir kalem bırak+Hl+Hr
he bring+p the keys ↔ anahtarlarI getir+DH
if X1 then X2 ↔ X1+ysA, X2
    if X1↔ X1 and X2↔X2.
```

□

Example 3 Given the example translations "I would like to look at it" ↔ "Ona bakmak isterim" and "Do not look at it" ↔ "Ona bakma" their lexical level representations are

```
i would like to look at it ↔ o+nA bak+mAk iste+Hr+yHm
do not look at it ↔ o+nA bak+mA .
```

Even from these structurally different translation examples, the following translation templates are learned:

```
X1 look at it ↔ o+nA bak X2
    if X1 ↔ X2
i would like to ↔ +mAk iste+Hr+yHm
do not ↔ +mA .
```

□

Example 4 Given the example translations "he can write" ↔ "yazabilir", "do not talk" ↔ "konuşma", "he can write while he is reading" ↔ "okurken yazabilir" and "do not talk while you are eating" ↔ "yemek yerken konuşma", their lexical level representations are

he can write ↔ yaz+yAbil+Hr
do not talk ↔ konuş+mA
he can write while he is read+ing ↔ oku+Hr+yken yaz+yAbil+Hr
do not talk while you are eat+ing ↔ yemek ye+Hr+yken konuş+mA .

The following useful translation templates are learned, from these translations examples:

X_1^1 while X_2^1+ing ↔ X_2^2+Hr+yken X_1^2
    if $X_1^1 \leftrightarrow X_1^2$ **and** and $X_2^1 \leftrightarrow X_2^2$
he is read ↔ oku
you are eat ↔ yemek ye . □

The last two translation templates may be used to fill in more complex translation templates.

Example 5 Natural languages are full of idiomatic expressions. For example, in Turkish, "kafayı yediler" is such an expression meaning "they have got crazy", while its literal translation would be "they ate the head". Since, the TTL algorithm sorts the templates according to their specificities, such idiomatic expressions can be handled easily. Consider that the following examples are provided to learn the templates: "we have got crazy" ↔ "kafayı yedik", "this is an apple" ↔ "bu bir elmadır", "this is an orange" ↔ "bu bir portakaldır", "i ate the apple" ↔ "elmayı yedim", and "you ate the orange" ↔ "portakalı yedin". The lexical level representations are

this is an apple ↔ bu bir elma+DHr
this is an orange ↔ bu bir portakal+DHr
they have get+p crazy ↔ kafa+yH ye+DH+lAr
we have get+p crazy ↔ kafa+yH ye+DH+k
i eat+p the apple ↔ elma+yH ye+DH+m
you eat+p the orange ↔ portakal+yH ye+DH+n .

From these translation examples, in the order of specificity, the following useful translation templates are learned:

X_1^1 have get+p crazy ↔ kafa+yH ye+DHX_1^2
    if $X_1^1 \leftrightarrow X_1^2$
this is an X_1^1 ↔ bu bir X_1^2+DHr
    if $X_1^1 \leftrightarrow X_1^2$
X_1^1 eat+p the X_2^1 ↔ X_2^2+yH ye+DHX_1^2
    if $X_1^1 \leftrightarrow X_1^2$ **and** $X_2^1 \leftrightarrow X_2^2$
they ↔ +lAr
we ↔ +k
apple ↔ elma
orange ↔ portakal
i ↔ +m
you ↔ +n . □

Example 6 Our example pairs do not need to be pairs of sentences. They can also be pairs of expressions. First two example pairs of the following four examples are pairs of expressions:

red car ↔ kırmızı araba
red truck ↔ kırmızı kamyon
he buy+p a pen ↔ bir kalem satın al+DH
he buy+p a book ↔ bir kitap satın al+DH.

From these examples, the following translation templates are learned:

red X^1 ↔ kırmızı X^2
    if $X^1 \leftrightarrow X^2$
car ↔ araba
truck ↔ kamyon
he buy+p a X^1 ↔ bir X^2 satın al+DH
    if $X^1 \leftrightarrow X^2$
pen ↔ kalem
book ↔ kitap. □

5. TRANSLATION

The templates learned by the TTL algorithm can be used in the translation directly. These templates can be used for translation in both directions. The outline of the translation process is given below:

1. First, the lexical level representation of the input sentence to be translated is derived.
2. The most specific translation templates matching the input are collected. These templates are those that are most similar to the sentence to be translated.
3. For each selected template, its variables are instantiated with the corresponding values in the source sentence. Then, templates matching these bound values are sought. If they are found successfully, their values are replaced in the variables corresponding to the sentence in the target language. This process continues recursively for the variables which may be introduced in this step.
4. The surface level representation of the sentence obtained in the previous step is generated.

For instance, after learning the templates in Example 5, if the input is given as "kafayı yedim", first its lexical level representation, which is "kafa+yH ye+dH+m", is derived. Although there are two matching templates (the first and the third), the most specific template matching this is

X_1^1 have get+p crazy ↔ kafa+yH ye+DHX_1^2
    if $X_1^1 \leftrightarrow X_1^2$.

The variable X_1^2 is instantiated with "+m". Then, the translation of "+m" is found to be "i" using

i ↔ +m.

Therefore, replacing the value of "i" for X_1^1 in the template, the lexical level representation "i have get+p crazy" is obtained. Finally, the surface level representation "I have got crazy" is derived easily. On the other hand, if the input sentence is "portakalı yedim", only the third template can be used, and the correct translation "I ate the orange" is obtained.

Note that, if the sentence in the source language is ambiguous, then templates corresponding to each sense will be retrieved, and the sentences for each sense will be generated. Among the possible translations, a human user can choose the right one according to the context. We expect that the correct answer will be among the first translations because of the usage of the specificity rule.

The translation phase is a recursive process for replacing variables in the templates. For example, to translate English sentence "If he bought a red pen then he can write" into Turkish, first we get its lexical form "if he buy+p a red pen then he can write". This sentence will match with the left side of the third translation template given in Example 2 by binding X_1^1 to "he buy+p a red pen" and X_2^1 to "he can write". Then, we will seek translations of these portions. The second expression "he can write" will be translated into "yaz+yAbil+Hr" because their correspondence was directly given in Example 4. On the other hand, the first one will match with the fourth translation template in Example 6 by binding X^1 to "red pen". When we seek

the translation of "red pen", it will match with the first translation template in Example 6 by binding X^1 to "pen". Then, "pen" will be translated into "kalem" by using the fifth translation template in Example 6. Thus,

"pen" is translated into "kalem",
"red pen" is translated into "kırmızı kalem",
"he buy+p a red pen" is translated into "bir kırmızı kalem satın al+DH",
"he can write" is translated into "yaz+yAbil+Hr",
Finally, "if he buy+p a red pen then he can write" is translated into
"bir kırmızı kalem satın al+DH+ysA, yaz+yAbil+Hr".

We get the surface form "bir kırmızı kalem satın aldıysa, yazabilir" for the lexical form of Turkish sentence.

6. CONCLUSION

In this paper, we have presented a model for learning translation templates between two languages. The model is based on a simple pattern matcher. We integrated this model with an example-based translation model into Generalized Exemplar-Based Machine Translation. We have implemented this model as the TTL (Translation Template Learner) algorithm. The TTL algorithm is illustrated in learning translation templates between Turkish and English. The approach is applicable to any pair of languages.

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

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

The accuracy of the translation templates learned by this approach is quite high with ensured grammaticality. Given that a translation is carried out using the rules learned, the accuracy of the output translation critically depends on the accuracy of the rules learned.

We do not require an extra operation to maintain the grammaticality and the style of the output, as in Kitano's EBMT model [8]. The information necessary to maintain these issues is directly provided by the translation templates.

The learning and translation times on the small training set are quite reasonable, and that indicates the program will scale up real large training corpora. Note that this algorithm is not specific to English and Turkish languages, but should be applicable to the task of learning machine translation between any pair of languages. Although the learning process on a large corpus will take a considerable amount of time, it is only one time job. After learning the translation templates, the translation process is fast.

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

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

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

REFERENCES

[1] D. Arnold, L. Balkan, R.L. Humphreys, and L. Sadler S. Meijer. *Machine Translation: An Introductory Guide.* NCC Blackwell (1994).

[2] J.G. Carbonell. Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In J.W. Shavlik and T.G. Dietterich, editors, *Readings in Machine Learning*, pp. 247–288, Morgan Kaufmann (1990).

[3] O. Furuse and H. Iida. Cooperation between transfer and analysis in example-based framework. In *Proceedings of COLING-92*, pp. 645–651, Nantes, France (1992).

[4] K. Goodman and S. Nirenburg. *KBMT-89: A Case Study in Knowledge Based Machine Translation.* Morgan Kaufmann (1992).

[5] H.A. Güvenir and I. Sirin. Classification by feature partitioning. *Machine Learning*, **23**(1):47–67 (1996).

[6] H.A. Güvenir and A. Tunç. Corpus-based learning of generalized parse tree rules for translation. In Gordon McCalla, editor, *Proceedings of the Eleventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence on Advances in Artificial Intelligence*, volume 1081 of *LNAI*, pp. 121–132, Springer, Berlin (1996).

[7] K.J. Hammond. *Proceedings of the Second Case-Based Reasoning Workshop.* Morgan Kaufmann (1989).

[8] H. Kitano. A comprehensive and practical model of memory-based machine translation. In *13. IJCAI*, Chambéry, France (1993).

[9] H. Kitano. A full-text experiment in example-based machine translation. In *Proceedings of the International Conference on New Methods in Language Processing, NeMLap* (1994).

[10] J.L. Kolodner, editor. *Proceedings of a Workshop on Case-Based Reasoning (DARPA).* Morgan Kaufmann (1988).

[11] D. Lonsdale, T. Mitamura, and E. Nyberg. Acquisition of large lexicons for practical knowledge-based mt. *Machine Translation*, **9**(3):251–283 (1994).

[12] D.L. Medin and M.M. Schaffer. Context theory of classification learning. *Psychological Review*, **85**:207–238 (1978).

[13] T. Mitumura and E. Nyberg. The KANT system: Fast, accurate, high-quality translation in practical domains. In *Proceedings of COLING-92*, pp. 1069–1073, Nantes, France (1992).

[14] M. Nagao. A framework of a mechanical translation between english and japanese by analogy principle. In A. Elithorn and R. Banerji, editors, *Artifical and Human Intelligence*, pp. 173–180, North-Holland (1984).

[15] A. Ram. Indexing and elaboration and refinement: Incremental learning of explanatory cases. *Machine Learning*, **10**:201–248 (1993).

[16] C.K. Riesbeck and R.C. Schank. *Inside Case-Based Reasoning.* Lawrence Erlbaum Assoc., Hillsdale, N.J. (1989).

[17] S. Salzberg. A nearest hyperrectangle learning method. *Machine Learning*, **6**:251–276 (1991).

[18] S. Sato. MBT2: a method for combining fragments of examples in example-based translation. *Artificial Intelligence*, **75**:31–50 (1995).

[19] S. Sato and M. Nagao. Toward memory-based translation. In *Proceedings of COLING-90*, pp. 247–252, Helsinki, Finland (1990).

[20] C. Stanfill and D. Waltz. Toward memory-based reasoning. *Communications of the ACM*, **29**(12):1213–1228 (1986).

[21] E. Sumita and H. Iida. Experiments and prospects of example-based machine translation. In *Proceedings of the 29th Annual Meeting of the Association for Computational Linguistics*, pp. 185–192, Berkeley, California, USA (1991).

Ordering Translation Templates by Assigning Confidence Factors*

Zeynep Öz and Ilyas Cicekli

Dept. of Comp. Eng. and Info. Sc., Bilkent University,
06533 Bilkent, Ankara, TURKEY,
{ozzey,ilyas}@cs.bilkent.edu.tr

Abstract. TTL (*Translation Template Learner*) algorithm learns lexical level correspondences between two translation examples by using analogical reasoning. The sentences used as translation examples have similar and different parts in the source language which must correspond to the similar and different parts in the target language. Therefore these correspondences are learned as translation templates. The learned translation templates are used in the translation of other sentences. However, we need to assign confidence factors to these translation templates to order translation results with respect to previously assigned confidence factors. This paper proposes a method for assigning confidence factors to translation templates learned by the TTL algorithm. Training data is used for collecting statistical information that will be used in confidence factor assignment process. In this process, each template is assigned a confidence factor according to the statistical information obtained from training data. Furthermore, some template combinations are also assigned confidence factors in order to eliminate certain combinations resulting bad translation.

1 Introduction

Traditional approaches to machine translation (MT) require detailed knowledge about languages and the world knowledge. Therefore corpus-based machine translation is a good alternative for avoiding them. *Example-based* machine translation (EBMT) is one of the main approaches of corpus-based machine translation and originally proposed by Nagao [12]. This approach is based on the idea of performing translation by imitating translation examples of similar sentences. It involves translating the source language into the target language via remindings from the previous translation cases as stated in Brona [5]. After this proposal several machine translation methods that utilize translation examples and bilingual corpora have been studied such as [13, 15, 16, 1]. EBMT is the marriage of the MT and *Case-based reasoning* techniques (CBR). Finding the

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

correspondence of units in a bilingual text, retrieving the best matches from previous translation examples, and producing the translation of the given input by using these examples are the fundamental phases in EBMT. Brown [2] and Gale [6] have proposed methods for establishing correspondence between sentences in bilingual corpora. Brown [3], Sadler [14] and Kaji [9] have tackled the problem of establishing correspondences between words and phrases in bilingual texts.

Statistical machine translation is another approach of corpus-based machine translation. Statistical MT techniques use statistical metrics to choose the best structures in the target language among all possible candidates. These techniques are useful for retrieving the best matches from the previous translation examples, which is a vital issue in EBMT. This fact motivated us to develop a machine translation system that is a combination of statistical MT and EBMT.

Using previous examples for learning from new examples is the main idea behind exemplar-based learning which is originally proposed by Medin and Schaffer [11]. This way of learning stores the examples in memory without any change in the representation. The characteristic examples stored in the memory are called exemplars.

In the translation process, providing the correspondences between the source and target languages is a very difficult task in EBMT. Although, manual encoding of the translation rules has been achieved by Kitano [10], when the corpus is very large, it becomes a complicated and error-prone task. Therefore Cicekli and Güvenir [7, 4] offered a technique in which the problem is taken as a machine learning task. Exemplars are stored in the form of templates that are generalized exemplars. A template is an example translation pair where some components (e.g., words stems and morphemes) are generalized by replacing them with variables in both sentences, and establishing bindings between variables. These templates are learned by using translation examples and finding the correspondences between the patterns in the source and target languages. The heuristic of the translation template learning (TTL) [7, 4] algorithm can be summarized as follows: Given two translation pairs, if there are some similarities in the source language, then the corresponding sentences in the target language must have similar parts, and they must be translations of the similar parts of the sentences in the source language. Similar parts are replaced with variables to get a template which is a generalized exemplar by this method. Translation examples are stored as a list of string formed by strings of root words and morphemes. In other words, the lexical level representation of the sentences are used.This representation of translation examples is suitable for learning algorithm. If we used surface level representation, the number of correspondences would be decreased and we could learn less number of generalized exemplars. For example the sentence pair **i came from school⇔ben okuldan geldim** is stored as:

i come+p from school⇔ben okul+DAn gel+DH+m

where *i, come, from, school* denote root words and *+p* denotes the past tense morpheme in English sentence, and *ben, okul, gel* denote root words and *+DAn, +DH, +m* denote ablative, past tense and first singular person morphemes in

Turkish sentence. The following translation pairs given in English and Turkish illustrates the heuristic:

I go+p to school by bus ⇔ okul +yA otobüs+ylA git+DH+m
I go+p to city by bus ⇔ şehir +yA otobüs+ylA git+DH+m

The similarities between the translation examples are underlined. The similarities in English are represented as **I go+p to** X^{L_1} **by bus**, and the corresponding similarities in Turkish as X^{L_2}**+yA otobüs+ylA git+DH+m** by replacing differences by variables. According to the heuristic, these similarities should correspond to each other. Here, X^{L_1} denotes a component that can be replaced by any appropriate structure in English and X^{L_2} refers to its translation in Turkish. In addition to this, it is also inferred that *school* is the translation of *okul* and *city* is the translation of *şehir*. This shows that it is possible to learn more than one template by using two translation examples.

The order of the translation templates that will be used for the translation of new sentences is an important fact for the soundness of the outputs, however, the early versions of the algorithm uses a simple criterion for the order of the translation templates inferred. We need to assign confidence factors, i.e., weights, to these translation templates to have more accurate translations. Confidence factor assignment is done by using training data and collecting some statistical information. In the learning phase of the algorithm, each template is given a template number. Since translation is bidirectional, templates (specific templates without variables and generalized ones with variables) are assigned two weights, one for left to right usage and one for right to left usage of that template by using the translation examples. In addition to these, some template combinations are also assigned confidence factors in order to eliminate bad translation results. Translation accuracy is increased by using these weights. In the translation process, the output translations which have the highest weights are selected among all possibilities. Thus, it is ensured that the correct answer will be among these selected output.

The rest of the paper is organized as follows. Section 2 explains the confidence factor assignment process to the templates. Translation algorithm is described in Section 3. In Section 4 performance results of the system are provided. Section 5 concludes the paper and gives some future directions.

2 Methods for Assigning Confidence Factors

The translation templates are ordered according to the number of terminal symbols of the templates in the previous version of TTL algorithm [7]. However, this criteria is not sufficient for large systems, and we need another method where a statistical method is a powerful candidate, in order to improve the soundness of the translation process. Therefore, in the new version of the TTL algorithm, learning translation templates is followed by a confidence factor assignment process in which each rule and some rule combinations are assigned weights. Our main resource for assigning confidence factor is the training data that is used in

the learning of translation templates. This process has three fundamental parts: Confidence factor assignment to facts (i.e. specific templates without variables), rules (i.e. generalized templates in which the similarities are replaced with variables) and rule combinations. These three parts are explained in detail in the following sections.

Our translation process is bidirectional. In other words, it is possible to give an input sentence in language L_1 and obtain a translation in language L_2 and vice versa. Therefore we have templates that will be used for translation from L_1 to L_2, (left to right) and from L_2 to L_1(right to left).

2.1 Method for Assigning Confidence Factors to Facts

In this section confidence factor assignment to facts, which are the simplest case of this process, are discussed. We do not need to consider any other rule during this process and we use only the translation examples.

Consider the case that, $rule_k$ is a fact which will be used for left to right translation. Assume that, it is in the form of $X \Leftrightarrow Y$ and we have training pairs in the form of $trainpair(X_i, Y_i)$ then the confidence factor of $rule_k$ for left to right translation is evaluated as follows:

- $N1$ denotes the number of training pairs where X is a substring of X_i and Y is a substring of Y_i
- $N2$ denotes the number of training pairs where X is a substring of X_i and Y is not a substring of Y_i
- $confidencefactor_{rule_k} = \frac{N1}{N1+N2}$

If $rule_k$ is a fact which will be used for right to left translation, everything will be the same except definition of $N2$

- $N2$ denotes the number of training pairs where X is not a substring of X_i and Y is a substring of Y_i

Now, we illustrate how to find the confidence factor of a fact by giving an example. Let us assume that our all training pairs are as follows:

he come+s⇔gel+Hr
he go+s⇔git+Hr
book+s⇔kitap+lAr
pen+s⇔kalem+lAr

where +Hr and +lAr denote present tense and plural morphemes in Turkish. If we want to find the confidence factor of the rule **+s → Hr** (this rule is a fact and it will be used in left to right translation) which has been learned from these training pairs, we can find its confidence factor as follows:

$N1 = 2$ from pairs 1 and 2
$N2 = 2$ from pairs 3 and 4
$confidencefactor_{rule} = \frac{N1}{N1+N2} = \frac{2}{2+2} = 0.5$

If *rule* is **+s←** **+Hr**, (i.e. it is a fact and it will be used in right to left translation), we can find its confidence factor as follows:

$N1 = 2$ from pairs 1 and 2
$N2 = 0$ no such pair
$confidencefactor_{rule} = \frac{N1}{N1+N2} = \frac{2}{2+0} = 1.0$

It is possible to have the same confidence factor for left to right and right to left usage of the same rule, but it is more probable to have different values. For example, in the following example we have same confidence factors in both direction.

1) if *rule* is **come→ gel** (i.e. it is a fact and it will be used in left to right translation), and our translation examples are the same with the previous example. Then we will find confidence factor of *rule* as:

$N1 = 1$ from pair 1
$N2 = 0$ no such pair
$confidencefactor_{rule} = \frac{N1}{N1+N2} = \frac{1}{1+0} = 1.0$

2) if *rule* is **come← gel** (i.e. it is a fact and it will be used in right to left translation), we will find confidence factor of *rule* as:

$N1 = 1$ from pair 1
$N2 = 0$ no such pair
$confidencefactor_{rule} = \frac{N1}{N1+N2} = \frac{1}{1+0} = 1.0$

2.2 Method for Assigning Confidence Factors to Rules

Assigning confidence factor to a rule, (a template that has variables in it) is a more complicated task if we try to find the confidence factor of that rule completely. Therefore, if $rule_k$ has variables which will be unified with other rules in the translation phase then we will assign a partial confidence factor to this rule by considering the parts which do not include variables according to the confidence factor formula used in the previous section. In the translation process, the variables are bound using some other rules or facts, and we find the whole confidence factor of this rule by multiplying the confidence factors of all rules which are used to bind the variables. The following is an example for this:

If $rule_k$ is

X^{L_1}**+s⇔** X^{L_2}**+Hr** if $X^{L_1} \Leftrightarrow X^{L_2}$

and our training pairs are the same with the previous example. Since X^{L_1}**+s** can be a substring of left sides of all pairs and $\mathbf{X}^{L_2}$**+Hr** can be a substring of right sides of pairs 1 and 2 by assuming that the variables can match one or more tokens of the string (i.e. variables can not match empty string), we will get the following confidence factor for left to right usage:

$N1 = 2$ from pairs 1 and 2
$N2 = 2$ from pairs 3 and 4
$partialconfidencefactor_{rule_k} = \frac{N1}{N1+N2} = \frac{2}{2+2} = 0.5$

Since X^{L_2}+**Hr** can be a substring of right sides of pairs 1 and 2 and $\mathbf{X}^{L_1}$+**s** can be a substring of left sides of pairs all pairs by assuming that the variables can match one or more tokens of a string, we will find the following confidence factor for right to left usage:

$N1 = 2$ from pairs 1 and 2
$N2 = 0$ no such pair
$partialconfidencefactor_{rule_k} = \frac{N1}{N1+N2} = \frac{2}{2+0} = 1.0$

In the translation phase, these partial confidence factors are multiplied by the confidence factors of the rules replacing variables to calculate the real confidence factor of that translation output.

2.3 Method for Assigning Confidence Factors to Rule Combinations

The most complicated task of the procedure is the assignment confidence factors to rule combinations. The reason for considering these rule combinations is the following: Although some rules or facts are assigned high confidence factors when they are considered as single rules or facts, they may have a very low confidence factor when they are used with other rules or facts. The algorithm of this assignment process is given in Table 1. The algorithm in Table 1 is used only for left to right translation. This algorithm is repeated for right to left translation by replacing X^{L_1} with X^{L_2}.

Table 1. Algorithm for assigning confidence factor to rule combinations

For each training pair $X^{L_1} \Leftrightarrow X^{L_2}$
- Find all corresponding $\mathbf{Xs}^{L_2}$ for X^{L_1} from training pairs
- Find all translations (**Ts**) with their proofs (**Ps**) of X^{L_1} from translation templates where proofs show the rules used in the translation
- For each $T_i \in$ **Ts** do the following steps
  If $T_i \in$ **Ts** is the same as $X_j \in \mathbf{Xs}^{L_2}$
  - Assign confidence factor of the rule combination $P_i \in$ **Ps** as 1

  else
  - Find distances between T_i and each $X_j \in \mathbf{Xs}^{L_2}$
  - Choose the minimum distance **d** among these distances
  - Assign confidence factor of this rule combination $P_i \in$ **Ps** as
    $confidencefactor_{P_i} = \frac{1}{1+d}$

At this point, calculation of the minimum distance between a translation result, T_i, and a part of training pair, $X_j \in \mathbf{Xs}^{L_2}$, needs more explanation. First of all, X_j and T_i are assumed to be points whose coordinates are (Length of X_j, 0) and (Length of Similarities between X_j and T_i, Length of Differences between X_j and T_i) in a two-dimensional space, respectively. Then the distance is calculated by using the Euclidean formula for calculating the distance between two points:

$$distance = \sqrt{(Lengthof X_j - Lengthof Similarities)^2 + (Lengthof Differences)^2}$$

Assume that we have X^{L_1}=`you come+p` and we obtained $\mathbf{Xs}^{L_2}$= {`gel+DH+n, siz gel+DH+nHz`} and **Ts**={`gel+Hr+DH+n, gel+DH+nHz`} then confidence factors for rule combinations used to find translations in **Ts** are computed as follows:

1) For T_1=`gel+Hr+DH+n` where T_1 is found by using n rules $i_1, \cdots, i_n$, the confidence factor of the rule combinations $i_1, \cdots, i_n$ is calculated as:

- Find the distance between T_1 and X_1:
  - Since similarities between T_1 and X_1 are [gel,+DH,+n], the length of similarities is 3.
  - Differences between T_1 and X_1 are [[],[+Hr]], and the length of differences is 1, since length of [+Hr] is 1.
  - **d1**=$\sqrt{(3-3)^2 + (1)^2} = 1$
- Find the distance between T_1 and X_2:
  - Since similarities between T_1 and X_2 are [gel,+DH], the length of similarities is 2.
  - Differences between T_1 and X_2 are [([siz],[]),([],[+Hr]),([+nHz],[+n])] and the length of differences is 3, since length of [siz] is 1, length of [+Hr] is 1 and length of [+nHz] or [+n] is 1, giving a total of 3.
  - **d2**=$\sqrt{(4-2)^2 + (3)^2} = \sqrt{13}$
- **min(d1,d2)=d1**=1 and confidence factor$_{[i_1,\cdots,i_n]}$= $\frac{1}{1+1}$=0.5

2) For T_2=`gel+DH+nHz` where T_2 is found by using m rules $j_1, \cdots, j_m$, the confidence factor of the rule combinations $j_1, \cdots, j_m$ is calculated as:

- Find the distance between T_2 and X_1:
  - Since similarities between T_2 and X_1 are [gel,+DH] and the length of similarities is 2.
  - Differences between T_2 and X_1 are [([+n],[+nHz])] and the length of differences is 1, since length of [+n] or [+nHz] is 1.
  - **d1**=$\sqrt{(3-2)^2 + (1)^2} = \sqrt{2}$
- Find the distance between T_2 and X_2:
  - Since similarities between T_2 and X_2 are [gel,+DH,+nHz]] and length of similarities is 3.
  - Differences between T_2 and X_2 are [([siz],[])] and the length of differences is 1, since length of [siz] is 1.
  - **d2**=$\sqrt{(4-3)^2 + (1)^2} = \sqrt{2}$

– **min(d1,d2)=d1** or **d2** and confidence factor$_{[j_1,\cdots,j_m]} = \frac{1}{1+\sqrt{2}}$

Note that, the length of differences is calculated by choosing the maximum of lengths in difference pairs.

These rule combinations are represented as tree structures. For example if $rule_i$ has two variables that are bound to $rule_j$ and $rule_k$, then the root of the tree is assumed to be $rule_i$ and its children are $rule_j$ and $rule_k$. If $rule_j$ or $rule_k$ has variables then they become the root of that subtree and their children become the numbers of the rules that are used in the binding of their variables. This tree structure is formed recursively. The tree structure will be helpful during the translation process and its usage will be explained in the next section.

3 Translation Process by Using Confidence Factors

Translation process can be summarized by the four steps given in Table 2. We find all possible translations by using the templates obtained in our learning phase. Then these results are evaluated according to their weights. These weights come either directly from the weights of rules or rule combinations. After the evaluation of the results, the ones that have the highest weights are given as the output, and the ones with lowest weights are eliminated. Therefore the correct output is ensured to be among these selected outputs, and hopefully will be on the top of the selected outputs.

The second step of the algorithm is the most important part of the translation process. Finding the confidence factors of these results is not as simple as it seems. We need both the confidence factors of the rules and rule combinations which are calculated in the learning process. The details of these calculations are given in Table 3. The rules that are pertaining to the result are found and a tree structure is obtained from these rules as explained in Section 2.3. Then this tree structure is used for comparison. If the result does not match a rule combination that is assigned a weight in the learning phase, then the comparison continues among the subtrees.

Table 2. Translation Algorithm

– Find all possible translations and their proofs
– Find confidence factors of these results by using the confidence factors assigned in the confidence factor assignment process.
– If one result is found more than once with different weights use the average of all possibilities for confidence factor.
– Sort results according to the calculated confidence factors in descending order by using a sort algorithm

Table 3. Algorithm for calculating confidence factors of the translations

Find the translation output's confidence factor by using the previously calculated confidence factors of rule combinations

- Find the set of rule combinations (**R**) which are assigned confidence factors
- If $rp = R_i \in \mathbf{R}$ then cf_{result} =cf$_{R_i}$ where rp is the resulting proof
  else cf_{result} =cf$_{rp_{proof}}$* cf$_{rp_{child_1}}$ * cf$_{rp_{child_2}}$ * $\cdots$ * cf$_{rp_{child_n}}$
  where if child$_k$ is a fact ($fact_m$), then cf$_{child_m}$ =cf$_{fact_m}$
  else calculate recursively cf$_{child_k}$ as a tree

4 Performance Results

In this section, the results of the simulation on small corpora are summarized. A training set of examples has contained 488 sentences. Total number of the translation templates that are learned in the learning phase is 4723. In the confidence factor assignment process 4723 templates for left to right usage (from English to Turkish), and 4723 templates for right to left usage (from Turkish to English) are assigned confidence factors. 55845 rule combinations for left to right usage and 53676 rule combinations for right to left usage are assigned confidence factors. Therefore, we obtained a total of 118967 confidence factor assignments.

Table 4. Performance Results

Type of data	Percentage of correct results in translations	Percentage of incorrect results in translations	Percentage of correct results in top 5 without weights	Percentage of correct results in top 5 with weights
Sentences selected from training data	42.0	58.0	44.0	80.0
New sentences not appearing in training data	33.0	67.0	40.0	60.0

In the translation process, we used two groups of sentences to evaluate the performance of the results. The first group of sentences are randomly selected from training data and the second group of sentences are the new sentences which do not occur in the training data. The results are obtained by using the previously assigned weights and they are sorted in ascending order according to these weights. We also produced the outputs without using the weights of the templates for comparison purposes. Then they are sent to the generator to

obtain surface forms from the lexical forms. In Table 4 the results with weights and without weights are summarized. The columns denote the percentage of the correct translations among all the results, percentage of the incorrect translations, and percentage of the correct translations seen in the top five results, respectively.

5 Conclusion and Future Work

In this paper, we have presented a statistical model for assigning confidence factors to the translation templates learned by the translation model offered in Cicekli [7, 4]. This translation model learns general translation patterns from the given translation examples by using analogy principle.

The early versions of the algorithm, translation templates are sorted according to their specificities (i.e., the number of terminals in templates). Although this way of sorting gives correct results, the accuracy was not high enough.The major contribution of this paper is assigning confidence factors to templates in order to improve the accuracy. Assigning confidence factor to these rules depends on the statistical data collected from translation examples which are assumed to be grammatically correct. As mentioned before, in the translation process, the output translations which have the highest weights are selected among all possibilities. Thus, it is ensured that the correct answer will be among these selected output and at the top of the list.

The algorithm is tested on Turkish and English for illustration purposes, but it is applicable to any pair of languages. On a small set of data, learning and translation times are reasonable enough. The accuracy of the results are promising. We need to test it on very large corpora. Thus, we are trying to form a large corpus for this purpose. The learning process on a large corpus will take a considerable amount of time, but it can be tolerated since it will be done only once and increase the translation accuracy.

In the future, the system accuracy can be increased by using a human assistance for the verification of the templates, morphological analysis etc. However, in order to fully automate the system, it will be better to use some additional reliable tools for parallel text alignment, disambiguation, etc.

References

1. R. D. Brown. Example-Based Machine Translation in the Pangloss System. In *Proceedings of COLING-96*, 1996.
2. P. F. Brown. Aligning Sentences in Parallel Corpora. In *Proceedings of the 29th Annual Meeting of th ACL*, pp:169-176, 1991.
3. P. F. Brown. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, pp:233-311, 1993.
4. I. Cicekli, and H. A. Güvenir. Learning Translation Rules From A Bilingual Corpus. *Proceedings of the 2nd International Conference on New Methods in Language Processing (NeMLaP-2)*, Ankara, Turkey, September 1996, pp:90-97.

5. B. Collins, and P. Cunningham. A Methodology for Example-Based Machine Translation. *Trinity College*, Dublin, 1995
6. W. A. Gale, and K.W. Church. A Program for Aligning Sentences in Bilingual Corpora. In *Proceedings of the 29th Annual Meeting of th ACL*, pp:177-184, 1991.
7. H. A. Güvenir, and I. Cicekli. Learning Translation Templates from Examples. *Information Systems* (accepted to be published).
8. H. A. Güvenir, and A. Tunc. 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.
9. H. Kaji, Y. Kida, and Y. Morimoto. Learning Translation Templates from Bilingual Text. In *Proceedings of COLING-92*, pp:672-678, 1992.
10. H. Kitano. A Comprehensive and Practical Model of Memory-Based Machine Translation. In Ruzena Bajcsy (Ed.) *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Volume 2, 1993, pp: 1276-1282.
11. D. L. Medin, and M. M. Schaffer. Context Theory of Classification Learning. *Psychological Review*, 85, 1978, pp:207-238.
12. M. A. Nagao. Framework of a Mechanical Translation between Japanese and English by Analogy Principle. *Artificial and Human Intelligence*, A. Elithorn and R. Banerji (eds.), NATO Publications, 1984.
13. V. Sadler. Working with Analogical Semantics: Disambiguation Techniques in DLT. *Foris Publications*, Dodrecht, Netherlands, 1989.
14. V. Sadler, and R. Vendelmans. Pilot Implementation of a Bilingual Knowledge Bank. In *Proceedings of COLING-90*, pp:449-451, 1990.
15. E. Sumita, H. Iida, and H. Kohyama. Translating with Examples: A New Approach to Machine Translation. In *Proceedings Third International Conference on Theoretical and Methodological issues in Machine Translation of Natural Language*, 1990.
16. E. Sumita, and Y. Tsutsumi. A Translation Aid System using flexible Text Retrieval Based on Syntax Matching. *TRL Res. Report TR-87-1019*, Tokyo Research Laboratory, IBM, Tokyo, Japan, 1988.

Assigning Confidence Factors to Translation Templates*

Zeynep Öz

Department of Computer Engineering and Information Sciences
Bilkent University
Ankara, 06533, Turkey

Abstract. This paper proposes a method for assigning confidence factors to translation templates learned by a TTL (*Translation Template Learner*) algorithm. The TTL algorithm learns lexical level correspondences by using analogical reasoning between two translation examples. The sentences used as translation examples have similar and different parts in the source language which must correspond to the similar and different parts in the target language. Therefore these correspondences are learned as translation templates. The learned translation templates are used in the translation of other sentences, and they are bidirectional. However, we need to assign confidence factors to these translation templates to order translation results with respect to confidence factors. Confidence factor assignment is done by using training data and collecting some statistical information. In the confidence factor assignment process both specific templates without variables and generalized ones with variables are considered. In addition to these assignments, some template combinations are also assigned confidence factors in order to eliminate certain combinations resulting bad translation.
Keywords: Exemplar Based Machine Learning, Example-Based Machine Translation, Confidence Factor Assignment.

1 Introduction

Corpus-based machine translation is a good method for avoiding problems like tractability, scalability and performance problems in the traditional machine translation (MT). *Statistical* and *example-based* machine translation (EBMT) are two main approaches of corpus-based machine translation. EBMT, originally proposed by Nagao [5], is based on the idea of performing translation by imitating translation examples of similar sentences. There are three fundamental phases in EBMT:

(i) Finding the correspondence of units in a bilingual text
(ii) A method that retrieves the best matches from previous translation examples
(iii) Producing the translation of the given input by using these examples

Statistical MT techniques use statistical metrics to choose the best structures in the target language among all possible candidates. Therefore, these techniques are useful as far as the method in (ii) is considered. This fact is the motivation behind the method

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

for developing a machine translation system that is a combination of statistical MT and EBMT.

Exemplar-based learning which is originally proposed by Medin and Schaffer [6] uses previous examples to learn from new ones. This way of learning stores the examples in memory without any change in the representation. The characteristic examples stored in the memory are called exemplars.

In EBMT, before translating an input sentence, the system should be provided the correspondences between the source and target languages; however this is not a simple task and has not been given enough consideration by the current EBMT systems. Manual encoding of the translation rules has been achieved by Kitano [4], however when the corpus is very large, it becomes a complicated and error-prone task. Therefore the acquisition problem is taken as a machine learning task by Cicekli and Güvenir [2, 3] for the automation of this process. Cicekli and Güvenir [2, 3] offered a technique which stores exemplars in the form of templates that are generalized exemplars. Templates are formed by using the example translation pairs. These templates are learned by using translation examples and finding the correspondences between the patterns in the source and target languages.

The heuristic of the translation template learning (TTL) [2, 3] algorithm can be summarized as follows: Given two translation pairs, if there are some similarities(differences) in the source language, then the corresponding sentences in the target language must have similar(different) parts, and they must be translations of the similar(different) parts of the sentences in the source language. If the sentences do not exhibit any similarity, then no correspondences are inferred obviously. Differing or similar parts are replaced with variables to get a template which is a generalized exemplar by this method. Although in the first version of the algorithm, the exemplars can be obtained if and only if the number of the similar or differing components between the source language sentences are equal to the number of the similar or differing components between the target language sentences, this restriction is removed by the new heuristics in the final version of the algorithm.

Translation examples are stored as a list of string formed by strings of root words and morphemes. In other words, the lexical level representation of the sentences are used. For example the sentence pair **i went to school⇔ben okula gittim** is stored as:

i go+p to school⇔ben okul+yA git+DH+m

where *i, go, to, school* denote root words and *+p* denotes the past tense morpheme in English sentence, and *ben, okul, git* denote root words and *+yA, +DH, +m* denote dative, past tense and first singular person morphemes in Turkish sentence.

This representation of translation examples is suitable for learning algorithm. If we used surface level representation, the number of correspondences would be decreased and we could learn less number of generalized exemplars.

The following translation pairs (from Cicekli [2] given in English and Turkish illustrates the heuristic:

I give+p the ticket to Mary↔ Mary'+yA bilet +yH ver+DH+m
I give+p the pen to Mary ↔ Mary'+yA kalem +yH ver+DH+m.

The similarities between the translation examples are underlined. The similarities in English are represented as **I give+p the** X^{L_1} **to Mary**, and the corresponding similarities in Turkish as **Mary'+yA** X^{L_2}**+yH ver+DH+m** by replacing differences by variables. According to the heuristic, these similarities should correspond to each other. Here, X^{L_1} denotes a component that can be replaced by any appropriate structure in English and X^{L_2} refers to its translation in Turkish. In addition to this, it is also inferred that *ticket* is the translation of *bilet* and *pen* is the translation of *kalem*. This shows that it is possible to learn more than one templates by using two translation examples.

The order of the translation templates that will be used for the translation of new sentences is an important fact for the soundness of the outputs, however, the early versions of the algorithm uses a simple criterion for the order of the translation templates inferred. We need to assign confidence factors, i.e., probabilities, to these translation templates to have more accurate translations. Confidence factor assignment is done by using training data and collecting some statistical information. In the learning phase of the algorithm, each template is given a template number and assigned two probabilities, one for left to right usage and one for right to left usage of that template by using the translation examples. In the confidence factor assignment process, specific templates without variables and generalized ones with variables are considered. In addition to these assignments, some template combinations are also assigned confidence factors in order to eliminate bad translation results. Translation accuracy is increased by using these probabilities. In the translation process, the output translations which have the highest probabilities are selected among all possibilities. Thus, it is ensured that the correct answer will be among these selected output.

The rest of the paper is organized as follows. Section 2 explains the confidence factor assignment process to facts, rules, and rule combinations. Translation algorithm is described in Section 3. Section 4 concludes the paper and gives some future directions.

2 Confidence Factor Assignment

The previous version of TTL algorithm uses only the number of terminal symbols in templates for the order of the translation templates. However, we need some more complicated methods, like statistical ones, in order to improve the soundness of the translation process. Therefore, in the new version of the TTL algorithm, learning translation templates is followed by a confidence factor assignment process in which each rule and some rule combinations are assigned some probabilities. This process has three fundamental parts: Confidence Factor Assignment to facts (i.e. templates without variables), rules and rule combinations. These three parts are explained in detail in the following sections.

Our translation process is bidirectional. In other words, it is possible to give an input sentence in language L_1 and obtain a translation in language L_2 and vice versa.

Therefore we have templates that will be used for translation from L_1 to L_2, (left to right) and from L_2 to L_1(right to left).

2.1 Confidence Factor Assignment to Facts

In this section confidence factor assignment to facts are discussed. Facts are the simplest case that can be thought of in confidence factor assignment process, since we do not need to consider any other rule.

Consider the case that, $rule_k$ is a fact which will be used for left to right translation. Assume that, it is in the form of $X \Leftrightarrow Y$ and we have training pairs in the form of $trainpair(X_i, Y_i)$ then

- $N1$ denotes the number of training pairs where X is a substring of X_i and Y is a substring of Y_i
- $N2$ denotes the number of training pairs where X is a substring of X_i and Y is not a substring of Y_i

$$confidencefactor_{rule_k} = \frac{N1}{N1+N2}$$

If $rule_k$ is a fact which will be used for right to left translation, everything will be the same except definition of $N2$

- $N2$ denotes the number of training pairs where X is not a substring of X_i and Y is a substring of Y_i

Consider the following example for the illustration:

If $rule_k$ is **you→ +n**, (i.e. it is a fact and it will be used in left to right translation), and our training pairs are the followings:

you come+p⇔gel+DH+n
you go+p⇔git+DH+n
you come+p⇔gel+DH+nHz
you go+p⇔git+DH+nHz

Then we will find the confidence factor of $rule_k$ as:
$N1 = 2$ from pairs 1 and 2
$N2 = 2$ from pairs 3 and 4

$$confidencefactor_{rule_k} = \frac{N1}{N1+N2} = \frac{2}{2+2} = 0.5$$

If $rule_k$ is **you← +n**, (i.e. it is a fact and it will be used in right to left translation) Then we will find confidence factor of $rule_k$ as:
$N1 = 2$ from pairs 1 and 2
$N2 = 0$ no such pair

$$confidencefactor_{rule_k} = \frac{N1}{N1+N2} = \frac{2}{2+0} = 1.0$$

It is possible to have the same confidence factor for left to right and right to left usage of the same rule, but it is more probable to have different values.

2.2 Confidence Factor Assignment to Rules

Assigning confidence factor to rule, (templates that have variables in them) is a more complicated task if we try to find the confidence factor of that rule completely. Therefore, if $rule_k$ has variables which will be unified with other rules in the translation phase then we will assign a partial confidence factor to this rule by considering the parts which do not include variables according to the confidence factor formula used in the previous section. In the translation process, the variables are bound using some other rules or facts, and we find the whole confidence factor of this rule by multiplying the confidence factors of all rules which are used to bind the variables. The following is an example for this:

If $rule_k$ is **you** X^{L_1}**+p**$\Leftrightarrow$ X^{L_2}**+DH+n** if $X^{L_1} \Leftrightarrow X^{L_2}$

and our training pairs are the same with the previous example. Since **you** X^{L_1}**+p** can be a substring of left sides of all pairs and $\mathbf{X}^{L_2}$**+DH+n** can be a substring of right sides of pairs 1 and 2 by assuming that the variables can match one or more tokens of the string, we will get the following confidence factor for left to right usage:

$N1 = 2$ from pairs 1 and 2

$N2 = 2$ from pairs 3 and 4

$partialconfidencefactor_{rule_k} = \frac{N1}{N1+N2} = \frac{2}{2+2} = 0.5$

Since X^{L_2}**+DH+n** can be a substring of right sides of pairs 1 and 2 and **you** $\mathbf{X}^{L_1}$**+p** can be a substring of left sides of pairs 1 and 2 by assuming that the variables can match one or more tokens of the string, we will find the following confidence factor for right to left usage:

$N1 = 2$ from pairs 1 and 2

$N2 = 0$ no such pair

$partialconfidencefactor_{rule_k} = \frac{N1}{N1+N2} = \frac{2}{2+0} = 1.0$

2.3 Confidence Factor Assignment to Rule Combinations

The most complicated task of the procedure is assigning confidence factors to rule combinations. The reason for considering these rule combinations is the following: Although some rules or facts are assigned high confidence factors when they are considered as single rules or facts, they may have a very low confidence factors when they are used with other rules or facts. The algorithm of this assignment process is given in Table 1. The algorithm in Table 1 is used only for left to right translation. This algorithm is repeated for right to left translation by replacing X^{L_1} with X^{L_2}.

At this point, calculation of the minimum distance between a translation result, T_i, and a part of training pair, $X_j \in \mathbf{Xs}^{L_2}$, needs more explanation. First of all, X_j and T_i are assumed to be points whose coordinates are (Length of X_j, 0) and (Length of

For each training pair $X^{L_1} \Leftrightarrow X^{L_2}$
  Find all corresponding $\mathbf{Xs}^{L_2}$ for X^{L_1} from training pairs
  Find all translations (**Ts**) with their proofs (**Ps**) of X^{L_1} from translation templates
  For each $T_i \in \mathbf{Ts}$ do the following steps
    If $T_i \in \mathbf{Ts}$ is the same as $X_j \in \mathbf{Xs}^{L_2}$
      If the rule(s) in proof $P_i \in \mathbf{Ps}$ obtained is not assigned a confidence factor previously then assign confidence factor of this rule combination as 1.
    Else
      Find distances between T_i and each $X_j \in \mathbf{Xs}^{L_2}$
      Choose the minimum distance **d** among these distances
      Assign confidence factor of this rule combination $P_i \in \mathbf{Ps}$ as

$$confidencefactor_{P_i} = \frac{1}{1+d}$$

Table 1. Algorithm for assigning confidence factor to rule combinations

Similarities between X_j and T_i, Length of Differences between X_j and T_i) in a two-dimensional space, respectively. Then the distance is calculated by using the Euclidean formula for calculating the distance between two points:

$$distance = \sqrt{(LengthofX_j - LengthofSimilarities)^2 + (LengthofDifferences)^2}$$

Assume that we have X^{L_1}=`you come+p` and we obtained $\mathbf{Xs}^{L_2}$= {`gel+DH+n, siz gel+DH+nHz`} and **Ts**={`gel+Hr+DH+n, gel+DH+nHz`} then
For T_1=`gel+Hr+DH+n`:

If T_1 is found by using n rules $i_1, \cdots, i_n$

Similarities between T_1 and X_1 are [gel,+DH,+n] and the length of similarities is 3.

Differences between T_1 and X_1 are [[],[+Hr]] and the length of differences is 1, since length of [+Hr] is 1.

d1=$\sqrt{(3-3)^2 + (1)^2} = 1$

Similarities between T_1 and X_2 are [gel,+DH] and the length of similarities is 2.

Differences between T_1 and X_2 are [([siz],[]),([],[+Hr]),([+nHz],[+n])] and the length of differences is 3, since length of [siz] is 1, length of [+Hr] is 1 and length of [+nHz] or [+n] is 1, giving a total of 3.

d2=$\sqrt{(4-2)^2 + (3)^2} = \sqrt{13}$

min(d1,d2)=d1=1 and confidence factor$_{[i_1,\cdots,i_n]}$=0.5

For T_2=`gel+DH+nHz`:

If T_2 is found by using m rules $j_1, \cdots, j_m$

Similarities between T_2 and X_1 are [gel,+DH] and the length of similarities is 2.

Differences between T_2 and X_1 are [([+n],[+nHz])] and the length of differences is 1, since length of [+n] or [+nHz] is 1.

d1=$\sqrt{(3-2)^2+(1)^2} = \sqrt{2}$

Similarities between T_2 and X_2 are [gel,+DH,+nHz]] and length of similarities is 3.

Differences between T_2 and X_2 are [([siz],[])] and the length of differences is 1, since length of [siz] is 1.

d2=$\sqrt{(4-3)^2+(1)^2} = \sqrt{2}$

min(d1,d2)=d1 or **d2** and confidence factor$_{[j_1,\cdots,j_m]} = \frac{1}{1+\sqrt{2}}$

Note that, the length of differences is calculated by choosing the maximum of lengths in difference pairs.

These rule combinations are represented as tree structures. For example if $rule_i$ has two variables that are bound to $rule_j$ and $rule_k$, then the root of the tree is assumed to be $rule_i$ and the children of it are $rule_j$ and $rule_k$. If $rule_j$ or $rule_k$ has variables then they become the root of that subtree and their children become the numbers of the rules that are used in the binding of their variables. This tree structure is formed recursively. The tree structure will be helpful during the translation process and its usage will be explained in the next section.

3 Translation Process

Translation process can be summarized by the four steps given in Table 2. We find all possible translations by using the templates obtained in our learning phase. Then these results are evaluated according to their probabilities. These probabilities come either directly from the rules' probabilities or from the rule combinations' probabilities. After the evaluation of the results, the ones that have the highest probabilities are given as the output, and the ones with lowest probabilities are eliminated. Therefore the correct output is ensured to be among these selected outputs, and hopefully will be on the top of the selected outputs.

The second step of the algorithm is the most important part of the translation process. Finding the confidence factors of these results is not as simple as it seems. We need both the confidence factors of the rules and rule combinations which are calculated in the learning process. The details of these calculations are given in Table 3. The rules that are pertaining to the result is found and a tree structure is obtained from these rules as explained in Section 2.3. Then this tree structure is used for comparison. If the result does not match a rule combination that is assigned a probability in the learning phase, then the comparison is continues among the subtrees.

4 Performance Results

In this section, the results of the simulation on small corpora will be summarized. A training set of examples has contained 82 sentences. These sentences are used to learn templates for present and past tenses. In the learning phase, 692 templates are obtained. In the confidence factor assignment process 692 templates for left to right usage (from English to Turkish), and 692 templates for right to left usage (from Turkish to English) are assigned confidence factors. 3752 rule combinations for left to right usage

- Find all possible translations and their proofs
- Find confidence factors of these results by using the confidence factors assigned in the confidence factor assignment process.
- If one results is found more than once with different probabilities use the average of all possibilities for confidence factor.
- Sort results according to the calculated confidence factors in descending order by using a sort algorithm

Table 2. Translation Algorithm

Find the translation output's confidence factor by using the previously calculated rule combinations' confidence factors

Find the set of rule combinations (**R**) whose lengths are equal to the length of resulting proof, (rp).

If $rp = R_i \in \mathbf{R}$ then cf_{result} =cf_{R_i}

else cf_{result} =$\text{cf}_{rp_{root}} * \text{cf}_{rp_{child_1}} * \text{cf}_{rp_{child_2}} * \cdots * \text{cf}_{rp_{child_n}}$

if child_k is a fact, fact_m, then cf_{child_m} =confidence factor_{fact_m}

else calculate recursively cf_{child_k} as a tree

Table 3. Algorithm for calculating confidence factors of the translations

5009 rule combinations for right to left usage are assigned confidence factor. Therefore, we obtained a total of 10145 confidence factor assignment.

Instance	Total	Correct	Incorrect	Eliminated
Unseen 1	145	22	123	111
Unseen 2	84	59	25	6
Unseen 3	79	9	70	60
Unseen 4	30	19	11	2
Unseen 5	80	57	23	2
Unseen 6	254	20	234	216
Seen 1	37	18	19	11
Seen 2	76	20	56	44
Seen 3	72	48	24	6
Seen 4	42	13	29	24
Seen 5	173	40	133	76
Seen 6	235	40	195	177

Table 4. Performance Results

In the translation process the results for seen (sentences which are used in the training set) and unseen (new sentences which does not occur in the training set) instances are found. The results are sorted according to their weights. Then they are

sent to the generator to obtain surface forms from the lexical forms. Some of the incorrect results are eliminated during the generation process. In Table 4 some of these results are provided. The columns denote the instances used for testing, total number of translations produced, the number of correct translations among all the results, the number of incorrect translations, and the number of translations eliminated during the generation of surface forms from the lexical forms, respectively. It should be noted that, these eliminated results only involve the incorrect translations. In some cases, the number of incorrect translations seem to be high, but generally their probabilities are lower than the correct results, so the top results include the correct translations rather than the incorrect ones.

5 Conclusion and Future Work

In this paper, we have presented a statistical model for assigning confidence factor to the translation templates learned by the translation model offered in Cicekli [2, 3]. This translation model learns general translation patterns from the given translation examples by using analogy principle.

If we guarantee that the given examples are grammatically correct, the accuracy of the translation templates learned by this approach is very high. The correctness of the rules determines the correctness of the translation results. We do not need to do any other operation for the translation output, all task is automatically carried out by translation templates. If the training examples used are large enough, (i.e large enough to contain examples which reflect the discourse function alternations and other kinds of linguistic complexities),then it is possible to learn templates which can be used in the translation of these kinds of sentences.

The early versions of the algorithm, translation templates are sorted according to their specificities (i.e., the number of terminals in templates). Although this way of sorting gives correct results, the accuracy was not high enough.The major contribution of this paper is the improvement in the accuracy of the rules learned. Assigning confidence factor to these rules depends on the statistical data collected from translation examples which are assumed to be grammatically correct. As mentioned before, in the translation process, the output translations which have the highest probabilities are selected among all possibilities. Thus, it is ensured that the correct answer will be among these selected output and at the top of the list.

The algorithm is tested on Turkish and English for illustration purposes, but it is applicable to any pair of languages. On a small set of data, learning and translation times are reasonable enough. The accuracy of the results are promising. We need to test it on a very large corpora. Thus, we are trying to form a large corpus for this purpose. The learning process on a large corpus will take a considerable amount of time, but it can be tolerated since it will be done only once. In the previous version of the algorithm, i.e., before using confidence factors, after learning the translation templates, the translation process was very fast. However, using confidence factors slowed down the translation process up to a certain level. There is a trade off between speed and the accuracy of the translation output, so it worths sacrificing from speed in order to increase the accuracy.

In the future, the system accuracy can be increased by using a human assistance for the verification of the templates, morphological analysis etc. However, in order to fully automate the system, it will be better to use some additional reliable tools for parallel text alignment, disambiguation, etc.

References

1. H. A. Guvenir, A. Tunc Corpus-Based Learning of Generalized Parse Tree Rules for Translation 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
2. H. A. Guvenir, I. Cicekli Learning Translation Templates from Examples *Proceedings of the 6th Annual Workshop on Information Technologies and Systems (WITS'96)*,Cleveland, Ohio, USA, December 1996, pp:112-121
3. I. Cicekli, H. A. Guvenir Learning Translation Rules From A Bilingual Corpus *Proceedings of the 2nd International Conference on New Methods in Language Processing (NeMLaP-2)*, Ankara, Turkey, September 1996, pp:90-97.
4. H. Kitano A Comprehensive and Practical Model of Memory-Based Machine Translation. In Ruzena Bajcsy (Ed.) *Proceedings of the Thirteenth International Joint Conference on Artificial Intelligence*, Morgan Kaufmann Volume 2, 1993, pp: 1276-1282.
5. M. A. Nagao Framework of a Mechanical Translation between Japanese and English by Analogy Principle *Artificial and Human Intelligence*, A. Elithorn and R. Banerji (eds.), NATO Publications, 1984.
6. D. L. Medin and M. M. Schaffer Context Theory of Classification learning *Psychological Review*, 85, 1978, pp:207-238.

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

2. İlgili Araştırma Grubu: EEEAG

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.

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

10. Proje ile ilgili Yayınlar (makale, tebliğ):
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.

11. Proje Sonuçlarının Gizlilik Durumu: Gizli Değil