T.C

BAHÇEŞEHİR ÜNİVERSİTESİ

IMPROVING THE QUALITY OF THE TURKISH ADDRESS RECORDS BY USING LEVENSHTEIN DISTANCE ALGORITHM

Master's Thesis

ÖZGÜR TUFAN

İSTANBUL, 2008

T.C

BAHÇEŞEHİR ÜNİVERSİTESİ

THE INSTITUTE OF SCIENCE

COMPUTER ENGINEERING

IMPROVING THE QUALITY OF THE TURKISH ADDRESS RECORDS BY USING LEVENSHTEIN DISTANCE ALGORITHM

Master's Thesis

ÖZGÜR TUFAN

Supervisor: ASST. PROF. DR. ORHAN GÖKÇÖL

İSTANBUL, 2008

T.C.

B A H Ç E Ş E H İ R Ü N İ V E R S İ T E S İ

INSTITUTE OF SCIENCE

COMPUTER ENGINEERING

Name of the thesis: Improving the Quality of the Turkish Address Records by using Levenshtein Distance Algorithm
Name/Last Name of the Student: Özgür Tufan
Date of Thesis Defense: 20 November 2008

The thesis has been approved by the Institute of Science.

Prof. Dr. Bülent ÖZGÜLER
Director

_____

I certify that this thesis meets all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Bülent ÖZGÜLER
Program Coordinator

_____

This is to certify that we have read this thesis and that we find it fully adequate in scope, quality and content, as a thesis for the degree of Master of Science.

Examining Committee Members	Signature
Asst. Prof. Dr. Orhan GÖKÇÖL	_____
Asst. Prof. Dr. Olcay KURŞUN	_____
Asst. Prof. Dr. M. Alper TUNGA	_____

ACKNOWLEDGEMENT

This thesis is dedicated to **my deceased grandfather; Ömer Lütfü Odabaş** and to **my beloved family**.

I would like to express my gratefulness to my supervisor **Asst. Prof. Dr. Orhan GÖKÇÖL** for his guidance and help on many topics.

And special thanks to my girlfriend **Fulya Sayım** for her limitless support, encouragement and faith on me.

ABSTRACT

IMPROVING THE QUALITY OF THE TURKISH
ADDRESS RECORDS BY USING
LEVENSHTEIN DISTANCE ALGORITHM

Tufan, Özgür

Computer Engineering

Supervisor: Asst. Prof. Dr. Orhan Gökçöl

November, 2008, 58 Pages

Address is the place where someone can be found when others look for him or her. This basic notion seems simple but its accuracy and consistency are very important. The cost of inaccurate addresses which are used by companies as a basic way of contacting to their customer is quite rising. There occur mistakes in address writing because of the fact that there is no unique standardization constituted for all addresses in Turkey. Turkish addresses are mostly written in the standard of address writers' mind in this situation. Frequently done mistakes in this address writing are not using standard abbreviations for address components such as street, road, parish and using meaningless shortening on city, county or other address components.

The aim of this study is finding these mistakes and improving the address quality with verification of addresses. First of all to do this, nonstandard abbreviations and meaningless characters are determined and replaced with true ones which are specified before. An algorithm is needed that can find similarity between the words to find similar words for the components of addresses such as city, county, districts or parishes which are shortened

inaccurately. There are several algorithms in the literature for this purpose. In this study, an approach based on the Levenshtein Distance Algorithm (LDA) is used. Levenshtein distance is a metric for measuring the amount of difference between two sequences. Although LDA is used as a core algorithm to find similarity between words, another faster algorithm which is the modified version of LDA is also used. Addresses are classified from zero to five according to their components after abbreviation replacement operations are accomplished. Then special situation on address components are determined.

Reference dataset provided by PTT is used on required address components with the algorithm which is more effective for that specified component and then improvement process is completed. An application is developed to do the address quality improvement. By using the application it is possible to do single or bulk addresses improvements and to compare the results of the different correction techniques. The incorrect address sets taken from a financial company are used to test the performance of the developed application. Addresses are reclassified according to their last values after the improvement process. As a result of all these processes, improvement rates up to 90 percent are reached for some address classes.

Keywords: String Similarity; Data Cleaning; Address Verification; Text Recognition and Correction; Levenshtein Distance.

ÖZET

LEVENSHTEIN UZAKLIK ALGORİTMASI KULLANILARAK
TÜRK SOKAK ADRESLERİNİN
KALİTESİNİN İYİLEŞTİRİLMESİ

Tufan, Özgür

Bilgisayar Mühendisliği

Tez Danışmanı: Yrd. Doç. Dr. Orhan Gökçöl

Kasım, 2008, 58 Sayfa

Adres bir kimsenin arandığında bulunabileceği yerdir. Bu temel kavram basit gibi görünse de doğruluğu ve tutarlılığı çok önemlidir. Günümüz dünyasında şirketlerin müşterilerine ulaşmasının temel yolu olarak kullandıkları adreslerin yanlış olmasının getirdiği mali yük oldukça fazladır. Adres yazımında yanlışlıklar Türkiye' deki bütün adresler için tam bir standart oluşturulamamasından kaynaklanmaktadır. Bu durumda kişiler adresleri kendi kafalarındaki standarda göre yazmaktadır. Bu yazım şeklinde en çok yapılan hatalar sokak, mahalle ve cadde gibi bileşenler için standart dışı kısaltmalar kullanılması; il,ilçe veya diğer adres bileşenlerinde anlamsız kısaltmalar oluşturulmasıdır.

Bu çalışmanın amacı bu tip hataların tespit edilip düzeltilerek adres kalitesinin iyileştirilmesidir. Bunun için öncelikle adreslerdeki standart dışı kısaltmalar ve anlamsız karakterler tespit edilip, daha önceden belirlenmiş olan doğrularıyla değiştirilir. İl, ilçe, semt ve ya mahalle gibi bileşenlerde yapılmış kısaltmalar için ise kelimeler arasındaki benzerlikleri

algılayabilcek bir algoritmaya ihtiyaç vardır. Literatürde bu ihtiyaca yönelik birçok algoritma vardır. Bu çalışmada Levenshtein uzaklık algoritmasını (LUA) esas alan bir yaklaşım kullanılmıştır. Levenshtein uzaklığı iki kelime arasındaki harf değişiklik miktarını ölçebilen bir yapıdır . Her ne kadar bu çalışmada kelime benzerliklerinin ölçümü için kullanılan temel algortima LUA olsa da, LUA üzerinde değişiklikler yapılıp elde edilen daha hızlı bir uzaklık algoritması da kullanılmıştır. Adreslerdeki standart dışı kısaltmaların değiştirilmesinden sonra adresler bileşenlerine göre sıfırdan beşe kadar altı sınıfa ayrılır. Daha sonra bu adresler üzerindeki özel durumlar tespit edilir.

Gerekli bileşenler üzerinde PTT tarafından sağlanan referans adresler kullanılarak o bileşen için daha etkili olan algoritma uygulanır ve iyileştirme işlemi tamamlanır. Adres kalitesi iyileştirmesini gerçekleştirmek için bir uygulama geliştirilmiştir. Uygulama kullanılarak tekil ve ya çoklu adres iyileştirmesi gerçekleştirmek ve farklı tekniklerin sonuçlarını karşılaştırılabilmek mümkündür. Geliştirilen uygulamanın performansını ölçmek için bir finans şirketinden alınan hatalı adres kümesi kullanılmıştır. İyileştirme işleminin sonucunda adresler sahip oldukları son değerlere göre tekrar sınıflanır. Bütün işlemlerin tamamlanmasından sonra bazı adres sınıflarında yüzde 90a varan iyileştirme oranlarına ulaşılmıştır.

Anahtar Kelimeler: Metin Benzerliği; Veri Temizliği; Adres Doğrulaması ; Kelime Tanıma ve Doğrulama; Levenshtein Uzaklığı.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

American National Standards Institute	:	ANSI
Artificial Neural Networks	:	ANN
Common Language Runtime	:	CLR
Customer Relations Management	:	CRM
Daitch-Mokotoff	:	D-M
European Computer Manufacturers Association	:	ECMA
Expectation-Maximization	:	EM
Extensible Markup Language	:	XML
Fast Distance	:	FD
General Directorate of Turkish Post	:	PTT
Integrated Development Environment	:	IDE
International Organization for Standardization	:	ISO
Levenshtein Distance Algorithm	:	LDA
Levenshtein Uzaklık Algoritması	:	LUA
Natural Language Processing	:	NLP
New York State Identification and Intelligence System	:	NYSIIS
Structured Query Language	:	SQL
University of California, Irvine	:	UCI

LIST OF SYMBOLS

Big O notation	:	O
Element of row i, column j of d matrix	:	`d[i,j]`

1. INTRODUCTION

Every day we go somewhere, do something or at least live at home. Every place we go, see and live have a common point; they all have an address.

Addresses may vary according to its source. Especially after internet joins our life, they can be classified in two main topics: street addresses and electronic addresses. Electronic addresses are not concern of this study; this study will be focusing on street addresses, their structure and addressing issues.

1.1 USE OF ADDRESSES IN DAILY LIFE

An **address** is a code and abstract concept expressing the fixed location of a home, business or other building on the earth's surface. (Wikipedia 2008)

Addresses have several functions:

i. Providing a means of physically locating a building, especially in a city where there are many buildings and streets,
ii. Identifying buildings as the end points of a postal system,
iii. A social function: someone's address can have a profound effect on their social standing,
iv. As parameters in statistics collection, especially in census-taking or the insurance industry.

Addresses take very important place especially in business applications such as financial services, insurance industry, shipments, marketing, telecommunication, government and many more (maybe all). This is why the address data is very valuable and quality of data has a major importance and not every address is good enough to be used for this kind of business applications. Real word data is dirty (Hernandez and Stolfo 1997) and using dirty data in business results with economical loss. So it needs cleaning and verification before it is used in business processes.

1.2 PROBLEM DEFINITON

Addresses are written by people and if there is human factor in any kind of process; it means that there is also risk arising from human-made errors. Especially in Turkey, it is hard to find a complete address structure for people because of the standardization problem. There exists a standard address structure, however it may show differences from city to city, from county to county or even from part of county to another part of it. Because of the unstructured nature of the Turkish address records, it is not very easy for citizens to follow these changes which depend on place.

Possible addressing problems can be listed as follows:

i. Wrong, missing or nonstandard county or city names (B.Çekmece/Büyük Çekmece, Buyükcekmece)
ii. Wrong, missing postal codes
iii. Different abbreviations used in addresses (MHL. BLVR. M. SKK.)
iv. Other wrong, missing address parts (streets, highroads etc.)

Figure 1.1. shows some sample addresses taken from the customer address datasets of a financial company in Turkey. It is clear that there is no unique representation for the addresses. There are some punctuation errors and missing or erroneous words.

Address 1: HOSSOHBET SOKAK ELBASAN APART. 7/9 D. 11 B,TAS İST 80000
Address 2: ÇANKAYA KONAK İZMİR
Address 3: K. MILLIYE C. P. REIS M. MUZAFFER OZCAN AP. 1/3 MERSİN
Address 4: 100YIL BLVR. Y. BARAJ GUZIDEERGIN SIT. ADANA
Address 5: 80. YIL ÖĞRETMEN EVİ BAĞBAŞI DENİZLİ
Address 6: IL TARIM MUD. BORNOVA İZMİR 35030
Address 7: BESIKTAS BARBOROS BU 56/58 K:2 D:17 İSTANBUL 80700

Figure 1.1 : Addressing Problem Examples

This thesis works on cleaning of addressing problems specified above and make verification for any kind of address whether it is written wrong or right. To do this, some of the techniques that Kukich mentioned in his paper (Kukich 1992) will be applied.

Faulty addresses are the real sources of the problem but what makes these addresses faulty also takes attention. Some of the reasons for faulty address writing are as follows;

i. Lazy typing of the address by form fillers,
ii. Lack of knowing the address,
iii. Change made on streets, road or county names,
iv. Bad data structure of previously developed applications that holds address data,
v. Fast data entering in rush business life.

The reason for addressing problem is generally caused by human factors. So it needs to be well examined and the proper solution needs to be found.

1.3 PREVIOUS WORK

Address cleaning studies especially concentrate on two areas; data cleaning and word matching. There are numerous of scientific work done by the researchers in the field of data cleansing and improving the data quality in such a way that it represents the information more accurately. Part of the works fall in natural language processing (NLP) and dictionary based whereas some of them use different string algorithms to find the incorrect words in a record set. There are also neural networks-like approaches which train the system on the correct/incorrect words or phrases and try to eliminate the erroneous data. In this part of the thesis, some studies done by other researchers are summarized by giving emphasis on their findings.

Data cleaning is an important step in the data mining process. Successful data mining applications require good quality data. An iterative refinement approach for data cleaning can be performed by the Expectation-Maximization (EM) algorithm (Amitava and Stephen 2007). It devises a data cleaning technique that smoothes out a substantial amount of attribute noise and handles missing attribute values as well. It iteratively refines each attribute-value using a predictor constructed from the previously refined values (known values in the first iteration). In their study Amitava and Stephen

demonstrated the effectiveness of the technique in smoothing out attribute noise and corroborate the efficacy of the technique by showing improved classification accuracy on a number of real world data sets from UCI (University of California, Irvine) repository. Moreover, it is also showed that this technique can easily be adapted to fill up missing attribute-values in classification problems more effectively than other standard approaches (Bilenko and Mooney 2003).

Often, in the real world, entities have two or more representations in databases. Duplicate records do not share a common key and/or they contain errors that make duplicate matching a difficult task (Peter 2005). Errors are introduced as the result of transcription errors, incomplete information, lack of standard formats, or any combination of these factors. Duplicate Record Detection covers similarity metrics that are commonly used to detect similar field entries, and authors present an extensive set of duplicate detection algorithms that can detect approximately duplicate records in a database (Elmagarmid et al. 2007). Elmagarmid also covered multiple techniques for improving the efficiency and scalability of approximate duplicate detection algorithms.

Researches aimed at correcting words in text have focused on three progressively more difficult problems: non-word error detection; isolated-word error correction; and context-dependent work correction (Kukich 1992). In response to the first problem, efficient pattern matching and n-gram analysis techniques have been developed for detecting strings that do not appear in a given word list (Brown et al. 1990). In response to the second problem, a variety of general and application-specific spelling correction techniques have been developed. Some of them were based on detailed studies of spelling error patterns. In response to the third problem, a few experiments using natural-language-processing tools or statistical-language models have been carried out (Deheer 1982).

As more online databases are integrated into digital libraries, the issue of quality control of the data becomes increasingly important, especially as it relates to the effective retrieval of information. The need to discover and reconcile variant forms of strings in bibliographic entries, i.e., authority work, will become more critical in the future.

Spelling variants, misspellings, and transliteration differences will all increase the difficulty of retrieving information. Approximate string matching has traditionally been used to help with this problem (French et al. 1997). Their study introduces the notion of approximate word matching and shows how it can be used to improve detection and categorization of variant forms.

Aşlıyan and Günel try to detect misspelled words in Turkish text using syllable n-gram frequencies (2007). They designed and implemented a system which decides whether or not a word is misspelled in Turkish text. Firstly, three databases of syllable monogram, bigram and trigram frequencies are constructed using the syllables that are derived from five different Turkish corpora. Then, the system takes words in Turkish text as an input and computes the probability distribution of words using syllable monogram, bigram and trigram frequencies from the databases. If the probability distribution of a word is zero, it is decided that this word is misspelled. For testing the system, it is constructed two text databases with the same words. One text database has 685 misspelled words. The other has 685 correctly spelled words. The words from these text databases are taken as inputs for the system. The system produces two results for each word: "Correctly spelled word" or "Misspelled word". The system that is designed with monogram and bigram frequencies has 86 percent success rate for the misspelled words and has 88 percent success rate for the correctly spelled words. According to the system designed with bigram and trigram frequencies, there is 97 percent success rate for the misspelled words and there is 98 percent success rate for the correctly spelled words.

In the 1990's, Geographic Information Systems started having a remarkable demand, since they are an innovative technology that allows visualizing information in a spatial way, along with its geographic distribution. Digital maps enterprises offer a variety of services, among them stand out the ratification of addresses: to check clients' databases for detection and correction of wrong entries, and then to validate the integrity of every new record that is inserted. Street address correction based on spelling techniques focus on development of an algorithm that improves the process of ratification with the goal of minimizing the human intervention required in the process, without sacrificing

quality (Mois et al. 2005). The benefits are better response times and reduction of service costs.

The postal address data and the domain information for address validation contain qualitative, numeric, interval and other types of data. The efficient processing of such data required for postal automation needs a robust data structure that facilitates their storage and access (Giovani 2002). A symbolic data structure is proposed to represent the postal address and the information relevant for validating the postal address is stored in a newly devised symbolic knowledge base (Nagabhushan et al. 2005). The symbolic representation gives a formal structure to the information and hence is more beneficial than other representations such as frames, which do not reflect the structure inherent in the domain knowledge. The process of postal address validation checks the different components of the postal address for consistency before using it for further processing. In the present work a symbolic knowledge base supported address validation system is developed and tested for about 500 addresses. The system efficiency is observed to be 95.6 percent in validating the addresses automatically.

1.4 THESIS ROADMAP

This thesis' work is divided into five main chapters. In chapter 1, brief information about addressing problems is given so far and tried to find answers to questions such as: What is an address? Where do we use addresses in daily life (especially in business)? What kind of errors occurs in addresses? What is the reason of these occurrences? What is data cleaning and word matching? Chapter 1 is basically an introduction to the problem details and previous studies done by other researchers.

In chapter 2, components of an address – street, highroad, region and county, city, postal codes- will be introduced with examples and mostly made errors will be showed. After showing the possible errors introduction of the required algorithms - Levenshtein Distance, Soundex and Fast Distance - to perform string similarity for the words in address components will be introduced.

Chapter 3 includes the application development phases. Firstly, information about the used technologies will be given. It starts with integrating the sample test data to database and making required abbreviation replacements on this test address data. Then the core parts of development will be introduced and information about the important functionalities will be given. Application of the algorithms shown in this chapter to find proper words for non-word parts of the addresses and replacement with matching words will be explained.

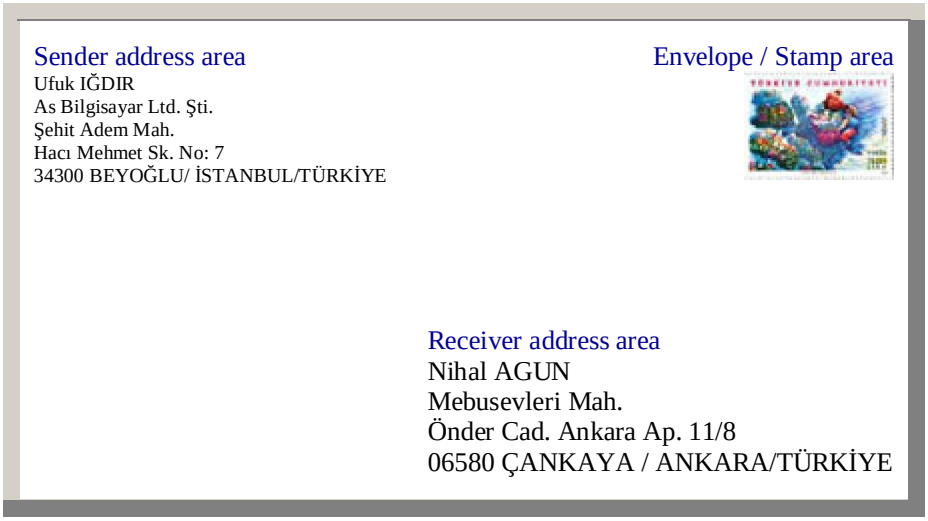
Chapter 4 contains case studies to check differences between algorithms on different parts of address components. Results of case studies are documented and discussed. The improvement rates of different addresses are calculated and the output of the application are also examined and discussed.

In conclusions chapter, accuracy of the program results are reported. Success rate percentages on different address classes are checked and the reasons for different resultings are discussed. This thesis ends with further comments about how it can be better and what are the limitations.

2. TURKISH STREET ADDRESS COMPONENTS, PROBLEMS AND SOLUTIONS

In today's world, people and business are very integrated with internet and most of them have emails. Business companies prefer sending customer requirements via emails to minimize the mailing costs. But especially in Turkey, it is not easy for every citizen to connect to the internet and get information. Additionally, e-mail messages can be messed up in junk mails which in turn their advantages are greatly reduced. As a law requirement the official documents have to be sent in paper. This also makes use of letters inevitable. This is why almost all the companies still have to use classical mailing system and inform customers with classic letters.

Figure 2.1 shows an example of properly typed address.



Source : PTT (2008). Proper Address Writing.

Figure 2.1: Properly Typed Address Format

Complete and true addresses must be written including the following information according to General Directorate of Turkish Post Office (PTT 2008).

i. Name and Surname
ii. Title (Business or firm)
iii. Parish/Village

iv. High Road/Street/Place/Site

v. Outdoor No/Indoor No

vi. Postal Code/District/County/City/Country

In most of the countries, postal addresses have a well defined template. A research made on 40 countries over address standardization shows that building names and street names are two core elements of addresses (KTÜ GISLab 2006). Table 2.1 shows the information and their percentage uses in a typical address record. According to the table, the most included address part is building name (besides the firm or person name).

Table 2.1: Percentage of Address Components in Other Countries

(1) Firm or person name (%92)
(2) Building number (%52)
(3) Street name (%52)
(4) Postal Code (%61)
(5) County or region name (%51)
(6) City name (%27)
(7) Country (%52)

Source : KTÜ GISLab (2006). A research about addresses.

Following the component orders given in Table 2.1, Table 2.2 is an example address showing the order of the address components in Turkey.

Table 2.2: Order of Address Components in Turkey

Sn. Mehmet Çete (1)
10 (2) Gül Sokak (3)
61250 (4) Yomra (5)
Trabzon (6) Türkiye (7)

As briefly explained in Chapter 1, Turkish address records are not structural and there may be numerous representations for a single address. Table 2.3 shows address writing combinations of the address components according to PTT. Row 4 and row 5 are the real error sources in typing the addresses.

Table 2.3: Address Writing Combinations

Row	Components
1	Real person name and surname
2	Business/Firm name/title
3	Parish or village name or postal subscriber box number
4	Highroad or street or place or site name, outdoor/indoor number (third row and fourth row can be combined, however outdoor/indoor number can be written in other row)
5	Postal code and district name or county name or city name (if district name or county name are not unique in Turkey, city name must be also written)
6	Country name (this can be added to fifth row and it is not used in writings inside the country)

Source : PTT (2008) Proper Address Writing.

Table 2.4 gives examples for combinations of an address with some disturbances in the address fields.

Table 2.4: Examples for Address Writing Combinations

Row	Address Fields	Address Fields
1	Hasan TAŞKIN	
2	PTT GENEL MÜD. POSTA TELGRAF D.BŞK.	PTT GENEL MÜD. POSTA TELGRAF D.BŞK.
3	DOĞANBEY MAH. ŞEHİTTEĞMENKALMAZ CAD.	DOĞANBEY MAH.
4	POSTA SARAYI A BLOK NO: 2/311	ŞEHİTTEĞMENKALMAZ CAD. NO: 2/311
5	06101 ALTINDAĞ/ANKARA	TR 06101 ALTINDAĞ/ANKARA
6	TÜRKİYE	
1		Hasan TAŞKIN
2	PTT GENEL MÜD. POSTA TELGRAF D.BŞK.	
3	DOĞANBEY MAH.	DOĞANBEY MAH.
4	ŞEHİT TEĞMEN KALMAZ CAD. 2/311	ŞEHİTTEĞMENKALMAZ CAD. 28/A
5	06101 ULUS/ANKARA/ TÜRKİYE	06101 ULUS/ANKARA
6		
1	Hasan TAŞKIN	Hasan TAŞKIN
2		
3	İSTASYON MAH.	YUKARIKAMIŞLI KÖYÜ
4	SAĞLIK SOK. NO:5	NO:51
5	06850 HASANOĞLAN/ELMADAĞ	06870 ELMADAĞ/ANKARA
6		

Addresses can be written in different combinations on letters but when it comes to save these addresses in digital databases, rules are different. These issues are explained in the next section.

2.1 COMPONENTS OF AN ADDRESS DATA IN DATASETS

Companies keep all kind of information in databases in business. Business companies want to take as much information as possible about their customers to use for their own good. Address data is one of these and it is the basic way of sending information to customers because all they have an address. Perhaps the most important area where customer addresses are used is Customer Relations Management (CRM) with which valid address records are very important.

Address structure shows differences in databases compare to letters. Database can hold data as much as it can, also it can keep information which are normally not needed on letters. As shown in Table 2.5, widely used structure for holding address data contains components such as customer number column, address type column, address column 1, address column 2, district column, county column, city column, postal code column.

Table 2.5: Sample Address Dataset

Customer #	Type	Address1	Address2	District	County	City
63125	E1	19 MAYIS MH.DR.ISMET OZTURK CD	SAADET HN.SK.30/13		ŞİŞLİ	İSTANBUL
63127	E1	BOSNA BULV.ATA -2 SİT.	ÇAM SOK. NO.5 D.6 MAVİÇAM CD.	ÇENGELKÖY	ÜSKÜDAR	İSTANBUL
63134	E1	DELİ HÜSEYİN PAŞA CD.	NO: 15/7		BAHÇELİEVLER	İSTANBUL
63140	E1	NECATIBEY CD.AYVAZ HAN	NO:207	KARAKOY	EMİNÖNÜ	İSTANBUL
63141	E1	DR.ERKIN CD.SAHIKA SK.NO:27	K:2 D.5	GOZTEPE	KADIKÖY	İSTANBUL
63142	E1	HALICILAR CD. SENTÜRK APT.	NO:5 D:15		BEYOĞLU	İSTANBUL

Dataset provided by PTT for postal codes which is used for verification of sample test addresses has parish/village column, postal code column, district column, county column and city column as shown in Table 2.6.

Table 2.6: Sample Postal Code Dataset

Parish/Village	Postal Code	District	County	City
KİREMİTHANE MAH.	1240	PTTEVLERİ	YÜREĞİR	ADANA
AKKÖPRÜ KÖYÜ	17200	KÖYLER	BİGA	ÇANAKKALE
GAZİÖMERBEY MAH.	22700	ÇARŞI	ENEZ	EDİRNE
BİZGİLİ KÖYÜ	29650	KÖYLER	KÖSE	GÜMÜŞHANE
İSTİKLAL MAH.	55310	BÜYÜKLÜ	TEKKEKÖY	SAMSUN
HACIKÖY MAHALLESİ	61400	YALIKÖY	VAKFIKEBİR	TRABZON

Abbreviations are used for some special words such as “mahalle” , “sokak” and many more in address records. People used different abbreviations for same words and that makes address readability harder. Components of addresses should not include abbreviations different than Table 2.7 shows. (PTT 2008)

Table 2.7: Abbreviations Used in Turkish Street Addresses

MAH.	MAHALLE(Sİ)
CAD.	CADDE(Sİ)
BLV.	BULVAR(I)
SOK.	SOKAK(GI)
APT.	APARTMANI
NO:	NUMARASI
PK	POSTA ABONE KUTUSU

Source : PTT (2008). Proper Address Writing.

Despite these abbreviations people use abbreviations “K:” for “KAT”, “D:” for “DAİRE”. These abbreviations should be part of “NO:” and not be used separately. Also using words like “YANI”,”KARŞISI”,”ÜSTÜ” must be avoided. Problems start here, most of the people do not avoid from anything while they are writing addresses and makes too many mistakes.

2.2 FREQUENTLY MADE MISTAKES IN ADDRESS WRITING

Writing an address wrong is mostly done because of lack of knowledge. People think that what they are writing is always true and if they understand that address others will too. But it is not that much easy and that is why rules for. Writing a complete and true address may vary from people to people. Let’s take a look how one address can be written in many different ways in Table 2.8.

Table 2.8: Different Address Writing Combinations

Address	District	County	City	Post Code
Hacışerif sok. No:15/2	Gümüşpala	Avcılar	İstanbul	34320
Gümüşpala mah. Hacışerif sok. No:15/2		Avcılar	İstanbul	
Gümüşpala mahallesi Hacışerif sokak 15/2		Avcılar		34320
G.Pala mhl. H.Şerif skk N.15-2		Avc	İstanbul	
G.Pala mahallesi H.Şerif sokak 15 k.1 d.2		Avc	İst	

In Table 2.8, six different addresses are written for a particular place. The first address is complete and accurate. Second address takes district to address part and use it as parish but missing postal code however address is still reasonable and can be used. Third address do not use abbreviations “Mah.” and “Sok.” also does not contain city but the postal code is true , address is not written valid but it is still reasonable. Fourth address uses “G.Pala” instead of “Gümüşpala” and “H.Şerif” instead of “Hacışerif”. The person writes the fourth address makes an abbreviation from his mind and thinks it is reasonable and if he understands it other will too. Also wrong abbreviations which are “mhl.” and “skk.” and “N.” are used instead of “Mahalle”, “Sokak” and “No:” It also has shortening in county and city columns. Address writer uses only first three letters of both county and city. The fourth address cannot be used as a valid address. Fifth address looks like fourth address and also has “K.” and “D.” which are not preferred for using.

As can be seen in Table 2.8, one can write an address in many different ways and in many wrong ways. Some of the problems occur in these addresses can be fixed without a human contact and some not. In the next sections, this thesis will focus on what kind of problems can be solved and what the solution might be.

Abbreviation problems can be solved by easy replacements. The sample addresses used in this study are searched, mostly done abbreviation mistakes are extracted and replaced that faulty ones with true abbreviation shown in Table 2.9.

Table 2.9: Wrong Abbreviations Used in Sample Addresses

MAH.	MAHALLESİ. MAHALLES MAHALLE. MAHALLE MAH. MAH MH. MH M. M
CAD.	CADDESİ. CADDESİ CADDE. CADDE CAD. CAD CD. CD C. C
BLV.	BULVARI. BULVARI BULVAR. BULVAR BLVR. BLVR BULV. BULV BUL. BUL BLV. BLV
SOK.	SOKAKI. SOKAKI SOKAGI. SOKAGI SOKAGİ. SOKAĞİ SOKAK SOK. SOK SK. SK S. S
APT.	APARTMANI. APARTMANI APARTMAN. APARTMAN APT. APT AP. AP
NO:	N. NO. N= NO= N: N NO

Some rules can be created for abbreviations and replacements can be made by using Table 2.9. But it is not almost possible to create rules for every written wrong word. For example if city column is “S.URFA” it is understood that is “ŞANLIURFA” but what if only “URFA” written. These are abbreviations which are not created by rules and it is not easy to make replacement by using table. If one tries, number of rows in table may increase to ten thousands and even more because everyone can create abbreviations from their mind if they wish. So at this point something else is needed. Let’s take a look at Table 2.10.

Table 2.10: Sample Shortenings on Addresses

Ş.URFA	ŞANLIURFA
AVC	AVCILAR
AFYON	AYYONKARAHİSAR
İSTNBL	İSTANBUL
MARAŞ	KAHRAMANMARAŞ
N.KEMAL	NAMIK KEMAL
E.ŞEHİR	ESKİŞEHİR

Such words can be many and writing a rule is not easy. But similarity can be seen between word couples. People use the similarity they created while doing these shortenings. So what is needed is an algorithm that can find similarity between words.

2.3 STRING SIMILARITY ALGORITHMS

The problem of word error correction entails three sub problems:

i) detection of an error,

ii) generation of candidate corrections,

iii) and ranking of candidate corrections.

In detection research, n-gram statistics initially plays a central role in text recognition techniques while dictionary-based methods dominated spelling correction techniques. But text recognition researchers quickly discovered that n-gram analysis alone was inadequate to the task of correction. Many other clever techniques were invented based on minimum edit distance algorithms, similarity keys, rule-based procedures, probability estimates and neural nets (Kukich 1992).

This thesis work will mainly be focusing on Levenshtein algorithm based on minimum edit distance and Soundex system based on similarity key techniques.

2.3.1 Levenshtein Distance

In information theory and computer science, the **Levenshtein distance (LD)** is a metric for measuring the amount of difference between two sequences (i.e., the so called edit distance). The LD between two strings is given by the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. A generalization of the LD (Damerau–Levenshtein distance) allows the transposition of two characters as an operation (Wikipedia 2008).

Table 2.11 shows the LD between "kitten" and "sitting" is 3, since the following three edits change one into the other, and there is no way to do it with fewer than three edits.

Table 2.11: Step by Step LD Example

Step 1	kitten → sitten (substitution of 's' for 'k')
Step 2	sitten → sittin (substitution of 'i' for 'e')
Step 3	sittin → sitting (insert 'g' at the end)

It can be considered a generalization of the Hamming distance, which is used for strings of the same length and only considers substitution edits. Table 2.12 gives the steps to be followed in order to complete the LD algorithm.

Table 2.12: Steps of Levenshtein Distance Algorithms

Step	Description
1	Set n to be the length of source text 's'. Set m to be the length of target text 't'. If n = 0, return m and exit. If m = 0, return n and exit. Construct a matrix containing 0..m rows and 0..n columns.
2	Initialize the first row to 0..n. Initialize the first column to 0..m.
3	Examine each character of s (i from 1 to n).
4	Examine each character of t (j from 1 to m).
5	If s[i] equals t[j], the cost is 0. If s[i] doesn't equal t[j], the cost is 1.
6	Set cell d[i,j] of the matrix equal to the minimum of: a. The cell immediately above plus 1: d[i-1,j] + 1. b. The cell immediately to the left plus 1: d[i,j-1] + 1. c. The cell diagonally above and to the left plus the cost: d[i-1,j-1] + cost.
7	After the iteration steps (3, 4, 5, 6) are complete, the distance is found in cell d[n,m].

Source : Wikipedia (2008). Levensthein distance algorithm.

Step by step example of computation of LD when the source string is "GUMBO" and the target string is "GAMBOL" is given in Figure 2.2 (Merriampark 2008). Following Figure 2, as the last step (step 7), the distance is in the lower right hand corner of the matrix, i.e. 2. This corresponds to our intuitive realization that "GUMBO" can be transformed into "GAMBOL" by substituting "A" for "U" and adding "L" (one substitution and 1 insertion = 2 changes). Another two examples are shown in Figure 3.3, "kitten" can be transformed into "sitting" by substituting "k" for "s", "e" for "i" and adding "g" (two substitution and 1 insertion = 3 changes), "saturday" can be transformed into "sunday" can be transformed into "saturday" by adding "at" and substituting "n" for "r" (two insertion and 1 substitution = 3 changes). The result of both examples are shown at right hand corner of the matrices and it is 3 for both.

Steps 1 and 2

		G	U	M	B	O
	0	1	2	3	4	5
G	1					
A	2					
M	3					
B	4					
O	5					
L	6					

Steps 3 to 6 When i = 1

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0				
A	2	1				
M	3	2				
B	4	3				
O	5	4				
L	6	5				

Steps 3 to 6 When i = 2

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1			
A	2	1	1			
M	3	2	2			
B	4	3	3			
O	5	4	4			
L	6	5	5			

Steps 3 to 6 When i = 3

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2		
A	2	1	1	2		
M	3	2	2	1		
B	4	3	3	2		
O	5	4	4	3		
L	6	5	5	4		

Steps 3 to 6 When i = 4

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2	3	
A	2	1	1	2	3	
M	3	2	2	1	2	
B	4	3	3	2	1	
O	5	4	4	3	2	
L	6	5	5	4	3	

Steps 3 to 6 When i = 5

		G	U	M	B	O
	0	1	2	3	4	5
G	1	0	1	2	3	4
A	2	1	1	2	3	4
M	3	2	2	1	2	3
B	4	3	3	2	1	2
O	5	4	4	3	2	1
L	6	5	5	4	3	2

Figure 2.2: Seven Steps LD Example Between Words "GUMBO" and "GAMBOL"

		k	**i**	**t**	**t**	**e**	**n**
	0	1	2	3	4	5	6
s	1	1	2	3	4	5	6
i	2	2	1	2	3	4	5
t	3	3	2	1	2	3	4
t	4	4	3	2	1	2	3
i	5	5	4	3	2	2	3
n	6	6	5	4	3	3	2
g	7	7	6	5	4	4	3

		S	**a**	**t**	**u**	**r**	**d**	**a**	**y**
	0	1	2	3	4	5	6	7	8
S	1	0	1	2	3	4	5	6	7
u	2	1	1	2	2	3	4	5	6
n	3	2	2	2	3	3	4	5	6
d	4	3	3	3	3	4	3	4	5
a	5	4	3	4	4	4	4	3	4
y	6	5	4	4	5	5	5	4	3

Figure 2.3: Examples of LD Between Words "Kitten" and "Sitting", "Saturday" and "Sunday"

Two examples of the resulting matrix are shown in Figure 2.3. The minimum steps to be taken are highlighted.

The invariant maintained throughout the algorithm is that we can transform the initial segment `s[1..i]` into `t[1..j]` using a minimum of `d[i,j]` operations. At the end, the bottom-right element of the array contains the answer.

As mentioned earlier, the invariant is that we can transform the initial segment `s[1..i]` into `t[1..j]` using a minimum of `d[i,j]` operations. This invariant holds since:

i. It is initially true on row and column 0 because `s[1..i]` can be transformed into the empty string `t[1..0]` by simply dropping all `i` characters. Similarly, we can transform `s[1..0]` to `t[1..j]` by simply adding all `j` characters.

ii. The minimum is taken over three distances, each of which is feasible:
    a. If we can transform `s[1..i]` to `t[1..j-1]` in `k` operations, then we can simply add `t[j]` afterwards to get `t[1..j]` in `k+1` operations.
    b. If we can transform `s[1..i-1]` to `t[1..j]` in `k` operations, then we can do the same operations on `s[1..i]` and then remove the original `s[i]` at the end in `k+1` operations.
    c. If we can transform `s[1..i-1]` to `t[1..j-1]` in `k` operations, we can do the same to `s[1..i]` and then do a substitution of `t[j]` for the original `s[i]` at the end if necessary, requiring `k+cost` operations.

iii. The operations required to transform `s[1..n]` into `t[1..m]` is of course the number required to transform all of `s` into all of `t`, and so `d[n,m]` holds our result.

This proof fails to validate that the number placed in `d[i,j]` is in fact minimal; this is more difficult to show, and involves an argument by contradiction in which we assume `d[i,j]` is smaller than the minimum of the three, and use this to show one of the three is not minimal (Wikipedia 2008).

Possible improvements to this algorithm include:

i. The algorithm can be adapted to use less space, $O(m)$ instead of $O(mn)$, since it only requires that the previous row and current row be stored at any one time.

ii. The number of insertions, deletions, and substitutions can be stored separately, or even the positions at which they occur, which is always `j`.

iii. The distance to the interval `[0,1]` can be normalized.

iv. If the distance is the only quantity in interest and if the distance is smaller than a threshold *k*, then it suffices to compute a diagonal stripe of width *2k+1* in the matrix. In this way, the algorithm can be run in $O(kl)$ time, where l is the length of the shortest string.[2]

v. Different penalty costs to insertion, deletion and substitution can be given. Penalty costs can be dependent on which characters are inserted, deleted or substituted.

vi. The initialization of `d[i,0]` can be moved inside the main outer loop.

vii. By initializing the first row of the matrix with *0*, the algorithm can be used for fuzzy string search of a string in a text. This modification gives the end-position of matching substrings of the text. To determine the start-position of the matching substrings, the number of insertions and deletions can be stored separately and used to compute the start-position from the end-position.

viii. This algorithm parallelizes poorly, due to a large number of data dependencies. However, all the **cost** values can be computed in parallel, and the algorithm can be adapted to perform the **minimum** function in phases to eliminate dependencies.

ix. By examining diagonals instead of rows, and by using lazy evaluation, Levenshtein distance can be found in $O(m\ (1 + d))$ time (where d is the Levenshtein distance), which is much faster than the regular dynamic programming algorithm if the distance is small.

2.3.2 Soundex Algorithm

Words can be misspelled or have multiple spellings, especially across different cultures or national sources. To help solve this problem, we need phonetic algorithms which can find similar sounding terms and names. Just such families of algorithms exist and have come to be called Soundex algorithms.

A Soundex search algorithm takes a written word, such as a person's name, as input and produces a character string that identifies a set of words that are (roughly) phonetically alike. It is very handy for searching large databases when the user has incomplete data. The original Soundex algorithm began in a patent by Robert C. Russell in 1918. The name "Soundex" came along later.

The method used by Soundex is based on the six phonetic classifications of human speech sounds (bilabial, labiodental, dental, alveolar, velar, and glottal), which in turn are based on where you put your lips and tongue to make the sounds. The algorithm itself is fairly straight forward to code and requires no backtracking or multiple passes over the input word. Table 2.13 shows steps of the Soundex algorithm.

Table 2.13: Steps of Soundex Algorithms

Step	Description
1	Capitalize all letters in the word and drop all punctuation marks. Pad the word with rightmost blanks as needed during each procedure step.
2	Retain the first letter of the word.
3	Change all occurrence of the following letters to '0' (zero): 'A', E', 'I', 'O', 'U', 'H', 'W', 'Y'.
4	Change letters from the following sets into the digit given: 1 = 'B', 'F', 'P', 'V' 2 = 'C', 'G', 'J', 'K', 'Q', 'S', 'X', 'Z' 3 = 'D','T' 4 = 'L' 5 = 'M','N' 6 = 'R'
5	Remove all pairs of digits which occur beside each other from the string that resulted after step (4).
6	Remove all zeros from the string that results from step 5.0 (placed there in step 3)
7	Pad the string that resulted from step (6) with trailing zeros and return only the first four positions, which will be of the form <uppercase letter> <digit> <digit> <digit>.

Source : Wikipedia (2008). Soundex algorithm.

Step 1.	Birthwisle à BIRTWISLE
Step 2.	BIRTWISLE à B
Step 3.	BIRTWISLE à BRTSL
Step 4.	BRTSL à B6324
Step 5.	(Nothing required here)
Step 6 and 7.	B6324 à B632

Figure 2.4: Example of Soundex Algorithm of Word "Birthwisle"

Following the algorithm rules, therefore Birtwisle turns into B632 as shown in Figure 2.4. Similar variants will also be given the same code, such as "Birtwistle", or "Bertwhistle".

The algorithm presented in this study is slightly improved over the originally patented algorithm. The original Soundex algorithm of 1918 starts to fail when the number of words in the database gets larger. For example, the diversity of names in a large database with many foreign spellings starts to put more and more phonetically unlike names into the same code. So Soundex variants are also developed.

A similar algorithm called "Reverse Soundex" prefixes the last letter of the name instead of the first.

The NYSIIS algorithm was introduced by the New York State Identification and Intelligence System as an improvement to the Soundex algorithm. NYSIIS handles some multi-character n-grams and maintains relative vowel positioning, whereas Soundex does not. The Celko Improved Soundex algorithm was introduced by Joe Celko in his book *SQL For Smarties: Advanced SQL Programming*.

As a response to deficiencies in the Soundex algorithm, Lawrence Philips developed the Metaphone algorithm for the same purpose at 1990. Philips later developed an improvement to Metaphone, which he called Double-Metaphone at 2000. Double-Metaphone includes a much larger encoding rule set than its predecessor, handles a subset of non-Latin characters, and returns a primary and a secondary encoding to account for different pronunciations of a single word in English (Wikipedia 2008).

Daitch-Mokotoff Soundex (D-M Soundex) was developed by genealogist Gary Mokotoff and later improved by genealogist Randy Daitch because of problems they encountered while trying to apply the Russell Soundex to Jews with Germanic or Slavic surnames (such as Moskowitz vs. Moskovitz or Levine vs. Lewin). D-M Soundex is sometimes referred to as "Jewish Soundex" or "Eastern European Soundex", although

the authors discourage the use of these nicknames. The D-M Soundex algorithm can return as many as 32 individual phonetic encodings for a single name. Results of D-M Soundex are returned in an all-numeric format between 100000 and 999999. This algorithm is much more complex than Russell Soundex (Wikipedia 2008).

3. DEVELOPMENT OF APPLICATION

3.1 TOOLS AND TECHNOLOGIES USED

The application coded for the requirements of this thesis is developed on Microsoft .NET environment and with the software and database technologies it supported.

3.1.1 .NET Framework

The Microsoft .NET Framework is a software technology that is available with several Microsoft Windows operating systems. It includes a large library of pre-coded solutions to common programming problems and a virtual machine that manages the execution of programs written specifically for the framework. The .NET Framework is a key Microsoft offering and is intended to be used by most new applications created for the Windows platform.

The pre-coded solutions that form the framework's Base Class Library cover a large range of programming needs in a number of areas, including user interface, data access, database connectivity, cryptography, web application development, numeric algorithms, and network communications. The class library is used by programmers, who combine it with their own code to produce applications.

Programs written for the .NET Framework execute in a software environment that manages the program's runtime requirements. Also part of the .NET Framework, this runtime environment is known as the Common Language Runtime (CLR). The CLR provides the appearance of an application virtual machine so that programmers need not consider the capabilities of the specific CPU that will execute the program. The CLR also provides other important services such as security, memory management, and

exception handling. The class library and the CLR together compose the .NET Framework (Wikipedia 2008).

3.1.2 C Sharp Programming Language

C# (pronounced C Sharp) is a multi-paradigm programming language that encompasses functional, imperative, generic, object-oriented (class-based), and component-oriented programming disciplines. It was developed by Microsoft as part of the .NET initiative and later approved as a standard by ECMA (ECMA-334) and ISO (ISO/IEC 23270). C# is one of the 44 programming languages supported by the .NET Framework's Common Language Runtime and is used extensively with Microsoft Visual Studio .NET (Wikipedia 2008).

C# is intended to be a simple, modern, general-purpose, object-oriented programming language. Anders Hejlsberg, the designer of Delphi, leads the team which is developing C#. It has an object-oriented syntax based on C++ and is heavily influenced by other programming languages such as Delphi and Java. It was initially named Cool, which stood for "C like Object Oriented Language". However, in July 2000, when Microsoft made the project public, the name of the programming language was given as C#. The most recent release is C# 3.0 which is used with Microsoft Visual Studio 2008.

The ECMA standard lists these design goals for C#:

i. C# is intended to be a simple, modern, general-purpose, object-oriented programming language.
ii. Because software robustness, durability and programmer productivity are important, the language should include strong type checking, array bounds checking, detection of attempts to use uninitialized variables, source code portability, and automatic garbage collection.
iii. The language is intended for use in developing software components that can take advantage of distributed environments.
iv. Programmer portability is very important, especially for those programmers already familiar with C and C++.
v. Support for internationalization is very important.

vi. C# is intended to be suitable for writing applications for both hosted and embedded systems, ranging from the very large that use sophisticated operating systems, down to the very small having dedicated functions.

vii. Although C# applications are intended to be economical with regard to memory and processing power requirements, the language is not intended to compete directly on performance and size with C or assembly language.

3.1.3 Xml

The Extensible Markup Language (XML) is a general-purpose specification for creating custom markup languages. It is classified as an extensible language because it allows its users to define their own elements. Its primary purpose is to help information systems share structured data, particularly via the Internet, and it is used both to encode documents and to serialize data (Wikipedia 2008).

It started as a simplified subset of the Standard Generalized Markup Language (SGML), and is designed to be relatively human-legible. By adding semantic constraints, application languages can be implemented in XML. XML is sometimes used as the specification language for such application languages.

XML is recommended by the World Wide Web Consortium (W3C). It is a fee-free open standard. The recommendation specifies both the lexical grammar and the requirements for parsing.

3.1.4 Microsoft Visual Studio .NET

Microsoft Visual Studio is the main Integrated Development Environment (IDE) from Microsoft. It can be used to develop console and Graphical user interface applications along with Windows Forms applications, web sites, web applications, and web services in both native code together with managed code for all platforms supported by

Microsoft Windows, Windows Mobile, Windows CE, .NET Framework, .NET Compact Framework and Microsoft Silverlight (Wikipedia 2008).

Visual Studio includes a code editor supporting IntelliSense as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a forms designer for building GUI applications, web designer, class designer, and database schema designer. It allows plug-ins to be added that enhance the functionality at almost every level - including adding support for source control systems (like Subversion and Visual SourceSafe) to adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle.

Visual Studio supports languages by means of language services, which allow any programming language to be supported (to varying degrees) by the code editor and debugger, provided a language-specific service has been authored. Built-in languages include C/C++ (via Visual C++), VB.NET (via Visual Basic .NET), and C# (via Visual C#). Support for other languages such as Chrome, F#, Python, and Ruby among others has been made available via language services which are to be installed separately.

3.1.5 Microsoft SQL Server

Microsoft SQL Server is a relational database management system (RDBMS) produced by Microsoft. Its primary language is SQL (Wikipedia 2008).

SQL (Structured Query Language) is a database computer language designed for the retrieval and management of data in relational database management systems (RDBMS), database schema creation and modification, and database object access control management.

SQL is a standard interactive and programming language for querying and modifying data and managing databases. Although SQL is both an ANSI and an ISO standard, many database products support SQL with proprietary extensions to the standard

language. The core of SQL is formed by a command language that allows the retrieval, insertion, updating, and deletion of data, and performing management and administrative functions.

3.2 DEVELOPMENT PHASES

An application to execute all required operation is developed for this study. Figure 3.1 shows the second window of the application which performs required similarity operations. A number 5 which represents the class of the addresses and a number 436 which represents the quantity of the addresses of that class are seen at top left corner of the Figure 3.1. “Tam Kontrol” button executes improvement of a single address, “Bütün Adresleri Kontrol Et” button executes improvement of all addresses and “Son Hal” button shows the last state of improved addresses in next window.

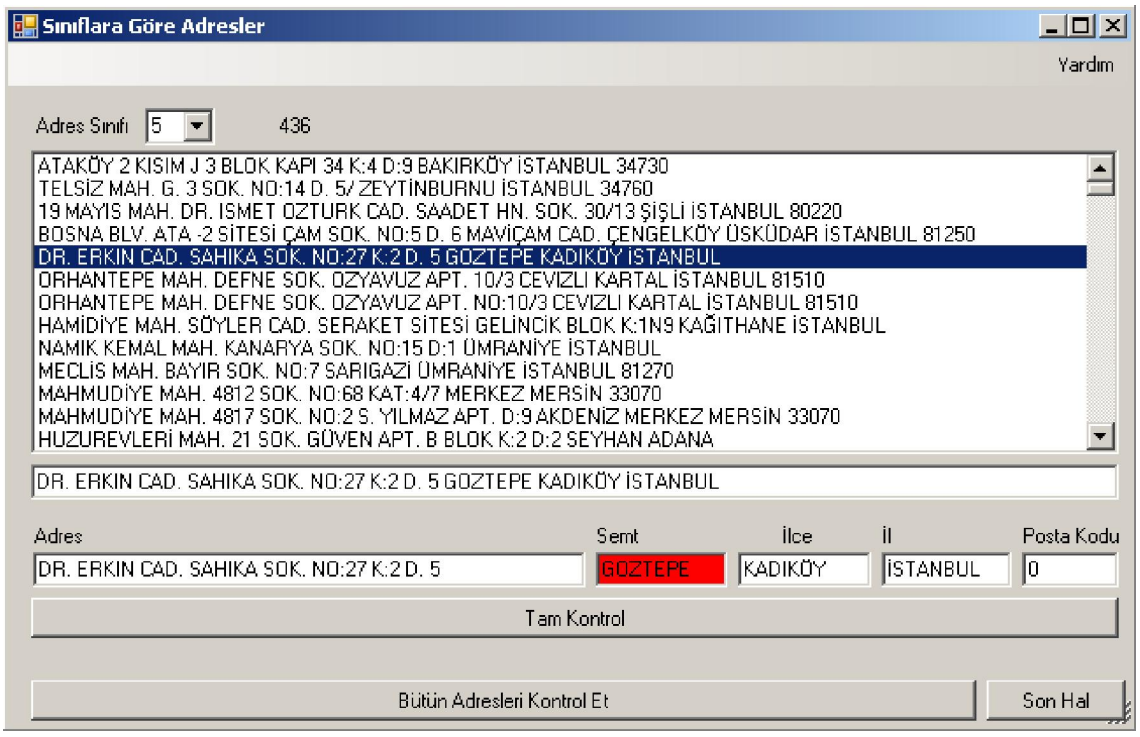


Figure 3.1: Second Window of the Application Where Improvement Algortihms are Applied

3.2.1 Preliminary Data Cleaning and Integration of Data to SQL Server

There are two datasets in excel format; one includes the sample test data and the other is the reference dataset. The sample dataset includes 1345 customer addresses of a medium sized Turkish Bank. Reference dataset is taken from PTT and it should adjust the abbreviation rules showed in Table 2.7. Even though the dataset belongs to PTT, they don't adjust their rules and use many different abbreviations for "MAH." Finding and replacing these abbreviations is the first data cleaning phase.

There exists words and abbreviations; "MAHALLESİ", "MHL.", "MHL.ALLESİ", "MH", "M.HALLE" etc. for "MAH." First of all, these are replaced with true abbreviation. This is important because "MAH." will be core abbreviation to find postal codes.

Import of datasets in excel file to Microsoft SQL Server 2005 is executed with Sql Server Import and Export Wizard. After the execution, a database called "TEZ" has two tables; "Adres" and "PostaKoduAdresleri". Table 3.1 shows the structure of Adres table in "TEZ" database. Adres table has 8 columns; Musteri No, Adres Turu, Adres 1, Adres 2, Semt , IlceAdi, IlAdi, PostaKodu. Musteri No and Adres Turu columns will not be used in later stages. Adres1 and Adres2 columns will be joined and used together. Semt , IlceAdi columns does not include any information for addresses but not for all. Table 3.2 shows the structure of PostaKoduAdresleri table in "TEZ" database. PostaKoduAdresleri table has 5 columns; mahalleKoy, postaKodu, belediyeSemt, ilce, il. This table is the reference table for the verification and the improvement of the sample addresses which is shown in Table 3.2.

Table 3.1: Adres Table in TEZ Database

Musteri No	Adres Turu	Adres1	Adres2	Semt	IlceAdi	IlAdi	PostaKodu
10068	E1	9.KISIM A/12-A BL	K.4 D.12 ATAKOY			İSTANBUL	34710
10082	E1	KARAYOLLARI C GUMUSP	AYAN ST D BL D 5 AVC			İSTANBUL	81580
10105	E1	HOSSOHBET S. ELBASAN	AP. 7/9 D.11 B,TAS			İSTANBUL	80700
10109	E1	A.BAYIR M 2.SK CIKM	GUNES AP 48/5			BANDIRMA	
10132	E1	100.YIL BULV. SISIK	AP. 131/7			SAMSUN	55200
10161	E1	KONAKLI SK N.19	FLORYA			İSTANBUL	34810
10163	E1	A.KUTSI TECER CD	3/23 MERTER			İSTANBUL	34010

Table 3.2: PostaKoduAdresleri Table in TEZ Database

mahalleKoy	postaKodu	belediyeSemt	ilce	il
KARASOKU MAH.	1010	YAĞCAMİ	SEYHAN	ADANA
KAYALIBAĞ MAH.	1010	YAĞCAMİ	SEYHAN	ADANA
TEPEBAĞ MAH.	1010	YAĞCAMİ	SEYHAN	ADANA
ULUCAMİ MAH.	1010	YAĞCAMİ	SEYHAN	ADANA
	1020	HÜKÜMET	SEYHAN	ADANA
ALİDEDE MAH.	1020	HÜKÜMET	SEYHAN	ADANA
BEŞOCAK MAH.	1020	HÜKÜMET	SEYHAN	ADANA
HÜRRİYET MAH.	1020	HÜKÜMET	SEYHAN	ADANA
KARASOKU MAH.	1020	HÜKÜMET	SEYHAN	ADANA

3.2.2 Abbreviation Replacement and Classification of Addresses

A class called “MyAddress” is implemented which does required core operations such as abbreviation replacements, space cleaning and addition, classification of addresses according to their quality.

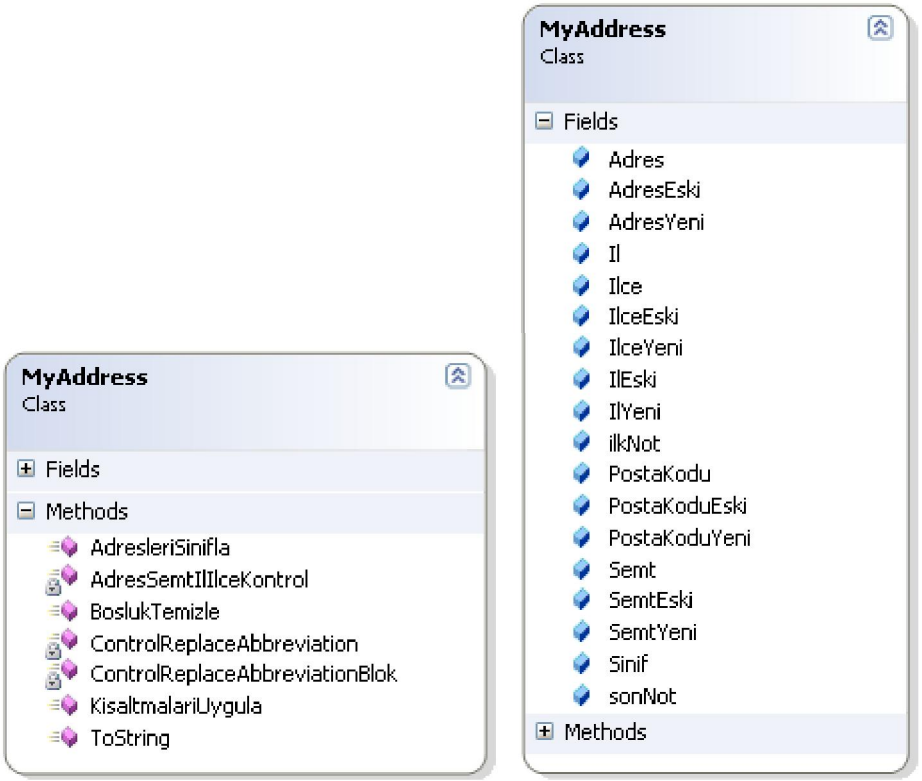


Figure 3.2: MyAddress Class Methods and Fields

Figure 3.1 shows the structure of MyAddress class. Fields ending with "Eski" hold the first state of data, fields ending with no addition hold data after abbreviation replacements and space cleaning are executed, fields ending with "Yeni" hold last state of data when overall process is accomplished and addresses are ready for saving operation.

As can be seen in Table 3.1 that, address columns of address rows are irregular. There are too many spaces or missing spaces and some unwanted characters. *BoslukTemizle* method executes required tasks below in order:

i. Replaces ";" characters with ":" of Address field
ii. Adds one space after "." character of Address field
iii. Replaces "\" character with "/"of Address field
iv. Replaces multi spaces with one space of Address field
v. Trims spaces of other fields

Faulty abbreviations need to be replaced with true ones. *ControlReplaceAbbreviation* reads possible wrong abbreviations shown in Table 2.9 which are used in addresses, from text file and replace them with true ones. *ControlReplaceAbbreviationBlok* also does the same for a special situation of "Blok" replacement. Then *KisaltmalariUygula* method calls *ControlReplaveAbbreviation* methods for each wrong abbreviation. It executes ten methods at all.

AdresSemtIlIlceKontrol method counts the number of nonempty Semt, Il and Ilce fields. It returns and integer value from zero to three. This value will later be used for classification of address.

Address classification is executed in *AdresleriSinifla* method. There exists six classes starting from zero (0) to five (5). *Sinif* variable holds the value. Classification is basically performed according to quantity of components addresses have. If an address has a parish or village its classification level is high besides if it has nonempty district, county and city columns then the classification level is best. Pseudocode of *AdresleriSinifla* method is shown in Figure 3.3.

MyAdres class is ready. The application takes *Adres* and *PostaKoduAdresleri* data from the server and keeps them in xml sheets thus no need to connect Sql server at the later stages of application anymore because data is taken from this xml sheet. To perform this operation addresses are loaded from database to .NET datasets and these datasets are written into an xml sheet with predefined method of .NET framework *WriteXml* which is called over a dataset instance**.**

It is time to read *Adres* data from xml file and load them into an instance of *MyAdres* class. Then required operations are executed on instance with the methods written in MyAdress class. So each row in dataset are loaded to instances of MyAdres and then *BoslukTemizle, KisaltmlariUygula, AdresleriSinifla* methods are called and then these instances are added to a list type of MyAdres. Later that list is serialized and saved so no need to load all these methods again every time the application is loaded and executed.

```
if (AdresSemtIlIlceKontrol() == 3)

        if (Adres.Contains(" KISIM ") || Adres.Contains(" MAH. "))
                Sinif = 5;
    else if (Adres.Contains(" CAD. "))
                if (Adres.Contains(" BLOK ") || Adres.Contains(" SİTESİ ") || Adres.Contains(" SOK.
        "))
        Sinif = 5;
                else
                        Sinif = 4;
    else if (Adres.Contains(" BLV. "))
                Sinif = 5;
    else if (Adres.Contains(" KÖYÜ "))
                Sinif = 5;
        else if (Adres.Contains(" BLOK ") || Adres.Contains(" SİTESİ ") || Adres.Contains(" SOK. "))
                Sinif = 4;
        else
                Sinif = 1;

else if (AdresSemtIlIlceKontrol() == 2)

        if (Adres.Contains(" KISIM ") || Adres.Contains(" MAH. "))
                Sinif = 5;
    else if (Adres.Contains(" CAD. "))
                Sinif = 4;
        else if (Adres.Contains(" BLV. "))
                Sinif = 4;
        else if (Adres.Contains(" KÖYÜ "))
                Sinif = 5;
        else if (Adres.Contains(" BLOK ") || Adres.Contains(" SİTESİ ") || Adres.Contains(" SOK. "))
                Sinif = 4;
        else
                Sinif = 1;

else if (AdresSemtIlIlceKontrol() == 1)

        if (Adres.Contains(" KISIM ") || Adres.Contains(" MAH. "))
                Sinif = 3;
        else if (Adres.Contains(" CAD. "))
                if (Adres.Contains(" BLOK ") || Adres.Contains(" SİTESİ ") || Adres.Contains(" SOK.
        "))
                        Sinif = 3;
                else
                        Sinif = 3;
        else if (Adres.Contains(" BLV. "))
                Sinif = 3;
        else if (Adres.Contains(" KÖYÜ "))
                Sinif = 4;
        else if (Adres.Contains(" BLOK ") || Adres.Contains(" SİTESİ ") || Adres.Contains(" SOK. "))
                Sinif = 2;
        else
                Sinif = 1;
else
    Sinif = 0;
```

Figure 3.3: AdresleriSinifla Method of MyAdres Class

Till this stage abbreviation replacements and classification process is completed. The next step is to start replacements of faulty word(s) of address components using similarity algorithms.

3.2.3 Replacement of Faulty Words and Addition of Missing Components

Address data is cleaned to get rid of the wrong abbreviations, space and some special character problems so far. The process starts by doing a consistency check to the address components. It is needed to check whether the city, county, district, parish or postal code is true or false. Sometimes these components might be empty so it is also needed to check if parish, district or county exist in “adres” column of sample addresses and verify these components with similarity algorithms.

The algorithms described in the previous chapter are then applied to perform the similarity tasks. The proper use of Soundex and Levenshtein algorithms are extremely important. Soundex gives faulty results for some cases. Soundex algorithm is replaced with improved one; double metaphone algorithm but this algorithm is also not effective for this study because the language used in this study is Turkish. The results given by double metaphone will be deeply looked for different cases in next section. It gives effective results for some scenarios but is not as much effective as expected for most situations.

Levenshtein algorithm is the core component used in this study but its basic concept was not good enough to meet requirements for some scenarios. A revised version which is supported with Hungarian method (Munkres 1957) is developed and used in this study. To solve the problem, similarity is calculated in three steps:

i) Each string is partitioned into a list of tokens.
ii) Computation of the similarity between tokens is executed by using Levenshtein algorithm
iii) The similarity between two token lists is computed.

This algorithm works good but it is slow so it is also implemented a fast distance algorithm to use less space $O(m)$ instead of $O(mn)$. Fast distance algorithm does not give accurate results as well as Levenshtein but it is really fast and works well for most cases. Compared with Levenshtein, fast distance performs 10 to 25 times faster.

The fast distance algorithm starts from the beginning of both words, compare the letters on the same position. The steps for the algorithm are as follows:

i. If words are same, move forward, else search the letter from the first word in the next given maxOffset(input of a method, default 5) letters in the second word and viceversa.

ii. Offset the words according to the closest found letter, add 1 to distance.

iii. Repeat until one of the words ends and add to the calculated distance half of the length of string remained unparsed from the other word.

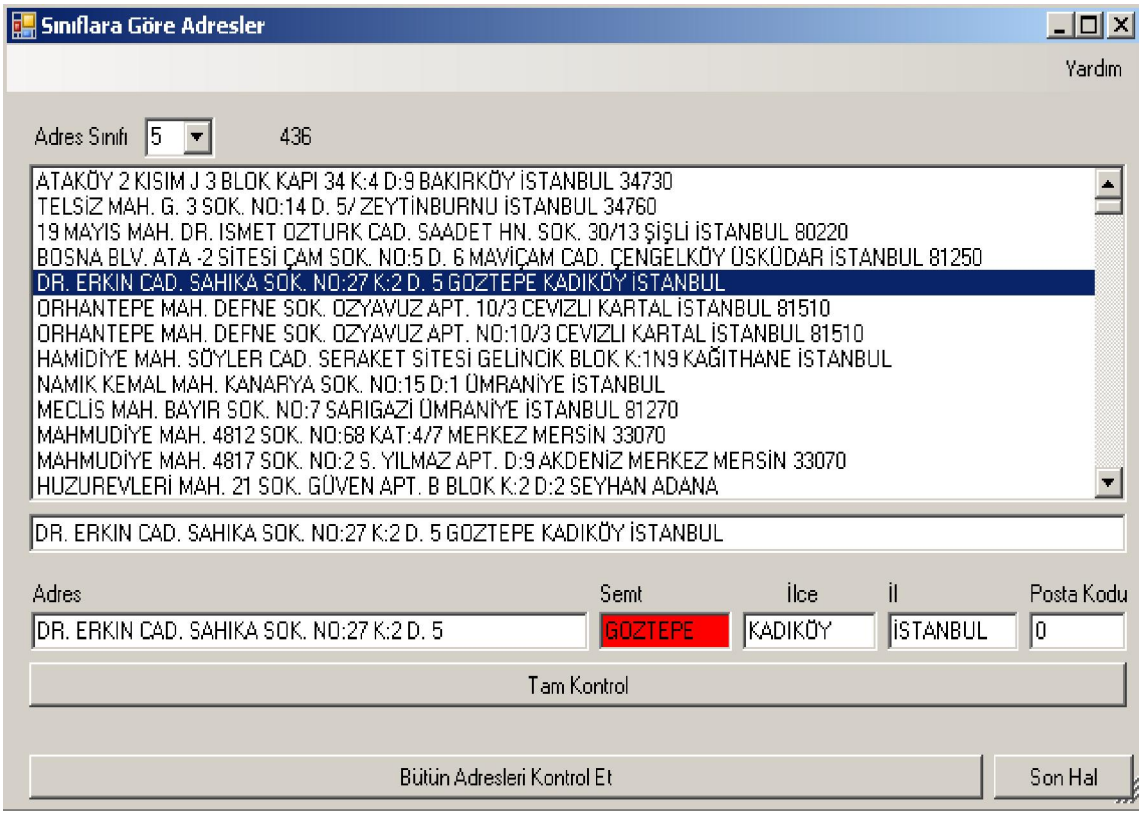


Figure 3.4: Sınıflara Göre Adres Window of the Application

The component shown in Figure 3.4, red background cell means that there is no such word in reference database for that district. It is pressed “Tam Kontrol” button to verify address. When “Sınıflara Göre Adresler” window is loaded, it firstly checks if city, county or district is valid or not. To do this, comparison of any of the components with the reference database is done and if there is no pair found then background of the textbox belongs to that component is painted to red. It is also needed to check whether a parish (string ends with “MAH.”) exist in first component of address and keep it for further verification.

Assume that there is an address like the one given in Table 3.3.

Table 3.3: Address with Faulty Components

Address	District	County	City	Post Code
FATİH SULTAN MEHMET MAH. 5. GEÇİCİ SOK. NO:69		SARIYR	İSTNBUL	0

When the control button is pressed it is expected to have a record as given in Table 3.4.

Table 3.4: Address with Correct Components

Address	District	County	City	Post Code
FATİHSULTANMEHMET MAH. 5. GEÇİCİ SOK. NO:69		SARIYER	İSTANBUL	33470

A function called *AddressAutoControl* which is called at beginning and at the end of “Tam Kontrol” is pressed is also implemented in the application. *AddressAutoControl* method first checks existence of the city component from reference dataset (*PostaKoduAdresleri* dataset). If the component is not found then it uses fast distance algorithm to find valid city. A similarity score of 70 percent is preset for finding city, if algorithm finds more than one city the similarity score is raised by 5 and does this until finding one unique city. If still nothing is found, county might be written besides city so it is checked and the value is passed to county component if county is also empty. After that *AddressAutoControl* method checks existence of the county component of the represented city from reference dataset, if component is not found then it uses fast distance algorithm to find valid county for the represented city. This time a 60 percent similarity score is given to find the county. If the algorithm finds more than one county,

the similarity score is raised by 5 and does this until finding one unique county. If no county is found it tries to find it in later stages.

Then *AddressAutoControl* method checks existence of district component under the represented city and county from reference dataset, if component is not found then it uses Levenshtein algorithm (revised one) with given 60 similarity starting similarity score to find valid district for the represented city and county. If algorithm finds more than one district, the similarity score is raised by 5 and does this until finding one unique district. If no district is found paste the content of district to address column, application tries to find it in later stages.

Levenshtein algorithm is used for district check because county might be empty and when it is empty, there exists a bigger word list for comparison. The algorithm is slower but more precise results are a need for this case. Before checking postal code existence of parish needs to be checked. *AddressAutoControl* method checks existence of parish or village word(s) in the address component under the represented city and county from reference dataset, if parish or village is not found then it uses Levenshtein to find valid parish or village under the represented city and county. A 50 percent similarity score is given for finding parish or village. If algorithm finds more than one result then the similarity score is raised by 5 and does this until finding one unique result. If no parish or village is found there is nothing else to find it. Now postal code can be found. At the end of *AddressAutoControl* method, postal code is found by making selection on references dataset. If the postal code is found, pass value to postal code column, if no postal code is found then current value doesn't change.

One of the most common mistake done in address writing is writing counties and districts into address column instead of county column and district column. Table 3.6 shows an erroneous record in which district and county are written in right places. Table 3.5 shows the same records but this time the components are written in wrong places.

Table 3.5: Address Written in Correct Format

Address	District	County	City	Post Code
2 KISIM J 3 BLOK KAPI 34 K:4 D:9	ATAKOY	BAKIRKOY	İSTANBUL	34156

Table 3.6: Address Written in Wrong Format

Address	District	County	City	Post Code
ATAKOY 2 KISIM J 3 BLOK KAPI 34 K:4 D:9 B. KÖY			İSTANBUL	34156

AddressAutoControl method is executed and it tries to find postal code. But still some components, especially district and county might be missing. The algorithm tries to find them if they exist in address column. First a check is done if both of district and county columns are empty then it is tried to find the district before county because finding district gives county immediately**.** Choosing every word in address column for comparison is not necessary. Words before special abbreviations ("MAH.", "CAD." etc.) are omitted; first word and last two words are taken for comparison. Suitable words are compared with Levenshtein algorithm with 60 percent similarity score, reference data is taken according to represented city because county and district are empty. If algorithm finds more than one result then similarity score is raised by 5 and does this until finding one unique result. If unique district is found then county that include this district is found using reference dataset. Both of district and county might be same after this process if so district's value is cleared and only county takes the found value. If algorithm does not give any results, another method tries to find appropriate word for county column only. Levenshtein is used with 50 percent similarity score this time. If algorithm finds more than one county then the similarity score is raised by 5 and does this until finding one unique county. Choosing every word in address column for comparison is not necessary again just like in previous case. So some words are omitted as done in district case. After trying to find district and county, *AddressAutoControl* method is called again to find postal code. By calling *AddressAutoControl* method again the process of pressing "Tam Kontrol" button is completed.

"Tam Kontrol" button executes verification of one address. All address are needed to be verified by pressing "Tüm Adresleri Kontrol Et" button. The list is filled with addresses

from all classes containing 0 to 5 when the button is pressed. Addresses are given scores between 0 and 10 before verification and after verification to compare the results. Addresses are also classified according to their verified values in later stages. When verification and scoring of all addresses are completed, address list is saved with serialization.

The three states of the address data can be seen in Figure 3.5.

i. At first state; no replacement is done,
ii. At second state; abbreviation replacements, space cleaning are done and address takes a score,
iii. Last state; all process is completed and address takes its last score.

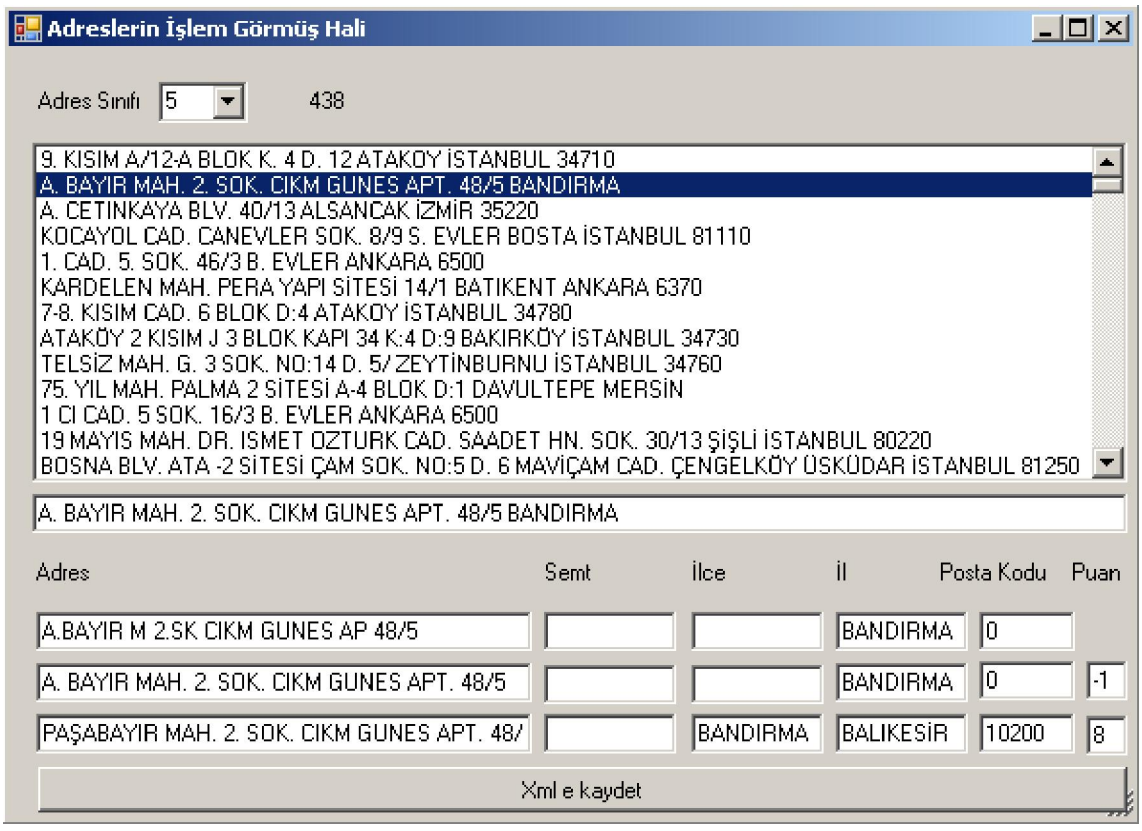


Figure 3.5: Adreslerin İşlem Görmüş Hali Window of the Application

“Xml e kaydet” button is pressed to save verified addresses in xml file in given format shown in Figure 3.6.

Figure 3.6: Address Data in XML File

4. CASE STUDIES, RESULTS AND DISCUSSIONS

Results of Levenshtein Distance (LD), Fast Distance (FD) and Soundex algorithms at different cases are given in this section.

4.1 CASE I: CITIES

In Case 1, the algorithm performances are examined for city names. The selected cities are processed by using Soundex, FD, LD and the results are documented in Table 4.1a,b and Table 4.2a,b.

Table 4.1a: Soundex and Levenshtein Distance of Sample Cities

City	SOUNDEX	LD %50	LD %60	LD %65	LD %70
ADANA	ADANA AYDIN	ADANA ADIYAMAN AMASYA ANKARA ARDAHAN	ADANA	ADANA	ADANA
MERSİN	MERSİN(İÇEL)	MARDİN MERSİN(İÇEL)	MARDİN MERSİN(İÇEL)	MARDİN	
ANTALYA	ANTALYA	AMASYA ANKARA ANTALYA KÜTAHYA	ANTALYA	ANTALYA	ANTALYA
DENİZLİ	DENİZLİ TUNCELİ	DENİZLİ	DENİZLİ	DENİZLİ	DENİZLİ
KOCAELİ		KIRKLARELİ KOCAELİ(İZMİT)			
ESKİŞEHİR	ESKİŞEHİR	ESKİŞEHİR KIRŞEHİR NEVŞEHİR	ESKİŞEHİR	ESKİŞEHİR	ESKİŞEHİR

Table 4.1b: Soundex and Levenshtein Distance of Sample Cities (cont)

ANKARA	ANKARA	ADANA AKSARAY ANKARA ANTALYA ÇANKIRI HAKKARİ KARS	ANKARA	ANKARA	ANKARA
BURSA	BURSA	BURDUR BURSA	BURSA	BURSA	BURSA

Soundex algorithm gives more than one results for some cases and also does not give any results for one case in Table 4.1a,b. Similarity ratio cannot be computed for the Soundex algorithm to decrease or to increase the ratio for finding different results. LD gives too many results at 50 percent similarity and does not give any results at 70 percent for two cases as seen in Table 4.1a,b. Nevertheless LD gives wrong result for "MERSİN". Since the results taken from Soundex and LD are not good enough, application does not use these algorithms for city similarity comparisons.

Table 4.2a: Soundex and Fast Distance of Sample Cities

City	**SOUNDEX**	**FD %50**	**FD %60**	**FD %65**	**FD %70**
ADANA	ADANA AYDIN	ADANA ADIYAMAN AMASYA	ADANA	ADANA	ADANA
MERSİN	MERSİN(İÇEL)	MARDİN MERSİN(İÇEL)	MARDİN MERSİN(İÇEL)	MARDİN MERSİN(İÇEL)	MERSİN(İÇEL)
ANTALYA	ANTALYA	ANKARA ANTALYA KÜTAHYA	ANTALYA	ANTALYA	ANTALYA
DENİZLİ	DENİZLİ TUNCELİ		DENİZLİ	DENİZLİ	DENİZLİ
KOCAELİ		KIRŞEHİR KOCAELİ(İZMİT)	KOCAELİ(İZMİT)	KOCAELİ(İZMİT)	KOCAELİ(İZMİT)

Table 4.2b: Soundex and Fast Distance of Sample Cities (cont)

ESKİŞEHİR	ESKİŞEHİR	ESKİŞEHİR KIRŞEHİR NEVŞEHİR	ESKİŞEHİR	ESKİŞEHİR	ESKİŞEHİR
ANKARA	ANKARA	AKSARAY ANKARA ANTALYA ARDAHAN	AKSARAY ANKARA	AKSARAY ANKARA	ANKARA
BURSA	BURSA	BURDUR BURSA MERSİN(İÇEL) ŞIRNAK	BURDUR BURSA	BURDUR BURSA	BURSA

FD suits best for city similarity comparisons because it gives correct results at 70 percent similarity ratios at all city samples as shown in Table 4.2a,b. It is specified in previous chapter that fast distance algorithm is used for city verification at 70 percent similarity level.

4.2 CASE II: COUNTIES

In Case 2, the algorithm performances are examined for county names. The selected counties are processed by using Soundex, FD, LD and the results are documented in Table 4.3a,b and Table 4.4.

Table 4.3a: Soundex and Levenshtein Distance of Sample Counties

County	**SOUNDEX**	**LD %50**	**LD %60**	**LD %65**	**LD %70**
AVCILAR	AVCILAR	AVCILAR BAĞCILAR ADALAR	AVCILAR BAĞCILAR	AVCILAR BAĞCILAR	AVCILAR BAĞCILAR
SEYHAN	SEYHAN CEYHAN	SEYHAN CEYHAN	SEYHAN CEYHAN	SEYHAN CEYHAN	SEYHAN CEYHAN
ŞİŞLİ	ŞİŞLİ ŞİLE	ŞİŞLİ ŞİLE	ŞİŞLİ ŞİLE	ŞİŞLİ	ŞİŞLİ
OSMANGAZİ	OSMANGAZİ	OSMANGAZİ ORHANGAZİ	OSMANGAZİ ORHANGAZİ	OSMANGAZİ ORHANGAZİ	OSMANGAZİ ORHANGAZİ

Table 4.3b: Soundex and Levenshtein Distance of Sample Counties

BAĞCILAR	BAĞCILAR	AVCILAR BAĞCILAR ADALAR	AVCILAR BAĞCILAR	AVCILAR BAĞCILAR	AVCILAR BAĞCILAR
YILDIRIM	YILDIRIM	YILDIRIM	YILDIRIM	YILDIRIM	YILDIRIM
AVC					

When it comes to county comparison, quantity of results for each case decreases as seen in Table 4.3a,b and Table 4.4. Soundex again does not give unique results. LD and FD algorithm gives almost same results this time. It is used FD algorithm for county similarity check because it is faster than LD algorithm and results are almost same between Table 4.3a,b and Table 4.4. It is specified in previous chapter that fast distance algorithm is used for county verification at 60 percent similarity level.

Table 4.4: Soundex and Fast Distance of Sample Counties

County	**SOUNDEX**	**FD %50**	**FD %60**	**FD %65**	**FD %70**
AVCILAR	AVCILAR	AVCILAR BAĞCILAR ADALAR	AVCILAR BAĞCILAR	AVCILAR	AVCILAR
SEYHAN	SEYHAN CEYHAN	SEYHAN CEYHAN	SEYHAN CEYHAN	SEYHAN CEYHAN	SEYHAN CEYHAN
ŞİŞLİ	ŞİŞLİ ŞİLE	ŞİŞLİ ŞİLE	ŞİŞLİ	ŞİŞLİ	ŞİŞLİ
OSMANGAZİ	OSMANGAZİ	OSMANGAZİ ORHANGAZİ	OSMANGAZİ ORHANGAZİ	OSMANGAZİ ORHANGAZİ	OSMANGAZİ ORHANGAZİ
BAĞCILAR	BAĞCILAR	AVCILAR BAĞCILAR	BAĞCILAR	BAĞCILAR	BAĞCILAR
YILDIRIM	YILDIRIM	YILDIRIM	YILDIRIM	YILDIRIM	YILDIRIM
AVC		AVCILAR ADALAR	AVCILAR	AVCILAR	AVCILAR

4.3 CASE III: DISTRICTS

In Case 3, the algorithm performances are examined for district names. The selected districts are processed by using Soundex, FD, LD and the results are documented in Table 4.5 and Table 4.6a,b.

Table 4.5: Soundex and Levenshtein Distance of Sample Districts

District	SOUNDEX	LD %50	LD %60	LD %65	LD %70
GÜLTEPE	GÜLTEPE	GÜLTEPE ÇELİKTEPE	GÜLTEPE	GÜLTEPE	GÜLTEPE
GOZTEPE	GÖZTEPE	GÖZTEPE	GÖZTEPE	GÖZTEPE	GÖZTEPE
AKDENİZ	AKDENİZ	AKDENİZ	AKDENİZ	AKDENİZ	AKDENİZ
ŞŞLİ	ŞİŞLİ ŞEYHLİ ŞİLE	ŞİŞLİ ŞEYHLİ ŞİLE	ŞİŞLİ	ŞİŞLİ	ŞİŞLİ
GLTPE	GÜLTEPE	GÜLTEPE MALTEPE GÖZTEPE	GÜLTEPE	GÜLTEPE	GÜLTEPE
RAMİ	RAMİ	ARAPCAMİ RAMİ	RAMİ	RAMİ	RAMİ
VEFA	VEFA	VEFA	VEFA	VEFA	VEFA

When the results of Table 4.5 and Table 4.6 are compared, it is seen that Soundex algorithm works well except the word “ŞŞLİ”. Soundex works well on true written words which are already controlled and does not need further controlling. So Soundex is eliminated. LD finds fewer results compare to FD algorithms at lower similarity rates. It is specified in previous chapter that Levenshtein algorithm is used for county verification at 60 percent similarity level.

Table 4.6: Soundex and Fast Distance of Sample Districts

District	SOUNDEX	FD %50	FD %60	FD %65	FD %70
GÜLTEPE	GÜLTEPE	GÜLTEPE SEYRANTEPE	GÜLTEPE	GÜLTEPE	GÜLTEPE
GOZTEPE	GÖZTEPE	GÖZTEPE	GÖZTEPE	GÖZTEPE	GÖZTEPE
AKDENİZ	AKDENİZ	AYDINCIK AKDENİZ	AKDENİZ	AKDENİZ	AKDENİZ
ŞŞLİ	ŞİŞLİ ŞEYHLİ ŞİLE	ÇELİKTEPE ŞİŞLİ HALİDEEDİP GÜNEŞLİ ŞEYHLİ SELİMİYE SELİMPAŞA	ŞİŞLİ	ŞİŞLİ	ŞİŞLİ
GLTPE	GÜLTEPE	ALTINTEPSİ GAYRETTEPE GÜLTEPE ALTUNİZADE GÜRPINAR			
RAMİ	RAMİ	ARAPCAMİ RAMİ HAMİDİYE RUMELİHİSARI HALİDEEDİP DEMİRKAPI RASİMPAŞA RAHMANLAR	RAMİ HAMİDİYE	RAMİ	RAMİ
VEFA	VEFA	KEMANKEŞ BEYAZIT VEFA SEFAKÖY KEMALPAŞA REŞADİYE CELALİYE NECİPFAZIL	VEFA SEFAKÖY	VEFA SEFAKÖY	VEFA SEFAKÖY

On the other hand, success rates of fast distance algorithm on short words (5 characters or less) observable decrease. On the other hand Levenshtein algorithm gives acceptable results at 60 percent similarity and over rates for all cases.

4.4 CASE IV: PARISHES

In case 4, comparison results of parishes are given at Table 4.7 and 4.8. Parishes are word or word groups where many faults occur. Especially when district or county or both of them are empty, quantity of words to compare on reference dataset increases and that concludes with worse success rate.

Table 4.7: Soundex and Levenshtein Distance of Sample Parishes

Parish	SOUNDEX	LD %50	LD %60	LD %70
NAMIK KEMAL		NAMIKKEMAL	NAMIKKEMAL	NAMIKKEMAL
TOROS	TOROS	BARBOROS TOROS	TOROS	TOROS
MEHMETCIK	MEHMETÇİK MAH.MUTSÖNMEZ	MEHMETÇİK EMECİK MEHMET AKİF	MEHMETÇİK MEHMET AKİF	MEHMETÇİK
GUMUSP				
MEYDAN KAVAGI	MEYDANKAVAĞI	MEYDANKAVAĞI	MEYDANKAVAĞI	MEYDANKAVAĞI

Table 4.8: Soundex and Fast Distance of Sample Parishes

Parish	SOUNDEX	FD %50	FD %60	FD %70
NAMIK KEMAL		NAMIKKEMAL	NAMIKKEMAL	NAMIKKEMAL
TOROS	TOROS	BARBOROS TOROS	BARBOROS TOROS	TOROS
MEHMETCIK	MEHMETÇİK MAH.MUTSÖNMEZ	MEHMET AKİF ERSOY MEHMETÇİK MAHMUTGAZİ MEHMET AKİF	MEHMET AKİF ERSOY MEHMETÇİK MEHMET AKİF	MEHMETÇİK MEHMET AKİF
GUMUSP		GÜMÜŞPALA		
MEYDAN KAVAGI	MEYDANKAVAĞI	MEYDANKAVAĞI	MEYDANKAVAĞI	MEYDANKAVAĞI

Good results are taken at 70 percent similarity levels at both of LD and FD algorithms. Soundex does not give results for some cases again. But using LD with lower rates gives more precise results and that is why it is specified in previous chapter that Levenshtein algorithm is used for county verification at 50 percent similarity levels.

4.5 CASE V: FLOW OF THE APPLICATION

It is time to look the application step by step in case 5. Figure 4.1 shows the starting window of the application. Sample and reference datasets are loaded when “Adresleri Yükle” is clicked and required operations are executed in order.

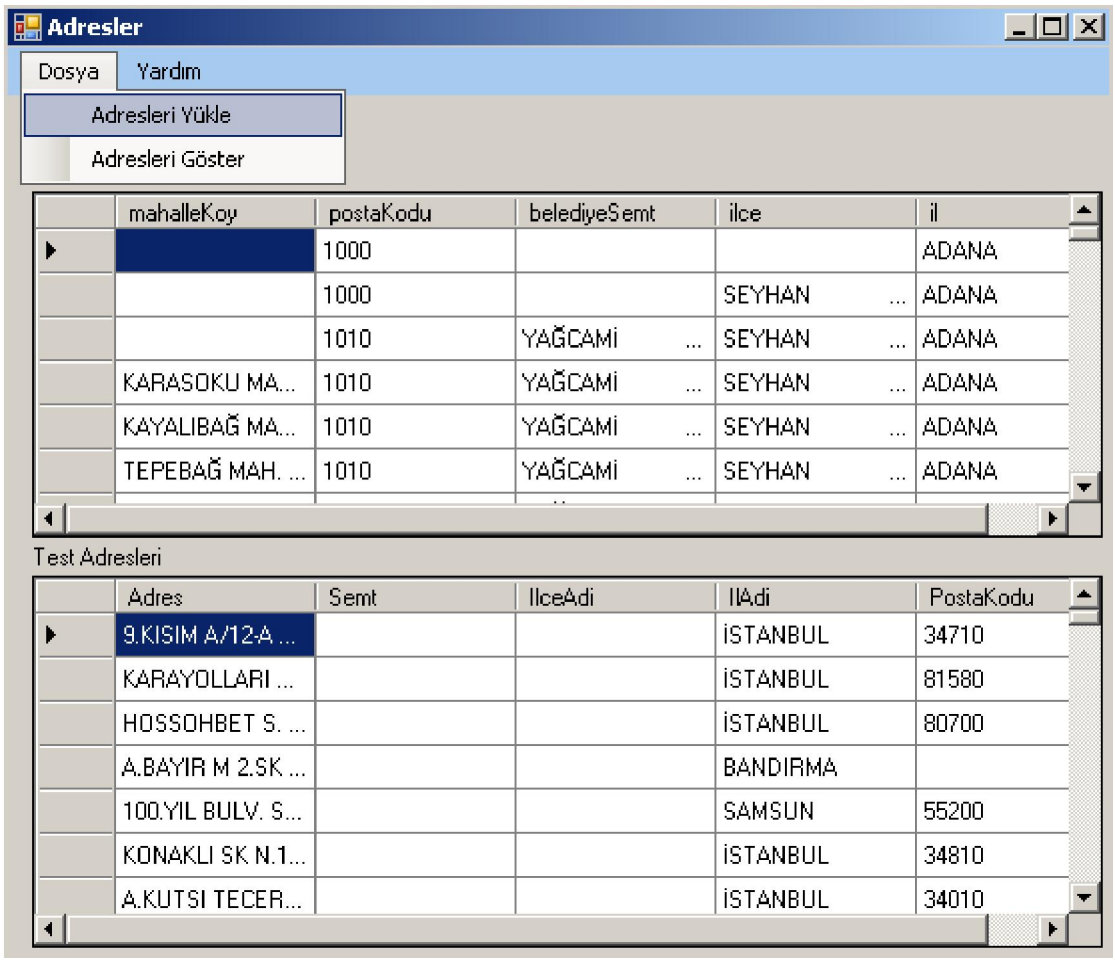


Figure 4.1: Adresler Window of the Application

After the execution of process is completed, required tasks are accomplished and status of sample addresses change from Table 4.9 to Table 4.10.

Table 4.9: Adres Table Before First Replacements

Adres	Semt	IlceAdi	IlAdi	PostaKodu
9.KISIM A/12-A BL K.4 D.12 ATAKOY			İSTANBUL	34710
KARAYOLLARI C GUMUSP AYAN ST D BL D 5 AVC			İSTANBUL	81580
HOSSOHBET S. ELBASAN AP. 7/9 D.11 B,TAS			İSTANBUL	80700
A.BAYIR M 2.SK CIKM GUNES AP 48/5			BANDIRMA	
100.YIL BULV. SISIK AP. 131/7			SAMSUN	55200
KONAKLI SK N.19 FLORYA			İSTANBUL	34810
A.KUTSI TECER CD 3/23 MERTER			İSTANBUL	34010
1376 S 10/14 ALSANCAK			İZMİR	35210
A.CETINKAYA BLV. 40/13 ALSANCAK			İZMİR	35220

Table 4.10: Adres Table After First Replacements

Adres	Semt	IlceAdi	IlAdi	PostaKodu
9. KISIM A/12-A BLOK K. 4 D. 12 ATAKOY			İSTANBUL	34710
KARAYOLLARI CAD. GUMUSP AYAN SİTESİ D BLOK D 5 AVC			İSTANBUL	81580
HOSSOHBET SOK. ELBASAN APT. 7/9 D. 11 B,TAS			İSTANBUL	80700
A. BAYIR MAH. 2. SOK. CIKM GUNES APT. 48/5			BANDIRMA	
100. YIL BLV. SISIK APT. 131/7			SAMSUN	55200
KONAKLI SOK. NO:19 FLORYA			İSTANBUL	34810
A. KUTSI TECER CAD. 3/23 MERTER			İSTANBUL	34010
1376 SOK. 10/14 ALSANCAK			İZMİR	35210
A.CETINKAYA BLV. 40/13 ALSANCAK			İZMİR	35220

Space cleaning and abbreviation replacements are completed and the addresses are reclassified after “Adresleri Yükle” is clicked. “Adresleri Göster” is clicked to see the last status of the addresses in next window. Next window that is seen in Figure 4.2 shows up. 436 fifth class, 348 fourth class, 205 third class, 128 second class, 219 first class and 9 zero class address can be seen in this window. The bigger address level means better address quality. Remaining improvement operations such as similarity check and reclassification are executed in this window.

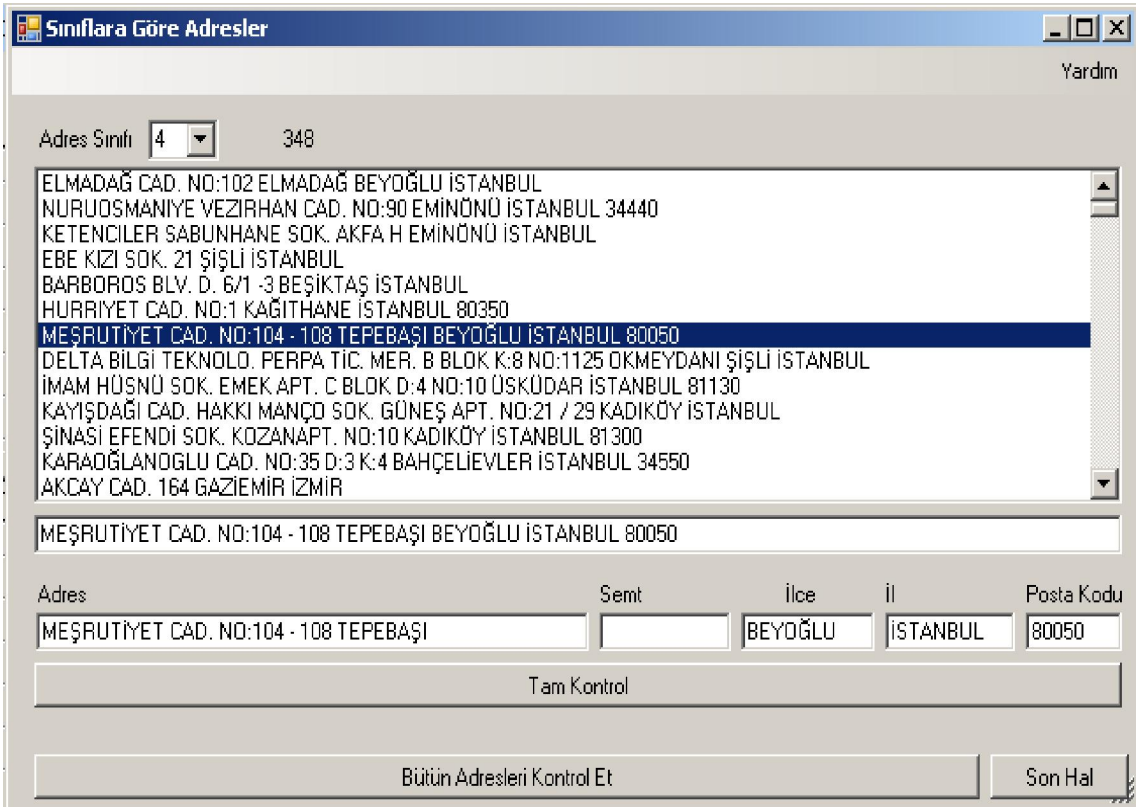


Figure 4.2: Address Shown in Siniflara Göre Adresler Window of the Application

"Tam Kontrol" button needs to be pressed to check an individual address. Table 4.11 shows some example outputs produced by the application after the button is pressed.

Table 4.11: Addresses Before and After Verification

Address	**District**	**County**	**City**	**Postal Code**
9032 SOK. 8/2 YESILYURT			İZMİR	35370
9032 SOK. 8/2	YESILYURT	KONAK	İZMİR	35160
75. YIL MAH. PALMA 2 SİTESİ A-4 BLOK D:1	DAVULTEPE		MERSİN	0
75.YIL MAH. PALMA 2 SİTESİ A-4 BLOK D:1	DAVULTEPE	AKDENİZ	MERSİN(İÇEL)	33320
SAGLIK MAH. 277 SOK. NO:5/3			MERSİN	0
SAĞLIK MAH. 277 SOK. NO:5/3			MERSİN(İÇEL)	33080
ORHANTEPE MAH. DEFNE SOK. OZYAVUZ APT. 10/3 CEVIZLI		KARTAL	İSTANBUL	81510
ORHANTEPE MAH. DEFNE SOK. OZYAVUZ APT. 10/3 CEVIZLI		KARTAL	İSTANBUL	34865

“Tam Kontrol” button checks one address. To check all addresses “Bütün Adresleri Kontrol Et” button is pressed. The overall process takes 9 minutes 19 seconds 613 milliseconds on HP Pavilion DV2190EA series notebook which have 1.83 GHz Core 2 Duo processor and 2 GB Ram with Windows XP operating system when “Bütün Adresleri Kontrol Et” button is pressed. To see the results in next window shown by Figure 4.3, “Son Hal” button is pressed.

It takes 3 minutes 6 seconds 664 milliseconds to execute 436 fifth class address data. Addresses are reclassified at the end of execution and 12 fourth class and 424 fifth class addresses are obtained. 81 of 424 fifth class addresses and 4 of 12 fourth class addresses cannot be improved. The success criteria for fifth class addresses is finding suitable parishes from dataset and improving quality by also verifying postal codes. Addresses which are not improved mostly do not have actual parishes and there is no match found from reference dataset. It is achieved 81 percent success rate on improving address quality of fifth class addresses.

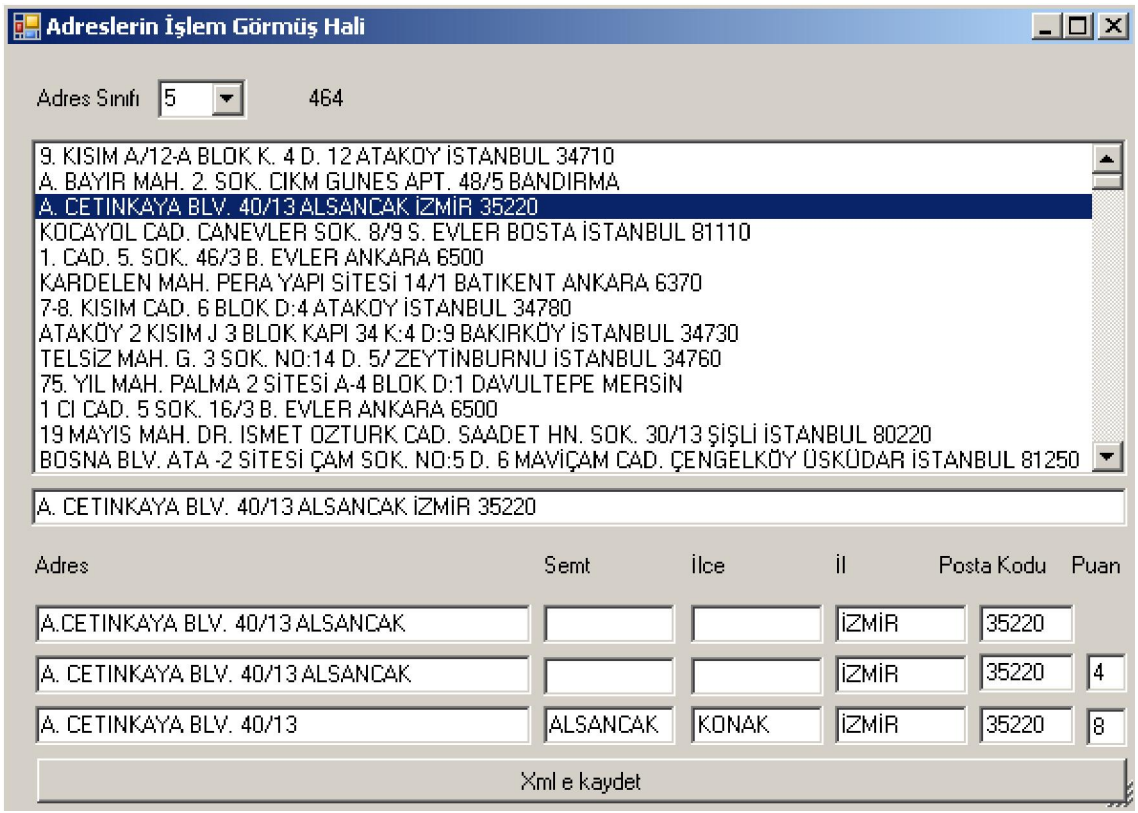


Figure 4.3: Three Status of an Address in Adreslerin İşlem Görmüş Hali Window of the Application

It takes 53 second 482 milliseconds to execute 348 fourth class address data. Addresses are reclassified at the end of execution and 5 fifth class and 343 fourth class addresses are obtained. 2 of 5 fifth class addresses and 29 of 343 fourth class addresses cannot be improved. Fourth class addresses mostly do not have parish or village information so the success criteria is finding hidden district or county words within address column and extract them to related address component column and then finding better postal codes. It is achieved 91 percent success rate on improving address quality of fourth class addresses.

It takes 3 minutes 30 second 571 milliseconds to execute 205 third class address data. Addresses are reclassified at the end of execution and 35 fifth class, 22 fourth class, 145 third class, 2 second class and 1 zero class addresses are obtained. 4 of 35 fifth class addresses, 4 of 22 fourth class addresses, 62 of 145 third class addresses, 1 of 2 second class and 1 of 1 zero class addresses cannot be improved. Addresses which increase its class level gives good results but addresses which still stay at third or lower class are hard to be improved. Because most of the districts or counties that are hidden in address column of third class addresses mostly cannot be identified or does not exist in reference dataset. It is achieved 60 percent success rate on improving address quality of third class addresses.

It takes 16 second 988 milliseconds to execute 128 second class address data. Addresses are reclassified at the end of execution and 54 fourth class, 72 second class, 1 first class and 1 zero class addresses are obtained. 5 of 54 fourth class addresses, 65 of 72 second class, 1 of 1 first class and 1 of 1 zero class address cannot be improved. Same things are also acceptable for second class addresses like third class addresses. Less address component means less quality and it is harder to improve these addresses. It is achieved 45 percent success rate on improving address quality of second class addresses.

It takes 33 second 629 milliseconds to execute 219 first class address data. Addresses are reclassified at the end of execution and 218 first class and 1 zero class addresses are obtained. 143 of 218 first class and 1 of 1 zero class addresses cannot be improved.

When it comes to the first class addresses there is almost nothing to improve because most of them are addresses of private places such as municipalities, military zones, schools, government offices, police station, private companies etc. and mostly only the name of association and city are written into the addresses. So success rate decreases and it is achieved 35 percent success rate on improving address quality of first class data. Results of all class improvements can be seen in Table 4.12.

Table 4.12: Improvement Rate of Addresses at Different Classes Before Reclassification

Class	**Rate**	**Class**	**Rate**
5	81%	**2**	45%
4	91%	**1**	30%
3	60%	**0**	0%

5. CONCLUSION

Application has different success rates on improving address quality according to their class; 81 percent success rate on fifth class, 91 percent success rate on fourth class, 60 percent success rate on third class, 45 percent success rate on second class and 30 percent success rate on first class addresses as shown in Table 4.12. As a result of these rates;

i. Fifth class addresses' quantity increases from 436 to 464 with 377 improved addresses,
ii. Fourth class addresses' quantity increases from 348 to 431 with 389 improved addresses,
iii. Third class addresses' quantity decreases from 205 to 145 with 83 improved addresses,
iv. Second class addresses' quantity decreases from 128 to 74 with 8 improved addresses,
v. First class addresses' quantity stays same at 219 with 74 improved addresses,
vi. Zero class addresses' quantity increases from 9 to 12 with none improved addresses.

Table 5.1 shows that success rate for improved addresses that are 81 percent for fifth class, 90 percent for fourth class, 55 percent for third class, 10 percent for second class, 33 percent for first class and 0 percent for zero class addresses. As a result of Table 5.1 quantities in Table 5.2 are obtained.

Table 5.1: Improvement Rate of Addresses at Different Classes After Reclassification

Class	**Rate**	**Class**	**Rate**
5	%81	**2**	%10
4	%90	**1**	%33
3	%55	**0**	%0

Improvement rates are acceptable for fifth and fourth class addresses and but not for the rest. The main reason of lack of improvement for lower class addresses is the status of reference dataset used. The reference dataset provided by PTT contains postal code

information up to parishes or villages. It has no information about streets, highroads, sites, avenue etc. Addresses of lower classes mostly have street and highroad information and application of this study cannot perform any execution on these parts of addresses. So the main need for better performance on improvement is a better reference dataset which covers every area of Turkey up to its each point.

Table 5.2: Address Quantities at Classification Levels

Class	Before	After	Class	Before	After
5	436	464	**2**	128	74
4	348	431	**1**	219	219
3	205	145	**0**	9	12

Fifth class addresses have parishes or villages and also contains nonempty district, county and city components. Reference dataset contains this information and improvement level is acceptable but still lower than fourth class because the success criteria for fifth and fourth level addresses differ. Fifth class addresses must verify parish or village; fourth class addresses must extract hidden district or county components. Success of this study should be evaluated with success rates of fourth and fifth class addresses in existence of used reference dataset.

Another reason for lack of improvement is that the character set used is Turkish. Whether preferring Turkish characters or not effects the similarity accuracy of fast distance and Levenshtein algorithms. If all non-English characters are disabled, improvement rates increase because most of the addresses of test dataset of this study use only English characters. If Turkish characters are omitted quality of the addresses can increase but readability of the addresses might decrease.

The study presented here can further be advanced in many ways.

i. N-gram analysis can be applied for non-word error correction.
ii. All characters in datasets might be replaced with English characters.
iii. Similarity rates of edit distance algorithms can be changed.
iv. Rule based techniques or probabilistic techniques can be applied.
v. County or city based improvement tasks can be applied.

REFERENCES

Books

Maimon, O., Rokach, L., 2005. *The Data Mining and Knowledge Discovery Handbook.* Springer US

Periodical Publications

Aşlıyan, R., Günel, K., Yakhno, T., 2007. Detecting Misspelled Words in Turkish Text Using Syllable n-gram Frequencies. *Lecture Notes in Computer Science (LNCS),* 4815, pp. 553-559.

Christen, P., 2005. Probabilistic data generation for deduplication and data linkage. *Springer Lecture Notes in Computer Science,* 3578, 109-116.

Domingos, P., 1998. Knowledge Discovery Via Multiple Models. *Intelligent Data Analysis,* 2(1), pp. 187-202.

Hall, P.A.V., *Dowling,* D.R., 1980. *Approximate String Matching. Computing Surveys,* 12(4), pp. 381-402.

Hernandez, M.A., Stolfo S.J., 1997. Real-world data is dirty*:* Data Cleansing and the Merge/Purge problem*. Journal of* Data *Mining and Knowledge Discovery,* 2, pp. 9-37.

Kukich, K., 1992. Techniques for automatically correcting words in text. *ACM Computing Survey*. 24, 4, pp. 377-439.

Mois, P., Sepúlveda, M., Pröschle, H., 2005. Street address correction based on spelling techniques. *Lecture Notes in Computer Science,* 3567, pp. 166-172.

Navarro, G., 2001. A Guided Tour to Approximate String Matching. *ACM Computing Survey* 33, 1, pp. 31-88.

Pollock, J.J., Zamora, A., 1987. Automatic spelling correction in scientific and scholarly text. *ACM Computing Surveys*, 27(4): pp. 358–368.

Shulz, K.U. Mihov, S., 2002. Fast String Correction with Levenshtein-Automata. *International Journal of Document Analysis and Recognition*, 5, 1, pp. 67-85.

Wagner, R.A., Fischer, M.J., 1974. The string-to-string correction problem. *Journal of the ACM* , 21(1): pp. 168-173.

French, C.J., Powell, A.L., Schulman, E., 1998. Applications of Approximate Word Matching in Information Retrieval. *Proceedings of the Sixth International Conference on Knowledge and Information Management*, pp. 9-15.

Nagabhushan, P., 1998. Towards Automation in Indian Postal Services: A Loud Thinking", *Technovision* , Spl Volume, pp. 128-139

Nagabhushan, P., Angadi, S.A., Anami, B.S., 2005. Symbolic Data Structure for Postal Address Representation and Address Validation Through Symbolic Knowledge Base. *PReMI* , pp. 388-394.

Other Publications

Hernandez, M.A., Stolfo, S.J. 1995. The merge/purge problem for large databases. *Proceedings of the ACM SIGMOD conference.*

Zobel, J., Dart, P., 1996. Phonetic string matching**:** Lessons from information retrieval. *Proc. 19th International Conference on Research and Development in Information Retrieval.*

Address 2008. http://en.wikipedia.org/wiki/Address_(geography) [cited November 2008]

C# 2008.http://en.wikipedia.org/wiki/C_Sharp_(programming_language) [cited November 2008]

Daitch Mokotoff 2008. http://en.wikipedia.org/wiki/Daitch-Mokotoff_Soundex [cited November 2008]

Double Metaphone 2008. http://en.wikipedia.org/wiki/Double_Metaphone [cited November 2008]

Levenshtein 2008. http://en.wikipedia.org/wiki/Levenshtein_distance [cited November 2008]

Levenshtein 2008. http://www.merriampark.com/ld.htm [cited November 2008]

.NET Framework 2008. http://en.wikipedia.org/wiki/.NET_Framework [cited November 2008]

Postal Code 2008. http://www.ptt.gov.tr/tr/interaktif/postakodu2.html [cited November 2008]

Research about Address 2008. http://www.gislab.ktu.edu.tr [cited November 2008]

Soundex 2008. http://www.creativyst.com
[cited November 2008]

SQL Server 2008. http://en.wikipedia.org/wiki/Microsoft_SQL_Server
[cited November 2008]

Visual Studio .NET 2008. http://en.wikipedia.org/wiki/Visual_Studio_.NET
[cited November 2008]

XML 2008. http://en.wikipedia.org/wiki/XML
[cited November 2008]

CURRICULUM VITAE

Name Surname : Özgür TUFAN

Address : Gümüşpala Mah. Gümüşpala Cad.
Odabaş Apt. No: 38/8
34320 Avcılar / İstanbul / Türkiye

Birth Place / Year : Trabzon - 1984

Languages : Turkish (native) - English

Elementary School : Bakırköy Primary School – 1995

High School : Trabzon Yomra Science High School - 2002

BSc : University of Bahçeşehir - 2006

MSc : University of Bahçeşehir – 2008

Name of Institute : Institute of Science

Name of Program : Computer Engineering

Publications :

Work Experience : January 2008 – July 2008
Software Developer
Signera

August 2006 – August 2007
Teaching and Research Assistant
University of Bahçeşehir, Software Engineering Department