

Address Name Resolution for Geocoding in Efficient Taxi Dispatching Services

Ali Yoseph Solomon

Date: May, 2020

Address Name Resolution for Geocoding in Efficient Taxi Dispatching Services

Candidate: Ali Yoseph Solomon

School of Study: School of Software

Student's Number: 3820181010

Supervisor: 汤世平

Chair of Defense Committee:

Degree Applied: Master of Software Engineering

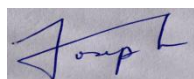
Major: Software Engineering

Date of Defense: September 7, 2020

Declaration of Originality

I hereby declare that this thesis is the result of an independent research that I have made under the supervision of my supervisor. To the best of my knowledge and belief, it does not contain any other published or unpublished research work of others, or any material which has been accepted for the award of another degree or diploma at Beijing Institute of Technology or other educational institutions, except where due acknowledgment has been made in the text. Whoever has contributed to this study is explicitly identified and appreciated in the Acknowledgements of the thesis.

Signature of Author:

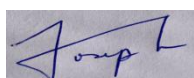


Date: September 30, 2020

Authorization Statement

I fully understand the regulations of Beijing Institute of Technology regarding the preservation and use of the degree thesis. BIT is granted the right to (1) preserve the original thesis and the copies of the thesis, as well as submit them to relevant authorities; (2) reproduce and preserve the thesis by means of photocopy, reduction printing and any other means; (3) keep the thesis for reference and borrowing; (4) reproduce, give and exchange the thesis for the purpose of academic exchange; (5) publish the thesis wholly and partially. (The regulations comes into force for confidential thesis only after declassification)

Student's signature:



Date: September 30, 2020

Supervisor's signature:



Date: September 30, 2020

研究成果声明


本人郑重声明:所提交的学位论文是我本人在指导教师的指导下进行的研究工作获得的研究成果。尽我所知,文中除特别标注和致谢的地方外,学位论文中不包含其他人已经发表或撰写过的研究成果,也不包含为获得北京理工大学或其它教育机构的学位或证书所使用过的材料。与我一同工作的合作者对此研究工作所做的任何贡献均已在学位论文中作了明确的说明并表示了谢意。

特此申明。

签 名:  日期: 09 月 30 日 2020 年

关于学位论文使用权的说明

本人完全了解北京理工大学有关保管、使用学位论文的规定,其中包括:①学校有权保管、并向有关部门送交学位论文的原件与复印件;②学校可以采用影印、缩印或其它复制手段复制并保存学位论文;③学校可允许学位论文被查阅或借阅;④学校可以学术交流为目的,复制赠送和交换学位论文;⑤学校可以公布学位论文的全部或部分内容(保密学位论文在解密后遵守此规定)。

签 名:  日期: 09 月 30 日 2020 年

导师签名:  日期: 09 月 30 日 2020 年

Acknowledgments

I thank Bas Ranzijn from university of Rotterdam for the great help in writing this thesis.

Thank you.

Abstract

With the growing of online taxi-hailing service market in developing cities such as Ethiopia's capital, the demand from the public to use their technology backed and low-cost services has also become urgent. Due to the high demand for safe and modernized transportation, most local taxi-hailing service providers have quickly gained plenty popularity over the past couple of years. Unlike other industrialized countries, many address systems in developing countries lack standards, making the addressing system ambiguous, incomplete or imprecise. Due to the lack of good addressing and geocoding system, as well as internet connectivity, service delivery to customers is a challenge for companies. Accurate interpretation of textual address information is a key issue to be solved in these systems.

A solution for address name resolution is proposed for such service providers, in which address data collected through customer service calls is to be geocoded to the corresponding geographical coordinates. As the data collected is characterized by its vagueness and full of misspelling or unknown places from common mapping facilities, the proposed solution implements string clustering algorithms for domain dictionary generation. The extracted domain specific dictionary is then further used in an n-gram based spell-checking algorithm. For each error word detected using the spell-checking algorithm, a candidate word is generated using string similarity measures. Each candidate address is then ranked using a cosine similarity measure. The processed address information is then geocoded using the Google Geocoding API. Experiments conducted on location data collected from a taxi hailing platform showed a remarkable increase in geocoding match rate. The results obtained from the geocoding process will help taxi-hailing service providers to do further spatial analysis as well as design and model a smart transportation systems.

Key words: Geocoding, Spell-checking, Taxi-hailing, Address Matching, String Similarity

Table of Contents

Acknowledgments.....	IV
Abstract.....	V
Table of Contents.....	VI

CHAPTER 1:

INTRODUCTION.....	VIII
-------------------	------

1.1 INTRODUCTION	1
1.2 PROBLEM DESCRIPTION	2
1.3 RESEARCH QUESTIONS	3
1.4 METHODOLOGY	5
1.5 THESIS OUTLINE	6

CHAPTER 2: LITERATURE REVIEW 7

2.1 GEOCODING	7
2.2 SPELL-CHECKING ALGORITHMS.....	9
2.2.1 ERROR DETECTION	10
2.2.2 CANDIDATE GENERATION AND ERROR CORRECTION	11
2.3 SIMILARITY METHODS.....	13
2.3.1 CHARACTER BASED SIMILARITY MEASURES	14
2.3.1.1 EDIT DISTANCE	14
2.3.1.2 JARO WINKLER	17
2.3.1.3 N-GRAMS	19
2.3.2 TOKEN BASED SIMILARITY MEASURES.....	20
2.3.2.1 TF-IDF	20
2.3.2.2 MONGE AND ELKAN'S RECURSIVE MATHCHING	21
2.3.3 PHONETICS SIMILARITY MEASURES.....	21

CHAPTER 3:SIMILARITY MEASURES TEST..... 26

3.1 SYSTEM OVERVIEW.....	26
3.2 TEST DESIGN.....	27
3.3 MEASURES FOR TESTING.....	28
3.4 DATASET.....	29
3.5 MEASURES TEST RESULT 32	

CHAPTER 4: ADDRESS MATCHING FOR GEOCODING..... 35

4.1 ADDRESS MATCHING AND SPELL CHECKING.....	35
--	----

4.2 METHODOLOGY.....	37
4.3 GEOCODING AND EVALUATION 37	
4.3.1 GEOCODING	38
4.3.2 EVALUATION	38
CHAPTER 5: CONCLUSION AND FUTURE WORK.....	41
5.1 RESULT AND CONCLUSION.....	41
5.2 IMPLICATION AND FUTURE WORK.....	42
REFERENCES	43

Chapter 1: Introduction

1.1 Introduction

Geocoding is the process of converting textual information describing spatial entities like addresses into one or more digital geographic representations. It is a routine task performed by organizations, of all sizes ranging from non-profit and commercial entities to local-, state- and national-level government agencies, as well as research groups and individuals, are required to perform geocoding for numerous mission-critical tasks. Digital maps and digital processing of location information are gaining high popularity in various commercial and non-profit applications for automated processing of location data. In order to navigate to an unknown area, users rely on computers to retrieve, display and store location information. Human users prefer locations by addresses or common names. Computers, on the other hand, reference locations through a coordinate system such as WGS84 latitude and longitude coordinates. The process of mapping such names or addresses to their location on a coordinate system is called *geocoding*.

Addresses are the most common georeferencing resource people use to communicate with others. An address is an identifier for a place where a person, organization, or the like is located or can be found [17]. Coordinate, on the other hand, is a group of numbers, longitude, and latitude, used to indicate a position in space. Addresses only truly matter where there are people. Therefore, addresses are usually a common reference to events and phenomena that take place in urban areas. As a result, most urban GIS applications, such as transportation, public health, public safety and more, are required to generate coordinates from addresses--as provided by the general public--in order to be able to visualize and analyze their data. Information on addresses, in relation to personal data, landmarks and points of interest (POI), is commonly stored in databases where each data will typically reference one particular geographic location or region.

As a result of advancements in technology, users can geocode text quickly using many online geocoding services like those offered by Google, OpenStreetMap, BING Location API, QGIS or HERE that are available both paid and free. Google's Geocoding API is certainly one of the best known options, when it comes to using an API to geocode addresses. Many online taxi-hailing platforms have benefited from the usage of these geocoding services enabling fast access to map-based geolocation. However, even though these geocoding services help in making the process of conventional geocoding easy, uncertainties concerning reference databases, positional errors and geocoding algorithms remain. There are two aspects to this error-prone process, first, the geocoding system needs to parse user query and derive the query intent, i.e. the platform needs to understand the address entity the query is referring to. Then the system needs to look up the coordinates of

the entity the query was referring to and return it as a result. Considering the human factor, where some address elements are often misspelled or abbreviated by users in a non-standard way, the first step is a non-trivial task. On the other hand, in contrast with industrialized countries, many address systems in developing countries lack standards, making the addressing system ambiguous, incomplete or imprecise. Considering addresses from all over the world, there is no pattern that all user queries fit in because address formats often contradict each other. On top of that, human users, like with spelling errors, may not adhere to a format, leaving names of address elements out or specifying them in an unexpected order. Such incomplete and missorted queries are often reused for different and unrelated address elements making them ambiguous. However, even with the best geocoding algorithm at hand, geocoding service can be as good as the data, reference data, it builds upon.

A problem that arises in geocoding is that the description of a location as requested by a user, may differ from the description of the location in the database. This may be due to misspellings, alternative spelling, abbreviations, different word order, and addition or omission of words. This thesis investigates the problem of matching identical addresses that differ due to these errors, examines the challenges of address classification and devises an accurate address classification and geocoding solution.

1.2 Problem Description

The subject of this thesis follows directly from a problem encountered in reality. The trend of the online ride-hailing service market is growing in Ethiopia's capital, Addis Ababa, and so is the demand from the public to use the technology backed and low-cost services. Due to the high demand for safe and modernized transportation, online taxi-hailing services, ETTA (Ethiopia Taxi), ZayRide, PickPick Meter Taxi, and Ride, some of the common platforms that are dominating the market in the capital, have gained plenty popularity since being introduced in 2017. Due the high popularity and demand of online taxi-hailing, the lack of good internet infrastructure as well as standardized addressing systems, service delivery to customers is a challenge for companies, therefore the need for appropriate geocoding tools in poorly mapped areas. In fact, even if an online mapping system is widely used, many developing countries, such as Ethiopia, suffer from good addressing and geocoding system, as well as internet connectivity.

To increase platform accessibility and market reach, all the taxi-hailing services operate through a call center where customers request for taxis by dialing a hotline number. Their call are processed by manually typing the pick up and drop-off locations in to the platform's online system. The problem arises as the requested pickup and drop of locations could be full of inconsistent, misspelled and ambiguous. This could lead to an erroneous data input which in return could lead to an inaccurate geographical representation of the

pickup address. An domain specific name resolution and duplicate matching process is then needed to increase the geocoding accuracy of the manually inputted addresses.

Although some people use the provided platform's application, yet still prefer to directly call to the matched driver to verbally explain their exact pickup location. This could in turn lead to a longer waiting time for passengers. Devising an accurate geographical representation of the addresses along with a standardized addressing system in the taxi hailing transportation domain is crucial.

Due to the essential life services they provide through their day-to-day interaction with customers, to quickly dispatch taxis to the customer's request location, these service providers need to have a good knowledge of the field they operate in. To deal with challenges, these companies have to create their geolocation systems that, over the years, become an important source of geographical and customers' addresses data.

Hence, we investigate addresses data for 3 years collected by Ethiopian taxi-hailing company, ETTA, through customer's calls. ETTA, founded in 2016, is the first taxi hailing platform in Ethiopia's capital, Addis Ababa. It operates via a taxi hotline available 24 hours allowing customers ways to access to the closest taxi. On any pickup request, verbal description of the pickup location is given by customer to call center officers along with destination location and user information, such as, customer name and phone number. The collected data is characterized by its vagueness and full of misspelling or unknown places from common mapping facilities. To address these challenges, we first need to produce an address standard by text mining historical data and creating a reference dataset before being able to achieve the matching operations.

1.3 Research Questions

In a transportation context, geocoding is a task that is often a fundamental step preformed prior to all operations that take place. As such, the quality of geocoding systems used within taxi-hailing service providers is of paramount concern to the company that provides such service platforms, end-users who benefit from using such platforms, and researchers who wish to use data produce through such services.

In developing countries, such as Ethiopia, it is challenging to perform this task, as human factors, misspelling and abbreviations of addresses, and the lack of standard address format along with quality internet connection, produce ambiguous and incomplete data leading to an imprecise and inaccurate geographical representation of an intended address. So far nothing has been done concerning the challenges mentioned above in Ethiopia, making this research the first. An automated solution to this problem reduces manual effort significantly from manual reading of taxi pickup addresses to automatic geocoding.

The main research problem in this thesis can be described as the following:

Given a textual address for taxi pickup in a city, geocode the address to the corresponding geographical location.

The main research problem can be split into three sub-research problems:

- How to create a standardize domain specific addressing format clean of any possible errors?
- How to match a given erroneous pickup/drop-off address to its corresponding correct addressing format?
- How to accurately match any given pickup/drop-off address location to its corresponding geographical representation?

In countries where there exists a standard address structure in terms of definition and the way people follow them, the solution to the defined problem can be achieved by using online geocoding services such as Google Geocoding API. When considering countries where such standard structure does not exist or people do not follow such structures due to cultural diversity and literacy level, geocoding with high accuracy becomes quite challenging.

The main guiding research question for this research thesis is:

- What machine learning techniques will increase the geocoding accuracy for addresses in transportation domain in Ethiopia?

The following specific objectives are involved:

- To produce an address standard by text mining historical data
- Creating domain dependent reference dataset
- Spelling Correction
- Data dimensionality reduction
- Novel Enhanced Geocoding
 - Querying online geocoding services using preprocessed address data
 - Machine Learning Classification Methods with Labeled Dataset

The geocoding algorithm in the main research makes use of the findings in the first two sub-research questions. All implementations are done in Python programming language, together with cross-platform geocoding services such as Google Geocoding API.

There is an important assumption, restricting the problem setting :

- We cannot get the feedback if the targeted location is among one of the user presented

locations, which makes it impossible to develop a learning matching algorithm.

- We use a manually crafted domain specific reference dictionary from the dataset itself. This dictionary is composed of correctly spelled single words within the whole dataset.

1.4 Research Methodology

The goal of this research is to accurately geocode addresses to their corresponding latitude/longitude coordinate pairs. This is a quantitative study which made use of a novel machine learning approaches to enhance and increase geocoding quality and accuracy. The accuracy of the geocoding will be measured by the match rate, the percentage of records that produce a reliable match. Results from [35] show that a match rate of 85% was necessary. Studies [36,37,38] also show that match rates are much lower in developing countries that lack standards in their addressing systems. In order to enhance the match rates, a spell checking algorithm will be used for each address at the input preprocessing stage.

In this research, I will investigate addresses data for 3 years collected by Ethiopian taxi-hailing company, ETTA. The data was collected through customer's calls to a taxi hotline. On any call, verbal description of the pickup location is given by customer to call center officers along with destination location and other customer information. The dataset, after applying the necessary preprocessing and data cleaning steps, will be used to generate word level domain dictionary. The dictionary will be used to create an addressing format as well as the matching of erroneous pickup/drop-off address to its corresponding correct addressing format. We will use the dataset to compare the performance of different **similarity measures** as well as **spell checking algorithms** selected from literature. Similarity measures measure the degree of similarity between two string while the spell checking algorithms generate candidate string for a given error word of a given address. The measures having the best performance will then be used for matching address locations. We then use the best performing similarity measures and spell-checking algorithms to match and address and transform it into a standardized addressing format free if errors. Using the standardized addresses we compare the performance of different **string clustering** and **duplicate matching algorithms**. We will use the best suggested clustering and duplicate matching algorithm to reduce the dimensionality of the dataset. Finally, we will perform the geocoding task using the Google Geocoding API and the reduced dataset as an input. The performance of the algorithm is compared with the baseline process of conventional geocoding of the addresses using the Google Geocoding API. For both algorithms, the percentage of correctly returned targeted locations is determined. The accuracy of the geocoding result will be manually determined.

1.5 Thesis Outline

In Chapter 2 we do a general literature study, after which we focus on the properties of similarity measures, spell checking algorithms, string clustering and duplicate matching algorithms proposed in the literature, in Chapter 3. In Chapter 4, the research methodology is defined and the best similarity measures and spell checking algorithms to match addresses as well as string clustering algorithms are determined, by testing on dataset used in reality, consisting of taxi pickup addresses in Addis Ababa, Ethiopia. Also in Chapter 4, an algorithm is proposed to generate word level domain dictionary as well as the system overview will be discussed. In Chapter 5, the best similarity measures and the spell checking algorithm along side the string clustering algorithm are used in a “matching algorithm”, developed to match input address with the targeted address in the database. If the address is not found in the database, the geocoding task is triggered to find the accurate geographical representation for the input address which will then be saved in the domain specific database of targeted addresses. A preprocessing method is given, which considerably speeds up the search. Also, a comparison is made with the baseline conventional process of geocoding using Google Geocoding API, as there are no previous standards set using this specific dataset in this domain. In Chapter 6 the conclusions are given, and possible future work is discussed.

Chapter 2: Related Work

The literature review for this thesis consists of two parts. The first part gives an overview of the relevant literature. The second part consists of an overview of all similarity and distance measures, spell checking, string clustering and duplicate matching algorithms available in literature.

This chapter describes the similarity measures, spell checking algorithms and string clustering and duplicate matching algorithms available in literature. In Chapter 3, for each input address in the proposed geocoding system several similarity measures, spell checking algorithms as well as string clustering and duplicate matching algorithms are selected. After implementation and testing of these measures, the best measure(s) from each are determined for the proposed geocoding system. Measurements are rated based on the combination of error correction and the matching quality for the spell checking algorithms and the similarity and string clustering measures respectively.

The measures presented in this chapter are categorized in character-based, token-based and phonetics measures. The important properties in each category are discussed. The spell correction [26] has been widely used in search engines, works as a "gate keeper" for query parsing. The major components of a spell checker are the error detector, the candidate generator and the spell corrector. The spell checking approaches are categorized by correction context and correction methods. Important properties in each category are discussed.

The terms distance measure and similarity measure are used without distinction in this thesis. However, a similarity measure can be seen as the inverse of a distance measure. The smaller the distance between strings, measured by a distance measure, the better the agreement is. A similarity measure is a number between zero and one, with one indicating perfect agreement of the strings and zero no agreement. It is often trivial to transform a distance measure into the corresponding similarity measure, because distance measures are typically bounded due to finite lengths of the strings.

2.1 Geocoding

A general review of the concept of geocoding has been given Goldberg[1], including historic development and future challenges. Goldberg defines geocoding as the act of turning descriptive locational input data, such as (postal) address or named place, into an absolute geographic reference code determined by a processing algorithm. The key steps

the processing algorithm involves include :

- Standardization and normalization of input into a format and syntax compatible with the reference dataset.
- Matching algorithm that picks best feature in the reference dataset ([2],[3],[4],[5]). This mainly involves the task to search the reference set for a match and output the geographical coordinates.
- Code generation mechanism that returns the geographical coordinates ([2],[4],[8],[9],[10]).

A key problem in geocoding, however, is the requested location description may differ from the description of location in the reference dataset due to misspelling, alternative spelling, abbreviations, different word order and addition or omission of words. Goldberg points out the important difficulties in geocoding: how to deal with erroneous or inconsistent input data and inaccuracy in the reference dataset. Goldberg emphasizes the need to massage the input data and transform it into a format consistent for finding best match. This approach promises a higher match rate at the price of accuracy ([10], [11], [12]).

Churches [7] states geocoding as a special case of record linkage. Record linkage refers to the process of joining records or entities that relate to the same real-world entity or event in one or more data collections. Churches and other researches ([13], [14], [15]) have used probabilistic approaches for standardization. Churches [7] proposed lexicon-based tokenization and probabilistic hidden Markov models to carry out the standardization pre-processing of names and addresses. The results from this research show that the hidden Markov models was found to be quick and produced equal or better standardization accuracy the widely-used rule-based deterministic approaches ([10],[11],[12]). Except for [Buckley[16]] takes a different approach based on fuzzy set theory and clustering of the database, an application of address matching in a multiple field record matching setting is not available. However, the risk that these results may not be accurate, as they are relying on the confidence level of their uncertainty measures, and they will in some cases produce erroneous results should be pointed out.

M. Kebe et al. [6] proposes a multi-agent system that enhance geocoding system in poorly mapped areas. Many developing countries compare to industrialized countries lack standard in their address systems, making the addressing system ambiguous, imprecise or incomplete. They constructed a knowledge base consisting of a base ontology by creating non-official address standard using data collected by an energy utility company, which then was used in a multi-agent system, where agents are assigned different tasks of the geocoding process with a global goal of finding the best possible match or approximation of a location based on the knowledge. The results of the proposed approach were then compared with that of Google geocoding API to verify its usefulness. Although it showed a great potential, however, it also shows issues considering local context semantics.

The standardization and normalization of input address into a format and syntax compatible with the reference dataset introduces two major study areas, spelling correction and string matching.

2.2 Spell-Checking Algorithms

Spell checking is a well known task in computational linguistics, dating back to the 1960s, most notably to the work of Damerau (1964). Spell checking is the process of detecting and sometimes providing suggestions for incorrectly spelled words in a text [41]. It has become a very important application to search engines. Companies like Google or Yahoo! Use log files of all user's queries to map the relation between misspellings and the intended spelling reaching very high accuracy. In the context of Geocoding, it is a crucial process that is applied in matching location records with the corresponding geographical points. The language of queries, however, is typically shorter than naturally occurring text, making this application of spell checking very specific.

Spelling errors can be brought down into several types [46]: non-word errors which are error words that are non-words, that is, words that cannot be found in a dictionary; and real-word errors which are error words that are valid words in the dictionary but invalid with respect to their context. For instance, "aple" is a non-word error, while "from" in "fill out the from" is a real-word error. Additionally, three different types of non-word errors exist and they are [46]: mistyping, which results from manual error related to the keyboard or typewriter, e.g. "necessary" mistyped as "mecessary"; cognitive error, which results when the writer does not know how to spell the word correctly, e.g. "necessary" mistyped as "nessecary"; and phonetic error, which results from the phonetic substitution of sequence of characters with another incorrect sequence, e.g. "parachute" mistyped as "parashoote".

Several spell-checking techniques and algorithms have been conceived, since the early days when text began to be manipulated by computers, with an improve in effectiveness and performance. Fundamentally, a spell-checker is made out of three components: An error detector that detects misspelled words, a candidate spellings generator that provides spelling suggestions for the detected misspelled word, and an error corrector that chooses the best correction out of the list of candidate spellings. All spelling checker tools uses a dictionary as a database. Every word from the written text is looked up in the dictionary. When a word is not found in the dictionary, it is detected as an error. In order to correct the error, a spell checker searches the dictionary for words that resemble the erroneous word most. These words are then suggested to the user who chooses the best word that was expected.

First, spell checking algorithms need to identify possible misspellings that a user may commit. Misspellings can be related to the writer's poor writing and spelling competence, to learning disabilities such as dyslexia, and also to simple typographic and performance

errors. These phenomena generate a wide range of different spelling possibilities that a spell checker should be trained to recognize.

The second function of spell checkers is to suggest the users' intended spelling of a misspelled word or at least to suggest a list of candidates in which the target word appears and to replace the incorrect with most likely corrected word. This is often done by calculating the distance between the misspelled word and a set of potential candidates. This is by no means trivial and several methods have been proposed to address this task.

2.2.1 Error Detection

The error detection process usually consists of checking to see if an input string is a valid index or dictionary word. Efficient techniques have been devised for detecting such types of error. The two most known are n-gram analysis and dictionary look-up.

N-gram Analysis

N-gram analysis is described as a method to find incorrectly spelled words in text and used for non-word errors. Instead of comparing each entire word in a text to a dictionary, just n-grams are controlled. A check is done by using an n-dimensional matrix where real n-gram frequencies are stored. The n-gram model is a probabilistic model [20,21,23,24] used for predicting the next item in a sequence of items. The items can be any linguistic entity such as, letters, words, phrases. Predominantly, the n-gram model is a word-based model used for predicting the next word in a particular sequence of words.

If a non-existent or rare n-gram is found the word is flagged as a misspelling, otherwise not. An n-gram is a set of consecutive characters taken from a string with a length of whatever n is set to. If n is set to one then the term used is a uni-gram, if n is two then the term is a bi-gram, if n is three then the term is tri-gram. N-gram tables can take on a variety of forms but the simplest is bi-gram array which is 2-D array of size 41*41 whose element represents all possible 2-D letter combination of alphabet. The n-grams algorithms have the major advantage that they require no knowledge of the language that it is used with and so it is often called language independent or a neutral string matching algorithm. Using n-grams to calculate for example the similarity between two strings is achieved by discovering the number of unique n-grams that they share and then calculating a similarity coefficient, which is the number of the n-grams in common (intersection), divided by the total number of n-grams in the two words (union).

Dictionary Look-up

This technique simply look-up every word in the dictionary if the word is not there then it is said to be an error. Dictionaries have their own characteristics and storage requirements. It is a straightforward task. Large dictionary might be a dictionary with most common word

combined with a set of additional dictionaries for specific topics such as computer science or economy. Big dictionary also uses more space and may take longer time to search. The non-word errors can be detected as mentioned above by checking each word against a dictionary. The drawbacks of this method are difficulties in keeping such a dictionary up to date. At the same time one should keep down system response time. Too small a dictionary can give the user too many false rejections of valid words. The most common technique used for gaining fast access in dictionary is Hash tables. In order to look-up a string, one has to compute its hash address and retrieve the word stored at that address in the pre constructed hash table. If the word stored at the hash address is different from the Input string, a misspelling is flagged. Hash tables main advantage is their random-access nature that eliminated the large number of comparisons needed to search the dictionary. The main disadvantage is the need to devise a clever hash function that avoids collisions. To store a word in the dictionary we calculate each hash function for the word and set the vector entries corresponding to the calculated values to true. To find out if a word belongs to the dictionary, you calculate the hash values for that word and look in the vector. If all entries corresponding to the values are true, then the word belongs to the dictionary, otherwise it does not.

2.2.2 Candidate Generation and Error Correction

The error correction process can use the similarity measure algorithms to measure the similarity between the wrongly spelled word and the words which are extracted by the first routine. It can use one of the similarity measure algorithms like edit distance, n-gram, dice coefficient etc.

Edit Distance

The minimum edit distance is another widely used approach proposed by [27]. It is defined as the minimum number of edit operations, insertion, deletion, transpose, and substitution, needed to transform a string of characters into another one. In spell-checking, the goal of Edit Distance is to eliminate candidate spellings that have the largest edit distance with respect to the error word. Candidates with larger distances are regarded as sharing fewer letters with the error word than other candidates.

The most known edit distance algorithms are, Levenshtein [28], uses a weighting approach that assigns a cost of 1 for each edit operation, Hamming [31], used to measure the distance between strings with the same length, and Lowest Common Subsequence[32], an idea that pivots around finding the maximum length of common subsequence, a series of characters from left to right that are not necessarily consecutive.

N-grams

N-grams can be used in two ways, either without a dictionary or together with a dictionary. Letter N-gram including tri-gram, bi-gram and uni-gram have been used in variety of ways in text recognition and spelling correction techniques. Used without a dictionary, n-grams are employed to find in which position in the misspelled word the error occurs. If there is a unique way to change the misspelled word so that it contains only valid n-grams, this is taken as the correction. The performance of this method is limited. It is that it is simple and does not require any dictionary. Together with a dictionary, n-grams are used to define the distance between words, but the words are always checked against the dictionary.

Similarity Keys

In this technique we map every string into a key such that the similarly spelled strings will have similar key. It is known as SOUNDIX system. In this it is not necessary to directly compare misspelled string to each word in dictionary. The Soundex algorithm [19], used for indexing words based on their phonetic sound, converts any string of words into a code where similarly coded words can be matched regardless of trivial differences in their spelling.

Rule Based Technique

It attempts to represent knowledge commonly spelled errors i.e. mistyping by mistake in the form of rules for converting it into a valid word. Each word which is correct can be taken as a suggestion. It consists of the process consisting of applying all applicable rules to a misspelled string.

Neural Network

This method works on small dictionaries. Back propagation algorithm is used in neural network. It consists of 3 layers input, hidden and output layer. It has potential to adapt specific error pattern. In this input information is represented by on-off pattern. $A=1$ indicates that node is turned on and $A=0$ means node is turned off. For e.g. in spell checking applications a misspelling represented as binary n-gram vector may be taken as input and output pattern might be vector of m elements means number of words in lexicon.

Neural nets can, according to Kukich [46], be used for spelling correction, since they have an inherent ability to retrieve relevant information despite incomplete or noisy input, or in this case misspellings. They can also be trained to adapt to users' error patterns and gradually improve and eventually maximize correction accuracy.

However, they have a major drawback. Kukich shows in [47] that running the learning cycles needed to obtain acceptable correction accuracy requires a very long time even for a small dictionary of about 500 words. Training time increases nonpolynomial with dictionary size.

Probabilistic Technique

Probabilistic algorithms use probabilities to determine suitable corrections for the misspelled word in spell correction and text recognition. Kukich [46] describes two kinds of probabilities used in spelling correction, transition probabilities and confusion probabilities. Transition probabilities represent probabilities that a letter x will be followed by another letter y . These probabilities are language dependent and can be estimated using n -gram frequency statistics from a relevant data source. Confusion probabilities represent probabilities that a letter x will be confused and substituted with another letter y . These probabilities are source dependent.

Variations

Salton [48] states that bi-grams and trigrams ($n=2,3$) are best suited for retrieving similar words to a given word. The use of unigrams ($n=1$) would produce too many matches, while longer string sequences ($n \geq 4$) would miss the common roots of many short words. Pfeifer et al. [49] proposed the use of additional blanks at the beginning and the end of the word to emphasize the first and last letters as well as increase the total number of n -grams involved. This could give a more precise result, especially for shorter words. In general, the number of n -grams in a word of length l , with k blanks added to both the beginning and the end, is $l + 2k - n + 1$. Matching speed can be improved by indexing dictionary entries by the n -grams they contain, to allow for fast dictionary look-up, and limiting the matching process to words which have at least one n -gram in common with the misspelled word.

Youssef Bassil & Mohammad Alwani [41] propose a parallel spell-checking algorithm that uses rich real-world word statistics from Yahoo! N-gram dataset to correct non-word and real-word spelling errors in computer text. The algorithm is divided into three sub-algorithms: Error Detection that detects misspellings, Candidates Generation that generates correction suggestions, and Error Correction that performs the contextual error correction. The proposed algorithm outclassed other spell-checking algorithms by correcting 94% of the total errors, distributed 99% of non-word errors and 65% of real-word errors. The use of N-gram word statistics as a dictionary model in the contextual error correcting process, which delivers wide ranging set of words and n -grams that cover domain specific terms, is the major reason behind the outstanding results.

2.3 Similarity Measures and Approximate String Matching

Approximate string matching is another study area that shares useful information. Navarro [22], most extensive and well-known approximate string matching survey, that broadly explains the history and application domains. Techniques of existing algorithms, such as dynamic programming, automata, bit-parallelism and filtering, are compared. It shows the focus in approximate string matching is to find the identical string in a text instead of finding identical strings in a database with records. There are two types of approximate

matching techniques : phonetic matching and pattern matching. Phonetics matching considers the similarity of letters in a string as to the usual sound they produce in English, whereas, pattern matching relies on comparing characters from each string, to detect points in common and to quantify their similarities based on that. Address matching and/or name matching, in which identical address or person needs to be found in a database, are related research areas.

Davis Jr.[42] proposes a method applicable to both address matching and name matching with an assumption of that both the input string and all strings in the database were of the same type unlike the different fields of input string and all strings in the data base in a record linkage setting. Their approach is capable of dealing with non-standard abbreviations, sub-string swaps, stop words and omissions of strings. A similarity measure was used, after a preprocessing step to standardize the data and limit the space, to determine the final similarity between the input entity and the entity in the database depending on if the the input string consist of one or multiple words. The result from this approach shows that more tests with larger volume of data are required, in order to adequately asses the computational efficiency of the proposed method.

2.3.1 Character Based Similarity Measures

In this section we propose the character-based similarity measures. Three categories of similarity measures are discussed in their own subsections. The first category consists of measures that are related to the edit distance, the second category consists of measures of the Jaro-Winkler type and in the last category the n-gram measures are given.

2.3.1.1 Edit Distance

Edit Distance or Levenshtein Distance

This is a well-known and widely used distance metric. The basic approach of the algorithms that belong to this method is to look at how many character changes (number of character insertions, deletions, substitutions or transpositions) are required to get from one string to another. It is defined as the smallest number of edit operations required to turn s_1 (String 1) into s_2 (String 2). In other words, edit distance is a way for quantifying how dissimilar two strings (e.g. words) are to one another. It is widely used in different application in the natural language processing, where automatic spell corrections can determine candidate correction for a word that is misspelled by finding words in the dictionary that have a low distance to the misspelled word. Each edit operation, in Levenshtein definition, has a unit cost (except a substitution of a character by itself has a zero cost), so the Levenshtein distance is equal to the minimum number of operations needed to transform one string to another. This could be mathematically formulated as $lev(|s_1|, |s_2|)$, with

$$lev_{s1,s2}(i,j) = \begin{cases} \max(i,j) & \text{if } \min(i,j) = 0, \\ \min \begin{cases} lev_{s1,s2}(i-1,j) + 1 \\ lev_{s1,s2}(i,j-1) + 1 \\ lev_{s1,s2}(i-1,j-1) + [s1_i \neq s2_j] \end{cases} & \text{otherwise} \end{cases}, \quad (1)$$

where the first element in the minimum corresponds to a deletion, the second to an insertion and the last to a substitution. $[s1 \neq s2]$ in the last element is an indicator function equal to one if the character at position i in $s1$ is not the same to the character at position j in $s2$, and zero otherwise. $\max(|s1|, |s2|)$ is the maximum edit distance whereas $\min(|s1|, |s2|)$ is the minimum edit distance.

To turn the Levenshtein distance metric into a similarity measure the distance should be divided by the upper bound on Levenshtein distance, which is the length of the longer string:

$$simLev(s1,s2) = \frac{lev_{s1,s2}}{m} \quad (2)$$

A useful property is that the Levenshtein distance is a true metric, because it satisfies the triangle inequality, which states that for a given triangle the sum of the length of any two sides must be greater or equal to the remaining side of the triangle. Also, the triangle inequality can be used to filter when determining the best match(es) out of a dictionary of strings, by selecting strings as “pivots”. A pivot is a string for which the edit distance between other strings and the pivot is determined beforehand, such that a lower bound on the edit distance between input query and the other strings can quickly be calculated. For a detailed explanation of the triangle inequality and it is used in filtering when applying the Levenshtein metric, see [33].

The Levenshtein distance soon gets too slow to use in practice, if the strings compared are larger. Several techniques have been introduced to reduce the calculation time when one input string needs to be compared with all strings in a dictionary [34].

Damerau-Levenshtein Distance

The Damerau-Levenshtein distance metric is an extension of the Levenshtein distance metric, as it also allows transpositions or substitutions of adjacent characters at a unit cost. Similar to the basic edit distance, Damerau-Levenshtein satisfies the triangle inequality. Variations of this algorithm assign different weights to edits based on the type of operation, phonetic similarities between the sounds typically represented by the relevant characters, and other considerations.

Hamming Distance

The Hamming distance measures in linear time with respect to the size of the strings, the number of different characters at the same positions in two equally sized string. This can be seen as the number of substitutions needed to turn one string into one another or the minimum number of errors that could have transformed one string into the other.

Bag Distance

The bag distance algorithm enumerates every character in s_1 that cannot be uniquely matched with a character in s_2 , and vice versa; the maximum of the two values is the bag distance. It is defined as the multiset of characters in a string. Let x be the bag of s_1 and y the bag of s_2 . Then the bag distance is defined as:

$$BagDist(s_1, s_2) = \max(|x-y|, |y-x|) , \quad (3)$$

where $|a-b|$ is the number of characters in a that is not in b , taking into account letters occurring more than once.

Bag distance places an upper bound on Levenshtein distance and has been proposed as a fast approximation thereof [39]. The bag distance is often used as a cheap approximation for the edit distance, or as a filtering step before applying different or more sophisticated similarity measures.

Longest Common Subsequence

The longest common subsequence of two strings is the maximum number of common, not=necessarily adjacent, characters that are in the same order in both strings. It is the problem of finding the longest subsequence common to all sequences in a set of sequences.

For example, consider the sequences (ABCD) and (ACBAD). They have 5 length-2 common sub-sequences: (AB), (AC), (AD), (BD), and (CD); 2 length-3 common sub-sequences: (ABD) and (ACD); and no longer common sub-sequences. So (ABD) and (ACD) are their longest common sub-sequences.

A similarity metric can be calculated in three ways: by dividing the longest common subsequence by the length of the shorter string, the length of the longer string or the average string length.

Longest Common Substring

The longest common substring is defined in a similar way as the longest common subsequence, but the measure calculates the number of common characters that are adjacent and in the same order in both strings. It is used to find the longest string(s) that is a substring of two or more strings.

For example, the longest common substring of the strings "ABABC", "BABCA" and

"ABCBA" is string "ABC" of length 3. Other common substrings are "A", "AB", "B", "BA", "BC" and "C".

Combination Edit Distance

The combination edit distance is a simple combination of the edit distance metric and longest common subsequence measure. It is mentioned here, because in a string similarity measure comparison study on matching names, it increased the matching quality opposed to the edit distance and longest common subsequence measures [40]. The combination edit distance is calculated as follows:

$$\text{sim}(s1, s2) = a * (1 - \frac{e}{n}) + (1 - a) * \frac{l}{m} \quad (4)$$

Where e is the edit distance between $s1$ and $s2$, l the longest common subsequence, and a a parameter between zero and one to set the weights for the edit distance metric and longest common subsequence measure.

2.3.1.2 Jaro-Winkler

Jaro

The Jaro distance (and the extension Jaro-Winkler, see below) is mostly applied on matching personal names. In general, it has a good performance on matching relatively short strings. It is not a true distance metric, because it does not obey the triangle inequality. In this measure, strings are more similar if they have matching characters within a specific range, depending on the length of the longer string. A penalty is given if matching characters are transposed (i.e., if they are not in the same order).

The Jaro distance measure can be calculated as follows:

$$\text{Jaro}(s1, s2) = \begin{cases} 0 & \text{if } c = 0 \\ 1/3 * (\frac{c}{|s1|} + \frac{c}{|s2|} + \frac{c-t}{c}) & \text{otherwise,} \end{cases} \quad (5)$$

where c is the number of matching characters and t the number of matching characters that is transposed. Two characters are considered matching if they are equal and positioned within the range $[\frac{\max(|s1|, |s2|)}{2} - 1, \frac{\max(|s1|, |s2|)}{2} + 1]$ of each other. The Jaro distance is actually an inverse distance measure, as its value increases with string similarity. Each character of $s1$ is

compared with all its matching characters in s2. The number of matching (but different sequence order) characters divided by 2 defines the number of transpositions.

Jaro-Winkler

Jaro-Winkler distance is an often used extension of the Jaro distance. Jaro-Winkler is equal to the value of the Jaro distance, plus a score if the prefixes of the strings are equal. Here, it is assumed that errors tend to occur less often at the start of the strings[]. The distance measure is calculated as follows:

$$Jaro-Winkler(s1,s2) = Jaro(s1,s2) + (l * \beta * (1 - Jaro(s1,s2))) , \quad (6)$$

Where l is the number of equal characters at the beginnings of the strings with a maximum often set to four characters, β is a constant, usually 0.1, scaling factor for how much the score is adjusted upwards for having common prefixes.

Although often referred to as a distance metric, the Jaro-Winkler is not a metric in the mathematical sense of that term because it does not obey the triangle inequality.

Jaro-Winkler with Longer String Adjustment

This extension of Jaro-Winkler is applicable to longer string (i.e, in this case at least five characters) that do meet the following criteria:

- $n \geq 5$
- $c - l \geq 2$
- $c - l \geq \frac{n-l}{2} ,$

where l the length of the common prefix and c the number of matching characters.

The idea is that longer strings need to have a higher similarity score if characters match, also when they are not in the prefix. The intuition is that the contribution of a common prefix for the similarity value decreases when the sizes of the compared string get larger. The first criterion above assures the strings are long enough and the second that enough characters match besides the characters in the common prefix. The last criterion checks if the number of matching characters that is not in the common prefix is large enough, relative to the size of the shorter string.

The similarity score similar to the regular Jaro-Winkler can be defined as:

$$Jaro - WinklerLSA(s1,s2) = Jaro(s1,s2) + (1 - Jaro(s1,s2)) * \frac{c-(l+1)}{n+m-2*(l-1)} \quad (7)$$

Sorted Jaro-Winkler and Permuted Jaro-Winkler

The Sorted Jaro-Winkler is a similarity measure that works similarly as the original Jaro-Winkler except here the strings $s1$ and $s2$ are first sorted in alphabetical order if they

consist more than one word. This measure will solve the problem of an input string with swapped words.

On the other hand the permuted Jaro-Winkler, similar to the sorted Jaro-Winkler, measures the basic Jaro-Winkler distance for all possible permutations of the words in the strings returning the highest score. This measure was developed to solve the problem of characters in a string not being in the right order[43].

2.3.1.3 N-grams

N-grams: Dice's Coefficient, Jaccard Similarity and Overlap Coefficient

N-grams, also called q-grams, are substrings of length n . A similarity measure is developed by creating the multisets of n -grams from s_1 and s_2 (considering the n -grams occurring more than once). The number of n -grams both strings have in common is then divided by the average number of n -grams in both strings (Dice's Coefficient), the number of n -grams in the longer string (Jaccard Similarity), or the number of n -grams in the shorter string (Overlap Coefficient).

Another application of n -grams that is frequently used in practice is to filter candidate strings by not exploring strings that have less than a threshold value T common n -grams with the input string. A similarity measure, may also be a measure that doesn't apply n -grams, is then applied to determine the best matching string. If a trigram that is in both string but occurs more in string s_1 than in string s_2 , the number of times it occurs in string s_2 is counted and vice versa.

Skip-grams

N-gram-like measures can also be calculated with skip-grams or 'open bigrams'[44]. The skip gram set of a string is the set of 2-grams plus all the combinations of two characters while skipping up to a chosen amount of character(s) in between. To illustrate, *cow* and *caw* have in common the open bigram *c_w*. Skip-grams are developed to deal with simple typographical errors such as insertion, a transposition, a deletion or a substitution.

Padded N-grams

Padded n -grams are substrings of length n after attaching $(n-1)$ spaces at the start and end of the string. This has empirically proven to give better results for matching and indexing than "normal" n -grams[44]. The idea of padding is to make it more robust against errors. A small error (e.g. insertion or deletion of a character) in a string of three characters, may lead to zero shared trigrams. By applying padded n -grams, there would still be some trigrams shared, that will lead to a useful distinction between targeted string and string that do not have any trigrams in common with the input string.

Positional N-grams

Positional n-grams include the positional information of the n-grams in the strings. Similar n-grams must be within a maximum distance to each other to count as a match. A disadvantage to this measure is that the measure may not give the desired results when having a substring addition or omission or, especially, when words are swapped.

As explained when discussing the padded n-grams, the basic n-gram measure will not perform well if the strings are very short and a typographical error occurs.

2.3.2 Token Based Similarity Measures

In this section, we discuss two different measures based on tokenizing the strings.

2.3.2.1 Term Frequency - Inverse Document Frequency (TF-IDF)

Tf-idf

TF-IDF (term frequency-inverse document frequency) is a statistical measure that evaluates how relevant a word is to a document in a collection of documents. This is done by multiplying two metrics: how many times a word appears in a document, and the inverse document frequency of the word across a set of documents. Tf-idf uses on one side that more frequently occurring parts of the string are important and on the other side that less frequently occurring parts give more specific information about the string. The measure is mainly used to match a longer string query with patterns in a text. The use of the Tf-idf is to determine the similarity values between one input string and many strings in a list.

The Tf-idf makes use of cosine similarity, which is the cosine of the angles between the vectors obtained by projecting the strings in a high dimensional space. The closer the vectors are, the higher the similarity.

For each token of a string, for example a word or a n-gram, the frequency of the token in the string is the term frequency. The inverse document frequency calculated as the number of times the token appears in all strings in the database is used as a factor to give lower weight to more common tokens.

The similarity between two strings can be calculated as:

$$sim(s1, s2) = \frac{\sum_{j=1}^{|T|} v_{s1}(j) * v_{s2}(j)}{\|v_{s1}\|_2 * \|v_{s2}\|_2}, \quad (8)$$

where $|T|$ is the number of tokens in the input string $s1$ and

$$v_s(w) = \log(tf_w + 1) * \log(idf_w) \quad (9)$$

where tf_w is the frequency of the token in the string and idf_w is the ratio of the number of strings in the database containing token w given the number of total records in the database. $\|v_s\|_2$ is the Euclidean norm of the vector of all tokens in the string.

The intuition is that the similarity score increases when the strings have tokens in common. If a token that the strings have in common does not occur much in the document (e.g. database), then it will have a higher contribution to the similarity score.

Soft Tf-idf

In the soft tf-idf, in case words are chosen as tokens for the tf-idf similarity measure, a similarity measure different than the tf-idf measure is first used as a filter since it is possible that no corresponding words can be found in the database if there is an error in the input string. If the similarity value between a string in the database and the input string is large enough, the string is compared with the input string. This final comparison is done by using the similarity values resulting from both the tf-idf measure and the other similarity measure.

2.3.2.2 Monge and Elkan's Recursive Matching Scheme

Monge and Elkan's Recursive Matching Scheme

This measure is proposed by Monge and Elkan[45] to calculate the similarity of two string consisting of several words. The degree of similarity of each word in the input string with all the words of the string in the database is calculated using a similarity measure. The maximum of all the measures is taken for every word of the input string, after which the final degree of similarity is obtained by taking the average overall maximums. This is formulated as:

$$MongeElkan(s1, s2) = \frac{1}{|A|} \sum_{i=1}^{|A|} \max \{sim(a_i, b_j)\}_{j=1}^{|B|}, \quad (10)$$

where $|A|$ is the number of words in $s1$, $|B|$ the number of words in $s2$, a_i word i in $s1$, b_j word j in $s2$ and sim is a similarity measure (Levenshtein, Jaro-Winkler, etc.).

The measure does not take into account the difference in number of words of both strings, which may produce some unexpected results.

2.3.3 Phonetic

Soundex

Initially designed for English language, Soundex is a phonetic similarity measure that makes use of the sound of the letters when pronounced. It is the most common phonetic

coding scheme. Letters that are sounding similar are assigned to the same group indicated with a number. The input string and the strings in the database are transformed into a code, consisting of the first letter of the string and three number corresponding to the groups to which the other letters in the string belong.

The Soundex measure is frequently used in filtering step. The string is kept as a candidate if the code corresponding to a string in the database has more characters in common with the code of the input string for a predefined threshold value.

Editex

Editex is a combination of soundex and edit distance based methods. It straightforwardly combines Levenshtein distance with ‘letter groups’ (*aeiouy, bp, ckq*, etc.) such that letters in a similar group frequently correspond to similar phonemes. As in Levenshtein distance, the minimal number of insertions, deletions, and replacements necessary to transform one string to another is computed, but edits that replace a letter with another letter from a different group are weighted more heavily, and deletions of letters that are frequently silent (*h* and *w*) are weighted less heavily than other deletions. The edit distance costs are 0 if the letters are equal, 1 if the letters belong to the same group and 2 otherwise.

In this research thesis, the Soundex similarity measure will be used in generating candidates for the spell checking algorithm to be implemented.

Below in Table 2-1, the summary of each similarity measure, as well as comments about the expectation of the matching quality, is given.

Table 2-1: Similarity measures

Similarity Measure	Expected Matching Quality
Edit Distance or Levenshtein Distance	Mainly suited for correcting typographical errors.
Damerau-Levenshtein	Mainly suited for correcting typographical errors.
Hamming Distance	Only applicable if s1 and s2 have the same size.
Bag Distance	It returns a lower bound on the edit distance and is often used as filtering step, not as a measure itself.
Longest Common Subsequence	Can deal with several error types, depending on how the similarity measure is calculated. Not applicable on input strings with swapped words.
Longest Common Sub-string	Can deal with several error types, depending on how the similarity measure

	is calculated.
Combination Edit Distance	.Is not applicable on a string with swapped words. Matching quality on other error types depends on parameter setting and how the longest common subsequence similarity value is calculated.
Affine Gap Distance	Mainly suited for correcting abbreviations and typographical errors
Smith-Waterman Distance	Extension of affine gap distance. In comparison with affine gap distance, it can better correct for errors made due to substituted letters that have the same sound when pronounced.
Jaro	Developed to correct for short personal names with errors. Not applicable if similar characters are not around the same position (may occur if there is an addition or omission and when words are swapped).
Jaro-Winkler	Same as Jaro
Jaro-Winkler with longer string adjustment	Same as Jaro
Sorted Jaro-Winkler	Same as Jaro, but it is developed to correct for string with swapped words.
Permuted Jaro-Winkler	Same as Sorted Jaro-Winkler, but it may be better in correcting if an addition or omission occurs.
N-grams	No obvious weakness, except for typographical errors in relatively short strings. Quality for correcting a specific error depends on how similarity is calculated.
Padded n-grams	Same as n-grams, but assigns more value to beginning and end of string
Positional N-grams	Same as n-grams. The measure is more robust against coincidental common n-grams, but it may go wrong when a substring is added or omitted and when words are swapped.
Skip grams	Same as n-grams, but better suited against typographical errors.
Tf-idf	Matching quality on different error types not clear beforehand. Will depend on

	which tokens are deleted and/or added to the set of tokens when an error is made.
Soft Tf-idf	Same as tf-idf, also depending on the similarity measure chosen before the tf-idf is applied.
Monge and Elkan's Recursive Matching Scheme	Matching quality depend on similarity measure chosen. It is likely the measure better in correcting for substring omission than correcting for substring additions, because for every word in the input string the highest similarity value is used.
Soundex	Higher matching quality on variant toponyms due to phonetically similar transliterations.

On the other hand, name variants and variations of spatial entities have been used to enhance geocoding and address search. It uses a log of user address searches to model user behavior with regards to spelling mistakes to generate spelling variants of properly spelled address tokens. The model uses a statistical approach, using user clicks to learn which typos and how often are made. The proposed approach augments the index of a geocoding system with spelling variants to handle queries with misspelling tokens. Compared to a geocoding system that supports edit distances, the proposed approach serves more misspelled queries correctly improving the recall of the system.

Besides erroneous or inconsistent input data, the inaccuracy of reference data set that the input string data is attempted to get matched with causes an important geocoding difficulty[1]. With the advent of Internet and the advancements in Web service technologies, geocoding can be accomplished by submitting a request to an online geocoding services, such as Google, OSM (Open Street Maps) and Baidu, with little knowledge and effort, unlike in the early GISs, where geocoding was limited to groups of professionals who understood the underlying process and associated uncertainties. Although accessibility and availability of geocoding services improved, uncertainties with the results they produce are still associated[1]. This issue is more magnified, especially in developing countries as well as rural areas, where the complexity of city plans and address formats, lack of standard address system and data scarcity are still open issues.

Studies [25,29,30] were conducted comparing the quality of geocoded results from online geocoding services. Common evaluation metrics, such as match rate, positional accuracy and repeatability were considered. It was found that Google, Bing, Yahoo! and MapQuest had the same level of quality. However, it is difficult to generalize the results of these studies to other regions, specifically to developing countries. The typical approach to solving this problem involves a decision of whether to geocode to a less precise level or to

include additional detail from other sources to determine the correct geocode.

In summary, we did find related works on address matching from geocoding, but for a standardized address within the standardized addressing system. There are papers for string matching in a non-standardized spell-checking setting (i.e., the problem we encounter in this thesis, but they are applied on personal names using the English language. It would be interesting to know what the best measure is to match address strings that are domain specific non-standard setting and how to define the overall similarity between addresses that are represented in a multiple field spell-checking and address matching setting.

Chapter 3: System Overview and Similarity Measures Test

In this chapter we look at the proposed system in details. We also discuss which algorithms are best suited for the goals stated in Section 1.3. We propose which distance measure(s) are best to match address strings. In order to choose algorithms suited for this purpose, we must first define how we will measure performance.

The rest of this chapter is presented as follows. First, we will define the general overview of the proposed system. Then it is explained how the test to determine the best similarity metrics are performed. Second an overview on which measures are selected for the test and the reason for selecting those is given. Thereafter, the algorithm for the address matching and geocoding is described. Then the characteristics of the dataset used for testing is described, and finally the results and conclusion on which measures are best are presented.

3.1 System Overview

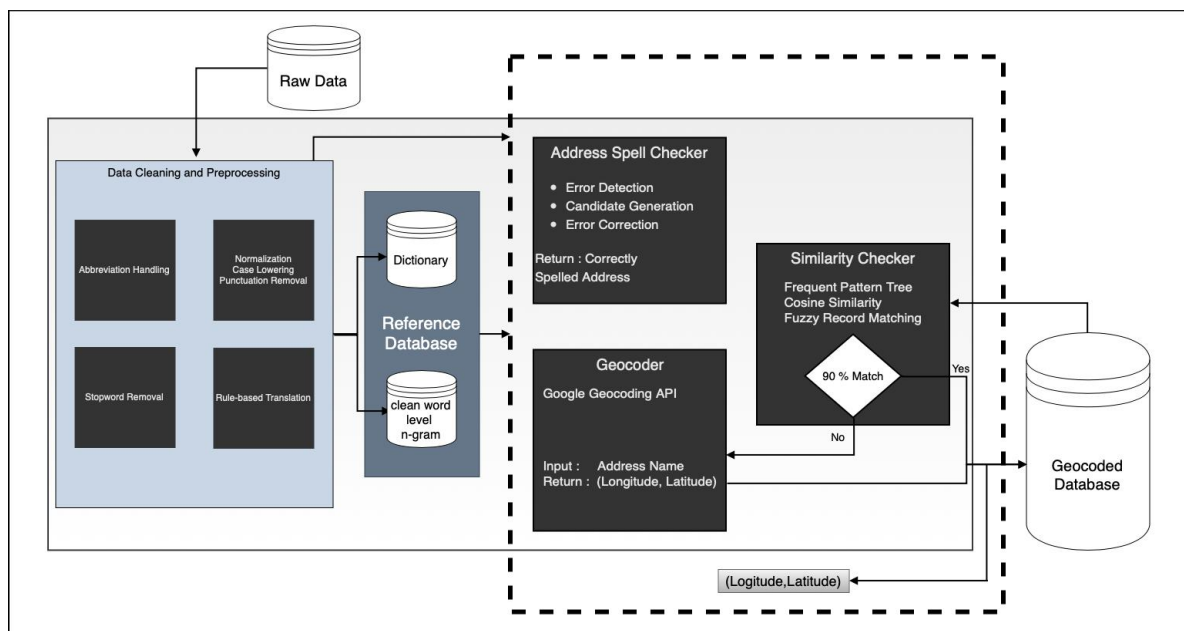


Figure 3-1 : System overview

As shown in the figure above, the system is composed of four main components. The data preprocessing, the domain specific dictionary, the spell-checking and address matching, and the geocoded address database. After the preprocessing step, which involve basic string

processing techniques such as abbreviation handling, stop word removal and string normalization, we extract domain specific dictionary that can be used for the spell-checking algorithm. The dictionary generation is done using the NLTK library as well as manual input. Some of the addresses, such as Capital Hotel or China Embassy, are represented using English words. For each unique word in the dataset if the word is an English word it is automatically added to the domain specific dictionary. For words that are not in the English dictionary we calculate the word frequency in the dataset and categorize them into there different levels, i.e. high frequency, medium frequency and low frequency. Each word is then manually checked for its spelling and added to the domain specific dictionary.

The main component of the proposed system, the spell-checking and address matching, uses the domain specific dictionary to check the spelling of a word as well as generating candidate addresses from misspelled addresses. This component is described in detail in Chapter 4 of this thesis work.

3.2 Test Design

Matching geographic names can be defined as the process of determining, within a given level of certainty, whether two strings correspond to the same place. Matching names is a challenging task, mainly because of spelling variations and some widely used practices, such as abbreviation. Spelling variations often cause problems when, for instance, names are dictated over the phone for manual input. Errors on common words can be detected using a dictionary but detecting errors on names requires different approach.

The first thing to observe is that geographical name, like personal names, share some common characteristics. We notice that words are usually small, ranging from three or four characters up to no more than a hundred characters. Special characters, such as accent marks, are used depending on the language, and sometimes are omitted. Abbreviations are common. Such observations are mostly from practice, but we have been able to confirm most of them during out experiments on preliminary test data.

The tests are performed with the use of a dataset collected from a taxi hailing platform, with locations typed in reality. The first thing to notice is that district strings and street/location names can be analyzed separately. Hence, we split every location present in the database in string of different types and put per database all string from the same type in a list. For a comparison of the measures, input strings matching exactly with a string in the database are neglected. Next to this, we consider only the unique strings in the dataset. For all string left over, every selected similarity measure outputs the location from the database it defines as most similar.

To be able to compare the performance of the measures, for every measure we count how often the targeted string is rated as “best” (i.e., most similar), second best and third best. We refer to the “best” returned string as the string given as first suggestion, the second best as second suggestion, etc. As mentioned above, we do not have a gold standard, which indicates that we do not know which string is the targeted string. At the same time, we did not have the possibility to go through the dataset by more than one person. Therefore, the author of the thesis defined manually which string is the targeted string.

For each rank in the list of suggestions, we did not assign a score, but interpreted the results in a more qualitative way. Intuitively, the decrease in matching quality between second and third suggestion is smaller than the decrease between first and second suggestion and smaller than between third and “no suggestion”. We absolutely prefer to get the target string as the first suggestion, but if it is not given as the first suggestion, we prefer to at least get the string in the suggestion list regardless being second or third position.

3.3 Measures for Testing

3.3.1 Address

For input address string we have chosen the similarity measure in the following table to compare:

Table 3-1 : Similarity measures for address

Edit Distance	Jaro-Winkler
Jaro	Bigrams Jaccard Similarity
Longest Common Substring	Trigrams Jaccard Similarity
Longest Common Subsequence	Tf-dif

From all the edit distance type measures, we did not implement the coherence edit distance, as it performed worse than the standard edit distance on personal names and shorter strings. Also the affine gap and Smith-Waterman distances are not considered as to not much abbreviations in district names are expected. On the other hand, Soundex is needed to determine similar characters. As this is a language dependent algorithm, the use of Soundex is crucial.

From all Jaro-Winkler types, permuted Jaro-Winkler as well as Sorted Jaro-Winkler are not implemented, because it takes a lot more time to determine the degree of similarity between two multiple word strings. We think it is not likely that the better matching quality will

outweigh these costs in computation time.

We do not expect the edit distance to perform well on address names, as the variation between string is large. Although the edit distance is suitable for correcting typographical errors, it will not be able to deal with omissions, abbreviations and extensions in strings, which is highly likely to occur often in address names.

Some remarks on the implementation of the measures and on the chosen parameter settings is given below:

- a) The longest common substring and the longest common subsequence are divided by the average string length to define the corresponding similarity measures. Dividing by the length of the shorter string and the length of the longer string would overlay favor the shorter and longer strings, respectively.
- b) The longest common substring measure is implemented in a way that it repeatedly finds and removes the longest common substring. The minimum length of a longest common substring is set to two.
- c) The maximum length of the common prefix in Jaro-Winkler is set to four
- d) The bigram and trigram measures are based on padded grams
- e) The TF-IDF is based on padded trigrams as tokens

3.4 Dataset

To test the similarity measures we selected to match address names, we make use of four datasets consisting of unique locations randomly selected from the main dataset that are presented to the Google Geocoding API by different taxi hailing platforms. ETTA gave permission to use the datasets in the thesis. The number of locations in Addis Ababa in the complete dataset is 41,271 for which 8,833 (21.40%) are unique location inputs. We performed a frequency distribution analysis of word lengths in the dataset. We observe that most address names have between one and four words. Almost 78% if them have between 8 and 20 characters.

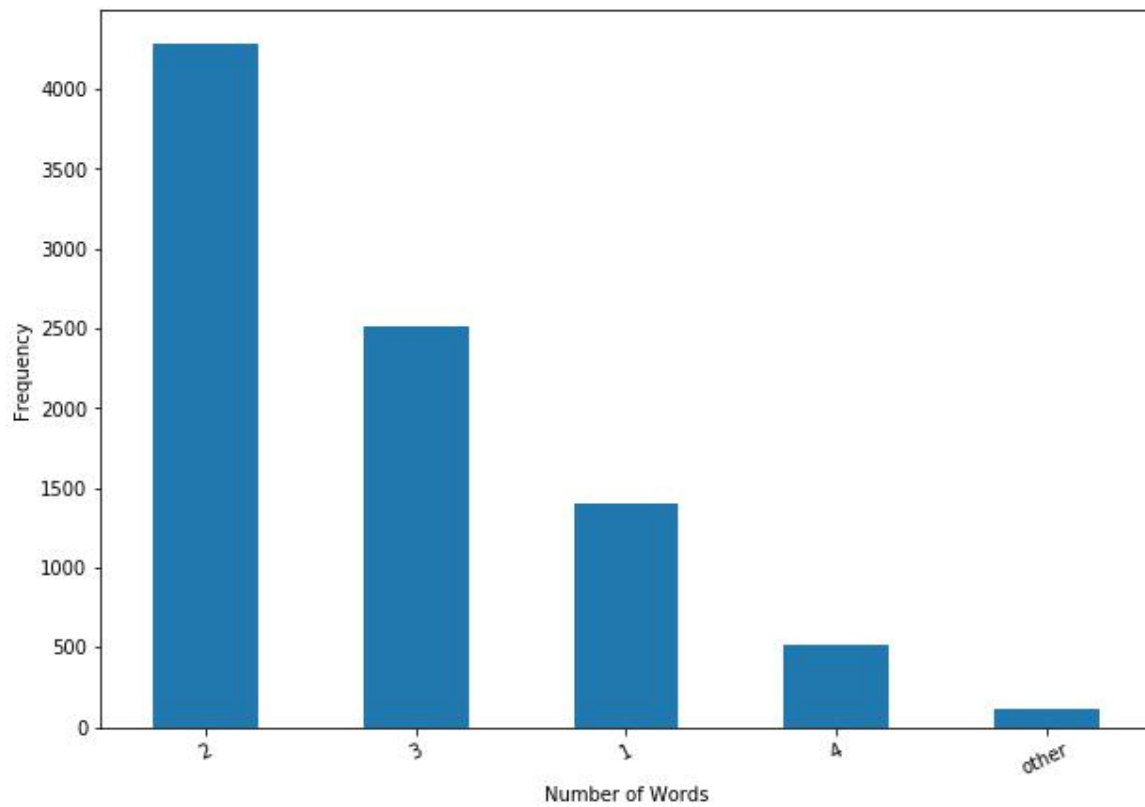


Figure 3-2: Frequency distribution of number of words

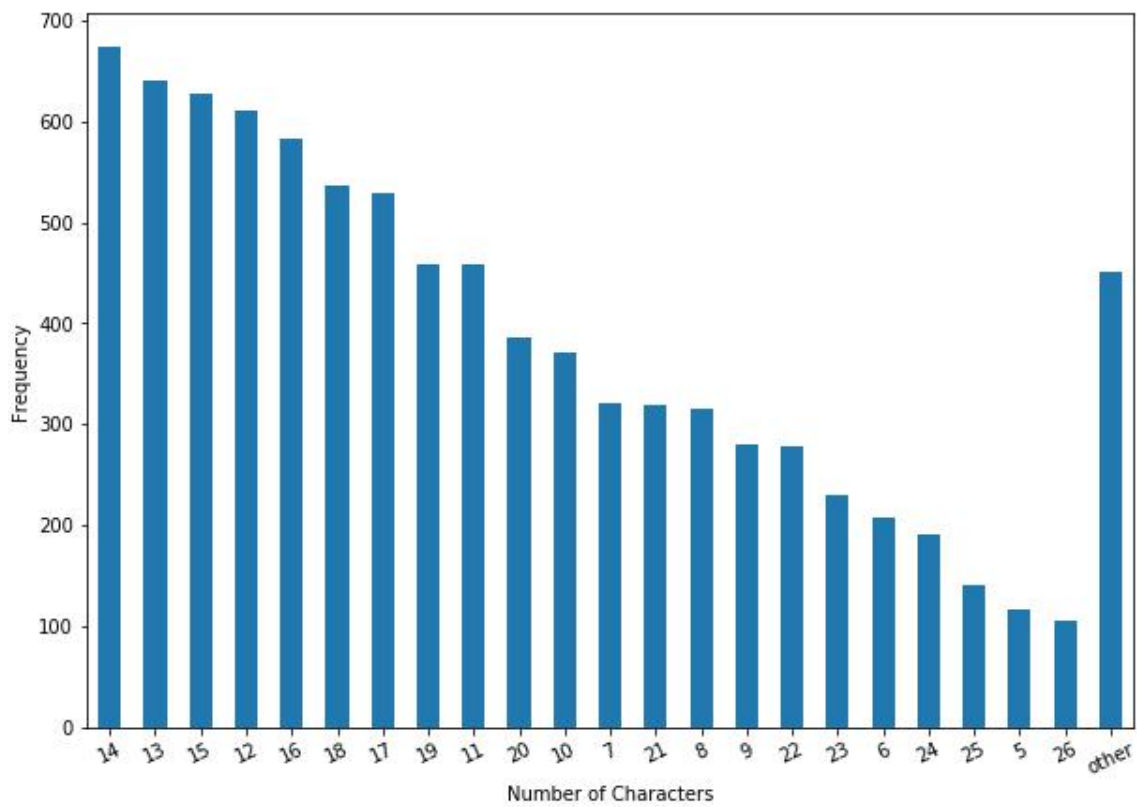


Figure 3-3: Frequency distribution of address name character lengths

First we determined the inputs that contain any error type with the given address. Then we determined the error types that do occur in the datasets. The main reason we did not define error types beforehand, is that it was unclear how specific an error type need to be defined. The selected measures are tested on the error types given in the table:

Table 3-2 : Error types and relative frequency of the errors per dataset

Error Type	Dataset (#unique errors)			
	977	1214	1468	1761
1- & 2- character errors (including omission and non-standard abbreviated strings)	66%	67%	62%	68%
Addition of short substring	12%	14%	24%	25%
Addition of long substring	22%	19%	4%	7%

All standard and non-standard abbreviations have been checked using a manually compiled list of <abbreviation, value> pair, where abbreviation is the standard and domain-specific abbreviation and value is its meaning, spelled completely. If an exact match is found between an input word and an abbreviation from the list, it is replaced by its full spelling. Our intention is to expand abbreviated strings which are quite common preceding in some kinds of place names, to their full description. We assume, heuristically, that the abbreviation has been used in order to save space or typing effort; therefore, we expect a large difference in size between the abbreviated word and its expected match in the pattern. We manually compile 60 unique domain specific abbreviation patterns and used it to check through the dataset. We then found that only 107 (0.01%) input addresses contained any of the predefined abbreviations.

When analyzing the four datasets, we notice that the two largest datasets are very similar with respect to error type and corresponding frequency for addition of short substring. Next to this, we can also notice that one error occurs way more often than others. Almost 3,566 (65%) of the total unique errors are 1-& 2- character errors. We decided to analyze 710(+/- 20%) of the string with this error type. Analyzing 710 strings having this error gives a good view on how the measures perform on the specific error type, while analyzing all would not be suitable due to time limitations.

The question rises how representative these datasets are. This is not an easy question to answer however, because it mainly depends on the application and types of errors expected in the application. It is possible to define which measures are best to implement since we have enough records to compare the performance of different measures for all different error types.

3.5 Measures Test Results

In this section the test results for the similarity measures are given and interpreted, leading to the conclusion concerning which measures are best to match address strings.

Table 3-3 : 1-character-edit and 2-charater-edit errors

Tested Number of Errors : 713		Total Number of errors : 3566		
Similarity Measure	First suggestion	Second suggestion	Third suggestion	No suggestion
Edit Distance	91.50%	2.32%	0.86%	5.32%
Longest Common Subsequence	94.10%	1.16%	0.29%	4.45%
Longest Common Substring	89.46%	1.74%	0.29%	8.51%
Jaro	85.40%	3.48%	1.74%	9.38%
Jaro-Winkler	87.43%	4.06%	1.45%	7.06%
Bi-grams Jaccard Similarity	91.78%	2.90%	0.58%	4.74%
Trigrams Jaccard Similarity	92.07%	2.61%	0.58%	4.74%
Tf-idf	70.76%	5.51%	4.35%	19.38%

We can see from the table above that the most measures perform rather well in correction for typographical errors. The longest common subsequence measure performs best, but all other measures perform reasonably well too. The bigrams are equally as good as trigrams, and Jaro-Winkler performs slightly better than Jaro.

As for the Tf-idf it has lower matching quality due to the reason that it sometimes fails to return a trivial location when the character-edit error introduces a trigram that has a much lower frequency in the database.

Table 3-4 : Addition of a short substring

Tested Number of Errors : 216		Total Number of errors : 1079		
Similarity Measure	First suggestion	Second suggestion	Third suggestion	No suggestion
Edit Distance	88.89	6.18%	3.70%	1.23%
Longest Common Subsequence	95.07%	2.47%	0%	3.70%
Longest Common Substring	97.53%	2.47%	0%	0%
Jaro	92.60%	3.70%	0%	3.70%
Jaro-Winkler	93.83%	2.47%	0%	3.70%
Bi-grams Jaccard Similarity	94.3%	4.70%	0%	1.00%
Trigrams Jaccard Similarity	96.30%	3.70	0%	0%
Tf-idf	88.89%	6.18%	3.70%	1.23%

The measures in general produce clearly better results than when matching with longer

added substrings. The Levenshtein edit distance and Tf-idf performing worse than the rest and the trigrams as well as longest common substring, again, giving the best results.

Table 3-5 : Addition of long substring

Tested Number of Errors : 127		Total Number of errors : 628		
Similarity Measure	First suggestion	Second suggestion	Third suggestion	No suggestion
Edit Distance	48.29%	11.22%	4.39%	36.10%
Longest Common Subsequence	57.07%	7.80%	3.90%	31.23%
Longest Common Substring	63.41%	8.29%	3.90%	24.40%
Jaro	90.16%	5.37%	1.98%	2.49%
Jaro-Winkler	92.07%	2.95%	2.49%	2.49%
Bi-grams Jaccard Similarity	65.37%	18.54%	1.95%	14.14%
Trigrams Jaccard Similarity	92.63%	2.93%	1.98%	1.00%
Tf-idf	90.19%	3.90%	1.98%	3.93%

The Levenshtein Edit distance does not perform well on types of errors where a long substring of at least six characters is added. The minimum Levenshtein distance is the difference between the string lengths, which lead to a Levenshtein distance between the input and targeted string of at least six in all cases. As the addition of sub-sequences and sub-strings are taken into account, the longest common sub-sequence and longest common sub-string apparently are not performing very well, due to the longer strings having characters in common with the addition being overly favored.

When looking at Jaro and Jaro-Winkler measures, due to the characteristics of the dataset, these measures perform excellent on matching with the added substring. The additions of the substrings are mostly behind the original string. Both the grams and Tf-idf measures yield good results. We can clearly notice that the trigrams Jaccard outperform the bigrams Jaccard.

As an overall conclusion, Jaro and Jaro-Winkler perform worse than trigrams Jaccard for all error types encountered within the dataset while trigrams Jaccard outperforms bigrams Jaccard. Character-edit errors can be corrected using both bigrams and trigrams as they show equally good results, but for other types the trigrams are substantially better. Similarly the same conclusion holds true for the Levenshtein distance: while the measure performs just about as good as the trigrams Jaccard measure for character-edit errors, but for all other error types it clearly performs worse. Finally, the longest common substring measure does not yield any advantages over the trigrams, as the trigrams Jaccard shows better results for all error types. However we will conclude by pointing out that those dominated measures might have advantages over the trigrams measure with respect to matching quality, which makes them potential candidates for the spell checking algorithm and address matching algorithms.

In the next chapter, we will be testing the proposed method for enhanced geocoding by address name matching. We will use the some of the measures from the above test when implementing the spell checking and correction algorithm.

Chapter 4: Address Matching for Geocoding

In this chapter, we propose an enhanced geocoding of addresses by implementing a spell checking and correction algorithm as well as address matching before carrying out the geocoding process.

The outline of the chapter is as follows. First, it is explained how the spell checking and the address matching algorithms are implemented. Second, an overview of the proposed method is given. This includes the structure and system overview to be implemented. There after measures for the address matching algorithm will be described based on the results from the test in the previous chapter. Then, the geocoding process as well as the evaluation are given. Finally the test and the evaluation results are presented.

4.1 Address Matching and Spell checking

For the address matching, we implement a spell checking and correction algorithm. The algorithm first determines error words from the given input address. This is achieved by simply doing a dictionary look up. Each input address will be tokenized to a word level unigram. For each word in the input string the error detection algorithm is then applied to determine whether the given word in the address input string exists in the database. The task of the error detection algorithm is to validate every word in the given input string against all unigrams in the dictionary database.

Input: Input address string and all clean addresses in the database

Output: Most similar address name (maybe exact match) or “no similar address is found”

```
for all string in the input address do
    check if its an error word;
    if error_word is found then
        candidates ← generate_candidates
        if there are candidates then
            nominees ← generate_nominee_address
            for all nominee addresses do
                calculate the frequency of the address in the database
                return the nominee address with the highest frequency
        else
            return “No Match Found”
```

Figure 4-1: Address Matching

Once a word is flagged to be an error, the candidate generation algorithm is applied to the flagged error word. The candidate generation algorithm exploits different similarity metric measures to search for unigrams in the dictionary having similarity with the error word. The top five unigrams having higher similarity degree to the error word will be selected as candidates. In case of an edit distance measure, unigrams with minimum edit distances are to be selected as candidates for the flagged error word. Next, is to select the best candidate as a correction for the error word. This is achieved by applying the error correction algorithm.

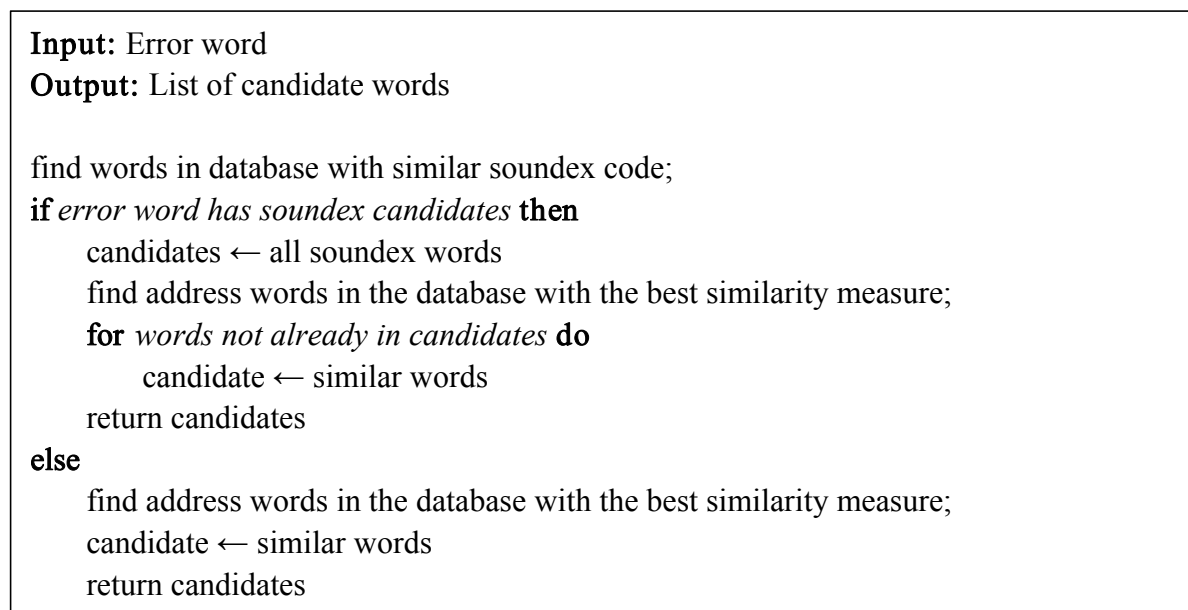


Figure 4-2 : Candidate Generation

The error correction algorithm first produces several nominee addresses, each containing one of the previously generated unigram candidate. Then, the frequency or the number of occurrence of each created nominee address in the dictionary is calculated. This is a manually generated dictionary that contains addresses from the dataset that are in any way error free. The candidate unigram that belongs to the nominee address with the highest frequency is asserted to be the real correction for the originally detected error word.

We will use the following measures for the spell checking and correction algorithm:

- Longest Common Subsequence
- Trigrams Jaccard
- Tf-idf

4.2 Methodology

In this section the general work flow of the proposed method is discussed. Once the preprocessing steps such as abbreviation handling, normalization, case lowering, punctuation removal... etc, were completed, we extracted and compiled domain specific dictionary that is to be used as a reference and lexicon for the spell checking algorithm. The dictionary generation was performed first by filtering out unique words already found within the English dictionary. For the location and specific words, we first used a string clustering algorithm to reduce the dimensionality caused by the different variation and misspelling of the words. Then, we manually selected the correct representation of the different variations of the words. Some addresses didn't have any form of correct representation within the dataset collected. Here we extracted any publicly available data, such as data from open street map and manually selected some of the address names as the data from the public is full of misspellings and variations. We extracted 29,222 (70.82%) clean addresses out of the total 41,261 data points, while any form of spelling error was found on 12,039 (29.18%) of the addresses.

Table 4-1 : Unique clean and erroneous data points

Total datapoints	41,261
Clean datapoints	29,222
Noisy datapoints	12,039
Unique datapoints	8,833
Unique clean datapoints	3,420
Unique noisy datapoints	5,413

Next, the address spell checker is applied to each address input that has any spelling error. Out of the 12,039 erroneous data points we extracted 3827 unique error words with spelling error from the datasets. As described in the previous section, for every error word in an input address, candidate words are generated from the domain specific dictionary using the above mentioned similarity measures. Then, every possible combination of words using the candidates generated is presented to the error correction. Finally, the correctly spelled address is returned by the spell checker which is in turn used to match with an already existing address in the database that is correctly geocoded. If an existing geocoded address with a similarity degree of 85% or more is not retrieved from the database, the geocoding algorithm is then applied to the address.

4.3 Geocoding and Evaluation

In this section, the geocoding process and the evaluation metrics are described.

4.3.1 Geocoding

Geocoding is the process of assigning latitude/longitude coordinate pairs to the description of a place entity by comparing the descriptive location-specific elements to those in reference data. This includes parsing input address, standardizing abbreviated values, assigning each address to a match key, indexing the needed categories, searching the reference data, assigning a score to each potential candidate, filtering the list of candidates based on the minimum match score, and delivering the best match.

The geocoding technique, consisted in an automatic method, is based on an online geocoding service accessible at <https://maps.googleapis.com>. The Google Geocoding API is a service that provides geocoding and reverse geocoding of addresses. After automatic online geocoding processing, latitude and longitude coordinates in the WGS 84 projection system for each address geocoded.

Many issues affect the quality of geocoding results, the scarcity of data in the reference dataset, the quality of the input address data, the accuracy of the reference data and the uncertainty introduced by the matching algorithm. Although many online geocoding services like Google, Yahoo!, HERE, or OpenStreetMap are easily accessible by the end users, however most of these systems are proprietary solutions, they neither reveal data nor algorithms used. The total number of addresses stored in the database was not provided by Google. This makes it hard to compare distinct aspects of such services.

In this research thesis, each unique address that is error free in any form is geocoded using the Google Geocoding API. The result that is obtained from the geocoding service is then categorized into two groups, partial match and exact match. Next all addresses in the partial match category are manually checked for correct representation. Finally, the addresses that are accurately geocoded are put into the address database that is further used as an alternative local geocoding system.

4.3.2 Evaluation

For our evaluation we use a lexicon of clean addresses each supplemented with the correct spellings of addresses and accurate geographical representation. We have geocoded all of the 3,420 unique clean addresses. The table below shows the results obtained from the Google geocoding API.

Table 4-2 : Geocoding of clean address

Geocoded Addresses	Partial Match	Accurate Match
3,420	1,036 (30%)	2,384 (70%)

Out of the 1036 partial matches we were able to notify that 18 of the addresses were matched to locations with similar name in another country. For instance, “New York Bole” was matched to “John F. Kennedy International Airport, Queens, New York”. We also found out that 323 of the partially matched addresses were a total misrepresentation. Since Google does not reveal how the Geocoding algorithm works, we will consider these addresses as erroneous addresses and will be subjected to an address matching. The rest of the partially matched addresses are indeed addresses that are matched partially with inaccuracy of 500 - 1000 meters. We will be using the unique clean addresses to match the erroneous addresses. For each dataset we counted the number of times our approach suggested the correct address among the top three addresses and also the number of times the correct address was placed first. The ranking of the addresses is done using the frequency of the address within the lexicon of addresses. In the table below we include the candidate score count for the spell-checking algorithm.

Table 4-3 : Recall accuracy of the address spell checking and correcting algorithm

Dataset	Found (Top 3)	Found (Top 1)	Not Found	% Recall
Test_data 1 (977)	845 (86.49%)	703 (71.95%)	132 (13.51%)	86.49 %
Test_data 2 (1214)	962 (79.24%)	794 (65.40%)	252 (20.76%)	79.24 %
Test_data 3 (1468)	1143 (77.86%)	922 (62.81%)	325 (22.14%)	77.86 %
Test_data 4 (1761)	1330 (75.53%)	1082 (61.44%)	431 (24.47%)	75.53 %

For the geocoding, we will compare the geocoded result of the input address with the geocoding result of the correctly matched address. The evaluation will be based on the match rate, i.e. the number of correctly matched addresses. The table below shows the match rate of the Google geocoding API on the raw input address and the address matched using the proposed approach.

Table 4-4 : Match rate of raw input data and matched address using Google Geocoding API

Method	Accurate Match	Partial Match	No Match	% Match Rate
Google Geocoding API	30.12%	31.79%	38.09%	61.91%

Address Matching for Geocoding	61.60%	16.37%	22.03%	77.97%

As we see in Table 4-3, the number of addresses that were not matched to existing clean addresses withing the lexicon-database is reduced significantly. If we consider the last column in the table above, an average recall of 79.78% is obtained using the spell-checking and correcting algorithm. This can be interpreted as out of the total 5,413 unique noisy input addresses, we managed to accurately match 3,540 (65.40%), and partially match 778 (14.38%) of the addresses with addresses from the domain specific database generated using the unique clean address inputs.

As table 4-4 shows, the result of the geocoding of addresses is very promising. Overall, there is a significant improvement in geocoding performance when using the proposed spell-checking and address matching method than directly geocoding the input address using Google geocoding API. The overall percentage of accurately geocoded locations is 77.97% and the difference with the conventional direct geocoding accuracy is 16.06%.

Chapter 5: Conclusion and Future work

5.1 Result and Conclusion

This thesis research can roughly be divided into two parts:

1. In chapter 3 and 4 it is discussed how to find the best similarity measure for matching addresses represented using strings with the targeted address in the database. The input string could vary from the targeted string due to misspellings, alternative spelling, abbreviation and different word order, addition/omission of a string/substring. This is a country specific research, that it is assumed that it is known in which country and city the address is located and that country is Addis Ababa, Ethiopia.

2. In the second part (Chapter 5), a spell checking and address matching algorithm is proposed to handle the inaccurate/misspelled input address data by correcting the misspelling and matching it with a correctly spelled address before the geocoding process in order to lower the uncertainty of geocoded results. A look up dictionary composed of correctly spelled domain specific words along side an n-gram address dataset was used to increase the accuracy of the spell-checking algorithm. Google geocoding API was then used for geocoding the preprocessed input address data.

Four datasets with locations as requested in reality are used, to determine the best similarity measure as well as to carry out the spell-checking and address matching process. We selected similarity measures from the available literature to compare with each other on several error types defined. The same datasets were then used for the conventional geocoding process using the Google Geocoding API.

As implicated in the table in the last chapter, the Geocoding API has a match rate of 61.91% on the raw input address while accurately geocoding only 30.12% of the whole unique input data. A significant increase on the geocoding match rate of addresses from 61.91% to 77.97% when a spell checking and an address matching was allowed prior to the geocoding. Clear visible on the table is the percentage of accurately matched addresses increased from 30.12% to 61.60%. Using this method, we can add certainty to those addresses that were a bit ambiguous due to the variation in addresses caused by different misspelling errors and misrepresentation.

Another important thing to notice from the table is that, although the percentage of addresses that were not matched decreased from 38.09% to 22.03%, however that is still a higher rate. The reason for such a high “No Match” rate is a phenomenon called OOV short

for Out of Vocabulary or Data sparseness which usually leads to false-positive and false-negative detection of out-of-dictionary words. Principally, a false-positive is a word judged to be misspelled, however, it is correct. For example, the addresses “*Haile Garment*” and “*Gerji Mebrat Haile*” have the common word “*Haile*” where in the first case is correctly spelled whereas a misspelling in the second address. Such terms are usually manifested as domain specific terms, and other type of words that cannot be found in a traditional dictionary. Contrariwise, a false-negative is a word judged to be correct, however, it is misspelled. They are usually real-word errors and homophones that result in valid words in the dictionary such as “*lime and line*” and “*hail and haile*”. Due to the scarcity of location data, the generated domain specific dictionary was not large enough to cover the entire dataset. In fact, obtaining large dictionaries is the only way to improve the spell-checking error detection and correction rate. Notwithstanding, it is not enough to get a larger dictionary but also a wide-ranging comprehensive one, encompassing domain-specific terms, and other usually out-of-dictionary words.

5.2 Implication and Future Work

There are many areas where this research can be implicated such as urban transportation, tourism, policy making, and socioeconomic analysis. Geocoding results can be latitude/longitude pairs, full or partial addresses and or point of interest names. Latitude/longitude pairs can further be used to study, design and model an optimized commercial or non-commercial transportation systems. Geocoding accuracy and data quality are critical when such studies focus on exposure mechanisms that operate over short distances. The results from research can be useful in designing optimized traffic network system that can serve the high demand in public transportation.

One of the main challenges to accurate geocoding is the availability of good reference data. This research will play a vital role by building a set of geographic features to match against as well as robust address characteristics that enable matching address records to feature locations. Several address models and policies can be designed using the results obtained from this research. For many applications the purpose of geocoding addresses is to associate the individual location with demographic and other socioeconomic variables. Results obtained from this research will form a base for further studies and researches as a form of a look up table.

There are two very important subjects to investigate in the future:

- How will the algorithm perform on other cities in Ethiopia and the rest of the world, because the algorithm and measures are tested only on location in Addis Ababa.
- How will the geocoded results obtained from this thesis research be used to design standardized address models as well as design traffic mobility platform by analyzing the output data.

References

- [1] D.W. Goldberg, J.P. Wilson, and C.A. Knoblock. From text to geographic coordinates: The current state of geocoding. *URISA Journal*, 19(1):33–46, 2007.
- [2] Drummond, W. J. 1995. Address matching: GIS technology for mapping human activity patterns. *Journal of the American Planning Association* 61(2): 240-51.
- [3] Vine, M. F., D. Degnan, and C. Hanchette. 1998. Geographic information systems: their use in environmental epidemiologic research. *Journal of Environmental Health* 61: 7-16.
- [4] Davis Jr., C. A., F. T. Fonseca, and K. A. De Vasconcelos Borges. 2003. A flexible addressing system for approximate geocoding, *GeoInfo 2003: Proceedings of the Fifth Brazilian Symposium on GeoInformatics*, Campos do Jordao, Sao Paulo, Brazil, October 2003.
- [5] Bakshi, R., C. A. Knoblock, and S. Thakkar. 2004. Exploiting online sources to accurately geocode addresses. In D. Pfoser, I. F. Cruz, and M. Ronthaler, eds., *ACM-GIS '04: Proceedings of the 12th ACM International Symposium on Advances in Geographic Information Systems*, Washington D.C., November 2004, 194-203.
- [6] Mansour K., Roger M. F., Claude L., Multi agent-based addresses geocoding for more efficient home delivery service in developing countries. *e-Infrastructure and e-Services for Developing Countries* 19, pp. 294-304.
- [7] T. Churches, P. Christen, K. Lim, and J.X. Zhu. Preparation of name and address data for record linkage using hidden Markov Models. *BMC Medical Informatics and Decision Making*, 2(9), December 2002.
- [8] Levine, N., and K. E. Kim. 1998. The spatial location of motor vehicle accidents: a methodology for geocoding intersections. *Computers, Environment, and Urban Systems* 22(6): 557-76.
- [9] Ratcliffe, J. H. 2001. On the accuracy of TIGER-type geocoded address data in relation to cadastral and census areal units. *Int. Journal of Geographical Information Science* 15(5): 473-85.
- [10] Cayo, M. R., and T. O. Talbot. 2003. Positional error in automated geocoding of residential addresses. *Int. Journal of Health Geographics* 2(10).
- [11] Krieger, N., P. D. Waterman, J. T. Chen, M.-J. Soobader, and S. V. Subramanian. 2003. Monitoring socioeconomic inequalities in sexually transmitted infections, tuberculosis, and violence: geocoding and choice of area-based socioeconomic measures. *Public Health Reports* 118(3): 240-60.
- [12] Rushton, G., M. Armstrong, J. Gittler, B. Greene, C. Pavlik, M. West, and D. Zimmerman. 2006. Geocoding in cancer research—a review. *American Journal of Preventive Medicine* 30(2): S16-S24.
- [13] Jaro, M. 1984. Record linkage research and the calibration of record linkage algorithms. Statistical Research Division Report Series SRD Report No. Census/SRD/RR-84/27. Washington, D.C.: U.S. Census Bureau. <http://www.census.gov/srd/papers/pdf/rr84-27.pdf>.
- [14] Christen, P., T. Churches, and A. Willmore. 2004. A probabilistic geocoding system based on a national address file. *Australian Data Mining Conference*, Cairns, AU, December 2004. <http://datamining.anu.edu.au/publications/2004/aus-dm2004.pdf>.

- [15] P. Christen and T. Churches. A probabilistic deduplication, record linkage and geocoding system. Proceedings of the Australian Research Council Health Data Mining Workshop, 2005.
- [16] Buckley, J. (1985). Fuzzy hierarchical analysis. *Fuzzy Sets and Systems*, 17(3), 233–247.
- [17] Zandbergen, P. A. (2009), Geocoding Quality and Implications for Spatial Analysis. *Geography Compass*, 3: 647-680. doi:10.1111/j.1749-8198.2008.00205.x
- [19] Soundex Code, Robert C. Russell, US Patent 1261167, 1918.
- [20] Markov, A.A., “Essai d’une recherche statistique sur le texte du roman “Eugène Iengouine””, *ull. cad. mper. ci. t. etersburg*, , 913.
- [21] Shannon, C.E, “A mathematical theory of communication”, *Bell system Technical Journal*, 27(3), 379-423, 1948.
- [22] G. Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33(1):31–88, March 2001.
- [23] Chomsky, N., “Three models for the description of language”, *IRI Transactions on Information Theory*, 2(3), 113-124, 1956.
- [24] Chomsky, N., “Syntactic Structures”, Mouton, The Hague, 1957.
- [25] Roongpiboonsopit D, Karimi HA. Comparative evaluation and analysis of online geocoding services. *Int J Geogr Inform Sci*. 2010;24:1081-100.
- [26] James L Peterson. Computer programs for detecting and correcting spelling errors. *Communications of the ACM*, 23(12):676–687, 1980. ---- Neha Gupta and Pratistha Mathur. Spell checking techniques in nlp: a survey. *International Journal of Advanced Research in Computer Science and Software Engineering*, 2(12), 2012.
- [27] Wagner, R.A. and Fischer, M. J, “The string-to-string correction problem”, *Journal of the Association for Computing Machinery*, 21, 168-173, 1974.
- [28] Levenshtein, V.I., “Binary codes capable of correcting deletions, insertions, and reversals”, *Cybernetics and Control Theory*, 10(8), 707- 710, 1966.
- [29] Roongpiboonsopit D, Karimi HA. Quality assessment of online street and rooftop geocoding services. *Cartography and Geographic Information Science*. 2010;37:301-18.
- [30] Pietro GD, Rinnone F. Online geocoding services: a benchmarking analysis to some European cities. 2017 Baltic Geodetic Congress (BGC Geomatics);2017Jun 22-25;Gdansk, Poland. USA: IEEE; 2017. p. 273-81.
- [31] Hamming, Richard W., “Error detecting and error correcting codes”, *Bell System Technical Journal* 29 (2): 147–160, 1950.
- [32] L. Allison and T.I. Dix., “A bit-string longest common-subsequence algorithm”, *Information Processing Letters*, 23:305–310, 1986.
- [33] G.R. Hjaltason and H. Samet. Index-driven similarity search in metric spaces. *ACM Transactions on Database Systems*, 28(4):517–580, December 2003.
- [34] G. Bard. Spelling-error tolerant, order independent pass-phrases via the Damerau-Levenshtein string-edit distance metric. *Fifth Australasian Symposium on ACSW Frontiers*, 68:117–124, 2007.
- [35] Ratcliffe, J. H. (2004). Geocoding crime and a first estimate of a minimum acceptable hit rate. *International Journal of Geographical Information Science* 18 (1), pp. 61– 72.
- [36] Cayo, M. R., and Talbot, T. O. (2003). Positional error in automated geocoding of residential addresses. *International Journal of Health Geographics* 2 (10) [Online]. Retrieved on 26 January 2009 from <http://www.ij-healthgeographics.com/content/2/1/10>

- [37] Dearwent, S. M., Jacobs, R. J., and Halbert, J. B. (2001). Locational uncertainty in georeferencing public health datasets. *Journal of Exposure Analysis and Environmental Epidemiology* 11, pp. 329– 334.
- [38] Drummond, W. J. (1995). Address matching: GIS technology for mapping human activity patterns. *Journal of the American Planning Association* 61 (2), pp. 240– 251.
- [39] Bartolini, I., Ciaccia, P., and Patella, M. 2002. String matching with metric trees using an approximate distance. In *String Processing & Information Retrieval (SPIRE)*, Lecture Notes in Computer Science, 2476, 271-283, Lisbon, Portugal
- [40] W.E. Yancey. Evaluating string comparator performance for record linkage. Technical Report Statistical Research Report Series RRS2005/05, US Bureau of the Census, Washington D.C., June 2005.
- [41] Youssef Bassil & Mohammad Alwani, “Parallel Spell-Checking Algorithm Based on Yahoo! N-Grams Dataset,” *International Journal of Research and Reviews in Computer Science*, Vol. 3, No. 1, February 2012.
- [42] Davis Jr, Clodoveu & Fonseca, Frederico & Borges, Karla. (2003). A Flexible Addressing System for Approximate Geocoding
- [43] P.E. Christen. A comparison of personal name matching: Techniques and practical issues. <http://astro.temple.edu/~joejupin/entitymatching/tr-cs-06-02.pdf>. Technical Report TR-CS- 06-02, Australian National University, 2006.
- [44] Keskustalo, H., Pirkola, A., Visala, K., Leppanen, E., and Jarvelin, K. 2003. Non-adjacent digrams improve matching of cross-lingual spelling variants. In *Proceedings of String Processing & Information Retrieval (SPIRE)* (Mauers, Brazil, October 8-10, 2003). Springer, New York, 252-265
- [45] A.E. Monge and C.P. Elkan. The field matching problem: Algorithms and applications. *Proc. Second Intl Conf. Knowledge Discovery and Data Mining (KDD 96)*, pages 267–270, 1996.
- [46] Kukich, K., “Techniques for automatically correcting words in text”, *ACM Computing Surveys*, 24(4), 377–439, 1992.
- [47] K. Kukich. A comparison of some novel and traditional lexical distance metrics for spelling correction. In *Proceedings of INNC-90-Paris*, Paris, France, July 1990, pages 309–313, 1990.
- [48] G. Salton. Automatic text transformations. In G. Salton, editor, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*, pages 425–470. Addison-Wesley, MA, USA, 1988.
- [49] U. Pfeifer, T. Poersch, and N. Fuhr. Searching proper names in databases. In R. Kuhlen and M. Rittberger, editors, *Hypertext - Information Retrieval - Multimedia, Synergieeffekte elektronischer Informationssysteme*, pages 259–276. Universitätsverlag Konstanz, Konstanz, Germany, 1995.